



---

# Final Report

---

**Hosein Mohebbi**

Iran University of Science and Technology  
hosein\_mohebbi@comp.iust.ac.ir  
98722531

**Sara Rajaei**

Iran University of Science and Technology  
sara\_rajaee@comp.iust.ac.ir  
98722071

## Abstract

In this report, we have a deep looking at transfer learning, and one of its most common applications (pre-trained models) in natural language processing. By applying Transformers that benefits the Attention mechanism in the large pre-trained models, a significant improvement appeared in almost all NLP tasks. BERT is one of these pre-trained models that can be used as a feature extractor or by fine-tuning. Fine-tuning is one of the transfer learning approaches applied to many downstream tasks in NLP. It is done by adding a classifier layer on the top of large pre-trained models which is parameter-inefficient in many tasks. To improve model performance and reduce parameters during fine-tuning, the Adapter proposed. Adapters by adding a few trainable parameters per task while original model parameters are frozen cause high parameter sharing and achieve near the state-of-the-art performance score. To demonstrate its effectiveness, we re-implement and test it on 21 diverse tasks, including GLUE benchmark. Also, we go a further step and propose an even less parameter method by sharing adapter modules among three consecutive layers. Our method gives almost the same results on different tasks but has much fewer parameters.

## 1 Introduction

Transfer learning is a 'design methodology' within machine learning. The general idea of transfer learning is to use knowledge learned from tasks, that have a lot of available labeled data, in settings with a few available labeled data. Thus, instead of starting the learning process from scratch (randomly initialized), we start from patterns that have been learned to solve a different task. By the advent of large Transformer-based models such as BERT and RoBERTa [Devlin et al. \(2018\)](#); [Liu et al. \(2019\)](#), transfer learning has become popular in NLP. Figure 1 describe a taxonomy for transfer learning in NLP [Pan and Yang \(2009\)](#); [Ruder \(2019\)](#).

Sequential transfer learning is defined as a setting where source and target tasks are different, and training is performed in sequence. That means models are not optimized jointly like in multi-task learning, but each task is learned separately. In this project, we mainly focus on this approach of transfer learning to explore and compare predominant methodologies, such as feature extractions, fine-tuning, and adapters. Finally, we show how adapters benefit from sequential learning and outperform other approaches in NLP.

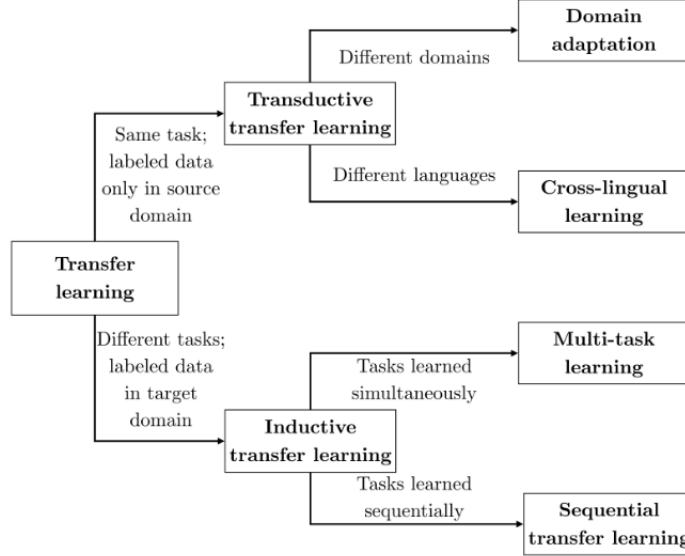


Figure 1: A taxonomy for transfer learning for NLP

## 2 Attention Is All You Need

Transformers introduced by Vaswani et al. (2017) make a significant improvement in almost all downstream tasks in NLP. The main idea behind the Transformer is attention and withhold recurrence to create global dependencies between inputs and outputs. A transformer consists of an encoder and a decoder. Each encoder is a stack of layers, in which, there are two sublayers. The first one is a multi-head self-attention unit, and the second one is a fully connected feed-forward network. The structure of the decoder in a transformer is the same as encoder except it has an additional sublayer. The extra sublayer performs multi-head attention over the output of the encoder stack. Figure 2 shows more details about Transformer architecture.

