Penalizing Confident Neural Networks

Ondrej Bohdal

4th Year Project Report
Artificial Intelligence and Mathematics
School of Informatics
University of Edinburgh

2018

Abstract

Neural networks often fit too closely to the training data, which results in training models that do not generalize well. There are various symptoms of over-fitting, but one of them is that the network assigns too much probability in the output distribution to a single class. We call such distributions over-confident. In this project, we explore an approach to supervised neural network training in which the loss function is augmented by a term that penalizes over-confident output distributions. Such regularization is called confidence penalty, and it is related to label smoothing regularization. We evaluate both confidence penalty and label smoothing, and compare them with models using either dropout or no regularization.

We study the underlying mathematical theory and conduct experiments to evaluate the usefulness of the approaches in practice. We implement the approaches using TensorFlow framework and evaluate them on a standard image recognition MNIST dataset. We show both confidence penalty and label smoothing lead to significantly lower test error rates than dropout or no regularization. We observe the improvements using both stochastic gradient descent and Adam optimization, although they are smaller when using Adam optimization. We also closely explore extensions of confidence penalty, namely thresholding and annealing. We show it is sufficient to use a basic version of the confidence penalty, while achieving comparable performance and using fewer hyperparameters.

Acknowledgements

I would like to thank my supervisor Steve Renals for proposing an interesting research project and providing me with useful feedback and guidance over the course of my work on the project. I would also like to thank my supervisor's PhD students, namely Joanna Rownicka and Joachim Fainberg for helping me solve complications when setting up some of the frameworks and libraries relevant to my work. Finally, I would like to thank my parents for helpful discussions and providing me with additional feedback on how to improve my report.

Table of Contents

Chapter 1

Introduction

Neural networks are some of the most successful machine learning algorithms, especially because of their strong performance on a variety of challenging tasks such as image [Szegedy et al., 2015] or speech [Graves et al., 2013] recognition. They are loosely inspired by the structure of connections between neurons in the brain, and they have proven to work well in practice. However, neural networks are often likely to over-fit to the training data, which typically results in worse generalization and less accurate predictions on unseen data. As a result, we need to utilize regularization techniques that help prevent over-fitting. Some of the most common regularization techniques include, for example, dropout [Srivastava et al., 2014], L1 and L2 regularization [Ng, 2004] and batch normalization [Ioffe and Szegedy, 2015].

In our work, we investigate in detail a recently proposed regularization technique called confidence penalty. Confidence penalty was proposed by [Pereyra et al., 2017], and the idea of confidence penalty is to penalize models giving peaky or over-confident output distributions. These models typically assign most probability in the output distribution to a single class, which has been shown to be a sign of over-fitting [Szegedy et al., 2016]. Consequently, we want to avoid training models that produce over-confident output distributions, which are best described by low entropy values. Confidence penalty is a regularization that penalizes models producing low entropy outputs. As part of our work, we aim to confirm confidence penalty regularization leads to higher entropy output distributions and higher classification accuracies on unseen data. Confidence penalty has various possible extensions, and we explore these as well to see if they reinforce the impact of confidence penalty.

Confidence penalty regularization is related to a technique called label smoothing [Szegedy et al., 2016]. As part of our research, we investigate both of them, compare with dropout and no regularization, and we also derive the precise relations between confidence penalty and label smoothing, inspiring new related forms of regularization.

1.1 Goals

The main goal of our work is to investigate confidence penalty in the context of supervised training of neural networks using a standard dataset. Our investigation is both theoretical and experimental, which means we do not focus only on implementing the approach and doing relevant experiments, but we also investigate the mathematical theory behind confidence penalty regularization. In particular, one of our further goals is to explore the connection between confidence penalty and label smoothing and show how precisely they are related to each other.

As part of investigating confidence penalty, we experimentally compare confidence penalty and label smoothing to a popular regularization called dropout and also to no regularization. Moreover, we explore the effect of using different optimization methods, stochastic gradient descent and Adam [Kingma and Ba, 2014], on both confidence penalty and label smoothing. As our further goal we explore two extensions of confidence penalty:

- confidence penalty thresholding, which is applying confidence penalty only when the entropy of the output distribution is below a certain threshold,
- confidence penalty annealing, which means making the confidence penalty either stronger or weaker as the training progresses. We investigate both alternatives.

1.2 Contributions

Our main contributions include:

- implementation and experimental investigation of confidence penalty and its extensions using a standard dataset,
- evaluation of confidence penalty and its extensions when using stochastic gradient descent and Adam optimization methods,
- experimental comparison of the performance of confidence penalty and its extensions with label smoothing, dropout and no regularization,
- theoretical investigation of confidence penalty, and in particular its relation to label smoothing, inspiring new related regularizations.

1.3 Report Outline

- **Chapter 2** describes the background useful for understanding the details of our work. We describe related work, explain why over-confidence is related to overfitting, and then we define and explain confidence penalty, its extensions and label smoothing. We also describe the optimizers, the data and the deep learning framework that we use.

- **Chapter 3** begins by defining the Kullback-Leibler divergence, which we then use to connect the notion of confidence penalty and label smoothing. We explain how precisely the two regularizations are connected, and based on our mathematical study of both, we propose several related regularizations.
- **Chapter 4** gives an overview of the experiments we have conducted, provides information about the network architecture, and it also gives insight into the technical details of our experiments.
- **Chapter 5** gives results of our experiments and provides their in-depth analysis. The chapter concludes by a discussion of our results compared to other results obtained in literature.
- **Chapter 6** introduces the further work we have done on a different, more complicated, the TIMIT speech recognition dataset.
- **Chapter 7** provides a summary of our work, our conclusions and also describes further possible directions that could be explored.

Chapter 2

Background

Over-fitting of neural networks has been a known problem for already a long time, and consequently many strategies have been devised to prevent this undesired behaviour. Over-fitting is a problem because it means neural networks fit too closely to the training data, and as a result they do not generalize well and do not give accurate predictions on unseen data. Some of the most common regularization techniques include L1 and L2 regularization (weight decay), dropout, data augmentation, which are explained in detail in Chapter 3 of [Nielsen, 2015] and Chapter 7 of [Goodfellow et al., 2016]. [Goodfellow et al., 2016] also describe other regularization techniques, including injecting noise at the output targets, adversarial examples or multi-task learning.

Various regularizations focus on various aspects of training neural networks, for example, some of them modify the data and some other ones encourage smaller weights. The regularization that we investigate in our work, confidence penalty, belongs to the class of output regularizations as it regularizes the network based on the outputs that it produces. [Hinton et al., 2015] look at the knowledge of a model in terms of the output distribution given an input, rather than the trained values of the parameters, which motivates the use of output regularizations. Moreover, the fact confidence penalty regularizes the model based on the output and not on the parameters of the network, makes the regularization more interpretable. Confidence penalty, as explained in [Pereyra et al., 2017], draws from the maximum entropy principle [Jaynes, 1957], which in short says that the most unbiased representation of knowledge of a system is given by the probability distribution with the maximum entropy.

Confidence penalty regularization has been proposed in [Pereyra et al., 2017] to prevent over-fitting and hence achieve higher classification accuracies on unseen data. The main strength of the paper by [Pereyra et al., 2017] is it evaluates the impact of confidence penalty on a variety of tasks such as image classification, speech recognition, language modelling and machine translation. The paper uses standard benchmarks and shows both confidence penalty and label smoothing help in improving state-of-the-art models without changing the hyperparameters of existing models. This makes the regularization widely applicable as it can be easily added to existing models. The paper focuses on evaluating the impact of confidence penalty on a variety of tasks, but it does not go into great depth in examining the impact of the confidence penalty. There

are some signs of further analysis and evaluation using the MNIST data set [LeCun and Cortes, 1998], but we can go significantly deeper and explain the effect of using confidence penalty in more detail. Moreover, [Pereyra et al., 2017] mention possible extensions of confidence penalty, but they do not evaluate them in the paper. However, we explore them in our work in great detail, which adds to the originality and novelty of our work. [Pereyra et al., 2017] is the paper forming the basis of our work. [Dubey et al., 2017] have performed further research into confidence penalty, showing confidence penalty is especially useful when only very limited amount of data is available.

As mentioned in [Pereyra et al., 2017], confidence penalty is related to label smoothing. Label smoothing is introduced in Chapter 7 of [Szegedy et al., 2016] as a way to improve generalization abilities of neural networks. A strong point of the paper is the authors give clear and detailed theoretical explanations of principles of label smoothing. The authors also provide a result for ImageNet [Russakovsky et al., 2015] with 1000 classes and report a consistent improvement. The weak point of the paper is that label smoothing is not thoroughly evaluated on various tasks, although that is clearly not the aim of the paper. However, as mentioned earlier, both confidence penalty and label smoothing have been investigated for various tasks in [Pereyra et al., 2017], and both of them have been shown to result in improvements in the classification accuracy for unseen data.

2.1 Loss Functions

Before formally defining confidence penalty, we review loss functions. Confidence penalty and, in fact, many other regularizations, including L1 and L2 regularization, form a term that is added to the loss function.

Loss function measures how far the output distribution given by the model is from the target distribution, and we typically minimise this value. In our research, we use cross entropy loss function, which is defined by

$$L(y,t) = -\sum_{k=1}^{K} t_k \log(y_k),$$

where y is the output distribution, t is the target label distribution for the current example, and K is the number of classes. In terms of our notation, we denote the natural logarithm of x as $\log(x)$. The base of the logarithm does not affect the loss function because using a different base value only rescales the loss function. Model outputs are calculated using softmax function

$$y_k = \frac{\exp(z_k)}{\sum_{i=1}^{K} \exp(z_i)},$$

where k is the class label and z_i is the logit or unnormalized log-probability for class i given as an output from the model.

The gradient of cross entropy with respect to the logits is [Szegedy et al., 2016]

$$\frac{\partial L(y,t)}{\partial z_k} = y_k - t_k. \tag{2.1}$$

The gradient is restricted to the interval between -1 and 1 because $0 \leq y_k, t_k \leq 1$.

Cross entropy is a common choice of loss function in deep learning because it works well in practice. Chapter 3 of [Nielsen, 2015] explains that cross entropy as a loss function does not suffer from learning slowdown that happens when using squared loss function. Moreover, cross entropy is used in both [Pereyra et al., 2017] and [Szegedy et al., 2016], which are the papers giving foundations to confidence penalty and label smoothing. [DiPietro, 2016] has shown that cross entropy for a classification task is equal to negative log-likelihood.

2.2 Over-confidence and Over-Fitting

In this section we summarize the main points from [Szegedy et al., 2016] that explain why over-confidence of neural networks is a symptom of over-fitting.

Given a target label distribution t, typically $t_k = 1$ for the true label k, and $t_i = 0$ for all labels $i \neq k$. As a result, negative log-likelihood is minimized when $\log(y_k)$ for true label k is maximized, which happens for $y_k = 1$. This means the model assigns the full probability to the correct class and zero probability to the incorrect classes. In practice, we cannot obtain $y_k = 1$ for finite logits due to the definition of softmax. However, the effect is similar when the logit for the true class is much larger than the logits for all other classes - formally when $z_k \gg z_i$ for all classes $i \neq k$.

[Szegedy et al., 2016] warn hard 0 or 1 classification targets may result in over-fitting because the model is not guaranteed to generalize if it learns to assign full probability to the true label for each training example. Moreover, [Szegedy et al., 2016] argue that it encourages large differences between the largest logit and all other logits, which combined with the gradient bounded between -1 and 1 reduces the ability of the model to adapt. Finally, [Szegedy et al., 2016] intuitively explain it happens because the model becomes too confident about its predictions.

2.3 Confidence Penalty

Confidence penalty [Pereyra et al., 2017] as a form of regularization aims to prevent neural networks from producing over-confident output distributions. Over-confident output distributions place most or all probability to a single class, which is a symptom of over-fitting as explained in [Szegedy et al., 2016]. As a result, confidence penalty should lead to better generalization of the neural network, which we would like to verify.

2.3.1 Definition

Let us formally define confidence penalty, following the definitions given in [Pereyra et al., 2017]. A neural network uses softmax function to produce a conditional distribution $p_\theta(y|x)$ over class distribution y with K classes given an example x under a fixed state θ of the system.

The probability of a class $k \in \{1, ..., K\}$ given by our model is calculated using softmax function

$$p_\theta(y_k|x) = \frac{\exp(z_k)}{\sum_{i=1}^{K} \exp(z_i)},$$

where z_i is the i-th logit (unnormalized log-probability for class i).

Next, we define entropy and loss function regularized by confidence penalty. The following formulas are valid only for a single example, but we can extend them to all examples by calculating the mean. Entropy is defined by

$$H(p_\theta(y|x)) = -\sum_{k=1}^{K} p_\theta(y_k|x) \log(p_\theta(y_k|x)).$$

Negative log-likelihood with confidence penalty regularization has the following form

$$L(\theta) = -\sum_{k=1}^{K} t_k \log(p_\theta(y_k|x)) - \beta H(p_\theta(y|x)), \tag{2.2}$$

where t is the ground-truth label distribution and β is the confidence penalty coefficient that controls how strong the confidence penalty is.

It is worth to point out the definitions in [Pereyra et al., 2017] use relatively unclear indexing in sums because, for example, some sums do not have any indices. Although lack of clarity of definitions is a weakness of [Pereyra et al., 2017], we try to use clear definitions in our work.

2.3.2 Gradient

In this section, we give an expression for the gradient of the entropy and loss function using confidence penalty with respect to the logits. Gradient of the entropy is useful when evaluating the gradient of our loss function regularized using confidence penalty. Gradient of the entropy with respect to logit z_i is

$$\frac{\partial H(p_\theta)}{\partial z_i} = p_\theta(y_i|x)(-\log p_\theta(y_i|x) - H(p_\theta)). \tag{2.3}$$

Instead of writing $H(p_\theta(y|x))$ in full, we use an abbreviation $H(p_\theta)$. The expression is stated in [Pereyra et al., 2017], but we derive it on our own. We include our own

derivation of the entropy gradient in Appendix A. The fact we can calculate entropy gradient using a relatively simple expression means we can use confidence penalty efficiently as part of back-propagation [Rumelhart et al., 1988] that is used for training neural networks.

Using Formulas 2.1 and 2.3, the gradient of the loss function using confidence penalty with respect to the i-th logit is

$$\frac{\partial L(\theta)}{\partial z_i} = p_\theta(y_i|x) - t_i + \beta p_\theta(y_i|x)\left(\log p_\theta(y_i|x) + H(p_\theta)\right).$$

2.3.3 Thresholded Confidence Penalty

It has been proposed in [Pereyra et al., 2017] that it could be potentially useful to strengthen the confidence penalty as training progresses. To achieve this, according to [Pereyra et al., 2017], we could only penalize output distributions when they are below a certain entropy threshold. [Pereyra et al., 2017] do not provide detailed investigation into thresholding confidence penalty, they only mention the objective form and that they did some preliminary experiments.

