Literature review of state of the art Transfer Learning and Multi-task Learning applications

Author
Jiaxi Zhao[*]
Department of Computer Science
TUM
Garching, Munich
jiaxi.zhao@tum.de

Abstract

A huge success has been achieved in Machine Learning and Deep Learning, the tremendous potential value behind them lies in massive labeled data. However, there are some domains in which there are not enough labeled data such as legal domain in natural language processing. As a result, data scarcity problem is a top ranking problem now. Both Multi-task Learning and Transfer Learning can be used to solve data scarcity problems by leveraging useful information among related tasks. In this paper, we first give an overview of both learning methods, then introduce their principle, structure, advantage and disadvantage of them separately. Many applications using MTL and TL into different modalities such as text, protein sequence and source code are introduced, during which their application area, use case, task, model structure, dataset and results are elaborated. There is a summary below every modality. Finally, a conclusion of the paper is presented.

1 Introduction

Machine learning methods enable models to learn and predict future data from experience automatically without being explicitly indicated, during which a huge amount of data is needed to train a good model. Especially deep learning, a subset of machine learning but with numerous layers with each layer provide an interpretation to the data it feed on. For a deep learning which has many layers and parameters, it usually need millions of labeled data to train an accurate model.
In the past several years, for those cases where enough data can be achieved, we have succeeded in training more and more accurate models using deep learning method. How good? For example, the performance of the latest residual network[1] on ImageNet is better than human in recognizing objects; Goggle's Smart Reply[2] can take over 10 percent of all mobile response; Speech recognition accuracy keeps increasing and has exceed manually typing now. [3]
However, those good performances are based on huge amount of labeled data. The truth is, there are certain tasks and domains where labeled data is painstakingly to get such as medical images. As a result, how to alleviate data scarcity problems is top ranking problems now.
That's when Multi-task Learning and Transfer Learning comes into being. While the goal of traditional Machine Learning methods is to train a single model to perform our desired task, Multi-task Learning and Transfer Learning broaden their eyes and combine other related tasks' knowledge into the target task based on the consumption that related tasks may share some common features and can benefit from each other.
Biologically, we can see this process from human being. When learning new tasks, we often apply

[*]Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

Preprint. Under review.

our previous learned knowledge to our new challenge. For instance, a new born baby first recognizes the faces of it's parents and then uses this knowledge to recognize other objects.
Technically, in computer vision, for example, different objects may shares similar colors, strips, edges, shapes recognition mechanism, which can offer extra information on other related tasks and improve on their performances.
Multi-task Learning and Transfer Learning are both excellent at solving data scarcity problems, currently, there are tremendous excellent papers of diversiform applications of these two learning methods. As a result, it's hard for beginners to choose suitable method when dealing with data scarcity problems. That is the motivation of this paper–to be a navigation light on data scarcity problem solving method. After reading the paper, people can get a general idea of which learning method to use on their specific problem.
The remainder of this paper is organized as follows. Chapter two first points out the differences between Multi-task Learning and Transfer Learning. Then illustrates both of them separately according to several parts: definition, reason to chose the method, principle and structure, types and issues. Chapter three introduce some classic language models widely used in Transfer and Multitask Learning. Chapter four contains the details of applications in text, protein sequence and source code. Chapter five is an overall conclusion about the paper.

2 Theory

Although both Multi-task Learning and Transfer Learning can solve data scarcity problem, they are different in principle. See Figure1[1]. In Multi-task Learning, several tasks can be learn at the same time. Every learning agent receives message from several tasks at once, and every task needs to be learned. In Transfer learning, tasks can be divided into two parts: source task and target task. When training source task, we know nothing about target task. After source task is trained properly, it can be feed to learning agent to improve target task's performance. We care most about the target task, the attention on source and target task are not symmetric in Transfer Learning.

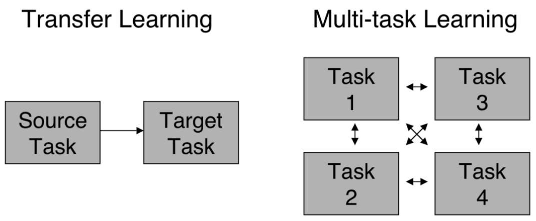


Figure 1: Comparison of Transfer Learning and Multi-task Learning structure [1]

Owing to the different principles, both learning methods have their own pros and cons. In Multi-task learning, every model needs to be trained from scratch, which means a large amount of data and time is required to get a good model. On the bright sight, the dynamic training method enables models receive arbitrary number of tasks. What's more, it can handle any task dependency graph (for example, T1 can learn from T2, and T3 from T4 and T6). In contrast, Transfer Learning can only learns T2 from T1.
In terms of Transfer Learning, it is more efficient and less data requirement. In practise, a pre-trained model can be used as a source task, all needed to do is fine-tune the pre-trained model to fit the target task.
To be more concrete, more details about Multi-task and Transfer Learning are elaborated at follows.

2.1 Multi-task Learning

2.1.1 What is Multi-task learning?

To start with, a definition of Multi-task Learning is given:

Definition 1. Given N learning tasks $(T_i)_{i=1}^{n}$ where all the tasks or a subset of them are related but not identical, multi-task learning aims to help improve the learning of a model for Ti by using the knowledge contained in the n tasks. See Figure2

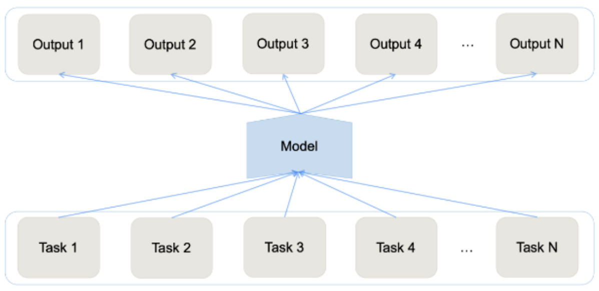


Figure 2: Multi-task Learning diagram [2]

For instance, there are 4 main accents in British English: Irish accent, Scottish accent, British accent and Welsh accent. However, every accent data is not enough to train a good classification model. Is is possible to improve the model performance if these 4 tasks are trained simultaneously? Yes. If these 4 tasks share some common features, the model may finds it easier to learn than trained isolately. In this case, all tasks share the same audio recognition method, entity detection method and relation extraction method. Thus, when N tasks share some identical basic features, train them simultaneously may better than separately.

2.1.2 Motivation

Machine learning methods usually train a single model on the desired task then fine tune the model until it achieves acceptable results. Although a good performance can be achieved in this way, they may ignore some extra useful information, namely, the information from other related tasks. The model of original task can generate better if it shares representations with other related tasks.
There are some mechanisms underlie Multi-task learning that make it an outstanding learning method:

- Data Augmentation:
  Training multiple tasks at the same time means having multiple data source and implicitly more data to train the model, which can help avoid overfitting and increase accuracy.

- Attention focusing:
  If data is limited and high dimensional, it's hard to decide which features are relevant to the task. Having multiple tasks trained simultaneously may solve the problem by offering additional evidence on the relativity of the features.

- Eavesdropping:
  Some features F may learns more from task A but much less from task B. It may because task A interact with F better or other features enhance the influence of task A. The accuracy of task B can be increased by features F learned from task A.

- Representation bias:
  Learning one task A suffers from overfitting, but learning task A and B jointly enables the model to obtain a more generalized representation. Multi-task Learning can learn a generalized, unbiased representation that suits for all related tasks.

Nowadays, Multi-task learning has achieves huge success in some domains, such as natural language processing, speech recognition and computer vision. It's a very promising domain and waits for us to explore.

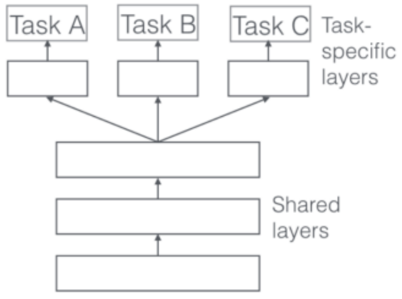
2.1.3 Two classic MTL structure for Deep Learning

So far we have introduced general idea and theoretical motivation of MTL, to be more concrete, let's have a closer look on its inside structure. In terns of Deep Learning, MTL is usually implemented with either hard or soft parameter sharing structure.
Before we start, a few simple notations need to be introduced:
Let$T_1, T_2, ...T_n$ be the set of tasks, each with $T_i = (X_i, y_i)$ data points and labels. Let M be the neural network to be trained, with k layers and parameter set:

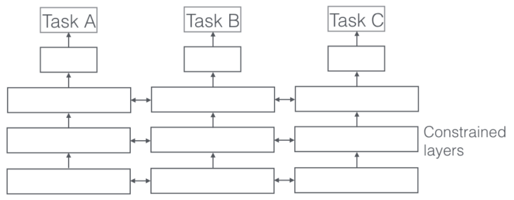
$$\Theta(M) = (P_1, P_2, ..., P_k)$$

Each Pi can vary between tasks, therefore we can view it as being made up of parameters for each task:$P_i = (P_{i1}, P_{i2}, ..., P_{ik})$
Each layer has its output, that we denote xi for Pi in general and x_{ij} for P_{ij} for the task-specific one. Hard and soft parameter sharing discussions are inspired from Ruder[2].

Hard parameter sharing
Hard parameter sharing is the most commonly used structure, all tasks share the same parameters while keep several task-specific output layers.
Here, we fix the bottom P_i, for $1 \leq i \leq k - h$(i.e.$P_{ij} = P_{it}, \forall 1 \leq j, t \leq n$), and we train h specific P_{kj} for all $1 \leq j \leq n$.
Hard parameter sharing does a great job in reduce the risk of over-fitting. It's easy to understand that the model needs to find a proper representation for all tasks, the more tasks the model has, the more general the representation is, the less over-fitting it coule be.



(a) Hard parameters sharing



(b) Soft parameter sharing

Figure 3: Hard parameter and Soft parameter sharing diagram [2]

Soft parameter sharing
In soft parameter sharing, each task has its own model and parameters. To make parameters more similar, the distance between parameters is regularization. Usually, we apply L2 norm Δ to measure difference of respective parameters.
This penalty is added to the overall loss to be optimized, at a certain rate λ:

$$L+ = \lambda k \sum i = 1n \sum j, t = 1\Delta(P_{ij}, P_{it})$$

2.1.4 Categories

Multi-task learning is a branch of machine learning, based on the nature of the learning method, MTL can be divided into several settings: multi-task supervised learning, multi-task unsupervised learning, multi-task semi-supervised learning, multi-task active learning, multi-task reinforcement learning, and multi-task online learning.
As mentioned above, Multi-task learning aims to improve performance of every single task by leveraging the useful information contains among them. As a result, MTL requires every task or

most of them are related. To be more concrete, relatedness can be encoded in three aspects: feature, parameter and instance.
Feature based models is based on the idea that different tasks share similar representations which can be a subset of original features. Parameter based model aims to find task relatedness from model parameters. Instance based model intends to construct the model for each task by instance weighting. A detailed illustration can be found in Yu Zhang[3]

2.2 Transfer Learning

2.2.1 What is Transfer Learning?

The definition of Transfer Learning is as following:
Definition 2. Given a source domain D_s and learning task T_s, a target domain D_t and learning task T_t, transfer learning aims to help improve the learning of the target predictive function $f_T(\Delta)$ in D_T using the knowledge in D_s and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$. See Figure4[4]

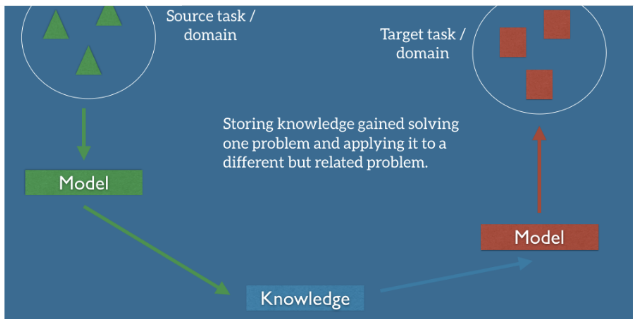

Source task / domain
Target task / domain
Storing knowledge gained solving one problem and applying it to a different but related problem.
Model
Model
Knowledge

Figure 4: Transfer Learning diagram [4]

For example, If we want to build a model for sentiment classification on restaurant comments, we can use a sentence classification model which is able to classify object and human description properly as a pre-trained model. A satisfying model can be trained only by fine-tune the pre-trained model. As we can see from the example, Transfer Learning aims to leverage the knowledge in pre-trained model to new related task or domain to speed up the training phase and lower the requirement of data.

2.2.2 Motivation

Traditional Machine Learning methods perform bad when there is no enough data for our target task or domain. It also breaks down when training and testing data doesn't have same features or distribution. In such cases, we have to find more labeled data or rebuild model from scratch for latest collected data. The truth is, it's time-consuming or even impossible to collect massive labeled data and rebuild the model. As a result, Transfer Learning comes into being. It makes transfer knowledge from one domain to another possible.
Transfer Learning has been widely used in some language modeling models such as Text, Protein Sequence and source code. More details can be found at chapter 3.

2.2.3 Deep Transfer Learning

In recent years, Deep Learning has been used in many fields such as text generation, image recognition, voice assistant and achieve huge success. However, the training time and labeled data required to train the model for the deep learning system is beyond imagination. Since there are already various well developed deep learning networks in some domains, why not share those existing networks with other new tasks? These pre-trained models are the backbone of our deep transfer learning method.
The important feature of deep learning model is the hierarchical representation of layers, which

means different layers learn different features. After some operations(usually pass through Fully Connected layers) the self studied layers are connected to a last layer to get the final output. The characteristic of this layer structure enables us to treat the pre-trained model without last several layers as a fixed feature extractor for other tasks.

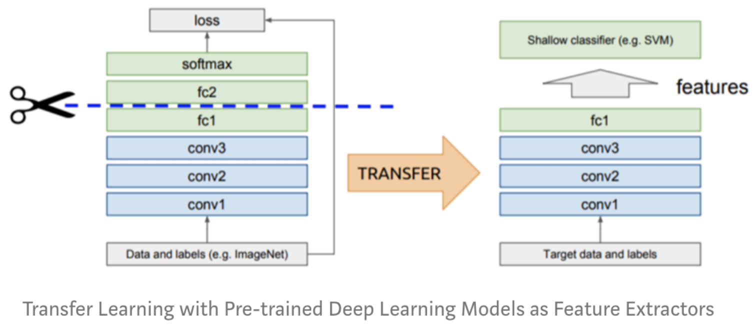

loss
softmax
fc2
fc1
conv3
conv2
conv1
Data and labels (e.g. ImageNet)
TRANSFER
Shallow classifier (e.g. SVM)
features
fc1
conv3
conv2
conv1
Target data and labels
Transfer Learning with Pre-trained Deep Learning Models as Feature Extractors

Figure 5: fine-tuning diagram [5]

As we can see from Figure 5[5], the key point of fine tuning the pre-trained model is to keep the parameters of the stable layers and only update the last several layers to fit the new task.
Then how to decide which layers to keep and which layers to fine tune? In practice, fine tune strategies depends on different tasks:
Figure 6[5] is a visual summary of fine-tune layer chosen methods.

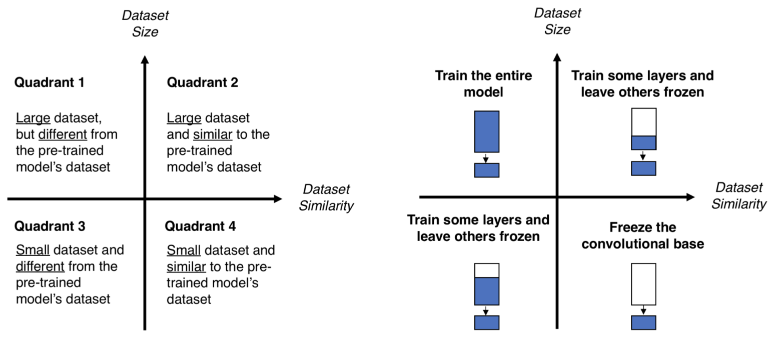

Dataset Size
Quadrant 1
Large dataset, but different from the pre-trained model's dataset
Quadrant 2
Large dataset and similar to the pre-trained model's dataset
Dataset Similarity
Quadrant 3
Small dataset and different from the pre-trained model's dataset
Quadrant 4
Small dataset and similar to the pre-trained model's dataset
Dataset Size
Train the entire model
Train some layers and leave others frozen
Dataset Similarity
Train some layers and leave others frozen
Freeze the convolutional base

Figure 6: visual summary of fine-tune layer chosen methods [5]

The entire model is divided into 2 parts, the upper bigger square contains convolution layers, and the bottom square stands for fully connected layers.
Quadrant1: when we have big dataset and small data similarity, we should train model from scratch since we have enough data to fit the new task. However, in practice, using a pre-trained model to initialize the parameters can be helpful.
Quadrant2: There are large dataset and the data is similar to pre-trained model. It is the best situdation. Because of the large amount of data, we can train a model from scratch without worrying about overfitting, but owing to the similarity of those two dataset it is time efficient to use the pretrained model. As a result, train some convolutional layers plus Fully Connected layers should be enough.
Quadrant3: Small dataset and different data. Everything is not good. The strategy is to find the balance between frozen layers and train layers, although it's hard. If we train whole layers, the model will overfit; if we train only last several layers, it will learn nothing. Perhaps we can use data augmentation methods to enrich data.
Quadrant4: For small, data related dataset, the best choice is just modify last several fully connected

layers, run the pre-trained model as a fixed feature extractor.

2.2.4 Categories

Based on the situation of source task and target task, transfer learning can be categorized into 3 types: Inductive TL, Transductive TL and Unsupervised TL. Figure 8 is a reference table of those 3 Transfer learning methods:

Transfer Learning Settings	Related Areas	Source Domain Labels	Target Domain Labels	Tasks
Inductive Transfer Learning	Multi-task Learning	Available	Available	Regression, Classification
	Self-taught Learning	Unavailable	Available	Regression, Classification
Transductive Transfer Learning	Domain Adaptation, Sample Selection Bias, Co-variate Shift	Available	Unavailable	Regression, Classification
Unsupervised Transfer Learning		Unavailable	Unavailable	Clustering, Dimensionality Reduction

Figure 7: Transfer Learning categories [6]

The above three learning methods can be further divided into four cases based on what knowledge could help improve performance for target domain or task, including:
Instance Transfer: Reusing knowledge from source domain to target would be helpful. However, in normal scenarios it can not be used directly. Instance transfer assumes that some data in source task can be reused in target task by re-weighting. The instance weight would be high for those similar data, and low for those dissimilar one.
Feature Transfer: It tries to minimize the domain divergence and reduce transport error rate by finding a suitable representation.
Parameter Transfer: This method is based on the assumption that related tasks or domains share some parameters. The knowledge can be transferred into new task by sharing those parameters.
Relation-Knowledge Transfer: It assumes data relation between source and target task is similar. Thus, what it tries to do is mapping relational knowledge between source and target task.
The following table[6] summaries the algorithms of Transfer learning method and corresponding cases:

	Inductive Transfer	Transductive Transfer	Unsupervised Transfer
Instance Transfer	TrAdaBoost	KMM, KLIEP	
Feature-Representation	Supervised-Feature Construction,Conv- Net	SCL, Conv-Net	Self Taught Clustering, TDA
Parameter Transfer	Regularization-based Method		
Relational Knowledge Transfer	TAMAR		

Figure 8: algorithms of Transfer learning method and corresponding cases

2.3 Issues

Both MTL and TL methods suffer from negative transfer. It means after knowledge transformation the performance of learning doesn't make any improvement but hurts in target domain. As for MTL, it assumes most of the tasks are closely related. However, each task might not be closely related to all of the available tasks. As for TL, it happens when source task is not sufficiently related to the target task or the representation of source and target task is not learned properly. How to avoid negative transfer is an important topic and can't get enough attention in the future.

3 General models for NLP

In this chapter, we focus on several landmark models in language modeling. A language model is a probability distribution over sequences of words. When given a corpus of tokens x = $(x_1, ..., x_T)$, the goal of language modeling is to estimate the joint probability P(x), which can be factorized as

P(x) = $\prod_T$ P($x_t|x_<t$). The higher the probability is, the more likely the sequence of x is. Then the problem can be reduced to the estimation of every conditional factor. Based on the difference of factor encoding methods, language modeling can be divided into multiple categories.

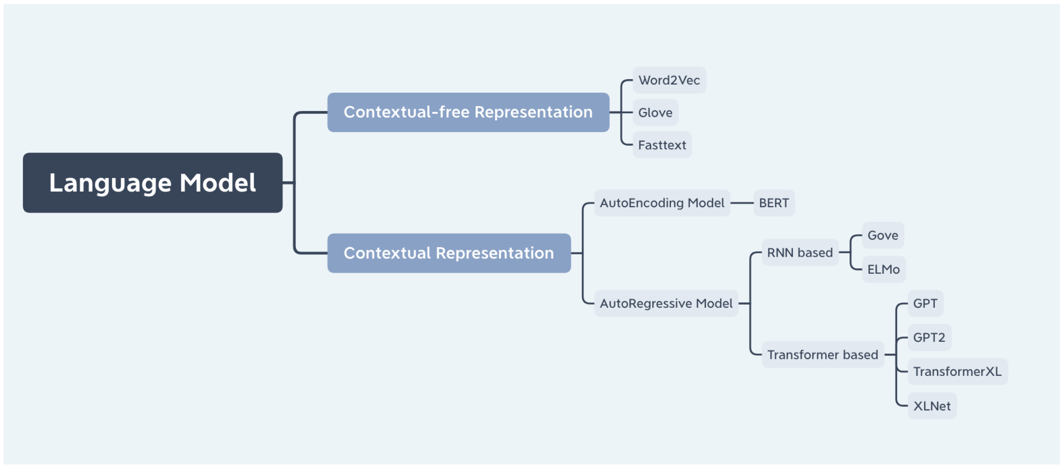


Figure 9: Categories of Language Modeling

As shown in figure9, based on the released time, I divide the models into two parts: Classic Approach, which are designed before 2016, and Modern Approach. This chapter is constructed exactly as the structure of Figure9 . The models marked in blue will be covered in more detail.

3.1 Classic Approach

3.1.1 Word2vec

The main task of word2vec is to learn high-quality word vectors from massive data sets with many words. Although the effort of finding an appropriate word representation has a long history, such as architecture of Neural Network Language Modeling(NNLM) [7], tf-idf [8] and Latent Semantic Analysis[9], none of those proposed models got a modest dimensionality of word vectors between 50 and 100 successfully after trained on more than a few hundred of millions of words.
Word2vec simplifies neural network by removing the non-linear layer, given the concern about minimize computational complexity. Two models are proposed in this paper[10] named CBOW model and Skip-gram model. As shown in Fig10.
CBOW aims to predict the word by the context. That is to say it maximizes the probability of the target word by looking at the context. While Skip-gram predict the current word based on context. At the last step of training process, we need to use softmax on the product of two word vector matrix[11] with the size of N*D (N is the number of vocabulary, D is feature dimention) to determine which words are chosen and back propagate to update the parameters. When data sets grow and the amount of vocabulary soars, it seems to be impossible to train the model. As a reult, word2vec propose 2 optimise methods: hierarchical softmax and negative sampling.[11].
It turns out Word2vec has been used to outperforms many previous SOTA tasks such as SemEval-2012 Task2[12], sentiment analysis[13] and paraphrase detection[14]. However, owing to the moving window-structure, word2vec can only see partial infornation instead of global for word prediction. Besides, word embedding learned from word2vec is context-independent, which means if one word has several different meanings, those embeddings will all be the same, which shouldn't be the case.

3.1.2 Glove

Glove[15] proposed a novel concept called co-occurrence counts matrix to take global information into consideration. The counts matrix can be denoted by X, whose entries X_{ij} tabulate the number of times word j occurs in the context of word i. Let $X_i = \sum X_{ik}$ be the number of times any word appears in the context of word i. Finally,let P_{ij}=P(j|i)= X_{ij}/X_i be the probability that word j appear in the context of word i. Given some insight on the co-occurrence ratio F(w_i, w_j, w_k) = $\frac{P_{ik}}{P_{jk}}$, the ratio can be small, large, which means k is related to either i or j, or equal to 1, which means it either related to i and j or related to neither of them. Glove tries to design a goal function J to fit the trend: J=$\sum_{i,j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - logX_{ij})^2$, and minimize the goal function to achieve word vector.

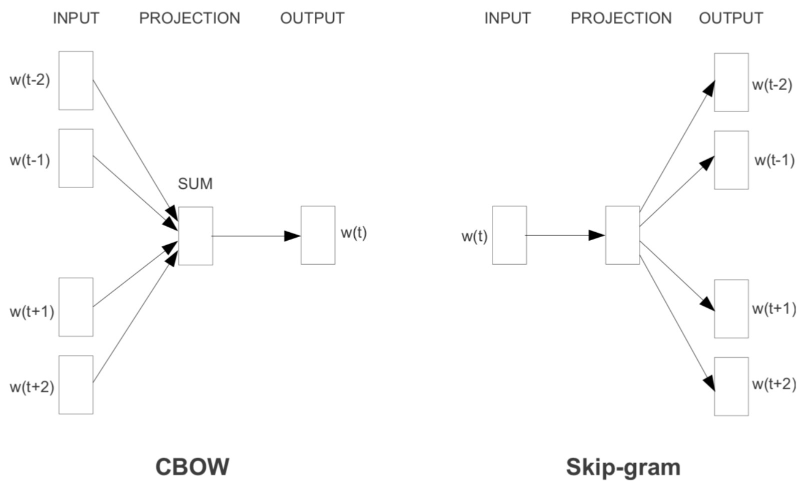


Figure 10: Architecture of CBOW model and Skip-gram model [10]

Glove performs better than skip-gram and CBOW models on word anaog tasks, word similarity tasks and Name-Entity Recognision task accroding to [15]. However, Glove also suffers from content insensitivity.

3.1.3 Fasttext

Fasttext model is designed by Facebook AI team, it can be viewed as another word embedding method which is an extension of the word2vec model. Instead of learning vectors for words directly, fastText represents each word as an n-gram of characters. For example, take the word "apple" with n =3, the input is divided in several subinput as <ap,app,ppl,ple,le>. The reason why it is designed in this way is that the decomposition helps capture the meaning of shorter words and allows the embeddings to understand suffixes and prefixes. As a result, fasttext can handle rare word, so even if a word wasn't seen during training, it can be broken down into n-grams to get its embeddings.For instance, if input has a word "uncomfortable" which the model never seen before, it can break the word into prefixes 'un' and 'comfortable'to help embedding. While Glove and word2vec can't provide word vector representation which is not in dictionary.
The process of fasttext is similar to CBOW, both of which are consist of input layer, hidden layer and output layer. What different is the input pf CBOW is the one-hot embedding of context of target word, while the input of fasttext is word embedding of all words in dictionary and their corresponding n-gram feature. More detail can be found at Github Page.

3.2 Modern Approach

Most models listed below this category except GPT2 are semi-supervised models using a combination of unsupervised pre-training and supervised fine-tuning. The advantage of unsupervised learning is that it can learn a universal representation that transfer to a wide range of tasks with little adaption. And the data domain of training tasks and target tasks don't need to be the same. The procedure can be deployed in two stages: First, training the language model on unlabeled data to learn an initial parameter of neural network model, then adapting these parameters got from the previous stage to a target task using corresponding labeled data.

As we mentioned above, this is the typical structure of Transfer Learning. However, these models can combine with Multi-task learning to preform a better performance. For example, Xiaodong Liu[16] designed a Multi-task Deep Neural Network(MT-DNN) structure, as shown in Figure11, with two stage. The first stage is pre-training of Lexicon Encoder and Transformer Encoder, here applys BERT technique by masking input sentences on both Lexicon encoders and transformer encoder. The second stage is fine-tuning, it receives contextual embedding from first stage, and go through different sub-flow to learn representation for each specific task. Mini batch based stochastic gradient descent (SGD) is applied. MT-DNN computes the loss across different task and applying the change to the model at the same time.

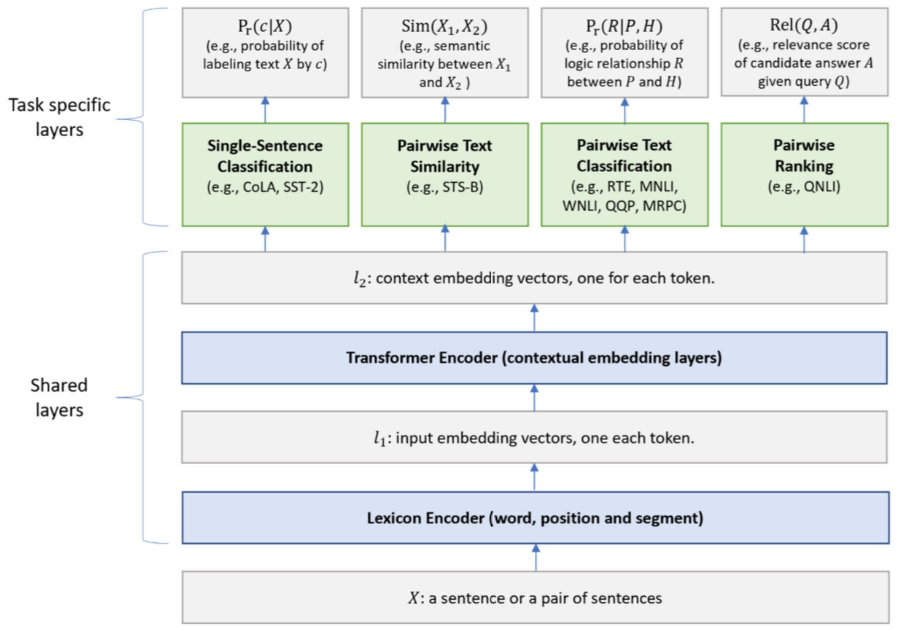

$P_r(c|X)$
(e.g., probability of labeling text X by c)
$\mathrm{Sim}(X_1, X_2)$
(e.g., semantic similarity between X_1 and X_2)
$P_r(R|P,H)$
(e.g., probability of logic relationship R between P and H)
$\mathrm{Rel}(Q,A)$
(e.g., relevance score of candidate answer A given query Q)
Task specific layers
Single-Sentence Classification (e.g., CoLA, SST-2)
Pairwise Text Similarity (e.g., STS-B)
Pairwise Text Classification (e.g., RTE, MNLI, WNLI, QQP, MRPC)
Pairwise Ranking (e.g., QNLI)
l_2: context embedding vectors, one for each token.
Transformer Encoder (contextual embedding layers)
Shared layers
l_1: input embedding vectors, one each token.
Lexicon Encoder (word, position and segment)
X: a sentence or a pair of sentences

Figure 11: Architecture of the MT-DNN model for representation learning [16]

Main difference from modern approach to classic approach is that modern approach can generate different word embeddings for a word that captures the context of a word. For example, the embeddings of word 'bank' are different in sentence "He went to *bank* yesterday" and "The girl is playing near the *bank*" because of their context.
Based on model structures, we divide modern approach models into two parts: models based on Recurrent Neural Network(RNN) and Transformer(attention) based models.

3.2.1 Module Category

Word2vec, Glove and Fasttest can only allow single context independent representation for each seen word, it's very challenging to ideally model polysemy and complex syntax and semantics relationship. RNN based models such as Elmo and Cove[17] make use of the time ordered characteristic in recurrent neural network to keep the position information and generate context dependent embeddings.
Translation based models such as BERT, XLNet, GPT2 refines RNN based models by adding attention mechanism, which allows longer range sentence prediction, and they also increase efficiency. Details are covered in the following chapters.