In calculating self-attention, we need to define a Query vector, Key vector, and a Value vector. These vectors come from the multiplication of the word embedding and a specific learned matrix for each of them. The result computed based on an attention function. The function is a mapping from a query and a set of key-value pairs to an output. The output computed as Formula 1.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

The paper benefits a new approach called Multi-head attention. In this approach instead of a single attention function, they apply h functions with different learned linear projection. It enables the model to jointly attend to information from different representation subspaces at different positions.

## 3 BERT

BERT(Bidirectional Encoder Representations from Transformers) Devlin et al. (2018) is a pre-trained model that has made a revolution in NLP fields. The model trained on English Wikipedia(2500M words) and the BooksCorpus (800M words) with two objective loss functions. Masked Language Modeling (masking 15% of all WordPiece tokens in each sequence at random, then predict them) and Next Sentence Prediction. The pre-trained procedure on such a massive datasets makes The model powerful to understand different layers of language knowledge and getting outstanding results in different language tasks.

BRET consists of some Encoder layers stacked (which the paper calls Transformer Blocks). The paper proposed BERT in two sizes, base and large. BERT-base has L =12 Transformer layers, each transformer has H=768 hidden state size and A= 12 attention heads. (total parameters = 110M). In

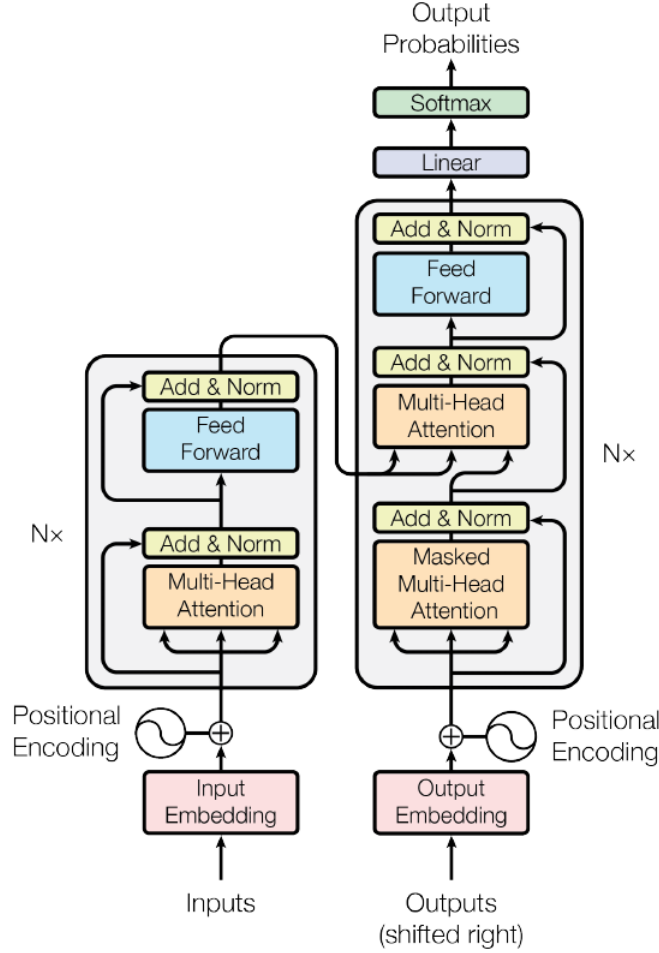


Figure 2: The Transformer model architecture

the BERT-large L, H, and A change to 24, 1024, and 16, respectively.(total parameters = 340M). BERT can be used in many downstream tasks straightforward by adding a classifier layer on top of it. Figure 3 shows BERT architecture for applying in some tasks.

The first token in BERT’s input sequence always is [CLS] token. It usually is used in a classification task as a representation of the whole sequence. There is also a token to define end of the sentence called [SEP]. In a pair-sentence task, we have two [SEP] tokens, one for each sentence. Aside from token embeddings, BERT also has a segment embedding and positional embedding in input sequences. Segment ID (token type ID) is a binary mask to identify different sequences in the model. In this way, the first sequence tokens is represented by 0, whereas the second one by 1. The positional ID is used by the model to identify every token’s position. Contrary to RNNs that have the position of each token embedded within them, transformers are unaware of the position of each token. The position ID is created for this purpose. The input embeddings are the sum of token embeddings, segmenta- tion embeddings and positional embeddings. Figure 4 provides more details.

