Abstract

Cross-entropy loss is a common choice when it comes to multiclass classification tasks and language modeling in particular. Minimizing this loss results in language models of very good quality. We show that it is possible to fine-tune these models and make them perform even better if they are fine-tuned with sum of cross-entropy loss and reverse Kullback-Leibler divergence. The latter is estimated using discriminator network that we train in advance. During fine-tuning we can use this discriminator to figure out if probabilities of some words are overestimated and reduce them in this case. The novel approach that we propose allows us to reach state-of-the-art quality on Penn TreeBank: perplexity of the fine-tuned model drops down by more than 0.5 and is now below 54.0 in standard evaluation setting; however, in dynamic evaluation framework the improvement is much less perceptible. Our fine-tuning algorithm is rather fast and requires almost no hyperparameter tuning. We test it on different datasets including WikiText-2 and large-scale dataset. In the former case we also reach state-of-the-art results.

1 Introduction

Normally, when it is necessary to fine-tune a model on the same dataset that was used for initial training of this model, various optimization tricks are applied. For instance, one may think about changing optimizer type (in papers by Yang et al. [2017] and Merity et al. [2017] language models that reach state-of-the-art performance are fine-tuned using Averaged Stochastic Gradient Descent (Polyak & Juditsky [1992]). It is also possible to use complex learning rate schedules: for example, to increase learning rate for several epochs (it can help to get out of local minima or saddle points) or to use cyclical learning rates (Smith [2015]).

In our paper we describe another way that can help in fine-tuning models that are designed to solve multiclass classification tasks by minimizing cross-entropy loss. We show that it is possible to use new loss function (namely, sum of cross-entropy loss and reverse Kullback-Leibler divergence) and fine-tune the initial model to optimize this new loss. We prove that such fine-tuning allows to reduce probabilities of words that for some reason were overestimated during initial training.

The described scheme needs careful estimation of reverse Kullback-Leibler divergence (KL-divergence). For this purpose we first train discriminator neural network that has the same architecture as the initial model. Its outputs help to estimate new loss that is used for fine-tuning. Although this approach was inspired by generative adversarial networks (GANs by Goodfellow et al. [2014]), our algorithm has very little in common with this type of generative networks. We do not fine-tune the initial language model to “fool” the discriminator – the discriminator is only used to estimate reverse KL-divergence. Loss functions for language model and discriminator are very different in contrast with GANs where generator and discriminator share the same value function and play minimax game. Also, our fine-tuning procedure is plain and stable. We need to train discriminator till optimality and then fine-tune the initial model till optimality, and it’s sufficient to do this only once.
Our main contributions are: 1) we propose a novel approach of fine-tuning language models by adding a special term to standard cross-entropy loss function, 2) we achieve state-of-the-art results on popular language modeling benchmarks Penn Treebank (PTB, Marcus et al. (1993)) and WikiText-2 (WT2, Merity et al. (2016)), 3) we show that our fine-tuning algorithm is relatively fast, easy to train and applicable to large-scale datasets, 4) we explain theory that lies behind the optimization scheme that we propose.

2 Fine-tuning of language models

Standard language modeling task is to predict the next word \(w_N\) given its left context \(c = \{w_1 \ldots w_{N-1}\}\). One of the most effective ways to do that is to train a recurrent neural network, usually LSTM (Hochreiter & Schmidhuber (1997)) or GRU (Cho et al. (2014)), to minimize cross-entropy loss between real data distribution \(p\) and distribution generated by the neural network \(q\):

\[
CE(p||q) = - \sum_{w \in W} p(w) \log q(w),
\]

(1)

where \(W\) is the vocabulary of all possible words language model can predict. Cross-entropy is widely used because this loss function is the expectation under probability measure \(p\) of negative log-likelihood of \(q\). So, minimizing it makes sense from the point of view of statistics and, which is also very important, cross-entropy is easy to estimate – we can just sample from \(p\) and evaluate negative log-likelihood of \(q\) on these samples.