3.2.2 Elmo

In Elmo, a bi-directional LSTM model is designed to generate word vectors. It improves on the linear combination method of traditional bi-directional models, which much enhance the performance. Belinkov[18] shows combining internal states in LSTM enrich word representations. To be more specific, higher-level LSTM states capture context-dependent aspects of word meaning while lower-level states are good at modeling syntax information. Elmo also benefit from subword units by

convoluting on character, which is useful when handling out of vocabulary(OOV) problems. Besides, Elmo takes full advantage of access to tremendous amount of data, and train the bi-LSTM model on a corpus with approximately 30 million sentences. All of them make Elmo outstand from previous models.
Let's have a closer look at the model detail. In traditional bi-directional language model, illustrated in Figure12, given a sequence of the N tokens $(t_1, t_2, ..., t_N)$, the task of feed forward part is computing the probability of sequence p$(t_1, t_2, ..., t_k)$ reference to the history $(t_1, t_2, ..., t_{k-1})$:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k|t_1, t_2, ..., t_{k-1})$$

Backward part is similar to forward part, except it runs over the sequence from back to the front, predicting previous token given future ones:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k|t_{k+1}, t_{k+2}, ..., t_N)$$

Traditional bi-LSTM combines both forward and backward equation, jointly maximize the log likelihood of the combination formulation:

$$\sum_{k=1}^{N}(logp(t_k|t_1, t_2, ..., t_{k-1}; \Theta_x; \overrightarrow{\Theta_{LSTM}}, \Theta_s + logp(t_k|t_{k+1}, t_{k+2}, ..., t_N; \Theta_x; \overleftarrow{\Theta_{LSTM}}, \Theta_s)$$

Θ_x and Θ_s are parameters for token representation and softmax layer in both forward and backward direction. Θ_{LSTM} is separate parameters for LSTM in each direction.
Elmo does some improvement on the combination of forward and backward propagation representation. Instead of simply add them together, Elmo applies linear combination of those bi-LSTM layers. Here, $\vec{h}_{k,j}^{LM}$ means a content dependent representation at each position k,layer j(j=1,...L).$\vec{h}_{k,L}^{LM}$ means the last layer of token k, which can be used to predict next token t_{k+1} with a Softmax layer. $\overleftarrow{h}_{k,L}^{LM}$ is similar meaning but for back propagation process. For each token t_k, 2L+1 representations are computed:

$$R_k = x_k^{LM}, \vec{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, ..., L = h_{k,j}^{LM} | j = 0, ..., L$$

where $h_{k,0}^{LM}$ is token layer and $h_{k,j}^{LM} = [\vec{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM}]$ for each bi-LSTM layer.
Before down-stream fine-tuning stage, Elmo integrates all layers in each token R_k into one single vector $ELMo_k^{task}$:

$$ELMo_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}$$

In the equation above, s^{task} are softmax-normalized weights and γ^{task} is scalar parameter to scale the entire ELMo vector. Sometime it also beneficial for layer normalization[19] when each bi-LSTM layers has different distribution.

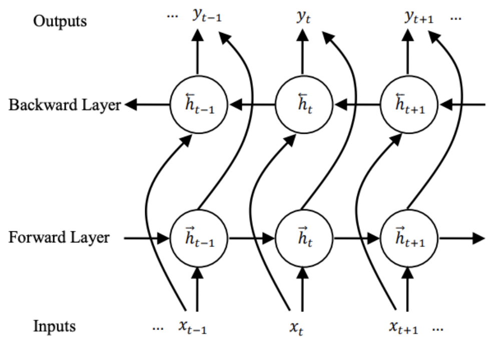
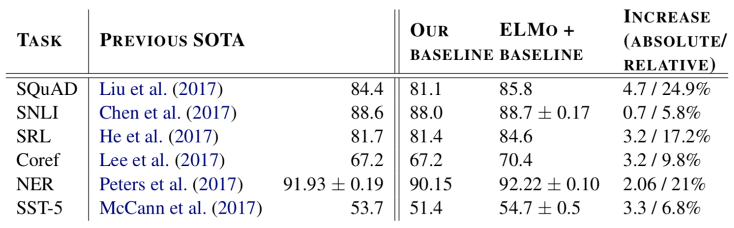


Figure 12: Architecture of a Bi-LSTM layer ELMo [19]

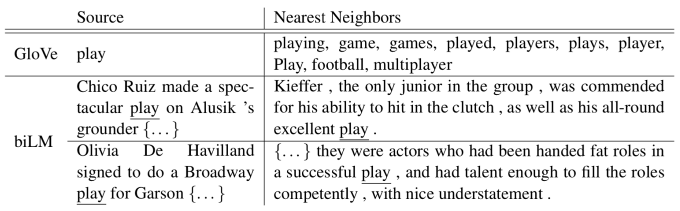
In some cases, using the pre-trined parameters $ELMo_k$ leads to significant drops in perplexity and increase in down-stream performance. In TagLM[20] it combines word embedding and Elmo embeddings together and achieve better results. This type of transfer learning techniques called feature-based learning. [21]

As shown in Mattew's paper [22], Elmo performs well in many applications. Figure13 shows performance of ElMo across six benchmark NLP tasks.
First line is Question answering task on the Stanford Question Answering Dataset (SQuAD). After adding ELMo to the baseline model, test set F1 improved by 4.7% from 81.1% to 85.8%, a 24.9% relative error reduction over the baseline.
Second line is Textual entailment task which determins whether a "hypothesis" is true, given a "premise". The Stanford Natural Language Inference (SNLI) corpus is used. Overall, adding ELMo to the ESIM model improves accuracy by an average of 0.7% across five random seeds. A five member ensemble pushes the overall accuracy to 89.3%, exceeding the previous ensemble best of 88.9%.
Third line is Semantic role labeling task, when adding ELmo model into the task, the F1 score increases 3.2% from 81.4% to 84.6%.
Fourth line is Coreference resolution task, adding ELMo improved the average F1 by 3.2% from 67.2% to 70.4%.
Fifth line is Named entity extraction task trained on Reuters RCV1 corpus tagged with four different entity types (PER, LOC, ORG, MISC). ELMo enhanced biLSTM-CRF achieves 92.22% F1 averaged over five runs, which is 2.02% improvement
The last line is Sentiment analysis task in the Stanford Sentiment Tree- bank.Replacing CoVe with ELMo in the BCN model results in a 1.0% absolute accuracy improvement over the state of the art.



TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

(a) Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks.



	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

(b) Nearest neighbors to "play" using GloVe and the context embeddings from a biLM.

Figure 13: ELMo performance construction table [22]

Figure13(b) compares the ability to solve polysemy of both GloVe and Elmo. We can see Elmo successfully generate two embeddings for word 'play' for "play sport" and "play music" seperately, which Glove can't.

Although Elmo performs better than previous tasks to some extend, RNN structure suffers from longer sentences memorization and gradient explosion and vanish problems.

3.2.3 Transformer

Although pre-training helps LSTM capture linguistic information, the structure of LSTM models restricts the prediction ability to a short range. Besides, ELmo also very inefficient and hard to execute parallelly since the model can't proceed to t_{k+1} until t_k is calculated. As a result, attention comes into being by discarding RNN structure and only use attention mechanism. Attention mechanism can both compel sequence model and transduct model without dependent on distance in the input or output sequence. Before Transformer[23] is released, there are models conjunct attention mechanism with RNN[24]. Transformer is the first model relies entirely on self-attention to compute representations of its input and output. The structure of Transformer is illustrated in Figure14.

Encoder As shown in Figure14, encoder is made up with several components: 1.input and positional embedding 2.Multi-head attention 3.add and layer normalization 4.feed forward layer. The function and principle are as follows:

1. Input and positional embedding
   Normal embedding method in self-attention can not take position information into consideration(More detail in multi-head attention part). For example, the word embeddings of "Germany" in the sentences "The flight from Germany to China" and "The flight from China to Germany" are the same, which should not be the case. As a result, position information should be added into embedding. That's how positional embeddings come into being. The final Token vector is the sum of Input embedding and Positional embedding. More detailed are covered in github[25].

2. Multi-head attention
   Figure 15 shows the structure of multi-head attention. We can observe from the figure that multi-head is consist of h scaled dot-product attention. To make it more clear, let's first introduce dot-product attention.

   - Scaled dot product attention
     can be viewed as a information extraction process. When extract information for one single token t_n, the corresponding query vector is $q_n = W_q t_n$. Tokent_n prepares for its own answer (key,value), computed as:

     $$k_n = W_k x_n$$
     $$v_i = W_v x_i$$

     Notice that key and value generation is irrelevant to query.
     The Correlation coefficient of every token t_n to token t_i can be illustrated as :

     $$score(q_n, k_i) =< q_n, k_i > /\sqrt{d}, i = 1, 2, ...L$$
     $$a_{ni} = \frac{exp^{score(q_n,k_i}}{\sum_{j=1}^{L} exp^{score(q_n,k_j}}$$

     <,> is dot product operation, d is vector dimention. The reason why the dot product need to be divided by $\sqrt{d}$ is to avoid the product lands in saturation area.
     The new vector of token t_n is:

     $$t'_n = \sum_{i=1}^{L} t_{ni} * v_i$$

     Mask is an optional operation here. If the length of sequence is shorter than batch length, <PAD> should be filled at the end of sentence token. Before proceeding to correlation coefficient calculation, those placeholder tokens should be masked.
     In practise all tokens are generated simultaneously as shown in the right of Figure15. Queries and keys matrix are of dimension L(Sentence length) * d_k(vector features dimention), and values of L * d_v. The new token vector $[t'_1, ..., t'_L]$ can be expressed as :

     $$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}} V)$$

- Multi-head attention
  Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. So that the model can integrate multiple aspects of information. With a single attention head, averaging inhibits this:
  MultiHead(Q, K, V) = Concat(head1, ..., headh)W^O
  where $head_i$ = Attention(QW_i^Q, KW_i^K, VW_i^V)

3. add and layer normalization
   Layer normalization is similar to Batch normalization[26], the difference is batch normalization normalize on features level, but layer normalization normalize on batch level. Add operation references to resNet[27] It adds original token to generated tokent' at each layer to avoid input degeneration.

4. feed forward layer
   The main goal of feed forward layer is to project the vector generated from multi-head attention to a larger space, which is easier to retrieve valuable information, and then projected into it's original space. SVM[28] also utilizes similar principle to project feature to higher dimensional space to make features linear separable.

Decoder Decoder is similar to encoder, except the multi-head attention is masked, which means when predicting token t_k it can only use information from previous tokens $[t_1, ..., t_{k-1}]$. And decoder also inserts a third sub-layer, from which encoder stack pass over the K,V matrix it learned to decoder.

The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much larger vector called logits vector.

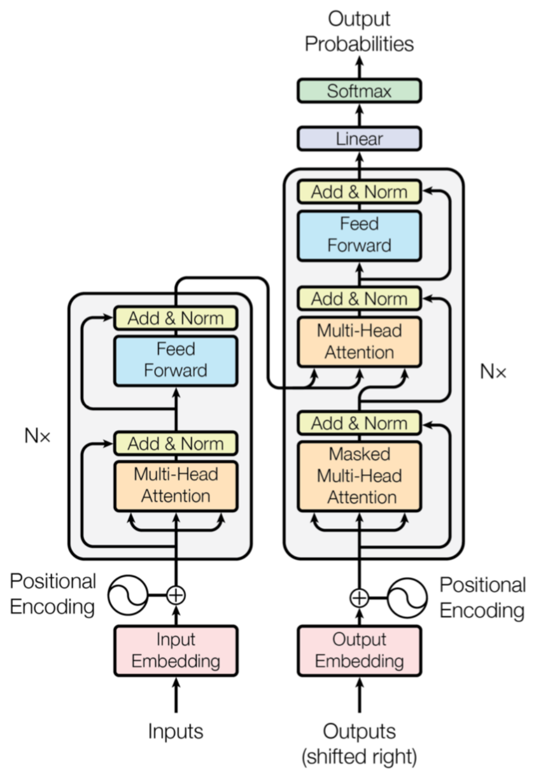


Figure 14: Architecture of Transformer [23]

For example, the size of training corpus is 10,000, which means the logits vector is 10,000 cells wide – each cell corresponding to the score of a unique word. That is the way to interpret the output of the model followed by the Linear layer.

The softmax layer then turns those scores into probabilities. The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

One thing worth paying attention to is Transformer is not a Language modeling model. Because in decoder it receives the whole encoding of original sentence from encoder, which can be seen as a cheat when generating contexts. However, either encoder or decoder method can be applied in language model since they are good at handling long-term dependencies in text, compared to other alternatives such as neural network. BERT upgrade its performance upon encoder, and CPT,CPT2,Transformer XL, XLNet refine the decoder part.

3.2.4 BERT

Elmo,GPT,GPT2,Transformer XL are all unidirectional language models, which means they can only attend to previous tokens in self-attention layers. As a result, they perform not quite satisfying when fine tuning token-level tasks such as question answering, where context from both directions are crucial.
BERT(Bidirectional Encoder Representations from Transformers)[29] is the first model to learn bidirectional presentation from unlabeled data in pretraining phrase by jointly conditioning on both context from left and right, after which the model can be fine tuned by just adding one extra output layer for a wide range of tasks. BERT achieves that by using "masked language model" as input and "next sentence prediction" task, which offers text-pair representations.

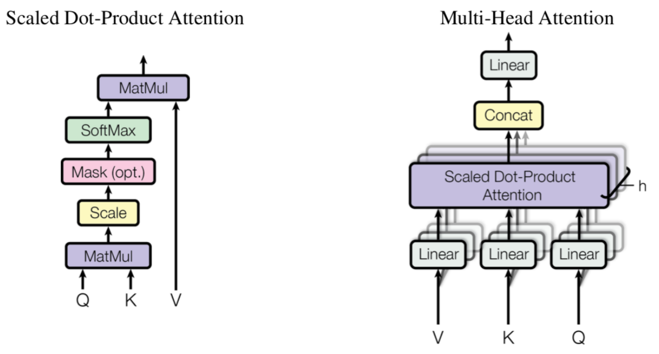


Figure 15: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.[29]

Pre-training In Pre-training, the model is trained by unlabeled data over different tasks. BERT uses Multi-layer bidirectional encoder based on encoder from Transformer[23].

Input/Output representation : BERT supports both single sentence and pair sentence <Question,Answer> representation in one token. For single sentence representation, Elmo uses WordPiece embeddings[30] which has 30,000 token vocabulary. The first token of each sentence is always a special classification token called [CLS], which encodes the overall information of the whole sentence. The final hidden state of [CLS] is used for classification tasks. For pair sentence representation, BERT separates them with a special token [SEP], besides, it also has one extra embedding to identify whether the sentence belongs to sentence A or sentence B. The final hidden layer of [CLS] is also used for further tsks. As shown in Figure 16, the input representation is consisted by summing 3 tokens:token embeddings, segment embeddings and position embeddings.

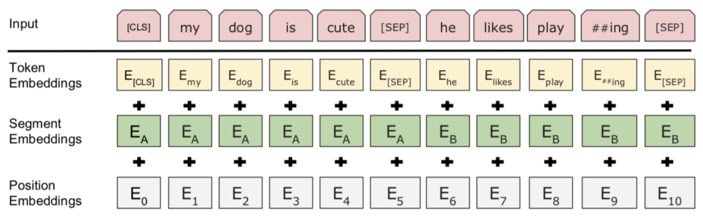


Figure 16: BERT input representation.[29]

Masked LM: Bi-directional information can't be acheived by simple concatenation of left-to-right and right-to-left as used in BERT. However, conditioning on both direction would let each word see itself. In order to solve this dilemma BERT masks a certain percentage of input token randomly and predict the [MASK] tokens. The shortcoming of the approach is there are mismatchs between pre-training and fine-tuning process, bacause there is no [MASK] tokens in fine-tuning. To alleviate the weakness, the strategy is 80% of the time replace the word with the [MASK] token, 10% of the time replace the word with a random word and 10% of the time keep the word unchanged. The final hidden state of masked token is fed into a softmax function over the whole vocabulary.

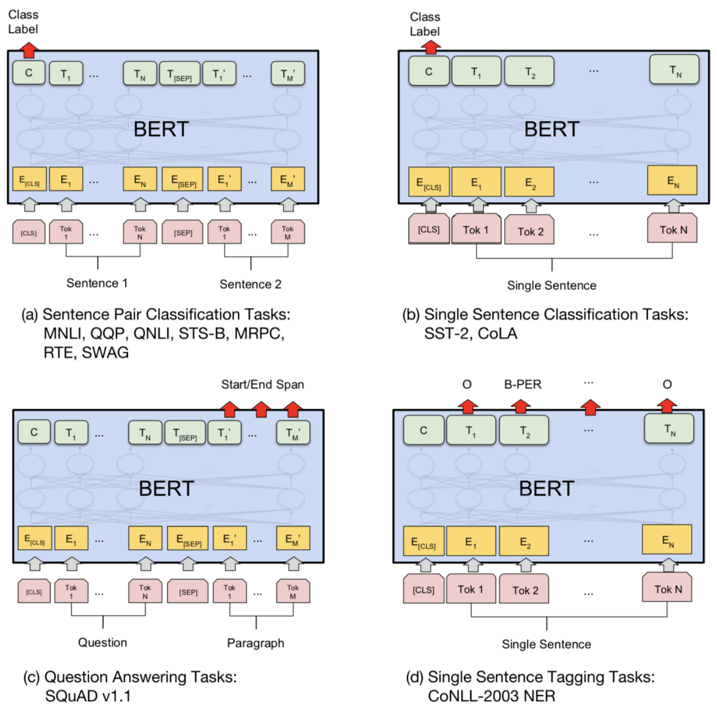
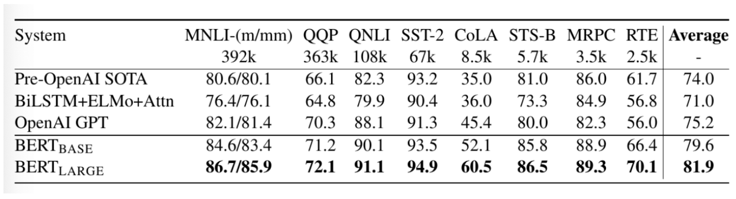

Class Label
BERT
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG
Class Label
BERT
Single Sentence
(b) Single Sentence Classification Tasks: SST-2, CoLA
Start/End Span
BERT
Question
Paragraph
(c) Question Answering Tasks: SQuAD v1.1
O
B-PER
O
BERT
Single Sentence
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figure 17: Illustrations of Fine-tuning BERT on Different Tasks.[29]



System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT$_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Figure 18: GLUE Test results [29]

Next sentence prediction: In order to extract the relation between sentence, next sentence prediction task is implemented by BERT. When trianing the task, 50% cases sentence A and B are sequential, 50% cases sentence A and B have no relationship. For example:

Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]
Label = IsNext
Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight not birds [SEP]
Label = NotNext

The task is a huge success by achieving 97%-98% accuracy.

Fine-tuning In fine-tuning, all parameters got from pre-training are fine-tuned using labeled data from downstream tasks. Figure17 is an illustration of BERT fine-tuning tasks.
For Simple Simplification taks(b), we proceed the special token [CLS] at the first position to a softmax layer and uses the output for fine-tuning.
For Sentence Pair classification tasks, we add segmentation tokens in pre-triaining and also uses softmax on [CLS] for futher fine-tuning.
For Single Sentence Tagging task(d) such as name entity recognision, except [CLS] and [SEP], all tokens generate a new tag. In this case, B-PER means the Beginning of a PERson. BERT use the output tags to fine-tuning. Question Answering task(c) is also a token-level task as (d) . It input a question and corresponding paragraph where answer is contained. It seeks answer in output tags.

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
BERT$_{\text{BASE}}$ (Single)	80.8	88.5	-	-
BERT$_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
BERT$_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
BERT$_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
BERT$_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

(a) SQuAD 1.1 results

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
BERT$_{\text{LARGE}}$ (Single)	78.7	81.9	80.0	83.1

(b) SQuAD 2.0 results

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
BERT$_{\text{BASE}}$	81.6	-
BERT$_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

(c) SWAG Dev and Test accuracy

Figure 19: BERT performance construction table[29]

Experiments BERT is conceptually simple and also empirically powerful. Figure 18 shows the examine result on GLUE, we can see BERT model obtains an accuracy score of 80.5, compared to OpenAI GPT, which obtains 72.8 as of the date of writing. Figure 19(a) is the result on Stanford Question Answering Dataset (SQuAD v1.1), which is a collection of 100k crowd sourced question/answer pairs. The task is to predict the answer text span in the passage given a question and a passage. We can see BERT model outperforms the top ensemble sys- tem in terms of F1 score.
Figure 19(b) is the extension of SQuAD 1.1, it makes the problem more realistic by avoiding short answers. We can see from the table that a +5.1 F1 improvement over the previous best system.

Figure 19(c) test accuracy on Situations With Adversarial Generations (SWAG) dataset, which contains 113k sentence-pair completion examples that evaluate grounded common- sense inference. The results shows BERT outperforms the authors' baseline ESIM+ELMo sys- tem by +27.1% and OpenAI GPT by 8.3%.

3.2.5 GPT

GPT[31] consists of two stages. The first stage is using masked multi-head-attention model(same as Transformer) to pre-train unlabeled data, this is followed by adapting a discriminative task on the model to fine-tuned labeled data.

Unsupervised pre-training Just like other language modeling method, the loss function of GPT is also the log probability of sentence generation:

$$L_1(U) = \sum_i logP(u_i|u_{i-k}, ..., u_{i-1}; \Theta)$$

where k is the size of context window. Θ is the parameters of the neural network the model uses. Note that GPT only capture uni-directional information.
Context tokens first go through a multi-headed self-attention layer followed by position-wise feedforward layers to produce an output distribution over target tokens:

$$\begin{gathered} h_0 = UW_e + W_p \\ h_l = transformer_block(h_{l-1} \forall i \in [1, n] \\ P(u) = softmax(4_n W_e^T) \end{gathered}$$

where U is context vector, n is layer number, and W_e is token embedding matrix W_p is positional embedding matrix.

Supervised fine-tuning A sequence of input tokens $x^1, ..., x^m$ are first passed through pre-trained model to obtain the final transformer block's activation h_l^m, which is fed into a linear output layer added in fine-tuning stage with parameters W_y to predict final output y:

$$P(y|x^1, ..., x^m) = softmax(h_l^m W_y)$$

The corresponding loss function is:

$$L_2(C) = \sum_{x,y} logP(y|x^1, ..., x^m)$$

GPT optimises the overall loss function:

$$L_3(C) = L_2(C) + \lambda * L_1(C)$$

where λ is the weight of loss function L_1 .

3.2.6 GPT2

GPT2[32] demostrates there is no need to fine-tune on a specific model if the model is trained on a new dataset of millions of webpages called WebText. GPT2, can be seen as a extension of GPT, is a 1.5B parameter Transformer which achieves SOA results on 7 out of 8 datasets now.
The main idea of GPT2 can be expressed as follows: The goal of language modeling is to estimate the probability of $P(x) = \prod_{i=1}^{n} p(s_n|s_1, ..., s_{n-1})$, which can be rewritten as $p(output|input)$. To perform a specific task, the model should not only condition on input but also on the task, which mean the formula should like this:$P(output|input, task)$. The task can also be extracted as a vector just as input and output. GPT2 holds the idea that supervised objective is the same as unsupervised objective but only evaluated on a subset of sequence. For example, if the input sentence is : "The translation of word Machine Learning in Chinese is Machine Learning" , the model can learn translation task and information retrieval task simultaneously. The problem becomes whether GPT2 is able to, in practice, optimize the unsupervised objective to convergence. As a result, GPT2 speculates that a language model with sufficient capacity will begin to learn to infer and perform the tasks demonstrated in natural language sequences by unsupervised learning.
There are several changes compared to GPT model. First, the choice of training data set is different.

GPT2 collects data from many different domains by web scrapes, which contains over 8 million documents for a total of 40GB text. Second, it changes the strategy for OOV problem. Since byte-level embedding is not as good as word-level embedding, GPT2 uses a compromise method proposed by Sennrich[33] which only split rare word during training. Last but not least, the model structure is modified.

	LAMBADA	LAMBADA	CBT-CN	CBT-NE	WikiText2	PTB	enwik8	text8	WikiText103	1BW
	(PPL)	(ACC)	(ACC)	(ACC)	(PPL)	(PPL)	(BPB)	(BPC)	(PPL)	(PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**	**40.31**	**0.97**	**1.02**	22.05	44.575
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**	42.16

Figure 20: Zero-shot results on many datasets. [32]

Question	Generated Answer	Correct	Probability
Who wrote the book the origin of species?	Charles Darwin	✓	83.4%
Who is the founder of the ubuntu project?	Mark Shuttleworth	✓	82.0%
Who is the quarterback for the green bay packers?	Aaron Rodgers	✓	81.1%
Panda is a national animal of which country?	China	✓	76.8%
Who came up with the theory of relativity?	Albert Einstein	✓	76.4%
When was the first star wars film released?	1977	✓	71.4%
What is the most common blood type in sweden?	A	✗	70.6%
Who is regarded as the founder of psychoanalysis?	Sigmund Freud	✓	69.3%
Who took the first steps on the moon in 1969?	Neil Armstrong	✓	66.8%
Who is the largest supermarket chain in the uk?	Tesco	✓	65.3%
What is the meaning of shalom in english?	peace	✓	64.0%
Who was the author of the art of war?	Sun Tzu	✓	59.6%
Largest state in the us by land mass?	California	✗	59.2%
Green algae is an example of which type of reproduction?	parthenogenesis	✗	56.5%
Vikram samvat calender is official in which country?	India	✓	55.6%
Who is mostly responsible for writing the declaration of independence?	Thomas Jefferson	✓	53.3%
What us state forms the western boundary of montana?	Montana	✗	52.3%
Who plays ser davos in game of thrones?	Peter Dinklage	✗	52.1%
Who appoints the chair of the federal reserve system?	Janet Yellen	✗	51.5%
State the process that divides one nucleus into two genetically identical nuclei?	mitosis	✓	50.7%
Who won the most mvp awards in the nba?	Michael Jordan	✗	50.2%
What river is associated with the city of rome?	the Tiber	✓	48.6%
Who is the first president to be impeached?	Andrew Johnson	✓	48.3%
Who is the head of the department of homeland security 2017?	John Kelly	✓	47.0%
What is the name given to the common currency to the european union?	Euro	✓	46.8%
What was the emperor name in star wars?	Palpatine	✓	46.5%
Do you have to have a gun permit to shoot at a range?	No	✓	46.4%
Who proposed evolution in 1859 as the basis of biological development?	Charles Darwin	✓	45.7%
Nuclear power plant that blew up in russia?	Chernobyl	✓	45.7%
Who played john connor in the original terminator?	Arnold Schwarzenegger	✗	45.2%

Figure 21: The 30 most confident answers generated by GPT-2 on the development set of Natural Questions sorted by their probability according to GPT-2.[32]

GPT2 achieves state-of-the-art scores on a variety of domain-specific language modeling tasks. GPT2 is not trained on any of the data specific to any of these tasks and is only evaluated on them as a final test; this is known as the "zero-shot" setting. Figure20 is about language modeling results, note that PPL is the abbreviation for Perplexity, the fewer the value is, the higher possibility a sentence has, the better the model is. BPC means bits-per-character, it is proposional to PPL. ACC standes for accuracy. Large improvements are noticed on small datasets such as Penn Treebank and WikiText-2 which have only 1 to 2 million training tokens and also on large datasets created to measure long-term dependencies like LAMBADA and the Children's Book Test. However, GPT2 is still worse than prior work on the One Billion Word Benchmark. This is likely due to 1BW's sentence level shuffling removes all long-range structure.

As for question answeing tasks, they are trained on Natural Questions dataset[34],which is a promising resource to test this more quantitatively. GPT2 has an accuracy of 63.1% on the 1% of questions it is most confident in. However, the performance of GPT2 is still much, much, worse than the 30 to 50% range of open domain question answering systems which hybridize information retrieval with extractive document question answering.

3.2.7 Transformer XL

TransformerXL[35] is an improvement model on Transformer, which learns dependency beyond a fixed length in sequence and resolves context fragmentation problem. Before we proceed to the mechanism of TransformerXL, let's have a refresh on the defect of Transformer model.

Segment-Level Recurrence with State Reuse As mentioned above, there is an optional mask operation in the Transformer attention part, which cuts or pads one sentence segment into a certain length. By doing so Transformer transfers all the input sequence into many fixed length segments, without any information flow across segments. However, it may leads to some problems:
First, the model cannot capture any longer-term dependency beyond the predefined context length. Besides, each fixed length segments are generated by consecutive symbols inside the segment without referring to any adjacent segments. Hence, the model neglects potential necessary contextual information which are essential to predict the several beginning words, leading to inferior performane. For example, if a long sentence is cutted into a half, the first several words in second segment will be hard to predict. We refer to this kind of problem as context fragmentation, as shown in figure22(a).

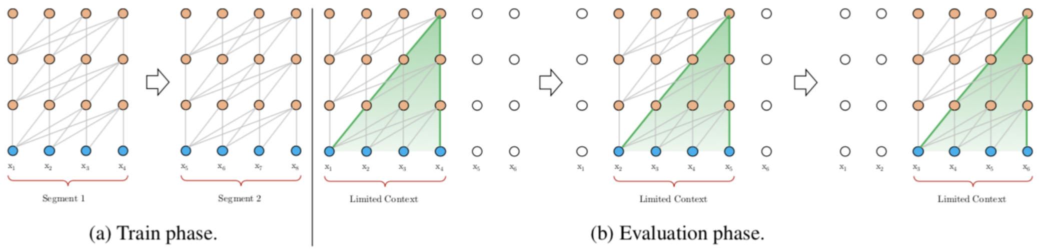


Figure 22: Illustration of the vanilla Transfer model with a segment length 4.[32]

During each step in evaluation stage, the segment is shifted to the right by only one position. and the whole segment needs to be processed from scratch. As we can see from Figure21(b), when predicting x_6, $[x_2, x_3, x_4, x_5]$ need to be calculated. Then when predicting x_7, we need to according to $[x_3, x_4, x_5, x_6]$, which doesn't make use of knowledge learned before. This kind of evaluation procedure is extremely expensive, can we save the output from the previous segment and utilize that in the next segment evaluation?

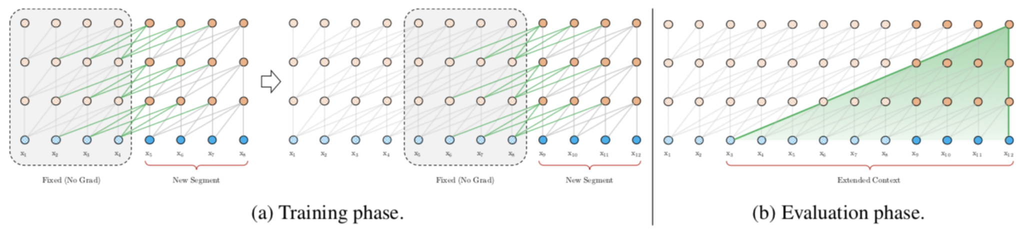


Figure 23: Illustration of the TransferXL with a segment length 4.[35]

That is exactly the main idea of TransformerXL. During training, the hidden state sequence calculated at previous segment is fixed and cached to be reused as a extra information for the next new segment. As shown in Figure 23(a). In evaluation phase, when computing $x_1 2$, $[x_1 1, ..., x_3]$ don't need to be recalculated. Detailed formula can be seen in Zihang's paper[35].

Relative Positional Encodings Transformer uses absolute position in the segment for encoding. However, due to reuse of hidden states, TransformerXL could get confused by which segment position the index refers to. As a result, TransformerXL needs to think up a new encoding method to keep the positional information. The detail implementation could be find in kim's github[36]

3.2.8 XLNet

XLNet[37] can be seen as an improvement on BERT, which neglects dependency between masked positions and suffers from pretrain-fintune data discrepancy, by maximizing the expected probability over all permutations of factorization order. Thanks to permutation operation, XLNet can also support bi-directional context detection. Besides, XLNet integrates the segment-level recurrence mechanism from Transfer-XL into pretraining.

Shortcoming of BERT BERT suffers from discrepancy caused by [MASK] (introduces at section 3.2.3), and oversimply assumes predicted tokens are irrelavent. To understand that, we first introduce the concept of Autoregressive model(AR) and Autoencoder model(AE):
In AR language model, when there is a text sequence x=$(x_1, ..., x_T)$, AR will factorize the sequence likelihood into a forward product $P(x) = \prod_{t=1}^{T} p(x_t|x_{<t})$ or a backward product $P(x) = \prod_{t=T}^{1} p(x_t|x_{>t})$. The task of AR model is maximize the likelihood of those product:

$$max_\theta log p_\theta(x) = \sum_{t=1}^{T} logo_\theta(x_t|x_{<t}) = \sum_{t=1}^{T} log\frac{exp(h_\theta(x_{1:t-1}^T e(x_t)}{\sum_{x'} exp(h_\theta(x_{1:t-1}^T e(x'))}$$

where $h_\theta(x_{1:t-1}$ denotes a context representation encoded by RNN or Transformer. e(x) is the embedding of x. Owing to the structure of AR language model, it can only encode a uni-directional context either forward or backward, which is not effective at bi-directional context modeling. Many models are based on AR language model such as GPT,GPT2,TransformerXL and XLNet. However, XLNet solves this problem by permutation operation.
AE language model aims to reconstruct original sentence sequence from corrupted input. An example of AE language model is BERT, who try to recover the original tokens from masked input. The structure allos AE language model to see the whole sentence sequence. In other words, AE supports bidirectional context for reconstruction. The expression to reconstruct masked tokens $\bar{x}$ from corrupted input $\hat{x}$ is:

$$max_\theta log p_\theta(\bar{x}|\hat{x}) \approx \sum_{t=1}^{T} m_t log p_\theta(x_t|\hat{x}) = \sum_{t=1}^{T} m_t log\frac{exp(H_\theta(x)_t^T e(x_t))}{\sum_x' exp(H_\theta(x)_t^T e(x'))}$$

where m_t = 1 indicates x_t is a masked token and H_θ is a transformer which maps a text sequence x with length T into a sequence of hidden vectors $H_\theta(x) = [H_\theta(x)_1, ..., H_\theta(x)_T]$

Notice that the expression in AE model uses Approximate equality sign $\approx$ instead of = in AR model. It means BERT factorizes the joint conditional probability $p(\bar{x}|\hat{x})$ according to the assumption that all masked tokens $\bar{x}$ are constructed separately, which doesn't make sense. For example, if the masked token for the sentence "New York is a city" is New and York, then they are correlated. By constract, AR model fits better since it's not based on independence hypothesis.