#### 4 To Tune or Not to Tune?

Peters et al. (2019) focused on two common paradigms for adaptation in transfer learning mentioned before, feature extractions, and fine-tuning a pre-trained model. Their empirical results across diverse NLP tasks with two state-of-the-art models (ELMo and BERT) show that the relative performance of fine-tuning vs. feature extraction depends on the similarity of pre-training and target tasks. They

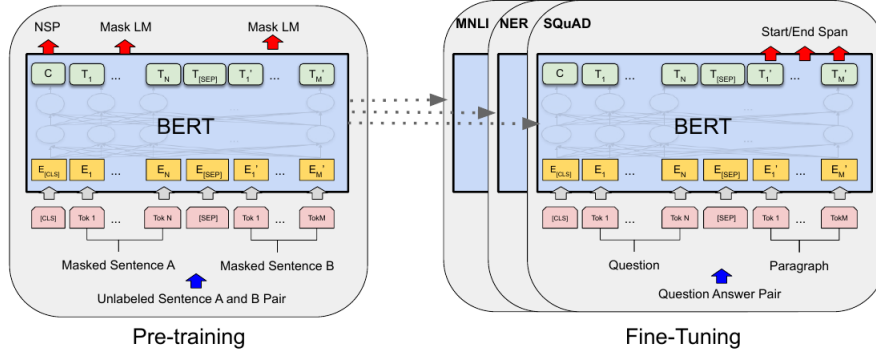


Figure 3: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned.

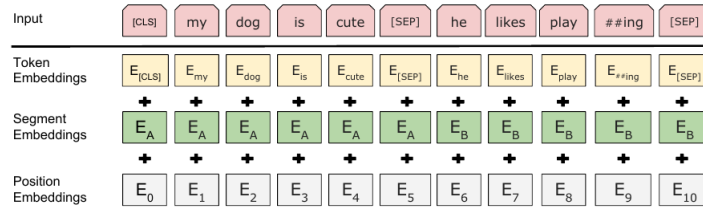


Figure 4: BERT input representation

explored possible explanations and provided practical recommendations for adapting pre-trained representations to NLP practitioners in Figure 5.

**Feature Extraction** For both ELMo and BERT, they extracted contextual representations of input sequences from all layers. During adaptation, they trained a linear weighted combination of layers which was used as input features to a task-specific model. They employed a bi-attentive classification network [McCann et al. \(2017\)](#) for Sentiment Analysis (SA) tasks, ESIM model [Chen et al. \(2016\)](#) for sentence pair tasks, and BiLSTM with a CRF layer [Lafferty et al. \(2001\)](#); [Lample et al. \(2016\)](#) for Name Entity Recognition (NER) tasks as a task-specific model.

**Fine-tuning: BERT** They fed the sentence representation into a softmax layer for text classification and sentence pair tasks. For NER, they extracted the first word piece representation for each token and added a softmax layer.

Conditions			Guidelines
Pretrain	Adapt.	Task	
Any	❄️	Any	Add many task parameters
Any	🔥	Any	Add minimal task parameters ⚠️ Hyper-parameters
Any	Any	Seq. / clas.	❄️ and 🔥 have similar performance
ELMo	Any	Sent. pair	use ❄️
BERT	Any	Sent. pair	use 🔥

Figure 5: Paper’s guidelines for using feature extraction and fine-tuning. Seq.: sequence labeling. Clas.: classification. Sent. pair: sentence pair tasks.










Pretraining	Adaptation	NER	SA	Nat. lang. inference		Semantic textual similarity		
		CoNLL 2003	SST-2	MNLI	SICK-E	SICK-R	MRPC	STS-B
Skip-thoughts		-	81.8	62.9	-	86.6	75.8	71.8
ELMo		91.7	<b>91.8</b>	<b>79.6</b>	<b>86.3</b>	<b>86.1</b>	<b>76.0</b>	<b>75.9</b>
		<b>91.9</b>	91.2	76.4	83.3	83.3	74.7	75.5
	$\Delta = $  	0.2	-0.6	-3.2	-3.3	-2.8	-1.3	-0.4
BERT-base		92.2	93.0	<b>84.6</b>	84.8	86.4	78.1	82.9
		<b>92.4</b>	<b>93.5</b>	<b>84.6</b>	<b>85.8</b>	<b>88.7</b>	<b>84.8</b>	<b>87.1</b>
	$\Delta = $  	0.2	0.5	0.0	1.0	2.3	6.7	4.2

Figure 6: The test set performance of feature extraction and fine-tuning approaches for ELMo and BERT-base compared Skip-thoughts [Kiros et al. \(2015\)](#). Settings that are good for fine-tuning are colored in red; settings good for feature extraction are colored in blue.

