Efficient Task Adaptation with Normalization

Wenxuan Zhou
University of Southern California
zhouwenx@usc.edu

Abstract

Large pre-trained text encoders like BERT start a new chapter in natural language processing. A common practice to apply pre-trained encoders to sequence classification tasks (e.g., classification of sentences or sentence pairs) is by feeding the embedding of `[CLS]` token (in the last layer) to a task-specific classification layer, and then fine tune the model parameters of pre-trained model and classifier jointly. In this paper, we conduct systematic analysis over several sequence classification datasets to examine the embedding values of `[CLS]` token before the fine tuning phase, and present the *imbalanced gradient* issue—i.e., the gradient of some dimensions may be much larger than others and thus dominate the optimization process. This problem will lead to sub-optimal model performance since the model will overfocus on some features. To solve this problem, we heuristically propose several simple normalization methods, which forces the statistics (variance, gradient norm, etc.) to be similar in each dimension. In experiments, we find that our proposed methods only boosts model performance on some datasets while do not work or even hurt the performance on others. We think our methods heavily depend on the embedding distribution. There is still much work to do in order to get a distribution-agnostic method.

1 Introduction

Transformer-based sentence encoders including BERT (Devlin et al., 2018) and its variants (Joshi et al., 2019; Liu et al., 2019) have achieved tremendous success on a wide range of natural language processing tasks. This demonstrates the power of transfer learning from large pre-trained language model to the downstream tasks with task-specific training data. In the current practice of applying BERT, two model transfer strategies, namely feature extraction and model fine-tuning, are widely adopted. The former one represents the input sequence as a linear combination of all layers in the Transformer model, where the weights are learnable, while the parameters in transformer remain freezed during the training stage of the downstream task. The latter one, as shown to be a more effective option (Peters et al., 2019), stacks a task-specific classifier on top of the pre-trained Transformer network, and updates the model parameters in both the classifier and the Transformer together, towards optimizing the downstream task with its training data. Specifically, for sequence classification, the embedding of the `[CLS]` token in the last layer of Transformer is fed into a fully connected network followed by a softmax classifier.

Despite the successes of fine-tuning pre-trained Transformers like BERT, the detailed mechanisms of how knowledge from pre-trained BERT are transferred to facilitate the downstream tasks is not yet well understood—e.g., whether the information stored in `[CLS]` token embedding can directly apply to the end task is still unclear. In the pre-training stage, the sentence encoder is either trained by next sentence prediction (NSP) or masked language model (MLM), which may introduce inductive bias that is irrelevant to end tasks. Also, during pre-training the input sentences are sampled from large-scale raw corpora, while in fine-tuning, the input sentences are sampled from training set of end tasks. This gap in both training objectives and input distribution may lead to highly skewed embedding distribution in the `[CLS]` token. In some embedding dimensions, the input sentences may be highly variant, while in other dimensions, the input sentences may have similar values. This difference in variance leads to difference in initial gradient – each embedding dimensions are trained with different pace (magnitude of gradient update), and some highly-variant dimensions may dominate the inference stage, and thus hurt the generalization

ability of fine-tuned models.

In this paper, we analyze the distribution of embedding dimensions on BERT model, and propose several methods to explicitly eliminate their difference:

- **Embedding Normalization.** This method is applied to the forward propagation. We first estimate the distribution statistics of the `[CLS]` embedding on the whole training set, then use these information to normalize each training example in fine-tuning.
- **Gradient Normalization.** This method is quite similar to batch normalization (Ioffe and Szegedy, 2015), but is applied to back propagation instead of forward propagation. Given a batch of gradient, it first calculates the average gradient norm of each coordinate in the batch, then scale the gradients towards the average gradient norm of all coordinates. This process ends with all coordinates having similar gradient norms.

We conduct experiments on several GLUE (Wang et al., 2018) text classification tasks and find that our method heavily depends on the embedding distribution.

In the rest of the paper, we first overview the fine-tuning process of BERT in Section 2, then present our analysis of biased embedding distribution and its negative impact in Section 3, and finally introduce a simple yet effective embedding normalization method in Section 4. We conduct experiments over several public datasets (as part of GLUE benchmark) in Section 5.