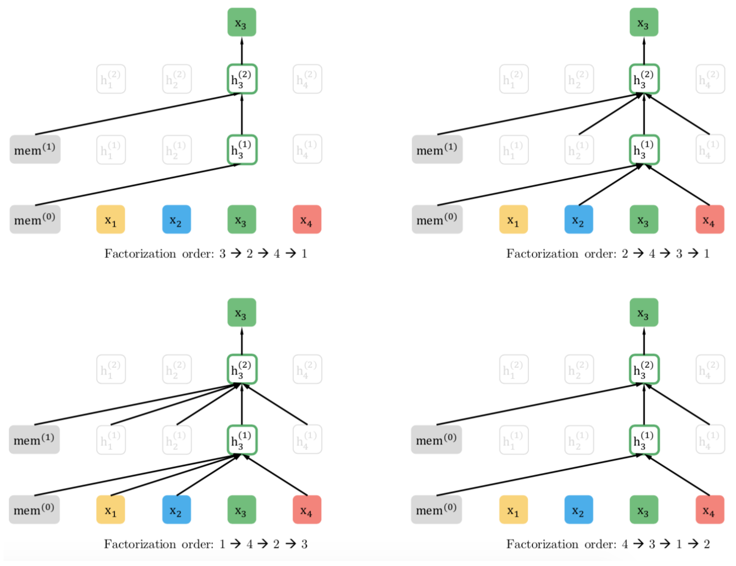
Improvement on XLNet The mechanism makes XLNet integrates all the advantages of both AE and AR model while avoiding their weaknesses are introduced as following:

1. Permutation
   For a sequence of length T, there are T! different orders to perform a valid auto-regressive factorization. For example, if T = 3, then the input sequence is $x = x_1 x_2 x_3$ and there are

3!=6 sequence orders:

$$
\begin{aligned}
p(x) &= p(x_1)p(x_2|x_1)p(x_3|x_1x_2) => 1->2->3 \\
p(x) &= p(x_1)p(x_2|x_1x_3)p(x_3|x_1) => 1->3->2 \\
p(x) &= p(x_1|x_2)p(x_2)p(x_3|x_1x_2) => 2->1->3 \\
p(x) &= p(x_1|x_2x_3)p(x_2)p(x_3|x_2) => 2->3->1 \\
p(x) &= p(x_1|x_3)p(x_2|x_1x_3)p(x_3) => 3->1->2
\end{aligned}
$$

Notice that $p(x_2|x_1x_3)$ represents for the probability of x_2 being the second word when the first word is x_1 and the third word is x_3. The sequence follows the probability is the dependency relationship. For example, 1->3->2 means the model guess the first word, and based on first word guess the third word, then the second one. We can see the permutation only permutes the factorization order, not the real sequence order, which means it keeps the original sequence order. It is achieved by masking mechanism in Transformer. If model parameters are shared across all factorization orders, theoretically, the model will learn to gather information from all positions on both sides.
Calculating T! possibilities is very time consuming. Hence, in practise we only randomly select t partial samples in T! permutations.



Factorization order: 3 → 2 → 4 → 1
Factorization order: 2 → 4 → 3 → 1
Factorization order: 1 → 4 → 2 → 3
Factorization order: 4 → 3 → 1 → 2

Figure 24: Illustration of the permutation language modeling objective for predicting x3 given the same input sequence x but with different factorization orders. [37]

Let us formalize the idea of permutation. Z_T represents for all possible permutation of sentence of length T. Then lst z_t and $z_{<t}$ be the t-th element and first t-a elements of one of permutations $z \in Z_T$.
The expected permutation language modeling can be expressed as:

$$max_\theta E_z Z_T[\sum_{t=1}^{T} logp\theta(x_{z_t}|x_{z<t})]$$

Basically, the expression means choose the most possible factorization order by adding the likelihood $p_{(x)}$ according to factorization order.
Figure 24 provides an overall picture of permutation. The factorization order of the top-left sub picture is 3->2->4->1, as a result, we can't attend to other tokens when predicting x3. As for bottom left sub picture, we can attend to all three other tokens.

2. Two Stream Self-Attention for Target-Aware Representation
While permutation solves bi-directional modeling problem, naive implementation with standard Transformer doesn't work. To be more specific, assume the input sentence is "I like New York", and the permutation factorization order is z=[1,3,4,2], if we want to predict z_3=4, then according to formula:

$$p_\theta(X_{z_3} = x|x_{z_1 z_2} = p_\theta(X_4 = x|x_1 x_3) = \frac{exp(e(x)^T h_\theta(x_1 x_3))}{\sum_{x'} exp(x(x')^T h_\theta(x_1 x_3))}$$

$p_\theta(X_4 = x)$ means the prob of forth word being York. And translate the formula above into natural language is : When the first work is I, the third word is New, what's the possibility of the York being the forth word.
We assume another permutation order z'=[1,3,2,4] and we want to get the pro of z_3 =2:

$$p_\theta(X_{z_3} = x|x_{z_1 z_2} = p_\theta(X_2 = x|x_1 x_3) = \frac{exp(e(x)^T h_\theta(x_1 x_3))}{\sum_{x'} exp(x(x')^T h_\theta(x_1 x_3))}$$

It calculates the probability of York being the second word when the first word is I and the third is New. Note that the above two formulas are the same, which does not make sense. Since New York is a city and York New means nothing.
Notice the hidden layer $h_\theta(x_{z<t})$ of Transformer network does not depend on the position it predicts,i.e., z_t. As a result, the same distribution is predicted regardless of the target position, which is not able to learn useful representation like we mentioned above.
To avoid that, re-parameterize the next-token distribution to be target position aware:

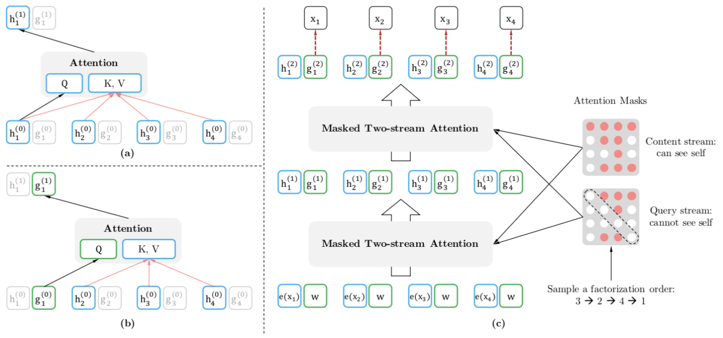
$$p_\theta(X_{z_t} = x|x_{z<t}) = \frac{exp(e(x)^T g_\theta(x_{z<t}, z_t))}{\sum_{x'} exp(e(x/)^T g_\theta(x_{z<t}, z_t))}$$

where $g_\theta(x_{z<t}, z_t)$denotes a new representation, which additionally take the current position z_t as input.

The idea of target-aware representation g_θ removes the uncertainty in target prediction, but how to formulate g_θ remains a unsolved problem. Notice that there are contradictory in Transformer architecture: (1) g_θ should only use position of target token z_t but not the context x_{z_t}, otherwise the word can see itself (2)to predict following tokens, g_θ should also encode context x_{z_t} to provide full contextual information.
To resolve such delimma, XLNet proposes two set of hidden representations, namely, two streams:
The content representation: $h_\theta(x_{z\le t})$ or abbreviated as h_{z_t}, which plays a similar role as standard Transformer. It encodes both the context and x_{z_t} itself.
The query representation $g_\theta(x_{z<t}, z_t)$ or abbreviated as g_{z_t} , which only has access to contextual information $x_{z<t}$ and position z_t .
Computationally, we initialize first query representation $g_i^{(0)}$ as a variance w, and the content representation as correspoding word embedding,i.e., $h_i^{(0)}$ = $e(x_i)$. For each i self-attention layer m = 1, . . . , M, each layer of query and content representation is:

$$g_{z_t}^{(m)} < -Attention(Q = g_{z_t}^{(m-1)}, KV = h_{z<t}^{(m-1)}; \Theta) \text{ (query stream: use } z_t \text{ but not } x_{z_t})$$
$$h_{z_t}^{(m)} < -Attention(Q = h_{z_t}^{(m-1)}, KV = h_{z\le t}^{(m-1)}; \Theta) \text{ (content stream: use both } z_t \text{ and } x_{z_t})$$

where Q, K, V denote the query, key, and value in an attention operation as Transformer.
During fine-tuning, XLNet can simply drop the query stream and use the content stream as a normal Transformer(-XL). Finally, the last-layer query representation $g_{z_t}^{(m)}$ is used to compute $p_\theta(X_{z_t} = x|x_{z<t})$
Figure 25 shows the calculation process of two stream permutation model. Top-left graph is attention calculation of Content Stream. Assuming the permutation order is 3->2->4->1, then when we predict the token at first position, we can reference to all 4 word message. Hence, KV = [$h_1^{(0)}, h_2^{(0)}, h_3^{(0)}, h_4^{(0)}$], and Q=$h_1^{(0)}$.
Bottom-left graph is process of Query stream calculation. Because Query stream doesn't know content of itself, KV=[$h_2^{(0)}, h_3^{(0)}, h_4^{(0)}$] and Q= $g_1^{(0)}$.
sub-graph (c) is the whole process of two stream calculation: Let start from the bottom to the top. h and g are initialized with $e(x_i)$ and W as mentioned above, then first layer output $h^{(0)}$ and $g^{(0)}$ of Content Mask and Query Mask are generated, the second layer,third layer...are all the same. Note that the first line of Content Mask is all red, which means the first word can attend to all word in sentence(1,2,3,4), while Query Mask can't attend to the word itself, so the diagonal of the matrix is while.



Attention
Q
K, V
(a)
(b)
Masked Two-stream Attention
Attention Masks
Content stream: can see self
Query stream: cannot see self
Sample a factorization order: 3 → 2 → 4 → 1
(c)

Figure 25: (a): Content stream attention, which is the same as the standard self-attention. (b): Query stream attention, which does not have access information about the content x_{z_t}) . (c): Overview of the permutation language modeling training with two-stream attention.[37]

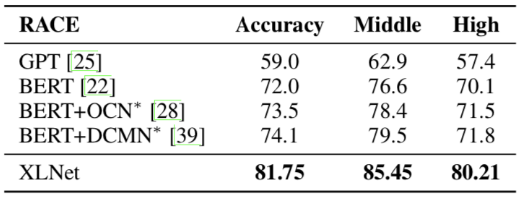
3. Partial Prediction
   While there are many advantages in permutation language modeling, it is slow to convergence due to the amount of possibilities. To reduce the difficulty, XLNet only predict the last tokens in a factorization order. The reason is the last several tokens possesses the longest context in the sequence given a certain factorization order.
   XLNet divided the whole factorization order into two sub-sequences $z_{\leq c}$ and $z_{>c}$, corresponding to non-target set and target set separately. A parameter K is introduced, $\frac{1}{K}$ tokens will be selected for prediction.

4. Incorporating Ideas from Transformer-XL
   XLNet integrate two main function from Tramsformer-XL, namely the relative positional encoding scheme and the segment recurrence mechanism. Both of them are elaborated in Transformer-XL

5. Modeling Multiple Segment
   Many downstream tasks have multiple input sequence such as question answering. Just like BERT, we choose two sentences, with 50% being continuous sequence and 50% dis-contiguous sequence. Then concat two segments as one sequence to perform permutation language modeling. Specifically, the input to XLNet is similar to BERT: [A, SEP, B, SEP, CLS], where "SEP" and "CLS" are two special symbols and "A" and "B" are the two segments. There is a slight different from BERT, here CLC is put to the end. Because self-attention calculation doesn't need to take position into consideration since BERT encode the position information to input vector. While XLNet only predict the last $\frac{1}{K}$ tokens, it put CLS at the end to guarantee all message is covered.
   There is one more different in BERT and XLNet, that XLNet uses relative segment encoding while BERT adds an absolute segment embedding to each position of word embedding. XLNet only cares whether two positions are within the same segment, instead of considering which specific segments they are from. When i attends to j, the segment encoding s_{ij} is used to compute an attention weight a_{ij} = (q_i + b)Ts_{ij}, where q_i is the query vector as in a standard attention operation and b is a learnable head-specific bias vector. Finally, the value a_{ij} is added to the normal attention weight.
   Using relative segment encoding has two benefits. First, the inductive bias b improves generalization. Second, it makes XLNet support fine-tuning on more than two input segments, which is not possible using absolute segment encoding.

Experiments First experiment is tested on RACE dataset, which contains near 100K questions taken from the English exams for middle and high school Chinese students in the age range between

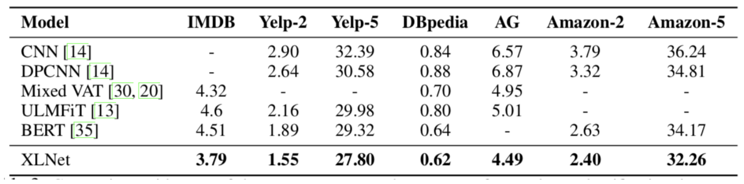
12 to 18, with the answers generated by human experts. The average length of the passages in RACE are longer than 300, which is significantly longer than other popular reading comprehension datasets such as SQuADAs. we can see from the result table, XLNet outperforms the SOA model by 7.6 points in accuracy and also surpassed other models such as BERT and GPT. One possible reason for that huge improvement maybe XLNet integrate Transformer-XL architecture which improves the capability of modeling long text.



RACE	Accuracy	Middle	High
GPT [25]	59.0	62.9	57.4
BERT [22]	72.0	76.6	70.1
BERT+OCN* [28]	73.5	78.4	71.5
BERT+DCMN* [39]	74.1	79.5	71.8
XLNet	**81.75**	**85.45**	**80.21**

Figure 26: Comparison with state-of-the-art results on the test set of RACE [37]

For text classification task, as shown in figure 27, XLNet is evaluated on the following benchmarks: IMDB, Yelp-2, Yelp-5, DBpedia, AG, Amazon-2, and Amazon-5. XLNet achieves better performance on all considered datasets, reducing the error rate by 16%, 18%, 5%, 9% and 5% on IMDB, Yelp-2, Yelp-5, Amazon-2, and Amazon-5 respectively compared to BERT.



Model	IMDB	Yelp-2	Yelp-5	DBpedia	AG	Amazon-2	Amazon-5
CNN [14]	-	2.90	32.39	0.84	6.57	3.79	36.24
DPCNN [14]	-	2.64	30.58	0.88	6.87	3.32	34.81
Mixed VAT [30, 20]	4.32	-	-	0.70	4.95	-	-
ULMFiT [13]	4.6	2.16	29.98	0.80	5.01	-	-
BERT [35]	4.51	1.89	29.32	0.64	-	2.63	34.17
XLNet	**3.79**	**1.55**	**27.80**	**0.62**	**4.49**	**2.40**	**32.26**

Figure 27: Comparison with state-of-the-art error rates on the test sets of several text classification datasets.[37]

4 Application

In this research paper, we include 3 modalities: text, protein sequence and source code, most of which are based on the general models mentioned above. All of them use language modeling models trained from self-supervised learning.

4.1 Text

4.1.1 Unsupervised Domain Adaption on Reading Comprehension

Tags:*Reading Comprehension, Domain Adaption,general domain, Transfer Learning*

- Background
  In NLP study, Reading Comprehension(RC) is a widely studied topic since it makes human-machine interaction more feasible. There are several datasets to train an end to end deep neural network for Reading Comprehension such as SQuAD[38], CoQA[39] and NewsQA[40]. Recently, Bert even performs better than human on SQuAD dataset. However, in real world applications not all data are labeled. Therefore a performance gap can be seen when transfering a model trained on source tasks to goal task.
  Unsupervised domain adaption is a method to solve the generalization problem. It is used to transfer knowledge from a labeled source domain to an unlabeled target domain. Before the paper is released, there are few domain adaption tasks focus on Reading Comprehension,

However, they are only applied to small RC dataset, which is not helpful to generalize representation.

- Model
The paper [41] concentrates on unsupervised domain adaption on RC datasets using Transfer Learning based on BERT modeling model. The adaption method this paper proposes called Conditional Adversarial Self-training(CASe). The method can be divided into two parts: First, fine-tune BERT model on source domain, then, at adaption stage, it applies alternated training strategy which consists of self-training and conditional adversarial learning.
The definition of text span based Reading Comprehension task can be describe as follows: Given a M token paragraph P = <$p_1, p_2, ..., p_M$> and q query Q=<$q_1, q_2, ..., q_L$> with L tokens, the corresponding answer A is <$p_{a^s}, p_{a^{s+1}}, ..., p_{a^e}$>. The goal is to find (a^s, a^e), which is the answer span of paragraph, satisfies the condition $0 \leq a^s \leq a^e \leq M$.
In unsupervised domain adaption task for Reading Comprehension, there are two data domains: one source domain and one target domain. In source domain, we have n labeled samples$(x_i, y_i)_{i=1}^n$, where x_i =(P_i, Q_i) and label $y_i = ({a_i}^s, {a_i}^e)$. In target domain we only have unlabeled samples $({x_j}')_{j=1}^{n'}$. The distribution of source domain and target domain are different, our goal is to find model which can reduce the distribution shift and generalize well between those domains.
CASe approach is illustrated in Figure below . As we can see from the graph, CASe if made up with 3 components: a BERT model, an self training network on target domain and a discriminator network.
Firstly, BERT is chosen as a pre-trained model, since BERT is trained on a huge corpus which makes sure that it can generate universal feature representations. Secondly, self training is used on target domain to predict sample labels in target domain. We assume that some predicted answers in source domain will be similar to that in target domain. Those similar samples $x' = (P, Q)$ in target domain is called pseudo-labeled samples. A filter is used at this stage to prevent model from learning target domain distribution wrongly on pseudo labels. Low similarity samples will be filtered out. Thirdly, conditional adversairal learning on both source and target domains is applied to minimize cross-domain discrepancy between source domain and target domain. Note that second and third steps are proceed iteratively.

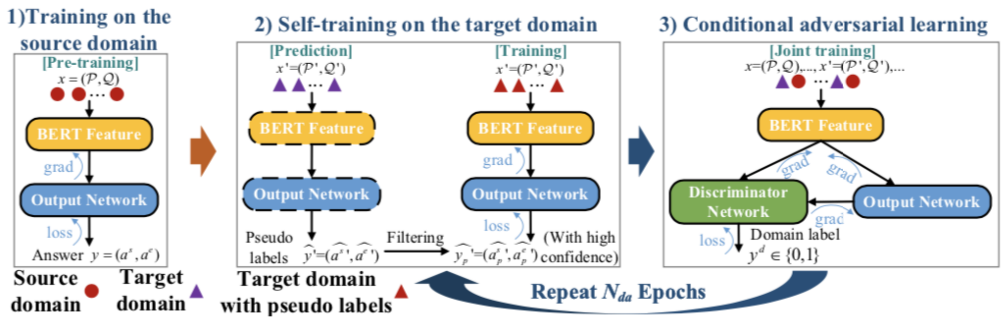


Figure 28: Framework of CASe, First step: training BERT on pre-tained dataset, Second step: self-training on target domain; Third step:Using conditional adversarial learning to minimize cross-domain discrepancy between source and target domains

- Dataset
SQuAD: it contains 86599 training samples and 10570 validation samples, the questions are proposed by people based on sentences in Wikipedia, the answers are in text span forms.
CNN and DailyMail: it includes 374K training data and 4k validation data, 872k training data and 64k valivation samples respectively. Questions are in cloze form, answers are masked entities in sentences.
NewsQA: QA pairs are generated by workers based on storied from CNN, with 76K training and 4K validation samples.

CoQA: it contains 109k training and 8k validation samples. Questions are in conversation forms and answers are either in text spans or yes/no types. **DROP**: It focuses on numerical data so the answers are in numbers or dates except text spans.

- Experiment
  The paper first generalizes BERT on 6 RC datasets and then perform CASe for unsupervised domain adaption on these datasets.

Datasets	SQuAD	CNN	DailyMail	NewsQA	CoQA	DROP
SQuAD	-	16.72 / 26.42	21.12 / 21.70	**40.03 / 57.42**	**29.58 / 39.58**	**19.06 / 29.73**
CNN	18.97 / 24.34	-	**81.53 / 83.59**	9.38 / 15.36	7.10 / 10.26	4.40 / 7.50
DailyMail	9.72 / 14.76	**77.22 / 79.73**	-	5.89 / 10.69	5.68 / 8.75	4.69 / 8.02
NewsQA	64.80 / 78.32	25.10 / 34.66	28.41 / 38.44	-	27.14 / 38.75	12.36 / 21.00
CoQA	**65.25 / 74.92**	18.21 / 24.76	22.65 / 28.12	37.74 / 53.85	-	14.75 / 21.60
DROP	55.53 / 68.36	14.32 / 22.26	17.44 / 25.78	28.36 / 44.35	16.15 / 24.82	-
Self	79.85 / 87.46	82.76 / 84.73	81.37 / / 83.33	52.05 / 67.41	48.98 / 63.99	44.67 / 52.51

Figure 29: Performance of zero-shot models on dev set when transferring among datasets. Rows represents for source dataset, columns is target dataset. Left value is exact match, right score is F1 score.

SQuAD	-	80.20 / 81.93	79.91 / 82.06	**51.56 / 66.79**	**50.77 / 65.94**	**48.45 / 57.33**
CNN	78.59 / 86.39	-	**83.40 / 85.06**	48.95 / 64.45	49.38 / 64.57	44.15 / 51.87
DailyMail	78.07 / 86.22	**82.44 / 84.36**	-	50.91 / 65.90	48.64 / 63.80	41.58 / 47.74
NewsQA	**78.87 / 87.06**	80.49 / 82.43	80.93 / 82.99	80.99 / 83.07	48.01 / 64.30	45.06 / 54.34
CoQA	78.24 / 85.80	76.34 / 78.22	78.12 / 79.88	50.80 / 65.55	-	41.43 / 49.40
DROP	74.81 / 83.67	80.38 / 82.21	80.78 / 82.96	50.01 / 65.16	46.27 / 62.67	-
Self	79.85 / 87.46	82.76 / 84.73	81.37 / / 83.33	52.05 / 67.41	48.98 / 63.99	44.67 / 52.51

Figure 30: Domain adaptation performance of CASe on dev sets of datasets. .Rows represents for source dataset, columns is target dataset. Left value is exact match, right score is F1 score.

The first plot is performance of zero-shot models before domain adaptation. And the second plot is performance after domain adaption, which shows standard CASe results. Rows represents for source datasets and columns shows target datasets. The left value is exact match, the right one is F1 score.

Be comparison, we can see CASe achives significant improvement compared to naive BERT. Zero shot model performs poorly on two different based datasets e.g., SQuAd to CNN, while CASe can elimibate such gaps. Domain adaption performs better than SELF in very alkie datasets such as CNN and DAILYMAIL.

4.1.2 TANDA: Transfer and Adapt Pre-Trained Transformer Models for Answer Sentence Selection

Tags:*Answer sentence selection, Question Answering, general domain, Transfer learning*

- Background
  Answer Sentence Selection(AS2) is a branch in Question Answering, it's task is choose the correct answer of the question when given a question and a corresponding set of answer sentence candidates. It is widely used in building conversational agents such as Google Home, Siri and Alexa. There are many language modeling methods used for QA tasks, e.g., ELMO, GPT, BERT and XLNet. They first pre-train models and fine tune them on target tasks. However, they suffers from high accuracy variance, for example, they may be incline to predict a single label. As a reult, TandA is introduced by Garg [42] to conquer the problem.
- Model
  The definition of AS2 is: given a question q and a set of answer sentence candidates S=$s_1, ..., s_n$, select a sentence s_k that can answer the question q correctly.
  TandA model is based on Transfer Learning and extends Transformer models by applying a two-step fine tuning: Transfer and Adapt(TandA).

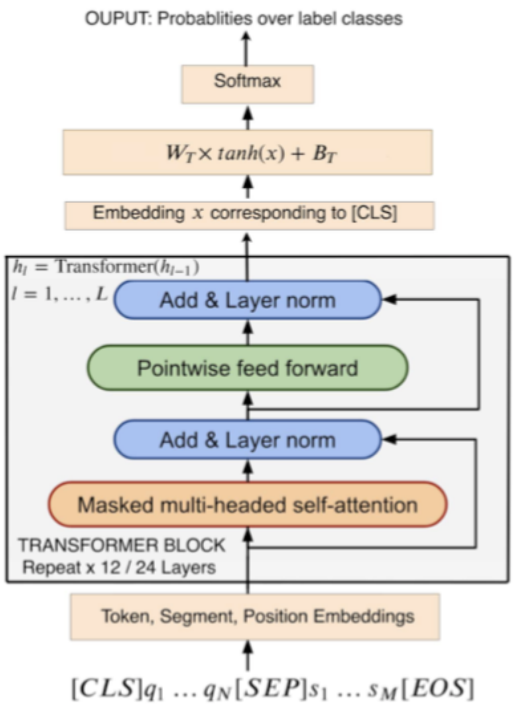

OUPUT: Probablities over label classes
Softmax
$W_T \times tanh(x) + B_T$
Embedding x corresponding to [CLS]
h_l = Transformer(h_{l-1})
$l = 1, \dots, L$
Add & Layer norm
Pointwise feed forward
Add & Layer norm
Masked multi-headed self-attention
TRANSFORMER BLOCK
Repeat x 12 / 24 Layers
Token, Segment, Position Embeddings
$[CLS]q_1 \dots q_N[SEP]s_1 \dots s_M[EOS]$

Figure 31: Transformer architecture for fine tuning on AS2

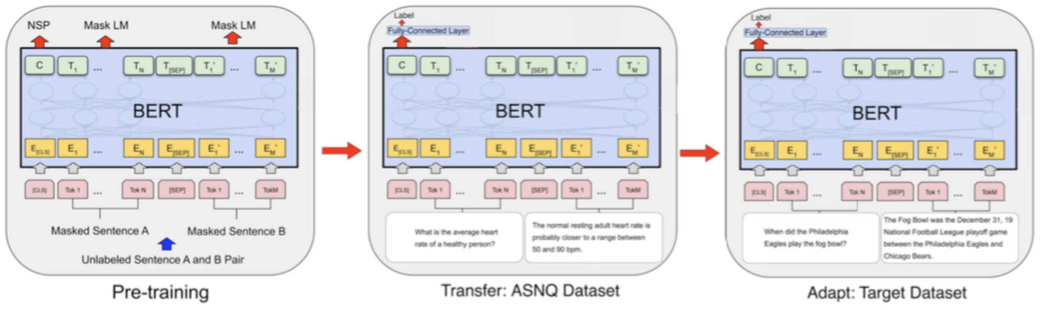

NSP
Mask LM
Mask LM
BERT
Masked Sentence A
Masked Sentence B
Unlabeled Sentence A and B Pair
Pre-training
Label
Fully-Connected Layer
BERT
What is the average heart rate of a healthy person?
The normal resting adult heart rate is probably closer to a range between 50 and 90 bpm.
Transfer: ASNQ Dataset
Label
Fully-Connected Layer
BERT
When did the Philadelphia Eagles play the fog bowl?
The Fog Bowl was the December 31, 19 National Football League playoff game between the Philadelphia Eagles and Chicago Bears.
Adapt: Target Dataset

Figure 32: Transfer and Adapt for Answer Sentence Selection, applied to BERT

In the first fine tune step, Transformer architecture is fine-tuned on AS2 task. See figure31, input is based on concatenation of question and sentense candidates pairs, segments and their positions. Note that sentenses tokens are separated by three tags,[CLS],[SEP] and [EOS], standing for beginning, separator and end of sentence. Then as what we mentioned in Transformer, The input is fed forward to several blocks, and the result is an embedding, representing the dependencies between words and segments of question and answer candidate pairs. For downstream tasks, output is processed to a fully connected layer with weights $W_T and B_T$. The probability of a text pair can be calculated after a softmax layer followed FC layer.
The left block in Figure32 illustrates pre-training step on Transformer model. Note that input are randomly masked so that the model can generalize well. Then in the middle block, the model is fine tuned on target task(AS2). In this case Answer Sentence Natural Questions (ASNQ) dataset, which is a corpus built by the author of the paper that contains annotated short answers for Natural Questions dataset to fits AS2 tasks better. Normal transfer learning models will stop at this stage, however, it requires (1) a large dataset for

target domain, which is difficult and costly to get. (2) merging source and target domain in a single move, while the model only need to be specific in the target data. As a result, TandA applys a second fine-tuning step to adapt the model to the target domain, which is shown at right block in Figure32.

- Dataset
WikiQA: WikiQA is an AS2 dataset, the answer sentences of which are extracted from Wikipedia and then manually labeled. As shown in table below, raw data means sum of all data, all- represents for those questions who don't have correct answer. all+ means question only have correct sentence, but no candidate sentences. clean data means those data without both all- and all+. In experiment, no all- data is used for training and clean data is used in validation.

Mode	Train		Dev		Test	
	Q	A	Q	A	Q	A
raw	2118	20360	296	2733	633	6165
no all-	873	8672	126	1130	243	2351
clean	857	8651	121	1126	237	2341

Figure 33: WikiQA dataset statistics

- Experiment
Table below shows MAP and MRR of various models for two methods: vanilla fine tuning and TandA. As we can see from the table, TandA performs a huge improvement compared to BERT. RoBERTa-Large with ASNQ->WikiQA dataset builds surprising state of art result with 0.920 and 0.933 in MAP and MRR respectively.

Model	**MAP**	**MRR**
Comp-Agg + LM + LC	0.764	0.784
Comp-Agg + LM + LC+ TL(QNLI)	0.834	0.848
BERT-B FT WikiQA	0.813	0.828
BERT-B FT ASNQ	0.884	0.898
BERT-B TANDA (ASNQ $\rightarrow$ WikiQA)	0.893	0.903
BERT-L FT WikiQA	0.836	0.853
BERT-L FT ASNQ	0.892	0.904
BERT-L TANDA (ASNQ $\rightarrow$ WikiQA)	0.904	0.912
RoBERTa-B FT ASNQ	0.882	0.894
RoBERTa-B TANDA (ASNQ $\rightarrow$ WikiQA)	0.889	0.901
RoBERTa-L FT ASNQ	0.910	0.919
RoBERTa-L TANDA (ASNQ $\rightarrow$ WikiQA)	**0.920**	**0.933**

Figure 34: Performance of different models on WikiQA dataset. Here Comp-Agg + LM + LC refers to a Compare- Aggregate model with Language Modeling and Latent Clustering. TL(QNLI) refers to Transfer Learning from the QNLI corpus. L and B stand for Large and Base, respectively.

4.1.3 Deep Transfer Learning for Thermal Dynamics Modeling in Smart Buildings

Tags:*Domain Adaptation, Thermal dynamics modeling, LSTM, real-estate domain, Transfer Learning*

- Background
A good living environment can be beneficial to people mentally and physically. One of the criterion is the ability to maintain thermal comfort in house. To achieve that, it requires adaptive thermal dynamic models to capture even slightly thermal changes in room and adjust for the disturbances.
Efforts have been taken to capture thermal dynamics, such as artificial neural network, support vector machine and deep neural network. However, those data-driven models can not transfer existing models to more buildings. In other word, they can not make use of the knowledge in existing models but derive models from scratch on different data every time which is very time-consuming. Besides, models can not be constructed well if there is a limited amount of data.

As a result, Jiang and Lee[43]build a method called deep supervised domain adaptation (DSDA) which applys Transfer Learning to fulfill domain adaptation that transfers prior knowledge from one building to another.

