Tunable Sensitivity to Large Errors in Neural Network Training

Gil Keren
Chair of Complex and Intelligent systems
University of Passau
Passau, Germany
gil.keren@uni-passau.de

Sivan Sabato
Department of Computer Science
Ben-Gurion University of the Negev
Beer Sheva, Israel

Björn Schuller
Chair of Complex and Intelligent systems
University of Passau
Passau, Germany,
Machine Learning Group
Imperial College London, U.K.

Abstract

When humans learn a new concept, they might ignore examples that they cannot make sense of at first, and only later focus on such examples, when they are more useful for learning. We propose incorporating this idea of tunable sensitivity for hard examples in neural network learning, using a new generalization of the cross-entropy gradient step, which can be used in place of the gradient in any gradient-based training method. The generalized gradient is parameterized by a value that controls the sensitivity of the training process to harder training examples. We tested our method on several benchmark datasets. We propose, and corroborate in our experiments, that the optimal level of sensitivity to hard example is positively correlated with the depth of the network. Moreover, the test prediction error obtained by our method is generally lower than that of the vanilla cross-entropy gradient learner. We therefore conclude that tunable sensitivity can be helpful for neural network learning.

1 Introduction

In recent years, neural networks have become empirically successful in a wide range of supervised learning applications, such as computer vision (Krizhevsky, Sutskever, and Hinton 2012; Szegedy et al. 2015), speech recognition (Hinton et al. 2012), natural language processing (Sutskever, Vinyals, and Le 2014) and computational paralinguistics (Keren and Schuller 2016; Keren et al. 2016). Standard implementations of training feed-forward neural networks for classification are based on gradient-based stochastic optimization, usually optimizing the empirical cross-entropy loss (Hinton 1989).

However, the cross-entropy is only a surrogate for the true objective of supervised network training, which is in most cases to reduce the probability of a prediction error (or in some case BLEU score, word-error-rate, etc). When optimizing using the cross-entropy loss, as we show below, the effect of training examples on the gradient is linear in the *prediction bias*, which is the difference between the network-predicted class probabilities and the target class probabilities. In particular, a wrong confident prediction induces a larger gradient than a similarly wrong, but less confident, prediction.

In contrast, humans sometimes employ a different approach to learning: when learning new concepts, they might ignore the examples they feel they do not understand, and focus more on the examples that are more useful to them. When improving proficiency regarding a familiar concept, they might focus on the harder examples, as these can contain more relevant information for the advanced learner. We make a first step towards incorporating this ability into neural network models, by proposing a learning algorithm with a tunable sensitivity to easy and hard training examples. Intuitions about human cognition have often inspired successful machine learning approaches (Bengio et al. 2009; Cho, Courville, and Bengio 2015; Lake et al. 2016). In this work we show that this can be the case also for tunable sensitivity.

Intuitively, the depth of the model should be positively correlated with the optimal sensitivity to hard examples. When the network is relatively shallow, its modeling capacity is limited. In this case, it might be better to reduce sensitivity to hard examples, since it is likely that these examples cannot be modeled correctly by the network, and so adjusting the model according to these examples might only degrade overall prediction accuracy. On the other hand, when the network is relatively deep, it has a high modeling capacity. In this case, it might be beneficial to allow more sensitivity to hard examples, thereby possibly improving the accuracy of the final learned model.

Our learning algorithm works by generalizing the cross-entropy gradient, where the new function can be used instead of the gradient in any gradient-based optimization method for neural networks. Many such training methods have been proposed, including, to name a few, Momentum (Polyak 1964), RMSProp (Tieleman and Hinton 2012), and Adam (Kingma and Ba 2015). The proposed generalization is parameterized by a value $k > 0$, that controls the sensitivity of the training process to hard examples, replacing the fixed dependence of the cross-entropy gradient. When $k = 1$ the proposed update rule is exactly the cross-entropy gradient. Smaller values of k decrease the sensitivity during training to hard examples, and larger values of k increase it.

We report experiments on several benchmark datasets. These experiments show, matching our expectations, that in almost all cases prediction error is improved using large values of k for deep networks, small values of k for shallow net-

works, and values close to the default $k = 1$ for networks of medium depth. They further show that using a tunable sensitivity parameter generally improves the results of learning.

The paper is structured as follows: In Section 1.1 related work is discussed. Section 2 presents our setting and notation. A framework for generalizing the loss gradient is developed in Section 3. Section 4 presents desired properties of the generalization, and our specific choice is given in Section 5. Experiment results are presented in Section 6, and we conclude in Section 7. Some of the analysis, and additional experimental results, are deferred to the supplementary material due to lack of space.

1.1 Related Work

The challenge of choosing the best optimization objective for neural network training is not a new one. In the past, the quadratic loss was typically used with gradient-based learning in neural networks (Rumelhart, Hinton, and Williams 1988), but a line of studies demonstrated both theoretically and empirically that the cross-entropy loss has preferable properties over the quadratic-loss, such as better learning speed (Levin and Fleisher 1988), better performance (Golik, Doetsch, and Ney 2013) and a more suitable shape of the error surface (Glorot and Bengio 2010). Other cost functions have also been considered. For instance, a novel cost function was proposed in (Silva et al. 2006), but it is not clearly advantageous to cross-entropy. The authors of (Bahdanau et al. 2015) address this question in a different setting of sequence prediction.

Our method allows controlling the sensitivity of the training process to examples with a large prediction bias. When this sensitivity is low, the method can be seen as a form of implicit outlier detection or noise reduction. Several previous works attempt to explicitly remove outliers or noise in neural network training. In one work (Smith and Martinez 2011), data is preprocessed to detect label noise induced from overlapping classes, and in another work (Jeatrakul, Wong, and Fung 2010) the authors use an auxiliary neural network to detect noisy examples. In contrast, our approach requires a minimal modification on gradient-based training algorithms for neural networks and allows emphasizing examples with a large prediction bias, instead of treating these as noise.