2 Fine-tuning Pre-trained Encoders for Text Classification

We consider the task of classifying sentences or sentence pairs, which aim to assign labels to the input sequences based on their content. It is one of the most fundamental problems in natural language processing. State-of-the-art methods in text classification take a pre-training fine-tuning process – they first train a text encoder on large-scale corpora in a self-supervised way, then adjust model parameters on end tasks in order to fit with their objectives.

In this paper, we use BERT as the base encoder, which can also represent other transformer-based sentence encoders such as RoBERTa (Liu et al., 2019) and Albert (Lan et al., 2019). BERT takes an input sentence and outputs the contextual embedding of each token. It is developed from the Transformer architecture (Vaswani et al., 2017). The basic unit of BERT is a series of self attention blocks, which are stacked by residual connection:

$$\mathbf{h}^{(l+1)} = \text{LayerNorm}(\mathbf{h}^l + \text{SelfAtt}(\mathbf{h}^l)).$$

Then at the last layer, the hidden state $\mathbf{h}^l$ is fed into a tanh layer to get the contextual embedding, where each embedding dimension is within range [-1, 1]:

$$\mathbf{h} = \tanh(\mathbf{W_f}\mathbf{h}^l + \mathbf{b_f}),$$

where $\mathbf{W_f} \in \mathcal{R}^{d_h \times d_h}, \mathbf{b_f} \in \mathcal{R}^{d_h}$ are pre-trained model parameters.

To enable BERT to do sentence classification, a special token `[CLS]` is padded at the beginning of the input sentence. There have been several strategies to pre-train the `[CLS]` embedding. In BERT and SpanBERT, the `[CLS]` token is trained by next sentence prediction (NSP) objective, which is a binary classification task for predicting whether two segments follow each other. Specifically, given two segments s_1, s_2 and final `[CLS]` embedding $\mathbf{h} \in \mathcal{R}^{d_h}$, NSP is predicted by:

$$\mathbf{P}(s_2 \text{ follows } s_1) = \frac{\mathrm{e}^{\mathbf{w}_1\mathbf{h}+b_1}}{\mathrm{e}^{\mathbf{w}_1\mathbf{h}+b_1} + \mathrm{e}^{\mathbf{w}_2\mathbf{h}+b_2}},$$

where $\mathbf{w}_1, \mathbf{w}_2 \in \mathcal{R}^{d_h}$ and $b_1, b_2 \in \mathcal{R}$ are trainable model parameters. While in RoBERTa, the `[CLS]` token is trained by masked language model (MLM) objective, which requires the model to predict the token of `[CLS]` from the `[CLS]` embedding.

To adapt the `[CLS]` embedding to end tasks, the pre-trained softmax classifier is replaced by a new one to predict the task label in the fine-tuning stage:

$$\mathbf{P}(c|\mathbf{h}) = \text{Softmax}(\mathbf{w}_c\mathbf{h} + b_c),$$

where $\mathbf{w}_c \in \mathcal{R}^{d_k}, b_c \in \mathcal{R}$ are random initialized model parameters. Then, all model parameters are jointly optimized using the end task data and the corresponding loss functions.

Although this fine-tuning method is very simple, it actually achieves the best performance. In some empirical studies (Sun et al., 2019), people tried several more advanced fine-tuning strategies, such as stacking the last 3 layers, weighted the hidden states in each layer by attention, but none of them outperforms the above method.

3 Visualizing Embedding Distribution

We conduct systematic analysis over four sequence classification datasets from GLUE (Wang et al., 2018) in this section to present the observations on embedding distribution and discuss its negative impact on the model performance.

To train neural networks from scratch, a common practice is to initialize model parameters by zero-centered i.i.d. random variables. It helps keep variances of each layer and each dimension in similar scale, which keeps the information flowing in forward propagation. Previous work (Sutskever et al., 2013) shows that if the model parameters are not carefully initialized, neural models would reach sub-optimal performance or even fail to learn.

However, this problem does not get much attention in the context of fine-tuning neural models. During fine-tuning, the bottom network is initialized by a pre-trained neural network specialized in a different task and trained on different data distribution. This dismatch of model parameters may lead to unexpected optimization step, and thus hurt the model performance.