#### 4.1 Analyses

Figure 6 shows their results comparing ELMo and BERT for both feature extraction and fine-tuning approaches across seven tasks. For BERT, fine-tuning outperforms feature extraction.

**Modeling pairwise interactions** LSTMs consider each token sequentially, while Transformers can relate each token to every other in each layer. They showed this can facilitate fine-tuning with Transformers on sentence pair tasks, on which ELMo feature extraction performs comparatively poorly.

**Impact of additional parameters** They found that additional parameters on the top of the network have a key role for feature extraction, but hurt performance with fine-tuning.

**Impact of target domain** They observed no significant correlation. At least for MNLI task, the distance from source and target domains does not seem to have a major impact on the adaptation performance.

**Representations at different layers** They found to knowledge for single sentence classification tasks seems mostly concentrated in the last layers, while pair-sentence classification tasks gradually build up information in the intermediate and last layers of the model.

## 5 Parameter-Efficient Transfer Learning for NLP

[Houlsby et al. \(2019\)](#) introduced adapters as an alternative approach for adaptation in transfer learning in NLP within deep transformer-based architectures. Adapters are task-specific neural modules that are added between layers of a pre-trained network. After coping weights from a pre-trained network, pre-trained weights will be frozen, and only Adapters will be trained.

### 5.1 Why Adapters?

Adapters provide numerous benefits over plain fully fine-tuning or other approaches that result in compact models such as multi-task learning:

- It is a lightweight alternative to fully fine-tuning that trains only a few trainable parameters per task without sacrificing performance.
- Yielding a high degree of parameter sharing between down-stream tasks due to being frozen of original network parameters.
- Unlike multi-task learning that requires simultaneous access to all tasks, it allows training on down-stream tasks sequentially. Thus, adding new tasks do not require complete joint retraining. Further, eliminates the hassle of weighing losses or balancing training set sizes.
- Training adapters for each task separately, leading to that the model not forgetting how to perform previous tasks (the problem of catastrophic forgetting).

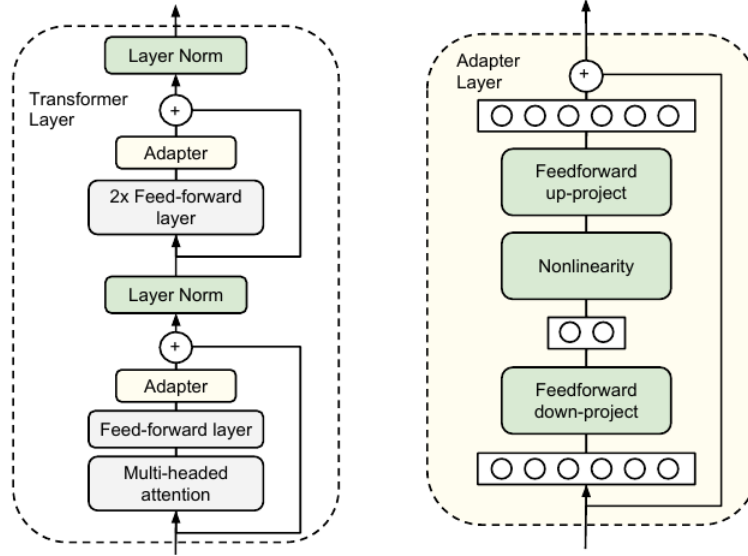


Figure 7: Architecture of the adapter module and its integration with the Transformer. During adapter tuning, the green layers are trained on the downstream data, this includes the adapter, the layer normalization parameters, and the final classification layer.

## 5.2 Adapter Architecture

Houlsby et al. (2019) performed an ablation study to find the best adapter architecture in terms of size and position in the transformer layer and found the following strategy in Figure 7 is more beneficial across many datasets. This strategy add a shared adapter module twice to each Transformer layer: after a projection following multi-headed attention and after two feed-forward layers. then, The output of adapter layer is passed to a new shared normalization layer.

To limit a number of parameters, they proposed a bottle-neck architecture for adapters, as shown in Figure 7. first, Adapters project original  $d$ -dimensional features into a smaller dimension,  $m$ , apply a nonlinearity, then project back to  $d$ -dimensions. The total number of parameters added per layer, including biases, is  $2md+d+m$ . By setting  $m \ll d$ , we can limit the number of parameters added per task. an Adapter module itself has a skip-connection internally. Apart from layers in an adapter module, normalization layer parameters are also trained per task.