- Model
  A Long short term memory(LSTM) network based sequence to sequence scheme is pre-trained using time sequence large data collected from source buildings and then fine-tuned the model by adapting target building, which shares different but related tasks with source buildings to the model. The goal of the approach is to learn prediction model which performs well on target building dataset.

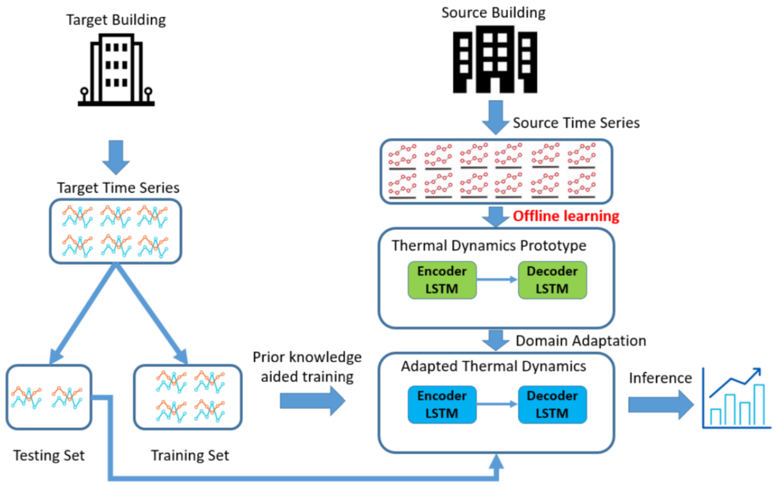


Figure 35: Proposed DSDA: LSTM S2S model pre-trained and adapted between source and target buildings

LSTM S2S architecture is chosen for pre-training. As shown in Figure36, LSTM maps input x_i^S to output $\hat{y}_i^S$. Every input is encoded as a vector, and final encoding of encoder is denoted by h_e^K. h_e^K is used as an initial state for decoder to cooperate with y_0 to activate decoder. Note that every time step there is a linear activation which is used to predict decoded state y_i^S. Every timestep the decoded state from previous step is fed to the current update. The parameter of the model is learned by minimizing mean square error loss L: $L(y_i^l, \hat{y}_i^l) = \frac{a}{n*L} = \sum_{i=1}^{L} |y_i^l - \hat{y}_i^l|^2$.After pre-trining, a limited amount of data from target building are applied to fine-tune the model to specific target tasks.

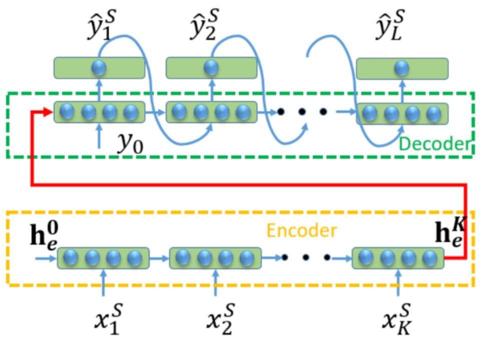
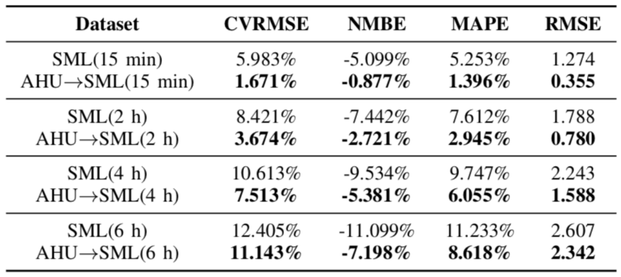


Figure 36: Structure of LSTM S2S

- Dataset
  Two datasets are considered: Datasets for building indoor temperature evolution: SML,

AHU. SML has 1373 building data, which is used for target building. The ratio of training and testing is 0.67. while AHU is used for source building, with size of 35098.

- Experiment
  For temperature evolution, Table below shows with the pre-traning of AHU data, the predictive performance is improved on SML data, regardless of the prediction time. Small time prediction performs better than long time prediction.



Dataset	CVRMSE	NMBE	MAPE	RMSE
SML(15 min)	5.983%	-5.099%	5.253%	1.274
AHU→SML(15 min)	**1.671%**	**-0.877%**	**1.396%**	**0.355**
SML(2 h)	8.421%	-7.442%	7.612%	1.788
AHU→SML(2 h)	**3.674%**	**-2.721%**	**2.945%**	**0.780**
SML(4 h)	10.613%	-9.534%	9.747%	2.243
AHU→SML(4 h)	**7.513%**	**-5.381%**	**6.055%**	**1.588**
SML(6 h)	12.405%	-11.099%	11.233%	2.607
AHU→SML(6 h)	**11.143%**	**-7.198%**	**8.618%**	**2.342**

Figure 37: Comparison table between learning from scratch and DSDA. CVRMSE means root mean square error, NMBE is normalized mean bias error, MAPE represents for mean absolute percentage error and root mean square error is RMSE.

4.1.4 NegBERT: A Transfer Learning Approach for Negation Detection and Scope Resolution

Tags:*Negation Detection, sentiment analysis, general domain, BERT, Transfer Learning*

- Background
  'Negation detection and Scope Resolution' is an important branch in sentiment analysis, since a negation reverses the entire meaning of text in its scope. The paper[44] focus on biomedical data where negation is used frequently. At first, people think it's feasible to solve negation and scope detection task by carefully designing rules. However, due to the complexities of human language, this method fails. After that, people use Deep Learning based methods, which is very promising. But due to the variety of ways of scope of a cue(negation word) and small dataset sized, it is particularly challenging to use DL to solve the problem. For example, the form of cues can be various:

  1. Affix: (im)possible, (a)typical, (non)uniform
  2. A single word: not, no, lack, fail
  3. A set of consecutive words or discontinuous words e.g,.neither...nor

- Model
  The paper[44] applys Transfer Learning to Nagation Detection problem. The approach is consist of two stages: Negation detection and then Scope resolution. The structure of each stage is a BERT-base connects with a classification layer.
  For negation detection, the following schema is used:

  1. Affix
  2. Normal Cue
  3. Part of a multi-word cue
  4. Not a Cue
  5. Padded tokens

  The first three schema corresponds to the cue types mentioned above, the fourth schema means the token is a normal word, and last schema is used to set the weight of that token to 0 to avoid training(padding strategy in BERT). There are two cue encode methods:

  1. Replace: Replace cue token with a special token which stands for the token type. For example, [im]polite is encoded as token[0] and not is encoded as token[1] neither and nor are encoded as token[2]

2. Augment: Add special token based on original word. For example: [im]polite becomes token[0]impolite. not turns to token[1]not, neither and nor becomes token[2]neither, token[2]nor.

For scope resolution, a binary labelling scheme is used, in which 0 represents for not a token and 1 is as a token.
The process of the system is shown in fig below. Note that the model needs to convert token level words to word-level. Here it uses softmax on each token for probability distribution over whole vocabulary.

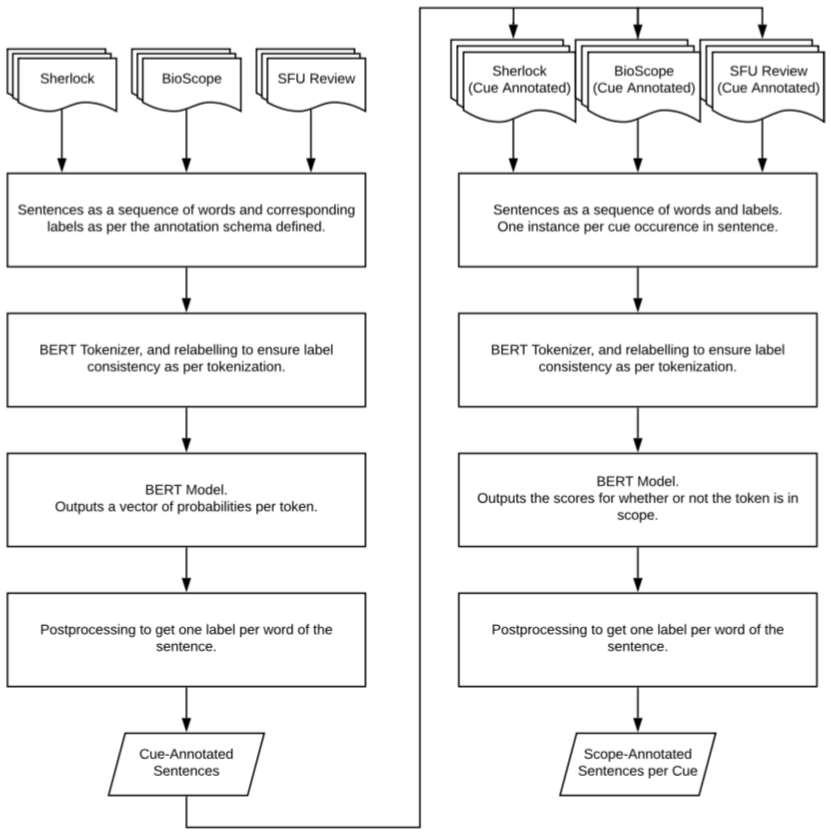
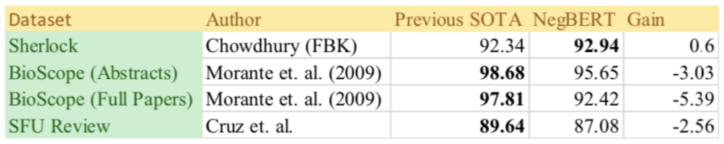

Sherlock
BioScope
SFU Review
Sentences as a sequence of words and corresponding labels as per the annotation schema defined.
BERT Tokenizer, and relabelling to ensure label consistency as per tokenization.
BERT Model. Outputs a vector of probabilities per token.
Postprocessing to get one label per word of the sentence.
Cue-Annotated Sentences
Sherlock (Cue Annotated)
BioScope (Cue Annotated)
SFU Review (Cue Annotated)
Sentences as a sequence of words and labels. One instance per cue occurence in sentence.
BERT Tokenizer, and relabelling to ensure label consistency as per tokenization.
BERT Model. Outputs the scores for whether or not the token is in scope.
Postprocessing to get one label per word of the sentence.
Scope-Annotated Sentences per Cue

Figure 38: Proposed System(NegBERT)

- dataset
  SFU Review Corpus Dataset, BioScope Corpus and Sherlock Corpus are used for training and validation. Both SFU and Bioscope corpus suffer from a problem that negation scopes are annotated as a single span of text delimited by punctuation, as a result, Sherlock Corpus is introduced. The model is trained on one dataset and test on all datasets.

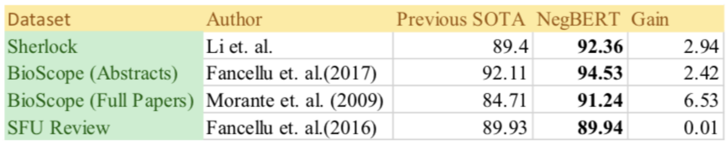
- Experiment
  Table is a summary on negation cue detection result, we can see there is a gap between previous SOTA model and NegBERT(paper's model), though NegBERT outperforms baseline system. The authors of paper think it is because of the small capacity of dataset. BERT need more examples to train for negation detection.



Dataset	Author	Previous SOTA	NegBERT	Gain
Sherlock	Chowdhury (FBK)	92.34	**92.94**	0.6
BioScope (Abstracts)	Morante et. al. (2009)	**98.68**	95.65	-3.03
BioScope (Full Papers)	Morante et. al. (2009)	**97.81**	92.42	-5.39
SFU Review	Cruz et. al.	**89.64**	87.08	-2.56

Figure 39: Negation Cue Detection Results (F1 score)

As for Scope resolution, NegBERT performs better than all previous SOTA models, since it encode the contextual information and transfer among millions of documents in downstream files.



Dataset	Author	Previous SOTA	NegBERT	Gain
Sherlock	Li et. al.	89.4	**92.36**	2.94
BioScope (Abstracts)	Fancellu et. al.(2017)	92.11	**94.53**	2.42
BioScope (Full Papers)	Morante et. al. (2009)	84.71	**91.24**	6.53
SFU Review	Fancellu et. al.(2016)	89.93	**89.94**	0.01

Figure 40: Scope Resolution Result (F1 score)

4.1.5 Improving Sentiment Analysis with Multi-task Learning of Negation

Tags:*Negation Detection, sentiment analysis, general domain, BiLSTM, Multi-task Learning*

- Background
  The use case of this paper is similar to the last one, both of them can solve negation detection task. The difference is this paper uses Multi-task learning while the last one uses Transfer Learning. Sentiment analysis is a popular task in NLP, which assigns polarity to text. The main task of the paper[45] is to solve sentiment analysis problem, with negation detection as an auxiliary task to see whether the incorporation can improve the performance of sentiment analysis. We can see relationship between negative detection and sentiment analysis in an example. If there is a sentebce:

  It's **not** so much a work of entertainment as it is a unique well-crafted psychological study of grief.

  If there is no negation detection and scope resolution, it is very likely that naive sentiment analysis mistakenly classify the sentence as a negative one.

- Model
  The structure of Model is shown in figure41. Sentiment and negation tasks share parameters from lower layers, where BiLSTM is used to extract features from embedding layers.
  For negation detection, the model uses a linear chain conditional random field(CRF) with Viterbi decoding to find labels with most probability. Note that BiLSTM predict cues and scopes in a single pass.

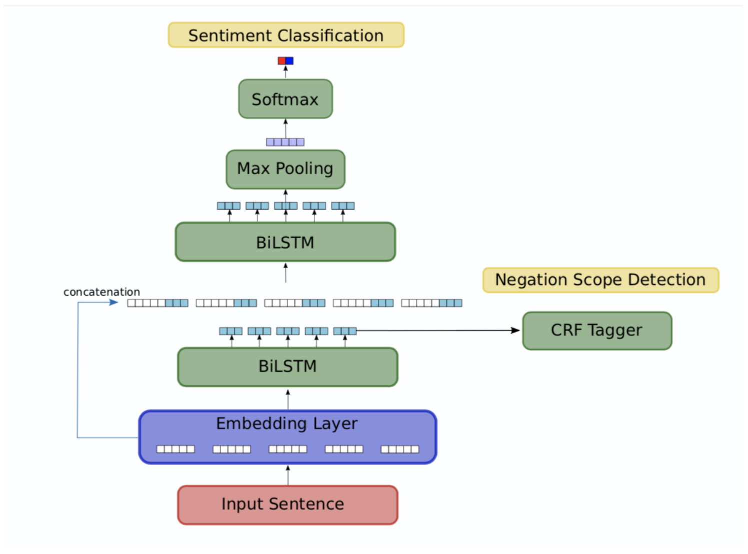


Figure 41: proposed multi-task model.

For sentiment analysis, the model uses skip-connection to concate each embedding to contextualized representation. Then the sequence is fed to a second sentiment-specific BiLSTM layer. After that BiLSTM passes through max-pooling and finally to a softmax layer to calculate the class probabilities. And cross entropy loss is used to train the classification.

Model	SST-binary	SST-fine	SemEval-binary	SemEval-fine
BOW	80.7	40.3	–	65.5
CNN	81.3 (1.1)	39.8 (0.7)	–	63.5 (1.3)
BiLSTM	82.6 (0.7)	45.6 (0.7)	–	65.1 (0.9)
SAN + RPR	84.2 (0.4)	48.1 (0.4)	–	72.2 (0.8)
Tree-LSTM	88.0 (0.3)	51.0 (0.5)	–	–
BERT	<u>94.9</u>	<u>53.0</u>	–	<u>75.1</u>
HEUR	83.78 (0.3)	45.29 (1.6)	82.59 (1.4)	65.59 (1.3)
STL	84.57 (1.0)	46.49 (0.7)	84.00 (1.0)	67.26 (0.6)
MTL-SFU	**86.04 (0.3)***	**46.75 (0.8)**	**84.02 (0.9)**	67.03 (1.0)
MTL-CD	**85.43 (0.9)***	**47.33 (0.6)***	83.52 (1.6)	**67.75 (1.0)***

Figure 42: Mean accuracy and standard deviation of several models including STL and MTL. <u>underlined</u> results means the best overall method, **bold** results means where MTL outperforms STL, A star* means the model performs extreamly better. The score of SemEval binary for several models are blank since the previous work doen't have the results on this data.

- Dataset
  For main task(sentiment classification) the paper uses sentence and tweet level classification such as Stanford sentiment Treebank(SST) and SemEval2013, while learning negation cues and scopes based on two different negation datasets such as SFU Review Corpus and Conan Doyle Neg.
  SFU Review Corpus: It contains 400 reviews fro, 8 domains: books, cars, hotels, computers, music, cookware and phones, which annotates negation and speculation at sentence level.
  ConanDoyle-neg: It contains Conan Doyle stories which are annotated negation cues, scopes and events manually. The training set includes 3640 sentences, with 848 negation sentences. The development set includes 787 sentences, with 144 negated sentences.
  Stanford Sentiment Treebank(SST): SST data used in pre-defined train, validation and test steps, with 11855 sentences retrieved from English movie reviews.
  SemEval2013: SemEval is based on tweet-level sentiment analysis including 9287 tweets with three different sentiments: positive, negative and neural. It is also used in pre-defined train, validation and test steps.

- Experiment
  Tabel shows the mean accuracy and standard deviation of several models: Single task sentiment(STL), multi-task models with SFU auxiliary negation data(MTL-SFU) and multi-task models with ConanDoyle-neg auxiliary negation data(MTL-CD) over 4 datasets.

  We can see that MTL performs better than STL models on most tasks. MTL-SFU does best on binary tasks, while MTL-CD is good at fine-grained tasks. Note that Tree-LSTM and BERT perform better than the proposed models, but Bi-LSTM is easier and faster to train. Next time a model with more complex basic structure should be build to enhance the performance. But overall, MTL models improves significantly on the STL models.

4.1.6 Multi-Task Deep Neural Networks for Natural Language Understanding

Tags:*Natural language understanding, general domain, Transformer, Multi-task Learning*

- Background
  Natural language understanding(NLU) tasks is fundamental in natural language processing. It can be used in many applications such as speech recognition, machine translation, chatbots, etc. As for NLU task, two popular methods are multi-task learning and pre-training. This paper[16] tries to combine those two approaches and achieve better performance on NLU tasks.

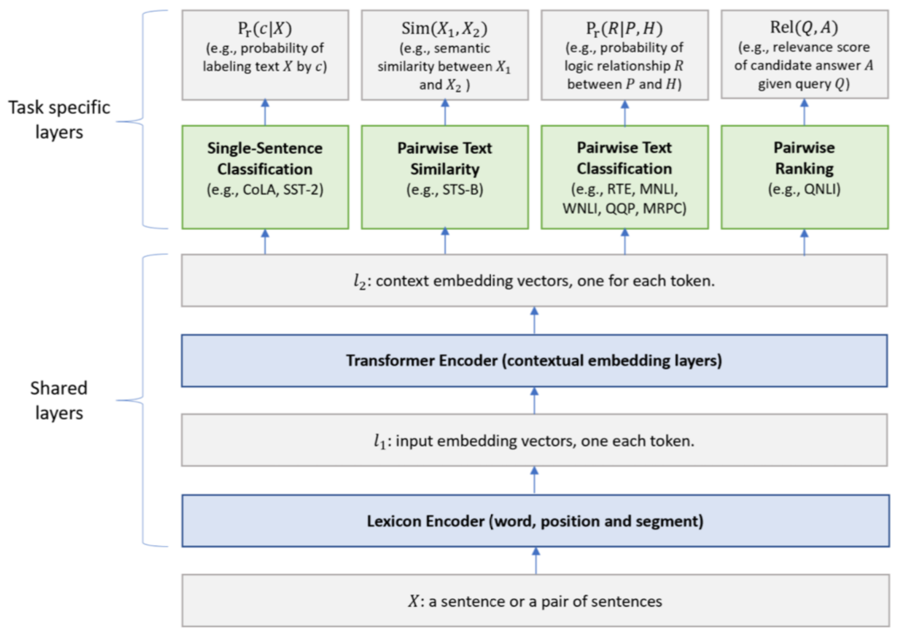
- Model
  The model presented in the paper is called Multi-Task Deep Neural Network model(MT-DNN), which is an extension based on BERT. As shown in figure43, the lower layers are shared among all four tasks. The upper layers are task-specific. In this case it has four tasks:

  1. Single-Sentence Classification: The model labels each given sentence using one of the predefined labels.
  2. Text Similarity: Given two sentences, the model predicts their semantic similarities.
  3. Pairwise Text Classification: The model labels the relationship between each sentence pairs with pre-defined labels.
  4. Relevance Ranking: Given a query and a list of candidate answers, the model rank the answers based on the relativity to the query.

  The input of the model is word embedding vectors X, then the Lexicon Encoder maps X into a sum of X's corresponding word, segment and positional embeddings, after which we got l_1. If the input X is a sentence pair(X_1, X_2), a special token [SEP] needs to be insert inbetween. In Transformer Encoder, the model captures the contextual message by self-attention mechanism for each word, l_2 vector is computed at this stage. After that, there comes to task-specific part. For each task, several task-specific parameters are defined to solve the corresponding problem. For example, inSingle-Sentence Classification Output, the probability of X having a label c can be computed as:

$$P_r(c|X) = softmax(W_{SST}^T * x)$$

  where W_{SST} is the task-specific parameter for Single Sentence Classification. To train the model, we nened two steps: pre-training and multi-task learning. For pre-training part, it is similar to BERT, the parameters used in Lexicon Encoder and Transformer Encoder are transfered from two unsupervised prediction tasks: mask language modeling and next sentence prediction. As for multi-task learning, it uses SGD to learn parameters for each tasks.



$P_r(c|X)$ (e.g., probability of labeling text X by c)
$\mathrm{Sim}(X_1, X_2)$ (e.g., semantic similarity between X_1 and X_2)
$P_r(R|P,H)$ (e.g., probability of logic relationship R between P and H)
$\mathrm{Rel}(Q,A)$ (e.g., relevance score of candidate answer A given query Q)
Task specific layers
Single-Sentence Classification (e.g., CoLA, SST-2)
Pairwise Text Similarity (e.g., STS-B)
Pairwise Text Classification (e.g., RTE, MNLI, WNLI, QQP, MRPC)
Pairwise Ranking (e.g., QNLI)
l_2: context embedding vectors, one for each token.
Transformer Encoder (contextual embedding layers)
Shared layers
l_1: input embedding vectors, one each token.
Lexicon Encoder (word, position and segment)
X: a sentence or a pair of sentences

Figure 43: Architecture of MT-DNN model for representation learning

- Dataset
  As shown in Table below, there are 3 main datasets: GLUE, SNLI and SciTail used for training and develping.
  GLUE: General Language Understanding Evaluation is a collection of 9 sub-NLP-tasks covered in Table below. It is widely used in many NLP models to enhance generalization and robustness.
  SNLI: Stanford Natural Language Inference dataset contains 570k manually labeled sentence pairs. It is only used for domain adaption here. **SciTail** A textual entailment dataset generated from science question answeing dataset(SciQ), which aims to assess if a given precondition entails a certain hypothesis. Since it is derived from scientific dataset, the sentences are challenging and lexical similarity of precondition and hypothesis is high, which makes SciTail difficult.It is only used for domain adaption here.

Corpus	Task	#Train	#Dev	#Test	#Label	Metrics
		Single-Sentence Classification (GLUE)				
CoLA	Acceptability	8.5k	1k	1k	2	Matthews corr
SST-2	Sentiment	67k	872	1.8k	2	Accuracy
		Pairwise Text Classification (GLUE)				
MNLI	NLI	393k	20k	20k	3	Accuracy
RTE	NLI	2.5k	276	3k	2	Accuracy
WNLI	NLI	634	71	146	2	Accuracy
QQP	Paraphrase	364k	40k	391k	2	Accuracy/F1
MRPC	Paraphrase	3.7k	408	1.7k	2	Accuracy/F1
		Text Similarity (GLUE)				
STS-B	Similarity	7k	1.5k	1.4k	1	Pearson/Spearman corr
		Relevance Ranking (GLUE)				
QNLI	QA/NLI	108k	5.7k	5.7k	2	Accuracy
		Pairwise Text Classification				
SNLI	NLI	549k	9.8k	9.8k	3	Accuracy
SciTail	NLI	23.5k	1.3k	2.1k	2	Accuracy

Figure 44: Statistics of three benchmarks: GLUE,SNLI and SciTail

- Experiment

Model	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	WNLI 634	AX	**Score**
BiLSTM+ELMo+Attn [1]	36.0	90.4	84.9/77.9	75.1/73.3	64.8/84.7	76.4/76.1	-	56.8	65.1	26.5	70.5
Singletask Pretrain Transformer [2]	45.4	91.3	82.3/75.7	82.0/80.0	70.3/88.5	82.1/81.4	-	56.0	53.4	29.8	72.8
GPT on STILTs [3]	47.2	93.1	87.7/83.7	85.3/84.8	70.1/88.1	80.8/80.6	-	69.1	65.1	29.4	76.9
$\text{BERT}^{4}_{\text{LARGE}}$	60.5	94.9	89.3/85.4	87.6/86.5	72.1/89.3	86.7/85.9	92.7	70.1	65.1	39.6	80.5
$\text{MT-DNN}_{\text{no-fine-tune}}$	58.9	94.6	**90.1/86.4**	89.5/88.8	**72.7/89.6**	86.5/85.8	**93.1**	79.1	65.1	39.4	81.7
MT-DNN	**62.5**	**95.6**	**91.1/88.2**	**89.5/88.8**	**72.7/89.6**	**86.7/86.0**	**93.1**	**81.4**	65.1	**40.3**	**82.7**
Human Performance	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0/92.8	91.2	93.6	95.9	-	87.1

Figure 45: GLUE test set results. State of art results are presented in **bold**, the results better than human performance are in blue bold. Note that <T-DNN uses $BERT_{Large}$ to initialize its shared layers.

MT-DNN model is initialized shared layers with pre-trained $BERT_{LARGE}$, then uses multi-task learning on GLUE dataset, finally fine-tuned on specific task. We can see from the above table that MT-DNN performs better than all models, except WNLI. Note that it achieves 2.2% absolution improvement over $BERT_{LARGE}$, inplies the main reason for the improvement is MTL since other elements are the same between those two models.
$MT-DNN_{no-fine-tune}$, just as its name, lacking of fine tune step compared to MD-DNN. We can see it still performs better than $BERT_{LARGE}$ among all tasks except CoLA. The reason is CoLA is a dataset with much smaller in-domain data than others, which means it is difficult to benefit from knowledge of other tasks. As a result, fine-tuning is necessary for a good performance.

4.1.7 MULTI-TASK SEQUENCE TO SEQUENCE LEARNING

Tags:*Language Translation, Sequence to Sequence, LSTM, general domain, Multi-task Learning*

- Background
  Language translation is an important task in many domains such as tourism and education industry. Multi-task learning can improve the generalization of target task by learning several ralated tasks simultaneously, sequence to sequence learning aims at map variable-length input sequences to variable-length output sequences. Both of them are good at language translation problem, but there are few works to combine them together. This paper[46] proposes *one-to-many* approach to solve translation and parsing using one encoder.
- Model

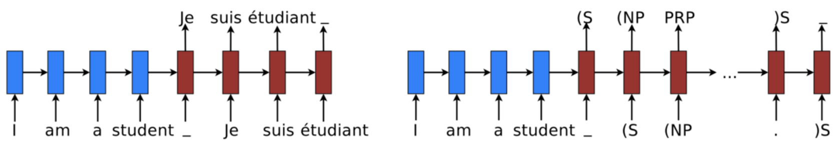

Je suis étudiant _
I am a student _ Je suis étudiant
(S (NP PRP)S _
I am a student _ (S (NP .)S
...

Figure 46: Sequence to sequence learning examples – (left) machine translation and (right) constituent parsing .

Figure46 illustrates sequence to sequence translation and pasring, which uses an encoder-decoder framework. As we can see, a representation s is generated by encoder, and decoder generates output sequence based on the representation. a bi-direction LSTM architecture is usually used by S2S task. Note that an attention mechanism is applied to handle long input sequences, decoder uses attention to decide which tokens to focus.
One-to many approach involves one encoder and multiple decoders, this is for those tasks like shown in Figure47 , which are share the same encoder. For example, the input sentence is English words, and there are three tasks: German word translation, tags parsing for constituency parsing and next sentence prediction. Note that to avoid cheating in unsupervised learning prediction task cased by sequence to sequence modeling where encoder has included all messages and passes that to decoder, the unsupervised task only encode half sentence and predict the other half, which is called autoencoders.

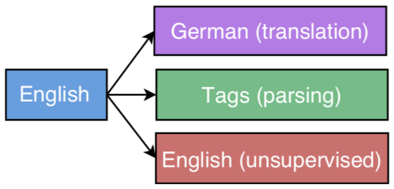


Figure 47: One to many approach setting.

- Dataset
  The paper uses WMT15 data for English and German translation training, and newstest2014(2737 sentences) for English to German translation validation, and newstest2015(2169 sentences) for German to English translation validation. For parsing, Penn Tree Bank(PTB) dataset are used for training and validation devided by a certain ratio.

- Experiment
  Table shows an improvement can be seen after adding a very small number of parsing mini-batches(0.01x means one parsing mini-batch every 100 translation mini-batches). More specificlly, even though single task baseline performs well, the best multi-task model still achieve a gain of +1.5 BLEU points over single model. What's better, parsing task can also benefit from translation task by gaining a +8.9 F_1 points over the baseline.

Task	**Translation**			**Parsing**
	Valid ppl	Test ppl	Test BLEU	Test F_1
(Luong et al., 2015a)	-	8.1	14.0	-
Our single-task systems				
Translation	8.8 (0.3)	8.3 (0.2)	14.3 (0.3)	-
PTB Parsing	-	-	-	43.3 (1.7)
Our multi-task systems				
Translation + PTB Parsing (1x)	8.5 (0.0)	8.2 (0.0)	14.7 (0.1)	54.5 (0.4)
Translation + PTB Parsing (0.1x)	8.3 (0.1)	7.9 (0.0)	15.1 (0.0)	**55.2** **(0.0)**
Translation + PTB Parsing (0.01x)	**8.2** (0.2)	**7.7** (0.2)	**15.8** (0.4)	39.8 (2.7)

Figure 48: English to German WMT'15 translation and Penn Tree Bank parsing. Results measurements: perplexities (ppl), BLEU scores, and parsing F1 for various systems. Best results are highlighted in boldface.

4.1.8 Exploring the Limites of Transfer Learning with a Unified Text-to-Text Transformer

Tags:*Transfer Learning Exploration*

- Background
  With the explosion of unlabeled text data on Internet such as Common Crawl, which offers ca.20TB text data extracted from Internet per month, leads to more attention on pre-training in Transfer Learning, since pre-training utilizes unlabeled data for unsupervised learning. There are various techniques in this field which makes it difficult to figure out which are the optimal algorithms, discriminate the affection of contributions and understand the space of existing methods. This paper [47] aims to achieve a more thorough understanding of those techniques by presenting a unified and systematically approach to transfer learning.
  The paper focus on 'text-to-text' Natural Language Processing problem, as a result, the paper can apply the same model, training procedure, training process to each included task. The paper covers many English-based NLP problems, such as question answering, document summarization, sentiment classification and so on. Those tasks are evaluated by the unified models by comparing their effectiveness, datasets and other factors.
- Optimal model parameters experiment

  1. Baseline Model

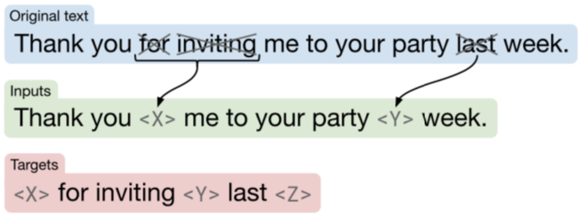

Original text
Thank you ~~for inviting~~ me to your party ~~last~~ week.
Inputs
Thank you <X> me to your party <Y> week.
Targets
<X> for inviting <Y> last <Z>

Figure 49: Illustration on the masked method used in the baseline model. For this specific case, in the sentence “Thank you for inviting me to your party last week.” words “for”, “inviting” and “last” are randomly chosen to reconstruction. Each masked sequence are replaced by one special token (<X> and <Y>) which is unique over the example. The target sequence consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final token <Z>.