3.1 Back Propagation in the Last Two Layers

To better understand the influence of the embedding distribution, we need to know its role in gradient calculation. Here, we only focus on the gradient of the last two layers. Recall the forward propagation is:

$$\begin{aligned}\mathbf{h} &= \tanh(\mathbf{W_f}\mathbf{h}^l + \mathbf{b_f}),\\ \mathbf{y} &= \text{Softmax}(\mathbf{W}_c\mathbf{h} + \mathbf{b_c}),\\ \mathcal{L} &= -\hat{\mathbf{y}}\log(\mathbf{y}).\end{aligned}$$

where $\mathbf{y}$, $\hat{\mathbf{y}}$ is the predicted probability and ground truth, respectively. Then the backward propagation is calculated by:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\mathbf{h}} &= (\mathbf{y}-\hat{\mathbf{y}})\mathbf{W_c}^T,\\ \frac{\partial\mathcal{L}}{\partial\mathbf{W_c}} &= (\mathbf{y}-\hat{\mathbf{y}})\mathbf{h}^T,\\ \frac{\partial\mathcal{L}}{\partial\mathbf{h}^l} &= \left((\mathbf{y}-\hat{\mathbf{y}})\mathbf{W_c}^T \circ (1-\mathbf{h}^2)\right)\mathbf{W_f}^T,\\ \frac{\partial\mathcal{L}}{\partial\mathbf{W_f}} &= \left((\mathbf{y}-\hat{\mathbf{y}})\mathbf{W_c}^T \circ (1-\mathbf{h}^2)\right)(\mathbf{h}^l)^T.\end{aligned}$$

For weights $\mathbf{W_f}$ and $\mathbf{W_c}$, they are either pre-trained or random-initialized, thus is independent to the input sentences. However, $\mathbf{h}$ and $\mathbf{h}^l$, which are generated by the pre-trained encoder, will play an important role in training:

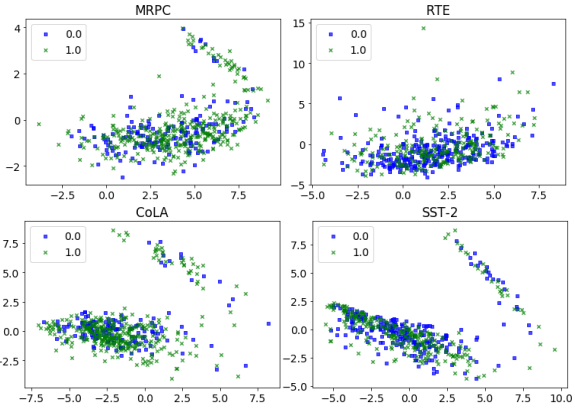

Figure 1: Visualization of the distribution of $\mathbf{h}^l$ on the training set of different fine-tuning tasks by t-SNE. Different colors stand for different class labels.

- **Influence of $\mathbf{h}$.** $\mathbf{h}$ is highly relevant to the gradient of $\mathbf{W_c}$. One problem is that **the gradient does not necessarily focus on relevant dimensions**. For example, assume that the first dimension of $\mathbf{h}$ follows a random uniform distribution within range $[0, 1]$ while the second dimension is equal to $0.01(\mathbf{y}-\hat{\mathbf{y}})$. Then the expected gradients of dimension 1 is $0.5\,\mathbb{E}[(\mathbf{y}-\hat{\mathbf{y}})]$, while the expected gradients of dimension 2 is $0.01\,\mathbb{E}[(\mathbf{y}-\hat{\mathbf{y}})^2]$. That is, when $\mathbb{E}[(\mathbf{y}-\hat{\mathbf{y}})]$ is large enough, the model is incapable of finding relevant dimensions.

- **Influence of $\mathbf{h}^l$.** $\mathbf{h}^l$ is highly relevant to $\mathbf{h}$ in the forward propagation. Large $\mathbf{h}^l$ may saturate the tanh function, and thus leads to **gradient vanishing** (the $1-\mathbf{h}^2$ term in back propagation).

Above analysis is based on SGD with momentum and only considers the last two layers, which may not apply to the real case, where the model is jointly optimized by Adam optimizer. However, it does bring some insights on what kind of normalization is desirable.

