Convolutional Neural Network and Convex Optimization

Si Chen and Yufei Wang
Department of Electrical and Computer Engineering
University of California San Diego
{sic046, yuw176}@ucsd.edu

Abstract

1 Introduction

Deep learning is a new area of machine learning research, which is recently be of interests of more and more researchers and organizations. Convolutional neural networks (CNNs) are a special kind of deep learning method, and it is particularly useful in field of computer vision. CNNs have achieved state of the art performance on many challenging computer vision contests, which brings CNNs a lot of attention.

Despite deep learning's great success on performance, there are always criticisms and concerns about this method. One of them are that it is not a convex problem. However, for convex problem, the models are usually too restricted to be powerful. Non-convex models are much harder to train, but almost none of the state of the art performance is achieved by purely convex optimization. So there is always a trade of between "easy-to-use" and "powerful".

In this project, we try to answer this question: can we make use of convex optimization techniques to improve the highly nonconvex deep neural network? We decomposite the CNNs to several locally convex components and improve the their performance using convex optimization methods from two perspective: modifying the last two layers of the network by making a linear combination of many sub-models; and replacing the original loss function by other loss functions from the convex optimization community.

2 Convolutional Neural Networks

In this section, we give a brief introduction of convolutional neural networks, which is the foundation of our project.

2.1 Theoretical basis: Convolutional neural networks

Convolutional neural networks(CNN) are a special kind of deep neural networks. It exploits local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. For a certain hidden layer m, the hidden units in it are connected to a local subset of units in the $(m-1)$th layer. Additionally, each sparse filter h_i is replicated across the entire visual field. The replicated units share the same parametrization, i.e. the same weight vector and same bias. The layer is called feature map.

Mathematically, a feature map h^k is obtained by convolving the input with a linear filter, adding a bias term and then applying a non-linear function, it can be shown as follow:

$$h^k_{ij} = \mathrm{f}((W^k * x)_{ij} + b_k) \tag{1}$$

where W^k and b_k are weight and bias of kth feature map, and f$(\cdot)$ is the nolinearity. In our experiments, Rectified Linear Units(ReLU) nonlinearity is used, which has been shown to be more efficient than conventional function tanh$(\cdot)$.[1] ReLU nonlinearity is as follow:

$$f(x) = \max(0, x) \tag{2}$$

Another important type of layers is pooling. It is a form of non-linear down-sampling. There are several types of pooling, two common types of which are max-pooling and average-pooling. They partition the input image into a set of non-overlapping or overlapping rectangles and outputs the maximum/average value for each such sub-region. By pooling, the model can reduce the computational complexity for upper layers, and can provide a form of translation invariance.

Typically, the last layer of a CNN is a logistic regression layer. Each unit of the output reflects a class membership probability:

$$P(Y = i|x, W, b) = softmax_i(Wx + b) = \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}} \tag{3}$$

The parameters of the network are trained using back propagation[2]. The loss function used for training is the negative-log likelihood of the training dataset D under the model:

$$L = \sum_{i=0}^{|D|} \log(P(Y = y^{(i)}|x^{(i)}, W, b)) \tag{4}$$

which can also be viewed as cross-entropy.

Finally, the prediction of the model is done by taking the argmax of the vector of $P(Y = i|x, W, b)$:

$$y_{pred} = \text{argmax}_i P(Y = i|x, W, b) \tag{5}$$

2.2 Overall architecture

The overall architecture of our CNN is shown in Figure 1. The architecture is built for CIFAR-10 dataset[1]. There are three convolutional layers and pooling layers alternatively. Overlapping pooling is performed. Each pooling layer consists of a grid of pooling units spaced $s = 2$ pixels apart, each summarizing a neighborhood of size 3×3 centered at the location of the pooling unit. The output of the network is a vector of class membership probability with 10 units, corresponding to 10 classes in our CIFAR-10 dataset.