## 6 Implementation

Our contributions are summarized as follows:

- Re-implementation of the approach introduced in Houlsby et al. (2019) and validating their results on different datasets.
- We propose and implemente two extensions to Houlsby’s architecture to investigate the impact of parameter sharing of adapters across BERT model.
- Code and pre-trained adapters for designing new experiments would be available on [github.com/hosein-m/TF-Adapter-BERT](https://github.com/hosein-m/TF-Adapter-BERT).

We implemente Adapter module using both popular frameworks: TensorFlow 2.0 (TF) and PyTorch (PT). In TF version, we create an Adapter Module from scratch and instantiate "TFBertLayer", "TFBertAttention", "TFBertSelfOutput" and "TFBertOutput" classes of HuggingFace by injecting our Adapters to them. In PT version, we used AdapterHub framework Pfeiffer et al. (2020) that is a library built on the top of HuggingFace. In all implementations, the pre-trained BERT (base, uncased) is derived from HuggingFace transformer library Wolf et al. (2019) as the original model. to perform classification or regression task we add fully connected layers to the pre-trained pooler on the top of the first token representation(called [CLS] token)in the last layer.

Model	Adapter sharing strategy	Trainable part of BERT	# Classifiers	# All trainable parameters without last classifier
Houlsby	Shared in each layer	-	1	1,189,632
Ours (not shared)	Not shared	Encoder’s layer norms	2	3,006,720
Ours (3-shared)	Shared in three consecutive layers	-	1	396,544

Table 1: The differences between these three models.

## 6.1 Training details

Our training details follow Houlsby et al. (2019). We optimized using Adam, that it’s learning rate is increased linearly over the first 10% of steps, and then decayed linearly to zero. Following the paper, we set max learning rate  $3e-4$  and the number of epochs 10 for GLUE benchmark Hendrycks and Gimpel (2016). Also, for additional text classification datasets, introduced in the paper, we set max learning rate  $1e-4$  and the number of epochs 50. All runs is trained on a GPU of Google Colab. The best model is selected according to the validation set score.

## 6.2 Models

First, we re-implement the approach introduced in Houlsby et al. (2019) and reproduce their results on GLUE benchmark Wang et al. (2019). In this method, following the approach proposed in the paper, we fix entire weights of BERT model and share adapter modules in each layer. As we mentioned before, adapters first project original  $d$ -dimensional features into a smaller dimension,  $m$ , apply a nonlinearity, then project back to  $d$  dimensions. The original dimension of features is 768. Following the paper, we set bottleneck size to 64, and use GELU Hendrycks and Gimpel (2016) activation function as a nonlinearity. We have two adapter modules in each layer but, by sharing them in each layer, the total number of parameters added per layer, including biases, is  $2md + d + m$ .

As a first extension to Houlsby’s architecture (further referred to as not shared), we make three changes in the model. First, we unfreeze norms layer of encoders in BERT model and allow them to fine-tune during training adapters. Second, two adapters in each layer are not shared. Finally, we add two classifier layers on the top of the pooler with Tanh activation function between them. To do so, the total number of parameters added per layer, including biases, is  $4md + 2d + 2m$ .

As a final extension (further referred to as 3-shared), we follow Houlsby’s architecture, but instead of sharing adapters in each layer, we share them in all three consecutive layers. The intuition comes from the literature where several studies have proven that different layers of BERT encode different types of knowledge like surface, syntactic, and semantic information Jawahar et al. (2019). Additionally, due to multiplying outputs with input embedding weights in MLM objective of BERT during pre-training to restoring high representations to the vocabulary representation, the last three layers of BERT play nearly the same role in the model.

Table 6.2 summarized differences between these three models.

## 6.3 Results

For GLUE tasks, we evaluate all three models and reporte test metrics provided by the submission website <sup>1</sup> in Table 6.3. Besides, to further validation that adapters yield compact, performant, models, we test one of our proposed models on additional text classification datasets contained diverse number of classes and training examples. As we have shown in Table 6.3 our proposed model significantly outperforms fully fine-tuning and Houlsby’s architecture.