3.2 Visualization of $\mathbf{h}^l$

To see whether BERT suffers from problems mentioned above, we visualize the distribution of `[CLS]` embedding of BERT-base on 4 text classification datasets, and show the average value of each embedding dimension in Figure 1 and Figure 2. We observe that the pre-trained `[CLS]` embedding of new datasets fall into two clusters, because the

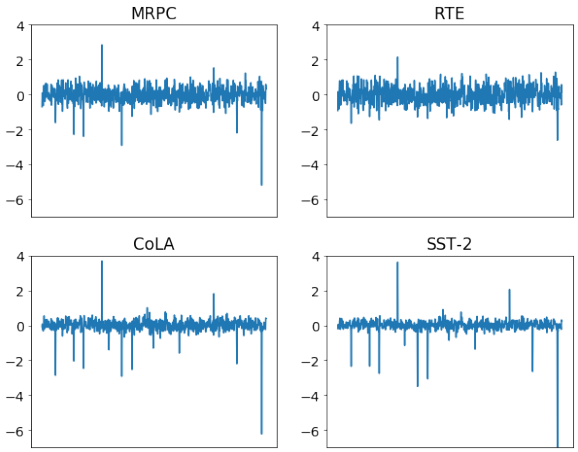

Figure 2: Average value of each $\mathbf{h}^l$ dimension on the training set. Some dimensions have very biased values.

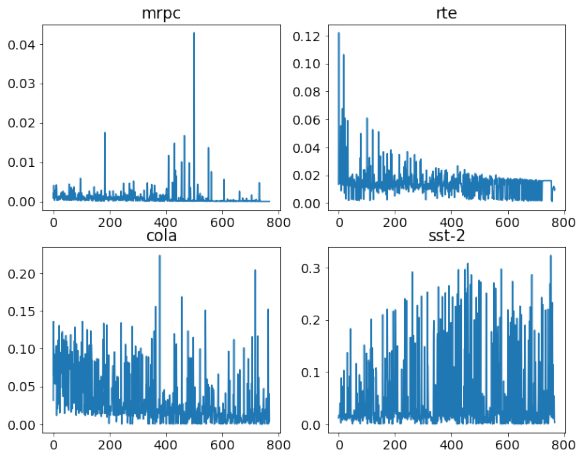

Figure 3: Standard deviation of each [CLS] embedding dimension on the training set. The dimensions are ranked by their importance (computed by Fisher score) from left to right.

[CLS] embedding is trained with next sentence prediction (NSP) objective in BERT. This shows the distribution of [CLS] embedding is skewed by the NSP task in pre-training. In terms of each embedding dimension, we find that most dimensions has an average value within 0.5, while some dimensions have very large average values. In forward propagation, these dimension may saturate the tanh activation function, and thus lead to gradient vanishing problem.

3.3 Visualization of h

To see whether the softmax layer can focus on the relevant coordinates, we rank the importance of each [CLS] embedding dimension of BERT-base by Fisher score on 4 text classification datasets, and show the variance of each embedding dimension in Figure 3. The fisher score of the j^{th} dimension is calculated by:

$$F\left(\mathbf{x}^j\right) = \frac{\sum_{k=1}^{c} n_k \left(\mu_k^j - \mu^j\right)^2}{\left(\sigma^j\right)^2} \qquad (1)$$

where $\left(\sigma^j\right)^2 = \sum_{k=1}^{c} n_k \left(\sigma_k^j\right)^2$. This score measures whether the feature can distinguish different classes. We observe that in all datasets, the variances of different dimensions are quite different, and large variances do not correspond to important dimensions generally. For example, in the MRPC dataset, one dimension has far higher variance than others, but provides little information for classification. Thus, with i.i.d. random initialized model parameters, this dimension will be very likely to dominate model prediction, and thus hurts model performance. This unbalanced distributions of variances motivate us to explicitly re-weigh each dimension in fine-tuning.

Another phenomenon we notice is that the standard deviation of datasets naturally falls into 3 types: (1) In MRPC, most dimensions have small variance, only some dimensions contain large amounts of noise. (2) In CoLA, SST-2, all dimensions have large variances, and there is no clear relationship between variance and feature importance. (3) In RTE (also MNLI, QNLI, WNLI), the feature variance is positively related to the importance – the more variant the distribution is, the more important this feature is. The different types help us to better understand the difficulty of developing a universal fine-tuning method – the method must be able to automatically adjust to different embedding distribution.