Minimizing cross-entropy is equivalent to minimizing KL-divergence because they differ by a constant term (by entropy of real data distribution \(p\)):

\[
KL(p||q) = \sum_{w \in W} p(w) \log \frac{p(w)}{q(w)}
\]

(2)

Cross-entropy and KL-divergence are known to be asymmetric – since they both are expectations of some expressions under measure \(p\), they pay less attention to the regions where \(p\) is very small. For language modeling it means that probabilities under measure \(q\) of very rare words (and also of some frequent words that appear very rarely in certain contexts) may be incorrect.

This drawback can be handled with if we consider reverse KL-divergence, i.e. KL-divergence between neural network distribution \(q\) and real data distribution \(p\):

\[
KL(q||p) = \sum_{w \in W} q(w) \log \frac{q(w)}{p(w)}
\]

(3)

Along with correcting the mentioned above issue we still want good statistical properties of the language model (i.e. we want likelihood of \(q\) to be high). This leads to the following scheme: 1) we train a language model to minimize \(CE(p||q)\), 2) then we fine-tune the language model to minimize \(CE(p||q) + KL(q||p)\).

However, in order to implement the second step we need to estimate reverse KL-divergence. The problem is that it is expectation under \(q\) of some expression depending on \(p\) which is intractable – the only thing we can do with \(p\) is to sample from it.

This discussion resembles the one about generative models, especially about GANs (e.g. by Arjovsky & Bottou (2017)). So, we decided to borrow the idea of training a discriminator from the paper where GANs were first introduced (Goodfellow et al. (2014)).

2.1 Training discriminator

The key idea is that if we have a well-trained discriminator, we can then estimate \(p\) in reverse KL-divergence since we know (see Proposition 1 in Goodfellow et al. (2014)) that the optimal discriminator output distribution is a rational expression of \(p\) and \(q\). The same idea has recently been used for generator samples selection in GANs (Azadi et al. (2018)).

We consider language modeling task, so the distributions \(p\) and \(q\) that we spoke about previously are actually conditional probability distributions \(p(\cdot|c)\) and \(q(\cdot|c)\) – probabilities of the next word
given its left context $c$. So, we want our discriminator to output \textit{conditional} probability that the next word comes from neural network distribution $q$. Let us denote this distribution by $r_{\varphi}(\cdot | c)$ where $\varphi$ are the parameters of the discriminator. The architecture of discriminator is exactly the same as the architecture of the language model, only the interpretation of logits change – in the language model softmax function is applied to them to get the probability of the next word $q(\cdot | c)$ whereas in the discriminator we apply sigmoid function to the logits to get the probability $r_{\varphi}(\cdot | c)$ that the next word was generated by the language model rather than came from real distribution $p(\cdot | c)$.

Loss function for discriminator $D(c, \varphi)$ is the same as the one used in standard GANs and is quite naturally given by expressions

$$D(c, \varphi) = D^{(q)}(c, \varphi) + D^{(p)}(c, \varphi),$$  \hfill (4)

$$D^{(q)}(c, \varphi) = -\mathbb{E}_{w \sim q(\cdot | c)}[\log r_{\varphi}(w | c)], \quad D^{(p)}(c, \varphi) = -\mathbb{E}_{w \sim p(\cdot | c)}[\log (1 - r_{\varphi}(w | c))],$$  \hfill (5)

where $q_0$ is the probability distribution given by the initial language model (well-trained to minimize cross-entropy loss). We put zero subscript here to underline that the parameters of this language model are fixed. We will stick to this notation for the rest of the paper.

In practice, for any real context $c$ and any true next word $w^*$ from the training corpus we will estimate this loss by the following expression:

$$\hat{D}(c, w^*, \varphi) = - \sum_{w \in W} q_0(w | c) \log r_{\varphi}(w | c) - \log (1 - r_{\varphi}(w^* | c))$$  \hfill (6)

As for left contexts $c$, we train discriminator only on examples of real contexts, so the overall loss function $D(\varphi)$ that we actually minimize is given by $D(\varphi) = \mathbb{E}_{c \sim p(\cdot)}[D(c, \varphi)]$.

It is worth mentioning that discriminator has to decide whether the next word comes from $p$ or $q$ given not only its left context $c$ (which is clear from notation) but also given the knowledge that this context $c$ comes from real distribution $p$. It becomes clear when we think of how we train this discriminator – we always observe only real contexts $c$ while training. We design discriminator training in this manner because our final goal is to fine-tune language model whereas discriminator plays an auxiliary role of delivering an approximation to real data distribution $p(\cdot | c)$, and we need to estimate this distribution only for real contexts $c$.

