HIERARCHICAL MULTITASK LEARNING FOR LANGUAGE MODELING WITH TRANSFORMERS

by

Çağla Aksoy

B.S., Computer Engineering, Boğaziçi University, 2017

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2020

ACKNOWLEDGEMENTS

I want to thank Tunga Güngör for his guidance during my graduate studies. I would also like to thank Fatma Başak Aydemir and Emin Erkan Korkmaz for accepting to be in my jury.

I thank Alper Ahmetoğlu for his help, patience, and supportive mind-opening discussions. It is a pleasure to research together in this field. I feel lucky to have him with me in all circumstances.

I am grateful to my mother for raising me as a confident woman with a love of learning. I especially thank my brother for being a role model for me with his sense of responsibility and honesty. Also, I thank my father and my Ate for their cordial supports every time.

I thank my colleagues, especially Zeynep Furtun, for providing me the flexibility to balance work and research.

The numerical calculations reported in this thesis were partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources) and TETAM servers.

ABSTRACT

HIERARCHICAL MULTITASK LEARNING FOR LANGUAGE MODELING WITH TRANSFORMERS

Recent works show that learning contextualized embeddings for words is beneficial for natural language processing (NLP) tasks. Bidirectional Encoder Representations from Transformers (BERT) is one successful example of this approach. It learns embeddings by solving two tasks, which are masked language model (masked LM) and the next sentence prediction (NSP). This procedure is known as pre-training. The pre-training of BERT can also be framed as a multitask learning problem. In this thesis, we adopt hierarchical multitask learning approaches for BERT pre-training. Pre-training tasks are solved at different layers instead of the last layer, and information from the NSP task is transferred to the masked LM task. Also, we propose a new pre-training task, bigram shift, to encode word order information. To evaluate the effectiveness of our proposed models, we choose two downstream tasks, one of which requires sentence-level embeddings (textual entailment), and the other requires contextualized embeddings of words (question answering). Due to computational restrictions, we use the downstream task data instead of a large dataset for the pre-training to see the performance of proposed models when given a restricted dataset. We test their performance on several probing tasks to analyze learned embeddings. Our results show that imposing a task hierarchy in pre-training improves the performance of embeddings.

ÖZET

HİYERARŞİK ÇOKLU GÖREV ÖĞRENİMİ YAKLAŞIMI İLE DÖNÜŞTÜRÜCÜLERDE DİL MODELLEME

Son çalışmalar kelimelerin bağlamsal gömmelerini kullanmanın alt görevler için faydalı olduğunu göstermiştir. Bu yaklaşımın başarılı bir örneği Dönüştürücülerden Çift yönlü Gizyazar Gösterimi'dir (DÇGG). DÇGG bağlamsal gömmeleri maskelenmiş dil modeli (maskelenmiş DM) ve sonraki cümle tahmini (SCT) olan iki görevi birlikte çözerek öğrenir. Bu işlem ön eğitim olarak adlandırılır. DÇGG'nin ön eğitimi aynı zamanda çoklu görev öğrenimi olarak da tasarlanabilir. Bu tezde, DÇGG'nin ön eğitimi için hiyerarşik çoklu görev öğrenimi yaklaşımları uygulanmıştır. Ön eğitim görevleri son katman yerine farklı katmanlarda çözülür ve SCT görevindeki bilgiler maskelenmiş DM görevine aktarılır. Ayrıca, iki-gram yerini değiştirme görevi ek bir ön eğitim görevi olarak kelimelerin dizilimine ait bilgileri kodlamak için önerilmiştir. Oluşturulan gömmeleri test etmek için iki farklı alt görev seçilmiştir. Bunlardan biri cümle düzeyinde gömmeler gerektiren metinsel gerektime problemidir. Diğeri ise kelime düzeyinde bağlamsal gömme gerektiren soru cevaplama problemidir. Hesaplama kısıtlamaları nedeniyle, önerilen modellerin ön eğitimi büyük veri seti yerine alt görev verileri kullanılarak yapılmıştır. Öğrenilen gömmeleri analiz etmek ve yorumlamak için tasarlanan çeşitli irdeleme problemlerinde bu gömmelerin performansları incelenmiştir. Sonuçlar, ön eğitimde görev hiyerarşisi uygulanmasının gömmelerin performansını arttırdığını göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

C_i	Count of ith word
c_m	Masked LM classifier
c_n	NSP classifier
c_s	Sentence-level classifier
c_t	Token-level classifier
d_k	Dimensionality of word vectors in transformer
e	Encoder stack
e_1	First (lower) encoder stack
e_2	Second encoder stack
h_i	Last layer activation ith word for GPT and BERT
h_{ij}	Hidden state of ith layer of jth word in BiLSTM
K	Key matrix
N_i	Number of documents where ith word occurs
p_m	WordPiece probabilities
p_n	Next sentence probabilities
p_s	Sentence classes probabilities
p_t	Token classes probabilities
Q	Query matrix
T	Threshold
V	Value matrix
x_i	Input word embedding of ith word
λ_i	Mixture parameter for ith component
$\overrightarrow{h}_{ij}$	Hidden state of ith layer of jth word in left-to-right LSTM
$\overleftarrow{h}_{ij}$	Hidden state of ith layer of jth word in right-to-left LSTM
`[CLS]`	Start token of a segment
`[MASK]`	Mask token

`[SEP]`	End token of a segment

LIST OF ACRONYMS/ABBREVIATIONS

ALBERT	A Lite Bidirectional Encoder Representations from Transformers
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short Term Memory
BoW	Bag-of-words
BS	Bigram Shift
CBOW	Continuous Bag-of-words
CI	Coordination Inversion
CR	Coreference Resolution
ELMo	Embeddings from Language Models
EM	Exact Match
EMD	Entity Mention Detection
F1	F-measure
FN	False Negatives
FP	False Positives
GPT	Generative Pre-training
IDF	Inverse Document Frequency
LM	Language Model
LSTM	Long Short Term Memory
MLP	Multi-layer Perceptron
MTL	Multitask Learning
MultiNLI	Multi-Genre Natural Language Inference
NER	Named Entity Recognition
NLP	Natural Language Processing
NSP	Next Sentence Prediction
OM	Semantic Odd Man Out
ON	Object Number
POS	Part-of-speech

RE	Relation Extraction
RNN	Recurrent Neural Networks
RoBERTa	Robustly Optimized Bidirectional Encoder Representations from Transformers Approach
SL	Sentence Length
SN	Subject Number
SOP	Sentence Order Prediction
SQuAD	The Stanford Question Answering Dataset
T	Tense
TC	Top Constituents
TD	Tree Depth
TF	Term Frequency
TP	True Positives
WC	Word Content

1. INTRODUCTION

Contextualized embeddings of words are successful in various natural language processing (NLP) problems. These are dynamic embeddings; they can be changed depending on the words and their context. These are obtained by pre-training deep architectures to solve different language model objectives. The knowledge encoded in these pre-trained models can be transferred to the various problems. Bidirectional Encoder Representations from Transformers (BERT) is one of the most successful models which create contextualized embeddings. It learns embeddings by optimizing two language model objectives at the same time. In this thesis, we framed BERT pre-training as a multitask learning problem therefore we propose some modifications to the original structure of the BERT. The multitask learning approach constructs one model for multiple objectives with a set of parameters that satisfy all objectives.

Natural language is an abstract representation of thoughts and objects which have similar meaning through a community. We use raw sensory information to represent images. However, this is not possible for texts, as they have no direct physical interpretation. Instead, we map texts to vector representations to indicate their differences and similarities.

Bag-of-words (BoW) is a straightforward method to create vectors for documents. It establishes a fixed-size vector for each text based on the usage of words in these texts. A vector has the same length as the number of words in the vocabulary, and each position in the vector represents a particular word. There are different methods to calculate values in these vectors: boolean values indicating whether the word is used in the text or not, the number of word occurrences in the text, relative frequencies with all the words in the text. Some words are used a lot in English texts such as “a”, “an”, “the”, and “of”; however, domain-specific terms are rarely used. When all words have the same importance, these non-informative words might create noise in the vector representation. As a more informative frequency calculation, term frequency-

inverse document frequency (TF-IDF) metric is used. TF of ith word in a document is calculated as follows:

$$\mathrm{TF}(i) = \frac{C_i}{\sum_j C_j} \tag{1.1}$$

where C_i is the count of ith word in the document, and j iterates over all words in the vocabulary. IDF of ith word is calculated as follows:

$$\mathrm{IDF}(i) = \log \frac{N}{N_i} \tag{1.2}$$

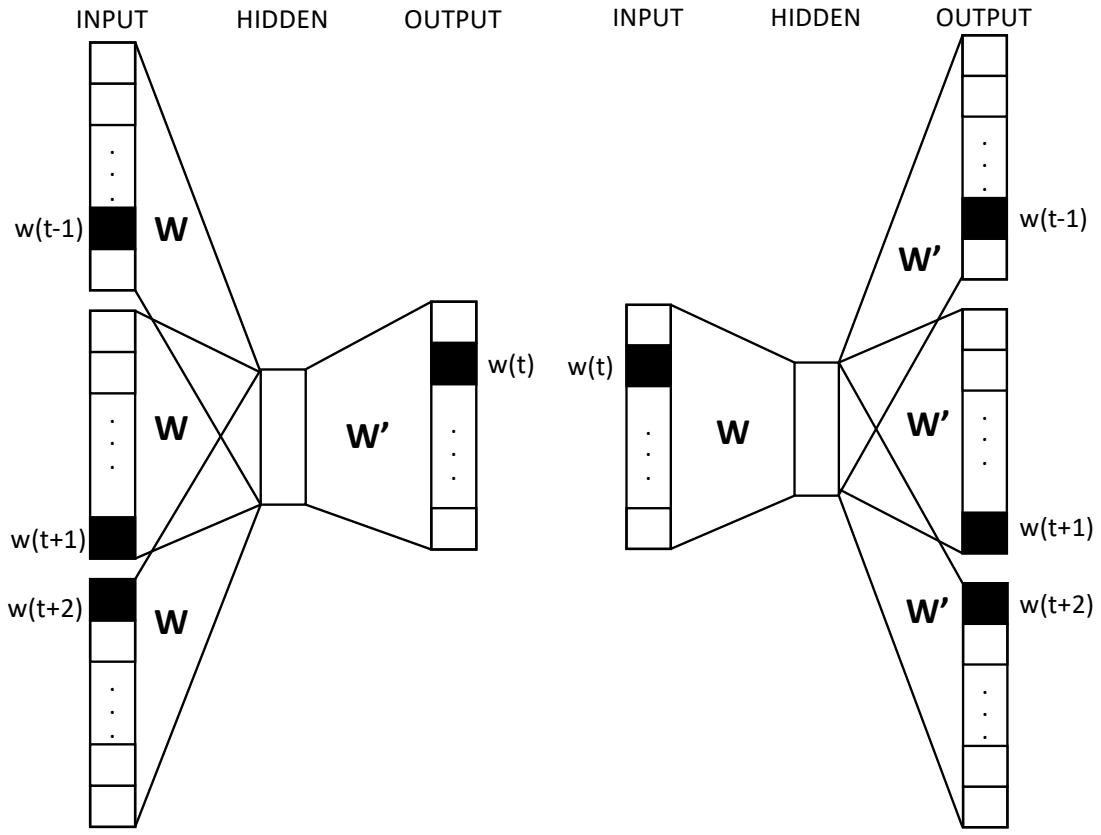
where N is the number of documents, N_i is the count of documents where ith word occurs. Therefore, rare words have higher IDF scores than common words. Then, TF-IDF for ith word in a document is computed as follows:

$$\text{TF-IDF}(i) = \mathrm{TF}(i) * \mathrm{IDF}(i) \tag{1.3}$$

In this function, IDF score is responsible for reducing the effect of frequently used words and increase the impact of rare words.

BoW is easy to implement, but it has many drawbacks. It only focuses on which words are used in a text; their order is ignored. It assumes that documents are similar when they have similar content. However, sequences can have different meanings when they are arranged in different orders. For example, “apple is red” and “is apple red” have the same BoW vector representations; however, the former is a fact, and the latter is a question. To compensate for the lack of word order information, n-grams can be used instead of using uni-gram. In uni-gram, each unique token in the corpus is counted as a single token while in n-gram, n consecutive tokens are treated as a single token. As n increases, the vocabulary size increases exponentially due to the number of token combinations. This approach only handles the order information of n-words while increasing the vector size. Because of the large vocabulary size, documents are represented by sparse vectors that inhibit training generalized classifiers.

Also, BoW creates vector representations for documents, not for words. Instead of creating representations for documents, one can also create representations for words and then combine these representations. There have been many studies in natural language processing (NLP) to find suitable word representations (embeddings) that carry information of a language. Even if finding these word representations can be computationally demanding, this can be advantageous since it is computed only once. These learned representations can be used for various downstream tasks such as sentiment classification and machine translation.



(a) CBOW (b) Skip-Gram

Figure 1.1. Word2Vec model architectures.

Word2Vec [1, 2] is one of the first successful examples of creating representations for words. Word2Vec finds word embeddings by predicting a word given its neighbor-

hood (Continuous Bag of Words (CBOW) Figure 1.1a) or predicting its neighborhood given the word (Skip-gram Figure 1.1b) for a given corpus. For both models, the input and the output are one-hot encoded vectors with vocabulary size. The output layer is a softmax layer to get probabilities. CBOW model is trained to maximize the following:

$$\sum_{-c \leq j \leq c, j \neq t} \log p(w_t | w_{t+j}) \tag{1.4}$$

where c is the window size of the context with center word w_t. Skip-Gram model is trained to maximize the following:

$$\sum_{-c \leq j \leq c, j \neq t} \log p(w_{t+j} | w_t) \tag{1.5}$$

These models are pre-trained with large scale data. After pre-training, hidden layer representations of words are used as the embeddings of these words for various tasks. Words that are used together have similar word embeddings due to the training strategy. For example, the vectors of "king" and "queen" have high cosine similarity while the vectors of "king" and "internet" have low cosine similarity. Furthermore, these vectors support algebraic operations, such as addition and subtraction. One famous example is that the resulting vector of the operation "king" - "man" + "woman" is most similar to the vector of "queen". Here, we remove "man" property from "king", then add "woman" property. As expected, this transformation leads to somewhere near "queen" vector. These functionalities are not possible with BoW, as it cannot create distributed representations. However, Word2Vec embeddings do not contain word order information and contextual information. Words can have different meanings depending on their contexts. For example, the word apple should have different representations for "apple versus windows" and "apple versus banana"; Word2Vec creates only one representation for "apple".