4 Proposed Methods and Variants

To solve the above problem, we propose to explicitly normalize the embedding of [CLS] in fine-tuning. Our major objectives are 1) eliminating large numeric values in embedding to avoid gradient vanishing and 2) normalizing the gradients in case of some dimensions dominating the training process. We divide our methods into two groups: embedding-based normalization and gradient-based normalization.

4.1 Embedding Normalization

Embedding normalization is motivated by computer vision. In embedding normalization, we test

the following widely-used data normalization methods:

- **Z-normalization**. Z-normalization (Goldin and Kanellakis, 1995) transforms the input vector into the output vector whose mean is near to 0 and standard deviation is near to 1. Specifically, given a set of input vectors $\mathcal{X} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$, Z-normalization computes the statistics of the whole dataset and transform each training example by:

$$\begin{aligned} \boldsymbol{\mu} &= \mathbb{E}_{\mathbf{x}\in\mathcal{X}}\left[\mathbf{x}\right], \\ \boldsymbol{\sigma}^2 &= \mathbb{E}_{\mathbf{x}\in\mathcal{X}}\left[(\mathbf{x}-\boldsymbol{\mu})^2\right], \\ \hat{\mathbf{x}}_\mathbf{i} &= \frac{\mathbf{x_i}-\boldsymbol{\mu}}{\boldsymbol{\sigma}+\epsilon}, \end{aligned} \tag{2}$$

  where $\boldsymbol{\mu}, \boldsymbol{\sigma}$ are vectors representing the mean and standard deviation of each embedding dimension, ϵ is a hyper-parameter to prevent small denominators. This method is also explored in fine-tuning setting (Varno et al., 2019), where it is theoretically proved to be effective for linear classifiers.

- **Min-Max Normalization**. Min-Max normalization linearly transforms each dimension of the input vectors to a range from -1 to 1:

$$\begin{aligned} \boldsymbol{\theta_1} &= \min_{\mathbf{x}\in\mathcal{X}} \mathbf{x} \\ \boldsymbol{\theta_2} &= \max_{\mathbf{x}\in\mathcal{X}} \mathbf{x} \\ \hat{\mathbf{x}}_\mathbf{i} &= \frac{2\mathbf{x_i}-\boldsymbol{\theta_1}-\boldsymbol{\theta_2}}{\boldsymbol{\theta_2}-\boldsymbol{\theta_1}+\epsilon}, \end{aligned} \tag{3}$$

  where $\min$ and $\max$ are element-wise minimum and maximum value of each dimension on $\mathcal{X}$. By stretching or compressing the embedding, this method ensures that each dimension is in the same scale and no extreme value occurs in the training set. This method is commonly used in computer vision to preprosess input images into unit scales.

- **L2 normalization**. L2 normalization transforms the L2 norm of each embedding dimension to an average value of 1:

$$\begin{aligned} \boldsymbol{\delta}^2 &= \mathbb{E}_{\mathbf{x}\in\mathcal{X}}\left[\mathbf{x}^2\right], \\ \hat{\mathbf{x}}_\mathbf{i} &= \frac{\mathbf{x_i}}{\boldsymbol{\delta}+\epsilon}. \end{aligned} \tag{4}$$

  This method constrains the L2 norm of each embedding not to be too large so as to ease the gradient exploding problem.

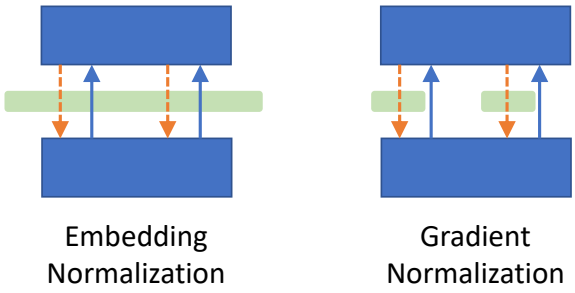

Figure 4: Comparison between embedding normalization and gradient normalization. In embedding normalization, normalization in the forward propagation also changes the calculation of back propagation. While in gradient normalization, only the backward propagation is changed.