A standard encoder-decoder Transformer is set as a baseline model, given the good performance it has on both classification and prediction tasks. Both encoder and decoder are consisted of 12 blocks. In block, there are self-attention, optional encoder-decoder attention and a feed forward network. The output of feed-forward network has dimensionality of 3072, followed by a ReLU activation function. The dimensionality of ’key’ and ’value’ in attention mechanism is 64. In total, there are about 220 million parameters in the model.
During experiment, people found that ’denoising’ objectives helps the model learn better. The authors randomly samples and drop pit 15% tokens of inputs. For consecutive drop-out spans, they are replaced by a single token. An example is shown in Figure below.
To make comparison, models without pre-training is also introduced. Accroding to the table blow, we can see models with pre-training gains a huge improvement. The only exception is on WMT English to French task. The reason is that the dataset has already large enough, so there’s no need to learn from pre-training data.

	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Baseline average	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Baseline standard deviation	0.235	0.065	0.343	0.416	0.112	0.090	0.108
No pre-training	66.22	17.60	50.31	53.04	25.86	**39.77**	24.04

Figure 50: Average and standard deviation of scores.

2. Comparing different model structures
This part makes comparison on 3 different models: Encoder-decoder; Language model and Prefix LM. As for language model, since it can’t foresee the tokens after the current token, only Transformer decoder is used.

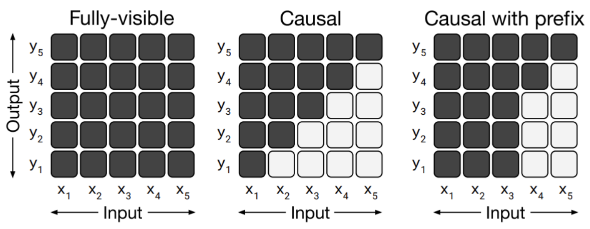
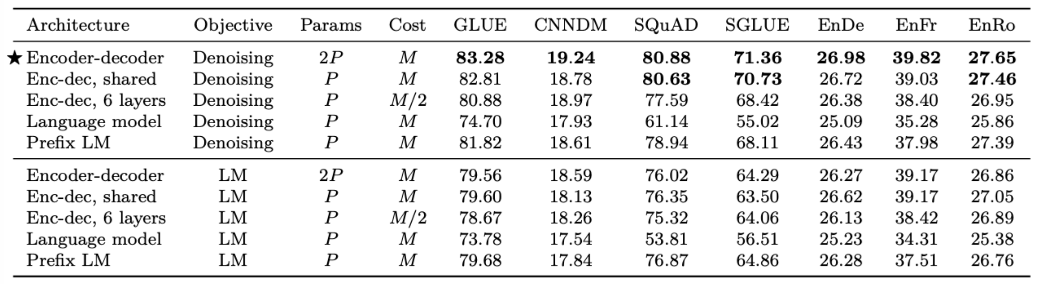


Figure 51: Different attention mask patterns. The input of the self-attention mechanism is x and output is y. A dark cell at row i and column j indicates that the self-attention mechanism can be applied to input element j at output timestamp i. A light cell indicates that the self-attention mechanism is not allowed.

Figure above is an explanation on Prefix LM. It can help model to see the whole portion of input sequence. For example, when there is a continuous sentence containing a premise, a hypothesis and a target, LM with prefix can be encoded like this: “mnli premise: I hate pigeons. hypothesis: My feelings towards pigeons are filled with animosity. target: entailment”. By doing this, the model will have a full vision about the input, and predict the classification of the task easier given the prefix("mnli" in this case).

Here we set the number of layers and parameters in a baseline model as L and P. M is set to be the number of FLOPs needed in a L-L-layer encoder-decoder model. Multiple configuration is set for encoder-decoder model as shown in Table below:



Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

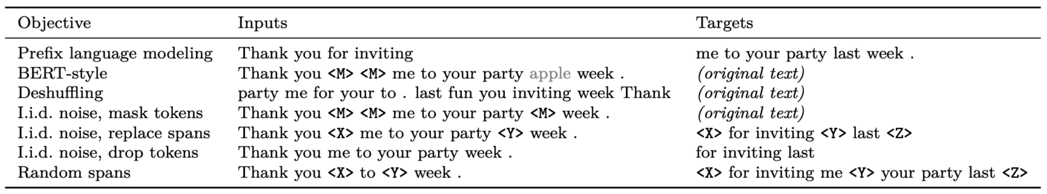
Figure 52: P is the number of parameters in a 12-layer base Transformer layer stack and M to refer to the FLOPs required to process a sequence using the encoder-decoder model.

In denoising objective, the model’s task is to predict missing or corrupted tokens in the input. In LM objective, the model is trained to predict consecutive tokens.
We can see from the table that encoder-decoder model with denoising objective performs better than any other models. Sharing parameters between encoder and decoder performs similarly with the best model. However, models with half parameters and with decoder only hurts the performance. Overall, training models with denoising objectives always outperforms training with LM.

3. Unsupervised objectives

   This section aims to explore the space of unsupervised objectives.

   – Different high-level approaches.
     To begin with, three different models are compared: First is "prefix language modeling", the input of which can be devided into 2 components, one used as encoder inputs one as target sequence to be predicted by decoder. Second is "masked language modeling" inspired from BERT. Third is deshuffling objective. The examples of input and targets of each model is presented as below:



Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style	Thank you `<M>` `<M>` me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
I.i.d. noise, mask tokens	Thank you `<M>` `<M>` me to your party `<M>` week .	*(original text)*
I.i.d. noise, replace spans	Thank you `<X>` me to your party `<Y>` week .	`<X>` for inviting `<Y>` last `<Z>`
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you `<X>` to `<Y>` week .	`<X>` for inviting me `<Y>` your party last `<Z>`

Figure 53: Instances of inputs and targets sequences offered by different unsupervised methods given input:“Thank you for inviting me to your party last week .” Here,all words were mapped to a single token. The task is to reconstruct the entire sequence given input. <M> denotes a shared mask token and <X>, <Y>, and <Z> denote sentinel tokens that are assigned unique token IDs.

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Prefix language modeling	80.69	18.94	77.99	65.27	**26.86**	39.73	**27.49**
BERT-style [Devlin et al., 2018]	**82.96**	**19.17**	**80.65**	**69.85**	**26.78**	**40.03**	**27.41**
Deshuffling	73.17	18.59	67.61	58.47	26.11	39.30	25.62

Figure 54: Performance of the three disparate pre-training objectives

The performance of those models are presented in Table below, we can see that BERT-style (with MASK) objective outperforms others, although Prefix language modeling performs as well on EnDe. Deshuffling performs the worst.

– Corruption strategies
  Since BERT-style performs best among all models, this section focus on BERT-style objective exploration. First objective we consider is BERT-style, which don't include random token swapping step but simply replace 15% of tokens in input with a mask token. A similar objective is replace the 15% tokens with "MASS". Third and forth models are corruption spans replacement and drop corrupted tokens. The goal of these two models is to prevent from long sequence self-attention deduction. The performance are listed in table below:

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
BERT-style [Devlin et al., 2018]	82.96	19.17	**80.65**	69.85	26.78	**40.03**	27.41
MASS-style [Song et al., 2019]	82.32	19.16	80.10	69.28	26.79	**39.89**	27.55
★ Replace corrupted spans	83.28	**19.24**	**80.88**	**71.36**	**26.98**	39.82	**27.65**
Drop corrupted tokens	**84.44**	**19.31**	**80.52**	68.67	**27.07**	39.76	**27.82**

Figure 55: Comparison of variants of the BERT-style pre-training objective. In the first two variants, the model is trained to reconstruct the original uncorrupted text segment. In the latter two, the model only predicts the sequence of corrupted token

– Corruption rate
  By far, we've been using corruption rate as 15%, the authors try to find whether corruption rate is an important influence on the result. They compare corruption rate of 10%,15%,25% and 50%. We can tell from figure below that corruption rate does affect the performance. Larger corruption rate means longer trargets, which could potentially slow down training speed. Setting 15% as corruption rate seems to be the most wise choice.

Corruption rate	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
10%	**82.82**	19.00	**80.38**	69.55	**26.87**	39.28	**27.44**
★ 15%	**83.28**	19.24	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
25%	**83.00**	**19.54**	**80.96**	70.48	**27.04**	**39.83**	**27.47**
50%	81.27	19.32	79.80	70.33	**27.01**	**39.90**	**27.49**

Figure 56: Performance of the i.i.d. corruption objective with different corruption rates.

– Corrupted span length
  The paper raises a question that whether the corruption spans will perform better than i.i.d. manner, as a result, they test the theory on 4 different settings other than i.i.d model: with average span length as 2,3,5 and 10. We can see from the result that using average span of 3 performs better than i.i.d objective on most cases. We could say corruption span does influence the result.

Span length	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Baseline (i.i.d.)	**83.28**	19.24	80.88	71.36	**26.98**	**39.82**	**27.65**
2	**83.54**	19.39	**82.09**	**72.20**	**26.76**	**39.99**	**27.63**
3	**83.49**	**19.62**	**81.84**	**72.53**	**26.86**	39.65	**27.62**
5	**83.40**	19.24	**82.05**	**72.23**	**26.88**	39.40	**27.53**
10	82.85	19.33	**81.84**	70.44	**26.79**	39.49	**27.69**

Figure 57: Performance of the span-corruption objective for different average span lengths. 15% corruption rate is used in all of the original text sequence.

Figure 58 is an overall conclusion on the best setting for pre-training objectives:

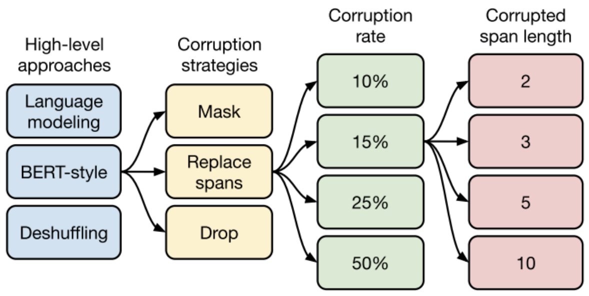


Figure 58: A flow chart of our exploration of unsupervised objectives

4. Pre-training dataset size
Unlabeled datasets
To verify the pre-training function of proposed dataset C4(illustrated in Dataset section) generated by the authors, a comparison is made with 5 other datasets: Unfiltered C4, RealNews-like, WebText-like, Wikipedia and Wikipedia + Toronto Books Corpus. All of them have different extraction sources and sizes. Here, unfiltered C4 means it does not filter out the bad words which are eliminated from C4.

Dataset	Size	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ C4	745GB	83.28	**19.24**	80.88	71.36	**26.98**	**39.82**	**27.65**
C4, unfiltered	6.1TB	81.46	19.14	78.78	68.04	26.55	39.34	27.21
RealNews-like	35GB	**83.83**	**19.23**	80.39	72.38	**26.75**	**39.90**	**27.48**
WebText-like	17GB	**84.03**	**19.31**	**81.42**	71.40	**26.80**	**39.74**	**27.59**
Wikipedia	16GB	81.85	**19.31**	81.29	68.01	**26.94**	39.69	**27.67**
Wikipedia + TBC	20GB	83.65	**19.28**	**82.08**	**73.24**	**26.77**	39.63	**27.57**

Figure 59: Performance resulting from pre-training on different datasets. The first four variants are based on our new C4 dataset.

We can see from the Table above that removing the bad words from C4 does help in improving performance. Besides, Wikipedia + PBC beats the record of C4 in several tasks, which because TBC covers exactly the domain of thoses tasks. Similarly, RealNews-like dataset performs the best on Exact Match score for ReCoRD, a dataset that measures reading comprehension on news articles. We can learn from the comparison that pre-training on in-domain unlabeled data can help in performance on downstream tasks. However, if we want to train a model which can be used in various tasks, we have to use data from arbitrary domains.

Pre-training dataset size
To test the influence of pre-training dataset size, the baseline model is trained on 5 different sizes corpus. As shown in table below, we can see that the performance

improves with the increasing size of dataset. One possible explanation is that the model memorizes the pre-training dataset owing to the repetition.

Number of tokens	Repeats	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Full dataset	0	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
2^{29}	64	**82.87**	**19.19**	**80.97**	**72.03**	**26.83**	**39.74**	**27.63**
2^{27}	256	82.62	**19.20**	79.78	69.97	**27.02**	**39.71**	27.33
2^{25}	1,024	79.55	18.57	76.27	64.76	26.38	39.56	26.80
2^{23}	4,096	76.34	18.33	70.92	59.29	26.37	38.84	25.81

Figure 60: Measuring the effect of artificially shrinking our C4 dataset

The training loss is shown in Figure above, we can confirm our assumption based on the low loss function attained by small dataset. As a result, the authors suggest to use as large pre-training datasets as possible.

5. Training strategy

   – Fine tuning method
     So far we've been discussed pre-training method, this section focus on training strategy. Except training the whole layers, we can also train layers partially. First, adapter layers can be adapted. The idea of adapter layer is to keep most of the original model fixed.Adatper layers are extra dense-ReLU sdense blocks which allows the output dimension matches input dimension. During fine-tuning, only adapyer layer and normalization parameters are updated. Second method for partial training is "gradual unfreezing" method, which means the layers are gradually unfrezzed during training.

Fine-tuning method	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ All parameters	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Adapter layers, $d = 32$	80.52	15.08	79.32	60.40	13.84	17.88	15.54
Adapter layers, $d = 128$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
Adapter layers, $d = 512$	81.54	17.78	79.18	64.30	23.45	33.98	25.81
Adapter layers, $d = 2048$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
Gradual unfreezing	82.50	18.95	79.17	**70.79**	26.71	39.02	26.93

Figure 61: Comparison of different alternative fine-tuning methods that only update a subset of the model's parameters. For adapter layers, d refers to the inner dimensionality of the adapters.

     As we can see from the Figure above, tasks with lower resource such as SQuAD performs better on lower dimention value d, whereas higher resource tasks need higher dimensionality. This means that it is feasible to improve fine-tune results using adapter layers when we know the task size. As for unfrezzing, it does speed up the fine-tuning but it also degrade the performance.

   – Multi-task learning

     The idea of multi-task learning is training a single model which can perform many tasks at the same. As a result, the training set should include multiple task sources, how to choose the data size of each task is a problem. Since if the size is too small, the task can never be learned. If the tasks are not relevant to each other, then it could case "negative transfer" problem, means the model would learn less when learn those tasks all together.
     There are three data fusion strategies: First is Examples-proportional mixing, means the sample is in proportional to the size of each task's dataset. Second one is Temperature-scaled mixing, means each size is abstracted by a portion $\frac{1}{T}$, where T is called temperature. The third one called Equal mixing, which means the example source is chosen in equal property from each task.
     The comparison results is in Table below, we can see that there are equal mixing strategy performs the worst, which may because low-resource tasks overfit and high-resource tasks underfit. As for k elements, there is a "sweet spot". This means

the performance improved as k increases and then decrease if k keep increasing. As for high resource course, the sweet spot may not true, since it always asks for more data to train. Finally, T=2 performs the best in most cases.

Mixing strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Baseline (pre-train/fine-tine)	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Equal	76.13	19.02	76.51	63.37	23.89	34.31	26.78
Examples-proportional, $K = 2^{16}$	80.45	19.04	77.25	69.95	24.35	34.99	27.10
Examples-proportional, $K = 2^{17}$	81.56	19.12	77.00	67.91	24.36	35.00	27.25
Examples-proportional, $K = 2^{18}$	81.67	19.07	78.17	67.94	24.57	35.19	27.39
Examples-proportional, $K = 2^{19}$	81.42	**19.24**	79.78	67.30	25.21	36.30	**27.76**
Examples-proportional, $K = 2^{20}$	80.80	**19.24**	**80.36**	67.38	25.66	36.93	**27.68**
Examples-proportional, $K = 2^{21}$	79.83	18.79	79.50	65.10	25.82	37.22	27.13
Temperature-scaled, $T = 2$	81.90	**19.28**	79.42	69.92	25.42	36.72	27.20
Temperature-scaled, $T = 4$	80.56	**19.22**	77.99	69.54	25.04	35.82	27.45
Temperature-scaled, $T = 8$	77.21	19.10	77.14	66.07	24.55	35.35	27.17

Figure 62: Multi-task training using different mixing methods. Examples proportional size is proportional to the total size of each dataset, with an artificial limit (K) on the maximum dataset size. Temperature-scaled mixing re-scales the sampling rates by a temperature T. For temperature-scaled mixing, an artificial dataset size limit of K = 2^{21} is applied.

– Multi-task learning and Fine-tuning combination
  Pure multi-task learning is to train a single model for different task, here we extend this approach by pre-training model on all tasks at once and then fine tune on individual supervised tasks. Here 5 models are considered. First model is the classic one task unsupervised learning, second is classic multi-task training, third is multi-task learning which combine fine-tuning and pre-training as described above. Fourth model is "leave one out", which means in pre-training, leave one task data out and use this task as fine-tuned data. This model aims to see the generalization ability of multi-task learning. The last model uses a supervised dataset for pre-training to see whether the absence of unsupaervised task will affect the result.

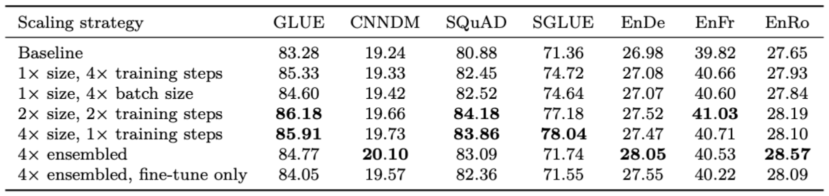
Training strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Unsupervised pre-training + fine-tuning	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	39.82	27.65
Multi-task training	81.42	**19.24**	79.78	67.30	25.21	36.30	27.76
Multi-task pre-training + fine-tuning	**83.11**	**19.12**	**80.26**	**71.03**	**27.08**	39.80	**28.07**
Leave-one-out multi-task training	81.98	19.05	79.97	**71.68**	**26.93**	39.79	**27.87**
Supervised multi-task pre-training	79.93	18.96	77.38	65.36	26.81	**40.13**	**28.04**

Figure 63: Comparison of unsupervised pre-training, multi-task learning, and various forms of multi-task pre-training.

  The result shows that multi-task learning with both finr-tuning and pre-training performs simialar to the baseline model. Leave one out model performs only slightly worse, which indicates that the model can generate the knowledge by pre-training. Finally, supervised multi-task pre training performs significantly worse except for translation, which means translation task benefit less from pre-training.

6. Scaling
   This chapter aims to examine the affectiveness of scaling, such as model size, training steps and ensembling. Here 6 scaling strategies are considered. The baseline model has 220M parameters and pre-trained and fine-tuned $2^1 9$ and $2^1 8$. 2*size means the model is twice big as baseline model. As for ensembled model, it ensemble 4 separately pre-trained and fine-tuned models and average their logits before feeding into softmax function for aggregate prediction, as shown in second last line. As for the last model, it is less time-consuming. It only trains one single model and produce 4 separate fine-tuned sub-models.
   As we can see from Figure64, there is no significant difference between training 4X steps and 4X batch size, both of which are beneficial. Increasing model size leads to an additional bumo compared to solely increasing training time or batch size. As for the observation of line 4,5, we can observe that increase training time or model

size can improve performance. Ensembling several models can improve performance, but without fine tuning separately performs a little worse but still better than baseline model, which offers a cheaper means to improve performance.



Scaling strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Baseline	83.28	19.24	80.88	71.36	26.98	39.82	27.65
1× size, 4× training steps	85.33	19.33	82.45	74.72	27.08	40.66	27.93
1× size, 4× batch size	84.60	19.42	82.52	74.64	27.07	40.60	27.84
2× size, 2× training steps	**86.18**	19.66	**84.18**	77.18	27.52	**41.03**	28.19
4× size, 1× training steps	**85.91**	19.73	**83.86**	**78.04**	27.47	40.71	28.10
4× ensembled	84.77	**20.10**	83.09	71.74	**28.05**	40.53	**28.57**
4× ensembled, fine-tune only	84.05	19.57	82.36	71.55	27.55	40.22	28.09

Figure 64: Comparison of different alternative fine-tuning methods that only update a subset of the model's parameters.

- Dataset
  In order to measure the effect of quality, characteristics and size of unlabeled data, the authors generate a new dataset by leveraging Common Crawl as text source. Common Crawl should not be unfamiliar with NLP task since it has been used in n-gram language model training, commonsense reasoning and pre-training et al. Common Crawl is cleaned by removing markup and other non-text content from HTML files. At this step, the data is still not natural language, but contains gibberish or boiler plate. Besides, not all sentences are relevant to our tasks. As a result, the follows strategies are used to extract text:

  – Only keep lines ended in a terminal punctuation mark.

  – Remove any page included word on "List of Dirty, Naughty, Obscene or Otherwise Bad Words".

  – Remove line with Javascript to avoid warning.

  – Remove pages containing phrase "lorem ipsum" to banish placeholder "lorem ipsum" text.

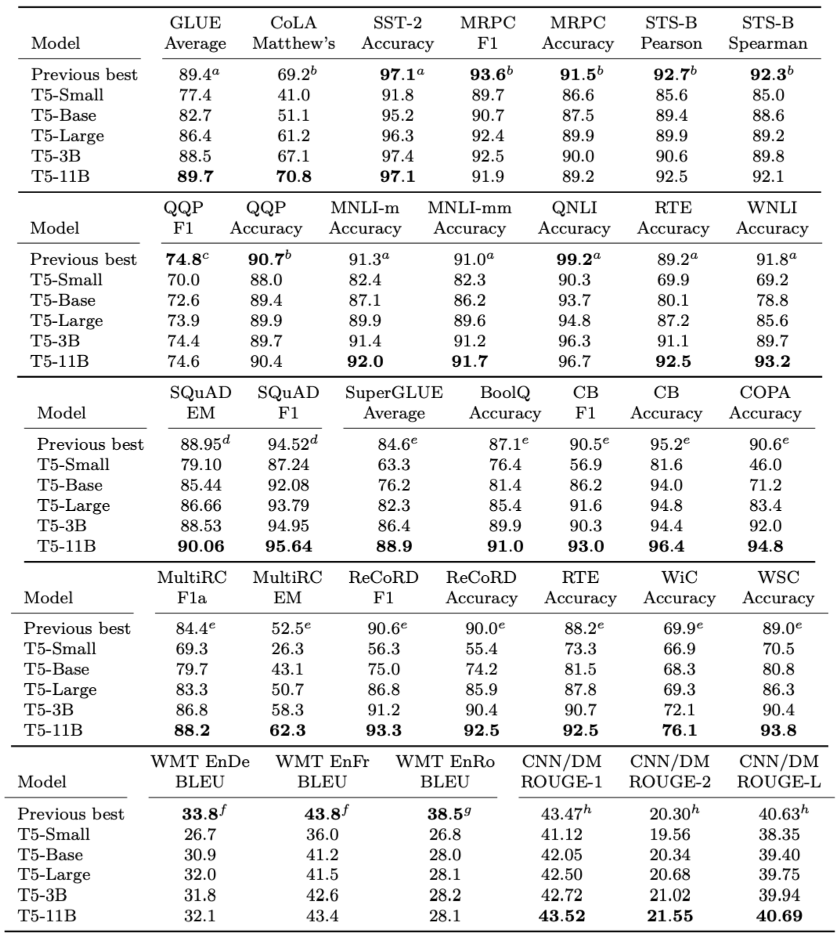
  – Remove pages containing curly bracket to eliminate code.

  – Remove all three-sentence span sentence to deduplicate dataset.

  – Filter out any pages which are not classified as English by langdetect.

  In the end, a dataset which is bigger than most pre-training datasets(about 750GB) and with clean and natural English text is generated. The dataset is named "Colossal Clean Crawled Corpus"(C4).

  When evaluating the performance of the model integrated with best structural strategies, multiple tasks are considered: machine translation, question answering, abstractive summarization and text classification. Here, GLUE and SuperGLUE focus on text classification tasks containing its own corresponding dataset such as Definite Pronoun Resolution(DPR). CNN/Daily Mail is for summarization; SQuAd for question answering and WMT English to German, French and Romanian translation. The data details can be seen in [47].



Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	MRPC F1	MRPC Accuracy	STS-B Pearson	STS-B Spearman
Previous best	89.4^a	69.2^b	**97.1**^{a}	**93.6**^{b}	**91.5**^{b}	**92.7**^{b}	**92.3**^{b}
T5-Small	77.4	41.0	91.8	89.7	86.6	85.6	85.0
T5-Base	82.7	51.1	95.2	90.7	87.5	89.4	88.6
T5-Large	86.4	61.2	96.3	92.4	89.9	89.9	89.2
T5-3B	88.5	67.1	97.4	92.5	90.0	90.6	89.8
T5-11B	**89.7**	**70.8**	**97.1**	91.9	89.2	92.5	92.1

Model	QQP F1	QQP Accuracy	MNLI-m Accuracy	MNLI-mm Accuracy	QNLI Accuracy	RTE Accuracy	WNLI Accuracy
Previous best	**74.8**^{c}	**90.7**^{b}	91.3^a	91.0^a	**99.2**^{a}	89.2^a	91.8^a
T5-Small	70.0	88.0	82.4	82.3	90.3	69.9	69.2
T5-Base	72.6	89.4	87.1	86.2	93.7	80.1	78.8
T5-Large	73.9	89.9	89.9	89.6	94.8	87.2	85.6
T5-3B	74.4	89.7	91.4	91.2	96.3	91.1	89.7
T5-11B	74.6	90.4	**92.0**	**91.7**	96.7	**92.5**	**93.2**

Model	SQuAD EM	SQuAD F1	SuperGLUE Average	BoolQ Accuracy	CB F1	CB Accuracy	COPA Accuracy
Previous best	88.95^d	94.52^d	84.6^e	87.1^e	90.5^e	95.2^e	90.6^e
T5-Small	79.10	87.24	63.3	76.4	56.9	81.6	46.0
T5-Base	85.44	92.08	76.2	81.4	86.2	94.0	71.2
T5-Large	86.66	93.79	82.3	85.4	91.6	94.8	83.4
T5-3B	88.53	94.95	86.4	89.9	90.3	94.4	92.0
T5-11B	**90.06**	**95.64**	**88.9**	**91.0**	**93.0**	**96.4**	**94.8**

Model	MultiRC F1a	MultiRC EM	ReCoRD F1	ReCoRD Accuracy	RTE Accuracy	WiC Accuracy	WSC Accuracy
Previous best	84.4^e	52.5^e	90.6^e	90.0^e	88.2^e	69.9^e	89.0^e
T5-Small	69.3	26.3	56.3	55.4	73.3	66.9	70.5
T5-Base	79.7	43.1	75.0	74.2	81.5	68.3	80.8
T5-Large	83.3	50.7	86.8	85.9	87.8	69.3	86.3
T5-3B	86.8	58.3	91.2	90.4	90.7	72.1	90.4
T5-11B	**88.2**	**62.3**	**93.3**	**92.5**	**92.5**	**76.1**	**93.8**

Model	WMT EnDe BLEU	WMT EnFr BLEU	WMT EnRo BLEU	CNN/DM ROUGE-1	CNN/DM ROUGE-2	CNN/DM ROUGE-L
Previous best	**33.8**^{f}	**43.8**^{f}	**38.5**^{g}	43.47^h	20.30^h	40.63^h
T5-Small	26.7	36.0	26.8	41.12	19.56	38.35
T5-Base	30.9	41.2	28.0	42.05	20.34	39.40
T5-Large	32.0	41.5	28.1	42.50	20.68	39.75
T5-3B	31.8	42.6	28.2	42.72	21.02	39.94
T5-11B	32.1	43.4	28.1	**43.52**	**21.55**	**40.69**

Figure 65: Performance of T5 variants on every task. Small, Base, Large, 3B, and 11B refer to model configurations with 60 million, 220 million, 770 million, 3 billion, and 11 billion parameters. First row of each table is the state-of-the-art result for the task, with the superscript denoting its source with references listed at the end of this caption.

- Wrap up Experiment
  Given the results of the experiments above, the final model uses BERT-style objective with a mean span length of 3 and 15% corruption rate; C4 as pre-traning dataset; To test the relationship between sizes and resources limitation,4 sizes are introduced: Base, Small, Large and "3B and 11B". milti-task pre-training style. To avoid negative transfer, we substitute the following artificial dataset sizes for our unlabeled data before using standard example-proportional mixing (described in Section 3.5.2): 710,000 for Small, 2,620,000 for Base, 8,660,000 for Large, 33,500,000 for 3B, and 133,000,000 for 11B. The results are shown as the following:

  Overall, the paper achieved state of-the-art performance on 17 out of the 24 tasks it considers. As expected, the largest (11 billion parameter) model performed best among all model size variants across all tasks. The T5-3B model variant did beat the previous state of the art in a few tasks, but scaling the model size to 11 billion parameters was the most important ingredient for achieving the best performance

4.1.9 Summary

This section covers 8 papers in total, with 4 Transfer Learning paper and 3 Multi-task Learning paper and one Transfer Learning survey. Based on the papers introduced above, Transfer Learning can be used in applications such as Reading Comprehension, Answer Sentence Selection, Dynamics

Thermal Models Building and Negation Detection, while Multi-task learning can be applied in Negation Detection, Natural understanding and translation.

4.2 Protein

4.2.1 MODELING THE LANGUAGE OF LIFE —— DEEP LEARNING PROTEIN SEQUENCES

Tags:*Protein Structure Prediction, Protein function prediction, General Domain, ELMo, Transfer Learning*

- Background
  How to predict protein structure and function from amino acid has always been a popular task in Computational Biology. With the increasing amount of protein datasets, machine learning and evolutionary information derived from multiple sequence alignments(MSA) of related proteins in the datasets are married to solve these problems and achieves great results. However, there are several limitations on evolutionary information: First, finding MSA sequences can be very computationally expensive. Plus, for small protein families, evolutionary information is not available, such as *Dark Proteome*. As a result, the paper[48], instead of searching evolutionary related proteins, transfers biophysical information generated by using bi-directional language model ELMo from large, unlabeled sequence dataset(UniRef50). The authors take references from NLP tasks, where models are pre-trained on large unlabeled corpus such as Wikipedia for next word prediction, and assumes that protein sequences can be learned like NLP sentences by ELMo model. Note that there are several differences between NLP and protein tasks. First, Protein has longer longest sequence than NLP, with 33,000 and 1024 separately, which is challenge for long-range dependencies capture. Second, most of proteins only use 20 amino acids and 5 additional characters, while there are millions words in NLP. Smaller vocabularies could cause similar proteins encoding problem. Third, the size of UniRef50 is almost ten times larger than largest NLP corpus, which increases training time by ten-fold.

- Model
  The core idea of the process is: first pre-trained a bi-directional ELMo model using UniRef50, then using the embeddings got from ELMo model to two prediction tasks:per-residue(word-level) and per-protein(sentence-level) prediction to assess the predictive ability of ELMo.
  ELMo adaption: ELMo is used to generate word embedding vectors, users don't need to fine-tune the model on specific datasets, which increase flexibility of the model. The model first receive a protein sequence as an input, and then it returns 3076 features for each residue in the protein sequence. The features are divided into 3 parts and fed into 3 internal ELMo layers, which makes each token a vector of length 1024. Since there is no fine-tune step in the model, the authors make a little change on the structure of ELMo to increase the generalization ability: First, reduce the token type size to 28(20 standard and 2 rare amino acids plus 3 special tokens and 3 tokens to indicate location). Second, increase unroll steps number to 100. Third, lower negative samples to 20 Last but not least, increase total token number to 9,688,889,953(token number of UniRef50).
  Per-residue level task: A two layer CNN model is trained to show the prediction power of SeqVec embedding from ELMo. As shown in the left half of Figure below: The first CNN layer contains 32 filters each with window size w=7. Then after going through a rectified linear unit(ReLU) and drop out, it goes to the second layer, where another CNN layer with window size 7 is applied again. The model is trained on several different input combinations: *DeepProf(14k parameters), DeepSeqVec(232K parameters), DeepProf+SeqVec(244k parameters), DeepProtVec(25k parameters), DeepOneHot(7k parameters) and DeepBLOUSUM65(8k parameters).*The details are covered in paper[48].
  Per-protein level task: It is demostrated by a feed forward neural network model. To get a single-sized input, it average the embedding of all residues in a certain protein sequence giving a 1024 dimentional vector for each protein. After a hidden layer, the features are compressed into 32 features. The final prediction can be got after batch normailization, dropout and activation function Relu.