As explained in [Pereyra et al., 2017], the motivation for using confidence penalty only when entropy is below a certain threshold is to have quick convergence, while preventing over-fitting near the end of training. This suggests that confidence penalty should be weak at the beginning of training and strong near convergence.

[Pereyra et al., 2017] propose entropy thresholding can be achieved by adding hinge loss to the loss function with the overall loss function

$$L(\theta) = -\sum_{k=1}^{K} t_k \log\left(p_\theta(y_k|x)\right) - \beta \max\left(0, \Gamma - H\left(p_\theta(y|x)\right)\right),$$

where Γ is the threshold value under which confidence penalty is applied.

However, we believe this is not the appropriate objective if we want to use confidence penalty in the same way as before but apply it only below a certain threshold value. When the entropy is below Γ, the formula proposed by [Pereyra et al., 2017] adds the confidence penalty to the negative log-likelihood instead of subtracting it as we did before. This is a difference because when minimizing the loss function, we encourage lower values of entropy than before, which completely counteracts the idea of using confidence penalty.

We propose the loss function for thresholded confidence penalty is of the form

$$L(\theta) = -\sum_{k=1}^{K} t_k \log\left(p_\theta(y_k|x)\right) - \beta \min\left(0, H\left(p_\theta(y|x)\right) - \Gamma\right).$$

In our proposed loss function, confidence penalty is applied only when it is below a certain threshold value, and when confidence penalty is used, we subtract it instead

of adding it. Adding a constant to the loss function does not change the behaviour of minimizing the objective.

A disadvantage of using entropy thresholds lies in adding another hyperparameter, and without further exploration, it may be difficult to choose a good value for it. We will find if it is worth to introduce this hyperparameter as part of evaluation of our experiments.

2.3.4 Annealed Confidence Penalty

Another possible approach to strengthen confidence penalty as training progresses is to use annealing. Annealing of confidence penalty is mentioned in [Pereyra et al., 2017], but the authors give no indication if they actually explored it. Similarly as before, the motivation for annealing that strengthens the confidence penalty over training is to help quick convergence and prevent over-fitting near the end of the training.

Contrary to [Pereyra et al., 2017], we investigate confidence penalty annealing in our work. We propose annealing with parameters based on our previous experiments to discover how well annealing could work. We introduce a parameter that describes the strength of confidence penalty during training. We give details of the implementation in Chapter 4.

We also try an annealing that decreases the confidence penalty over training. The motivation is to explore an alternative point of view that the network should be penalized more at the beginning, when the network may be unjustifiably quickly confident about its predictions. When the network is trained for sufficiently long time, its confident predictions may be more justified, and as a result we should not penalize it for confidence as much as we did initially. Although this alternative point of view is not mentioned in [Pereyra et al., 2017], we have decided to explore it. We believe both types of annealing have reasonable justification, and we perform experiments for both.

2.4 Label Smoothing

2.4.1 Definition

Label smoothing regularization is introduced in Section 7 of [Szegedy et al., 2016] as a way of improving generalization of neural networks. In this subsection we define label smoothing following [Szegedy et al., 2016], illustrating it with an own example.

Let $u(i)$ be a fixed distribution over labels $i \in \{1, ..., K\}$, independent of the training example x and let ε be a smoothing parameter. For an example with ground-truth label k, we replace the target label distribution $t(i|x) = \delta_{i,k}$ with

$$t'(i|x) = (1-\varepsilon)\,\delta_{i,k} + \varepsilon u(i),$$

where $\delta_{i,k}$ is the Dirac delta that is equal to 1 when $i = k$ and 0 when $i \neq k$. $t'(i|x)$ is a combination of the original ground-truth distribution $t(i|x)$ and the fixed distribution $u(i)$ with weights $1-\varepsilon$ and ε, respectively. We will use $u(i)$ and u_i notation interchangeably.

[Szegedy et al., 2016] propose $u(i)$ is the prior distribution over labels. In case of perfectly balanced datasets, $u(i)$ is the uniform distribution, hence $u(i) = \frac{1}{K}$ if there are K different classes.

The loss function when using label smoothing becomes

$$\begin{aligned} L(\theta) &= -\sum_{i=1}^{K} t'_i \log\left(p_\theta(y_i|x)\right) \\ &= -\sum_{i=1}^{K} \log\left(p_\theta(y_i|x)\right)\left((1-\varepsilon)\,\delta_{i,k} + \varepsilon u(i)\right) \\ &= -(1-\varepsilon)\sum_{i=1}^{K} t_i \log\left(p_\theta(y_i|x)\right) - \varepsilon \sum_{i=1}^{K} \log\left(p_\theta(y_i|x)\right) u(i) \\ &= (1-\varepsilon)\,H\left(t, p_\theta(y|x)\right) + \varepsilon H\left(u, p_\theta(y|x)\right). \end{aligned} \tag{2.4}$$

In the final step in Formula 2.4, we have used the definition of cross entropy

$$H(p,q) = -\sum_{i=1}^{K} p_i \log(q_i),$$

where p and q are probability distributions. The value of cross entropy is minimized when p and q are identical, assuming p is fixed [Goodfellow et al., 2016].

We illustrate the definitions from [Szegedy et al., 2016] using an own example to aid understanding. Assuming we have 5 classes and perfectly balanced data, $u(i) = \frac{1}{5}$ over classes i. The target label distribution for $\varepsilon = 0.05$ after label smoothing becomes

$$t'(i|x) = (1-0.05)\,\delta_{i,k} + \frac{0.05}{5}.$$

As a result, a target label distribution $t = [0,0,1,0,0]$ becomes a distribution

$$t' = [0.01, 0.01, 0.96, 0.01, 0.01].$$

Contrary to [Pereyra et al., 2017], [Szegedy et al., 2016] give clear, precise and easily understandable definitions, which have significantly helped us understand and fix the definitions given in [Pereyra et al., 2017]. Moreover, [Szegedy et al., 2016] give sound reasons why their proposed technique should achieve the goal, which we believe is another strength of the paper.

2.4.2 Gradient

Similarly as for confidence penalty, let us find an expression for the gradient of the loss function that uses label smoothing regularization. Using Formulas 2.1 and 2.4, we obtain

$$\begin{aligned}\frac{\partial L(\theta)}{\partial z_i} &= \frac{\partial\left((1-\varepsilon)H\left(t, p_\theta\left(y|x\right)\right)+\varepsilon H\left(u, p_\theta\left(y|x\right)\right)\right)}{\partial z_i} \\ &= (1-\varepsilon)\left(p_\theta\left(y_i|x\right)-t_i\right)+\varepsilon\left(p_\theta\left(y_i|x\right)-u\left(i\right)\right),\end{aligned}$$

where $u(i)$ is the fixed distribution over labels i, and for perfectly balanced datasets with K classes, $u(i) = \frac{1}{K}$ for all labels i. The remainder of our notation is the same as before. We used the expressions for gradients in our experiments, and we checked them by comparing to the gradients calculated automatically by TensorFlow "gradients" function [TensorFlow, 2018b].

2.5 Optimizers

We evaluate the effect of confidence penalty and label smoothing using two different optimization methods, stochastic gradient descent and Adam [Kingma and Ba, 2014]. The aim is to investigate if the optimization method we use has an impact on the improvements in generalization that we should get by using confidence penalty and label smoothing.

2.5.1 Stochastic Gradient Descent Optimizer

Stochastic gradient descent (SGD) as an extension of gradient descent does not use the whole training set at once to calculate gradients used for updating weights. Instead, SGD takes a randomly selected mini-batch of training examples, and uses them to approximately calculate the weight update. As a result, SGD can perform many more updates of weights of the network than gradient descent at a given time, which helps train the network faster.

2.5.2 Adam Optimizer

Adam optimizer [Kingma and Ba, 2014] has become a standard choice when it comes to choosing optimizers because its default hyperparameters typically require no or only little of tuning to achieve strong performance. Adam optimizer belongs to the category of adaptive optimizers as it computes adaptive learning rates for various parameters based on estimates of the first and second gradient moments. Adam is relatively similar to another optimization method RMSprop [Tieleman and Hinton, 2012], but Adam uses also momentum.

2.6 Data

We perform experiments using the MNIST dataset [LeCun and Cortes, 1998]. The MNIST dataset contains labelled images of hand-written digits, and it has 60,000 training and 10,000 test examples. From 60,000 training examples, we use the first 10,000 examples as a held-out validation set, and we use the remaining 50,000 examples for training. The task is to predict the correct digit given its image. There are 10 different classes of digits, and the images are grayscale with resolution 28x28. The dataset is not perfectly balanced, but differences in the numbers of examples from various classes are relatively small. The MNIST dataset is often used as a benchmark for machine learning, and it is commonly used to evaluate new machine learning approaches.

We use a permutation-invariant version of the MNIST task, which means we do not use models that assume any spatial relations. For example, convolutional neural networks [LeCun et al., 1998] assume spatial relations, but deep fully-connected neural networks do not.

We have decided to use the MNIST dataset because it is a commonly used benchmark, and it is one of the datasets used in [Pereyra et al., 2017], which allows us to compare our results and conclusions. Moreover, the fact MNIST has only 10 classes helps us analyse the impact of using various regularizations in more detail.

To give an illustration of various approaches to the MNIST task and what test error rates can be obtained, we provide a short summary in Table 2.1.

Architecture	Test Error Rate (%)	Reference
Conv DropConnect	0.21	[Wan et al., 2013]
Conv maxout + dropout	0.45	[Goodfellow et al., 2013]
2-layer CNN + 2-layer NN	0.53	[Jarrett et al., 2009]
Conv Fastfood, 2048 units	0.71	[Yang et al., 2015]
Conv net LeNet-4	1.10	[LeCun et al., 1998]
3-layer NN, 1024 units, dropout	1.25	[Srivastava et al., 2014]
2-layer NN, 1000 units	4.50	[LeCun et al., 1998]

Table 2.1: Test error rates for the MNIST task obtained using various architectures.

It is possible to obtain very low test error rates for the MNIST task. However, the aim of our work is not to obtain the lowest test error rate on MNIST. Our aim is to explore the usefulness of confidence penalty and label smoothing on a relatively standard architecture.

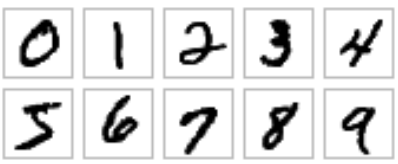

Figure 2.1: Examples of MNIST digits [TensorFlow, 2018a].

2.7 TensorFlow

In order to implement and evaluate our approaches, we use TensorFlow framework [Abadi et al., 2015]. TensorFlow is an open-source machine learning framework that uses data flow graphs for numerical computation. It is developed by Google Brain team, and it has achieved great popularity so far. We have used TensorFlow in connection with Python, which is a programming language we know well and which is commonly used in machine learning and data science.

We have decided to use TensorFlow, and not an alternative deep learning framework such as Keras [Chollet, 2015], MXNet [Chen et al., 2015] or Theano [Theano Development Team, 2016] because of multiple reasons. TensorFlow has a great documentation with many examples, it is very powerful, and it also enables us to work at a reasonably low level of abstraction. Moreover, as found by Andrej Karpathy in [Karpathy, 2017], TensorFlow was the most commonly mentioned framework in papers uploaded to arXiv in March 2017, which means TensorFlow is very popular among researchers.

TensorFlow is a framework with a lot of functionality, and consequently learning it is not simple. Moreover, TensorFlow algorithms are complex, which means it takes effort to use some of the TensorFlow features correctly. Even though TensorFlow has great documentation with many examples, it is often useful to examine the source code in detail as it is needed to understand how specifically some method works and what outputs we can expect.

Confidence penalty is not implemented in TensorFlow, hence we implemented it on our own. However, label smoothing is already a part of softmax cross entropy loss function in TensorFlow [TensorFlow, 2018c], so in order to evaluate label smoothing, we only need to specify its value as a parameter. On the other hand, implementing it on our own would be simple as it only includes modifying a one-hot encoded target vector using the formulas we have described for label smoothing.

Chapter 3

Confidence Penalty and Label Smoothing

As we will show in this chapter, confidence penalty and label smoothing are related to each other by "reversing the direction" of the Kullback-Leibler divergence. Based on our detailed analysis, we will propose new related forms of regularization. The fact the two regularizations are related to each other using the Kullback-Leibler divergence is stated in [Pereyra et al., 2017], however, it is not explained further in detail in the paper. The mathematical reasoning for confidence penalty and label smoothing that we present in this section is our own.

3.1 KL Divergence

The Kullback-Leibler (KL) divergence measures the discrepancy between two distributions p and q. The definition of the KL divergence for discrete probability distributions is

$$D_{\mathrm{KL}}(p\,||\,q) = \sum_{i=1}^{K} p(i) \log \frac{p(i)}{q(i)}.$$

The KL divergence is non-negative, and it is zero only when the two distributions are identical. The KL divergence is not a distance because $D_{\mathrm{KL}}(p\,||\,q) \neq D_{\mathrm{KL}}(q\,||\,p)$ [Goodfellow et al., 2016].

Moreover, [Goodfellow et al., 2016] state that

$$H(p,q) = H(p) + D_{\mathrm{KL}}(p\,||\,q), \tag{3.1}$$

where $H(p,q) = -\sum_{i=1}^{K} p_i \log(q_i)$ is the cross entropy between p and q and $H(p) = -\sum_{i=1}^{K} p_i \log(p_i)$ is the entropy of p.