Contextualized embeddings are proposed to mitigate this problem. They are obtained by pre-training deep architectures to solve different language model objectives. These pre-trained architectures, which can create dynamic embeddings depending on

the words and their context, are used for various tasks. ELMo (Embeddings from Language Models) [3] uses bidirectional long-short term memory (BiLSTM) [4] to predict a word given its context. Since BiLSTM is used for creating embeddings, both left-to-right and right-to-left contexts are implicitly encoded. With the invention of Transformer [5], researchers began to shift from BiLSTM-based methods to transformer-based methods. The transformer is shown to be more appropriate for training in large datasets due to its self-attention mechanism [6]. OpenAI GPT (Generative Pre-training) [7] has the same objective as ELMo in the forward direction, except it uses transformer architecture. BERT (Bidirectional Encoder Representations from Transformers) [8] also uses transformer architecture with bidirectional pre-training tasks. Training objectives affect the information encoded in embeddings. Each objective and architecture presumes a different inductive bias.

In this thesis, we focused on BERT as it uses multiple training objectives, which are the next sentence prediction (NSP) and the masked language model (masked LM). These objectives can create an inhibitory effect or a regulatory effect on each other. For this reason, we applied a hierarchical multitask learning approach to BERT by modifying its original structure. Our motivation is to create embeddings that encode the information from each task in a balanced way. Our contributions are as follows:

- Instead of training masked LM and NSP classifiers with the last layer embeddings, we trained masked LM classifiers with embeddings from lower layers of the transformer (Lower Mask). We do the same experiment for the NSP classifier as well (Lower NSP). By evaluating the performance of the embeddings on downstream tasks, we provide insights about the hierarchy between pre-training tasks.
- We incorporate the input or the output of the NSP classifier to the input of the masked LM classifier in order to enrich the sentence-level embedding.
- We propose a new pre-training objective, bigram shift, in addition to masked LM and NSP tasks to enforce embeddings also to learn word order information.

Our experimental results show that Lower NSP has a competitive performance when compared with the original BERT structure. We also evaluate the learned embeddings on probing tasks to provide useful insights into training strategies. Results on probing task experiments show that using bigram shift task for pre-training is useful for specific tasks.

The remaining part of this thesis is organized as follows. In Chapter 2, we mention preliminaries about contextual embeddings and multitask learning. In Section 3, we explain our methods in detail. In Chapter 4, we report our experiment results. Lastly, we give a conclusion in Chapter 5.

2. PRELIMINARIES

2.1. Deep Contextualized Embeddings

Understanding the usage of words is essential to solve language-related problems. The pre-trained language models (LM) aim to learn general information about the language. It has been shown that transferring the knowledge encoded in these models to the various tasks is useful [3, 7, 8]. These LMs are pre-trained with large-scale data with different deep architectures and different objectives. Then, output or hidden representations of these deep architectures are used as embeddings of words. These embeddings can be easily fine-tuned on various downstream tasks with high performance. As these models take words with their context as input, they can generate dynamic embeddings depending on the context. Therefore, there is no global vector for a given word with these models. This section explains three common models that are shown to be successful in many tasks: ELMo, OpenAI GPT, and BERT.

2.1.1. Embeddings from Language Models

ELMo is pre-trained with the next word prediction task in a sequence with a vast amount of data. Due to the nature of the task, preparing the data is not difficult; we can use plain texts without any label. This task forces the model to learn the most possible and meaningful sequences and word order rules of a language. ELMo is based on BiLSTM. LSTM is the special type of recurrent neural network (RNN) that can cope with long term dependencies. It has a chain-like structure to take sequential input. Its cell state carries sequential information for long steps, and its gating mechanism prevents vanishing or exploding gradient problem. BiLSTM (Figure 2.1) processes the sequence from both left-to-right and right-to-left direction, then combines them. In pre-training, $\overrightarrow{h_{Lt}}$ is used with a classifier to predict the $(t+1)$th word in the sequence where L is the last layer index. The same is done in the reverse direction to predict the previous word; $\overleftarrow{h_{L(t+1)}}$ is used with a classifier to predict the (t)th word. With

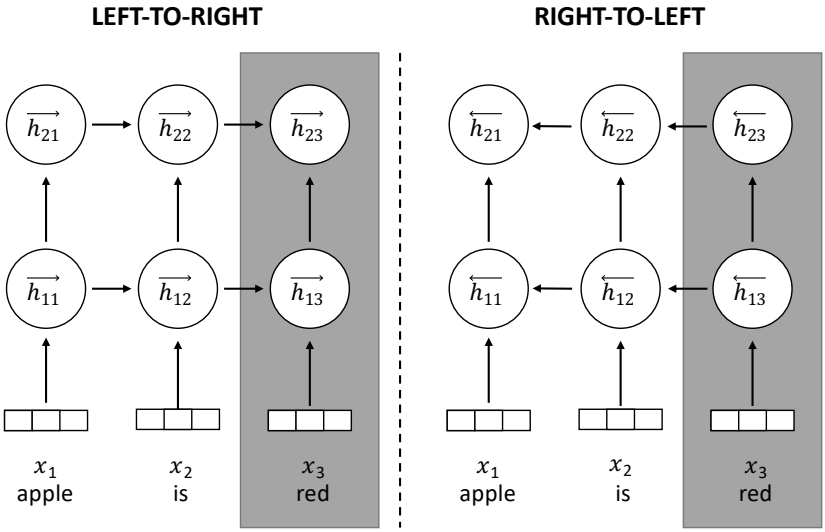


Figure 2.1. BiLSTM model. Each circle is an LSTM cell. x denotes initial word embeddings and h_{ij} denotes hidden states where i is the layer index and j is the jth word in the sequence.

this structure, ELMo creates embeddings that contain both the next and previous word information. For downstream tasks, a linear combination of all hidden states and initial embedding is used to get different information encoded in each layer. For example in Figure 2.1, "apple is red" is represented as a sequence of vectors x_1, x_2, x_3 where x_i is a fixed-size word embedding (e.g. word2vec). BiLSTM takes this input and creates hidden states. Then the embedding of "red" is created as follows:

$$E_{red} = \lambda_1 x_3 + \lambda_2[\overrightarrow{h_{13}}; \overleftarrow{h_{13}}] + \lambda_3[\overrightarrow{h_{23}}; \overleftarrow{h_{23}}] \tag{2.1}$$

where λ_i is trainable parameter with the constraint $\sum_i \lambda_i = 1$.

2.1.2. Generative Pre-training

GPT learns the LM with the same objective as ELMo that is the next word prediction. Instead of BiLSTM, it uses transformer decoder architecture. ELMo encodes both left-to-right and right-to-left information via BiLSTM. However, the transformer decoder can only process the sequence in the forward direction. One of the biggest

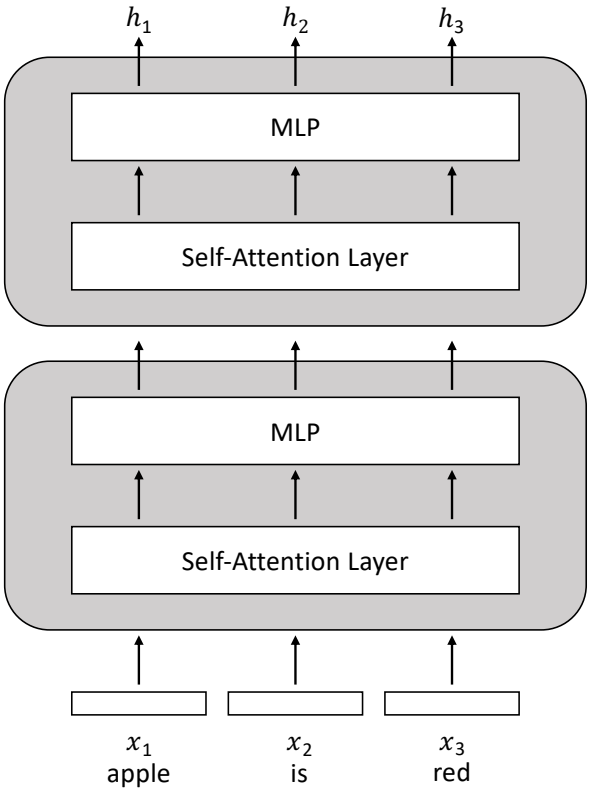


Figure 2.2. GPT model. This example model has two decoder layers, but the original has 12 layers. x_i denotes the input embedding of ith word. h_i denotes the last layer activation of ith word.

advantages of the transformer is its ability to model very long sequences. In BiLSTM, backpropagating the error from the last timestep to the first timestep is problematic for long sequences. However, the error backpropagation is independent of the sequence size in the transformer. Also, it processes the words in the sequence in parallel, unlike BiLSTM, which leads to faster training.

GPT model structure is shown in Figure 2.2. In pre-training, to predict the next word, the last layer output of the current word is used with a linear layer with softmax function. For example, to predict word "red", h_2 is used. For downstream tasks, the last layer outputs (h_i) are used as contextualized word embeddings. Later, GPT is proceeded by GPT-2 [9] and GPT-3 [10] that have huge number of parameters with increasing text generation performance.

2.1.3. Transformer

The transformer has two main parts: the encoder and the decoder. Both components can have a different number of layers as in MLP. Each encoder layer includes the self-attention layer and MLP, and each decoder layer includes the self-attention layer, encoder-decoder attention layer, and MLP. While encoder takes all the sequence as input, the decoder only attends to previous words. Self-attention provides attending to the most relevant parts in the sequence to process a word. There are three special vectors for each word in the sequence which are query, key, and value vectors. These vectors are trainable parameters. While processing a word, the scores for each word is calculated by taking the dot product of the query vector of the current word and the key vectors of others. The softmax of the scores can be treated as attention scores. The weighted average of value vectors with these attention scores is taken to create new representations of words. These representations not only include information about the word itself but information from all words. In multi-head attention, multiple self-attention modules gather information from different segments of the sequence.

Multi-head attention is calculated as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.2}$$

where Q, K, V are query, key and value matrices respectively, and d_k is the dimensionality of word vectors. These are calculated by multiplying the previous layer's activations with trainable matrices. QK^T gives information about how the words related to each other. Then each word takes a convex combination of values of all words to create its new representation.

The transformer does not process words sequentially; therefore, we need to indicate the position of the words explicitly. For this reason, positional embeddings are added to word embeddings. These positional embeddings can be either trainable or constant.

2.1.4. Bidirectional Encoder Representations from Transformers

BERT pre-training (Figure 2.3) is done by predicting randomly masked words (masked language model, masked LM) and predicting whether two sentences are consecutive or not (next sentence prediction, NSP). These two objectives are optimized simultaneously. It uses transformer encoder architecture. As mentioned above, ELMo can create contextual embeddings by encoding both left-to-right and right-to-left directions, while GPT can only include left-to-right information because of the pre-training task (next word prediction) and transformer decoder structure. However, these methods do not gather information *simultaneously* from both the left and right contexts. BERT handles this issue by predicting masked words with a multi-head attention mechanism, which allows the model to attend both previous and later words. In addition to word embeddings, BERT also learns a sentence embedding, which helps training some downstream tasks such as textual entailment. BERT forces the model to learn sentence relations with the NSP task. Two different models, named $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$, which have different hyper-parameters, were published in the original paper. The for-

mer has 12 encoder layers; the latter has 24 encoder layers.

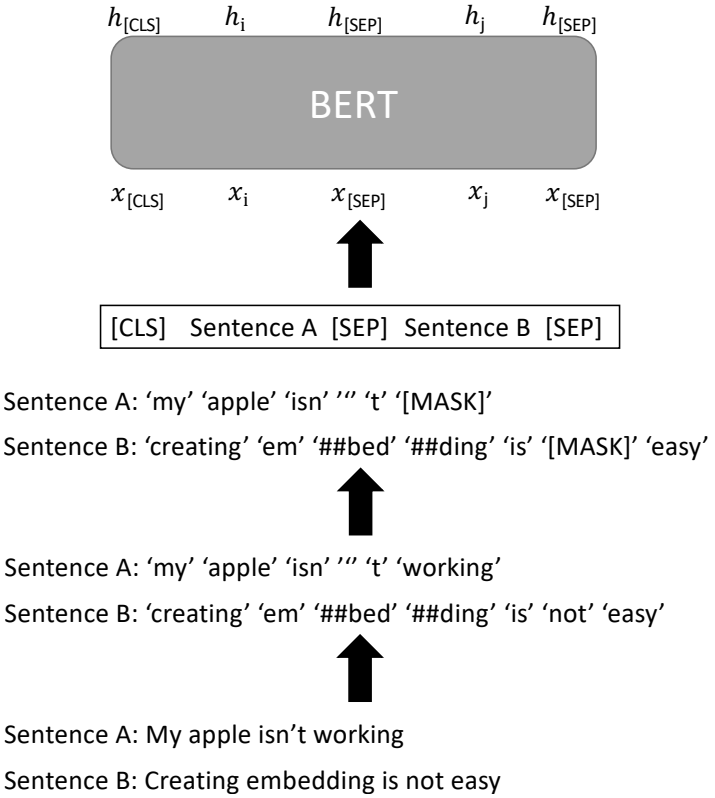


Figure 2.3. BERT model. x_i denotes the input embedding of ith word. h_i denotes the last layer activation of ith word.

The details of BERT training are as follows. BERT takes words of a sequence as its input. This sequence consists of either a single segment or a pair of two segments. Each segment includes one or more natural sentences or portions of sentences. It uses the WordPiece [11] tokenization technique, which divides the words into commonly used sub-words. There are additional special tokens: `[CLS]` as a start token and `[SEP]` as an end token to indicate segment ends. The last hidden state of the `[CLS]` token is the sequence embedding. Each token x_i is represented by the summation of three embeddings: token embeddings, segment embeddings, and positional embeddings. Here, segment embeddings represent the order of the segments, and the position embeddings show the index of the token in the sequence. These embeddings are trainable and updated in both pre-training and fine-tuning steps. For masked LM task, 15% of all WordPiece tokens in each sequence are masked randomly. The data is duplicated ten

times; therefore, different words are masked for each sequence. Final hidden vectors corresponding to these masked tokens are fed into a softmax layer to predict the correct WordPiece tokens. This method is analogous to denoising autoencoders [12] where we corrupt the input and predict its original version. In the masking step, the selected token is replaced with `[MASK]` token 80% of the time or with a random token 10% of the time or unchanged 10% of the time. Segments in the input sequence are either consecutive sentences or two unrelated sentences. The last hidden vector corresponding to `[CLS]` token (sequence embedding) is fed into another softmax layer to predict "isNext" or "notNext" label. In the pre-training step, masked LM and NSP classifiers are trained simultaneously using the same training instances. The training instances are generated in two sizes: short length sequences (up to 128 tokens) and long length sequences (up to 512 tokens). The model is trained with short length sequences for 90% of the steps, then with long length sequences for the rest 10% of the steps to make faster pre-training.