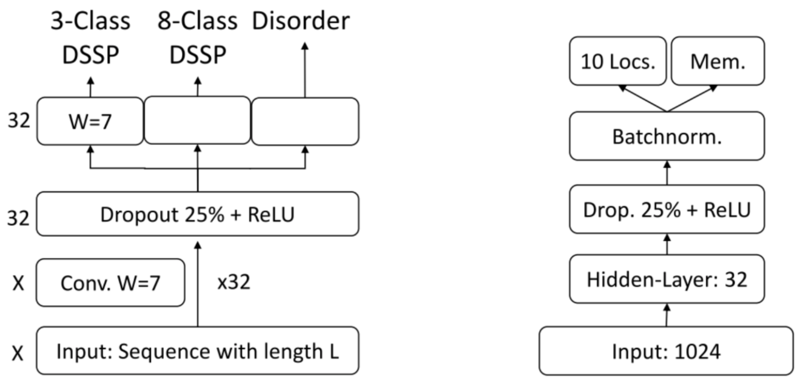


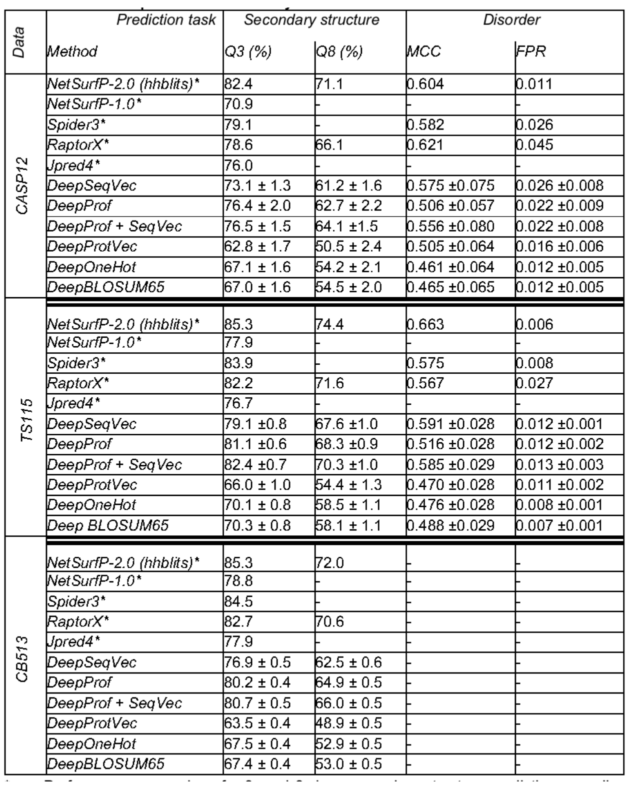
Figure 66: Left diagram shows the architecture for residue-level prediction. W is the kernal size of CNN(W=9 means convolition of size 9*1) The right diagram is the structure of protein-level task.

- Dataset
  UniRef50: For pre-training, the paper uses UniRef50 to train ELMo for Sequence Vectors. UniRef50 is a redundancy reduced subset of UniProt database, with 50% pairwise sequence identity(PIDE). It contains 33M proteins and 9,577,889,953 residues. Each protein sequence is treated as a sentence and amino acid as a single word.
  NetSurfP-2.0: It is used to train pre-residue level tasks such as secondary structure prediction and intrinsic disorder detection. NetSurfP-2.0 is published by a recent work which performs well on secondary structure prediction.[49]. It has 10,837 sequence unique proteins sequence. Sequences are mapped by "structure integration with function, taxonomy and sequence'(SIFTS), those who has same length with NetSurfP-2.0 from mapping are kept.
  There are 3 test sets: **TS115**: 115 proteins from high-quality structures. **CB513**:513 non-redundant sequences and **CASP 12**:21 protein from CASP12 free-modeling targets.

  DeepLoc: DeepLoc is for per-protein level tasks such as localization prediction and membrane proteins detection. DeepLoc is released in [49], it is used to train and validate Localization prediction task. Plus, the author of DeepLoc make it map input proteins to ten classes, with additional labels whether is is water soluble or membrane bound.

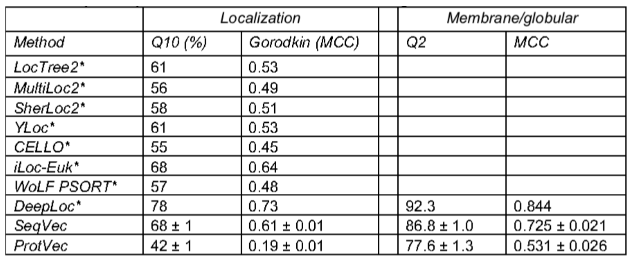
- Experiment
  As for per-residue performance, secondary structure task uses three-state perresidue accuracy(Q3) and eight-state analog(Q8) measures. while disorder is evaluated by Matthew's correlation coefficient (MCC) and the False-Positive Rate (FPR).
  As for Per-protein performance, it uses two-state per set accuracy (Q2) ten-state per set accuracy (Q10) and MCC.

  As we can see from Fig67, for per-residue tasks the proposed model achieves high performance but not the best.
  For second structure prediction, NetSurP2.0 gets the best score. Using evolutionary information(DeepProf with HHblits profiles) has 4-6 percentages lower than NetSurfP2.0(Q3=76-81%). SeqVec embedding achieves 2-3 percentages below only using evolutionary information model(Q3=73%-79%). While combination of sequence and envolutional message improve single usage a little but still doesn't get the top performance (Q3=77-82%). For intrinsic disorder task, NetSurfP-2.0 still perform best. However, embedding alone task(DeepSeqVec) performs better than evolutional model(DeepProf) this time. Combination of evolutionary information with embedding didn't beat SeqVec.



Data	Prediction task	Secondary structure		Disorder	
	Method	Q3 (%)	Q8 (%)	MCC	FPR
CASP12	NetSurfP-2.0 (hhblits)*	82.4	71.1	0.604	0.011
	NetSurfP-1.0*	70.9	-	-	-
	Spider3*	79.1	-	0.582	0.026
	RaptorX*	78.6	66.1	0.621	0.045
	Jpred4*	76.0	-	-	-
	DeepSeqVec	73.1 ± 1.3	61.2 ± 1.6	0.575 ±0.075	0.026 ±0.008
	DeepProf	76.4 ± 2.0	62.7 ± 2.2	0.506 ±0.057	0.022 ±0.009
	DeepProf + SeqVec	76.5 ± 1.5	64.1 ±1.5	0.556 ±0.080	0.022 ±0.008
	DeepProtVec	62.8 ± 1.7	50.5 ± 2.4	0.505 ±0.064	0.016 ±0.006
	DeepOneHot	67.1 ± 1.6	54.2 ± 2.1	0.461 ±0.064	0.012 ±0.005
	DeepBLOSUM65	67.0 ± 1.6	54.5 ± 2.0	0.465 ±0.065	0.012 ±0.005
TS115	NetSurfP-2.0 (hhblits)*	85.3	74.4	0.663	0.006
	NetSurfP-1.0*	77.9	-	-	-
	Spider3*	83.9	-	0.575	0.008
	RaptorX*	82.2	71.6	0.567	0.027
	Jpred4*	76.7	-	-	-
	DeepSeqVec	79.1 ±0.8	67.6 ±1.0	0.591 ±0.028	0.012 ±0.001
	DeepProf	81.1 ±0.6	68.3 ±0.9	0.516 ±0.028	0.012 ±0.002
	DeepProf + SeqVec	82.4 ±0.7	70.3 ±1.0	0.585 ±0.029	0.013 ±0.003
	DeepProtVec	66.0 ± 1.0	54.4 ± 1.3	0.470 ±0.028	0.011 ±0.002
	DeepOneHot	70.1 ± 0.8	58.5 ± 1.1	0.476 ±0.028	0.008 ±0.001
	Deep BLOSUM65	70.3 ± 0.8	58.1 ± 1.1	0.488 ±0.029	0.007 ±0.001
CB513	NetSurfP-2.0 (hhblits)*	85.3	72.0	-	-
	NetSurfP-1.0*	78.8	-	-	-
	Spider3*	84.5	-	-	-
	RaptorX*	82.7	70.6	-	-
	Jpred4*	77.9	-	-	-
	DeepSeqVec	76.9 ± 0.5	62.5 ± 0.6	-	-
	DeepProf	80.2 ± 0.4	64.9 ± 0.5	-	-
	DeepProf + SeqVec	80.7 ± 0.5	66.0 ± 0.5	-	-
	DeepProtVec	63.5 ± 0.4	48.9 ± 0.5	-	-
	DeepOneHot	67.5 ± 0.4	52.9 ± 0.5	-	-
	DeepBLOSUM65	67.4 ± 0.4	53.0 ± 0.5	-	-

Figure 67: Per-residue prediction



	Localization			Membrane/globular	
Method	Q10 (%)	Gorodkin (MCC)		Q2	MCC
LocTree2*	61	0.53			
MultiLoc2*	56	0.49			
SherLoc2*	58	0.51			
YLoc*	61	0.53			
CELLO*	55	0.45			
iLoc-Euk*	68	0.64			
WoLF PSORT*	57	0.48			
DeepLoc*	78	0.73		92.3	0.844
SeqVec	68 ± 1	0.61 ± 0.01		86.8 ± 1.0	0.725 ± 0.021
ProtVec	42 ± 1	0.19 ± 0.01		77.6 ± 1.3	0.531 ± 0.026

Figure 68: Per-protein prediction

For localization task, embedding alone didn't reach top score, but very close. Plus, it outperforms evolutionary information by up to Q10=13%.

Performance on membrane-bound and water-soluble proteins classification followed a similar trend: while DeepLoc still performed best (Q2=92.3, MCC=0.844), sequential model SeqVec performs just slightly lower than DeepLoc (Q2=86.8+/-1.0, MCC=0.725+/-0.021). However, another method using only single sequences– ProtVec performed significantly worse (Q2=77.6+/-1.3, MCC=0.531+/-0.026).

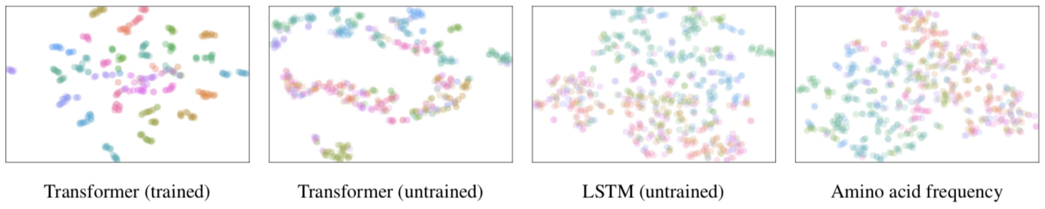
4.2.2 BIOLOGICAL STRUCTURE AND FUNCTION EMERGE FROM SCALING UNSUPERVISED LEARNING TO 250 MILLION PROTEIN SEQUENCES

Tags:*protein structure prediction, protein function prediction, General Domain, Transformer, Transfer Learning*

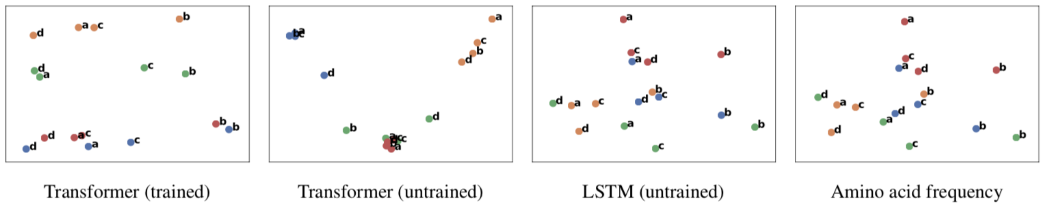
- Background
  Due to the decreasing costs of sequencing technology, the biological sequence datasets

size is growing exponentially. With the help pf artificial intelligence technology, predictive and generative techniques can be extracted from sequence data. Many researchers such as Yanofsky et al[50] Goebel et al[51] and Altschuh[52] have allowed the opinion that biological function and structure are encoded in the statistics of protein sequences. As a result, the goal of the paper[53] is to extract general and transferable protein information from raw sequences.
Self-supervised language modeling has achieve a good performance on many text processing tasks, the paper uses this modeling method to protein data instead of text data. Given amino acid sequence has longer and smaller vocabulary than text data, it is unclear if the idea will fit in the new domain.

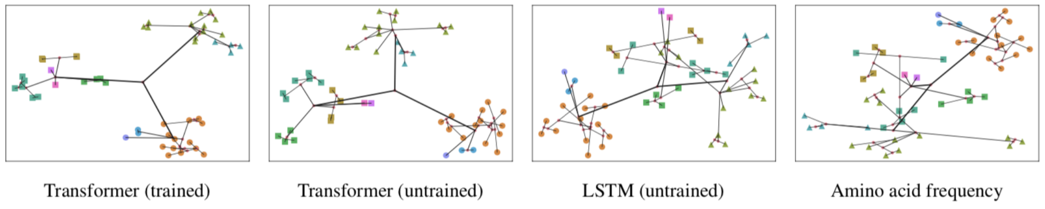
- Model
  Bi-directional Transformer is used for protein structure and function emergence. The input of the model are raw character sequence of amino acids. Then the inputs are fed into internal blocks with self-attention mechanism which enables the model to build up contextual connections across sequence and model long range pairwise dependencies among amino acids. The final hidden representation output of the Transformer are sequence vectors, one for each character in sequence. For orthologous protein embedding, the output is summed by averaging features among all sequence characters. Each sequence can be presented by one point. The details are shown in experiment section.



(a) Sequence representations (t-SNE)



(b) Sequence representations (PCA)



(c) Species representations with phylogeny overlay

Figure 69: Protein sequence representations encode and organize orthological, species, and phylogentic information of sequences. (a)visualizes representations of several orthologous groups from different species via t-SNE.Sequences from the same orthologous group have the same color.(b)visualizes ortholog sequence representations via PCA (a linear projection technique).Character label denotes species and genes are colored by orthologous group.(c)visualizes per-species averaged sequence representations via a learned hierarchical distance projection; a phylogenetic hierarchy (colors represent distinct phyla) is overlaid.

- Dataset
  The model is trained on a large protein sequence dataset which contains a rich evolutionary

sequence samples called Uniparc database. It includes 250 million sequences with 86 billion amino acids. The size of the data can compare to the data used in high capacity neural network architecture on natural language.

- Experiment
  Embeddings of Orthologous Proteins Orthologous means the genes which are derived from a common ancestor by speciation. Those genes share same function while the sequences are different. This task aims to find orthology by protein embeddings got from Transformer.

  Similarity-based spatial organization of orthologous genes: Orthologous genes share high similarity in sequence order from their ancestry. The similarity can be captured by frequency baseline, as shown in Figure51(a), LSTM clusters are similar to frequency baseline, the clusters in untrained Transformer are more compact. While trained Transformer shows a more tight cluster representing for each orthologous genes, which means it identifies the similarity of amino acids.

  Principal components and directional encoding: This tasks encodes both orthology and species information by *direction* in vector to find similar representations of orthologous genes. In this case, the task uses principal component analysis(PCA) for directions recovery of variation in representative vectors. It chooses 4 orthologous genes across 4 species to observe the directions among them. In figure 51(b), the horizontal axis is species and vertical axis represents for ortholog family. We can see the baseline representation clusters well on vertical axis but not on horizontal ones. As for other methods, LSTM is basically the same as baseline, while untrained Transformer has a tighter cluster, which means Transformer architecture can capture higher order information without any learning.

  Phylogenetic organization: Species representations can be shown by averaging per-protein embeddings of protein-subsets. As illustrates from 51(c), it shows the projected representations of 48 randomly-selected species evenly split across the three taxonomic domains, suggests that Transformer representations cohere strongly with natural phylogenetic organization, in contrast to the LSTM and frequency baselines.

4.2.3 Evaluating Protein Transfer Learning with TAPE

Tags:*Protein model evaluation, Structure Prediction, Evolutionary Understanding, Protein Engineering, DNN, Transfer Learning*

- Background
  With the development of sequencing technologies, the size of protein databases explodes. However, there is a big gap between the size of protein sequence databases and size of labeled subsets. As a result, self-supervised learning is used to extract information from unannotated data, which works quite well in NLP domain. To figure out whether it is feasible, the paper[54] introduces Tasks Assessing Protein Embedding(TAPE), which is a evaluation on semi-supervised learning on protein sequences. TAPE evaluate the performance of learned protein embedding on five different supervised tasks.

- Model
  This paper is consist of 3 architectures: an LSTM, a Transformer and a dilated residual network. Those 3 architectures are designed to have similar amount of parameters. LSTM contains two three layer LSTMs with 1024 hidden units, while Transfermer consists of 12 layer with 512 hidden units and 8 attention heads. ResNet has 35 residual blocks, each with two convolutional layers, 256 filters and kernal size 9.

- Dataset
  Unsupervised and supervised datasets are introduced in this area: **Unlabeled Sequence Dataset**

Pfam[55], which is a database with 31 million protein domains, is used as a pre-training corpus for TAPE. Pfam is clustered into several *families* based on their evolution. 1% of data are constructed as a fully heldout families, and remaining data is split into two parts: 95% training data and 5% test data. The training and testing sets can help test in-diestribution generalization, while heldout families measures out-of distribution generalization.

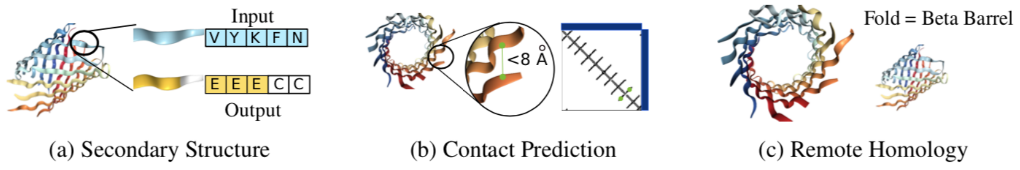


Figure 70: Structure of supervised tasks. (a) Secondary structure with input amino acide as blue segment and corresponding labels output as yellow and whilte. (b)A contact map, indicating whether amino acids at position i,j in the sequence are within 8 angstroms of each other or not. (c) Fold level remote homology class for the protein.

Supervised Dataset
TAPE includes 5 downstream supervised tasks, which can be categorised into 3 kinds: Structure prediction, evolutionary understanding and protein engineering:
Secondary Structure(SS) Prediction(Structure Prediction Task):SS is an important character for protein prediction, which is used to enrich the features for higher-level models. The task uses Klausen et al.[49], each input amino acid x_i is mapped to a label $y_i \in Helix, Strand, Other$ as shown in Figure(a).
Contact Prediction(Structure Prediction Task): As shown in Figure51(b). In this task, each input amino acids pair x_i, x_j from sequence x is mapped to a label $y_{ij} \in 0, 1$, with label 0 means the amino acides are not "in contact" (<8A apart), and 1 means they are in contact. The function can be used to robust 3D protein structure modeling. The data is from ProteinNet dataset.
Remote Homology Detection(Evolutionary Understanding Task): Detecting remote homology is widely used in microbiology and medicine domain for tasks such as detecting emerging antibiotic resistant genes or discovering new CAS enzymes. The input is mapped to a label ranging from 1 to 1195, representing different possibilities of protein folds. The details are illustrated in Figure51(c). The data is from Hou et al[56].

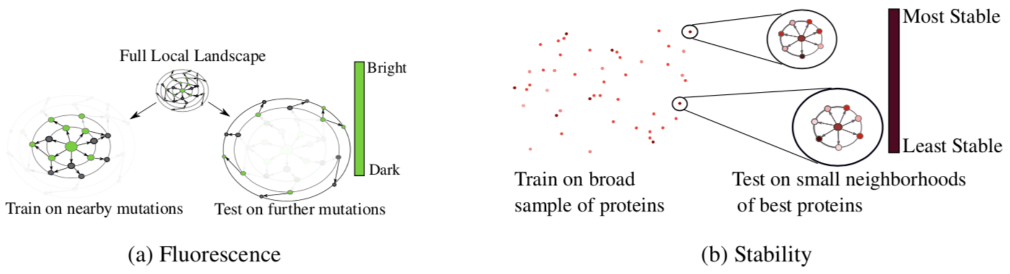

Full Local Landscape
Bright
Dark
Train on nearby mutations
Test on further mutations
(a) Fluorescence
Most Stable
Least Stable
Train on broad sample of proteins
Test on small neighborhoods of best proteins
(b) Stability

Figure 71: Protein Engineering Tasks. A parent protein is mutated to explore the local landscape in both tasks. The dots represent proteins and direction arrow $x- > y$ means y is the next mutation of x of parent p. (a)The Fluorescence task consists of training on small neighborhood of the parent green fluorescent protein (GFP) and then testing on a more distant proteins. (b) The Stability task consists of training on a broad sample of proteins, followed by testing on one-mutation neighborhoods of the most promising sampled proteins.

Fluorescence Landscape Prediction (Protein Engineering Task):This regression task maps each input protein x to its log-flourescence intensity y, which measures the ability to distinguish between very similar inputs. The data is from Sarkisyan et al[57]. Details are shown in Figure 53(a).

Stability Landscape Prediction (Protein Engineering Task):The input of this regression task is mapped to a label which indicates what is the most extreme circumstances the input protein x can maintain its fold above a concentration threshold. The data is from Rocklin et al[58]. The task measures the model's ability to generalize from a broad sampling of related protein sequences and information localization.

- Experiment
  We can see from Figure below that self-supervised model with pre-training phrase outperforms almost all models and tasks. However, alignment based input shows powerful performance on secondary structure prediction task and contact prediction task. That's why many state of the art prediction tasks use alignment based inputs.
  Among the three models, Transformer performs worst in secondary structure prediction, but has best performance on fluorescence and stability tasks. While ResNet is the contrary. This indicates it's not wise to judge a model's performance on a single task, but creating multi-task benchmarks such as TAPE.

Method		Structure		Evolutionary	Engineering	
		SS	Contact	Homology	Fluorescence	Stability
No Pretrain	Transformer	0.70	0.32	0.09	0.22	-0.06
	LSTM	0.71	0.19	0.12	0.21	0.28
	ResNet	0.70	0.20	0.10	-0.28	0.61
Pretrain	Transformer	0.73	0.36	0.21	**0.68**	**0.73**
	LSTM	0.75	0.39	**0.26**	0.67	0.69
	ResNet	0.75	0.29	0.17	0.21	**0.73**
Supervised [11]	LSTM	0.73	0.40	0.17	0.33	0.64
UniRep [12]	mLSTM	0.73	0.34	0.23	0.67	**0.73**
Baseline	One-hot	0.69	0.29	0.09	0.14	0.19
	Alignment	**0.80**	**0.64**	0.09	N/A	N/A

Figure 72: Results on downstream supervised tasks(The measurement is accuracy or precision, the higher the better. Maximum value is 1.0)

4.2.4 UDSMProt: Universal Deep Sequence Models for Protein Classification

Tags:*Protein Classification, Gene ontology prediction, remote homology and fold detection, General Domain, LSTM, Transfer Learning Learning*

- Background
  It is important in bioinformatics to deduce protein properties from amino acids sequences. Many methods use additional handcrafted features combine with amino acid sequences for protein properties deduction. The features usually includes functional annotations and homologous proteins sequence incorporation. However, handcrafted features rely heavily on experience, which could fail in generalization. Besides, the number of features should be proportional to the size of database, which means the time complexity grows exponentially with database size.
  Deep learning can be used to solve the above problem by directly predict properties by processing amino acid sequences. Self-supervised models have been used in Natural Language Processing across various tasks and receive good performances. Self-supervised models first pre-train model on large unlabeled data to learn implicit knowledge. Then fine-tuned on specific dataset.
  Similar to the application in NLP, protein classification tasks sees amino acids sequences as words, protein and its domains as text paragraphs and uses self-supervised learning to build classification models. There are many literature approaches for this domain, which shows pretraining models using self-supervised learning gain a huge improvement on performance compared to those models trained from scratch on many tasks. The problem is, none of those models make experience comparison with state of the art approaches which rely on handcrafted features such as Position-Specific Scoring Matrics (PSSM) derived via BLAST.

This paper[59] aims to pre-trained a deep protein classification model on Swiss-Prot and fine-tuned on gene ontology prediction and remote homology and fold detection tasks. Then it also shows a explicit comparison to SOTA featured based models.

- Model
  UDSMProt
  UDSMProt is abbreviation of Universal Deep Sequence Models for Protein Classification. The idea is to pre-train a SOTA RNN model on language modeling tasks. The goal of UDSMProt is to train a model which can fit to different classification problems by a single shared architecture. In fine tuning phrase, the initial weights are obtained from pretraining.

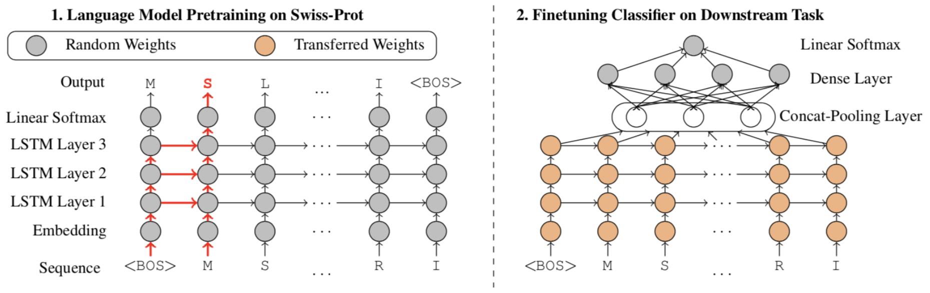


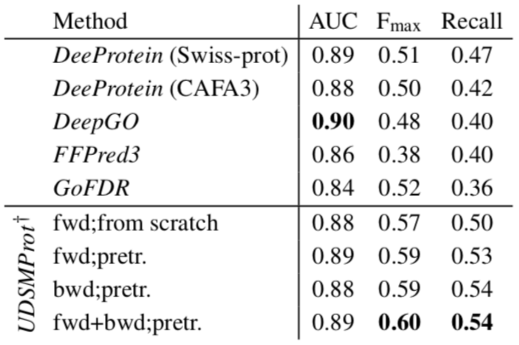
Figure 73: Schematic illustration of the training procedure. The inputs are amino acid sequence. <BOS> represents for the beginning of the sequence. Red arrow means context transfer. The right plot is for fin-tuning, all weight are initialized by pre-training step.

  In this paper[59] it uses 3-layer AWD-LSTM language model. As illustrated in Figure31(a), the model is constructed as follows: The input is from Swiss-Prot databse, after internal layers, the output layer is replaced by a pooling layer and two fully connected layers. In fine-tuing step, the layers are unfrozed layer by layer for optimization, during which learning rate is reduced by 2 percent compared to the previous layer. To get bi-directional information, forward and backward LSTM ar trained separately. The paper combines them by averaging the output probabilities of those two classifying models.
  Baseline Model
  To present a in-depth comparisons, several baseline models are introduced. Those models have SOTA performance on classification literature benchmarks using only feature-extracted methods. Features are chosen from PSSM, which is a beneficial input features matrix based on multiple sequence alignment by position specific iteration called BLAST(PSI-BLAST). It is used for protein relatives search, where closely related proteins is detected for initial general profile sequence generation. The new generated sequence is used as the query for next profile sequence searching.

- Dataset
  The model is trained on Swiss-Prot database[60], which contains 26 unique amino acides with 20 standard and 6 non-standard amino acids. The size of Swiss-Prot database is 560K, which has a complete annotation of protein properties.
  SCOP 1.67 dataset is used in remote homology and fold detection task. In this dataset, all protein is identified by their super family clusters, which is easy for homology classification tasks.

- Experiment
  Gene Ontology Prediction
  The goal of Gene Ontology is to unify a subset of vocabulary for protein attributes representation. It includes 3 domains: cellular components, molecular functions and biological processes, here we only focus on molecular function(MF). *DeeProtein, DeepGo, FFPred3 and GoFDR* are models relied on handcrafted features, they trained on different dayasets. For example, DeeProtein is trained on two different datasets Swiss-Prot and

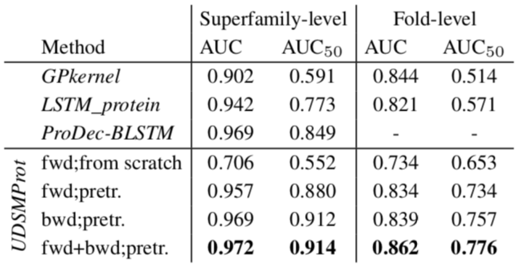
CAFA3, with 50% higher sequence similarity than Swiss-Prot.
As shown in the table below, UDSMProt models perform better than previous handcrafted features models, in both F score and recall. And forward-backward model is the best in UDSMProt model. This means pretrining model can transfer implicit knowledge from previous data.



	Method	AUC	F_{max}	Recall
	DeeProtein (Swiss-prot)	0.89	0.51	0.47
	DeeProtein (CAFA3)	0.88	0.50	0.42
	DeepGO	**0.90**	0.48	0.40
	FFPred3	0.86	0.38	0.40
	GoFDR	0.84	0.52	0.36
UDSMProt†	fwd;from scratch	0.88	0.57	0.50
	fwd;pretr.	0.89	0.59	0.53
	bwd;pretr.	0.88	0.59	0.54
	fwd+bwd;pretr.	0.89	**0.60**	**0.54**

Figure 74: Gene ontology (Molecular function) prediction performance evaluated on the DeepGO,FFPred3, DeeProtein and GoFDR

Reomote Homology and Fold Detection
Remote Homology can be used to classify protein structure and function, which is very important in computational biology. Some reference models are introduced, such as ProDec-BLSTM which is a bi-directional RNN trained by PSSM input features and GPKernel model, which applys kernel methods.
The metrics covered in this tasks are AUC and AUC50. AUC(50) calculates the Area under the curve up to the first 50 (x) negative examples, which allows for a better characterization of the classifier in the domain of small false positive rates than AUC.



		Superfamily-level		Fold-level	
	Method	AUC	AUC_{50}	AUC	AUC_{50}
	GPkernel	0.902	0.591	0.844	0.514
	LSTM_protein	0.942	0.773	0.821	0.571
	ProDec-BLSTM	0.969	0.849	-	-
UDSMProt	fwd;from scratch	0.706	0.552	0.734	0.653
	fwd;pretr.	0.957	0.880	0.834	0.734
	bwd;pretr.	0.969	0.912	0.839	0.757
	fwd+bwd;pretr.	**0.972**	**0.914**	**0.862**	**0.776**

Figure 75: Remote homology and fold detection performance on the SCOP 1.67 benchmark dataset compared to GPkernel, LSTM-protein and ProDec-BLSTM

As shown above, LSTM model outperforms UDSMProt model which is trained from scratch. This may because the small amount of data in this tasks. But this problem is overcomed by pre-training. We can see forward+backward pretraining model performs the best at both superfamily detection and fold level detection.

4.2.5 Predicting human protein function with multi- task deep neural networks

Tags:*Protein function Prediction, General Domain, DNN, Multi-task Learning*

- Background
  The function of majority of amino acid sequences still partly or completely unknown, since there are over 60 million sequences in UniProtKB database, while only 600 thousand of

them are functional annotated on Gene Ontology(GO). People try to transfer sequences knowledge by mapping between controlled vocabularies and GO from orthologous proteins and family assignments. However, sometimes there are protein sequences which has no annotations and lack information for at least one GO domain.
Machine learning can help with the problem. It can model the relationship between function and features extracted from source data. In this case, even if the protein sequence is lack of similar sequences with known function or has misleading alignment results, the model can still generate the function by the sequence pattern.
This paper[61] seeks to investigate the performance of multi-task DNN(MTDNN) in protein function prediction task which is believed to gain extra score from the dependencies among functional classes.

- Model
  MTDNN: The model is consists of two stages, first stage is a feedforward layers shared among all tasks and the second stage is task-specific feedforward layers where the tasks are parallel stacked upon the first stage. The structure figure is shown in Figure51(a) below. Each hidden layer if fully connected to the previous layer, with batch normalization and dropout implement during the feed forward transportation.

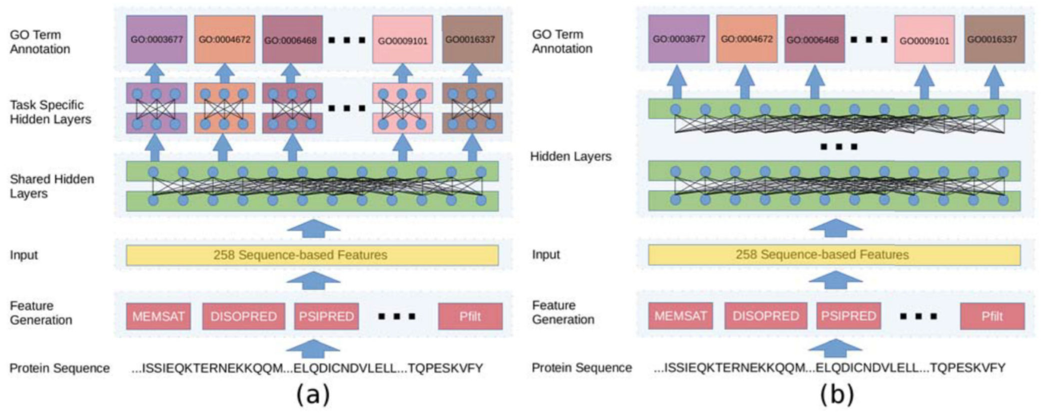


Figure 76: (a)Schematic diagram of MTDNN. MTDNN trains separate models for each tasks for different vocabulary in GO terms (b)MLDNN. It trains one model to classify different labels.