## 6.4 Analysis

Following the paper Houlsby et al. (2019), we performed an ablation to determine which adapters are influential. For this, we remove some trained adapters and re-evaluate the model (without re-training) on the validation set. Figure 8 shows changes in the performance when removing adapters from all possible continuous layer spans. The experiment is performed on BERT(base, uncased) with adapter size of 64 on CoLA. First, we observe that removing any single layer’s adapters has only a small impact on performance (elements on the heatmaps’ diagonals). In contrast, when all the adapters are

<sup>1</sup><https://gluebenchmark.com/>

Model	CoLA	SST-2	MRPC	STS-B	QQP	MNLI(m)	MNLI(mm)	QNLI	RTE	Total
BERT(base, uncased)(full fine-tuning)	52.1	93.5	88.9	87.1	71.2	84.6	83.4	90.5	66.4	79.7
Houlsby (our implementation)	55.2	90.9	88.7	82.5	71.4	84.1	83.3	90.6	68.2	79.4
Ours (not shared)	55.6	93.4	87.8	83.9	71.4	84	83.1	90.7	66	79.2
Ours (3-shared)	53.5	92.6	88.6	84.6	70.9	83.5	82.9	90.3	67.6	79.3

Table 2: Results on GLUE test sets scored using GLUE evaluation server. MRPC and QQP are evaluated using F1 score. STS-B is evaluated using Spearman’s correlation coefficient. CoLA is evaluated using Matthew’s Correlation. The other tasks are evaluated using accuracy.

Dataset	BERT-base (full fine-tuning)	Ours (not shared)
Crowdfower airline	83.6	83.3
Crowdfower corporate messaging	92.5	92.2
Crowdfower disasters	85.3	84
Crowdfower economic news relevance	82.1	84.6
Crowdfower emotion	38.4	41.6
Crowdfower political audience	80.9	78.5
Crowdfower political bias	75.2	76.5
Crowdfower political message	38.9	79.3
Crowdfower primary emotions	36.9	35.1
Crowdfower progressive opinion	71.6	81.4
News aggregator dataset	96.3	92.2
Crowdfower US economic performance	75.3	71
Average	71.4	<b>74.9</b>

Table 3: Test accuracy for additional classification tasks

remove from the network, the performance drops substantially: This indicates that although each adapter has a small influence on the overall network, the overall effect is large.

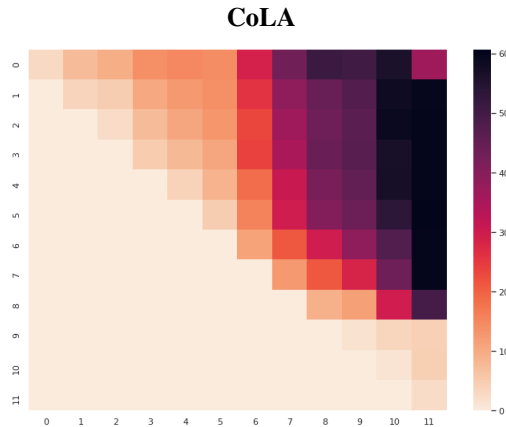


Figure 8: Ablation of trained adapters from continuous layer spans. The heatmap shows the relative decrease in validation accuracy to the fully trained adapted model. The y and x axes indicates the first and last layers ablated (inclusive), respectively. The diagonal cells, indicate ablation of a single layer’s adapters. The cell in the top-right indicates the ablation of all the adapters. Cells in the lower triangle are meaningless, and are set to 0%.



## References

- Chen, Q., Zhu, X., Ling, Z., Wei, S., Jiang, H., and Inkpen, D. (2016). Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.
- Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. *arXiv preprint arXiv:1902.00751*.
- Jawahar, G., Sagot, B., and Seddah, D. (2019). What does bert learn about the structure of language?
- Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.
- Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.
- Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.
- McCann, B., Bradbury, J., Xiong, C., and Socher, R. (2017). Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305.
- Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.
- Peters, M. E., Ruder, S., and Smith, N. A. (2019). To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987*.
- Pfeiffer, J., Rücklé, A., Poth, C., Kamath, A., Vulić, I., Ruder, S., Cho, K., and Gurevych, I. (2020). Adapterhub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*.
- Ruder, S. (2019). *Neural transfer learning for natural language processing*. PhD thesis, NUI Galway.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.
- Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In the Proceedings of ICLR.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2019). Huggingface’s transformers: State-of-the-art natural language processing. *ArXiv*, pages arXiv–1910.