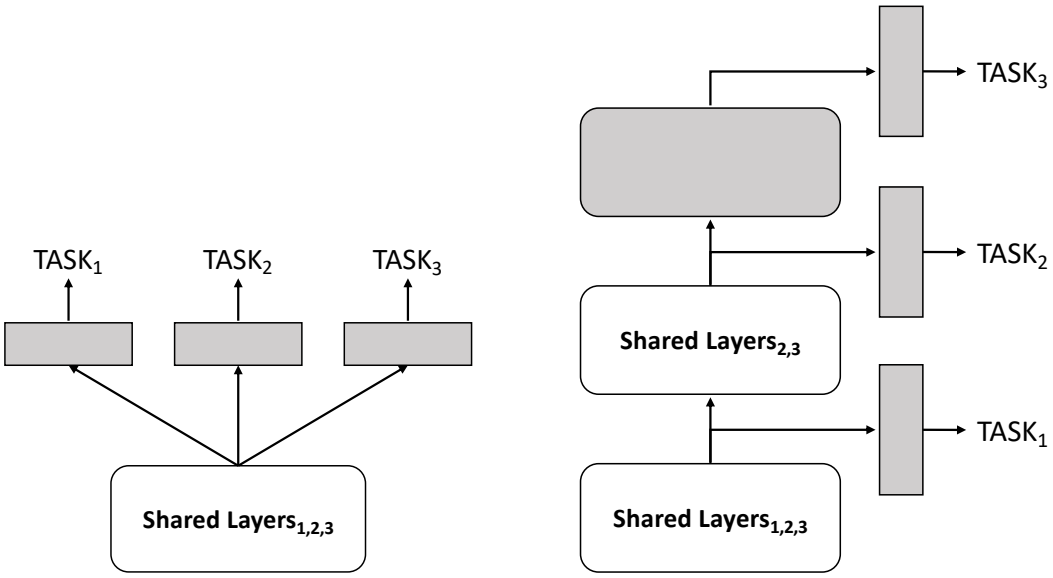
There are different studies to replicate and improve BERT model architecture. Robustly Optimized BERT Approach (RoBERTa) [13] makes some adjustments for pre-training. The model is pre-trained longer on the more data with larger batch sizes to have a better LM. The original BERT uses the sub-word level vocabulary with size 30K tokens, RoBERTa uses a more extensive vocabulary with size 50K tokens. The NSP loss is removed; the model is pre-trained with only masked LM task. It takes full sentences as input instead of partial segments. While the original BERT makes the masking operation for once before training, RoBERTa makes it in each step randomly. This dynamic masking operation provides the model to see different masked words for sequences in each iteration. Also, RoBERTa does not use shorter length sequences; the model is pre-trained with full-length sequences. A Lite BERT (ALBERT) [14] proposes two ways to reduce the parameter size of the BERT structure. The first technique is the factorization of the embedding parameters. Instead of using one big sized parameter matrix to project WordPiece embeddings to the hidden representations, two different factorized matrices with fewer parameters are used. The second technique is the cross-layer parameter sharing. The constraint is that all layers will have the same parameters

in the model. They also proposed a new pre-training task, sentence order prediction (SOP), instead of NSP. This task is predicting whether two consecutive segments are swapped or not. SOP forces the model to evaluate the coherence between two segments, while NSP focuses on their relatedness.

2.2. Multitask Learning

Multitask learning (MTL) [15] is an approach to construct one model for learning related tasks to achieve a better generalization performance. Each task assumes a different inductive bias by its nature. MTL forces the model to find a hypothesis, a set of learned parameters that should satisfy all tasks. This leads to a regularized model that contains all inductive biases. MTL is inspired by human learning. While trying to learn new things, we usually apply the knowledge we have gained from similar and relative events. There are successful examples of MTL in various domains such as computer vision [16], and reinforcement learning [17]. As there is only one model for multiple tasks, some parameters have to be shared across tasks. In deep learning architectures, there are different ways to design shared parameters of the model [18]. One way is to share initial layers for all tasks and keep task-specific output layers specialized for each task. This is known as *hard parameter sharing*. Another way is to use different models for each task but force them to be similar by applying regularization techniques known as *soft parameter sharing*. Generally, the last layer activations are used to solve multiple tasks (Figure 2.4a). Besides this parallel structure, the hierarchical MTL approach models tasks successively (Figure 2.4b). With this approach, complex tasks are predicted at deeper layers to take advantage of representations that are learned from lower-level tasks [19–21].

There are various issues about MTL. One issue is the task selection. MTL does not guarantee an increment in the performance; in case of training a model with tasks that do not support each other, the accuracy for each task can be lower than their single task structures [22]. Even if the selected tasks are related to each other, their complexities can be different. There is a risk that straightforward tasks might overfit the



(a) Parallel MTL (b) Hierarchical MTL

Figure 2.4. MTL architectures. Parameters of gray-colored layers are learned to solve each task independently. Parameters of shared layers are learned to solve more than one task which are indicated by task numbers.

data before more complex tasks have not been learned yet. This risk can be eliminated to some extent, with some regularization techniques and by modeling task hierarchies. Another critical issue is to decide the order of tasks during training iterations. The model should be regularized to avoid catastrophic forgetting where previously learned parameters are forgotten while adapting to new tasks.

There are several works for a hierarchical MTL approach on different NLP tasks. Søgaard and Goldberg [19] applied this approach for some of the most known NLP tasks: part-of-speech (POS) tagging, chunking, and parsing. From a linguistic perspective, POS tags can be used as features for chunking and parsing problems; therefore, the POS tagging task is positioned as a lower-level task by predicting it at lower levels of the deep network. Different models are trained for POS tagging+chunking and POS tagging+parsing task combinations: single-task training (without POS tagging task), parallel MTL (POS tagging task is learned at the outer layer, Figure 2.4a), and hierarchical MTL (POS tagging task is learned at the inner layers, Figure 2.4b). BiLSTM is used for these experiments. For each training iteration, a task is selected randomly to compute the expected loss and update parameters.

Hashimoto et al. [20] have chosen the tasks depending on their linguistic hierarchies. Respectively, a deep network is designed for POS tagging, chunking, dependency parsing, semantic relatedness, and textual entailment. The first two tasks are word-level, third is syntactic, and the last two tasks are semantic. BiLSTM is used for each task, and its number of layers is successively increased by order of the task. There are shortcut (skip) connections from lower-level tasks to deeper-level tasks. They propose a regularization technique to deal with the catastrophic forgetting problem. During training, tasks are not selected randomly but selected by their hierarchy in the model. At each epoch, the model is trained over the full dataset of a task. For the next task, parameters are regularized not to deviate too much from the previous task. This is known as *successive regularization*.

Sanh et al. [21] have chosen named entity recognition (NER), entity mention detection (EMD), coreference resolution (CR) and relation extraction (RE) tasks. They focused on semantic tasks because they thought that semantic and syntactic tasks might not be related to being learned together. The CR and RE tasks are positioned at the deepest level. For each task, BiLSTM is implemented as an encoder, and its output is fed to the corresponding conditional random fields. As in [20], there are shortcut connections from the lowest level to the deeper-level encoders. The order of tasks during training is random.

3. METHODS

In the original BERT pre-training, there are two objectives: masked LM and NSP. These are optimized simultaneously to learn a language model (LM). From a multitask learning point of view, minimizing the loss of two objectives at the same time should enhance the performance if these two tasks are related to each other [22]. In this work, we focus on BERT as it uses multiple training objectives. We aim to apply successful approaches from multitask learning to LM learning. To our knowledge, there is no prior work that analyzes the relation between these two tasks. As the relation of tasks is an essential factor, another important factor is the task complexity. For example, if one of these tasks is more straightforward than the other, the model may overfit to easier task while the training for the other task is not complete. This is an unwanted consequence since a part of the model memorizes the task instead of generalizing it. This could have been prevented by early stopping; however, it is hard to decide when to stop even for one task [23].

In order to describe the contributions of this thesis, we first define BERT in a formal way. Let s be the example sequence:

$$s = [x_{\texttt{[CLS]}}, x_1, x_2, \ldots, x_{\texttt{[SEP]}}, x_k, x_{k+1}, \ldots, x_{\texttt{[MASK]}}, \ldots, x_{\texttt{[SEP]}}] \tag{3.1}$$

where x_i is the summation of token, segment and position embeddings of ith token in the sequence. `[CLS]`, `[SEP]`, and `[MASK]` are special tokens: $x_{\texttt{[CLS]}}$ embedding represents the start of the sequence, $x_{\texttt{[SEP]}}$ embedding indicates the end of each segment, $x_{\texttt{[MASK]}}$ embedding indicates the masked tokens. Each x_i embedding and $x_{\texttt{[CLS]}}$, $x_{\texttt{[SEP]}}$, $x_{\texttt{[MASK]}}$ embeddings are trainable input vectors. In the original $\text{BERT}_{\text{base}}$ architecture, there is an encoder stack with 12 encoder layers. Let us denote this encoder stack by E. E takes the whole sequence s as input, and creates the last layer activation o_i for

ith word in the sequence:

$$[o_{\texttt{[CLS]}}, o_1, o_2, \ldots, o_{\texttt{[SEP]}}, o_k, o_{k+1}, \ldots, o_{\texttt{[MASK]}}, \ldots, o_{\texttt{[SEP]}}] = E(s) \tag{3.2}$$

For downstream tasks, o_i is used as contextualized embedding of ith word. For pre-training, $o_{\texttt{[CLS]}}$ and $o_{\texttt{[MASK]}}$ are used to train masked LM and NSP classifiers:

$$p_m = C_m(o_{\texttt{[MASK]}}) \tag{3.3}$$

$$p_n = C_n(o_{\texttt{[CLS]}}) \tag{3.4}$$

where C_m is the masked LM classifier, C_n is the NSP classifier, p_m and p_n are output probabilities of WordPiece tokens and the next sentence probability, respectively.

3.1. Changes in Pre-training

We experimented a multitask learning approach to BERT pre-training in several ways:

(i) NSP classifier is trained with `[CLS]` embedding from a selected lower layer instead of the last layer. We refer to this model as Lower NSP (Figure 3.1a). Here `[CLS]` is the start token of the input. This embedding is considered to be a sentence-level embedding [8]. Since we do not exactly know the task hierarchy, we also do the opposite where the masked LM classifier is trained with the embeddings from a selected lower layer, which correspond to masked tokens in the input. We refer to this model as Lower Mask (Figure 3.1b).

Motivation: Modeling task hierarchies might be beneficial for a better LM. Learning the hierarchy between these tasks can shed light on other studies from a linguistic perspective.

Definition: In Lower NSP and Lower Mask, there are two encoder stacks instead of one. These encoder stacks are represented by E_1 and E_2. E_1 is called the first or lower encoder, E_2 is called the second encoder. To have a fair comparison with

the $\text{BERT}_{\text{base}}$, the total number of encoder layers in these stacks is equal to 12. For example, if E_1 has five layers, then E_2 has seven layers. Let us denote the last layer activations of E_1 and E_2 as h_i and o_i for ith token, respectively:

$$h = [h_{\texttt{[CLS]}}, h_1, h_2, \ldots, h_{\texttt{[SEP]}}, h_k, h_{k+1}, \ldots, h_{\texttt{[MASK]}}, \ldots, h_{\texttt{[SEP]}}] = E_1(s) \quad (3.5)$$

$$[o_{\texttt{[CLS]}}, o_1, o_2, \ldots, o_{\texttt{[SEP]}}, o_k, o_{k+1}, \ldots, o_{\texttt{[MASK]}}, \ldots, o_{\texttt{[SEP]}}] = E_2(h) \quad (3.6)$$

where s is the input sequence as in Equation 3.1. Instead of computing $E(s)$ as in the original BERT model, here $E_2(E_1(s))$ is computed. In the original structure, errors of both tasks are backpropagated from the last layer activations, $E(s)$. In our proposed structure, the error for one task is backpropagated from $E_1(s)$ and the error for the other task is backpropagated from $E_2(h)$. This forces encoder stacks to specialize for different tasks. For Lower NSP, `[CLS]` embedding from $E_1(s)$ is used to train NSP classifier. For Lower Mask, `[MASK]` embeddings from $E_1(s)$ are used to train masked LM classifier. These are summarized in Table 3.1.

Table 3.1. Inputs to NSP and masked LM classifiers for different models.

Models	NSP classifier	Masked LM classifier
Original BERT	$o_{\texttt{[CLS]}}$	$o_{\texttt{[MASK]}}$
Lower NSP	$h_{\texttt{[CLS]}}$	$o_{\texttt{[MASK]}}$
Lower Mask	$o_{\texttt{[CLS]}}$	$h_{\texttt{[MASK]}}$

(ii) We concatenate `[CLS]` embedding, (Figure 3.2a), or the output of the NSP classifier to the input of masked LM classifier (Figure 3.2b). This will (1) indirectly regularize the sentence-level embedding since it will be used for both NSP and masked LM tasks (2) explicitly provide sentence-level information to masked LM classifier.

Motivation: Regularization of the sentence-level embedding.

Definition: There is no change in the inputs of the NSP classifier for all models. However, `[CLS]` embedding, which is used to train the NSP classifier, is also used

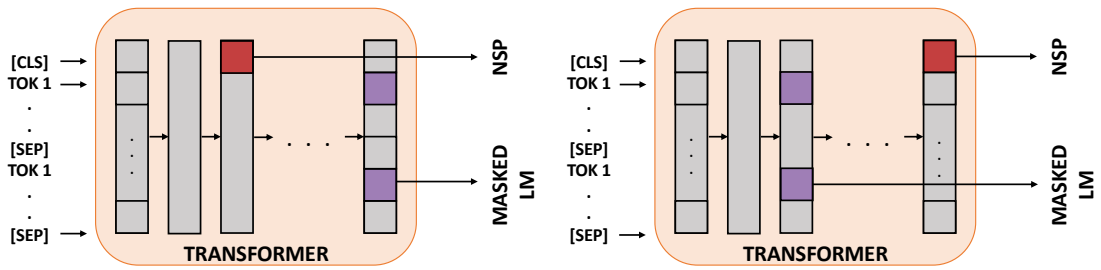
as an additional input to masked LM classifier. These changes are summarized in Table 3.2.