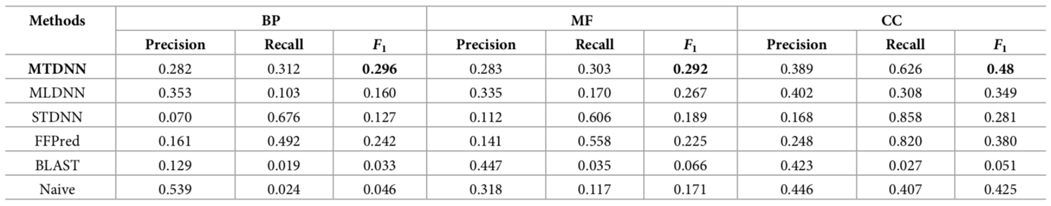
  After the shared hidden layers, the model trained separate models for each branch to predict subset of descendants in the vocabulary. For example, there are three domains in each GO terms which are used to test the model: biological process(BP), molecular function(MF) and cellular component(CC). For vocabulary GO:0002376, it has 38 branches in total: 18 BP domain, 11 MF and 9 in CC domain. MTDNN returns one combined result by computing the mean value of different branches.

  Multi-label deep neural networks(MLDNN): MLDNN is also a feed forward multi-layer architecture. As shown in Figure51(b). The difference is the output is generated simultaneously activated by sigmoid functions, the output is returned as a confidence score for classifying the term.

  Single task deep neural networks(STDNN): It emplys a single fully connected feedforward DNN for a single GO term. As a result, the number of model should be equal to the number of terms. Each of them are independent from each other.

- Dataset
  Protein annotations are achieved from Gene Ontology Annotation(GOA) database. GO is also used for term definitions and semantic relations. There are three domains in GO terms which are used to test the model: biological process(BP), molecular function(MF) and cellular component(CC), each domain has at least 150 annotations and 500 putative negative examples.

- Experiment
  Precision, Recall and F_1 score are three measurements for testing.
  The evaluation is based on standard measure of binary classification accuracy. MTDNN has highest F_1 score in all three domains, while naive method and BLAST perform the worst. As for recall, FFPred and STDNN has highest score, but lower precision. MTDNN is the contrary, with higher precision but lower recall. We can see FFPred and STDNN has high true positive but also high false positive. MTDNN provides a more balanced result on precision and recall over those domains, while keeping a high F_1 score.



Methods	BP			MF			CC		
	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
MTDNN	0.282	0.312	**0.296**	0.283	0.303	**0.292**	0.389	0.626	**0.48**
MLDNN	0.353	0.103	0.160	0.335	0.170	0.267	0.402	0.308	0.349
STDNN	0.070	0.676	0.127	0.112	0.606	0.189	0.168	0.858	0.281
FFPred	0.161	0.492	0.242	0.141	0.558	0.225	0.248	0.820	0.380
BLAST	0.129	0.019	0.033	0.447	0.035	0.066	0.423	0.027	0.051
Naive	0.539	0.024	0.046	0.318	0.117	0.171	0.446	0.407	0.425

Figure 77: Performance comparison for different models

4.2.6 Summary

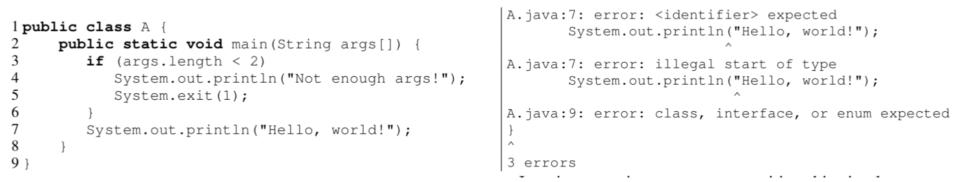
5 Papers about protein modality are introduced in this research paper, 4 of them use Transfer Learning and 1 uses Multi-task learning. Although all of the 5 models uses deep learning architecture, only 2 paper is based on Language Models such as ELMo and Transformer and all those papers are released in recent 2 years. It shows applying language models in protein domain is still an green area waiting for exploration. In this case Transfer Learning is popular than Multitask learning. According to the paper covered in this section, Transfer learning can be used in protein structure prediction, function prediction, classification, while Multi-task learning is used in function prediction for gene ontology database. The limitation of Multi-task learning in protein domain could be it is tricky to choose the training dataset, since it needs to encode useful shared information. The choice of Transfer Learning is flexible by comparison.

4.3 Source Code

4.3.1 Syntax and Sensibility: Using language models to detect and correct syntax errors

Tags:*Syntax detection, General domain, LSTM, Transfer Learning*

- Background
  Both naive and experienced programmers can meet syntax errors, this is an issue especially for naive programmers who don't have much experience in debugging. However, current debugging tools could be misleading. For example, see Figure below. Left graph is source code in Java, it misses another right brace } to match the { in line 3. The right graph is the error report. We can see it misleads users with wrong location identification, indicating other lines instead of line 3.



```
1 public class A {
2     public static void main(String args[]) {
3        if (args.length < 2)
4           System.out.println("Not enough args!");
5           System.exit(1);
6        }
7        System.out.println("Hello, world!");
8     }
9 }
```

```
A.java:7: error: <identifier> expected
        System.out.println("Hello, world!");
                          ^
A.java:7: error: illegal start of type
        System.out.println("Hello, world!");
                           ^
A.java:9: error: class, interface, or enum expected
}
^
3 errors
```

(a) Invalid syntax sample code in Java, missing { at line 3 (b) corresponding error report

Figure 78: Misleading debug error report

Imaging a novice programmer saw this report, which even include professional jargon like *error: illegal start of type.* The programmer may too frustrated to give up programming.

What beginners expect is some reports more straight forward such as "There is a missing brace to match the other pair in line 3".
This paper[62] aims to find and fix **single token syntax errors**, which can be described as follows:

Given a source code with a syntax error, how to find exact location and provide a single token solution to the error.

It builds two models for the problem: n-gram model and long-short-term-memory neural network(LSTM). Both of them are trained on large set of Java source code corpus extracted from GitHub. The improvement of this paper is the test domain doesn't have to be the same as training domain compared to previous work which only limited to the same domain and programming assignment. The test data in this paper is from real syntax errors extracted from Blackbox repository.

- Model
  Building n-gram models for syntax error correction
  Error location detection:
  10-gram model breaks source code into 21 token long windows. Cross entropy loss is computed for each token in the window, the score of the middle token is the average entropy loss of those 21 windows. This setting ensures the context before and after the central word are considered. The token has highest score is believed to be the most likely error location.
  Error fixing:
  For top 10 score candidates, the model tries 3 strategies to fix the error: deletion, insertion and substitution. Deletion is easy to apply since it only has one option. For insertion and substitution, multiple tokens can be inserted or substituted. As a result, all tokens has frequency larger than 1000 are tried. For each possible solution, the cross entropy of the new 21 token long window is computed, the fix with lowest score is applied.

  Building LSTM for syntax error correction
  Before feeding data into LSTM network, all words are transfered into tokens, as described in next section, then each token is converted by one-hot encoding. Every file is encoded as a matrix, the columns are as many as tokens in a file, and rows are as many as vocabulary size. The model aims to find the syntax error location from the likelihood of adjacent tokens. If the likelihood is small, it means there is a big probability that there is a syntax error. To fix the error, all possibility of tokens given adjacent words need to be know.

  The sum of those possibilities is 1.0
  Two LSTM models is built, one is feedforward one is backward. For forward model, prefix is used as input. For backward models the input is suffix. The problem is, two models may return different results. For example, feed forward model may suggests *this* is the likely fix word, while backward declare {. with this method, we can detect text from different side. As training input, the model moving a sliding window over the tokens of each source code file. The window size is 20.

  Syntax error detection
  Two independent probability distribution are used to quantify whether the token is usual(likely) or unusual(unlikely). For a discrete distribution $p(x)$ and a distribution $\hat{q}(x)$, cross entropy loss is calculated as follows:
  $$H(p, \hat{q}) = -\sum_{x \in X} p(x) log_2 \hat{q}(x)$$
  if the value is 0, then it means $\hat{q}$ mimics p exactly. As a result, the more the value close to 0, the more natural the value is, the more likely the value is chosen. Here $p(x)$ equals 1 if x is the token at current position, 0 if it's not.
  The final unnaturalness degree is examined by sum of feed forward model and backward model:
  $$unnaturalness = H(p, \hat{q}_f) + H(p, \hat{q}_b)$$
  The position has highest unnaturalness is the most likely position of synta error.
  Error Fixing

For error fixing, guess and check strategy is used, which means every possible solution(insertion, deletion and substitution) is used to test whether it returns a syntactically-valid file. The possible location suggestions are given by the union of two set: $top_j(\hat{q}_f) \cup top_j(\hat{q}_b)$.

- Dataset
  The task can be divided into 2 steps: finding the error and fixing the error. For both steps, the likelihood of adjacent token given context should be computed:

  $$P(adjacent - token|context)$$

  In order to calculate probability , n-gram models and LSTM model should be trained, as shown in Figure below. The training data is extracted from variety Java corpus from GitHub, then tokenized, and finally the tokenized tokens can be used for adjacent word prediction.

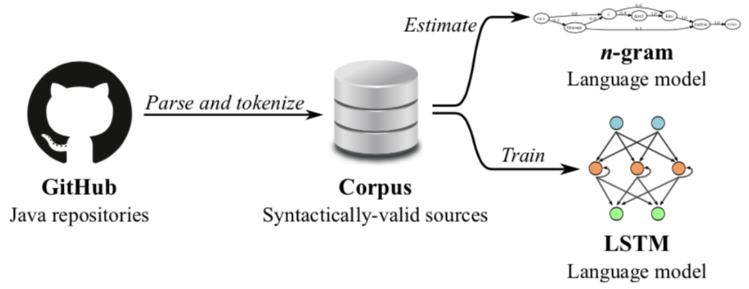
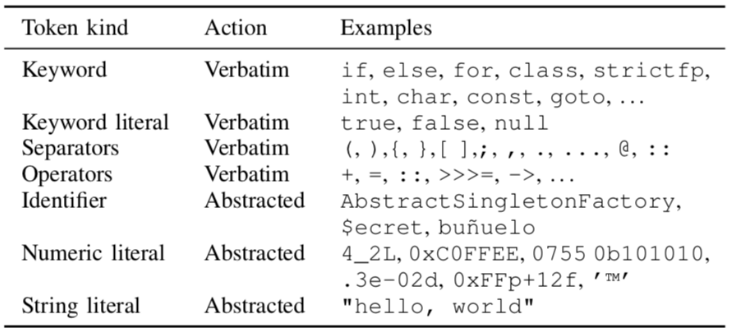


Figure 79: Methodology for training language models

As for training data, first Java source code are mined on GitHub, the authors first download 9993 Java repositories by stars successfully, then for each repository, they extracted every code file with suffix .java. Javac's scanner and parser implemented in OpenJDK are also be used to tokenize and parse every downloaded Java file. At the end, the authors tokenized 2,322,481 syntactical correct files out of 2,346,323 Java files. All source code, repository, data and repository licenses are stored in an SQLite3 database.
Secnodly, for tokenization, one vocabulary containing all possible unique tokens is maintained. However, since every new file may contain out-of-vocabulary tokens, the paper just deems those tokens as irrelevant. Certain tokens are abstracted to assign enough unique tokens for every words in vocabulary. Also, some tokens can not be abstracted but assign precise values and names. such as key word and operators. As shown in Figure below, abstracted tokens are different from file to file while verbatim tokens is shared among files.



Token kind	Action	Examples
Keyword	Verbatim	`if`, `else`, `for`, `class`, `strictfp`, `int`, `char`, `const`, `goto`, ...
Keyword literal	Verbatim	`true`, `false`, `null`
Separators	Verbatim	`(`, `)`,`{`, `}`,`[ ]`,`;`, `,`, `.`, `...`, `@`, `::`
Operators	Verbatim	`+`, `=`, `::`, `>>>=`, `->`, ...
Identifier	Abstracted	`AbstractSingletonFactory`, `$ecret`, `buñuelo`
Numeric literal	Abstracted	`4_2L`, `0xC0FFEE`, `0755` `0b101010`, `.3e-02d`, `0xFFp+12f`, `'™'`
String literal	Abstracted	`"hello, world"`

Figure 80: Token types and action types whether it's verbatim or abstracted.

As for testing data, it is extracted from Blackbox, which is a continually updated repository from BlueJ Java IDE– designed for Java programming beginners. The advantage using data from Blackbox is it contains wild and realistic data written by novices to test syntax error detection.
First step is to retrieve source code with both syntax error and fixed solutions. The authors

took all data collection up to July 1,2017, UTC and iterates through each pairs <former compilation, later compilation> and find those pairs whose former compilation failed and second compilation is succeeded. After that 1,715,312 data pairs are collected, among which 57.39% are single token syntax errors. Those problems can be solved by insertion, substitution or deletion, the distribution is illustrated by the table below:

Edit Operation	Instances	Percentage (%)
Insertion	223,948	22.75
Substitution	77,846	7.91
Deletion	682,677	69.34

Figure 81: Summary of single token syntax error types.

- Experiment
  First task is syntax error finding. The performance is measured by mean reciprocal rank(MRR):

  $$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{rank_q}$$

  Reciprocal rank is the inverse of the rank of the first correct location found in an ordered list of syntax error locations for a file q. Q is a set of total solutions attempted. MRR value is from 0 to 1, where 1 is the best score, meaning the first suggestion is the correct error location. Table below is the MRR value when finding the location of syntax error, in general guess, for a file has 100 lines the MRR value could be 0.002. But from the table, we can see the MRR score improved a lot.

Model	Qualification	1	2	3	4	5	All
10-gram Abstract	Location	.41	.42	.41	.40	.40	.41
	Valid Fix	.39	.39	.39	.38	.38	.39
	True Fix	.36	.36	.36	.35	.35	.36
10-gram Concrete	Location	.07	.07	.07	.08	.07	.07
	Valid Fix	.06	.06	.06	.07	.06	.06
	True Fix	.04	.04	.04	.04	.04	.04
LSTM 1	Location	.06	.05	.05	.05	.05	.05
(RMSProp)	Valid Fix	.06	.05	.05	.05	.05	.05
(no reshuffling)	True Fix	.05	.04	.04	.04	.04	.04
LSTM 2	Location	.52	.53	.53	.50	.50	.52
(Adam)	Valid Fix	.52	.53	.52	.49	.50	.51
(reshuffling)	True Fix	.46	.46	.46	.44	.44	.46

Figure 82: MRR score of using n-gram and LSTM models for syntax error location detection

  Second task is syntax error fixing. n all cases, there is a clustering of reciprocal ranks at 1.0, meaning that the 10- gram Abstract model tool can suggest the true fix as its first suggestion 30.22% of the time. The 10-gram Concrete model tool can suggest the true fix as its first suggestion 3.52% of the time. The LSTM 1 tool can suggest the true fix as its first suggestion 3.84% of the time.

4.3.2 Neural Code Comprehension: A Learnable Representation of Code Semantics

Tags:*Code semantic embedding, general domain, LSTM, n-gram, Transfer learning*

- Background
  With the explosion of code data, 1 billion commits written in 337 different languages are committed on GitHub in year 2017. As a result, how to understand the code turns to be an important task for many fields. This task is tough since each language has different

syntax(different operation orders), but remains semantically related(return the same output). In order to improve code comprehension, Natural language processing(NLP) concept is introduced, which is based on the hypothesis that software, like language, is also a form of communication. Software corpora is similar to language corpora. In NLP, the inputs are usually encoded to tokens. So for code comprehension task, the input(eg,. keywords or braces) should be processed into tokens such as one-hot embedding or word2vec models, which maps the input into lower dimentions. RNN model is trained on those tokens and achieves huge success in some domains such as summarization, function prediction and classification. However, the privious work has 2 constraints. First, they all work on single programming language and can not generate on future languages. Secondly, current methods can't deal with code with loop and targeting function.

The paper[63] aims to represent code semantics in a robust way. The pipeline is illustrated in below: First, the model accepts multiple source code languages, and uses LLVM Compiler Infrastructure to present Intermediate Representation(IR). Then the representation is processed to a robust representation called *conteXtual Flow Gtaphs*(XFGs), which supports loop and function calls. After that, the result is used to train individual statement embeddings called *inst2vec*, which is fed to RNNs for task specific models.

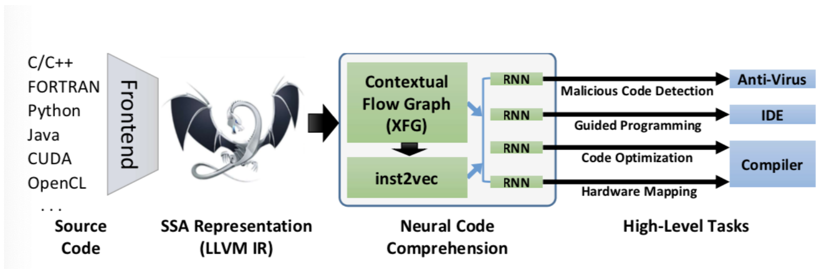
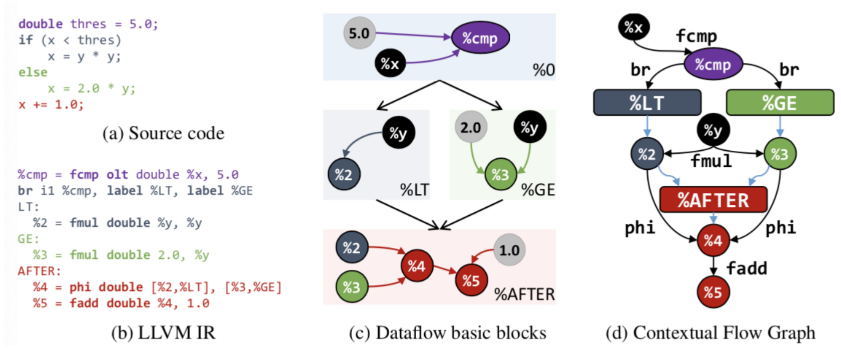


Figure 83: Overview of Neural Code Comprehension pipeline

- Model
  As illustrated from figure above, the Neural Code Comprehension can be divided into 4 parts: LLVM to dived IR statements into blocks, XFG construction to catch flow among blocks, inst2vec to embed statement in continuous space and last stage: train task specific RNN models. *Compilation–LLVM*



```
double thres = 5.0;
if (x < thres)
    x = y * y;
else
    x = 2.0 * y;
x += 1.0;
```

(a) Source code

```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y
AFTER:
  %4 = phi double [%2,%LT], [%3,%GE]
  %5 = fadd double %4, 1.0
```

(b) LLVM IR

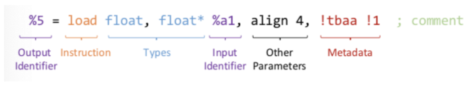
(c) Dataflow basic blocks

(d) Contextual Flow Graph

Figure 84: Contextual flow processing scheme.

Figure (a) and (b) illustrates original code and it's corresponding LLVM IR representation. The detail of LLVM IR statement is shown in figure (a). In LLVM, IR is given in Static Single Assignment(SSA) form. As shown in figure(b). This form ensures every variable is assigned only once, which makes it easier to track between IR statements. To represent control flor relatuibs sycg as loops, SSA defines Φ-expressions. The expression can list all

possible outcomes and optimize code among all branches. In figure(b), the identifier %4 is constructed from a Φ -expression that can take either the value of %2 or %3, depending on the value of x.
ConteXtual Flow Graph(XFG)
Only has LLVM IR is not enough since it can not capture dataflow from different blocks, as illustrated in figure(c). As a result, XFG is introduced. See Figure(d), the edge between blocks represents data dependences(black words).
The process of generating XFG Construction from LLVM IR is as follows: Firstly, read LLVM IR statements once, storing function names and return statements. Secondly, pass over the statements again, at the mean time adding nodes and edges according to the following rule-set: (a) Data dependencies within a basic block are connected.
(b) Inter-block dependencies (e.g., Φ-expressions) are both connected directly and through the label identifier (statement-less edges).
(c) Identifiers without a dataflow parent are connected to their root (label or program root).


```
%5 = load float, float* %a1, align 4, !tbaa !1  ; comment
```

(a) Anatomy of an LLVM IR statement.

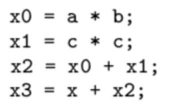

```
x0 = a * b;
x1 = c * c;
x2 = x0 + x1;
x3 = x + x2;
```
(b) SSA of x += (a*b)+(c*c)

Figure 85: Expressions for LLVM IR and SSA

inst2vec
With the help of XFGs, the embedding for each statements can be trained. To train embeddings, a skip-gram model is used. The data used here is covered in detail at dataset part, in this section we only mentioned the setup and training of the model. First, after given a set of XFGs which is extracted from LLVM IR files, neighboring statement pairs based on a fixed context size N is generated. The pairs appears less than 300 times is removed from dataset. The 200 dimension inst2vec model is trained for 5 epochs using Tensorflow and Adam optimizer.
Task specific models
Three different tasks are introduced to evaluate inst2vec. RNN models with same architecture is tuned for those 3 tasks. The input of RNN is inst2vec with a context size of 2, followed by 2 stacked LSTM layers with 200 neurons for each layer, batch normalization and ReLu activation functions, Adam optimiser are used. The Data used for those 3 tasks are introduced at next section.

- Dataset
  inst2vec training data
  preprocessing is implemented before transport XFGs data into inst2vec. Comments and metadata are discarded. Then it uses <INT/FLOAT/STRING> and %ID to replace immediate values(numeric constants, strings) and identifiers. Figure below shows the statement contraction.

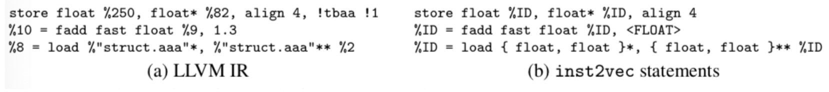
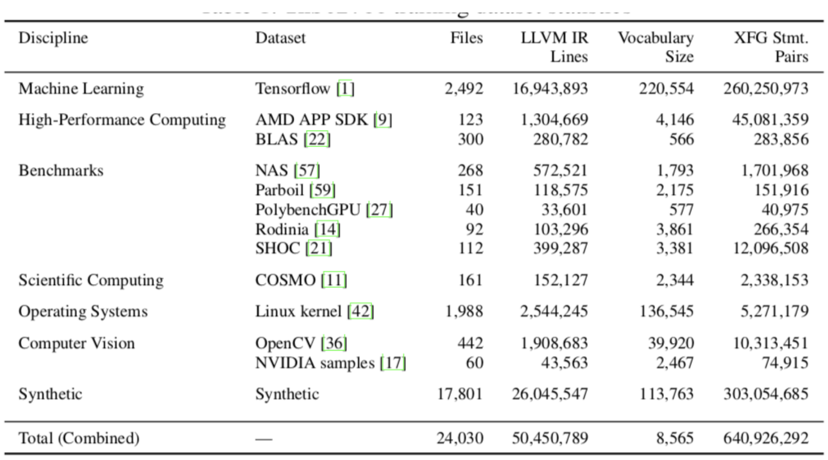

```
store float %250, float* %82, align 4, !tbaa !1
%10 = fadd fast float %9, 1.3
%8 = load %"struct.aaa"*, %"struct.aaa"** %2
```
(a) LLVM IR
```
store float %ID, float* %ID, align 4
%ID = fadd fast float %ID, <FLOAT>
%ID = load { float, float }*, { float, float }** %ID
```
(b) inst2vec statements

Figure 86: Before and after processing LLVM IR to inst2vec statements

Table below shows the code corpora and vocabulary statistics of the inst2vec dataset. The choice of corpora is bsed on their disciplines such as benchmarks, operating systems, computer vision and machine learning. The code in the dataset is written in C, C++, FORTRAN, and OpenCL, and is compiled for Intel CPUs as well as NVIDIA and AMD GPUs.

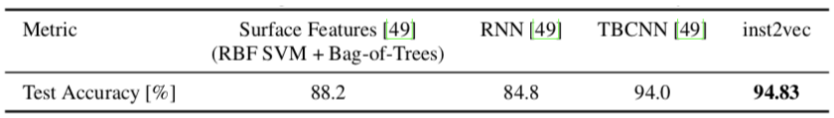


Discipline	Dataset	Files	LLVM IR Lines	Vocabulary Size	XFG Stmt. Pairs
Machine Learning	Tensorflow [1]	2,492	16,943,893	220,554	260,250,973
High-Performance Computing	AMD APP SDK [9]	123	1,304,669	4,146	45,081,359
	BLAS [22]	300	280,782	566	283,856
Benchmarks	NAS [57]	268	572,521	1,793	1,701,968
	Parboil [59]	151	118,575	2,175	151,916
	PolybenchGPU [27]	40	33,601	577	40,975
	Rodinia [14]	92	103,296	3,861	266,354
	SHOC [21]	112	399,287	3,381	12,096,508
Scientific Computing	COSMO [11]	161	152,127	2,344	2,338,153
Operating Systems	Linux kernel [42]	1,988	2,544,245	136,545	5,271,179
Computer Vision	OpenCV [36]	442	1,908,683	39,920	10,313,451
	NVIDIA samples [17]	60	43,563	2,467	74,915
Synthetic	Synthetic	17,801	26,045,547	113,763	303,054,685
Total (Combined)	—	24,030	50,450,789	8,565	640,926,292

Figure 87: inst2vec training dataset statistics

RNN training data
For algorithm classification task, the training data is from POJ-104 dataset, collected from Open Judfe System. It's a 500 people contribution dataset with 104 program classes. For compute device mapping and thread coarsening factor prediction task, OpenCL code dataset3 provided by Cummins et al. [18] is used.

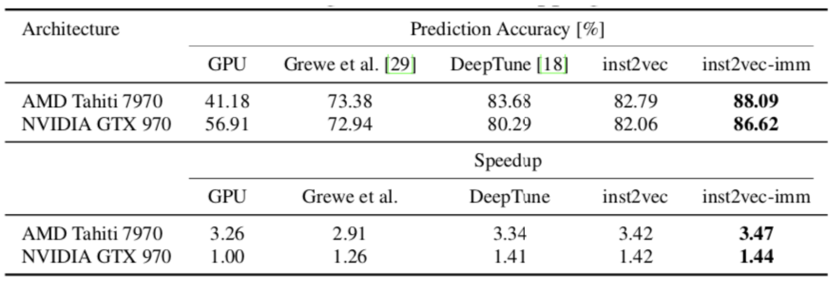
- Experiment
For Algorithm Classification task, the input of model is embedded source code and output is program class the model predicts. For comparison, several State of the art models are introduced. Tree-Based CNNs(TBCNN), which is the best classifier in POJ-104 dataset. We can see from Figure below that onst2vec outperforms TBCNN even though the dataset used during embedding generation doesn't contain POJ-104.



Metric	Surface Features [49] (RBF SVM + Bag-of-Trees)	RNN [49]	TBCNN [49]	inst2vec
Test Accuracy [%]	88.2	84.8	94.0	**94.83**

Figure 88: classification test accuracy

For Heterogeneous Compute Device Mapping, the task is to predict whether a given OpenCL program will run faster on CPU or on GPU given by its input data size, code and number of threads used in a group. Here XFGs and inst2vec inputs are concated with thread and data size, and trained by 10 fold cross-validation. The result table below shows the results of heterogeneous device mapping. Besides inst2vec and ins2vec-imm(with immediate value handling), feature extraction model proposed by Grew et al[64] and DeepTune[65] model are also introduced. We can see from the table that inst2vec performs better than Grewe et al. and is similar to DeepTune.



Architecture	Prediction Accuracy [%]				
	GPU	Grewe et al. [29]	DeepTune [18]	inst2vec	inst2vec-imm
AMD Tahiti 7970	41.18	73.38	83.68	82.79	**88.09**
NVIDIA GTX 970	56.91	72.94	80.29	82.06	**86.62**
	Speedup				
	GPU	Grewe et al.	DeepTune	inst2vec	inst2vec-imm
AMD Tahiti 7970	3.26	2.91	3.34	3.42	**3.47**
NVIDIA GTX 970	1.00	1.26	1.41	1.42	**1.44**

Figure 89: Heterogeneous device mapping results on CPU–AMD Tahiti 7970 and GPU NVDIA GTX 970.

For Optimal Thread Coarsening Factor Prediction task, it measures the amount of work per GPU thread does for a certain OpenCL source code. To find out how much inst2vec speed the threads up, manual features proposed by Magni et al[66] DeeoTune and DeepTune with transfer learning are mentioned. We can tell from table below that inst2vec proforms the best in half platforms. While ins2vec-imm's performance is similar to DeepTune-TL, it fails to outperform DeepTune-TL in AMD Tahiti 7970 and NVIDIA GTX 480. This may owe to the small size of the training data for this task.

Computing Platform	Magni et al. [46]	DeepTune [18]	DeepTune-TL [18]	inst2vec	inst2vec-imm
AMD Radeon HD 5900	1.21	1.10	1.17	**1.37**	1.28
AMD Tahiti 7970	1.01	1.05	**1.23**	1.10	1.18
NVIDIA GTX 480	0.86	1.10	**1.14**	1.07	1.11
NVIDIA Tesla K20c	0.94	0.99	0.93	**1.06**	1.00

Figure 90: Speedups achieved by coarsening threads. AMDs are CPU and NVIDIAs are GPU

4.3.3 Learning-based Recursive Aggregation of Abstract Syntax Trees for Code Clone Detection

Tags:*Code Clone Detection, Abstract Syntax Tree, general domain, Transfer Learning*

- Background
  Nowadays, online source code can be easily cloned from open-source code repositories, forums and app stores, code clones has made up a large part of software system. As a result, code clone detection becomes an important part of software examination. Besides, code similarity detection can also be used in bug detection, performance prediction and information retrieval in software contexts.
  Recently, many researches are studied for code clone detection, during which researchers find the embeddings of code tokens are in different forms, such as sequence, tree or other graph of discrete tokens. To represent for code inputs and their combinations, Abstract Syntax Trees(ASTs), identifiers, Control Flow Graphs and Bytecode are used.
  This paper[67] uses AST to learn from clone/non-clone data and generate code representations during which Siamese Network is used to share the weights from 2 RNN models, that aggregate the ASTs of two Java methods. It figures out that We find that the most important factor for a good model is a pre-trained embedding. We show how error scaling solves the class imbalance problem of supervised code clone detection.

- Model
  Siamese Network compares the results of 2 RNN models which are used to encode 2 code fragments separately. It aims to maximize the cosine similarity between clone codes and minimize that of non-clone pairs. As illustrated in the figure below, the Siamese network can be trained by forward and backward propagation. RNN models is used to process ASTs. The ASTs are constructed in a binary tree. Every node contains two parts: node type(eg., ForStatement, StringLiteral and InfixExpression) and node content(eg., "false","Hello World!" and "1024"). The method to construct the tree structure is covered at dataset-preprocessnig part.

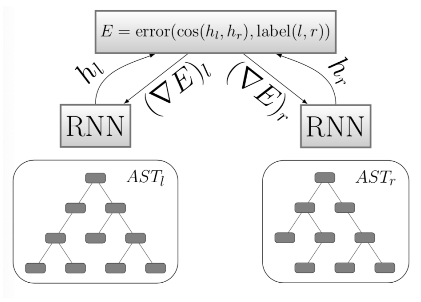


Figure 91: The Siamese network during training

During the training process, one LSTM architecture traverses through all the tree nodes. Each node combines the output of its children. Children nodes receives the hidden states h_l, h_r and cell statesc_l, c_r. $x_t and x_c$ are the vector representations. Those 6 vectors are the input of LSTM architecture. If some node are empty, just set the corresponding vector value as 0.