3.2 Confidence Penalty via KL Divergence

Let us use Formula 3.1 to rewrite Formula 2.2 in terms of the KL divergence. We assume u is the uniform distribution over K classes, hence $u_i = \frac{1}{K}$ for every class $i \in \{1, ..., K\}$. We obtain

$$\begin{aligned}
L(\theta) &= -\sum_{i=1}^{K} t_i \log\left(p_\theta(y_i|x)\right) - \beta H\left(p_\theta(y|x)\right) \\
&= H\left(t, p_\theta(y|x)\right) - \beta H\left(p_\theta(y|x)\right) \\
&= H\left(t, p_\theta(y|x)\right) - \beta\left(H\left(p_\theta(y|x), u\right) - D_{\mathrm{KL}}\left(p_\theta(y|x) \,||\, u\right)\right) \\
&= H\left(t, p_\theta(y|x)\right) - \beta\left(-\sum_{i=1}^{K} p_\theta(y_i|x)\log(u_i) - D_{\mathrm{KL}}\left(p_\theta(y|x) \,||\, u\right)\right) \\
&= H\left(t, p_\theta(y|x)\right) - \beta\left(-\sum_{i=1}^{K} p_\theta(y_i|x)\log\left(\frac{1}{K}\right) - D_{\mathrm{KL}}\left(p_\theta(y|x) \,||\, u\right)\right) \\
&= H\left(t, p_\theta(y|x)\right) + \beta\left(-\log(K) + D_{\mathrm{KL}}\left(p_\theta(y|x) \,||\, u\right)\right),
\end{aligned}$$

in which we can ignore constants since they do not affect minimizing the loss function, and hence we can write

$$L(\theta) = H\left(t, p_\theta(y|x)\right) + \beta D_{\mathrm{KL}}\left(p_\theta(y|x) \,||\, u\right). \tag{3.2}$$

We can express $L(\theta)$ also as

$$L(\theta) = H\left(t, p_\theta(y|x)\right) - \beta\left(-\sum_{i=1}^{K} p_\theta(y_i|x)\log\left(p_\theta(y_i|x)\right)\right). \tag{3.3}$$

The entropy term $H\left(p_\theta(y|x)\right)$ becomes maximized when $p_\theta(y|x)$ is the uniform distribution.

3.3 Label Smoothing via KL Divergence

Let us write the loss function that includes label smoothing as described in Formula 2.4 in terms of the KL divergence. We use the same notation as we used for confidence penalty, and u is the uniform distribution. We obtain

$$\begin{aligned}
L(\theta) &= (1-\varepsilon) H\left(t, p_\theta(y|x)\right) + \varepsilon H\left(u, p_\theta(y|x)\right) \\
&= (1-\varepsilon) H\left(t, p_\theta(y|x)\right) + \varepsilon\left(H(u) + D_{\mathrm{KL}}\left(u \,||\, p_\theta(y|x)\right)\right) \\
&= (1-\varepsilon) H\left(t, p_\theta(y|x)\right) + \varepsilon\left(\log(K) + D_{\mathrm{KL}}\left(u \,||\, p_\theta(y|x)\right)\right),
\end{aligned}$$

which we can divide by $(1-\varepsilon)$ as it does not change the way the loss function is minimized, and after ignoring constants, we obtain

$$L(\theta) = H(t, p_\theta(y|x)) + \frac{\varepsilon}{1-\varepsilon} D_{\mathrm{KL}}(u \,||\, p_\theta(y|x)). \tag{3.4}$$

Using Formula 2.4 we can express $L(\theta)$ also as

$$L(\theta) = H(t, p_\theta(y|x)) + \frac{\varepsilon}{1-\varepsilon} \left(-\sum_{i=1}^{K} u_i \log(p_\theta(y_i|x)) \right). \tag{3.5}$$

The cross entropy term $H(u, p_\theta(y|x)) = -\sum_{i=1}^{K} u_i \log(p_\theta(y_i|x))$ becomes minimized when $p_\theta(y|x)$ and u are identical.

3.4 Connection between Confidence Penalty and Label Smoothing

We have expressed the loss function for both confidence penalty and label smoothing using the KL divergence between the output distribution $p_\theta(y|x)$ and the uniform distribution u and vice-versa. Comparing Formulas 3.2 and 3.4, we see confidence penalty is related to label smoothing by "reversing the direction" of the KL divergence.

If we did not use the uniform distribution u, we would not be able to connect the loss functions for confidence penalty and label smoothing after rescaling and ignoring constants. The reason is we have utilized $H(u) = H(p_\theta(y|x), u)$, which does not hold true in general. However, it holds for uniform distribution $u_i = \frac{1}{K}$ for all classes i as both $H(u)$ and $H(p_\theta(y|x), u)$ are equal to $\log(K)$, where K is the number of classes.

3.5 Related Regularizations

As mentioned in [Szegedy et al., 2016], u is a fixed distribution over labels, which does not need to be uniform. In fact, [Szegedy et al., 2016] propose u is the prior distribution over labels, however, they conduct experiments only using uniform distribution. We have used uniform distribution in our case as well. MNIST digits are not perfectly balanced, but we can use the uniform distribution as an approximation. When the prior distribution over labels is u in general, the loss function with label smoothing remains

$$\begin{aligned} L(\theta) &= (1-\varepsilon) H(t, p_\theta(y|x)) + \varepsilon H(u, p_\theta(y|x)) \\ &= H(t, p_\theta(y|x)) + \frac{\varepsilon}{1-\varepsilon} \left(-\sum_{i=1}^{K} u_i \log(p_\theta(y_i|x)) \right). \end{aligned}$$

Comparing Formulas 3.3 and 3.5, we can write the regularized loss function as

$$L(\theta) = H(t, p_\theta(y|x)) + \gamma \left(-\sum_{i=1}^{K} f_i(p_\theta(y|x)) \log(p_\theta(y_i|x)) \right),$$

for some real-valued coefficient γ and some function $f_i(p_\theta(y|x))$ that takes as input the output distribution and gives a value for the given class i. Whether γ is positive or negative depends on the particular function $f_i(p_\theta(y|x))$. For example, the coefficient is negative for confidence penalty and positive for label smoothing, assuming label smoothing value ε varies between 0 and 1.

One possible function $f_i(p_\theta(y|x))$ could apply label smoothing only over the top three classes as predicted by the model. We could smooth them using the uniform distribution $u_i = \frac{1}{K}$ for all classes i. We can define such function as

$$f_i(p_\theta(y|x)) = \begin{cases} 0 & \text{if} \quad i \in \{c_1, c_2, c_3\} \\ \frac{1}{K} & \text{if} \quad i \notin \{c_1, c_2, c_3\}, \end{cases}$$

where labels c_1, c_2, c_3 denote the indices of the three classes with the largest probabilities in the output distribution $p_\theta(y|x)$. In fact, we can take a different number of classes than three. The selected number of classes would most likely depend on the total number of classes. This approach is relatively similar to label smoothing, so we would use a positive value of coefficient γ. Since we include a coefficient γ, we only need a fixed distribution u. The distribution does not need to be equal to $\frac{1}{K}$ as it could also be an indicator function giving value 1 for class $i \in \{c_1, c_2, c_3\}$ and 0 otherwise. The scaling could be a part of finding the best coefficient γ during hyperparameter tuning using the validation set.

It is also possible to have pre-defined class indices for smoothing when the maximum probability in output distribution $p_\theta(y|x)$ is for class k. Such an approach could be useful for more complicated datasets with many similar objects, for example, CIFAR-100 [Krizhevsky, 2009]. In fact, classes in CIFAR-100 belong to superclasses, which would make our approach particularly suitable for CIFAR-100. For example, superclass trees includes classes maple, oak, palm, pine, willow, and we could smooth only over these classes, if, for example, oak class has the largest probability. In terms of other datasets, for example, TIMIT speech recognition dataset [Garofolo et al., 1993], we could smooth over similar phones.

We have not conducted experiments using these related forms of regularization, but it could be done as part of future work, especially when using suitable datasets such as CIFAR-100 or TIMIT. We have not found these related regularizations in literature, and it is likely they are completely new.

Chapter 4

Description of Experiments

To confirm confidence penalty and label smoothing improve generalization of neural networks, we need to conduct experiments. In this chapter, we describe background of the experiments we have conducted, and we provide relevant technical information.

4.1 Overview of Experiments

We first conduct experiments using

- confidence penalty,
- label smoothing,
- dropout,
- no regularization.

Confidence penalty and label smoothing serve as our main point of experimentation, and dropout and no regularization serve for comparison. We could alternatively also explore what happens if we use confidence penalty or label smoothing on top of using dropout, but we have decided to compare them separately to have direct comparisons between the regularizations.

Next, we conduct experiments using extensions of confidence penalty:

- basic confidence penalty,
- thresholded confidence penalty,
- increasingly annealed confidence penalty,
- decreasingly annealed confidence penalty.

We perform both sets of experiments using SGD and Adam optimization methods.

4.2 General Network Architecture

Let us define the neural network architecture that we use in our experiments for the MNIST task. We use the network architecture and hyperparameters suggested in [Pereyra et al., 2017] so that we can have comparable results.

As we focus on permutation-invariant MNIST task that ignores spatial relations, we use fully-connected neural networks. We use ReLU activation function [Nair and Hinton, 2010], 1024 units per layer and two hidden layers. We show the architecture of our network in Figure 4.1, except that our network has 784 input units, 1024 units in both hidden layers and 10 output units. We have created the diagram using [Isaksson, 2017] library. We use 784 input units because each MNIST image has $28 \times 28 = 784$ features.

ReLU activation function is defined by

$$\text{relu}(x) = \max(0, x).$$

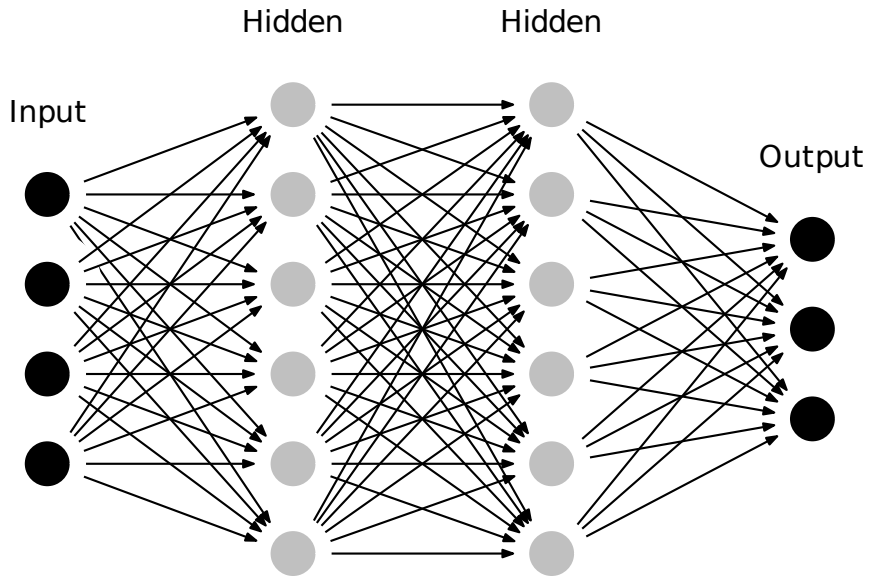

Figure 4.1: Diagram of a network with architecture similar to our network.

We initialize the weights of the network from a truncated normal distribution with mean 0 and standard deviation 0.01 (all generated values are within 2 standard deviations from the mean), constant 0 biases, and we use a batch size of 128 samples. [Pereyra et al., 2017] do not state all details of their experiments, for example, they do not mention the batch size for the MNIST task. However, to keep our experiments

consistent with previous publications in the area, we have chosen the batch size of 128, which is used in [Kingma and Ba, 2014].

We first optimize our models using SGD with learning rate of 0.05 as suggested in [Pereyra et al., 2017]. We use this learning rate for all experiments, including for dropout. [Pereyra et al., 2017] use learning rate of 0.001 for dropout, but based on our experiments, 0.05 allows efficient learning for dropout and makes the results more comparable among the regularizations. Second, we use Adam optimization with parameters as suggested in [Kingma and Ba, 2014]. This means we use learning rate of 0.001, β_1 of 0.9 and β_2 of 0.999. [Pereyra et al., 2017] have only evaluated their network for the MNIST task using SGD method.

We train models using various confidence penalty weights, label smoothing values and dropout rates to select the coefficients giving the most accurate predictions on the validation set. For confidence penalty, we try weights [0.1, 0.3, 0.5, 1.0, 2.0, 4.0, 8.0]. After selecting the best confidence penalty weight, we try confidence penalty thresholds of [0.1, 0.5, 1.0, 1.5, 2.0]. The maximum possible entropy for 10 classes is $\log 10 \approx 2.303$, and it is obtained for the uniform distribution. We try label smoothing values of [0.05, 0.1, 0.2, 0.3, 0.4, 0.5], and if the optimal label smoothing value is on the lower boundary, we also try values [0.005, 0.01, 0.02]. Our confidence penalty weights and label smoothing values are based on [Pereyra et al., 2017]. We also try to find the best dropout value, and we try all dropout rates between 0 and 0.9, taking steps of 0.1. [Pereyra et al., 2017] do not state the value of dropout they use in their experiments, but we believe it is useful to find the best dropout value to make effective comparisons between the regularizations. We apply dropout after each of the two hidden layers.

In terms of confidence penalty annealing, both strengthening and weakening, we introduce a parameter γ that describes the strength of confidence penalty at a given iteration. For strengthening annealing, parameter γ starts at 0, and we increase it by 0.1 up to 1 every 1000 steps. Afterwards, we multiply it by 1.01 every 1000 steps, and when the coefficient is larger than 2, we stop increasing it further. We have heuristically set the parameter to be equal to 1 after 10,000 iterations because 10,000 iterations seemed to be a point when the majority of training in terms of validation accuracy of the unregularized neural network was done - for both optimizations we have investigated. We multiply parameter γ with the best confidence penalty weight as obtained using hyperparameter tuning. For weakening annealing, parameter γ starts at 2, and then we decrease it by 0.1 down to 1 every 1000 steps. Afterwards, we multiply it by 0.99 every 1000 steps, and when the coefficient is smaller than 0.5, we stop decreasing it further. Our weakening annealing has been designed to be an inverse form of the strengthening annealing. Of course, we could also try different types of annealing.

We repeat all experiments 5 times with random weight initializations except for finding the optimal threshold that we repeated only 3 times. However, all our test results are reported for 5 trials. Repeating the experiments enables us to quantify the uncertainty in our results. Moreover, we repeat our hyperparameter tuning experiments multiple times because we want to choose a value which consistently gives good results and not only for a specific initialization. We choose the hyperparameter that achieves the highest mean validation accuracy across the multiple runs.

We use early stopping in our experiments. After no improvement in validation accuracy for 100,000 iterations, we stop and use the weights we saved when the highest validation accuracy was obtained. We train for a maximum of 1 million iterations, which corresponds to about 2560 epochs. Based on the plots shown in [Pereyra et al., 2017], they seem to train for a similar maximum number of iterations as we do.

4.3 Implementation

Our initial implementation of the network for classifying MNIST digits loosely followed the structure of the implementation in tutorial [TensorFlow, 2017a] with the full code given in [TensorFlow, 2017b]. The network used in [TensorFlow, 2017a] is significantly different from the one we use, but following the structure of their code helped us write well-structured code. We have extended our baseline implementation of the network significantly over time, adding new features as desired.