### 2.2 Fine-tuning with discriminator

If the discriminator was trained till optimality, the following equation holds (see proof in Goodfellow et al. (2014)):

$$r(w | c) = \frac{q_0(w | c)}{q_0(w | c) + p(w | c)}$$  \hfill (7)

We omit the subscript $\varphi$ here for simplicity. From this equation we can get an estimation for real data distribution:

$$\hat{p}(w | c) = q_0(w | c) \frac{1 - r(w | c)}{r(w | c)}$$  \hfill (8)

Language model loss during fine-tuning is given by

$$L(c, \theta) = CE(p(\cdot | c)||q_\theta(\cdot | c)) + KL(q_\theta(\cdot | c)||p(\cdot | c)),$$  \hfill (9)

where $\theta$ are parameters of the fine-tuned model. Cross-entropy term is calculated as usual. As for reverse KL-divergence, we can calculate it in two different ways since the following identity holds:

$$KL(q_0(\cdot | c)||p(\cdot | c)) = \mathbb{E}_{w \sim q_0(\cdot | c)} \left[ \log \frac{q_0(w | c)}{p(w | c)} \right] = \mathbb{E}_{w \sim p(\cdot | c)} \left[ \frac{q_0(w | c)}{p(w | c)} \log \frac{q_0(w | c)}{p(w | c)} \right]$$  \hfill (10)

Plugging in the estimation for $p$ from (8) into either left or right expectation, we obtain two possible estimations of total loss for any real context $c$ and true next word $w^*$:

$$t(c, w, \theta) = \frac{q_0(w | c)}{q_0(w | c)} \cdot \frac{r(w | c)}{1 - r(w | c)}$$  \hfill (11)
Table 1: Discriminator quality (PTB). Marks \((p)\) and \((q)\) mean the same as corresponding superscripts in (5).

| Training Option                        | Validation loss | Test loss | Test loss \((p)\) | Test loss \((q)\) |
|----------------------------------------|-----------------|-----------|-------------------|-------------------|
| No dropout, no regularization          | 1.379775        | 1.380679  | 0.69229           | 0.68839           |
| No dropout, with regularization        | 1.379754        | 1.380646  | 0.69223           | 0.68842           |
| With dropout, no regularization        | 1.379783        | 1.380661  | 0.69225           | 0.68841           |
| With dropout, with regularization      | **1.379751**    | 1.380651  | 0.69228           | 0.68837           |

\[ \hat{L}_1(c, w^*, \theta) = -\log q_\theta(w^*|c) + \sum_{w \in W} q_\theta(w|c) \log t(c, w, \theta) \]  
\[ \hat{L}_2(c, w^*, \theta) = -\log q_\theta(w^*|c) + t(c, w^*, \theta) \log t(c, w^* \theta) \]

As in the case of discriminator training, the overall loss function \(L(\theta)\) that we actually minimize while model fine-tuning is given by \(L(\theta) = \mathbb{E}_{c \sim p(\cdot)}[L(c, \theta)]\).

In section 3.2 we show that fine-tuning is effective only with estimator \(\hat{L}_2\). We also prove there that if for some real context \(c\) and true next word \(w^*\) probability \(q_\theta(w^*|c)\) is greater than estimation \(\hat{p}(w^*|c)\) of real probability, then gradient steps during fine-tuning are made towards decreasing \(q_\theta(w^*|c)\) (unlike fine-tuning with cross-entropy loss where gradient steps always increase \(q_\theta(w^*|c)\)). This is the main reason why language models benefit from our fine-tuning.

3 Experiments

We perform various experiments on different datasets. We illustrate all basic properties of our fine-tuning scheme on a classical language modeling benchmark Penn Treebank. The approach is also shown to be scalable to large datasets and to work well on another popular language modeling benchmark WikiText-2.

3.1 Penn Treebank

Figure 1: Discriminator training curves (PTB). Marks \((p)\) and \((q)\) on the right plot mean the same as corresponding superscripts in (5).