3 Sub-model Convolutional Network

In this section, we start from introducing an effective technique called dropout. Inspired by dropout, we then introduce the concept of sub-model, and explore an improvement of network by linearly combining output of sub-models. We use convex optimization to find optimal weights of the combination.

3.1 Dropout and sub-model combination

Dropout is a recently-introduced technique to reduce overfitting [1]. It consists of setting to zero the output of each hidden neuron with probability of 0.5 in training procedure. So every time a training example is presented, the neural network samples a different architecture, which we denote here as a sub-model m_k. All these sub-models share weights. At test time, we use all neurons but multiply their outputs by 0.5, which is an approximation to taking the geometric mean of the predictive

[1] http://www.cs.toronto.edu/ kriz/cifar.html

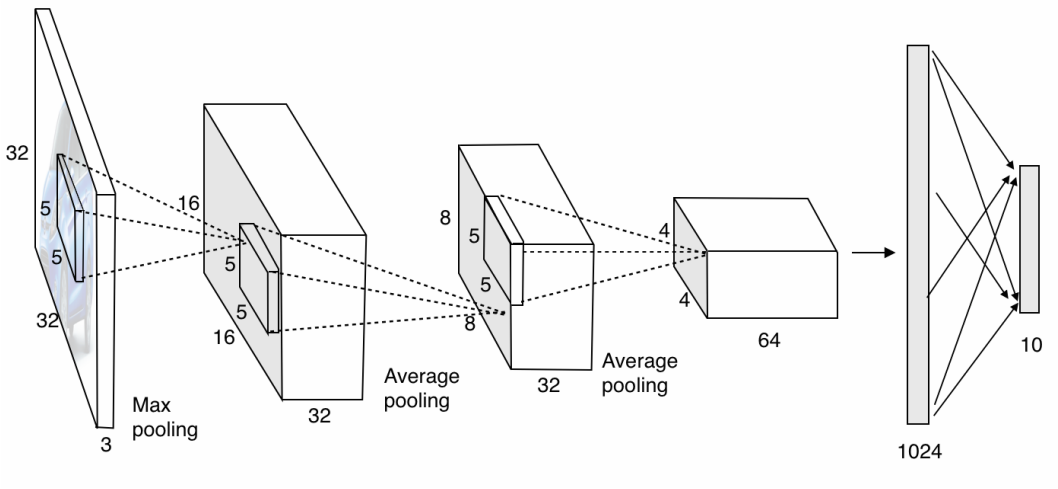


Figure 1: The architecture of our CNN.

distributions $P_{m_k} = [P_{m_k}(Y = 1), P_{m_k}(Y = 2), ..., P_{m_k}(Y = d)]^t$ produced by the sub-models (where d is the number of classes):

$$\begin{aligned}\lim_{n\to\infty}(P_{m_1}(Y=i)\cdot P_{m_2}(Y=i)\cdot\cdots\cdot P_{m_n}(Y=i))^{\frac{1}{n}} &= \lim_{n\to\infty}\left(\frac{e^{(b_i+W_i\frac{1}{n}\sum_{k=1}^{n}h_{m_k})}}{(\sum_{j=1}^{d}e^{(nb_j+W_j\sum_{k=1}^{n}h_{m_k})})^{\frac{1}{n}}}\right)\\ &= \frac{e^{(b_i+W_i\cdot h_{comb})}}{\lim_{n\to\infty}(\sum_{j=1}^{d}e^{(nb_j+W_j\sum_{k=1}^{n}h_{m_k})})^{\frac{1}{n}}}\end{aligned} \tag{6}$$

$$P_{comb}(Y=i) = \frac{e^{b_i+W_i\cdot h_{comb}}}{\sum_{j=1}^{d}e^{b_j+W_j\cdot h_{comb}}} \tag{7}$$

where h_{m_k} is the neurons of the penultimate layer of sub-model m_k, and h_{comb} is that of the combined model by using all the neurons but multiply them by 0.5. $P_{comb}(\cdot)$ is the class membership probabilities of the combined model.