4.4 Running Experiments

In order to run experiments, we have mainly utilized MSc Teaching Cluster with GPUs that the School of Informatics offers. This involved learning how to run experiments using GridEngine [SGE Community, 2017], and it also involved setting up TensorFlow GPU framework and other libraries on the cluster, inside a virtual environment. Setting up GPU version of TensorFlow on the cluster turned out to include multiple challenges as it was necessary to correctly set various paths to NVIDIA CUDA [NVIDIA, 2018a] and cuDNN [NVIDIA, 2018b] libraries, and as a result the whole process took a few days. However, using GPU has been helpful for running the experiments as the experiments took significantly less time than they would take using CPU.

4.5 Features of the Experiments

When running our experiments, we keep track of the following in order to analyse the the effect of the regularization:

- weights and biases distributions over layers,
- t-SNE visualizations over layers,
- output distribution confusion matrices,
- densities of predicted probability magnitudes,
- training and validation error rate over training,
- training and validation gradient norm over training,
- distribution of entropies over training.

4.6 t-SNE Visualizations

t-SNE (stochastic neighbour embedding using t-distribution) [van der Maaten and Hinton, 2008] is a dimensionality reduction technique suitable for visualization of high-dimensional datasets. In our case, t-SNE visualizations help us analyse the behaviour of the network across layers.

We use t-SNE visualization from Scikit-Learn [Pedregosa et al., 2011], with the documentation available at [Scikit-Learn, 2017b]. We use the default parameters for t-SNE, and we use PCA (principal component analysis) available from [Scikit-Learn, 2017a] to decrease the dimensionality of inputs, outputs of the first and the second hidden layer to 50 dimensions. Reducing dimensionality to 50 is suggested in [Scikit-Learn, 2017b] and also used in [Maaten, 2014]. PCA is not a part of t-SNE algorithm and it needs to be done separately as part of pre-processing the data. We need to decrease the dimensionality to a smaller number of dimensions to run t-SNE algorithm in a reasonable amount of time. A downside of using PCA before t-SNE is PCA does not retain all variability in the data, however PCA can also eliminate some of the noise in the data. In our case, we have usually retained between 70% and 100% of the original variance of the data.

Chapter 5

Results and Analysis of Experiments

This chapter describes and analyses the results of our experiments. We first mention the values of the hyperparameters that we found to work best in terms of the lowest mean validation error rates. For clarity, we summarise their values in Table 5.1. We used these hyperparameter values in our further experiments, the results of which we analyse next. We have done experiments using two optimization methods, and we will first focus on the results obtained using SGD optimization.

5.1 Various Regularizations with SGD

5.1.1 Error Rates

After selecting the best coefficient for confidence penalty, label smoothing and dropout, we compare test error rates given by the various models. We have repeated the experiments 5 times to be able to find the mean test error rate and the standard error of the mean. We give the results in Table 5.2, and we clearly see confidence penalty achieves the lowest test error rates, followed by label smoothing and dropout. The results confirm confidence penalty and label smoothing help the network generalize better, even better than dropout. We will now explore why confidence penalty and label smoothing help the network generalize.

Let us compare the validation error rates during training for various regularizations to see the impact of confidence penalty, label smoothing and dropout on validation er-

	Dropout Rate	Label Smoothing Value	Confidence Penalty Weight	Confidence Penalty Threshold
SGD	0.7	0.3	0.3	2.0
Adam	0.4	0.02	0.3	1.5

Table 5.1: Values of the hyperparameters for various regularizations and both SGD and Adam optimization methods.

Regularization	Test Error Rate (%)
No regularization	2.056 ± 0.009
Dropout	1.588 ± 0.025
Label smoothing	1.384 ± 0.017
Confidence penalty	1.330 ± 0.020

Table 5.2: Test error rates for various regularizations when using SGD.

ror rates over training. In Figure 5.1, we show the validation error rates for the four regularizations. We see confidence penalty, label smoothing and dropout all lead to large improvements over the unregularized alternative. Confidence penalty leads to the lowest validation error rates, with label smoothing and dropout giving consistently marginally higher validation error rates, which confirms our expectation that regularization is helpful for improving generalization abilities of the models. As we use early stopping, the networks train for different numbers of iterations. The model with the best weights is the one 100,000 iterations before stopping.

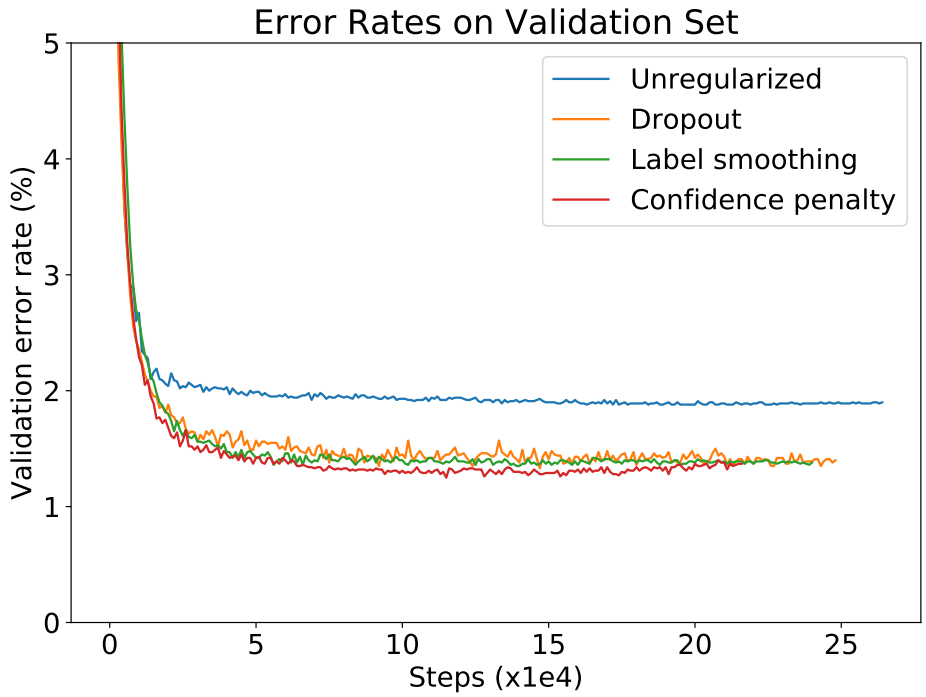

Figure 5.1: Validation error rates over training for various regularizations when using SGD optimization.

We see in Figure 5.1 that all four models have similar behaviour during about the first 10,000 iterations. However, the unregularized model then stops improving significantly, while the regularized models continue improving. This may be happening because of over-fitting to the training data, which dropout, confidence penalty and label smoothing aim to prevent.

5.1.2 Over-Fitting

To confirm over-fitting is a problem for the model with no regularization, we show the training and validation loss in Figure 5.2. We see over-fitting begins after about 10,000 steps. The over-fitting is not strong, but it may be the reason why the validation error rate stops decreasing for the unregularized model.

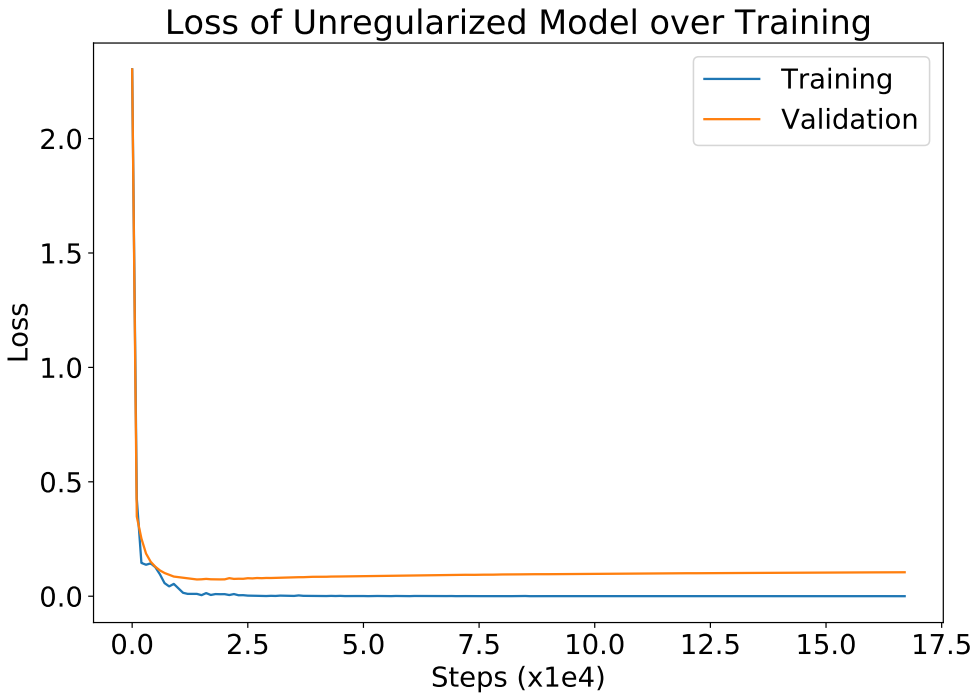

Figure 5.2: Training and validation loss over training for unregularized model, optimized using SGD.

To see if over-fitting is present for any of the regularized models, we show the validation loss of all four models in Figure 5.3. Based on the plot, it is not clear if over-fitting is present for any of the regularized models. However, even if over-fitting is present for any of them, it happens later and is significantly smaller than for the unregularized model. This may explain why validation error rates continue decreasing for the regularized models. Label smoothing has significantly larger validation loss than the other models, which we believe comes from the fact we use a relatively large label smoothing value that gives less target probability to the true label, which is also the one predicted the most often. However, the large loss does not seem to negatively impact the validation error rate as we have chosen the label smoothing value after hyperparameter tuning involving 5 repetitions.

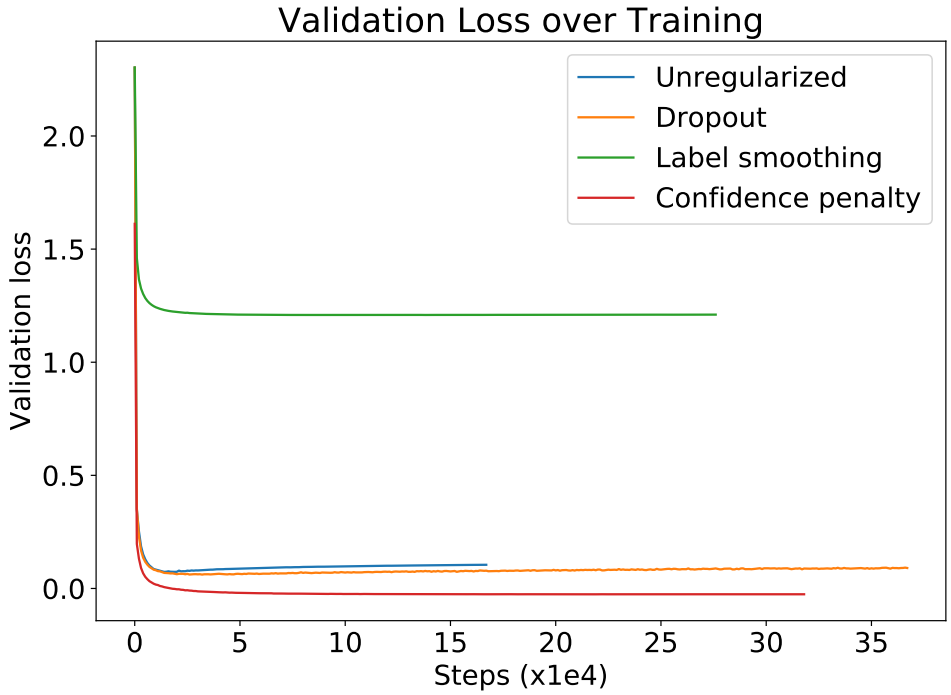

Figure 5.3: Validation loss over training for various regularizations, optimized using SGD.

5.1.3 Distribution of Output Probabilities

To see if confidence penalty and label smoothing actually lead to less confident predictions, we examine the distribution of magnitudes of output probabilities. Our results are shown in Figure 5.4, and we have used validation data and the model with the best weights obtained during training. We see both the unregularized model and the model using dropout give predictions with output probabilities either close to 0 or 1. This means these two models learn to give over-confident predictions. On the other hand, confidence penalty and label smoothing both give smoother output distributions, which means they achieve the aim of giving less confident predictions. As mentioned earlier, confident predictions are a symptom of over-fitting, and the fact we get less confident predictions should explain why both confidence penalty and label smoothing lead to better generalization.

With a similar aim as for Figure 5.4, we examine confusion matrices that show how much output probability a model gives to the various classes. From Figure 5.5, we see all four models give most probability to the correct class. However, the unregularized model and the model using dropout give significantly less probability to other classes than models using label smoothing or confidence penalty. This perfectly agrees with our findings from Figure 5.4. Confusion matrices allow us to explore how much probability the model assigns to specific classes, and as a result we see which classes are the most confusing for the model. For example, we can see the unregularized model

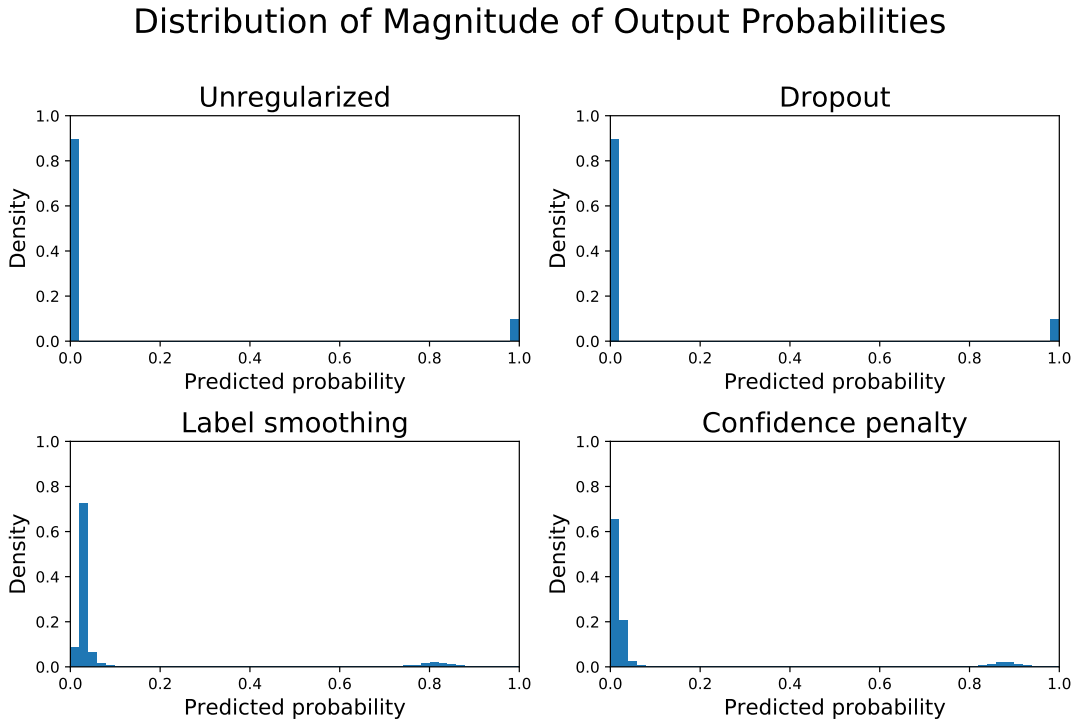

Figure 5.4: Distribution of magnitudes of output probabilities on the validation set for various regularizations, using SGD optimization.

often confuses digits 0 and 6 or 4 and 9, which makes sense because the digits look similar. However, when using label smoothing or confidence penalty, it is not easy to distinguish which classes the model confuses the most, which can be a disadvantage of using the regularization. For label smoothing, it is most likely caused by giving small target probabilities equally to all other classes, which results in small differences in output probabilities for incorrect classes. The situation with confidence penalty is similar, but the differences are marginally more visible.