4.2 Model Fine-tuning and Prediction with Normalized Embedding

In fine-tuning, we first get the `[CLS]` embeddings of each training example with pre-trained sentence encoder, then use them to calculate the normalization statistics ($\boldsymbol{\mu}$, $\boldsymbol{\sigma}$, etc.). In training phase, we use the normalization statistics to transform the `[CLS]` embedding $\mathbf{h}$ of input sentences to normalized embedding $\hat{\mathbf{h}}$, which are taken as inputs for upper layers. For example, with L2 normalization, the output of the neural network becomes:

$$\begin{aligned} \hat{\mathbf{h}} &= \frac{\mathbf{h}}{\boldsymbol{\delta}+\epsilon}, \\ \mathbf{k} &= \text{FFN}(\hat{\mathbf{h}}), \\ \mathbf{P}(c|\mathbf{h}) &= \text{Softmax}(\mathbf{w}_c\mathbf{k}+b_c), \end{aligned}$$

where $\boldsymbol{\delta}$ is a normalization statistics. Although the parameters of the pre-trained model is updated in fine-tuning, we do not update the normalization statistics for simplicity. In prediction phase, the `[CLS]` embeddings of new examples are normalized in a similar way, using the normalization statistics of the training set.

4.3 Gradient Normalization

Gradient normalization is inspired by the weighted multi-task learning problems. The goal of gradient normalization is to ensure the gradients of each dimension are in similar magnitude. The motivation is that tuning a set of features generalizes better than tuning a few features. Specifically, given the gradient $\mathbf{G}$ of a batch $\mathcal{B}$, the normalized gradients

Method / Task	MRPC	RTE	SST-2
BERT-base	88.2 ± 0.5 / 88.1 / 89.2	71.1 ± 1.1 / 71.5 / 72.2	91.9 ± 0.9 / 91.4 / **92.9**
BERT-base + Z-normalization	88.6 ± 0.3 / 88.4 / 88.9	69.4 ± 2.0 / 69.7 / 72.2	92.2 ± 0.1 / 92.2 / 92.3
BERT-base + Min-Max normalization	88.7 ± 0.4 / 88.6 / 89.3	72.0 ± 0.5 / 71.8 / 72.9	**92.2 ± 0.4** / 92.3 / 92.8
BERT-base + L2 normalization	88.7 ± 0.3 / 88.6 / 89.3	**72.4 ± 0.5 / 72.6** / 73.3	92.2 ± 0.3 / 92.2 / 92.5
BERT-base + Gradient Normalization ($\alpha = 0.5$)	88.5 ± 0.5 / 88.5 / 88.8	72.5 ± 0.9 / 72.2 / 74.0	92.2 ± 0.4 / 92.3 / 92.7
BERT-base + Gradient Normalization ($\alpha = 1.0$)	**88.7 ± 0.5 / 88.8 / 89.3**	72.4 ± 2.0 / 71.8 / **74.7**	92.2 ± 0.4 / 92.3 / 92.8
BERT-large-wwm	88.4 ± 0.8 / 88.9 / 89.1	73.4 ± 1.8 / 72.9 / 75.8	94.4 ± 0.2 / 94.4 / 94.6
BERT-large-wwm + Z-normalization	88.1 ± 1.3 / 88.6 / 89.3	74.4 ± 1.7 / 74.7 / 76.2	93.7 ± 0.3 / 93.7 / 93.9
BERT-large-wwm + Min-Max normalization	**88.8 ± 0.5 / 88.9** / 89.5	73.7 ± 1.4 / 74.0 / 74.7	93.7 ± 0.3 / 93.5 / 94.0
BERT-large-wwm + L2 normalization	88.6 ± 1.0 / 88.8 / **90.0**	**74.8 ± 1.2 / 74.7 / 76.2**	94.3 ± 0.2 / 94.4 / 94.5

Table 1: Results (mean±std, median, and max) on the dev sets of GLUE from 5 runs with different random seeds.