The interplay between "easy" and "hard" examples during neural network training has been addressed in the framework of Curriculum Learning (Bengio et al. 2009). In this framework it is suggested that training could be more successful if the network is first presented with easy examples, and harder examples are gradually added to the training process. In another work (Kumar, Packer, and Koller 2010), the authors define easy and hard examples based on the fit to the current model parameters. They propose a curriculum learning algorithm in which a tunable parameter controls the proportions of easy and hard examples presented to a learner at each phase. Our method is simpler than curriculum learning approaches, in that the examples can be presented at random order to the network. In addition, our method allows also a heightened sensitivity to harder examples. In a more recent work (Zaremba and Sutskever 2014), the authors indeed find that a curriculum in which harder examples are presented in early phases outperforms a curriculum that at first uses only easy examples.

2 Setting and Notation

For any integer n, denote $[n] = \{1, \ldots, n\}$. For a vector v, its i'th coordinate is denoted $v(i)$.

We consider a standard feed-forward multilayer neural network (Svozil, Kvasnicka, and Pospichal 1997), where the output layer is a softmax layer (Bridle 1990), with n units, each representing a class. Let Θ denote the neural network parameters, and let $z_j(x; \Theta)$ denote the value of output unit j when the network has parameters Θ, before the applying the softmax function. Applying the softmax function, the probability assigned by the network to class j is $p_j(x; \Theta) := e^{z_j} / \sum_{i=1}^{n} e^{z_i}$. The label predicted by the network for example x is $\hat{y}(x; \Theta) = \operatorname{argmax}_{j \in [n]} p_j(x; \Theta)$. We consider the task of supervised learning of Θ, using a labeled training sample $S = \{(x_i, y_i)\}_{i=1}^{m}$, , where $y_i \in [n]$, by optimizing the loss function: $L(\Theta) := \sum_{i=1}^{m} \ell((x_i, y_i); \Theta)$. A popular choice for ℓ is the cross-entropy cost function, defined by $\ell((x, y); \Theta) := -\log p_y(x; \Theta)$.

3 Generalizing the gradient

Our proposed method allows controlling the sensitivity of the training procedure to examples on which the network has large errors in prediction, by means of generalizing the gradient. A naïve alternative towards the same goal would be using an exponential version of the cross-entropy loss: $\ell = -|\log(p_y)^k|$, where p_y is the probability assigned to the correct class and k is a hyperparameter controlling the sensitivity level. However, the derivative of this function with respect to p_y is an undesired term since it is not monotone in k for a fixed p_y, resulting in lack of relevant meaning for small or large values of k. The gradient resulting from the above form is of a desired form only for $k = 1$, due to cancellation of terms from the derivatives of l and the softmax function. Another naïve option would be to consider $l = -\log(p_y^k)$, but this is only a scaled version of the cross-entropy loss and amounts to a change in the learning rate.

In general, controlling the loss function alone is not sufficient for controlling the relative importance to the training procedure of examples on which the network has large and small errors in prediction. Indeed, when computing the gradients, the derivative of the loss function is being multiplied by the derivative of the softmax function, and the latter is a term that also contains the probabilities assigned by the model to the different classes. Alternatively, controlling the parameters updates themselves, as we describe below, is a more direct way of achieving the desired effect.

Let (x, y) be a single labeled example in the training set, and consider the partial derivative of $\ell(\Theta; (x, y))$ with respect to some parameter θ in Θ. We have

$$\frac{\partial \ell((x, y); \Theta)}{\partial \theta} = \sum_{j=1}^{n} \frac{\partial \ell}{\partial z_j} \frac{\partial z_j}{\partial \theta},$$

where z_j is the input to the softmax layer when the input example is x, and the network parameters are Θ.

If ℓ is the cross-entropy loss, we have $\frac{\partial \ell}{\partial z_j} = \frac{\partial \ell}{\partial p_y}\frac{\partial p_y}{\partial z_j}$ and

$$\frac{\partial \ell}{\partial p_y} = -\frac{1}{p_y},$$
$$\frac{\partial p_y}{\partial z_j} = \begin{cases} p_y(1-p_y) & j = y, \\ -p_y p_j & j \neq y. \end{cases}$$

Hence

$$\frac{\partial \ell}{\partial z_j} = \begin{cases} p_j - 1 & y = j \\ p_j & \text{otherwise.} \end{cases}$$

For given x, y, Θ, define the *prediction bias* of the network for example x on class j, denoted by ϵ_j, as the (signed) difference between the probability assigned by the network to class j and the probability that should have been assigned, based on the true label of this example. We get $\epsilon_j = p_j - 1$ for $j = y$, and $\epsilon_j = p_j$ otherwise. Thus, for the cross-entropy loss,

$$\frac{\partial \ell}{\partial \theta} = \sum_{j=1}^{n} \frac{\partial z_j}{\partial \theta}\epsilon_j. \tag{1}$$

In other words, when using the cross entropy loss, the effect of any single training example on the gradient is linear in the prediction bias of the current network on this example.

As discussed in Section 1, it is likely that in many cases, the results of training could be improved if the effect of a single example on the gradient is not linear in the prediction bias. Therefore, we propose a generalization of the gradient that allows non-linear dependence in ϵ.

For given x, y, Θ and for $j \in \{1, \ldots, n\}$, define $f : [-1, 1]^n \to \mathbb{R}^n$, let $\epsilon = (\epsilon_1, \ldots, \epsilon_n)$, and consider the following generalization of $\frac{\partial \ell}{\partial \theta}$:

$$g(\theta) := \sum_{j=1}^{n} \frac{\partial z_j}{\partial \theta} f_j(\epsilon). \tag{2}$$

Here f_j is the j'th component of f. When f is the identity, we have $f_j(\epsilon) \equiv \frac{\partial \ell}{\partial z_j}$, and $g(\theta) = \frac{\partial \ell}{\partial \theta}$. However, we are now at liberty to study other assignments for f.