Apparently, the predictive distribution of the combined model is only a biased approximation of taking the mean of the predictive distribution of all sub-models.

In our experiments (architecture shown in Figure 1), dropout is performed in the output of penultimate layer, and $d = 10$ is the number of classes of our dataset.

Here, we want to explore a better option of combining the sub-models. Rather than giving an approximation, we want to actually take the linear combination of the predictive distribution of n sub-models:

$$P_{l.comb}(Y=i) = \sum_{k=1}^{n} l_k \times P_{m_k}(Y=i) \tag{8}$$

where l_k is the weight of the distribution of kth model. The best weight $l = [l_1, l_2, ..., l_n]^t$ can be obtained by solving following optimization problem:

$$\begin{array}{ll}\min\limits_{l} & \sum_{i=1}^{N}\|P_i\cdot l - y_i\|_2^2\\ \text{s.t.} & l \;\succeq\; 0\end{array} \tag{9}$$

where N is the number of all the training examples, y_i is the 10×1 binary column vector indicating the true label of ith data point, and $P_i = [P_{m_1}, P_{m_2}, ..., P_{m_n}]$ is a $10\times n$ matrix, each column indicating the predictive probabilities of each sub-model. By minimizing the sum of squared l2 norm of difference between predicted distribution and true label, we find the non-negative weight vector l.

The objective function can be simplified as following:

$$\begin{aligned}\sum_{i=1}^{N} \|P_i \cdot l - y_i\|_2^2 &= \sum_{i=1}^{N} (P_i \cdot l - y_i)^t (P_i \cdot l - y_i) \\ &= (P \cdot l - y)^t (P \cdot l - y) \\ &= l^t P^t P l - 2 y^t P l + y^t y\end{aligned} \tag{10}$$

where $y = \left[y_1^t, y_2^t, ...y_N^t\right]^t$ is the concatenation of label indicator vectors of all data points, which is a $10N \times 1$ vector. $P = \left[P_1^t, P_2^t, ..., P_n^t\right]^t$ is the concatenation of probability distributions of all data points, which is a $10N \times n$ matrix. Then the optimization problem can be written as follow:

$$\begin{aligned}\min_{l} \quad & l^t P^t P l - 2 y^t P l + y^t y \\ \text{s.t.} \quad & l \geq 0\end{aligned} \tag{11}$$

This is a Quadratic Optimization Problem (QP), apparently a convex optimization problem, and is easy to solve.

3.2 Sub-model combination in non-dropout CNN

Now let's move back to CNN without dropout training. The same idea of sub-model combination can also be utilized to improve the performance of CNN when the network is trained without dropout.

The sub-models can be obtained by similar fashion with the dropout sub-models: given the already trained CNN model without dropout, randomly set penultimate units to zero with probability of 50%, and multiply the remaining units by 2:

$$\Pr(h_{m_i}^j = 2h_{orig}^j) = \Pr(h_{m_i}^j = 0) = \frac{1}{2}. \tag{12}$$

where h_{m_i} is the 1024 penultimate-layer unit vector of the sub-model m_i, h_{orig} is the 1024-unit vector of the trained CNN model, and $h_{(\cdot)}^j$ is the jth element of the vector.

Then, we can obtain the predictive distribution of each sub-model P_{m_k}, and the new model takes linear combination of the predictive distribution of n sub-models, which is the same as the dropout condition:

$$P_{l.comb}(Y = i) = \sum_{k=1}^{n} l_k \times P_{m_k}(Y = i) \tag{13}$$

Therefore, the optimization problem is very similar with dropout condition (Equation 11), the only difference being the multiplication by 2 when obtaining the penultimate layer of sub-models.

For both dropout network and non-dropout network, we expect improvement in performance on test set using a weighted combination of sub-models, when the number of sub-models used is large enough.

4 Convex Loss functions

Section 3 uses convex optimization for finding optimal combination of sub-models in CNN, while this section seeks to utilize convex optimization technique in deep learning from another perspective: focusing on alternatives of loss functions.