Table 3.2. Inputs to masked LM classifier for concatenated models.

Models	No concat.	CLS concat.	NSP concat.
Original BERT	$o_{\texttt{[MASK]}}$	$[o_{\texttt{[CLS]}}; o_{\texttt{[MASK]}}]$	$[p_n; o_{\texttt{[MASK]}}]$
Lower NSP	$o_{\texttt{[MASK]}}$	$[h_{\texttt{[CLS]}}; o_{\texttt{[MASK]}}]$	$[p_n; o_{\texttt{[MASK]}}]$
Lower Mask	$h_{\texttt{[MASK]}}$	$[o_{\texttt{[CLS]}}; h_{\texttt{[MASK]}}]$	$[p_n; h_{\texttt{[MASK]}}]$

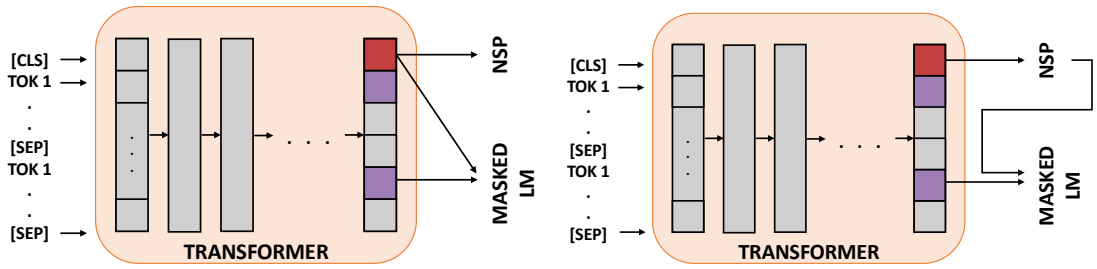
(iii) Motivated from probing tasks [24], we randomly swap the order of bigrams 15% of the time for each input. We use an additional classifier, which predicts whether a token is in the right position in the sentence or not. This will force embeddings to encapsulate the word order information.

(iv) In our experiments for Lower NSP models, we see that the NSP classifier fits faster than the masked LM classifier. Further training will cause overfitting for the NSP classifier. Therefore we do experiments on freezing all parts that affect the NSP classifier. We refer to this model as Lower NSP-freeze.



(a) Lower NSP (b) Lower Mask

Figure 3.1. Hierarchical BERT architectures. (a) using lower level embedding of `[CLS]` token for NSP classifier; (b) using lower level embeddings of masked tokens for masked LM classifier.



(a) CLS concatenation (b) NSP concatenation

Figure 3.2. Concatenation techniques. (a) concatenation of `[CLS]` embedding to each input of masked LM classifier; (b) concatenation of the output of NSP classifier to each input of masked LM classifier.

3.2. Changes in Fine-tuning

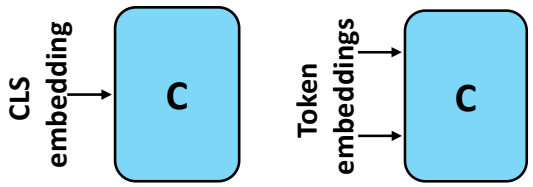
There are several changes in the fine-tuning for downstream tasks to accommodate the changes in the pre-training. In the original BERT structure, last layer embeddings are used for both sentence-level tasks (Figure 3.3a) and token-level tasks (Figure 3.3b):

$$p_s = C_s(o_{\texttt{[CLS]}}) \tag{3.7}$$

$$p_t = C_t(o_i) \tag{3.8}$$

where C_s is the sentence-level classifier, C_t is the token-level classifier, p_s and p_t are output probabilities of sentence and token classes, respectively. Sentence-level task means there is only one output label for a sentence (sequence). For example, the prediction of the sentiment of a given sequence can be considered as a sentence-level task. Token-level task means each token in a sequence is labeled. For instance, in POS-tagging problem, each token in the sequence has to be predicted.

With our hierarchical approach, `[CLS]` embedding is selected from the layer which we train NSP classifier (Figure 3.1a); token embeddings are selected from the layer



(a) Sentence-level task (b) Token-level task

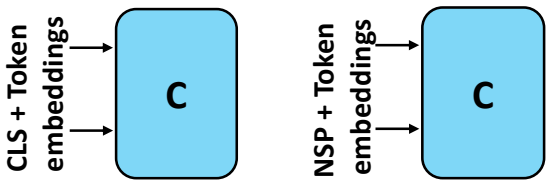
Figure 3.3. Downstream architectures. c is a classifier. (a) `[CLS]` embedding from NSP level is used for sentence-level tasks. (b) Embeddings from masked LM level are used for token-level tasks.

which we train masked LM classifier (Figure 3.1b). For Lower NSP, `[CLS]` embedding from $E_1(s)$ is used to train C_s and token embeddings from $E_2(h)$ are used to train C_t. For Lower Mask, `[CLS]` embedding from $E_2(h)$ is used to train C_s and token embeddings from $E_1(s)$ are used to train C_t. These changes are summarized in Table 3.3.

Table 3.3. Inputs for sentence-level and token-level downstream tasks.

Models	Sentence-level task	Token-level task
Original BERT	$o_{\texttt{[CLS]}}$	o_i
Lower NSP	$h_{\texttt{[CLS]}}$	o_i
Lower Mask	$o_{\texttt{[CLS]}}$	h_i

If `[CLS]` embedding or NSP output is included in the pre-training, we also include this extra information in the fine-tuning for token-level tasks (Figure 3.4a, 3.4b). Since they are trained together in the pre-training step, token embeddings may depend on this extra information. There is no change in the input for sentence-level tasks. These are summarized in Table 3.4.



(a) CLS concatenation (b) NSP concatenation

Figure 3.4. Downstream architectures with concatenated parts. c is a classifier. In (a), `[CLS]` embedding from NSP level is concatenated to each input. In (b), NSP classifier output is concatenated to each input.

Table 3.4. Inputs for token-level downstream tasks for concatenated models.

Models	No concat.	CLS concat.	NSP concat.
Original BERT	o_i	$[o_{\texttt{[CLS]}}; o_i]$	$[p_n; o_i]$
Lower NSP	o_i	$[h_{\texttt{[CLS]}}; o_i]$	$[p_n; o_i]$
Lower Mask	h_i	$[o_{\texttt{[CLS]}}; h_i]$	$[p_n; h_i]$

4. EXPERIMENTS AND RESULTS

4.1. Datasets

We tested different proposed architectures on several datasets. Due to computational issues, we make pre-training on small datasets as opposed to the original BERT. For this, we have two different approaches. First, we used the raw text data of the downstream task for pre-training. In the second approach, we use a different raw text with similar size of the first data. We experimented on two downstream tasks: question answering (token-level task) and textual entailment (sentence-level task).

For question answering, we used SQuAD1.1 (The Standford Question Answering Dataset) [25] and SQuAD2.0 [26] datasets. These two datasets are used for both pre-training and fine-tuning on downstream tasks. Additionally, we used WikiText-2 for pre-training. We followed the same strategy as in BERT to create pre-training data from paragraphs of these three datasets separately. We pre-train the model with inputs that contain less than 128 tokens for 90% of steps, as in BERT. However, for the rest of 10% of steps, we used inputs containing less than 384 tokens instead of 512.

For textual entailment, we used Multi-Genre Natural Language Inference corpus [27] (MultiNLI). This dataset does not contain paragraphs but independent sentences. Therefore we did not use this dataset for pre-training. Instead, we used the models pre-trained on the WikiText-2 dataset.

4.1.1. Pre-training Data Preparation

For SQuAD1.1, SQuAD2.0, WikiText-2, each paragraph is treated as a separate document while creating pre-training data. As a first step, these documents are split into segments by some punctuations, e.g. “.”, “:”, “?”, “!”, “?”. The separated segments are pre-processed by an uncased tokenizer, making tokens lowercase and dividing

them into WordPiece tokens. Generating pre-training sequences from these processed documents is duplicated ten times to create more diverse training examples. As mentioned above, the training examples are created with two different maximum combined sequence lengths: 128 and 384. For both sequence lengths, shorter sequences are created with a 10% probability. To prepare instances for the NSP task, a random number of consecutive segments are received as the first sequence for each document. The second sequence is taken from the remaining segments in the same document with a 50% probability (NSP positive). Different consecutive segments are taken from another randomly selected document with a 50% probability (NSP negative). After these two sequences are randomly truncated form the beginning or end based on the maximum sequence length, they are combined with specific tokens `[CLS]` and `[SEP]`. Except for these specific tokens, the tokens are randomly masked with the same percentages as in the original BERT.

Table 4.1. The number of pre-training examples in the train sets. Short column represents the sequences with a maximum length of 128 tokens. Long column represents the sequences with a maximum length of 384 tokens.

Train Data	**SQuAD1.1**		**SQuAD2.0**		**WikiText-2**	
	Short	Long	Short	Long	Short	Long
NSP-Positive	151,099	129,960	151,968	130,887	125,093	104,099
NSP-Negative	252,206	197,469	254,986	198,905	229,304	183,603

Table 4.2. The number of pre-training examples in the validation sets.

Validation Data	**SQuAD1.1**		**SQuAD2.0**		**WikiText-2**	
	Short	Long	Short	Long	Short	Long
NSP-Positive	17,096	14,469	10,355	8,589	13,017	11,083
NSP-Negative	28,457	21,758	16,845	12,895	23,696	19,138

Dataset information regarding SQuAD1.1, SQuAD2.0 and WikiText-2 datasets are shown in Tables 4.1 and 4.2 for train and validation sets, respectively. Notice

that there are more NSP-negative examples for all datasets. This is because there are documents that consist of a single segment that prevents the dataset creation procedure from generating an NSP-positive example.

4.1.2. Downstream Data Preparation

Question answering is a token-level task, where the model predicts the start and end index of the answer in the paragraph when a question-paragraph pair is given. It is mainly an extractive question answering. The answer is direct in the paragraph; it is not formed with different words. SQuAD datasets include paragraphs and different questions related to these paragraphs. SQuAD2.0 is an extended version of SQuAD1.1. It contains questions that do not have a possible answer in the given context.

Table 4.3. The number of examples in the SQuAD sets.

	SQuAD1.1		**SQuAD2.0**	
	Train	Validation	Train	Validation
Total examples	87,599	10,570	130,319	11,873
No-answer examples	0	0	43,498	5,945

SQuAD dataset information is given in Table 4.3. These question-paragraph pairs are pre-processed to train two classifiers: one predicts which token is the start, and the other predicts which token is the end of the answer, with BERT embeddings as their inputs. As mentioned above, the maximum combined sequence length for pre-training a BERT model is 384 in our architecture. Therefore, the question-paragraph pairs are processed to have a maximum of 384 WordPiece tokens. The question is treated as the first sequence and the paragraph as the second sequence for each example. The maximum length of 64 WordPiece tokens is set for the first sequence. Firstly, the question and the paragraph are tokenized separately. If the tokenized question is longer than 64, the first 64 WordPiece tokens are taken. The tokenized paragraph is divided into alternative small pieces containing the answer with 128 stride size. Thus, it is

possible to create more than one training instance from a single question-paragraph pair. These two tokenized and truncated sequences are combined with the specific tokens `[CLS]` and `[SEP]`. In Table 4.4, an example question-paragraph pair is shown. Different from SQuAD1.1, the start and end indices of the no-answer instances are labeled as 0 in SQuAD2.0. The number of examples to train QA classifiers regarding SQuAD1.1 and SQuAD2.0 datasets are shown in Table 4.5.

Table 4.4. Data preparation example from SQuAD1.1 dataset.

Question: What could a teacher help in organizing? Paragraph: A teacher's professional duties may extend beyond formal teaching. Outside of the classroom teachers may accompany students on field trips, supervise study halls, help with the organization of school functions, and serve as supervisors for extracurricular activities. In some education systems, teachers may have responsibility for student discipline. Answer: school functions
TOKENIZATION
Question: [‘what’, ‘could’, ‘a’, ‘teacher’, ‘help’, ‘in’, ‘organizing’, ‘?’]
Paragraph: [‘a’, ‘teacher’, ‘”, ‘s’, ‘professional’, ‘duties’, ‘may’, ‘extend’, ‘beyond’, ‘formal’, ‘teaching’, ‘.’, ‘outside’, ‘of’, ‘the’, ‘classroom’, ‘teachers’, ‘may’, ‘accompany’, ‘students’, ‘on’, ‘field’, ‘trips’, ‘,’, ‘supervise’, ‘study’, ‘halls’, ‘,’, ‘help’, ‘with’, ‘the’, ‘organization’, ‘of’, ‘school’, ‘functions’, ‘,’, ‘and’, ‘serve’, ‘as’, ‘supervisors’, ‘for’, ‘extra’, ‘##cu’, ‘##rricular’, ‘activities’, ‘.’, ‘in’, ‘some’, ‘education’, ‘systems’, ‘,’, ‘teachers’, ‘may’, ‘have’, ‘responsibility’, ‘for’, ‘student’, ‘discipline’, ‘.’]
COMBINATION
[`[CLS]` ‘what’, ‘could’, ‘a’, ‘teacher’, ‘help’, ‘in’, ‘organizing’, ‘?’ `[SEP]` ‘a’, ‘teacher’, ‘”, ‘s’, ‘professional’, ‘duties’, ‘may’, ‘extend’, ‘beyond’, ‘formal’, ‘teaching’, ‘.’, ‘outside’, ‘of’, ‘the’, ‘classroom’, ‘teachers’, ‘may’, ‘accompany’, ‘students’, ‘on’, ‘field’, ‘trips’, ‘,’, ‘supervise’, ‘study’, ‘halls’, ‘,’, ‘help’, ‘with’, ‘the’, ‘organization’, ‘of’, ‘school’, ‘functions’, ‘,’, ‘and’, ‘serve’, ‘as’, ‘supervisors’, ‘for’, ‘extra’, ‘##cu’, ‘##rricular’, ‘activities’, ‘.’, ‘in’, ‘some’, ‘education’, ‘systems’, ‘,’, ‘teachers’, ‘may’, ‘have’, ‘responsibility’, ‘for’, ‘student’, ‘discipline’, ‘.’ `[SEP]`]

Table 4.5. The number of fine-tuning examples for question-answering task.