We call the vector of values of $g(\theta)$ for θ in Θ a *pseudo-gradient*, and propose to use g in place of the gradient within any gradient-based algorithm. In this way, optimization of the cross-entropy loss is replaced by a different algorithm of a similar form. However, as we show in Section 5.2, g is not necessarily the gradient of any loss function.

4 Properties of f

Consider what types of functions are reasonable to use for f instead of the identity. First, we expect f to be monotonic non-decreasing, so that a larger prediction bias never results in a smaller update. This is a reasonable requirement if we cannot identify outliers, that is, training examples that have a wrong label. We further expect f to be positive when $j \neq y$ and negative otherwise.

In addition to these natural properties, we introduce an additional property that we wish to enforce. To motivate this property, we consider the following simple example. Assume a network with one hidden layer and a softmax layer (see Figure 1), where the inputs to the softmax layer are $z_j = \langle w_j, h \rangle + b_j$ and the outputs of the hidden layer are $h(i) = \langle w'_i, x \rangle + b'_i$, where x is the input vector, and b'_i, w'_i are the scalar bias and weight vector between the input layer and the hidden layer. Suppose that at some point during training, hidden unit i is connected to all units j in the softmax layer with the same positive weight a. In other words, for all $j \in [n]$, $w_j(i) = a$. Now, suppose that the training process encounters a training example (x, y), and let l be some input coordinate.

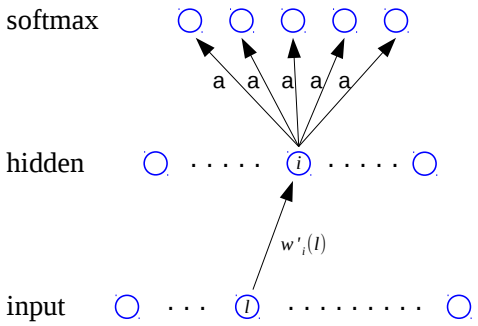

Figure 1: Illustrating the state of the network discussed in above.

What is the change to the weight $w'_i(l)$ that this training example should cause? Clearly it need not change if $x(l) = 0$, so we consider the case $x(l) \neq 0$. Only the value $h(i)$ is directly affected by changing $w'_i(l)$. From the definition of $p_j(x; \Theta)$, the predicted probabilities are fully determined by the ratios $e^{z_j}/e^{z_{j'}}$, or equivalently, by the differences $z_j - z_{j'}$, for all $j, j' \in [n]$. Now, $z_j - z_{j'} = \langle w_j, h \rangle + b_j - \langle w_{j'}, h \rangle + b_{j'}$. Therefore, $\frac{\partial(z_j - z_{j'})}{\partial h(i)} = w_j(i) - w_{j'}(i) = a - a = 0$, and therefore

$$\frac{\partial(z_j - z_{j'})}{\partial w'_i(l)} = \frac{\partial(z_j - z_{j'})}{\partial h(i)}\frac{\partial h(i)}{\partial w'_i(l)} = 0.$$

We conclude that in the case of equal weights from unit i to all output units, there is no reason to change the weight $w'_i(l)$ for any l. Moreover, preliminary experiments show that in these cases it is desirable to keep the weight stationary, as otherwise it can cause numerical instability due to explosion or decay of weights.

Therefore, we would like to guarantee this behavior also for our pseudo-gradients. Therefore, we require $g(w'_i(l)) = 0$ in this case. It follows that

$$0 = g(w'_i(l)) = \sum_{j=1}^{n} \frac{\partial z_j}{\partial w'_i(l)} f(\epsilon_j)$$
$$= \sum_{j=1}^{n} \frac{\partial z_j}{\partial h(i)}\frac{\partial h(i)}{\partial w'_i(l)} f(\epsilon_j) = \sum_{j=1}^{n} a \cdot x(l) \cdot f(\epsilon_j).$$

Dividing by $a \cdot x(l)$, we get the following desired property for the function f, for any vector ϵ of prediction biases:

$$f_y(\epsilon) = -\sum_{j \neq y} f_j(\epsilon). \quad (3)$$

Note that this indeed holds for the cross-entropy loss, since $\sum_{j \in [n]} \epsilon_j = 0$, and in the case of cross-entropy, f is the identity.

5 Our choice of f

In the case of the cross-entropy, f is the identity, leading to a linear dependence on ϵ. A natural generalization is to consider higher order polynomials. Combining this approach with the requirement in Eq. (3), we get the following assignment for f, where $k > 0$ is a parameter.

$$f_j(\epsilon) = \begin{cases} -|\epsilon_y|^k & j = y, \\ \frac{|\epsilon_y|^k}{\sum_{i \neq y} \epsilon_i^k} \cdot \epsilon_j^k & \text{otherwise.} \end{cases} \quad (4)$$

The expression $\frac{|\epsilon_y|^k}{\sum_{i \neq y} \epsilon_i^k}$ is a normalization term which makes sure Eq. (3) is satisfied. Setting $k = 1$, we get that $g(\theta)$ is the gradient of the cross-entropy loss. Other values of k result in different pseudo-gradients.

To illustrate the relationship between the value of k and the effect of prediction biases of different sizes on the pseudo-gradient, we plot $f_y(\epsilon)$ as a function of ϵ_y for several values of k (see Figure 2). Note that absolute values of the pseudo-gradient are of little importance, since in gradient-based algorithms, the gradient (or in our case, the pseudo-gradient) is usually multiplied by a scalar learning rate which can be tuned.

As the figure shows, when k is large, the pseudo-gradient is more strongly affected by large prediction biases, compared to small ones. This follows since $\frac{|\epsilon|^k}{|\epsilon'|^k}$ is monotonic increasing in k for $\epsilon > \epsilon'$. On the other hand, when using a small positive k we get that $\frac{|\epsilon|^k}{|\epsilon'|^k}$ tends to 1, therefore, the pseudo-gradient in this case would be much less sensitive to examples with large prediction biases. Thus, the choice of f, parameterized by k, allows tuning the sensitivity of the training process to large errors. We note that there could be other reasonable choices for f which have similar desirable properties. We leave the investigation of such other choices to future work.