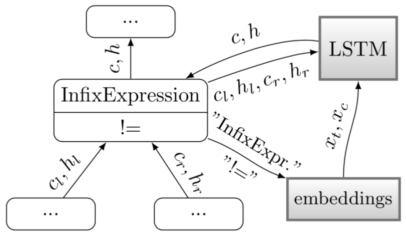


Figure 92: The Siamese network during training

- Dataset
  For supervised learning, labels should be given. To get ground truth data, famous BigCloneBench bench mark dataset [68] of method level for Java code clones is used.
  The raw Java source code is parsed to generate AST tree. Three modification is made during the generation. For example, given a phrase of source code called "Hello World":

  public static void main(String args[]){
  System.out.println("Hello, World!");
  }

  First, method name are cut out to guarantee the model won't depend on method name. Then, additional nodes are proposed to form equivalent binary tree. If a node has more than two children, the children are divided into half. We assign the bigger half to a new left child, and the others to another new right child node. This step is repeated for both children until the smaller set of children is less than three. As shown in figure below, the tree is deeper than before. Finally, the rare node contents are mapped to UNKNOWN tokens. as shown in red block. After AST tree is generated, the trees with depth deeper than 28 and nodes more than 1000 are eliminated. At last 227 candidates are cut off.

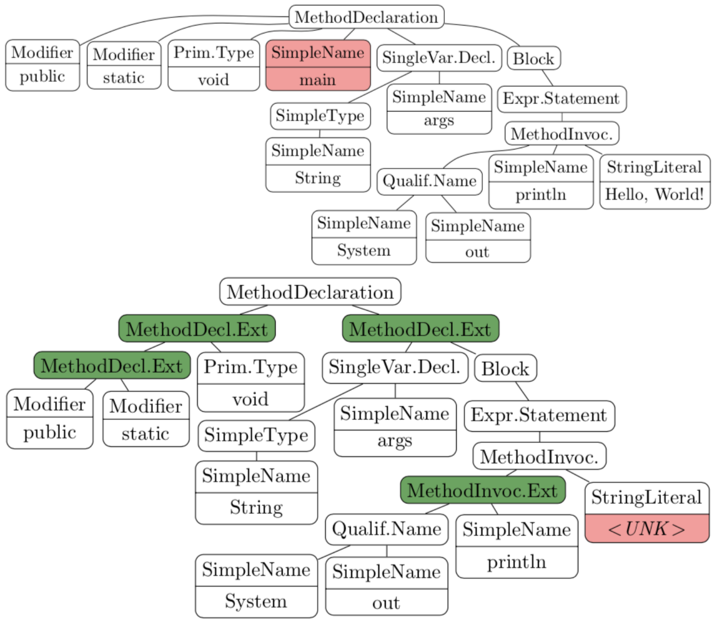


Figure 93: AST tree of Hello World code, before and after preprocessing

- Experiment
  For experiment, the paper test the function of pretraining embeddings. It compares the performance of the RvNN model with and without pretrained the embeddings of its AST node vectors. See the figure below, the average performance drops from 0.845 to only 0.746 AUC.
  We included also the baseline that uses the pretrained embeddings (”Emb. Φ (14D)”). It can surpass many of the non-optimally configured models. Especially, it outperforms the otherwise optimally configured model, that starts training with no pretrained embeddings. We can know from this comparison that a good pretrained embedding has a higher impact on performance than a complex model on top of it.[68]
  The case "with output gate" means run the model with an additional LSTM output gate. However, this may lead to parameters evpload, which causes overfitting.
  Margin means the extent of gap between two trees AST_l and AST_r. It is set to 0 by defalt. The first line indicates full scaling dataset, which means dividing the errors occurring for non-clone pairs by the ratio of number of non-clones to clones, works best, as seen in Table IV.

  The last row using unaware split performs much better than other cases, the reason is the model is trained on the same clusters as the evaluated data, while previous cases' training and testing data are from different clusters. As a result, evaluating supervised code clone detection using same clusters could lead to misleadingly good result.

	#params	**AUC$_0$**	**AUC$_1$**	**AUC$_2$**	**∅AUC**
$10\times$neg. scal.	201,494	0.744	0.817	0.912	**0.824**
margin $m = 0.9$	201,494	0.750	0.793	0.885	**0.810**
Emb. ∅ (14D)	10,694	0.784	0.752	0.885	**0.807**
margin $m = 0.5$	201,494	0.726	0.813	0.856	**0.798**
with output gate	249,194	0.700	0.763	0.822	**0.761**
no pretraining	201,494	0.668	0.807	0.763	**0.746**
*Unaware split**	*201,494*	*0.990**	*0.997**	*0.991**	***0.993****

Figure 94: THE PERFORMANCE FOR DIMENSIONALITY 150 W.R.T. OTHER CHANGES

4.3.4 Summarizing Source Code with Transferred API Knowledge

Tags:*API knowledge summarization, code summarization, general domain, GRU, Transfer Learning*

- Background
  Code summarization is a description on source code in natural language, it aims to help program comprehension and code search. Code Comments, as one of the most used code summarization method, can be really helpful when there are many people working on one project together and for further maintenance. However, high-quality code comments are in short in software industry. Comments are often out-dated, not available or unmatched. Besides, it is also very time-consuing to write a good code comment. Information Retrieval(IR) methods are used to generate summaries, however, it has 2 main constrains. First, it is hard for IR to locate accurate keywords if method name and identifier name are not accurate. Secondly, it relys on the similarity between code snippet. If there is no similar code snippet, it can not output accurate summaries.In the last few years, a lot of efforts are put into the research on using deep learning approaches to generate code summaries. Those experience showed that deep learning increases the effectiveness of code summarization. However Deep Learning still treat source code as a plain text, which omits some details such as identifier naming conventions and Application Programming Interface(API).
  API is important when summarize the code contents since it closely relates to code functions. Specific API sequence corresponds to specific features. For example, the following API *FileRead.new, BufferReader.new, BufferReader.read and BufferReader.close* are ususally included when realize function "read a file". As a result, this paper[69] is inspired by Transfer Learning, aims to use pre-trained API knowledge learned in a different but related task on code summarization. It proposes a novel method called TL-CodeSum, which generates a Java code description with the help of a transferred API knowledge from another

API summarization task. As a result, there are two main tasks for TL-Code Sum: API Summarization Task and Code Summarization Task.

- Model
  API Sequence Summarization Task
  This task aims to map API and natural language description. It uses basic Sequence to Sequence(Seq2Seq) model, which is widely used in Machine Translation, Text Summaization and etc. As shown in the left part in Figure95. It consists of two parts: encoder and decoder.

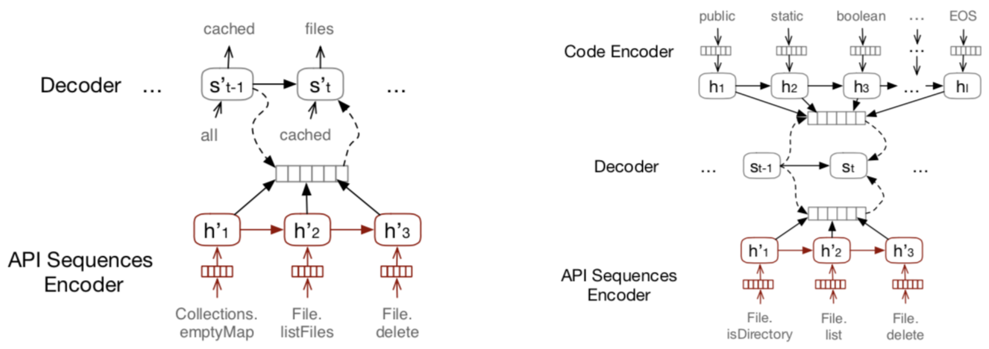


Figure 95: The architecture of TL-CodeSum (a)API Sequence Summarization (b)Code Summarization with Transferred API knowledge

  Since API innovation in Java $A'^{(i)} = [a'_1, ..., a'_m]$ has a sequence order, RNN structure is used in API encoder to read the API sequence one by one. Each API innovation sequence is encoded into a vector representing the API knowledge. To capture the features more accurately, attention mechanism is used during encoding. Decoder is another RNN which uses the vector to generate corresponding natural language description of the API innovation $D'^{(i)} = [d'_1, ..., d'_n]$. Both RNN are implemented as GRU[70], which need less parameters and widely used in RNN
  Code Summarization Task
  As shown in right half of Figure78, code summarization model is implemented in basic Seq2Seq model. Besides encoder and decoder, TL-CodeSum also adds another API encoder which contains API summarization transferred from previous model. TL-CodeSum model aims to generate summaries of source code with the help pf API knowledge obtained from API sequence summarization model.

- Dataset
  Two datasets are used in training. One for API summarization one for code summarization. Both of them are collected from GitHub, Java project from 2009 to 2014 are used to train API summarization, while Java projects from 2015 to 2016 is for code summarization task.To have a high quality performance, only projects have more than 20 stars are set as preliminary dataset. Those datasets are first parsed into AST trees, then Java methods, the API sequence contained in those methods and corresponding Javadoc comments are extracted. Source code is tokenized before they are fed to encoders and comments are used as code summaries. However, not all comments are useful. Empty or one-word descriptions are removed. The description of common methods such as getter, setter, constructor are also excluded.
  Finally, 340992 pairs of <API sequence, summary> for API summarization and 69708 pairs of <API sequence, code, summary> for code summarization task are obtained.

- Experiment
  Two metrics are used in this paper, IR metrics and Machine Translation(MT) metrics. For IR metrics, precision, recall and F-score are used. For MT metrics, BLEU score is used, which computs the n-gram precision of candidate sequence. METEOR focus on recall and evaluates the translation hypotheses by aligning the result to references and calculate

sentence-level similarities.
The baseline model is CODE-NN, which is a cutting edged code summarization method. It generates each word by a global attention mechanism. Besides CODE-NN, models generate summarization given API only and Code only are also introduced. API+Code model means it trained without transferring API knowledge. Moreover, to examine the benefits of API knowledge, fine-tuning the whole network(fine tune TL-CodeSum) and fixed API knowledge(fixed TL-CodeSum) are introduced.

Approaches	Precision	Recall	F-score
CODE-NN	26.21	14.17	18.40
API-Only	30.72	21.14	25.05
Code-Only	38.89	28.81	33.10
API+Code	41.06	30.34	34.90
TL-CodeSum(fixed)	**42.20**	34.38	37.89
TL-CodeSum(fine-tuned)	40.78	**35.41**	**37.91**

Figure 96: Precision, Recall and F-score for baseline models and proposed model

Figure above illustrates the result on IR metrics using different models. It shows Code only and API only models performs better than CODE-NN, which uses embedding of tokens directly. Combining source code with API knowledge improve the performance a lot. The precision decreases in fine tune models, however, the recall and F-score are both increased. Overall, TL-CodeSum surpasses all other approaches on generating code summaries.

As for MT metrics, we can see TL-CodeSum clearly outperforms the remaining models. Using API and Code only performs similar to Code-NN. Integrating API knowledge with source code largely imrpoves both BLEU score and METEOR score. Thoses models with API transferred knowledge surpass those don't by more than 12 percentage. The data are shown in figure below.

Approaches	BlEU-4 score	METEOR
CODE-NN	25.3	6.92
API-Only	26.45	10.71
Code-Only	35.50	14.78
API+Code	37.28	15.88
TL-CodeSum(fixed)	36.42	18.07
TL-CodeSum(fine-tuned)	**41.98**	**18.81**

Figure 97: BLEU and METEOR for baseline models and proposed model

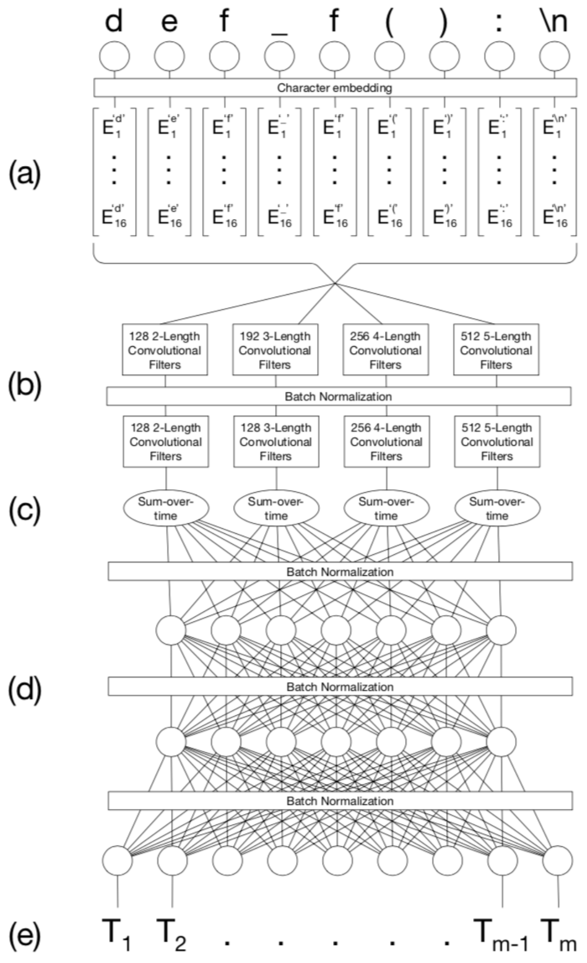
4.3.5 A Language-Agnostic Model for Semantic Source Code Labeling

Tags:*semantic labeling, multi-label classification, deep learning, CNN, general domain, Transfer Learning*

- Background
  Code reuse is helpful to improve coding efficiency, which makes it an essential part in software developing. However, with the exploding quantity of available source code online, how to generate accurate and meaningful semantic labels for source code snippets becomes a problem. Some efforts are put into the problem. For example, Santanu and Prakash[71] created pattern language to allow users to write generic codelike schematics. The drawback is that the schema fail to capture the general functionality of a complete program and scale poorly on large code corpora. Bajracharya et al. built a search engine names Sourcerer which extract features from corpus to improve keyword search function. The disadvantage is it suffers from custon language specific parser. Building a model that can work across all programming language, libraries and projects is difficult to achieve. As a result, this

paper[72] aims to propose a novel framework for source code labels generation on arbitrary language, length and domain. The paper first uses Stack Overflow as a source code extraction dataset, and it is also the first applies multi-classification which supports label realistic source code ducuments besides simple SO snippets.

- Model
  Convolutional neural networks(CNN) is chosen since it not only has less computation cost than LSTM but can simultaneously provide predictions for multiple labels.



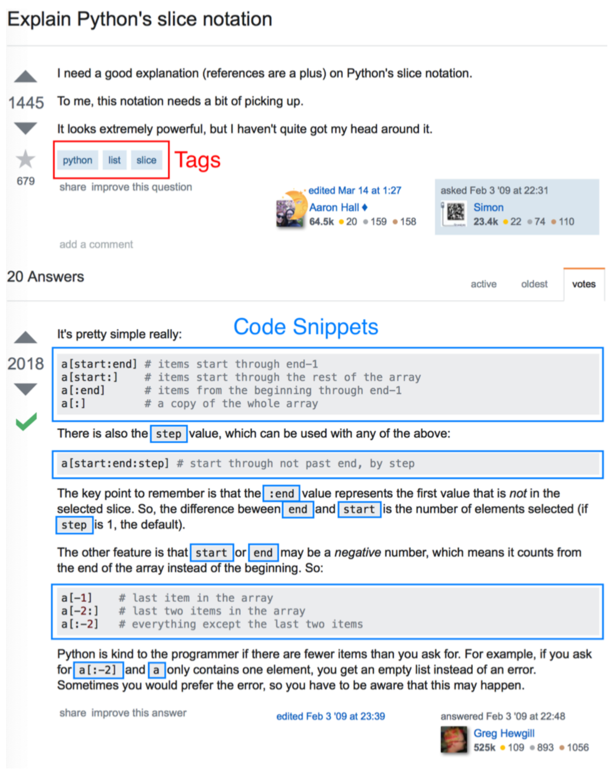
d e f _ f ( ) : \n
Character embedding
(a)
(b)
128 2-Length Convolutional Filters
192 3-Length Convolutional Filters
256 4-Length Convolutional Filters
512 5-Length Convolutional Filters
Batch Normalization
128 3-Length Convolutional Filters
(c)
Sum-over-time
(d)
(e)
T1 T2 . . . . . Tm-1 Tm

Figure 98: An overview of the neural network architecture. (a) The characters from original sentence are transformed into character embeddings (b Stacked convolutional filters in different lengths using ReLU as activation function are applied to character embeddings (c) We perform sum-over-time pooling on the output of each stacked convolution. (d) A flattened vector is fed into two fully-connected, batch-normalized, dense layers with ReLU activations. (e) Each output node uses a logistic activation function to produce a value from 0 to 1, representing the probability of a given label.

Figure above is the structure of the neural network architecture. First, each character is transformed into a 16-dimensional real valued vector. The reason why the paper choose embeddings on character level instead of word level is that character embeddings can predict source code without language limitation and the unique identifiers on word level is to massive to cover.
The embeddings are transformed by PCA to two dimensions vectors to enhance the intuition of character embedding. After that the embeddings are fed to convolutional layers to preserve information about words and sequences. Using sum over time pooling on the

stacked convolution matrix allows the model to obtain a fixed-length vector regardless of the initial input size.
After two bateh normalized layers, logistic activation is used on each neuron to get the probability of occurance of each tag. Binary cross entropy is used as loss function.

- Dataset
  The source code are extracted from Stack Overflow(SO), a ask and answer forum on computer programming. Users can ask questions, answe questions, attach code snippets and vote for the answers. The advantage of using SO is that it has a huge number of code snippets, wide set of tags and keeps updated by users about new technologies.


Explain Python's slice notation

1445

I need a good explanation (references are a plus) on Python's slice notation.

To me, this notation needs a bit of picking up.

It looks extremely powerful, but I haven't quite got my head around it.

python list slice

679

share improve this question

edited Mar 14 at 1:27
Aaron Hall ♦
64.5k ● 20 ● 159 ● 158

asked Feb 3 '09 at 22:31
Simon
23.4k ● 22 ● 74 ● 110

add a comment

20 Answers

active oldest votes

2018

It's pretty simple really:

```
a[start:end] # items start through end-1
a[start:]    # items start through the rest of the array
a[:end]      # items from the beginning through end-1
a[:]         # a copy of the whole array
```

There is also the `step` value, which can be used with any of the above:

```
a[start:end:step] # start through not past end, by step
```

The key point to remember is that the `:end` value represents the first value that is *not* in the selected slice. So, the difference beween `end` and `start` is the number of elements selected (if `step` is 1, the default).

The other feature is that `start` or `end` may be a *negative* number, which means it counts from the end of the array instead of the beginning. So:

```
a[-1]    # last item in the array
a[-2:]   # last two items in the array
a[:-2]   # everything except the last two items
```

Python is kind to the programmer if there are fewer items than you ask for. For example, if you ask for `a[:-2]` and `a` only contains one element, you get an empty list instead of an error. Sometimes you would prefer the error, so you have to be aware that this may happen.

share improve this answer

edited Feb 3 '09 at 23:39

answered Feb 3 '09 at 22:48
Greg Hewgill
525k ● 109 ● 893 ● 1056

Figure 99: Stack Overflow thread with a question and answer.

Figure above is a screen shoot of SO, question asker can attach maximum five tags. In this case, the asker add "python", "list" and "slice" as the tag. The question answerers can answer in both text and code snippets, which is boxed in blue. The snippets don't necessary to be a full function code, but oarticular functionality. However, several problems need to be sorted out: how to associate tages with snippets? how to filter out redundant and useless data?
For the first question, the paper assigns concates snippets in a single post separated by a new line characters and assign tags to each post.
The remaining problem is the snippets are not guaranteed to be useful. It can suffer from short snippets. Figure below is a statistics on the length summary on short snippets. We can see from the figure that many snippets have empty string or only a few characters long. This is bad because it barely contains any useful information. The solution proposed by the paper is to set a threshold length as 10. If the snippet length is greater or equal 10, then it is useful. They come to the conclusion based on massive snippets observation.

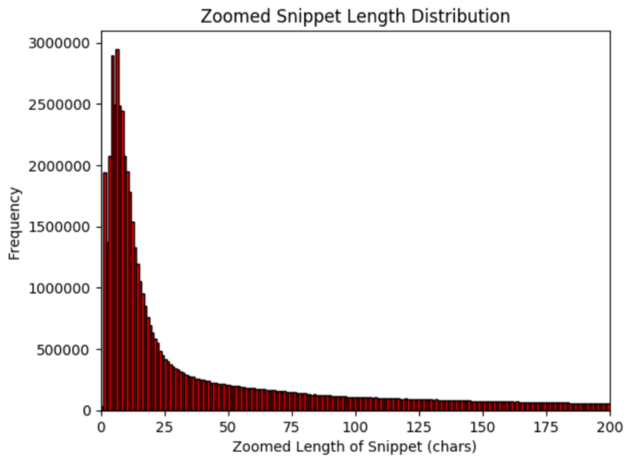
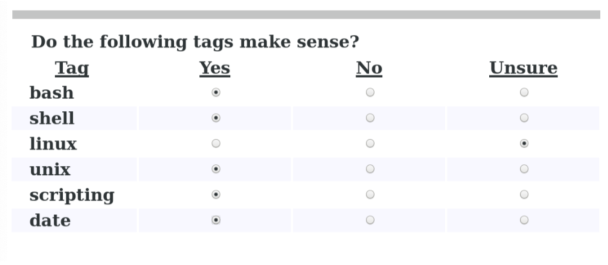


Figure 100: A zoomed view of the snippet length distribution, with 1 bin equal to 1 character. There are many strings that are empty or only a few characters long.

The snippet can also has "missing label phenomenon", which means the post is irrelevant to the question(judged by the vote score) or the question does not have a label. This is easy to tackle for machine learning models, since it can be treated as a negative example.
Third problem can be "rare tag problem", since some tags can be rarely used. In the paper, they cut off tags with fewer than 1000 positive samples.
Source Code Validation data is obtained from randomly samples from GitHub which never revealed in training step. However, the samples are lack of labels. In order to fix that, the paper uses human validation by offering a GUI to users and let them tag the samples. The labels labeled unsure are discarded. The GUI is shown in figure below.



Do the following tags make sense?

Tag	Yes	No	Unsure
bash	◉	○	○
shell	◉	○	○
linux	○	○	◉
unix	◉	○	○
scripting	◉	○	○
date	◉	○	○

Figure 101: The GUI for human validation of model outputs on source code documents.

- Experiment
  The main task of Source Code Validation is to test the ability of model trained on Stack Overflow in predicting on arbitrary source code. The figure below shows the model obtained 0.769 AUC. And the top 1 accuracy with human validation on source code is 86.6% accuracy. We note that this is better than the analogous performance on Stack Overflow, which indicates that, on source code, the model performs better for the first tag, but worse for the rest.

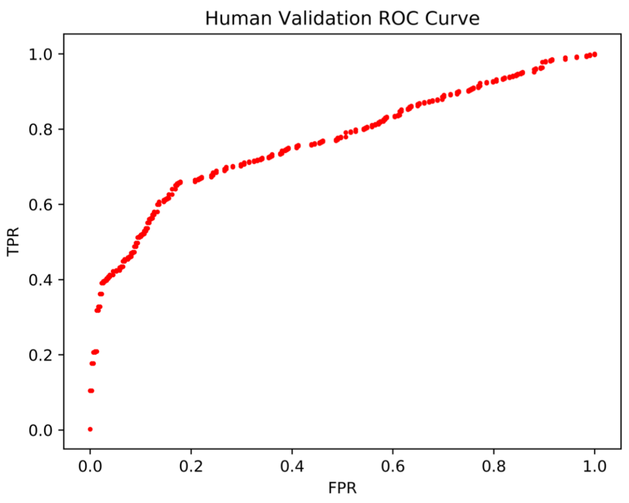


Figure 102: Human validation ROC curve with a 0.769 AUC. This differs from the Stack Overflow AUC values because it operates on the results of human validation, which is lim- ited to only a few tags per document.

4.3.6 Summary

For code modality, I only find papers using Transfer Learning. Given to that I assume using Multi-task Learning in code is still in green area, the reason might be it is hard to find different related tasks. The idea of method proposed in the paper using Transfer learning are similar. First they train word embeddings or additional knowledge embeddings on massive dataset, and then transfer that to an unseen data. As we can see from the results, transfer learning does improve the performance on code modality.

5 Conclusion

This research paper aims to offer a guideline for researchers to help them decide when to use Transfer Learning and when to use Multi-task Learning when dealing with data scarcity problem. It introduces 4 types of Language Models: Contextual free model, AutoEncoding model, RNN based model and Transformer based model, with 3 modalities using those models: Text, Protein and Source Code. As we can observe from their results, Transfer Learning and Multi-task Learning are widely used in Text, and both of them performs well. While in Protein, Transfer Learning is more popular than Multi-task Learning, and modern language models are not widely used in this type, which means more study should be worked on the field. As for Source Code, only application using Transfer Learning are found, and the model does not performs much better than Single Task Learning, with only 10 percentage enhancement.
To make a conclusion, when we have enough pre-trained data to generalize data pattern, we could use Transfer Learning, when we can easily come up with some related tasks so that they can benefit from each other to improve their performance, we can try out Multi-task Learning.
During my investigation, I found it is hard to make comparison between TL and MTL since they are not applied to the same task using the sane dataset and matrix. I hope in the future, researchers could focus more on comparing TL and MTL in one task using the same dataset. By horizon comparison, we can have a clearer idea of which method suits for which tasks.

References

[1] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010.

[2] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[3] Yu Zhang and Qiang Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2017.

[4] Transfer learning - machine learning's next frontier, author = Sebastian Ruder. `https://ruder.io/transfer-learning/`. Written on March,21, 2017.

[5] Transfer learning from pre-trained models, author = Pedro Marcelino. `https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751`. Written on October,23, 2018.

[6] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[7] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[8] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[9] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[11] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[12] David A Jurgens, Peter D Turney, Saif M Mohammad, and Keith J Holyoak. Semeval-2012 task 2: Measuring degrees of relational similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 356–364. Association for Computational Linguistics, 2012.

[13] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[14] Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems*, pages 801–809, 2011.

[15] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[16] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*, 2019.

[17] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305, 2017.

[18] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173*, 2017.

[19] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[20] Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*, 2017.

[21] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. *arXiv preprint arXiv:1902.00751*, 2019.

[22] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[24] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[25] Huggingface. Transformer on github. `https://github.com/huggingface/transformers`. Written by huggingface.

[26] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization, 2016.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[28] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[30] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[31] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[32] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

[33] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[34] Tom Kwiatkowski, Sharon Goldwater, Luke Zettlemoyer, and Mark Steedman. A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 234–244. Association for Computational Linguistics, 2012.

[35] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

[36] kimiyoung. Transformerxl on github. `https://github.com/kimiyoung/transformer-xl`. Written by kimiyoung.

[37] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[38] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[39] Siva Reddy, Danqi Chen, and Christopher D Manning. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019.

[40] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830*, 2016.

[41] Yu Cao, Meng Fang, Baosheng Yu, and Joey Tianyi Zhou. Unsupervised domain adaptation on reading comprehension, 2019.

[42] Siddhant Garg, Thuy Vu, and Alessandro Moschitti. Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection, 2019.

[43] Zhanhong Jiang and Young M. Lee. Deep transfer learning for thermal dynamics modeling in smart buildings, 2019.

[44] Aditya Khandelwal and Suraj Sawant. Negbert: A transfer learning approach for negation detection and scope resolution, 2019.

[45] Jeremy Barnes, Erik Velldal, and Lilja Øvrelid. Improving sentiment analysis with multi-task learning of negation, 2019.

[46] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[47] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[48] Michael Heinzinger, Ahmed Elnaggar, Yu Wang, Christian Dallago, Dmitrii Nachaev, Florian Matthes, and Burkhard Rost. Modeling the language of life-deep learning protein sequences. *bioRxiv*, page 614313, 2019.

[49] Michael Schantz Klausen, Martin Closter Jespersen, Henrik Nielsen, Kamilla Kjaergaard Jensen, Vanessa Isabell Jurtz, Casper Kaae Soenderby, Morten Otto Alexander Sommer, Ole Winther, Morten Nielsen, Bent Petersen, et al. Netsurfp-2.0: Improved prediction of protein structural features by integrated deep learning. *Proteins: Structure, Function, and Bioinformatics*, 87(6):520–527, 2019.

[50] Charles Yanofsky, Bruce C Carlton, John R Guest, Don R Helinski, and Ulf Henning. On the colinearity of gene structure and protein structure. *Proceedings of the National Academy of Sciences of the United States of America*, 51(2):266, 1964.

[51] Ulrike Göbel, Chris Sander, Reinhard Schneider, and Alfonso Valencia. Correlated mutations and residue contacts in proteins. *Proteins: Structure, Function, and Bioinformatics*, 18(4):309–317, 1994.

[52] DANIÈLE Altschuh, AM Lesk, AC Bloomer, and A Klug. Correlation of co-ordinated amino acid substitutions with function in viruses related to tobacco mosaic virus. *Journal of molecular biology*, 193(4):693–707, 1987.

[53] Alexander Rives, Siddharth Goyal, Joshua Meier, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, page 622803, 2019.

[54] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Xi Chen, John Canny, Pieter Abbeel, and Yun S Song. Evaluating protein transfer learning with tape. *arXiv preprint arXiv:1906.08230*, 2019.

[55] Sara El-Gebali, Jaina Mistry, Alex Bateman, Sean R Eddy, Aurélien Luciani, Simon C Potter, Matloob Qureshi, Lorna J Richardson, Gustavo A Salazar, Alfredo Smart, et al. The pfam protein families database in 2019. *Nucleic acids research*, 47(D1):D427–D432, 2018.

[56] Jie Hou, Badri Adhikari, and Jianlin Cheng. Deepsf: deep convolutional neural network for mapping protein sequences to folds. *Bioinformatics*, 34(8):1295–1303, 2017.

[57] Karen S Sarkisyan, Dmitry A Bolotin, Margarita V Meer, Dinara R Usmanova, Alexander S Mishin, George V Sharonov, Dmitry N Ivankov, Nina G Bozhanova, Mikhail S Baranov, Onuralp Soylemez, et al. Local fitness landscape of the green fluorescent protein. *Nature*, 533(7603):397, 2016.

[58] Gabriel J Rocklin, Tamuka M Chidyausiku, Inna Goreshnik, Alex Ford, Scott Houliston, Alexander Lemak, Lauren Carter, Rashmi Ravichandran, Vikram K Mulligan, Aaron Chevalier, et al. Global analysis of protein folding using massively parallel design, synthesis, and testing. *Science*, 357(6347):168–175, 2017.

[59] Nils Strodthoff, Patrick Wagner, Markus Wenzel, and Wojciech Samek. Udsmprot: Universal deep sequence models for protein classification. *bioRxiv*, page 704874, 2019.

[60] UniProt Consortium et al. Uniprot: the universal protein knowledgebase. *Nucleic acids research*, 46(5):2699, 2018.

[61] Rui Fa, Domenico Cozzetto, Cen Wan, and David T Jones. Predicting human protein function with multi-task deep neural networks. *PloS one*, 13(6):e0198216, 2018.

[62] Eddie Antonio Santos, Joshua Charles Campbell, Dhvani Patel, Abram Hindle, and José Nelson Amaral. Syntax and sensibility: Using language models to detect and correct syntax errors. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 311–322. IEEE, 2018.

[63] Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. Neural code comprehension: a learnable representation of code semantics. In *Advances in Neural Information Processing Systems*, pages 3585–3597, 2018.

[64] Michael FP O'Boyle, Zheng Wang, and Dominik Grewe. Portable mapping of data parallel programs to opencl for heterogeneous systems. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 1–10. IEEE Computer Society, 2013.

[65] Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. End-to-end deep learning of optimization heuristics. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 219–232. IEEE, 2017.

[66] Alberto Magni, Christophe Dubach, and Michael O'Boyle. Automatic optimization of thread-coarsening for graphics processors. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 455–466. ACM, 2014.

[67] Lutz Büch and Artur Andrzejak. Learning-based recursive aggregation of abstract syntax trees for code clone detection. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 95–104. IEEE, 2019.

[68] Jeffrey Svajlenko and Chanchal K Roy. Evaluating clone detection tools with bigclonebench. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 131–140. IEEE, 2015.

[69] Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. Summarizing source code with transferred api knowledge. 2018.

[70] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[71] Santanu Paul and Atul Prakash. A framework for source code search using program patterns. *IEEE Transactions on Software Engineering*, 20(6):463–475, 1994.

[72] Ben Gelman, Bryan Hoyle, Jessica Moore, Joshua Saxe, and David Slater. A language-agnostic model for semantic source code labeling. In *Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis*, pages 36–44. ACM, 2018.