	SQuAD1.1		**SQuAD2.0**	
	Train	Validation	Train	Validation
Total examples	87,995	10,671	131,944	12,232
No-answer examples	0	0	44,732	6,244

Textual entailment is a sentence-level classification task which predicts the relation of two sentences: premise and hypothesis. There are three classes: entailment (the premise entails the hypothesis), contradiction (the premise contradicts the hypothesis), and neutral. MultiNLI has two validation sets: matched set and mismatched set. The matched set contains examples from the same source as the training set; the mismatched set includes examples from different sources.

Table 4.6. The number of examples in the MultiNLI sets.

	Train	**Matched Validation**	**Mismatched Validation**
Total examples	392,702	9,815	9,832

MultiNLI dataset information is given in Table 4.6. Each premise-hypothesis pair is pre-processed for the BERT input structure. Firstly, they are tokenized separately. If the total length of these tokenized sequences is longer than 384, they are truncated randomly until 384. Finally, they are combined with `[CLS]` and `[SEP]` tokens. In Table 4.7, an example premise-hypothesis pair is shown. The processed data size is equal to the original MultiNLI dataset.

4.1.3. Probing Tasks

To evaluate the quality of the learned embeddings, we used probing tasks in [24]. We only assess embeddings that are pre-trained on the WikiText-2 dataset since these are used for both question answering and textual entailment tasks. Probing tasks are

Table 4.7. Data preparation example from MultiNLI dataset.

Premise: At the other end of Pennsylvania Avenue, people began to line up for a White House tour. Hypothesis: People formed a line at the end of Pennsylvania Avenue. Label: entailment
TOKENIZATION
Premise: [‘at’, ‘the’, ‘other’, ‘end’, ‘of’, ‘pennsylvania’, ‘avenue’, ‘,’, ‘people’, ‘began’, ‘to’, ‘line’, ‘up’, ‘for’, ‘a’, ‘white’, ‘house’, ‘tour’, ‘.’]
Hypothesis: [‘people’, ‘formed’, ‘a’, ‘line’, ‘at’, ‘the’, ‘end’, ‘of’, ‘pennsylvania’, ‘avenue’, ‘.’]
COMBINATION
[`[CLS]` ‘at’, ‘the’, ‘other’, ‘end’, ‘of’, ‘pennsylvania’, ‘avenue’, ‘,’, ‘people’, ‘began’, ‘to’, ‘line’, ‘up’, ‘for’, ‘a’, ‘white’, ‘house’, ‘tour’, ‘.’ `[SEP]` ‘people’, ‘formed’, ‘a’, ‘line’, ‘at’, ‘the’, ‘end’, ‘of’, ‘pennsylvania’, ‘avenue’, ‘.’ `[SEP]`]

sentence-level classification problems that require different linguistic properties. If a classifier trained by the sentence embeddings is successful for a probing task, it can be said that these embeddings encode the linguistic property that the task focuses on. It is a valuable method for the interpretation of the learned embeddings.

Ten different probing tasks are categorized into three groups: surface information tasks, syntactic information tasks, and semantic information tasks. Surface information tasks are sentence length (SL) and word content (WC). Syntactic information tasks are tree depth (TD), bigram shift (BS), and top constituents (TC). Semantic information tasks are tense (T), subject number (SN), object number (ON), semantic odd man out (OM) and coordination inversion (CI). For each task, there is a train set with 100K examples, a validation set with 10K examples, and a test set with 10K examples. In these sets, the number of classes is balanced. These sets are pre-processed for the BERT input structure. Each example is tokenized with a WordPiece tokenizer. Then, `[CLS]` token is added to the beginning of the tokenized sentence, `[SEP]` token added to the end. Definitions of the tasks are as follows:

- Sentence Length (SL): Predict the length of the sentences. Sentences are divided into six categories based on the number of tokens they contain. These categories are determined by the following ranges: 5-8, 9-12, 13-16, 17-20, 21-25, 26-28. It is to test if the sentence embedding includes the count of the tokens that form the sentence.
- Word Content (WC): Predict whether a sentence contains one of the most frequently used words or not. 1000 common words are chosen. It is to test if the sentence embedding can extract information about the words used in the sentence.
- Tree Depth (TD): Predict the maximum depth of the phrase structure tree. It is to test if the sentence embedding encodes the information about the syntactic structure of the sentence.
- Bigram Shift (BS): Predict whether a sentence contains swapped bigram or not. It is to test if the sentence embedding can capture the knowledge of the natural word orders.
- Top Constituents (TC): Predict the sentence's top constituents sequence. It is to test if the sentence embedding encodes the information about the construction of the sentence.
- Tense (T): Predict the main verb of the sentence is past or present. It is to test if the sentence embedding carries information regarding time.
- Subject Number (SN): Predict the number of the subject is singular or plural. It is to test if the sentence embedding contains information about the subject.
- Object Number (ON): Predict the number of the direct object is singular or plural. It is to test if the sentence embedding contains information about the object.
- Semantic Odd Man Out (OM): Predict whether there is a word in the sentence replaced with a randomly selected word. It is to test if the sentence embedding includes information about the consistency of the sentence.
- Coordination Inversion (CI): Predict the sentence with a conjunction is inverted or not. It is to test if the sentence embedding includes information about the integrity of the whole sentence.

4.2. Training Details

We made modifications that are explained in Section 3 to the $\text{BERT}_{\text{base}}$ architecture. We pre-trained all variants of architectures in our pre-training datasets and fine-tuned them on downstream tasks. We select 1e-4 as the learning rate with 1e-4 weight decay for pre-training, and 1e-5 as the learning rate with no weight decay for fine-tuning. These hyperparameters are found by grid search done on $\text{BERT}_{\text{base}}$ architecture. We set the batch size to be 32 for inputs with length up to 128 tokens and one for longer inputs in the pre-training. Batch size for fine-tuning is set to be one for all downstream tasks. We set dropout rate to 0.1. For all experiments, Adam optimizer [28] is used with default beta parameters and AMSGrad [29] option. All experiments are implemented with PyTorch automatic differentiation library [30]. We trained our models with NVIDIA V100 GPU. We used TRUBA computation facilities for our computations. In this architecture, pre-training took around one and a half-day for one model, and fine-tuning took around 20 hours for one model. The codes are available at `https://github.com/caksoy/master_thesis`.

4.3. Pre-training Results

$\text{BERT}_{\text{base}}$ and our modified architectures are pre-trained to minimize the sum of NSP binary cross-entropy loss and masked LM cross-entropy loss. For Lower NSP and Lower Mask architectures, we experimented with all intermediate layers to choose the best level of selected embeddings for the lower classifier. As we used $\text{BERT}_{\text{base}}$ architecture, which has 12 encoder layers, there are 11 different intermediate layers to select the lower classifier level. As an additional experiment, we use `[CLS]` concatenation or NSP concatenation for BERT and lower architectures. Figure 4.1 shows the relation between masked LM loss and NSP loss with the mask layer for Lower Mask architectures. Firstly, we see that as the mask layer increases, the masked LM performance increases as well. This shows that the masked LM task indeed requires deeper layers. Secondly, there is a sweet spot for the NSP task (fourth or fifth mask layer), where it performs the best for most of the models. At this spot, deeper layers can specialize in

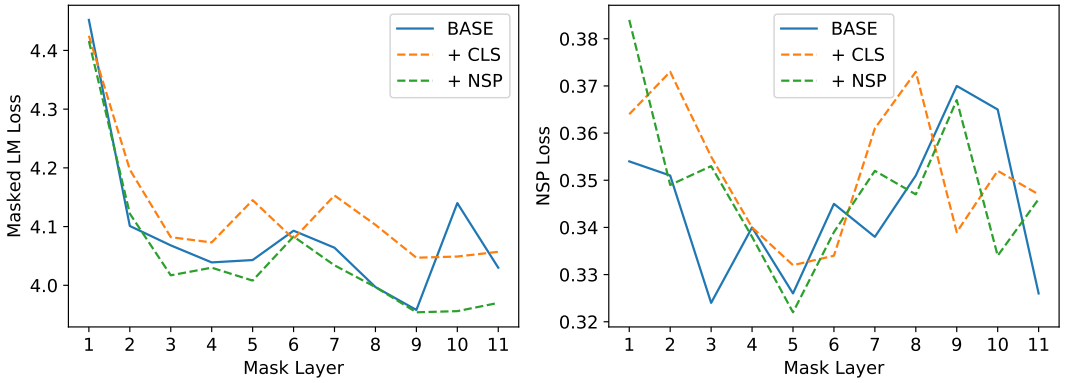
the NSP task. When we go left from this spot, the under-performance of the masked LM task harms the NSP performance. On the other hand, when we go right, the model focuses on the masked LM task and cannot specialize in the NSP task.

Figure 4.2 shows the effect of the NSP layer on the masked LM loss and NSP loss for Lower NSP architectures. Unlike masked LM task, fewer layers are sufficient to train NSP task (second column). This gives a clue about task complexities: the masked LM task is more complicated because it requires deeper layers. In the first column, we see that the masked LM performance decreases as we increase the NSP layer; there is no sweet spot as in Lower Mask architectures. Numerical results of Figure 4.1 and 4.2 are given in Table A.1, A.2, A.3, A.4, A.5, A.6.

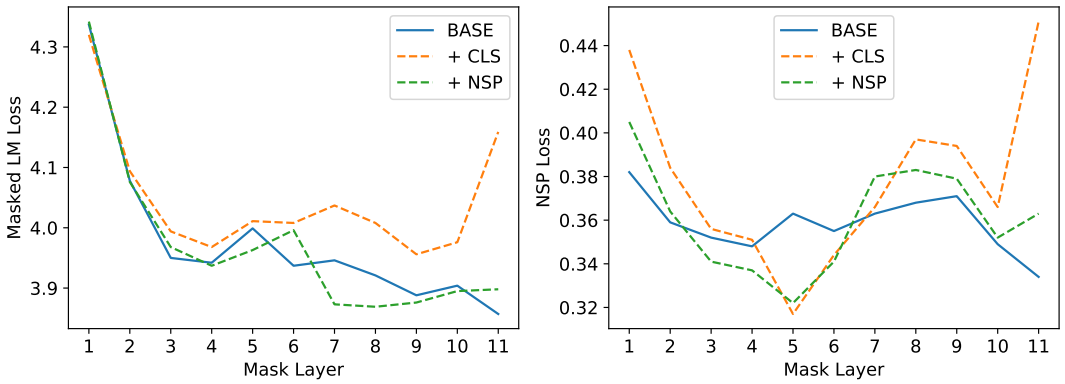
Apart from these models, we freeze NSP related parts of the model for cases in which Lower NSP classifier overfitted to data before masked LM training. In addition, to see the NSP task's importance, we removed the NSP classifier as in RoBERTa.

Table 4.8 shows the selected layers for the lower architectures. Layers with the minimum total loss are selected, except when one of the tasks performs very poorly. Generally, deeper layers are selected in Lower Mask architectures compared to Lower NSP architectures.

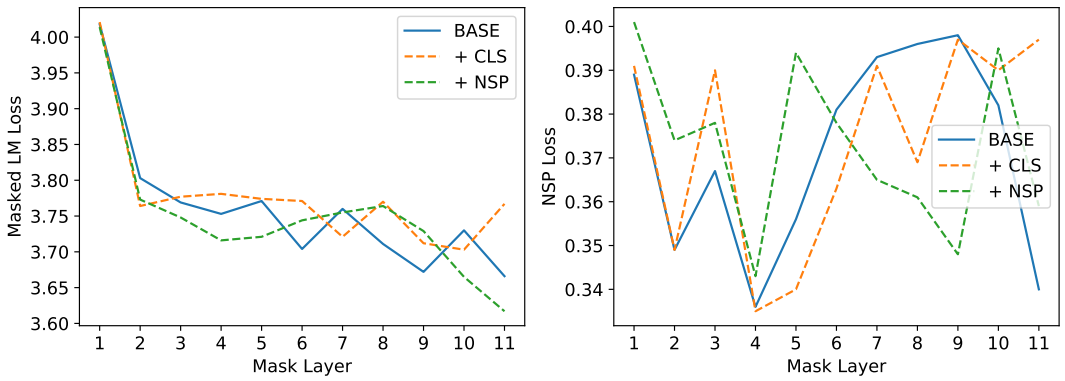
Tables 4.9 and 4.10 show NSP and mask accuracy of the selected pre-trained models on validation sets. We see that pre-trained BERT embeddings in Vaswani et al. [5] outperform all approaches except NSP accuracy on WikiText-2. This is because there are more NSP-negative examples in WikiText-2 than SQuAD sets, and the pre-trained BERT is biased to predict NSP as positive. However, we note that we do not compare our models with the pre-trained BERT since we only pre-train our models with small datasets. In tables, BERT is the replica, except it is pre-trained on the small data. Therefore, we compare our approaches with this model. There is no significant difference between pre-training performances. We should note that we evaluate models based on their fine-tuning performances instead of pre-training results.