5.1 A Toy Example

To further motivate our choice of f, we describe a very simple example of a distribution and a neural network. Consider a neural network with no hidden layers, and only one input unit connected to two softmax units. Denoting the input by x, the input to softmax unit i is $z_i = w_i x + b_i$, where w_i and b_i are the network weights and biases respectively.

It is not hard to see that the set of possible prediction functions $x \mapsto \hat{y}(x; \Theta)$ that can be represented by this network is exactly the set of threshold functions of the form $\hat{y}(x; \Theta) = \text{sign}(x - t)$ or $\hat{y}(x; \Theta) = -\text{sign}(x - t)$.

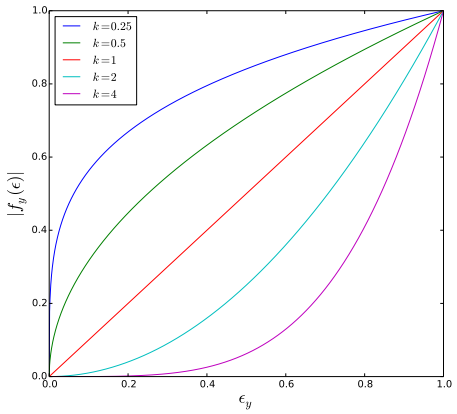

Figure 2: Size of $f_y(\epsilon)$ for different choices of k. Lines are in the same order as in the legend.

For convenience assume the labels mapped to the two softmax units are named $\{-1, +1\}$. Let $\alpha \in (\frac{1}{2}, 1)$, and suppose that labeled examples are drawn independently at random from the following distribution $\mathcal{D}$ over $\mathbb{R} \times \{-1, +1\}$: Examples are uniform in $[-1, 1]$; Labels of examples in $[0, \alpha]$ are deterministically 1, and they are -1 for all other examples. For this distribution, the prediction function with the smallest prediction error that can be represented by the network is $x \mapsto \text{sign}(x)$.

However, optimizing the cross-entropy loss on the distribution, or in the limit of a large training sample, would result in a different threshold, leading to a larger prediction error (for a detailed analysis see Appendix A in the supplementary material). Intuitively, this can be traced to the fact that the examples in $(\alpha, 1]$ cannot be classified correctly by this network when the threshold is close to 0, but they still affect the optimal threshold for the cross-entropy loss.

Thus, for this simple case, there is motivation to move away from optimizing the cross-entropy, to a different update rule that is less sensitive to large errors. This reduced sensitivity is achieved by our update rule with $k < 1$. On the other hand, larger values of k would result in higher sensitivity to large errors, thereby degrading the classification accuracy even more.

We thus expect that when training the network using our new update rule, the prediction error of the resulting network should be monotonically increasing with k, hence values of k which are smaller than 1 would give a smaller error. We tested this hypothesis by training this simple network on a synthetic dataset generated according to the distribution $\mathcal{D}$ described above, with $\alpha = 0.95$.

We generated 30,000 examples for each of the training, validation and test datasets. The biases were initialized to 0 and the weights were initialized from a uniform distribution on $(-0.1, 0.1)$. We used batch gradient descent with a learning rate of 0.01 for optimization of the four parameters, where the gradient is replaced with the pseudo-gradient

Table 1: Experiment results for single-layer networks

				Test Error		Test Cross-Entropy Loss	
Dataset	Layer Size	Momentum	Selected k	$k = 1$	Selected k	$k = 1$	Selected k
MNIST	400	0.5	0.5	1.76%	**1.74%**	**0.078**	0.167
MNIST	800	0.5	0.5	1.67%	**1.65%**	**0.072**	0.150
MNIST	1100	0.5	0.5	1.67%	**1.65%**	**0.071**	0.145
SVHN	400	0.5	0.25	16.88%	**16.16%**	**0.661**	1.576
SVHN	800	0.5	0.125	16.09%	**15.64%**	**0.648**	3.108
SVHN	1100	0.5	0.25	16.04%	**15.53%**	**0.626**	1.525
CIFAR-10	400	0.5	0.25	48.32%	**47.06%**	**1.430**	3.034
CIFAR-10	800	0.5	0.125	46.91%	**46.01%**	**1.388**	5.645
CIFAR-10	1100	0.5	0.25	46.43%	**45.84%**	**1.410**	2.820
CIFAR-100	400	0.5	0.25	75.18%	**74.41%**	**3.302**	6.931
CIFAR-100	800	0.5	0.25	74.04%	**73.78%**	**3.260**	7.449
CIFAR-100	1100	0.5	0.125	73.69%	**73.11%**	**3.239**	13.557

Table 2: Toy example experiment results

k	Test error	Threshold	CE Loss
4	8.36%	0.116	0.489
2	6.73%	0.085	0.361
1	4.90%	0.049	0.288
0.5	4.27%	0.037	0.299
0.25	4.04%	0.030	0.405
0.125	3.94%	0.028	0.625
0.0625	3.61%	0.022	1.190

from Eq. (2), using the function f defined in Eq. (4). f is parameterized by k, and we performed this experiment using values of k between 0.0625 and 4. After each epoch, we computed the prediction error on the validation set, and training was stopped after 3000 epochs in which this error was not changed by more than 0.001%. The values of the parameters at the end of training were used to compute the misclassification rate on the test set.

Table 2 reports the results for these experiments, averaged over 10 runs for each value of k. The results confirm our hypothesis regarding the behavior of the network for the different values of k, and further motivate the possible benefits of using $k \neq 1$. Note that while the prediction error is monotonic in k in this experiment, the cross-entropy is not, again demonstrating the fact that optimizing the cross-entropy is not optimal in this case.

5.2 Non-existence of a Cost Function for f

It is natural to ask whether, with our choice of f in Eq. (4), $g(\theta)$ is the gradient of another cost function, instead of the cross-entropy. The following lemma demonstrates that this is not the case.