We started with current state-of-the-art language model based on the idea of mixture of softmaxes proposed by [Yang et al. (2017)](http://www.example.com). All parameters and hyperparameters were taken from this paper. Discriminator’s architecture is the same as that of the language model. Training scheme is also the same – we train discriminator with Stochastic Gradient Descent (SGD) and switch to Averaged Stochastic Gradient Descent (ASGD) when the quality ceases to improve.

Training curves on Figure 1 show that discriminator training is very stable and fast – it takes around 20 epochs to converge. On the figure there is a line corresponding to loss equal to \(\log 4\). This is the best loss discriminator can have in case the initial language model is optimal (i.e. \(q_0 \equiv p\)). Since language model is not, discriminator can obtain lower level of loss which we actually observe.
Table 2: Quality after fine-tuning (PTB). Marks "P-eval" and "Q-eval" stand for (13) and (12) estimators correspondingly. Mark "no discriminator" means that for these estimators we took \( r(w|c) \equiv 0.5 \).

| Fine-tuning Option | Val. loss | Val. ppl. | Val. rev. KL-div. | Test ppl. |
|--------------------|-----------|-----------|-------------------|-----------|
| Initial model\(^1\) | 4.01588   | 56.77     | 0.01476           | 54.66     |
| No dropout, P-eval, with discriminator | 4.04757   | 55.75     | 0.02604           | 53.91     |
| With Dropout, P-eval, with discriminator | 4.05360   | 56.23     | 0.02413           | 54.34     |
| No dropout, Q-eval, with discriminator | both total loss and perplexity increase on validation | both total loss and perplexity increase on validation |
| With Dropout, Q-eval, with discriminator | both total loss and perplexity increase on validation | both total loss and perplexity increase on validation |
| No dropout, P-eval, no discriminator | 4.06014   | 56.41     | 0.02756           | 54.45     |
| With Dropout, P-eval, no discriminator | both total loss and perplexity increase on validation | both total loss and perplexity increase on validation |
| No dropout, Q-eval, no discriminator | both total loss and perplexity increase on validation | both total loss and perplexity increase on validation |
| With Dropout, Q-eval, no discriminator | both total loss and perplexity increase on validation | both total loss and perplexity increase on validation |

Also, Table[1] shows that it is not so important whether we apply dropout and other regularization techniques used in the original paper (\( L_2 \)-regularization on activations) or not – one can see that the results differ insignificantly. Discriminator training dependence on learning rate is also mild. We varied learning rate from 0.1 to 5.0 (with gradient clipping at 0.25) and loss on validation set differed by values of order \( 10^{-5} \).

We found out that the best performance on validation set was when we applied both dropout and all other regularization techniques and set learning rate to 1.0, so for further experiments with fine-tuning we chose this discriminator.

The results of fine-tuning with the procedure proposed in section 2.2 are given in Table 2. We compared various fine-tuning options. It is possible to fine-tune the language model with or without dropout (in the first case the same dropout probabilities are chosen as the ones from training the initial language model). We also compared two different ways of estimating reverse KL-divergence expressed by formulas (12) and (13). To show that discriminator is an important part of our fine-tuning algorithm, we made experiment with "random" discriminator (i.e. when in formula (7) we have \( \frac{1}{2} \) for all words and contexts; or, equivalently, when we always approximate \( p \) with \( q_0 \) without any knowledge from discriminator). Unlike discriminator training, we found out that learning rate has some impact on the performance of fine-tuned model. All models from Table 2 were trained with learning rate 1.0 (with gradient clipping at 0.25) that was chosen because it minimized validation perplexity (exponent of cross-entropy) for the best of fine-tuning options mentioned in the table. It took about 40 epochs to fine-tune the language model in the setting that resulted in state-of-the-art quality.

We have to note that we couldn’t reproduce results from [Yang et al. (2017)]. The results of our training were slightly worse and perplexity differed by 0.2 – 0.4 on PTB and WT2 (see Table 5).

Table 2 demonstrates that all models do not get better in terms of total loss. Nevertheless, some of them still continue to improve perplexity.