(a) SQuAD1.1

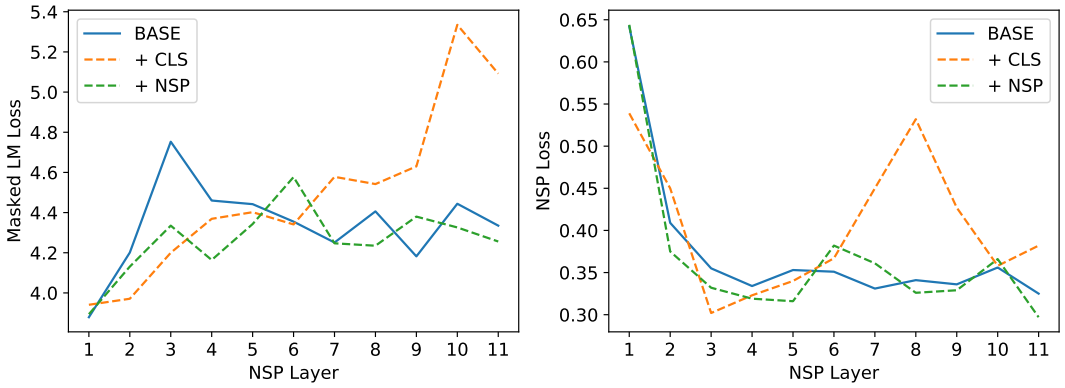


(b) SQuAD2.0

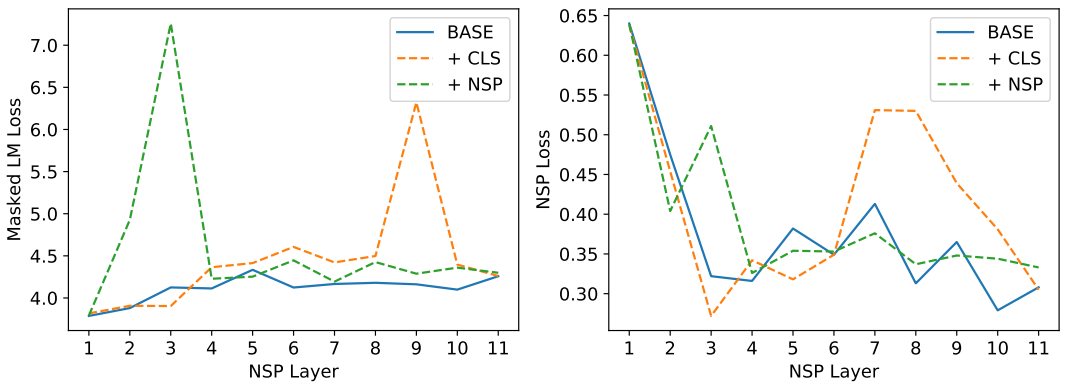


(c) WikiText-2

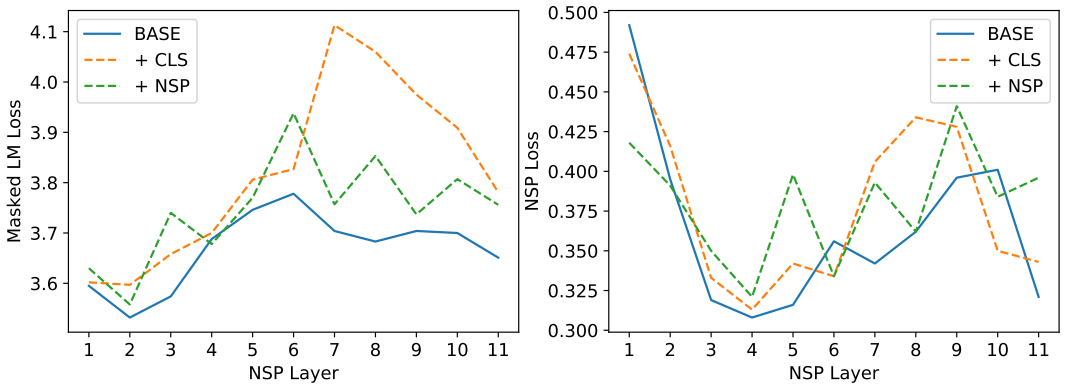
Figure 4.1. NSP-Masked LM loss curves versus mask layer for lower mask architectures for each validation set. For BASE, neither `[CLS]` embedding nor NSP output is used. + `[CLS]` and + NSP indicate the concatenated parts.



(a) SQuAD1.1



(b) SQuAD2.0



(c) WikiText-2

Figure 4.2. NSP-Masked LM loss curves versus NSP layer for lower NSP architectures for each validation set. For BASE, neither `[CLS]` embedding nor NSP output is used. + `[CLS]` and + NSP indicate the concatenated parts.

Table 4.8. Selected layers for lower architectures. PT (pre-training) column represents whether `[CLS]` embedding or NSP output is used in pre-training.

Models	**PT**		**SQuAD1.1**	**SQuAD2.0**	**WikiText-2**
	CLS	NSP			
Lower NSP	-	-	9	10	3
	+	-	3	3	3
	-	+	4	4	4
Lower Mask	-	-	9	11	11
	+	-	9	4	4
	-	+	10	5	4
Lower NSP-Freeze	-	-	9	10	3

4.4. Fine-tuning Results

The best models for each architecture are fine-tuned on downstream tasks (question answering and textual entailment). We add three baselines:

(i) Pre-trained $\text{BERT}_{\text{base}}$ model is fine-tuned.

(ii) $\text{BERT}_{\text{base}}$ architecture without any pre-training is directly fine-tuned to understand whether the performance is due to architecture or large-scale training.

(iii) Another modern architecture due to its success, BiLSTM, is fine-tuned with fastText [31] word embeddings.

Fine-tuning results of QA classifiers on SQuAD1.1 and SQuAD2.0 validation sets are shown in Table 4.11 and 4.12. Here, exact match (EM) measures whether the prediction overlaps with the truth exactly. F-measure (F1) evaluates partial overlaps [25]. An example of the calculations of these metrics is shown in Table 4.13. In this example, the first prediction is the same with the truth, so its EM score is equal to 1.

Table 4.9. Accuracies of pre-trained models on validation sets. SQuAD column represents the pre-training set. Pre-training column represents whether `[CLS]` embedding or NSP output is used in pre-training.

Models	**Pre-training**		**SQuAD**	
	CLS	NSP	SQuAD1.1 NSP / Mask	SQuAD2.0 NSP / Mask
Pre-trained BERT	-	-	95.9 / 60.8	94.7 / 61.0
BERT	-	-	90.4 / 35.6	90.0 / 36.7
	+	-	89.2 / 33.7	89.2 / 36.3
	-	+	87.7 / 35.0	91.0 / 36.5
Lower NSP	-	-	85.8 / 34.5	86.3 / 34.6
	+	-	88.4 / 33.7	89.0 / 36.6
	-	+	90.1 / 35.2	88.7 / 32.6
Lower Mask	-	-	87.1 / 36.6	87.1 / 37.9
	+	-	79.3 / 34.8	87.1 / 36.1
	-	+	85.9 / 36.1	89.1 / 36.0
Lower NSP-Freeze	-	-	88.8 / 34.2	89.4 / 34.4
Without NSP	-	-	- / 38.7	- / 39.3
Bigram Shift	-	-	89.2 / 34.4	87.7 / 32.7

Table 4.10. Accuracies of pre-trained models on validation set. WikiText-2 column represents the pre-training set. Pre-training column represents whether `[CLS]` embedding or NSP output is used in pre-training.

Models	**Pre-training**		**WikiText-2**	
	CLS	NSP	NSP	Mask
Pre-trained BERT	-	-	79.8	56.0
BERT	-	-	87.7	38.7
	+	-	86.8	38.5
	-	+	84.7	38.0
Lower NSP	-	-	87.1	40.5
	+	-	87.4	40.6
	-	+	87.1	38.7
Lower Mask	-	-	84.1	38.1
	+	-	85.6	37.8
	-	+	85.6	38.0
Lower NSP-Freeze	-	-	87.9	41.3
Bigram Shift	-	-	84.3	33.5

Table 4.11. Exact Match (EM) and F-measure (F1) results of QA classifiers on validation sets. SQuAD represents pre-training set. Conc. (concatenation) indicates whether `[CLS]` embedding or NSP output is used in both pre-training and fine-tuning.

Models	**Conc.**		**SQuAD**	
	CLS	NSP	SQuAD1.1 EM / F1	SQuAD2.0 EM / F1
Pre-trained BERT	-	-	75.9 / 85.4	71.1 / 73.8
BERT	-	-	49.5 / 60.6	54.8 / 57.4
	+	-	48.1 / 59.1	56.7 / 59.7
	-	+	48.7 / 60.2	55.5 / 58.5
Lower NSP	-	-	47.2 / 58.3	56.3 / 58.8
	+	-	47.0 / 58.5	55.8 / 58.3
	-	+	52.1 / 63.4	54.8 / 57.3
Lower Mask	-	-	27.5 / 34.0	48.1 / 48.8
	+	-	43.3 / 53.5	47.6 / 48.2
	-	+	45.5 / 56.3	51.2 / 53.4
Lower NSP-Freeze	-	-	40.9 / 51.4	51.7 / 53.4
Without NSP	-	-	30.2 / 37.5	46.9 / 47.5
Bigram Shift	-	-	47.0 / 57.6	55.6 / 58.0
BiLSTM	-	-	22.9 / 29.3	46.4 / 46.7
Without Pre-training	-	-	7.4 / 12.1	51.0 / 51.0

Table 4.12. Exact Match (EM) and F-measure (F1) results of QA classifiers on validation sets. WikiText-2 is the pre-training set. Conc. (concatenation) indicates whether `[CLS]` embedding or NSP output is used in both pre-training and fine-tuning.

Models	**Conc.**		**WikiText-2**	
	CLS	NSP	SQuAD1.1 EM / F1	SQuAD2.0 EM / F1
Pre-trained BERT	-	-	75.9 / 85.4	71.1 / 73.8
BERT	-	-	48.1 / 60.5	54.7 / 57.9
	+	-	48.8 / 60.6	55.0 / 58.0
	-	+	48.2 / 59.7	51.2 / 53.2
Lower NSP	-	-	51.4 / 62.7	54.1 / 55.9
	+	-	46.8 / 57.6	53.6 / 56.6
	-	+	46.0 / 57.6	54.7 / 57.8
Lower Mask	-	-	41.8 / 52.2	47.1 / 47.4
	+	-	25.0 / 31.8	46.9 / 47.4
	-	+	23.7 / 30.2	48.1 / 48.6
Lower NSP-Freeze	-	-	39.7 / 49.5	50.2 / 52.3
Bigram Shift	-	-	34.5 / 45.7	46.0 / 48.4
BiLSTM	-	-	22.9 / 29.3	46.4 / 46.7
Without Pre-training	-	-	7.4 / 12.1	51.0 / 51.0

Table 4.13. Example for exact match and F-measure calculation. TP, FP, FN are used for true positives, false positives, and false negatives respectively.

Question: What could a teacher help in organizing? Paragraph: A teacher's professional duties may extend beyond formal teaching. Outside of the classroom teachers may accompany students on field trips, supervise study halls, help with the organization of school functions, and serve as supervisors for extracurricular activities. In some education systems, teachers may have responsibility for student discipline. Truth: school functions Prediction 1: school functions (start prediction: school, end prediction: functions) Prediction 2: the organization of school (start prediction: the, end prediction: school)
Evaluate Prediction 1
$EM_1 = 1$ TP $= 2$, FP $= 0$, FN $= 0$, recall $= 1$, precision $= 1$, $F1_1 = 1$
Evaluate Prediction 2
$EM_2 = 0$ TP $= 1$, FP $= 3$, FN $= 1$, recall $= 0.5$, precision $= 0.25$, $F1_2 = 0.33$

F1 is calculated for each truth-prediction pair based on the counts of true positives, false positives, and false negatives. True positives are common tokens between truth and prediction. False positives are tokens in prediction, not in truth. False negatives are tokens in truth, not in prediction. F1 is calculated as follows:

$$F1 = \frac{2 * \text{recall} * \text{precision}}{\text{recall} + \text{precision}} \tag{4.1}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.2}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.3}$$

where TP, FN, FP are counts of true positives, false negatives, and false positives, respectively. Recall is the ratio of truly predicted tokens to the truth tokens. Precision is the ratio of truly predicted tokens to the predicted tokens. The EM and F1 scores for the whole dataset are calculated by taking the average of the scores for the samples.

To test the stability of proposed architectures, we made three runs for some of the most successful classifiers which are BERT and Lower NSP architectures. Due to the computational issues, we could not do this for other models. Table 4.14 shows the average EM and F1 scores of BERT and Lower NSP models which are pre-trained on the WikiText-2 set. While SQuAD1.1 results do not differ in performance too much, SQuAD2.0 results have larger standard deviations, most probably due to no-answer examples as they increase the variation.

One obvious result is the underperformance of Lower Mask models. However, Lower NSP models show competitive performance with BERT, even better in some datasets. This might give a clue about the task hierarchy between masked LM and NSP tasks. One might expect that the masked LM task is more straightforward than the NSP task because the former makes predictions in the word-level while the latter makes predictions in the sentence-level. However, the results suggest that the masked LM task is complicated as it requires deeper layers to perform successfully. NSP task can be done without encoding the full knowledge of each word. Unlike RoBERTa,

Table 4.14. Average results of QA classifiers on validation sets. EM and F1 stand for Exact Match and F-measure, respectively. WikiText-2 is the pre-training set. Conc. (concatenation) indicates whether `[CLS]` embedding or NSP output is used.

Models	**Conc.**		**WikiText-2**	
	CLS	NSP	SQuAD1.1 EM / F1	SQuAD2.0 EM / F1
BERT	-	-	47. 9 ± 0.2 / 60.3 ± 0.3	54.2 ± 1.0 / 57.6 ± 1.2
	+	-	49.5 ± 0.6 / 60.6 ± 0.2	54.7 ± 0.5 / 57.4 ± 0.6
	-	+	48.0 ± 0.2 / 59.4 ± 0.3	52.0 ± 1.2 / 53.9 ± 1.4
Lower NSP	-	-	51.7 ± 0.4 / 62.9 ± 0.3	55.5 ± 1.3 / 57.8 ± 1.6
	+	-	47.1 ± 0.3 / 57.8 ± 0.6	54.4 ± 0.7 / 56.8 ± 0.3
	-	+	46.2 ± 0.2 / 57.7 ± 0.4	54.8 ± 0.3 / 57.6 ± 0.2

the performance drops when we remove NSP loss. The reason for this might be pre-training with small data; the model may not need sentence-level task when it has access to many word combinations. Instead of removing NSP loss, one promising direction can be training NSP task at lower layers for the large-scale training. As in Table 4.9, when we remove NSP loss, the masked LM performance increases in pre-training. However, the language model, which is trained with only masked LM objective, is not sufficient for question answering task. NSP task forces the embeddings to contain sentence-level information to make them more effective in fine-tuning.

Bigram Shift model performs slightly worse than other models when the pre-training is done on SQuAD datasets. One possible explanation is that the question answering problem does not require the word order information. On WikiText-2, the results are dramatically worse. This might be due to optimization; a better hyperparameter search can increment the performance. We will show that the Bigram Shift model is more effective for specific NLP tasks in later experiments.