Lemma 1. *Assume f as in Eq. (4) with $k \neq 1$, and $g(\Theta)$ the resulting pseudo-gradient. There exists a neural network for which the $g(\Theta)$ is not a gradient of any cost function.*

The proof of is lemma is left for the supplemental material. Note that the above lemma does not exclude the possibility that a gradient-based algorithm that uses g instead of the gradient still somehow optimizes some cost function.

6 Experiments

For our experiments, we used four classification benchmark datasets from the field of computer vision: The MNIST dataset (LeCun et al. 1998), the Street View House Numbers dataset (SVHN) (Netzer et al. 2011) and the CIFAR-10 and CIFAR-100 datasets (Krizhevsky and Hinton 2009). A more detailed description of the datasets can be found in Appendix C.1 in the supplementary material.

The neural networks we experimented with are feed-forward neural networks that contain one, three or five hidden layers of various layer sizes. For optimization, we used stochastic gradient descent with momentum (Sutskever et al. 2013) with several values of momentum and a minibatch size of 128 examples. For each value of k, we replaced the gradient in the algorithm with the pseudo-gradient from Eq. (2), using the function f defined in Eq. (4). For the multi-layer experiments we also used Gradient-Clipping (Pascanu, Mikolov, and Bengio 2013) with a threshold of 100. In the hidden layers, biases were initialized to 0 and for the weights we used the initialization scheme from (Glorot and Bengio 2010). Both biases and weights in the softmax layer were initialized to 0.

In each experiment, we used cross-validation to select the best value of k. The learning rate was optimized using cross-validation for each value of k separately, as the size of the pseudo-gradient can be significantly different between different values of k, as evident from Eq. (4). We compared the test error between the models using the selected k and $k = 1$, each with its best performing learning rate. Additional details about the experiment process can be found in Appendix C.2 in the supplementary material.

We report the test error of each of the trained models for MNIST, SVHN, CIFAR-10 and CIFAR-100 in Tables 1, 3 and 4 for networks with one, three and five layers respectively. Additional experiments are reported in Appendix C.1 in the supplementary material. We further report the cross-entropy values using the selected k and the default $k = 1$.

Table 3: Experiment results for 3-layer networks

Dataset	Layer Sizes	Mom'	Selected k	Test Error		Test CE Loss	
				$k=1$	Selected k	$k=1$	Selected k
MNIST	400	0.5	1	—	—	—	—
MNIST	800	0.5	1	—	—	—	—
SVHN	400	0.5	2	16.52%	16.52%	1.604	**0.968**
SVHN	800	0.5	1	—	—	—	—
CIFAR-10	400	0.5	2	46.81%	**46.63%**	3.023	**2.121**
CIFAR-10	800	0.5	1	—	—	—	—
CIFAR-100	400	0.5	0.5	75.20%	**74.95%**	**3.378**	4.511
CIFAR-100	800	0.5	1	—	—	—	—

Table 4: Experiment results for 5-layer networks

Dataset	Layer Sizes	Mom'	Selected k	Test Error		Test CE Loss	
				$k=1$	Selected k	$k=1$	Selected k
MNIST	400	0.5	0.5	1.71%	**1.69%**	**0.113**	0.224
MNIST	800	0.5	0.25	1.61%	**1.60%**	**0.118**	0.390
SVHN	400	0.5	4	17.41%	**16.49%**	1.436	**0.708**
SVHN	800	0.5	0.5	17.07%	**16.61%**	**1.343**	2.604
CIFAR-10	400	0.5	2	48.05%	**47.85%**	2.017	**1.962**
CIFAR-10	800	0.5	4	**44.21%**	44.24%	4.610	**1.677**
CIFAR-100	400	0.5	2	75.69%	**75.48%**	3.611	**3.228**
CIFAR-100	800	0.5	2	74.10%	**73.57%**	4.650	**4.439**
MNIST	400	0.9	1	—	—	—	—
MNIST	800	0.9	4	**1.58%**	1.60	0.098	**0.060**
SVHN	400	0.9	4	17.89%	**16.54%**	1.284	**0.718**
SVHN	800	0.9	2	16.24%	**15.73%**	1.647	**0.998**
CIFAR-10	400	0.9	4	47.91%	**47.57%**	2.202	**1.648**
CIFAR-10	800	0.9	2	45.69%	**44.11%**	3.316	**2.171**
CIFAR-100	400	0.9	1	—	—	—	—
CIFAR-100	800	0.9	4	**74.32%**	74.62%	3.872	**3.432**

Several observations are evident from the experiment results. First, aligned with our hypothesis, the value of k selected by the cross-validation scheme was almost always smaller than 1, for the shallow networks, larger than one for the deep networks, and close to one for networks with medium depth. Indeed, the capacity of network is positively correlated with the optimal sensitivity to hard examples.

Second, for the shallow networks the cross-entropy loss on the test set was always worse for the selected k than for $k = 1$. This implies that indeed, by using a different value of k we are not optimizing the cross-entropy loss, yet are improving the success of optimizing the true prediction error. On the contrary, in the experiments with three and five layers, the cross entropy is also improved by selecting the larger k. This is an interesting phenomenon, which might be explained by the fact that examples with a large prediction bias have a high cross-entropy loss, and so focusing training on these examples reduces the empirical cross-entropy loss, and therefore also the true cross-entropy loss.

To summarize, our experiments show that overall, cross-validating over the value of k usually yields improved results over $k = 1$, and that, as expected, the optimal value of k grows with the depth of the network.

7 Conclusions

Inspired by an intuition in human cognition, in this work we proposed a generalization of the cross-entropy gradient step in which a tunable parameter controls the sensitivity of the training process to hard examples. Our experiments show that, as we expected, the optimal level of sensitivity to hard examples is positively correlated with the depth of the network. Moreover, the experiments demonstrate that selecting the value of the sensitivity parameter using cross validation leads overall to improved prediction error performance on a variety of benchmark datasets.