5.1.4 t-SNE Visualizations

Next, we utilize t-SNE visualizations on the validation set to see how much using the neural network helps in separating the data. We first look at the t-SNE visualizations over layers as obtained by the model using confidence penalty. We show the visualizations in Figure 5.6. We see the visualized input data is significantly mixed among the classes, but with using further layers, the data, as visualized using t-SNE, become better and better separated into clusters. In the output layer, the outputs are clearly separated into clusters that mostly contain examples from only one class. There are a few examples that are part of an incorrect cluster, but their number is small. Having a closer look at the misclassified examples, they belong to classes that can be easily confused, for example digits 4 and 9.

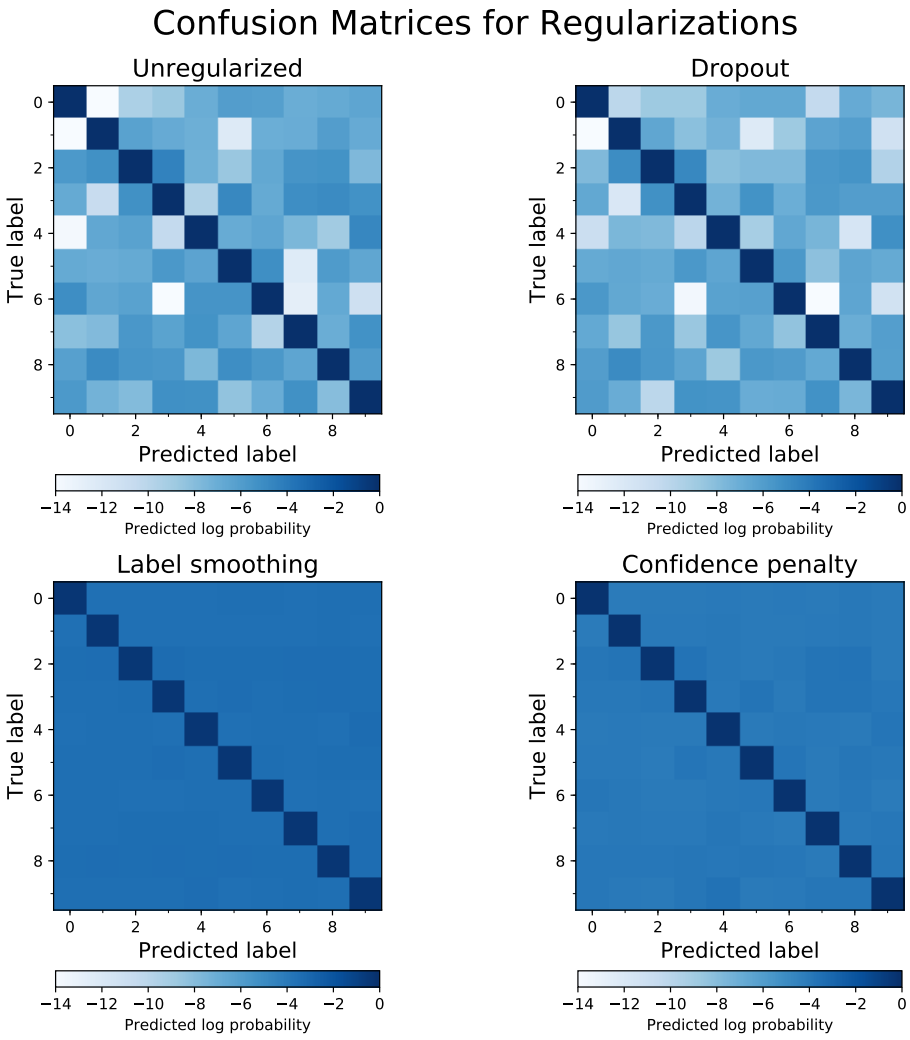

Figure 5.5: Confusion matrices of output distributions on validation set for various regularizations, using SGD optimization.

We next use t-SNE visualizations for outputs of the output layer as obtained using different regularizations. The results are shown in Figure 5.7. We see all models produce outputs that are separated into clusters, although the unregularized model produces outputs that are less clearly separated than it is in the case of confidence penalty, label smoothing or dropout. This is related to the fact that the regularized models achieve lower validation error rates, and hence classify or separate the data better. The precise shape of the clusters is not significant, as explained in [Wattenberg et al., 2016]. However, what is important is how clearly the clusters are separated and how many examples belong to incorrect clusters as this enables us to visually compare the effectiveness of various regularizations.

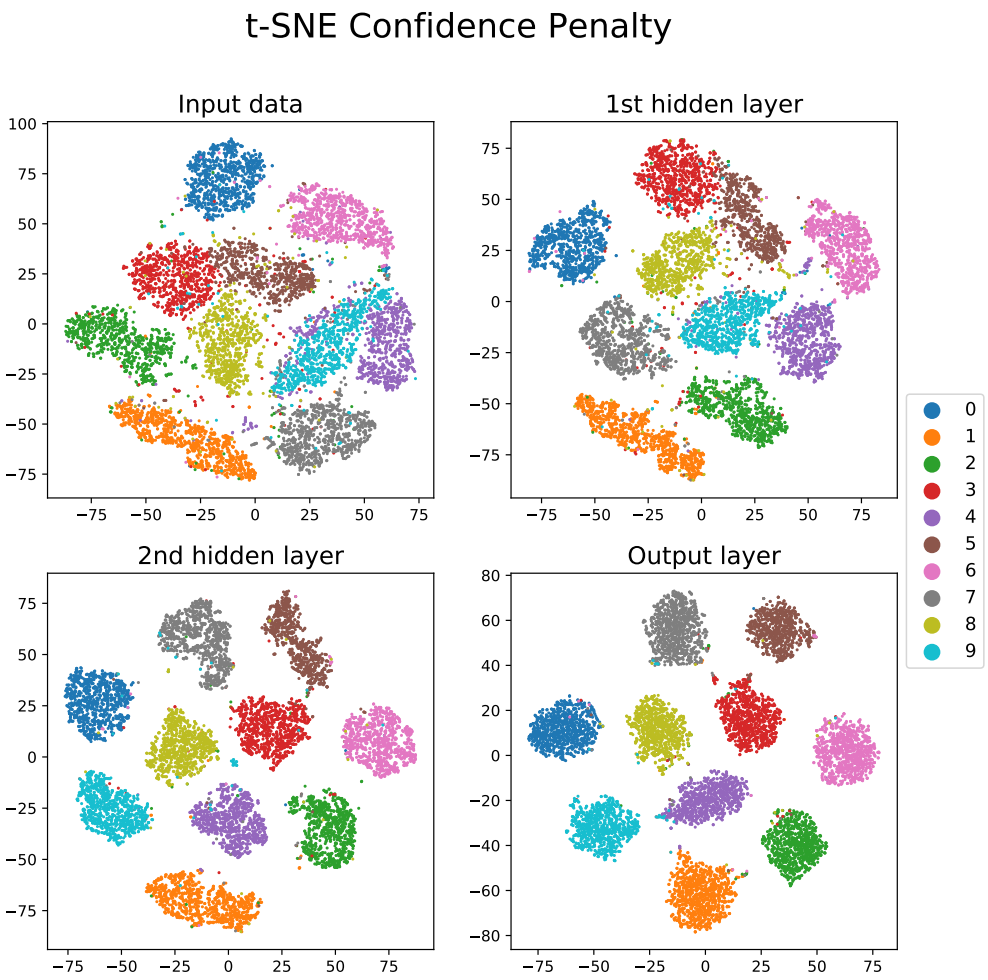

Figure 5.6: t-SNE visualization of outputs across layers for confidence penalty regularization, using SGD optimization.

5.1.5 Weight Distributions and Model Adaptability

It has been hypothesised in [Szegedy et al., 2016] that using label smoothing should lead to more adaptable models. We would like to find if using confidence penalty also leads to more adaptable models, and we would like to confirm it is truly the case for label smoothing. In order to find how adaptable the models are, we have decided to analyse the weights and biases of the models. We specifically look at their distribution obtained during the point of the lowest validation error rate. As a reminder, all our weights are initialized from a truncated normal distribution with mean 0 and standard deviation of 0.01. The biases are all initialized to 0.

We show the weights and biases distributions over layers and various regularizations in Figure 5.8. We have selected the number of bins to be 1000 for weights and 100

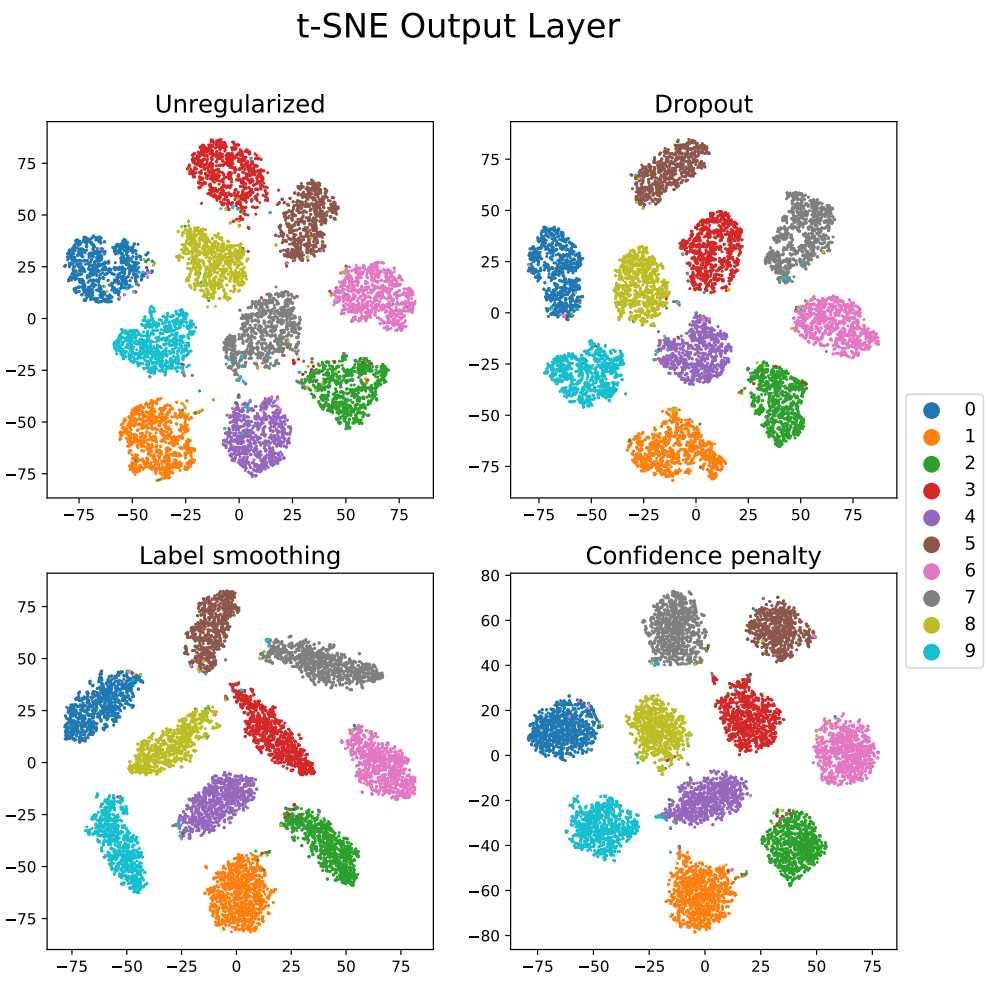

Figure 5.7: t-SNE visualization of output layers given by various regularizations, using SGD optimization.

for biases. 1000 bins correspond approximately to the square root of the number of weights per layer. We have decided not to show the counts in histograms as it is not relevant to our analysis, however we include the range of weight values as it is potentially useful information for comparison. The range of counts for a fixed layer is the same across regularizations.

We see the distribution of weights in the first and second hidden layer is approximately the same for models without regularization, label smoothing and confidence penalty. However, the biases in the first and the second layer are clearly more varied than without regularization, which shows confidence penalty and label smoothing both lead to more adaptable models. Output weights for label smoothing and confidence penalty have marginally wider span than for the unregularized model, which is again a further indication confidence penalty and label smoothing both lead to more adaptable models.