Self-supervised pre-training shows its contribution once more. Even when pre-training is done on the downstream data with a poorly constructed architecture (Lower Mask), it performs better than the original transformer without any pre-training. In Table 4.11, without pre-training model has 51.0 F1 score on SQuAD2.0. However, this is not a good model as it predicts that all question-paragraph pairs have no answer (51% of examples in the validation set have no answer). `[CLS]` token is labeled as the start token and end token for the samples with no answer. Therefore, start and end indices for these samples are 0. To improve the model scores (EM and F1), we set a threshold $T = 1.0$ by grid search as in the original BERT. The sum of the classifiers' maximum logits is compared to the sum of the logits for the `[CLS]` token. If `[CLS]` logits are lower than the maximum logits by at least T, the classifiers predict the indices with the maximum logits; otherwise, the model will predict the sample has no answer.

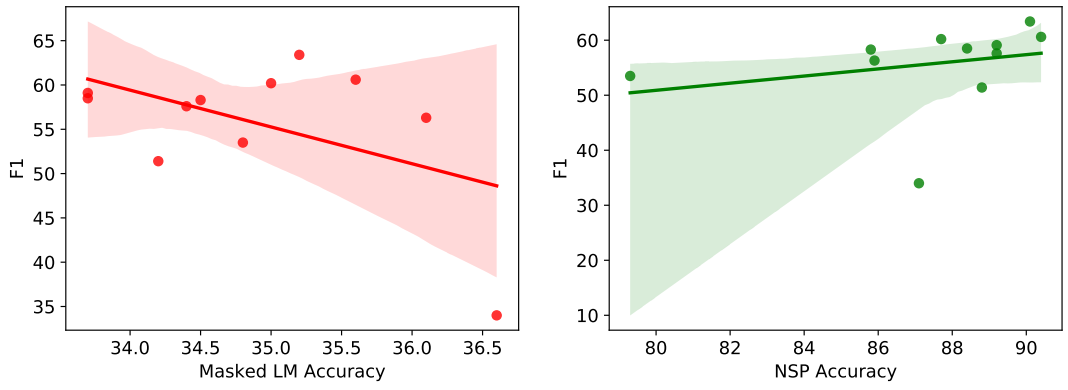
Furthermore, all models outperform BiLSTM with fastText. For this model, the data is created differently. Firstly, samples are tokenized with a basic tokenizer instead of a WordPiece tokenizer. Then, each token is converted into its fastText vector with 300 dimensions. Notice that, fastText is also a model pre-trained on large-scale data. A transformer pre-trained directly on downstream data has better representations for the specific downstream tasks than BiLSTM with fastText. This shows that the transformer learns better language models than BiLSTM with fastText, probably due to its architecture.

Results on the MultiNLI corpus for both matched and mismatched validation sets are shown in Table 4.15. We cannot use concatenated parts for fine-tuning, as we do in question answering. Only `[CLS]` embedding from NSP level is used for fine-tuning. As in QA results, pre-training on small data has a bootstrapping effect compared with the model without pre-training. All models except Lower NSP-Freeze outperforms BiLSTM. This shows that the transformer aggregates sentence-level information better than BiLSTM. This is an interesting result since BiLSTM sees each word explicitly to encode sentence-level embedding while the transformer encodes this implicitly via the NSP task. While BERT has the best performance, Lower NSP is also compet-

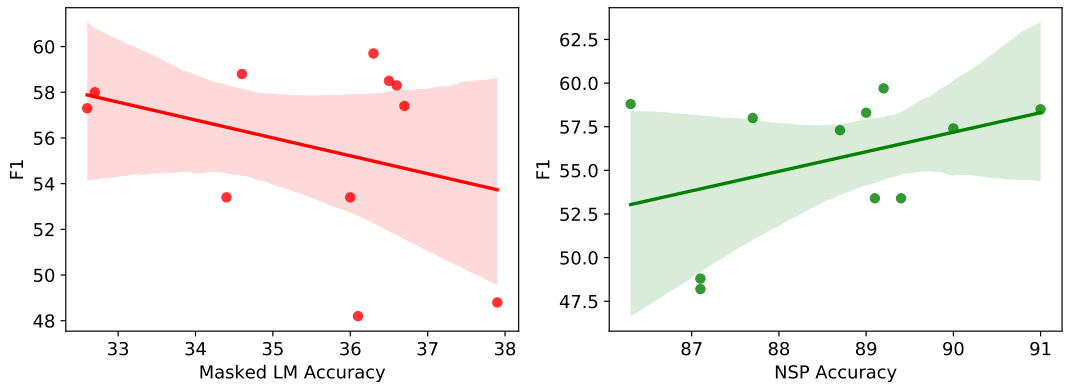
Table 4.15. Accuracies on MultiNLI validation sets. Pre-training column represents whether `[CLS]` embedding or NSP output is used in pre-training.

Models	**Pre-training**		**Accuracy**	
	CLS	NSP	Matched	Mismatched
Pre-trained BERT	-	-	84.4	84.9
BERT	-	-	71.7	72.5
	+	-	70.2	70.8
	-	+	71.1	71.3
Lower Mask	-	-	69.9	70.7
	+	-	68.5	69.2
	-	+	67.1	67.9
Lower NSP	-	-	69.1	70.2
	+	-	70.3	71.9
	-	+	70.5	71.3
Lower NSP-Freeze	-	-	45.2	45.3
Bigram Shift	-	-	69.8	70.7
BiLSTM	-	-	66.8	67.3
Without Pre-training	-	-	61.2	61.4

itive. Unlike question answering, Lower Mask and Bigram Shift models have better performance.



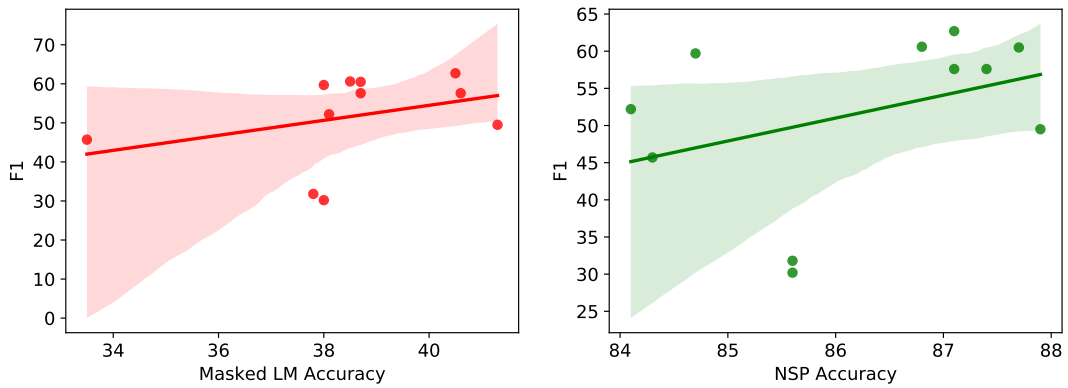
(a) SQuAD1.1



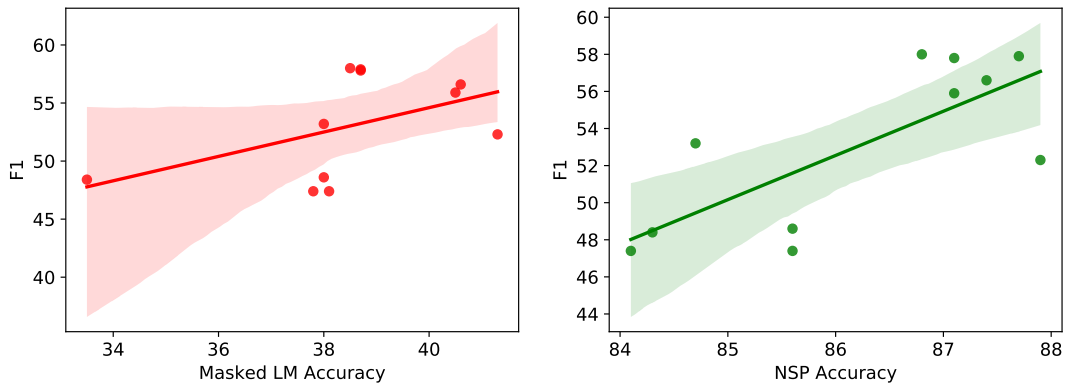
(b) SQuAD2.0

Figure 4.3. Pre-training results versus fine-tuning results for QA classifiers which are pre-trained on SQuAD sets.

Figure 4.3 and 4.4 show the relation between pre-training results and fine-tuning results for QA classifiers. Masked LM accuracy is slightly negatively correlated to the QA classifier score when it is pre-trained on SQuAD sets. However, it is opposite when the model is pre-trained on the WikiText-2 set. The second column in both figures shows that NSP performance positively affects the performance of the QA classifiers. This highlights the importance of the NSP task for a language model.



(a) WikiText-2 - SQuAD1.1



(b) WikiText-2 - SQuAD2.0

Figure 4.4. Pre-training results versus fine-tuning results for QA classifiers which are pre-trained on WikiText-2 set.

4.5. Ablation Studies

Table 4.16. Exact Match (EM) and F-measure (F1) results about using concatenated parts in pre-training (PT) and/or fine-tuning (FT).

Models	**PT**		**FT**		**SQuAD**	
	CLS	NSP	CLS	NSP	SQuAD1.1 EM / F1	SQuAD2.0 EM / F1
BERT	-	-	-	-	49.5 / 60.6	54.8 / 57.4
	-	-	+	-	50.0 / 61.6	55.1 / 58.4
	-	-	-	+	50.4 / 61.7	56.5 / 59.5
	+	-	+	-	48.1 / 59.1	56.7 / 59.7
	-	+	-	+	48.7 / 60.2	55.5 / 58.5
Lower NSP	-	-	-	-	47.2 / 58.3	56.3 / 58.8
	-	-	+	-	47.9 / 59.5	56.7 / 59.6
	-	-	-	+	48.3 / 59.4	56.3 / 59.3
	+	-	+	-	47.0 / 58.5	55.8 / 58.3
	-	+	-	+	52.1 / 63.4	54.8 / 57.3
	+	-	-	-	47.1 / 58.3	55.6 / 58.3
	-	+	-	-	51.0 / 62.8	54.2 / 56.9

To evaluate the effect of concatenated parts (`[CLS]` embedding or NSP output), we also test the following architectures: (1) use in pre-training (2) use in fine-tuning (3) use in both pre-training and fine-tuning (4) not using. QA classifier results for these experiments are shown in Table 4.16 and 4.17. Ablation studies are done for BERT and Lower NSP architectures since these architectures have better performance in fine-tuning. Even if the model is pre-trained without using concatenated parts, using these extra inputs in fine-tuning slightly increases the performance. While the QA classifiers predict whether tokens are the start and end of an answer, using condensed information about the whole sequence may increase the performance. In some cases, the peak performance is achieved by using the concatenated parts in both pre-training

Table 4.17. Exact Match (EM) and F-measure (F1) results about using concatenated parts in pre-training (PT) and/or fine-tuning (FT).

Models	**PT**		**FT**		**WikiText-2**	
	CLS	NSP	CLS	NSP	SQuAD1.1 EM / F1	SQuAD2.0 EM / F1
BERT	-	-	-	-	48.1 / 60.5	54.7 / 57.9
	-	-	+	-	48.0 / 60.7	54.9 / 58.2
	-	-	-	+	47.4 / 60.1	55.6 / 58.9
	+	-	+	-	48.8 / 60.6	55.0 / 58.0
	-	+	-	+	48.2 / 59.7	51.2 / 53.2
Lower NSP	-	-	-	-	51.4 / 62.7	54.1 / 55.9
	-	-	+	-	50.7 / 62.1	54.0 / 57.1
	-	-	-	+	50.6 / 62.3	54.6 / 57.7
	+	-	+	-	46.8 / 57.6	53.6 / 56.6
	-	+	-	+	46.0 / 57.6	54.7 / 57.8
	+	-	-	-	46.9 / 58.0	53.4 / 56.3
	-	+	-	-	46.5 / 58.1	55.7 / 58.4

and fine-tuning. Removing concatenated parts during fine-tuning when used in pre-training, does not change the performance too much. This may be because the token embeddings contain enough information about the whole sequence and do not need explicit sequence-level information.

4.6. Probing Tasks

For probing tasks, a multi-layer perceptron (MLP) with two hidden layers with 128 units is used. Here, the transformer part is frozen, and we only train the probing task classifier. The learning rate is set to be 1e-3 with no weight decay. The batch size is 32. The classifier uses `[CLS]` embedding from the NSP level as input.

The results of probing tasks are shown in Table 4.18 and 4.19. These results suggest the followings:

- In sentence length and tree depth tasks, pre-trained BERT performs worse than some models. These models can create sentence-level embeddings with better syntactic information: the count of tokens and the sentence structure. This is quite surprising considering pre-trained BERT's large-scale training.
- Our models cannot handle the word content task. The models may not be able to distinguish 1000 words determined since they are pre-trained on a small dataset. Another reason is that it may not be possible to retrieve word information from a sentence embedding for our models. Besides, the models are not successful at semantic odd man out task (close to random guess). This may also indicate that sentence embeddings do not contain detailed information about individual words.
- Bigram shift model achieves good results for tasks that require order information (tree depth, bigram shift, coordination inversion). While it is not as good as other models in downstream tasks, including order information might be beneficial for other tasks.
- Lower NSP models perform better than Lower Mask models and better than BERT for some tasks. We can say that Lower NSP constructs more informative

sentence-level representation than the others.

Table 4.18. Accuracies of probing tasks. PT (pre-training) column represents whether `[CLS]` embedding or NSP output is used in pre-training.

Models	**PT**		**SL**	**WC**	**TD**	**BS**	**TC**
	CLS	NSP					
PT. BERT	-	-	68.3	32.4	34.3	86.5	75.2
BERT	-	-	83.8	9.6	36.0	62.5	70.1
	+	-	75.7	10.1	34.6	57.9	63.6
	-	+	84.9	12.4	34.9	60.4	60.7
Lower Mask	-	-	73.2	2.2	29.0	56.1	61.3
	+	-	43.8	1.8	23.5	54.1	30.7
	-	+	41.7	0.8	21.3	52.1	26.7
Lower NSP	-	-	91.0	5.6	31.7	52.9	57.5
	+	-	86.1	11.1	34.5	60.0	67.9
	-	+	90.1	1.1	31.0	57.6	71.8
L. NSP-Freeze	-	-	93.2	1.5	33.7	53.8	72.3
Bigram Shift	-	-	88.6	3.8	37.2	70.3	65.6

Table 4.19. Accuracies of probing tasks. PT (pre-training) column represents whether `[CLS]` embedding or NSP output is used in pre-training.