The proposed approach is not limited to feed-forward neural networks — it can be used in any gradient-based training algorithm, and for any network architecture. In future work, we plan to study this method as a tool for improving training in other architectures, such as convolutional networks and recurrent neural networks, as well as experimenting with different levels of sensitivity to hard examples in different stages of the training procedure, and combining the predictions of models with different levels of this sensitivity.

Acknowledgments

This work has been supported by the European Communitys Seventh Framework Programme through the ERC Starting Grant No. 338164 (iHEARu). Sivan Sabato was supported in part by the Israel Science Foundation (grant No. 555/15).

References

Bahdanau, D.; Serdyuk, D.; Brakel, P.; Ke, N. R.; Chorowski, J.; Courville, A.; and Bengio, Y. 2015. Task loss estimation for sequence prediction. *arXiv preprint arXiv:1511.06456*.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proc. of the 26th annual International Conference on Machine Learning (ICML)*, 41–48. Montreal, Canada: ACM.

Bridle, J. S. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*. Springer. 227–236.

Cho, K.; Courville, A.; and Bengio, Y. 2015. Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia* 17(11):1875–1886.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proc. of International Conference on Artificial Intelligence and Statistics*, 249–256.

Golik, P.; Doetsch, P.; and Ney, H. 2013. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Proc. of INTERSPEECH*, 1756–1760.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-r.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE* 29(6):82–97.

Hinton, G. E. 1989. Connectionist learning procedures. *Artificial intelligence* 40(1):185–234.

Jeatrakul, P.; Wong, K. W.; and Fung, C. C. 2010. Data cleaning for classification using misclassification analysis. *Journal of Advanced Computational Intelligence and Intelligent Informatics* 14(3):297–302.

Keren, G., and Schuller, B. 2016. Convolutional RNN: an enhanced model for extracting features from sequential data. In *Proc. of 2016 International Joint Conference on Neural Networks (IJCNN)*, 3412–3419.

Keren, G.; Deng, J.; Pohjalainen, J.; and Schuller, B. 2016. Convolutional neural networks with data augmentation for classifying speakers native language. In *Proc. of INTERSPEECH*, 2393–2397.

Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Proc. of Advances in Neural Information Processing Systems (NIPS)*, 1097–1105.

Kumar, M. P.; Packer, B.; and Koller, D. 2010. Self-paced learning for latent variable models. In *Proc. of Advances in Neural Information Processing Systems (NIPS)*, 1189–1197.

Lake, B. M.; Ullman, T. D.; Tenenbaum, J. B.; and Gershman, S. J. 2016. Building machines that learn and think like people. *arXiv preprint arXiv:1604.00289*.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Levin, E., and Fleisher, M. 1988. Accelerated learning in layered neural networks. *Complex systems* 2:625–640.

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*. Granada, Spain.

Pascanu, R.; Mikolov, T.; and Bengio, Y. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 1310–1318.

Polyak, B. T. 1964. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics* 4(5):1–17.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5:3.

Silva, L. M.; De Sa, J. M.; Alexandre, L.; et al. 2006. New developments of the Z-EDM algorithm. In *Intelligent Systems Design and Applications*, volume 1, 1067–1072.

Smith, M. R., and Martinez, T. 2011. Improving classification accuracy by identifying and removing instances that should be misclassified. In *The 2011 International Joint Conference on Neural Networks (IJCNN)*, 2690–2697.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *Proc. of the 30th International Conference on Machine Learning (ICML)*, 1139–1147.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Svozil, D.; Kvasnicka, V.; and Pospichal, J. 1997. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems* 39(1):43–62.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tieleman, T., and Hinton, G. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*.

Zaremba, W., and Sutskever, I. 2014. Learning to execute. *arXiv preprint arXiv:1410.4615*.

Tunable Sensitivity to Large Errors in Neural Network Training

Supplementary material

A Proof for Toy Example

Consider the neural network from the toy example in Section 5.1. In this network, there exists one classification threshold such that examples above or below it are classified to different classes. We prove that for a large enough training set, the value of the cross-entropy cost is not minimal when the threshold is at 0.

Suppose that there is an assignment of network parameters that minimizes the cross-entropy which induces a threshold at 0. The output of the softmax layer is determined uniquely by $\frac{e^{z_0}}{e^{z_1}}$, or equivalently by $z_0 - z_1 = x(w_0 - w_1) + b_0 - b_1$. Therefore, we can assume without loss of generality that $w_1 = b_1 = 0$. Denote $w := w_0, b := b_0$. If $w = 0$ in the minimizing assignment, then all examples are classified as members of the same class and in particular, the classification threshold is not zero. Therefore we may assume $w \neq 0$. In this case, the classification threshold is $\frac{-b}{w}$. Since we assume a minimal solution at zero, the minimizing assignment must have $b = 0$.

When the training set size approaches infinity, the cross-entropy on the sample approaches the expected cross-entropy on $\mathcal{D}$. Let $\mathrm{CE}(w, b)$ be the expected cross-entropy on $\mathcal{D}$ for network parameter values w, b. Then

$$\mathrm{CE}(w,b) = -\frac{1}{2}\Big(\int_{-1}^{0}\log(p_0(x))dx + \int_{0}^{\alpha}\log(p_1(x))dx + \int_{\alpha}^{1}\log(p_0(x))dx\Big).$$

And we have:

$$\log(p_0(x)) = \log\frac{e^{wx+b}}{e^{wx+b}+1} = wx + b - \log(e^{wx+b}+1),$$
$$\log(p_1(x)) = \log\frac{1}{e^{wx+b}+1} = -\log(e^{wx+b}+1).$$

Therefore

$$\begin{aligned}\frac{\partial \mathrm{CE}(w,b)}{\partial b} &= \\ -\frac{1}{2}\frac{\partial}{\partial b}\Big(&\int_{-1}^{0}(wx+b)dx - \int_{-1}^{0}\log(e^{wx+b}+1)dx \\ &- \int_{0}^{\alpha}\log(e^{wx+b}+1)dx \\ &+ \int_{\alpha}^{1}(wx+b)dx - \int_{\alpha}^{1}\log(e^{wx+b}+1)dx\Big) \\ = -\frac{1}{2}(1 - &\frac{\partial}{\partial b}\left(\int_{-1}^{1}\log(e^{wx+b}+1)dx\right) + 1 - \alpha).\end{aligned}$$