is calculate by:

$$\begin{aligned}\boldsymbol{\mu} &= \mathbb{E}_{\mathbf{g}\in\mathbf{G}}\left[||\mathbf{g}||_2\right], \\ \theta &= \text{Avg}(\boldsymbol{\mu}), \\ \boldsymbol{\gamma} &= \left(\frac{\theta}{\boldsymbol{\mu}+\epsilon}\right)^{\alpha} \\ \mathbf{g}' &= \mathbf{g}\boldsymbol{\gamma}.\end{aligned}$$

Where α is a hyper parameter that controls the strength of gradient normalization. When $\alpha = 1$, the gradients of all coordinates are forced to be the same. When $\alpha = 0$, the gradient normalization is turned off. Different from embedding normalization, the gradient statistics are dynamically computed from the current batch.

4.4 Comparison between Embedding Normalization and Gradient Normalization

The major difference between these two methods is shown in Figure 4. Embedding normalization changes both forward and backward propagation, while gradient normalization only changes the back propagation. This makes gradient normalization a better choice, since it can modify the model in a more "controlled" way. The other reason is that in embedding normalization, it is hard to take the random-initialized weight $\mathbf{W_c}$ into account, which also has a big impact on the model performance. While in gradient normalization, $\mathbf{W_c}$ appears in the gradient, thus our normalization also depends on the weight initialization.

5 Experiments

5.1 Experiment Settings

We implemented our methods on a BERT implementation [1]. The model is optimized with AdamW optimizer using a learning rate of 5e-5 for BERT-base and 2e-5 for BERT-large. The learning rate is scheduled by a linear warmup for the first 6% of steps followed by a linear decay to 0. The model is fine-tuned for 10 epochs on each task. We apply early stopping according to task-specific metrics on the dev set. Other hyper-parameters are same as pre-training.

[1] https://github.com/huggingface/transformers

Method / Task	MNLI-m	MNLI-mm	QNLI
BERT-base	84.21 / 84.29	84.56 / 84.79	91.25 / 91.38
Z-normalization	84.01 / 84.05	84.67 / 84.91	91.14 / 91.34
Min-Max normalization	83.79 / 83.89	84.27 / 84.40	91.27 / 91.36
L2 normalization	**84.52 / 84.66**	**84.91 / 85.40**	**91.38 / 91.58**

Table 2: Results (median, max) on the dev sets from 3 runs with different random seeds. We use BERT-base as the sentence encoder.

5.2 Main Results

The experiment results are presented in Table 1. Overall, our methods achieve consistent improvements over fine-tuning with the BERT-base encoder. Among three normalization methods, L2 normalization and gradient normalization is the most effective one, while Min-Max normalization and Z-normalization sometimes given worse results. We think it is because that some dimensions have very small variance. Z-normalization and Min-Max normalization may greatly stretch the embedding distribution and amplify the noise in embedding. It can be solved with a more careful selection of ϵ. With the BERT-large encoder, L2 normalization still helps, but the performance gains are much less. We also conduct experiments on MNLI (433k sentences) and QNLI (130k sentences) datasets with the BERT-base encoder. Results are presented in Table 2. Again, L2 normalization improves model performance on both datasets.

6 Conclusion

In this paper, we analyzed the embedding distribution of `[CLS]` token in pre-trained sentence encoders. We showed that this problem may lead to issues such as gradient vanishing and dominate features. To solve the problem, we proposed several normalization techniques, which demonstrate their effectiveness on the BERT-base model. Future work includes finding the theoretical foundations of our methods, and develop a method that works for all models and datasets.

References

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

Dina Q Goldin and Paris C Kanellakis. 1995. On similarity queries for time-series data: constraint specification and implementation. In *International Conference on Principles and Practice of Constraint Programming*, pages 137–153. Springer.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2019. Spanbert: Improving pre-training by representing and predicting spans. *arXiv preprint arXiv:1907.10529*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Matthew Peters, Sebastian Ruder, and Noah A Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987*.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? *arXiv preprint arXiv:1905.05583*.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

Farshid Varno, Behrouz Haji Soleimani, Marzie Saghayi, Lisa Di Jorio, and Stan Matwin. 2019. Efficient neural task adaptation by maximum entropy initialization. *arXiv preprint arXiv:1905.10698*.

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

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.