Models	**PT**		**T**	**SN**	**ON**	**OM**	**CI**
	CLS	NSP					
PT. BERT	-	-	88.7	83.0	77.8	64.3	74.4
BERT	-	-	75.9	74.7	69.9	49.7	59.8
	+	-	70.2	73.8	68.9	50.5	55.6
	-	+	72.2	73.2	69.2	49.9	58.6
Lower Mask	-	-	73.6	73.0	67.1	50.2	56.8
	+	-	70.5	62.2	58.2	49.8	51.5
	-	+	66.6	62.3	58.0	49.9	51.7
Lower NSP	-	-	71.7	71.9	66.4	51.1	56.0
	+	-	74.9	75.4	71.5	50.8	57.2
	-	+	70.9	72.7	65.5	50.3	57.6
L. NSP-Freeze	-	-	75.9	75.2	68.8	50.3	57.2
Bigram Shift	-	-	70.5	72.7	68.2	49.9	59.8

5. CONCLUSIONS

- We propose to pre-train BERT with a hierarchical multitask learning approach. Our results on restricted data (due to computational resources) show that this approach achieves better or equal performance. To provide detailed insights about the pre-training task hierarchies and complexities, all intermediate layers are experimented to choose the best level for lower classifiers. The results show that the masked LM task requires more layers, while the NSP task can be learned with fewer layers.
- Lower NSP models are more successful than the Lower Mask models, and competitive with the original BERT in downstream tasks. Training the NSP classifier with lower-level embeddings leads to better contextualized embeddings for these tasks.
- We incorporate sentence-level information to solve word-level tasks in both pre-training and fine-tuning on question answering task. This also shows a slight increment in performance. For pre-training, it regularizes the sentence-level embeddings with masked LM task. For fine-tuning, it provides information about the whole sequence while making predictions on token-level.
- We propose an additional pre-training task, bigram shift, which causes embeddings to contain word order information. We show that the model pre-trained with this objective performs better than the pre-trained BERT in some probing tasks. Bigram Shift might be beneficial for some specific NLP tasks that require order information.
- According to the results, higher NSP accuracy provides slightly better QA classifiers. Besides, removing the NSP loss during pre-training has a negative effect on question answering task when we have limited data. It shows the importance of the NSP task for pre-training a language model.
- Even if the models are pre-trained with small data, they perform better than the original transformer without any pre-training. It shows the benefit of learning a language model.

- Generally, our models for both question answering and textual entailment outperform BiLSTM with fastText. Transformer architecture is more suitable than BiLSTM to learn a better language model.
- Probing tasks are used to evaluate different types of learned embeddings. These tasks show that different training techniques lead embeddings to contain different linguistic properties. This is an essential point since there are various problems in the NLP domain that require different needs. Therefore selecting an appropriate pre-training strategy is an important factor. For example, for a downstream task that requires word order information, the model which is pre-trained with bigram shift task can be fine-tuned. One surprising result is that some of our proposed models perform better than the pre-trained BERT for some probing tasks. This supports our motivation to frame BERT pre-training as a multitask learning problem.
- Although the English language has vast amounts of data, this is not the case for other languages such as Turkish. For some languages with restricted sources, our proposed Lower NSP architecture and its variants can be used to construct better token and sentence embeddings. Furthermore, for domain-specific problems, the Lower NSP model can be pre-trained directly on the downstream data.

We believe that implementing these techniques to large-scale training will further advance the state-of-the-art. One promising future work is to find a strategy that automatically analyzes task hierarchies and complexities to create a more effective model architecture. This strategy can be tested on training masked LM, NSP, and bigram shift classifiers at the same time. Another future direction is to fine-tune our proposed architectures on various downstream tasks to make a more comprehensive evaluation.

REFERENCES

1. Mikolov, T., K. Chen, G. Corrado and J. Dean, "Efficient estimation of word representations in vector space", *arXiv preprint arXiv:1301.3781*, 2013.

2. Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado and J. Dean, "Distributed representations of words and phrases and their compositionality", *Advances in Neural Information Processing Systems*, pp. 3111–3119, 2013.

3. Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee and L. Zettlemoyer, "Deep contextualized word representations", *arXiv preprint arXiv:1802.05365*, 2018.

4. Hochreiter, S. and J. Schmidhuber, "Long short-term memory", *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

5. Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need", *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

6. Lin, Z., M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou and Y. Bengio, "A structured self-attentive sentence embedding", *arXiv preprint arXiv:1703.03130*, 2017.

7. Radford, A., K. Narasimhan, T. Salimans and I. Sutskever, "Improving language understanding by generative pre-training", *OpenAI Blog https://openai.com/blog/language-unsupervised*, last accessed on 2018.

8. Devlin, J., M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding", *arXiv preprint arXiv:1810.04805*, 2018.

9. Radford, A., J. Wu, D. Amodei, D. Amodei, J. Clark, M. Brundage and I. Sutskever, "Better language models and their implications", *OpenAI Blog https://openai. com/blog/better-language-models*, last accessed on 2019.

10. Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners", *arXiv preprint arXiv:2005.14165*, 2020.

11. Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation", *arXiv preprint arXiv:1609.08144*, 2016.

12. Hinton, G. E. and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks", *Science*, Vol. 313, No. 5786, pp. 504–507, 2006.

13. Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach", *arXiv preprint arXiv:1907.11692*, 2019.

14. Lan, Z., M. Chen, S. Goodman, K. Gimpel, P. Sharma and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations", *arXiv preprint arXiv:1909.11942*, 2019.

15. Caruana, R., "Multitask learning", *Machine Learning*, Vol. 28, No. 1, pp. 41–75, 1997.

16. Zamir, A. R., A. Sax, W. Shen, L. J. Guibas, J. Malik and S. Savarese, "Taskonomy: Disentangling task transfer learning", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3712–3722, 2018.

17. Yang, Z., K. Merrick, H. Abbass and L. Jin, "Multi-task deep reinforcement learning for continuous action control", *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 3301–3307, 2017.

18. Ruder, S., "An overview of multi-task learning in deep neural networks", *arXiv preprint arXiv:1706.05098*, 2017.

19. Søgaard, A. and Y. Goldberg, "Deep multi-task learning with low level tasks supervised at lower layers", *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 231–235, 2016.

20. Hashimoto, K., C. Xiong, Y. Tsuruoka and R. Socher, "A joint many-task model: Growing a neural network for multiple NLP tasks", *arXiv preprint arXiv:1611.01587*, 2016.

21. Sanh, V., T. Wolf and S. Ruder, "A hierarchical multi-task approach for learning embeddings from semantic tasks", *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 6949–6956, 2019.

22. Bingel, J. and A. Søgaard, "Identifying beneficial task relations for multi-task learning in deep neural networks", *arXiv preprint arXiv:1702.08303*, 2017.

23. Prechelt, L., "Early stopping-but when?", *Neural Networks: Tricks of the Trade*, pp. 55–69, Springer, 1998.

24. Conneau, A., G. Kruszewski, G. Lample, L. Barrault and M. Baroni, "What you can cram into a single vector: Probing sentence embeddings for linguistic properties", *arXiv preprint arXiv:1805.01070*, 2018.

25. Rajpurkar, P., J. Zhang, K. Lopyrev and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text", *arXiv preprint arXiv:1606.05250*, 2016.

26. Rajpurkar, P., R. Jia and P. Liang, "Know what you don't know: Unanswerable questions for SQuAD", *arXiv preprint arXiv:1806.03822*, 2018.

27. Williams, A., N. Nangia and S. R. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference", *arXiv preprint arXiv:1704.05426*,

2017.

28. Kingma, D. P. and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

29. Reddi, S. J., S. Kale and S. Kumar, "On the convergence of Adam and beyond", *arXiv preprint arXiv:1904.09237*, 2019.

30. Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen *et al.*, "PyTorch: An imperative style, high-performance deep learning library", *Advances in Neural Information Processing Systems*, pp. 8026–8037, 2019.

31. Bojanowski, P., E. Grave, A. Joulin and T. Mikolov, "Enriching word vectors with subword information", *Transactions of the Association for Computational Linguistics*, Vol. 5, pp. 135–146, 2017.

APPENDIX A: Results of Lower Architectures

Table A.1. NSP-Masked LM losses of lower mask models with mask layers on the SQuAD1.1 validation set. BASE indicates the models in which neither `[CLS]` nor NSP output is used. + CLS and + NSP indicates the models with these extra inputs.

Mask Layer	**Base**		**+ CLS**		**+ NSP**	
	NSP	Masked LM	NSP	Masked LM	NSP	Masked LM
1	0.354	4.452	0.364	4.425	0.384	4.416
2	0.351	4.101	0.373	4.197	0.349	4.122
3	0.324	4.068	0.355	4.082	0.353	4.017
4	0.34	4.039	0.34	4.073	0.338	4.03
5	0.326	4.043	0.332	4.145	0.322	4.008
6	0.345	4.093	0.334	4.079	0.339	4.083
7	0.338	4.064	0.361	4.153	0.352	4.034
8	0.351	3.997	0.373	4.103	0.347	3.997
9	0.37	3.958	0.339	4.047	0.367	3.954
10	0.365	4.14	0.352	4.049	0.334	3.956
11	0.326	4.03	0.347	4.057	0.346	3.97

Table A.2. NSP-Masked LM losses of lower mask models with mask layers on the SQuAD2.0 validation set. BASE indicates the models in which neither `[CLS]` nor NSP output is used. + CLS and + NSP indicates the models with these extra inputs.

Mask Layer	**Base**		**+ CLS**		**+ NSP**	
	NSP	Masked LM	NSP	Masked LM	NSP	Masked LM
1	0.382	4.337	0.438	4.32	0.405	4.342
2	0.359	4.078	0.384	4.094	0.364	4.076
3	0.352	3.95	0.356	3.994	0.341	3.968
4	0.348	3.942	0.351	3.968	0.337	3.937
5	0.363	3.999	0.317	4.011	0.322	3.963
6	0.355	3.937	0.344	4.008	0.341	3.996
7	0.363	3.946	0.366	4.037	0.38	3.873
8	0.368	3.921	0.397	4.008	0.383	3.869
9	0.371	3.888	0.394	3.956	0.379	3.876
10	0.349	3.904	0.366	3.976	0.352	3.895
11	0.334	3.857	0.451	4.159	0.363	3.898

Table A.3. NSP-Masked LM losses of lower mask models with mask layers on the WikiText-2 validation set. BASE indicates the models in which neither `[CLS]` nor NSP output is used. + CLS and + NSP indicates the models with these extra inputs.

Mask Layer	**Base**		**+ CLS**		**+ NSP**	
	NSP	Masked LM	NSP	Masked LM	NSP	Masked LM
1	0.389	4.018	0.391	4.021	0.401	4.014
2	0.349	3.803	0.349	3.764	0.374	3.773
3	0.367	3.769	0.39	3.777	0.378	3.748
4	0.336	3.753	0.335	3.781	0.343	3.716
5	0.356	3.771	0.34	3.774	0.394	3.721
6	0.381	3.704	0.363	3.771	0.378	3.744
7	0.393	3.76	0.391	3.721	0.365	3.755
8	0.396	3.711	0.369	3.77	0.361	3.764
9	0.398	3.672	0.397	3.712	0.348	3.729
10	0.382	3.73	0.39	3.703	0.395	3.665
11	0.34	3.666	0.397	3.767	0.359	3.617

Table A.4. NSP-Masked LM losses of lower NSP models with NSP layers on the SQuAD1.1 validation set. BASE indicates the models in which neither `[CLS]` nor NSP output is used. + CLS and + NSP indicates the models with these extra inputs.

NSP Layer	**Base**		**+ CLS**		**+ NSP**	
	NSP	Masked LM	NSP	Masked LM	NSP	Masked LM
1	0.642	3.879	0.539	3.941	0.644	3.894
2	0.409	4.201	0.45	3.971	0.375	4.131
3	0.355	4.753	0.302	4.2	0.332	4.335
4	0.334	4.46	0.323	4.369	0.319	4.164
5	0.353	4.442	0.34	4.402	0.316	4.343
6	0.351	4.355	0.367	4.341	0.382	4.577
7	0.331	4.251	0.45	4.578	0.361	4.247
8	0.341	4.406	0.532	4.542	0.326	4.235
9	0.336	4.182	0.427	4.63	0.329	4.38
10	0.356	4.444	0.358	5.335	0.366	4.326
11	0.325	4.335	0.382	5.092	0.297	4.256

Table A.5. NSP-Masked LM losses of lower NSP models with NSP layers on the SQuAD2.0 validation set. BASE indicates the models in which neither `[CLS]` nor NSP output is used. + CLS and + NSP indicates the models with these extra inputs.

NSP Layer	**Base**		**+ CLS**		**+ NSP**	
	NSP	Masked LM	NSP	Masked LM	NSP	Masked LM
1	0.64	3.787	0.638	3.817	0.639	3.791
2	0.475	3.88	0.454	3.907	0.404	4.927
3	0.322	4.125	0.272	3.904	0.511	7.258
4	0.316	4.113	0.342	4.365	0.326	4.228
5	0.382	4.334	0.318	4.414	0.354	4.253
6	0.349	4.124	0.349	4.606	0.353	4.446
7	0.413	4.166	0.531	4.422	0.376	4.194
8	0.313	4.18	0.53	4.498	0.337	4.426
9	0.365	4.162	0.439	6.322	0.348	4.288
10	0.279	4.099	0.381	4.396	0.344	4.36
11	0.308	4.258	0.305	4.261	0.333	4.299

Table A.6. NSP-Masked LM losses of lower NSP models with NSP layers on the WikiText-2 validation set. BASE indicates the models in which neither `[CLS]` nor NSP output is used. + CLS and + NSP indicates the models with these extra inputs.

NSP Layer	**Base**		**+ CLS**		**+ NSP**	
	NSP	Masked LM	NSP	Masked LM	NSP	Masked LM
1	0.492	3.595	0.474	3.602	0.418	3.63
2	0.395	3.532	0.416	3.597	0.391	3.558
3	0.319	3.574	0.333	3.658	0.35	3.74
4	0.308	3.688	0.313	3.7	0.321	3.678
5	0.316	3.746	0.342	3.806	0.398	3.77
6	0.356	3.778	0.334	3.827	0.334	3.938
7	0.342	3.704	0.406	4.113	0.393	3.757
8	0.362	3.683	0.434	4.06	0.362	3.853
9	0.396	3.704	0.428	3.975	0.441	3.737
10	0.401	3.7	0.35	3.909	0.384	3.807
11	0.321	3.651	0.343	3.781	0.396	3.756