3.2 Analysis

First, although estimating reverse KL-divergence using expression \( \hat{L}_1 \) from formula (12) and \( \hat{L}_2 \) from (13) give approximately the same results (difference is of order \( 10^{-5} \)) during test stage, estimation of this divergence during training with dropout using the first formula is by one order of magnitude greater than estimation obtained by applying the second one. The reason is that these estimators contain expressions \( \frac{q_0(w|c)}{q_0(w|c)} \), where denominator is calculated from fixed neural network without dropout while numerator is calculated using dropout scheme. Although dropout is designed in such a way that expectation of the output is the same as if no units were dropped out, this scheme still has some variance. So, even at the beginning of fine-tuning the ratio \( \frac{q_0(w|c)}{q_0(w|c)} \) doesn’t equal exactly to 1. Instead, it is a random variable with mean 1 and some variance. Also, our discriminator is only close to being optimal but it is not exactly optimal – there is some error in the ratio \( r(w|c) \frac{1-r(w|c)}{1-r(w|c)} \).

\(^1\)reverse KL-divergence is estimated using (13); estimation based on (12) differs by term of order \( 10^{-5} \).
as well. Thus, calculation $t(w, c, \theta)$ from (11) adds some error (some bias) into estimators $\hat{L}_1$ and $\hat{L}_2$ and training with dropout adds some variance to them. Moreover, the first estimator requires calculating $t(w, c, \theta)$ for every word from vocabulary $W$ which ruins the training. It can observed from Table 2. One can also note that even when we use $\hat{L}_2$ (which requires calculating $t(c, w, \theta)$ only once for true next word $w^*$) and apply dropout during fine-tuning, the model still cannot improve total loss and cross-entropy is worse than in the scheme where we fine-tune the model without dropout.

So, one of the conclusions that can be made is that dropout makes our fine-tuning scheme impractical because of the properties of the estimators for reverse KL-divergence. That’s why we have to choose $\hat{L}_2$ as the estimator and fine-tuning without dropout regardless the size of the model or size of the dataset we train this model on.

The second interesting observation is that even though we inevitably overfit in terms of total loss (because we do not apply dropout and dataset is small), we can still get better in terms of cross-entropy. This phenomenon is illustrated by Figure 2. Training curves there are drawn for the case that was chosen as the most effective one (i.e. no dropout and using $\hat{L}_2$ estimator, or "P-eval" as it is referred to in Table 2).

Note that estimated reverse KL-divergence during training drops below zero. It means that actual reverse KL-divergence gets very close to zero on training (and we overfit mostly due to this part of the total loss) but its estimation becomes eventually negative because $\hat{p}$ isn’t accurate for some words and contexts which are detected and utilized by our language model during fine-tuning.

The following proposition explains why our fine-tuning method can decrease perplexity:

**Proposition 1.** Let $\hat{p} = \hat{p}(w^*|c)$, $q_0 = q_0(w^*|c)$, $v = \nabla q_0$, $\varepsilon > 0$. Then

$$
\nabla\theta(- \log q_0) = -\frac{1}{q_0} \cdot v,
\nabla\theta\left(\frac{q_0}{\hat{p}} \log \frac{q_0}{\hat{p}}\right) = \left(\frac{1}{\hat{p}} \left(1 + \log \frac{q_0}{\hat{p}}\right)\right) \cdot v \quad (14)
$$

$$
q_0 = \hat{p} - \varepsilon \implies \nabla\theta\left(- \log q_0 + \frac{q_0}{\hat{p}} \log \frac{q_0}{\hat{p}}\right) = \left(-\frac{2\varepsilon}{\hat{p}^2} - \frac{3\varepsilon^2}{2\hat{p}^3} + o(\varepsilon^2)\right) \cdot v \quad (15)
$$

$$
q_0 = \hat{p} + \varepsilon \implies \nabla\theta\left(- \log q_0 + \frac{q_0}{\hat{p}} \log \frac{q_0}{\hat{p}}\right) = \left(\frac{2\varepsilon}{\hat{p}^2} - \frac{3\varepsilon^2}{2\hat{p}^3} + o(\varepsilon^2)\right) \cdot v \quad (16)
$$

**Proof.** Formulas (14) are obvious. Applying Taylor’s expansion proves formulas (15) and (16). For details see appendix A. ☐
Table 3: Perplexity on PTB and WT2.