Differentiating under the integral sign, we get

$$\frac{\partial \mathrm{CE}(w,b)}{\partial b} = -\frac{1}{2}\left(2 - \alpha - \int_{-1}^{1}\frac{e^{wx+b}}{e^{wx+b}+1}\right)$$

Since we assume the cross-entropy has a minimal solution with $b = 0$, we have

$$\begin{aligned}0 &= -2\frac{\partial \mathrm{CE}(w, b=0)}{\partial b} \\ &= 2 - \alpha - \frac{1}{w}(\log(e^w+1) - \log(e^{-w}+1)).\end{aligned}$$

Therefore

$$w(2-\alpha) = \log\frac{e^w+1}{e^{-w}+1} = \log(e^w) = w.$$

Since $\alpha \neq 1$, it must be that $w = 0$. This contradicts our assumption, hence the cross-entropy does not have a minimal solution with a threshold at 0.

B Proof of Lemma 1

Proof. Consider a neural network with three units in the output layer, and at least one hidden layer. Let (x, y) be a labeled example, and suppose that there exists some cost function $\bar{\ell}((x, y); \Theta)$, differentiable in Θ, such that for g as defined in Eq. (2) and f defined in Eq. (4) for some $k > 0$, we have $g(\theta) = \frac{\partial \bar{\ell}}{\partial \theta}$ for each parameter θ in Θ. We now show that this is only possible if $k = 1$.

Under the assumption on $\bar{\ell}$, for any two parameters θ_1, θ_2,

$$\frac{\partial}{\partial \theta_2}\left(\frac{\partial \bar{\ell}}{\partial \theta_1}\right) = \frac{\partial^2 \bar{\ell}}{\partial \theta_1 \theta_2} = \frac{\partial}{\partial \theta_1}\left(\frac{\partial \bar{\ell}}{\partial \theta_2}\right),$$

hence

$$\frac{\partial g(\theta_1)}{\partial \theta_2} = \frac{\partial g(\theta_2)}{\partial \theta_1}. \tag{5}$$

Recall our notations: $h(i)$ is the output of unit i in the last hidden layer before the softmax layer, $w_j(i)$ is the weight between the hidden unit i in the last hidden layer, and unit j in the softmax layer, z_j is the input to unit j in the softmax layer, and b_j is the bias of unit j in the softmax layer.

Let (x, y) such that $y = 1$. From Eq. (5) and Eq. (2), we have

$$\frac{\partial}{\partial w_2(1)}\sum_{j=1}^{n}\frac{\partial z_j}{\partial w_1(1)}f_j(\epsilon) = \frac{\partial}{\partial w_1(1)}\sum_{j=1}^{n}\frac{\partial z_j}{\partial w_2(1)}f_j(\epsilon).$$

Plugging in f as defined in Eq. (4), and using the fact that $\frac{\partial z_j}{\partial w_i(1)} = 0$ for $i \neq j$, we get:

$$\begin{aligned}-\frac{\partial}{\partial w_2(1)}\left(\frac{\partial z_1}{\partial w_1(1)}\cdot|\epsilon_1|^k\right) = \\ \frac{\partial}{\partial w_1(1)}\left(\frac{\partial z_2}{\partial w_2(1)}\cdot\frac{\epsilon_2^k}{\epsilon_2^k+\epsilon_3^k}\cdot|\epsilon_1|^k\right).\end{aligned}$$

Since $y = 1$, we have $\epsilon = (p_1 - 1, p_2, p_3)$. In addition, $\frac{\partial z_j}{\partial w_j(1)} = h(1)$ and $\frac{\partial h(1)}{\partial w_j(1)} = 0$ for $j \in [2]$. Therefore

$$\begin{aligned}-\frac{\partial}{\partial w_2(1)}\left((1-p_1)^k\right) = \\ \frac{\partial}{\partial w_1(1)}\left(\frac{p_2^k}{p_2^k+p_3^k}(1-p_1)^k\right).\end{aligned} \tag{6}$$

Next, we evaluate each side of the equation separately, using the following:

$$\frac{\partial p_j}{\partial w_j(1)} = \frac{\partial p_j}{\partial z_j}\frac{\partial z_j}{\partial w_j(1)} = h(1)p_j(1-p_j),$$
$$\forall j \neq i, \quad \frac{\partial p_j}{\partial w_i(1)} = \frac{\partial p_j}{\partial z_i}\frac{\partial z_i}{\partial w_i(1)} = -h(1)p_i p_j.$$

For the LHS of Eq. (6), we have

$$-\frac{\partial}{\partial w_2(1)}(1-p_1)^k = -k(1-p_1)^{k-1}h(1)p_1p_2.$$

For the RHS,

$$\frac{\partial}{\partial w_1(1)}\frac{p_2^k}{p_2^k+p_3^k}(1-p_1)^k = -\frac{kh(1)p_1p_2^k(1-p_1)^k}{p_2^k+p_3^k}.$$

Hence Eq. (6) holds if and only if:

$$1 = \frac{p_2^{k-1}(1-p_1)}{p_2^k+p_3^k}.$$

For $k = 1$, this equality holds since $p_1 + p_2 + p_3 = 1$. However, for any $k \neq 1$, there are values of p_1, p_2, p_3 such that this does not hold. We conclude that our choice of f does not lead to a pseudo-gradient g which is the gradient of any cost function. □

C Additional Experiment details and Results

C.1 datasets

The MNIST dataset (LeCun et al. 1998), consisting of grayscale 28x28 pixel images of handwritten digits, with 10 classes, 60,000 training examples and 10,000 test examples, the Street View House Numbers dataset (SVHN) (Netzer et al. 2011), consisting of RGB 32x32 pixel images of digits cropped from house numbers, with 10 classes 73,257 training examples and 26,032 test examples and the CIFAR-10 and CIFAR-100 datasets (Krizhevsky and Hinton 2009), consisting of RGB 32x32 pixel images of 10/100 object classes, with 50,000 training examples and 10,000 test examples. All datasets were linearly transformed such that all features are in the interval $[-1, 1]$.