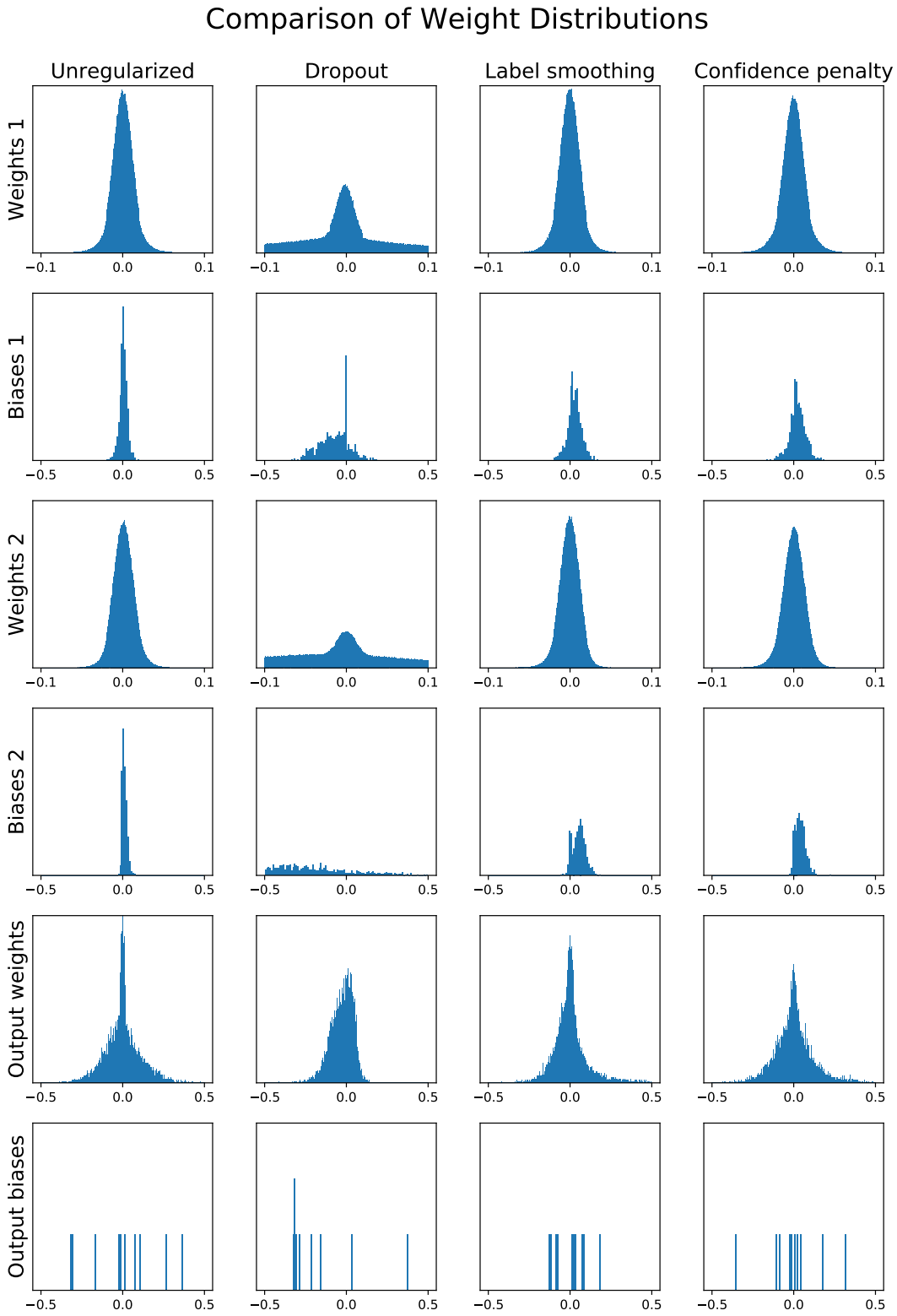

Figure 5.8: Comparison of weight and bias distributions over layers for various regularizations, model optimized using SGD.

While there seem to be improvements in adaptability due to using confidence penalty or label smoothing, the improvements do not seem to be particularly large. However, the weights and biases for dropout are significantly more varied across layers compared to the other regularizations. This is likely because of the way dropout works - only some of the units are used at a time. As a result, using fewer units may result in larger modifications in weights for the units that are used. However, the units are updated less frequently than without dropout.

5.2 Extensions of Confidence Penalty with SGD

5.2.1 Results

We have explored various extensions of confidence penalty. In particular, we investigated thresholding of confidence penalty and confidence penalty annealing, both strengthening and weakening. For confidence penalty thresholding, as part of hyperparameter tuning we found threshold of 2.0 works the best among the ones considered. We report test error rates for various extensions in Table 5.3. We see increasingly annealed confidence penalty and basic confidence penalty work the best, with increasingly annealed confidence penalty having a lower mean test error rate. However, the error bounds for them overlap. Consequently, it may be sufficient to use the basic confidence penalty as it is quite difficult to choose good annealing in general. Using a threshold did not help, and weakening annealing achieved the worst results among the extensions. We have decided to report the results on a test set because we only aim to give information about the final performance of the models and not use them further.

To understand why confidence penalty extensions achieve the results given in Table 5.3, we analyse the distribution of magnitudes of output probabilities using validation data. We show the distributions in Figure 5.9. We notice increasingly annealed confidence penalty smooths the output distributions the most, which is probably the reason why it achieves the lowest test error rate as over-confident output distributions have been associated with over-fitting and worse generalization. The distributions for basic and thresholded confidence penalty look very similar, and they both achieve, in fact, similar results. Decreasingly annealed confidence penalty achieves the worst results, and it has a peaked distribution which goes against the aim of using confidence penalty. Based on these results, it seems strengthening annealing could be helpful, while weakening annealing not.

CONFIDENCE PENALTY	TEST ERROR RATE (%)
BASIC	1.330 ± 0.020
THRESHOLDED (2.0)	1.376 ± 0.010
INCREASINGLY ANNEALED	1.318 ± 0.017
DECREASINGLY ANNEALED	1.442 ± 0.029

Table 5.3: Test error rates for extensions of confidence penalty when using SGD.

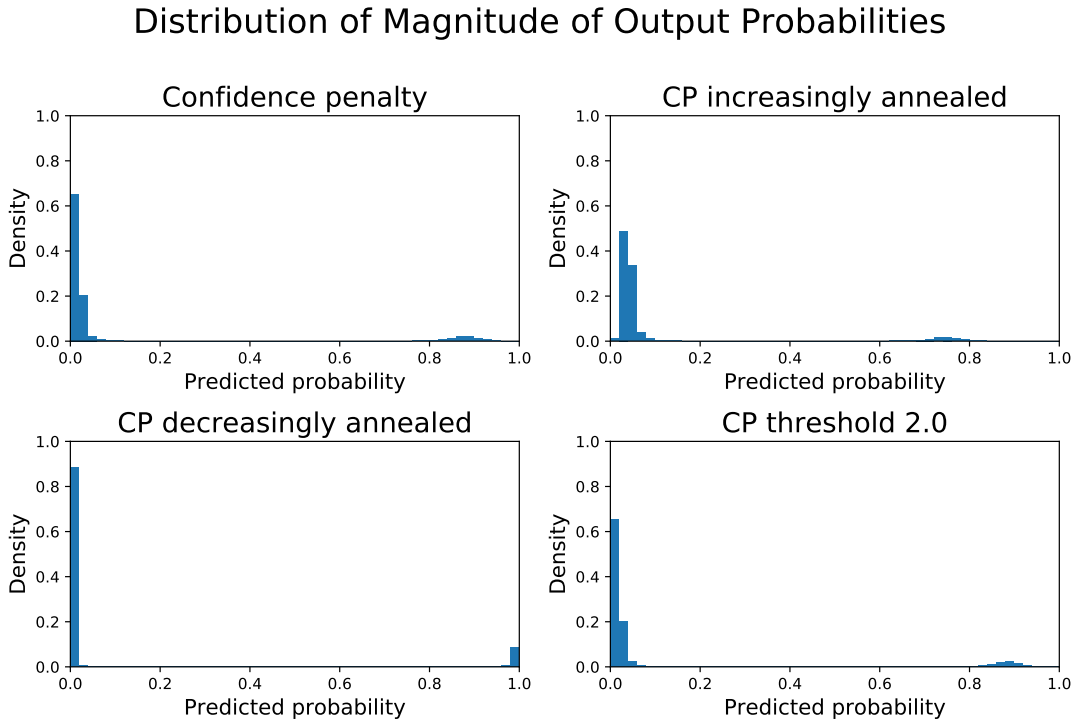

Figure 5.9: Distribution of magnitudes of output probabilities for extensions of confidence penalty, using validation set and SGD optimization.

5.2.2 Analysis of Thresholding

In order to understand the effect of thresholding better, we examine the distribution of output probabilities for various threshold values. We examine both histograms of magnitudes of output distributions and confusion matrices in Figure 5.10. We see that if we use low threshold values, the effect of confidence penalty on the output distributions is small. Threshold of 1.0 has more impact on smoothing the output distribution than thresholds of 0.1 or 0.5. We believe this is because confidence penalty is applied significantly less frequently in such cases and so has only small impact. When using a threshold above 1.0, we could not see noticeable differences between threshold of 1.0 and larger thresholds, and because of that we do not include plots for larger threshold values. The results may be similar because the penalty is applied similarly often for larger threshold values.

To give more context for the entropy threshold values, we give examples of distributions having approximately the given entropy values (each example has 10 elements):

- entropy 0.1: $[0.0001, ..., 0.0001, 0.9991]$,
- entropy 0.5: $[0.01, ..., 0.01, 0.91]$,
- entropy 1.0: $[0.025, ..., 0.025, 0.775]$.

In order to understand why larger confidence penalty threshold values work better than the smaller ones, we show the ratios of confident output predictions on validation data during training in Figure 5.11. We use various level of confidence - confident output predictions are those that have an entropy lower than the given threshold value. As part of the plot, we show the ratios of confident predictions for all four regularizations as this will help us better understand the behaviour of the regularizations.

In terms of confidence penalty, we see that with entropy thresholds such as 0.1 and 0.5, only a relatively small number of predictions is confident. This is particularly noticeable towards the end of the training when most predictions are not confident. This means confidence penalty has fulfilled its aim since the number of confident predictions is low. As for no regularization and dropout, both of them result in larger and larger ratios of confident predictions, including very confident predictions with entropies less than 0.1. We see that without targetting the confidence of output distributions, confident predictions become prevalent.

Let us examine what this means in terms of extensions of confidence penalty. Entropy thresholding as a form of making the penalty stronger could make sense because the ratio of confident prediction increases over time if confidence penalty is not used. As a result, the threshold would result in the penalty being stronger towards the end of the training as it would be applied more often. However, this ignores interaction with applying confidence penalty to the model, while the interaction makes things a lot more complicated.

When applying confidence penalty, the ratio of confident predictions becomes significantly smaller over time, and as a result, the thresholded penalty would result in applying the penalty rarely towards the end of the training. As a result, it would weaken the penalty over training, going against the original idea. What exactly would happen most likely depends on the specific value of the threshold chosen, but our results for confidence penalty thresholding suggest thresholding achieves marginally worse results than no thresholding. However, even if thresholding would help, a relevant question arises if it is worth trying to find a good threshold value that would work well on the particular problem.

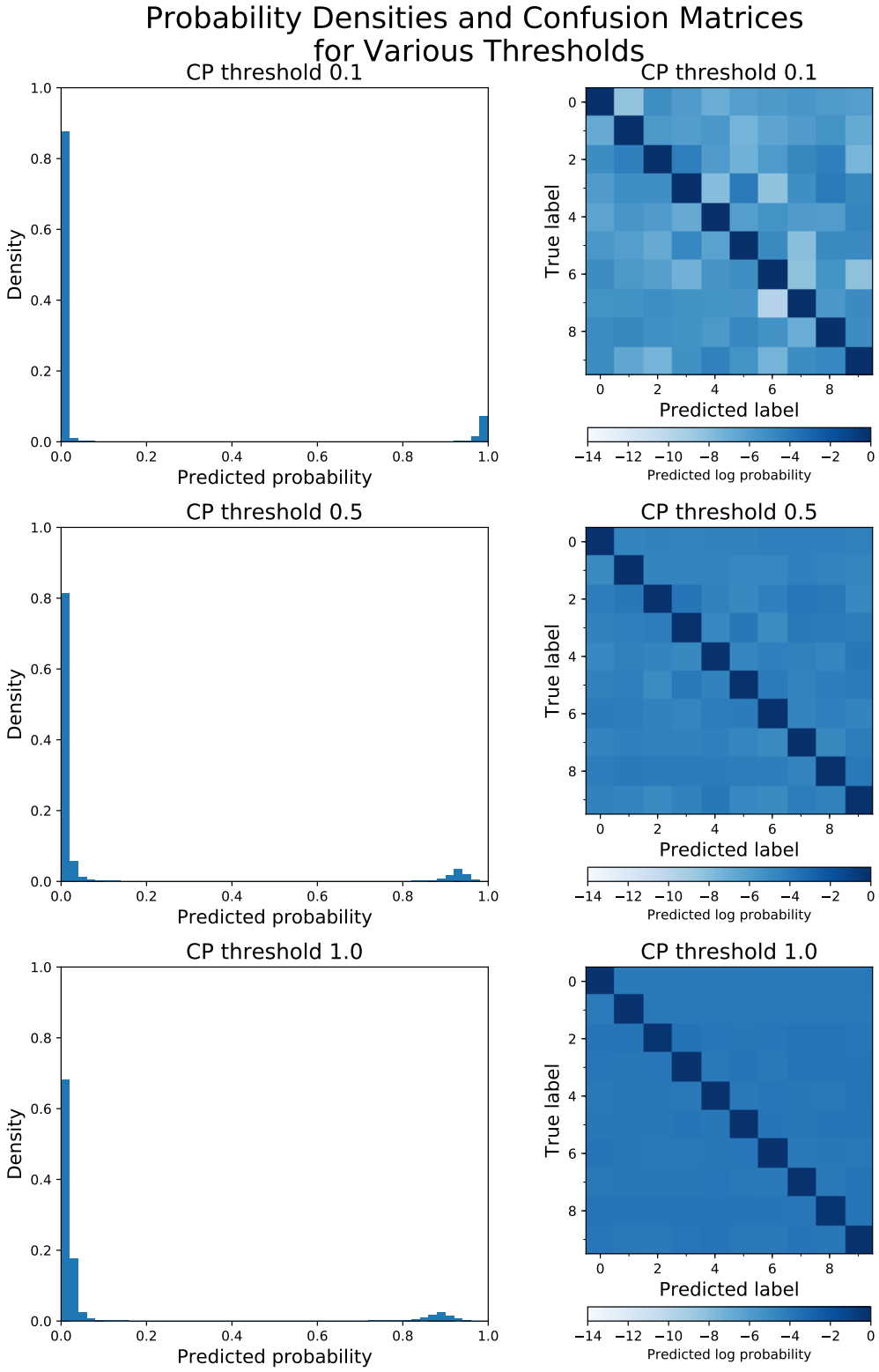

Figure 5.10: Distribution of magnitudes of output probabilities and confusion matrices for several confidence penalty threshold values, using validation set and SGD.