| Model                                | Penn Treebank | WikiText-2 |
|--------------------------------------|---------------|------------|
|                                      |   Valid      |   Test     |   Valid    |   Test     |
| Mixture of softmaxes, original paper |  56.54       |  54.44     |  63.88     |  61.45     |
| Mixture of softmaxes, results of our training |  56.77       |  54.66     |  64.35     |  61.90     |
| Mixture of softmaxes, our fine-tuning |  55.75       |  53.91     |  63.52     |  61.19     |
| Recent results with TrellisNet       |        −      |  64.19     |        −    |        −    |
| Mixture of softmaxes, original paper + dyn. eval. |  48.33       |  47.09     |  42.41     |  40.68     |
| Mixture of softmaxes, our fine-tuning + dyn. eval. |  48.25       |  47.68     |  42.83     |  40.96     |

examples we have \( q_0 \approx \hat{p} \) and \( \log \frac{p}{\hat{p}} \approx 0 \). This observation along with identities (14) shows that \( \mathbf{v} \) is the direction of the gradient of reverse KL-divergence and \(-\mathbf{v}\) is the direction of the gradient of cross-entropy loss. If we consider fine-tuning with standard cross-entropy loss, we always take a gradient step in the direction \( \mathbf{v} \) to increase probability \( q_0 \) of the true word, even if it is already big. It seems unreasonable. In our fine-tuning scheme, when \( q_0 < \hat{p} \) (i.e. we know that most likely probability \( q_0 \) of true word is underestimated) we take a step in the direction \( \mathbf{v} \) towards increasing \( q_0 \), and when \( q_0 > \hat{p} \) (i.e. we know that most likely probability \( q_0 \) of true word is overestimated) we take a step in the opposite direction \(-\mathbf{v}\) towards decreasing \( q_0 \) even though it is probability of the true next word. If \( q_0 = \hat{p} \) (i.e. we know that most likely \( q_0 \) is already a good approximation of real probability) then we do not take gradient step at all.

Comparison of right-hand sides of formulas (15) and (16) with (14) shows that the steps in our fine-tuning scheme are much smaller (by a factor of \( \hat{\theta} \)) than if we were fine-tuning with just cross-entropy loss. This fact combined with the previous argument shows that training with standard cross-entropy loss can lead to small overestimation of some words, and the fine-tuning scheme that we propose is able to make little steps to correct it.

### 3.3 WikiText-2

The model utilizing mixture of softmaxes idea (Yang et al. 2017) was used for the experiments on WikiText-2. We chose the same parameters and hyperparameters that allowed to reach state-of-the-art results on this benchmark.

Discriminator was trained to reach loss \( \hat{L} = 1.3815 \) (\( \hat{L}^{(p)} = 0.6918 \) and \( \hat{L}^{(q)} = 0.6897 \)) on test set. Training curves for fine-tuning of state-of-the-art model on WikiText-2 look similar to those for PTB. They are given in appendix B.

Discriminator training and language model fine-tuning took 25 and 8 epochs correspondingly. Learning rates for both were set to 1.0 because they resulted in best performance on validation set. These were the only hyperparameters that we had to tune. Similarly to PTB, discriminator was trained with dropout (the same dropout rates as the ones used for training initial model). Fine-tuning was performed without dropout, since we’ve shown (see section 3.2) that this is a necessary condition for our algorithm to work.

Perplexity of the fine-tuned models (both for PTB and WikiText-2) are shown in Table 3. Dynamic evaluation parameters were tuned using grid search on validation set. For PTB we also mention the results obtained recently by Bai et al. (2018) with TrellisNet.

### 3.4 Large-scale experiments

Large-scale dataset we trained our model on consisted of 4Gb of English texts gathered from the Internet (blogs, news articles, forums, etc.) Vocabulary consisted of top 100k words from training corpus. Validation and test sets’ sizes were around 100Mb. The model that we chose was a single-layer LSTM with 500 hidden units. We used differentiated softmax (Chen et al. 2016) with output embeddings dimension varying from 150 for high-frequency words to 16 for low-frequency words.