C.2 Choosing the value of k

In each experiment, we used cross-validation to select the best value of k. For networks with one hidden layer, k was selected out of the values $\{4, 2, 1, 0.5, 0.25, 0.125, 0.0625\}$. For networks with 3 or 5 hidden layers, k was selected out of the values $\{4, 2, 1, 0.5, 0.25\}$, removing the smaller values of k due to performance considerations (in preliminary experiments, these small values yielded poor results for deep networks). The learning rate was optimized using cross-validation for each value of k separately, as the size of the pseudo-gradient can be significantly different between different values of k, as evident from Eq. (4).

For each experiment configuration, defined by a dataset, network architecture and momentum, we selected an initial learning rate η, based on preliminary experiments on the training set. Then the following procedure was carried out for $\eta/2, \eta, 2\eta$, for every tested value of k:

1. Randomly split the training set into 5 equal parts, $S_1, \ldots, S_5$.
2. Run the iterative training procedure on $S_1 \cup S_2 \cup S_3$, until there is no improvement in test prediction error for 15 epochs on the early stopping set, S_4.
3. Select the network model that did the best on S_4.
4. Calculate the validation error of the selected model on the validation set, S_5.
5. Repeat the process for t times after permuting the roles of $S_1, \ldots, S_5$. We set $t = 10$ for MNIST, and $t = 7$ for CIFAR-10/100 and SVHN.
6. Let $\mathrm{err}_{k,\eta}$ be the average of the t validation errors.

We then found $\mathrm{argmin}_\eta \mathrm{err}_{k,\eta}$. If the minimum was found with the minimal or the maximal η that we tried, we also performed the above process using half the η or double the η, respectively. This continued iteratively until there was no need to add learning rates. At the end of this process we selected $(k^*, \eta^*) = \mathrm{argmin}_{k,\eta} \mathrm{err}_{k,\eta}$, and retrained the network with parameters k^*, η^* on the training sample, using one fifth of the sample as an early stopping set. We compared the test error of the resulting model to the test error of a model retrained in the same way, except that we set $k = 1$ (leading to standard cross-entropy training), and the learning rate to $\eta_1^* = \mathrm{argmin}_\eta \mathrm{err}_{1,\eta}$. The final learning rates in the selected models were in the range $[10^{-1}, 10]$ for MNIST, and $[10^{-4}, 1]$ for the other datasets.

C.3 Results

Additional experiment results with momentum values other than 0.5 are reported in Table 5, Table 6.

Table 5: Experiment results for single-layer networks

Dataset	Layer Size	Momentum	Selected k	Test Error		Test Cross-Entropy Loss	
				$k = 1$	Selected k	$k = 1$	Selected k
MNIST	400	0	0.5	1.71%	**1.70%**	**0.0757**	0.148
MNIST	800	0	0.5	**1.66%**	1.67%	**0.070**	0.137
MNIST	1100	0	0.5	1.64%	**1.62%**	**0.068**	0.131
MNIST	400	0.9	0.5	1.75%	1.75%	**0.073**	0.140
MNIST	800	0.9	2	1.71%	**1.63%**	0.070	**0.054**
MNIST	1100	0.9	0.5	1.74%	**1.69%**	**0.069**	0.127
SVHN	400	0	0.25	16.84%	**16.09%**	**0.658**	1.575
SVHN	800	0	0.25	16.19%	**15.71%**	**0.641**	1.534
SVHN	1100	0	0.25	15.97%	**15.68%**	**0.636**	1.493
SVHN	400	0.9	0.125	16.65%	**16.30%**	**0.679**	2.861
SVHN	800	0.9	0.25	16.15%	**15.68%**	**0.675**	1.632
SVHN	1100	0.9	0.25	15.85%	**15.47%**	**0.640**	1.657
CIFAR-10	400	0	0.125	48.15%	**46.91%**	**1.435**	5.609
CIFAR-10	800	0	0.125	46.92%	**46.14%**	**1.390**	5.390
CIFAR-10	1100	0	0.125	46.63%	**46.00%**	**1.356**	5.290
CIFAR-10	400	0.9	0.0625	48.19%	**46.71%**	**1.518**	11.049
CIFAR-10	800	0.9	0.125	47.09%	**46.16%**	**1.616**	5.294
CIFAR-10	1100	0.9	0.125	46.71%	**45.77%**	**1.850**	5.904
CIFAR-100	400	0.9	0.25	74.96%	**74.28%**	**3.306**	7.348
CIFAR-100	800	0.9	0.125	74.12%	**73.47%**	**3.327**	13.267
CIFAR-100	1100	0.9	0.25	73.47%	**73.19%**	**3.235**	7.489

Table 6: Experiment results for 3-layer networks

Dataset	Layer Sizes	Mom'	Selected k	Test Error		Test CE Loss	
				$k = 1$	Selected k	$k = 1$	Selected k
MNIST	400	0	1	—	—	—	—
MNIST	800	0	1	—	—	—	—
MNIST	400	0.9	1	—	—	—	—
MNIST	800	0.9	0.5	1.60%	**1.53%**	**0.091**	0.189
SVHN	400	0.9	1	—	—	—	—
SVHN	800	0.9	2	16.14%	**15.96%**	1.651	**1.062**
CIFAR-10	400	0.9	2	47.52%	**46.92%**	2.226	**2.010**
CIFAR-10	800	0.9	2	45.27%	**44.26%**	2.855	**2.341**
CIFAR-100	400	0.9	0.25	74.97%	**74.52%**	**3.356**	8.520
CIFAR-100	800	0.9	0.5	74.48%	**73.17%**	**4.133**	8.642