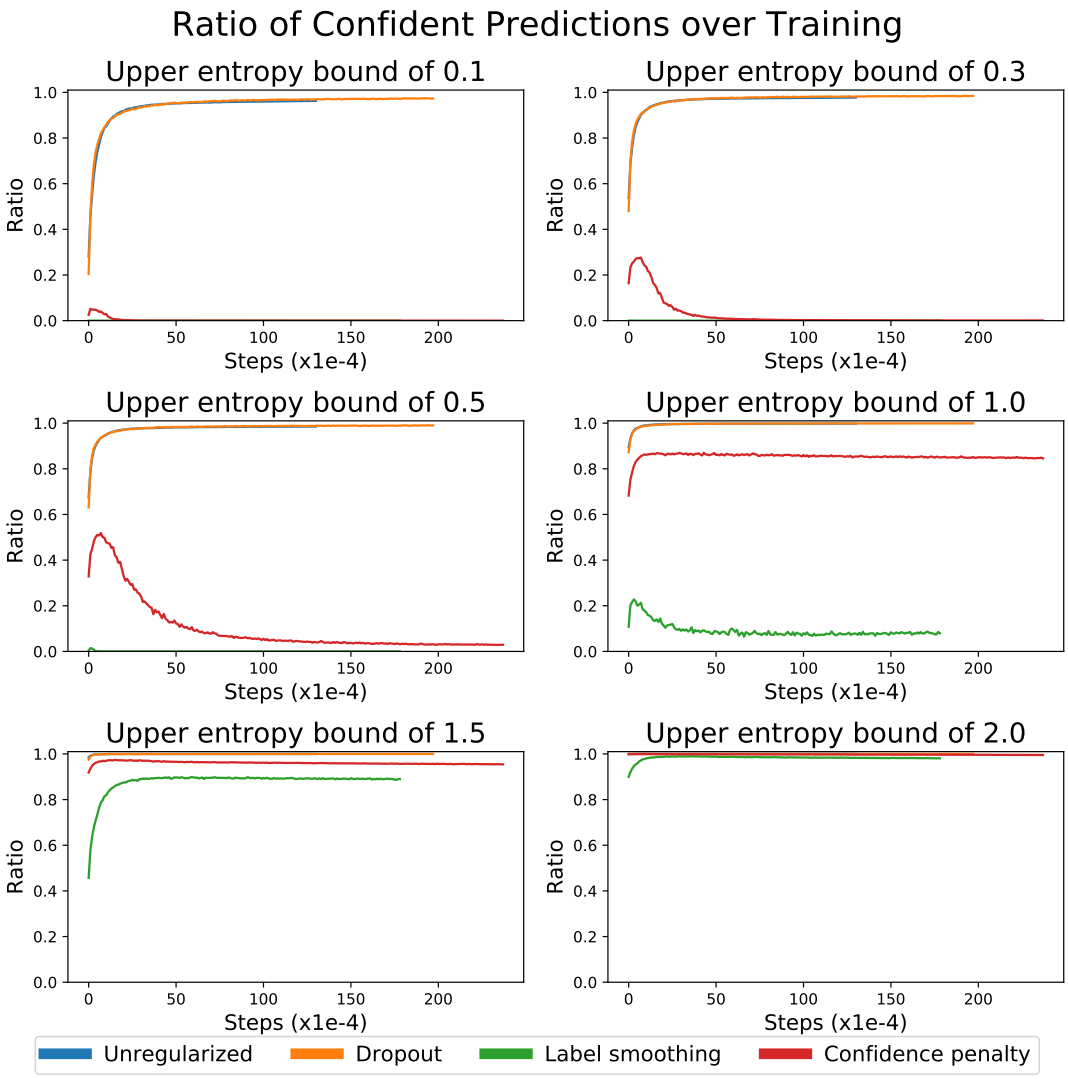

Figure 5.11: Ratios of confident predictions during training for various regularizations. Several threshold values for confidence are shown, and validation set and SGD optimization is used.

5.2.3 Implications for Confidence Penalty Extensions

Figure 5.11 suggests making the confidence penalty stronger in later stages could help because there are only few confident predictions when using confidence penalty. As a result, we would encourage the network to produce even smoother predictions when there is less space to produce smoother distributions. Our results on the test set suggest this hypothesis is sensible as the annealed confidence penalty achieves marginally lower test error rate than basic confidence penalty. However, the difference is very small and the error bounds for strengthening annealing and basic confidence penalty overlap, which means we cannot be certain that the annealing works better. Moreover, it is important to find good annealing because too smooth output distributions may not be helpful for better generalization. We need to find the right balance, and this may require a lot of hyperparameter tuning that may not be worth a small improvement in error rates for unseen data.

Furthermore, we notice label smoothing with our chosen label smoothing value produces even smoother (less confident) predictions than confidence penalty. However, based on our results, label smoothing achieves a worse test error rate than confidence penalty, which is an indication we may need to avoid very strong smoothing.

5.3 Various Regularizations with Adam

We have also performed experiments using Adam optimizer to examine if the positive impact of confidence penalty and label smoothing remains when using another very popular optimization method.

We give the results we have obtained using the best hyperparameters on the test set in Table 5.4. We notice models using no regularization or dropout achieve significantly lower test error rates than with SGD optimizer. This finding agrees with the results reported in [Kingma and Ba, 2014] that Adam optimizer leads to improvements in accuracy compared to SGD. The test error rates for label smoothing and confidence penalty remain on relatively similar values as with SGD, suggesting they work better than dropout or no regularization when using Adam optimizer. The benefit of using confidence penalty and label smoothing is smaller than with SGD, but we believe still worth utilizing. We have conducted a similar in-depth analysis as we did with SGD optimizer, and our findings remain similar. Consequently, we do not describe them in the report as the points mentioned previously still hold.

Regularization	Test Error Rate (%)
No regularization	1.496 ± 0.069
Dropout	1.424 ± 0.070
Label smoothing	1.332 ± 0.017
Confidence penalty	1.348 ± 0.038

Table 5.4: Test error rates for various regularizations when using Adam.

5.4 Extensions of Confidence Penalty with Adam

Similarly as with SGD, we explored extensions of confidence penalty when using Adam optimizer. We summarise our findings in Table 5.5. We notice that on average, basic confidence penalty achieves the smallest test error rate. However, all four variations of confidence penalty have similar mean test error rates. Moreover, their error bounds overlap, which means we cannot be certain about the differences in their performance. The results suggest that with Adam optimizer the variations of confidence penalty do not matter much, and consequently it seems the best is to use the basic confidence penalty as we can achieve a similar performance while using fewer hyperparameters.

CONFIDENCE PENALTY	TEST ERROR RATE (%)
BASIC	1.348 ± 0.038
THRESHOLDED (1.5)	1.370 ± 0.035
INCREASINGLY ANNEALED	1.376 ± 0.029
DECREASINGLY ANNEALED	1.362 ± 0.025

Table 5.5: Test error rates for extensions of confidence penalty when using Adam.

5.5 Discussion of Results

We have confirmed confidence penalty and label smoothing both lead to improved generalization of the model. The improvements are particularly significant when using SGD optimization. With Adam optimization, the improvements are present as well but the improvements in generalization are smaller due to better-performing baselines. We have also investigated extensions of confidence penalty, and we have shown it is most likely the best to use the basic confidence penalty even though strengthening annealing may work marginally better when using SGD.

We designed our experiments to be comparable with results published in literature, so let us compare them now. [Pereyra et al., 2017] used SGD and showed improvements using confidence penalty and label smoothing for our network, which agrees with our conclusions. We describe the results they achieved in Table 5.6 for permutation-invariant MNIST task. If we allowed learning spatial relations between features, the MNIST task test error rates could reach as low as 0.21% that have been obtained using a technique called DropConnect [Wan et al., 2013]. However, our task was to evaluate in detail the usefulness of confidence penalty and label smoothing on a standard dataset for a standard network and not build the best possible classifier for the MNIST task.

Our test error rates do not reach as low values as the ones mentioned in [Pereyra et al., 2017]. However, we reach the same conclusion: confidence penalty and label smoothing help in improving the generalization abilities of the network. Moreover, our results agree the improvement in the test error rate is larger than with dropout regularization.

Regularization	Test Error Rate (%)
Dropout	1.28 ± 0.06
Label smoothing	1.23 ± 0.06
Confidence penalty	1.17 ± 0.06

Table 5.6: Test error rates obtained in [Pereyra et al., 2017] for various regularizations when using SGD.

Let us discuss why our test error rates may be worse than the ones mentioned in [Pereyra et al., 2017]. One of the main factors could be they retrained the models using the full training dataset after tuning the hyperparameters. We have not retrained the models as we have rather focused on investigating the impact of confidence penalty and label smoothing in detail. There are also some hyperparameters, for example, batch size, which [Pereyra et al., 2017] do not describe in their paper, and consequently it is likely the values of some of our hyperparameters differ.

Chapter 6

Further Work - Speech Recognition

We started to explore the impact of confidence penalty and label smoothing also for the TIMIT dataset [Garofolo et al., 1993] as an optional extension of our work. The TIMIT dataset is a standard speech recognition dataset, which [Pereyra et al., 2017] use as well. [Pereyra et al., 2017] evaluate the impact of confidence penalty and label smoothing on a sequence-to-sequence model with attention [Bahdanau et al., 2014] that does not have a very strong baseline performance. In fact, they mention other alternative models that perform better, and we have decided to evaluate one of them.

We use a DNN-HMM (Deep Neural Network-Hidden Markov Model) model proposed by [Mohamed et al., 2012] to conduct our experiments. This model is commonly used and, for example, it is a part of the popular Kaldi automatic speech recognition (ASR) toolkit [Povey et al., 2011]. We believe using a commonly used model will enable us to show better the practical impact of confidence penalty and label smoothing.

6.1 Framework Set-Up and Pre-Processing

As we do most of our work in TensorFlow, we have decided to use TFKaldi library [Renkens, 2017] that uses TensorFlow to train a neural network described in [Mohamed et al., 2012] while using Kaldi alignments as targets. To train a neural network using TFKaldi, we need to do a relatively large amount of pre-processing of the TIMIT data using Kaldi toolkit. To do the pre-processing correctly, we closely followed the instructions from labs for ASR course [Renals and Shimodaira, 2018]. In summary, the pre-processing involved the following:

- setting-up Kaldi,
- data preparation,
- feature extraction,
- building a HMM-GMM (HMM-Gaussian Mixture Model) monophone model,
- creating alignments.

Setting up Kaldi and TFKaldi took a relatively long time as there were various complications involved. One of the complications was that TFKaldi uses an old version of TensorFlow, and as a result, we had to fix various problems with outdated TensorFlow syntax. TFKaldi was last updated in January 2017, but TensorFlow evolves very quickly, which causes some relatively new libraries become incompatible with new TensorFlow versions. Moreover, we met additional complications when setting up Kaldi and TFKaldi in a virtual environment on a GPU cluster. As a result, setting up the software on the GPU cluster took a longer time than we expected.

6.2 Baseline Experiments

Our next task was to find a suitable network architecture that would perform well on the given problem. We split the training set into training and validation set with ratios 0.9 and 0.1. We then used the validation set to find the best phone error rate (PER) given by the model. We used all 61 phones during training and decoding, and we used relevant Kaldi script to reduce the 61 phones to 39 phones during scoring to calculate PER. Based on the neural networks investigated in [Mohamed et al., 2012], we decided to try networks with 4, 5 and 6 layers with either 1024 or 2048 units per layer (6 networks together).

The best PERs on the validation set varied between 29% and 35% with the best results obtained for the networks with 6 layers. The network with 6 layers and 2048 units per layer obtained PER lower by only about 0.2% compared to the network with 6 layers and 1024 units. Moreover, it took significantly less time to train the network with 1024 units per layer compared to the one with 2048 units per layer. As a result, we have decided to use a network with 6 layers and 1024 units per layer in our further experiments. In terms of other parameters, we used the default values given in the TFKaldi config file to train the network.

The results we obtained using the previously described networks were significantly worse than the ones mentioned in [Mohamed et al., 2012]. We tried to use a similar architecture and similar parameters to the ones used by [Mohamed et al., 2012], however, their PERs on the validation set varied between about 20.5% and 21.5%. We tried to improve our model by including energy parameter in our features and adding double delta dynamic information to the features, but our results remained similar as before. As mentioned in [Meyer, 2016], the quality of the resulting DNN-HMM model is also affected by the quality of the HMM-GMM model that we trained initially, so it may be possible that [Mohamed et al., 2012] used a better HMM-GMM model.

6.3 Regularization

Despite worse results than we expected, we tried to apply confidence penalty, label smoothing and dropout to our selected network. Dropout was already an optional parameter in the network provided by TFKaldi, so we only needed to specify a value.

We performed an initial experiment with a dropout rate of 0.15. The result was similar as without any dropout, in fact, it was marginally worse, but that may be due to randomness.

As confidence penalty and label smoothing was not a part of the network provided by TFKaldi, we implemented it on our own, following the approach we used for the MNIST dataset. We have also added an option to specify the confidence penalty weight and label smoothing value directly inside the TFKaldi config files.

Despite following implementation that worked well for MNIST, we achieved PER of about 90% for both confidence penalty and label smoothing. We tried various confidence penalty weights and label smoothing values, but without any substantial improvement in PER. In fact, it is likely the obtained PERs were too high to be a result of using high confidence penalty weights or label smoothing values. Consequently, our application of confidence penalty and label smoothing was most likely not compatible with the implementation of the network provided by TFKaldi, but we have not been able to find the source of the problem. Since our baseline PERs were significantly worse than we expected, and we were not able to use confidence penalty or label smoothing with TFKaldi, we decided to explore if we can use Keras-Kaldi library [Kumar, 2017] instead.

6.4 Keras-Kaldi Library

Setting up Keras-Kaldi, similarly as TFKaldi, took a certain amount of time. We originally planned to work with TensorFlow, but since we did not find an alternative library using TensorFlow, we decided to use Keras. Our initial experiments using a network with 6 layers and 1024 units per layer resulted in PER of about 27% on the validation set, which is quite a significant improvement compared to our results using TFKaldi. However, it is still significantly worse than what [Mohamed et al., 2012] report. Similarly as before, we tried to implement confidence penalty and label smoothing. We managed to run experiments using confidence penalty, but PER obtained on the validation set was again unacceptably high, in fact about 75%.

We then noticed a significant problem with Keras-Kaldi library. It appears Keras-Kaldi does not use one-hot encoded targets but rather integer targets, which makes label smoothing and most likely also confidence penalty not compatible with the library unless we change possibly substantial parts of it. We checked TFKaldi before implementing confidence penalty and label smoothing, and it should use one-hot encoded targets, which means the problem with our results obtained for TFKaldi is most likely different. We stopped at this point due to running out of time, and having already done a lot of research into confidence penalty and label smoothing using the MNIST dataset.