Deep neural network is well-known for its high non-convexity. However, we can still borrow some ideas from the convex optimization literature, where people have found many useful convex loss functions for classification tasks. In this section, we will talk about three different convex loss functions, and discuss how to use them in multiclass classification problem within the deep learning architecture.

4.1 Cross-entropy

Cross-entropy (CE) is the most standard loss function for neural networks, which is defined by:

$$H(p,q) = -\sum_{x} p(x) log q(x), \tag{14}$$

where p and q are ground-truth posterior probability and estimated posterior probability respectively. Cross-entropy can be directly used for multiclass classification. This has been discussed in Section 2.1.

4.2 Hinge Loss

Hinge loss is widely used in Support Vector Machines (SVMs). Given training data and its corresponding labels (x_n, y_n), $n = 1, ..., N$, $x_n \in R^D$, $t_n \in -1, +1$, for a binary classification problem, SVMs learning includes the following convex optimization:

$$\begin{array}{lll} \min\limits_{w,\psi} & \frac{1}{2} w^T w + C \sum_{n=1}^{N} \psi_n & \\ \text{s.t.} & w^T x_n t_n & \geq \quad 1 - \psi_n \forall n \\ & \psi_n & \geq 0 \quad \forall n \end{array} \tag{15}$$

Hinge loss, while doesn't have a clear probability interpretation as CE, can be regarded as a maximum margin objective function, and it can be directly used in the deep learning architecture by deriving the corresponding unconstraint optimization problem:

$$\min_{w} \quad \frac{1}{2} w^T w + C \sum_{n=1}^{N} \mathbf{max}(1 - w^T x_n t_n, 0) \tag{16}$$

The second term of the object function is the Hinge loss. For the first term, there is a specific term called Weight Decay in the deep learning literature (which can also be seen as L2 regularization). The derivative w.r.t x is:

$$\frac{\partial l(x)}{\partial x_n} = -C t_n (\mathbb{I}(1 > w^T x_n t_n)), \tag{17}$$

which is fast to compute.

Another similar loss function is the squared hinge loss:

$$\sum_{n=1}^{N} \mathbf{max}(1 - w^T x_n t_n, 0)^2, \tag{18}$$

which is also differentiable and easy to compute. It imposes a bigger loss for points which violate the margin.

To predict the class label, for a binary classification problem, it's simple:

$$\begin{cases} +1, if \quad w^T x > 0 \\ -1, \quad otherwise \end{cases} \tag{19}$$

4.3 Multiclass Deep Neural Network with Hinge Loss

Many methods have been proposed to generalize SVM to deal with multiclass problem. Here we talk about two most popular ways: One-Versus-All method and One-Versus-One method.

4.3.1 One-Versus-All

This is the earliest method for SVM multiclass classification. The basic idea is to train K SVM models where K is the number of classes. The ith SVM uses the data with label i as the positive samples and the data with other labels as the negative samples. Interestingly, this idea can be easily implemented in deep learning architecture. We can have K output units on top of the penultimate layer with the hinge loss. Then each output unit corresponds to one of these One-Versus-All SVMs. We can also regard this as replacing the softmax and cross-entropy layer with the hinge loss layer. This kind of method works well for problem with a small number of classes. When the class number grows larger, the imbalance between positive and negative samples for each SVM becomes intolerable.