Fine-tuning curves are shown on Figure 3. The initial model perplexity 128.55 went down to 126.95 (discriminator loss was 1.3658). On this large-scale dataset we do not observe overfitting in terms of total loss unlike on small datasets that were considered earlier.
4 Conclusion

We have presented a novel optimization trick that makes it possible to fine-tune language models at any scale. The key idea is to modify cross-entropy loss by adding reverse KL-divergence that is estimated using discriminator that has to be trained first. It enables model to decide whether probability of current word is underestimated or overestimated, and to choose the direction of gradient step depending on that. This approach allowed us to reach state-of-the-art quality in language modeling task on popular benchmarks Penn Treebank and WikiText-2. Apart from being effective, our approach is also fast and easy to use since it requires almost no hyperparameter tuning.

References

Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. CoRR, abs/1701.04862, 2017. URL http://arxiv.org/abs/1701.04862.

Samaneh Azadi, Catherine Olsson, Trevor Darrell, Ian Goodfellow, and Augustus Odena. Discriminator Rejection Sampling. ArXiv e-prints, October 2018.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis Networks for Sequence Modeling. ArXiv e-prints, October 2018.

Wenlin Chen, David Grangier, and Michael Auli. Strategies for training large vocabulary neural language models. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 1975–1985. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1186. URL http://www.aclweb.org/anthology/P16-1186.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1179. URL http://www.aclweb.org/anthology/D14-1179.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), Advances in Neural Information Processing Systems 27, pp. 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Comput., 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. Comput. Linguist., 19(2):313–330, June 1993. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=972470.972475.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. ArXiv e-prints, September 2016.
A Proposition proof

Proof of formula (15) (recall that \( q_\theta = \hat{p} - \varepsilon \)):

\[
\frac{\partial}{\partial q_\theta} \left( - \log q_\theta + \frac{q_\theta}{\hat{p}} \log \frac{q_\theta}{\hat{p}} \right) = \left( - \frac{1}{q_\theta} + \frac{1}{\hat{p}} \left( 1 + \log \frac{q_\theta}{\hat{p}} \right) \right) = - \left( \frac{1}{\hat{p}} - \varepsilon \right) \frac{1}{\hat{p}} \log \left( 1 - \varepsilon \right)
\]

\[
= - \frac{\varepsilon}{\hat{p}^2} \left( 1 - \frac{\varepsilon}{\hat{p}} \right)^{-1} + \frac{1}{\hat{p}} \left( \frac{\varepsilon}{\hat{p}} - \frac{\varepsilon^2}{2\hat{p}^2} + o(\varepsilon^2) \right) = - \frac{\varepsilon}{\hat{p}^2} \left( 1 + \frac{\varepsilon}{\hat{p}} + o(\varepsilon) \right) - \frac{\varepsilon}{\hat{p}^2} \frac{\varepsilon}{2\hat{p}^2} + o(\varepsilon^2)
\]

Proof of formula (16) (recall that \( q_\theta = \hat{p} + \varepsilon \)):

\[
\frac{\partial}{\partial q_\theta} \left( - \log q_\theta + \frac{q_\theta}{\hat{p}} \log \frac{q_\theta}{\hat{p}} \right) = \left( - \frac{1}{q_\theta} + \frac{1}{\hat{p}} \left( 1 + \log \frac{q_\theta}{\hat{p}} \right) \right) = \left( \frac{1}{\hat{p}} - \frac{1}{\hat{p} + \varepsilon} \right) + \frac{1}{\hat{p}} \log \left( 1 + \frac{\varepsilon}{\hat{p}} \right)
\]

\[
= \frac{\varepsilon}{\hat{p}^2} \left( 1 + \frac{\varepsilon}{\hat{p}} \right)^{-1} + \frac{1}{\hat{p}} \left( \frac{\varepsilon}{\hat{p}} - \frac{\varepsilon^2}{2\hat{p}^2} + o(\varepsilon^2) \right) = \frac{\varepsilon}{\hat{p}^2} \left( 1 - \frac{\varepsilon}{\hat{p}} + o(\varepsilon) \right) + \frac{\varepsilon}{\hat{p}^2} - \frac{\varepsilon^2}{2\hat{p}^2} + o(\varepsilon^2)
\]

B Training curves

Figure 4: Fine-tuning curves (WT2).