Chapter 7

Conclusions

7.1 Summary and Evaluation

In our work, we evaluated in detail the impact of confidence penalty and label smoothing regularization using a standard dataset MNIST. We explored both the theoretical and practical aspects of the regularizations.

On the theoretical side, we showed how precisely confidence penalty and label smoothing are related to each other via the KL divergence. Based on this connection, we generalized the regularizations and proposed new related forms of regularization. Our theoretical investigation also involved deriving gradients of the loss functions using confidence penalty and label smoothing, showing they have relatively simple forms. Gradients with simple forms allow using confidence penalty and label smoothing efficiently as part of back-propagation. Moreover, we explained why confident output distributions are a symptom of over-fitting, and how confidence penalty and label smoothing could solve the problem.

We performed experiments using TensorFlow framework, and our experiments were designed to systematically evaluate the impact of using confidence penalty and label smoothing as opposed to using dropout or no regularization. We performed experiments using SGD and Adam optimizers, confirming confidence penalty and label smoothing improve generalization abilities of neural networks for both optimizations. We did multiple repetitions of our experiments to reach conclusions that are less affected by randomness, which typically affects training neural networks. As part of our experiments, we examined in detail various features of the models, for example, the distributions of weights over layers and various regularizations, showing confidence penalty, label smoothing and dropout lead to an increased flexibility of the model. We also analysed histograms and confusion matrices of output distributions for various regularizations, confirming both confidence penalty and label smoothing make the output distributions smoother.

As a further key part of our work that has not been investigated in detail in literature so far, we evaluated extensions of confidence penalty - thresholding and annealing of confidence penalty. We tried both strengthening and weakening annealing. In order to

understand why extensions of confidence penalty may or may not work, we examined the ratios of confident outputs for various regularizations as the training proceeded. After analysing the results, we discovered it is not needed to use extensions of confidence penalty as we can achieve comparable results using the basic version of confidence penalty, while using fewer hyperparameters. However, we suggested strengthening annealing may lead to better performance, particularly when using SGD, noting it may be difficult to choose good annealing parameters for a particular problem.

We have also done further work using the TIMIT speech recognition dataset, but we have not been able to resolve all complications we met due to lack of time.

All in all, we have performed a detailed investigation of confidence penalty and label smoothing for a selected standard deep learning dataset. We explored both the theoretical and practical aspects, and we have confirmed confidence penalty and label smoothing are two related regularizations that work well in practice.

7.2 Further Related Approaches

Confidence penalty and label smoothing are not the only approaches that regularize the model using either the output or target distribution. For example, [Xie et al., 2016] regularize neural networks by adding noise to the training data labels, which in practice means changing some of the labels to incorrect labels. Moreover, [Pereyra et al., 2017] mention it is possible to smooth labels using a teacher model [Hinton et al., 2015] or also based on the model's own distribution as done in [Reed et al., 2014].

7.3 Future Work

The next task could be to investigate the impact of confidence penalty and label smoothing on other datasets. One suitable dataset is the TIMIT speech recognition dataset, which we tried to use as well. [Pereyra et al., 2017] evaluated the performance of confidence penalty and label smoothing on various datasets, including TIMIT, and so it may not be necessary to focus on exploring the effect of confidence penalty and label smoothing across various tasks. In our work, we pursued a different approach than [Pereyra et al., 2017] as we focused on evaluating confidence penalty in depth, and we explored also its extensions and the connection to label smoothing.

However, what could actually be worth studying are the new regularizations based on label smoothing that we proposed in Chapter 3. It is possible they could lead to further improvements in classification accuracy on unseen data, and we believe it could be worth to explore them in the future.

It would be also particularly interesting to explore if it is possible to use the idea of confidence penalty for a regression task. Confidence penalty is based on probability output distributions given by the neural networks, but it could be possible to use the idea to create a new form of regularization useful for regression.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Accessed on 4/4/2018.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2014. URL `http://arxiv.org/abs/1409.0473`.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems. In *NIPS*, 2015. URL `http://arxiv.org/abs/1512.01274`.

François Chollet. Keras, 2015. URL `https://keras.io`. Accessed on 4/4/2018.

Rob DiPietro. A friendly introduction to cross-entropy loss, 2016. URL `https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/`. Accessed on 25/3/2018.

Abhimanyu Dubey, Otkrist Gupta, and Ramesh Raskar. Regularizing prediction entropy enhances deep learning with limited data. 2017. URL `https://pdfs.semanticscholar.org/ccae/b0ee222cbd43c519a282e51b1cc867f10361.pdf`.

John S. Garofolo, Lori F. Lamel, William M. Fisherand Jonathan G. Fiscus, David S. Pallett, Nancy L. Dahlgren, and Victor Zue. The DARPA TIMIT acoustic-phonetic continuous speech corpus. 1993. URL `https://catalog.ldc.upenn.edu/ldc93s1`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua

Bengio. Maxout networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1319–III–1327. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3043084.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. pages 6645–6649, 2013. ISSN 1520-6149. doi: 10.1109/ICASSP.2013.6638947. URL http://ieeexplore.ieee.org/document/6638947/.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. URL http://arxiv.org/abs/1503.02531.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456. PMLR, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

Martin Isaksson. Create a drawing of a feed-forward neural network., 2017. URL https://github.com/martisak/dotnets. Accessed on 4/4/2018.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, 2009. doi: 10.1109/ICCV.2009.5459469. URL http://ieeexplore.ieee.org/document/5459469/.

Edwin Thompson Jaynes. Information theory and statistical mechanics. II. *Phys. Rev.*, 108:171–190, 1957. doi: 10.1103/PhysRev.108.171. URL https://link.aps.org/doi/10.1103/PhysRev.108.171.

Andrej Karpathy. A peek at trends in machine learning. 2017. URL https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014. URL http://arxiv.org/abs/1412.6980.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Pavan Kumar. Keras interface for Kaldi ASR, 2017. URL https://github.com/dspavankumar/keras-kaldi. Accessed on 1/4/2018.

Yann LeCun and Corinna Cortes. The MNIST database of handwritten digits, 1998. URL http://yann.lecun.com/exdb/mnist/. Accessed on 4/4/2018.

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. volume 86, pages 2278–2324, 1998. doi: 10.1109/5.726791. URL http://ieeexplore.ieee.org/document/726791/.

Laurens Van Der Maaten. Accelerating t-SNE using tree-based algorithms. *J. Mach.*

Learn. Res., 15(1):3221–3245, 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2697068.

Josh Meyer. How to train a deep neural net acoustic model with Kaldi, 2016. URL http://jrmeyer.github.io/asr/2016/12/15/DNN-AM-Kaldi.html. Accessed on 4/4/2018.

Abdel R. Mohamed, George E. Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, 2012. ISSN 1558-7916. doi: 10.1109/TASL.2011.2109382. URL http://ieeexplore.ieee.org/document/5704567/.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL http://dl.acm.org/citation.cfm?id=3104322.3104425.

Andrew Y. Ng. Feature selection, L1 vs. L2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 78–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015435. URL http://doi.acm.org/10.1145/1015330.1015435.

Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL http://neuralnetworksanddeeplearning.com/chap3.html.

NVIDIA. CUDA toolkit, 2018a. URL https://developer.nvidia.com/cuda-toolkit. Accessed on 31/3/2018.

NVIDIA. NVIDIA cuDNN, 2018b. URL https://developer.nvidia.com/cudnn. Accessed on 31/3/2018.

Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-Learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL http://dl.acm.org/citation.cfm?id=1953048.2078195.

Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey E. Hinton. Regularizing neural networks by penalizing confident output distributions. In *ICLR*, 2017. URL http://arxiv.org/abs/1701.06548.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The Kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, 2011. URL http://kaldi-asr.org/. IEEE Catalog No.: CFP11SRW-USB.

Scott E. Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. 2014. URL http://arxiv.org/abs/1412.6596.

Steve Renals and Hiroshi Shimodaira. Automatic speech recognition (ASR) 2017-18: Labs, 2018. URL https://www.inf.ed.ac.uk/teaching/courses/asr/labs.html. Accessed on 25/3/2018.

Vincent Renkens. Kaldi with TensorFlow neural net, 2017. URL https://github.com/vrenkens/tfkaldi. Accessed on 1/4/2018.

Peter Roelants. How to implement a neural network intermezzo 2, 2017. URL http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/. Accessed on 28/3/2018.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL http://dl.acm.org/citation.cfm?id=65669.104451.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. URL http://dx.doi.org/10.1007/s11263-015-0816-y.

Scikit-Learn. sklearn.decomposition.PCA, 2017a. URL http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. Accessed on 30/3/2018.

Scikit-Learn. sklearn.manifold.TSNE, 2017b. URL http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html. Accessed on 30/3/2018.

SGE Community. Son of Grid Engine, 2017. URL https://arc.liv.ac.uk/trac/SGE. Accessed on 31/3/2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. pages 1–9, 2015. ISSN 1063-6919. doi: 10.1109/CVPR.2015.7298594. URL http://ieeexplore.ieee.org/document/7298594/.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. pages 2818–2826, 2016. doi: 10.1109/CVPR.2016.308. URL http://ieeexplore.ieee.org/document/7780677/.

TensorFlow. Deep MNIST for experts, 2017a. URL https://www.tensorflow.org/versions/r1.3/get_started/mnist/pros. Accessed on 29/3/2018.

TensorFlow. mnist_deep.py, 2017b. URL https://github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/examples/tutorials/mnist/mnist_deep.py. Accessed on 29/3/2018.

TensorFlow. MNIST digits 0-9, 2018a. URL https://www.tensorflow.org/images/mnist_0-9.png. Accessed on 3/4/2018.

TensorFlow. tf.gradients, 2018b. URL https://www.tensorflow.org/api_docs/python/tf/gradients. Accessed on 26/3/2018.

TensorFlow. tf.losses.softmax_cross_entropy, 2018c. URL https://www.tensorflow.org/api_docs/python/tf/losses/softmax_cross_entropy. Accessed on 4/4/2018.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. 2016. URL http://arxiv.org/abs/1605.02688.

Tijmen Tieleman and Geoffrey E. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accessed on 4/4/2018.

Laurens van der Maaten and Geoffrey Hinton. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9: 2579-2605, 2008. URL http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using DropConnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 2013. PMLR. URL http://proceedings.mlr.press/v28/wan13.html.

Martin Wattenberg, Fernanda Vigas, and Ian Johnson. How to use t-SNE effectively. *Distill*, 2016. doi: 10.23915/distill.00002. URL http://distill.pub/2016/misread-tsne.

Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. DisturbLabel: Regularizing CNN on the loss layer. pages 4753–4762, 2016. doi: 10.1109/CVPR.2016.514. URL http://ieeexplore.ieee.org/document/7780883/.

Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alexander J. Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *2015 IEEE International Conference on Computer Vision (ICCV)*, volume 00, pages 1476–1483, 2015. doi: 10.1109/ICCV.2015.173. URL doi.ieeecomputersociety.org/10.1109/ICCV.2015.173.

Appendix A

Entropy Gradient Derivation

In this section we prove that

$$\frac{\partial H\left(p_\theta\left(y|x\right)\right)}{\partial z_i} = p_\theta\left(y_i|x\right)\left(-\log p_\theta\left(y_i|x\right) - H\left(p_\theta\right)\right).$$

As described in [Pereyra et al., 2017], $p_\theta\left(y_i|x\right)$ is obtained using softmax, which means

$$p_\theta\left(y_i|x\right) = \frac{\exp\left(z_i\right)}{\sum_{j=1}^{K}\exp\left(z_j\right)}.$$

To make our derivation more concise, let p_i denote $p_\theta\left(y_i|x\right)$. Now we can write:

$$\begin{aligned}
\frac{\partial H\left(p_\theta\left(y|x\right)\right)}{\partial z_i} &= -\sum_{j=1}^{K}\frac{\partial\left(p_j\log\left(p_j\right)\right)}{\partial z_i} \\
&= -\sum_{j=1}^{K}\left(\frac{\partial p_j}{\partial z_i}\log\left(p_j\right) + p_j\frac{1}{p_j}\frac{\partial p_j}{\partial z_i}\right) \\
&= -\sum_{j=1}^{K}\frac{\partial p_j}{\partial z_i} - \sum_{j=1}^{K}\frac{\partial p_j}{\partial z_i}\log\left(p_j\right) \\
&= -\left(p_i\left(1-p_i\right) - \sum_{\substack{j=1\\ j\neq i}}^{K} p_i p_j\right) - \left(p_i\left(1-p_i\right)\log\left(p_i\right) - \sum_{\substack{j=1\\ j\neq i}}^{K} p_i p_j\log\left(p_j\right)\right),
\end{aligned}$$

where in the last step we used

$$\frac{\partial p_j}{\partial z_i} = \begin{cases} p_i\left(1-p_i\right) & \text{if} \quad i = j \\ -p_j p_i & \text{if} \quad i \neq j, \end{cases}$$

which is proved in [Roelants, 2017].

Let us rewrite the expression further to get to the desired expression:

$$
\begin{aligned}
\frac{\partial H\left(p_\theta\left(y|x\right)\right)}{\partial z_i} &= -p_i\left(1-p_i\right) + \sum_{\substack{j=1 \\ j\neq i}}^{K} p_i p_j - p_i\left(1-p_i\right)\log\left(p_i\right) + \sum_{\substack{j=1 \\ j\neq i}}^{K} p_i p_j \log\left(p_j\right) \\
&= p_i\left(-\left(1-p_i\right) + \sum_{\substack{j=1 \\ j\neq i}}^{K} p_j - \left(1-p_i\right)\log\left(p_i\right) + \sum_{\substack{j=1 \\ j\neq i}}^{K} p_j \log\left(p_j\right)\right) \\
&= p_i\left(-1 + p_i + \sum_{\substack{j=1 \\ j\neq i}}^{K} p_j - \log\left(p_i\right) + p_i\log\left(p_i\right) + \sum_{\substack{j=1 \\ j\neq i}}^{K} p_j \log\left(p_j\right)\right) \\
&= p_i\left(-1 + \sum_{j=1}^{K} p_j - \log\left(p_i\right) + \sum_{j=1}^{K} p_j \log\left(p_j\right)\right) \\
&= p_i\left(-1 + 1 - \log\left(p_i\right) - H\left(p_\theta\left(y|x\right)\right)\right) \\
&= p_\theta\left(y_i|x\right)\left(-\log\left(p_\theta\left(y_i|x\right)\right) - H\left(p_\theta\left(y|x\right)\right)\right),
\end{aligned}
$$

which is exactly what we wanted to show.