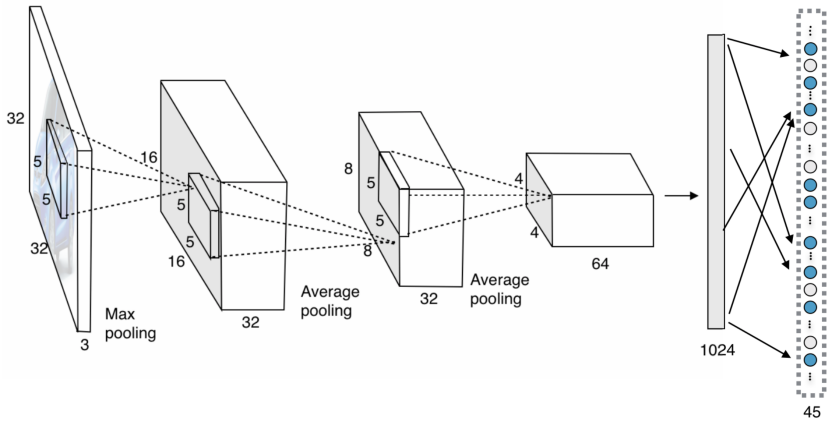


Figure 2: Deep Neural Network with One-Versus-One Hinge Loss. Ultimate layer consists of $9 \times 10/2 = 45$ units. For each training data, only 9 units (colored blue) are activated.

4.3.2 One-Versus-One

The One-Versus-One method is to construct $K(K-1)/2$ classifiers. Each is responsible for one of the $K(K-1)/2$ class pairs. While testing, we simply let all the binary classifiers vote for the most probable one, which is also called the "Max Wins" strategy. To incorporate this method to deep learning architecture, we can have $K(K-1)/2$ output units, each of which corresponds to one classifier. All of the units use hinge loss. For each training data, only $K-1$ of these units are activated. The other units are temperorily cut off. This is illustrated in Fig. 2

The good property here is that for each SVM, the positive and negative training samples are balanced, no matter how many classes there are. However, the number of SVMs needed are quadratic in K, which makes it not scalable.

4.4 Ranking Loss

Another way to directly model the classification problem is the pairwise-ranking loss. Since we just want to pick the one with the highest linear output at test time. This leads to the following minimization problem:

$$\text{Loss} = \sum_{n=1}^{N} \sum_{i \in C_n^*} \mathbf{max}(0, 1 - f_j(x_n) + f_i(x_n)), \tag{20}$$

where j is the label of the nth data and C_n^* indicates the set of all the other labels. This loss can be directly generalized to multilabel predition problem and is widely used in many recommandation systems.

5 Experiments

For all the experiments, the training and test data are CIFAR-10 dataset. There are 50,000 images in training set and 10,000 images in test set.

5.1 Sub-model convolutional network

In this section, we use Theano[3] to train and test the CNN model, and cvx toolbox[2] in matlab to solve the convex optimization problem finding best weights.

[2]http://cvxr.com

Network	Accuracy on test set (%)
Dropout	81.540
Non-dropout	79.517

Table 1: Baseline: Accuracies of dropout/non-dropout networks

5.1.1 Sub-model combination results

We separately train the two networks with and without dropout, and the the results on the test set are shown in Table 1, this is the baseline of our experiments in this section. For both networks, the learning rate is initialized to 0.001. Before reaching the final point, we decrease the learning rate twice(for each time we reduce it by a factor of 10). Accuracy of dropout network is about 2% higher than nondropout network, which shows the advantage of dropout method.

Then, we randomly extract 4800 sub-models from dropout/non-dropout network respectively, and use different number of sub-models to obtain the combined model. For each combined model, we find the optimal weight l by solving the convex optimization problem. The plot of the accuracy on test set versus number of sub-models is shown in Figure 3. The dotted line in both sub-figures are the accuracy of the original model, which is the approximation of geometric mean of all possible sub-models. We use this accuracy as a baseline.

For both networks, our sub-model CNNs outperform the original networks when the number of sub-models is large enough. Clearly, the accuracy tends to increase with the number of sub-models used. This is intuitive: the sub-models are chosen randomly; therefore the more sub-models we use, the more probable we have useful sub-models and better combinations.

There is one concern though: As is shown in Figure 3(a), for the network trained by dropout, the improvement is limited: approximately 0.2% improvement is achieved. On the other hand, for network trained without dropout, the improvement by weighted combination of sub-models is much more significant: the accuracy of original model is 79.52%, and the weighed combination of sub-models is always better than the original model, even when there are only 100 sub-models. The improvement is up to ???? when the number of sub-models reaches to ???.

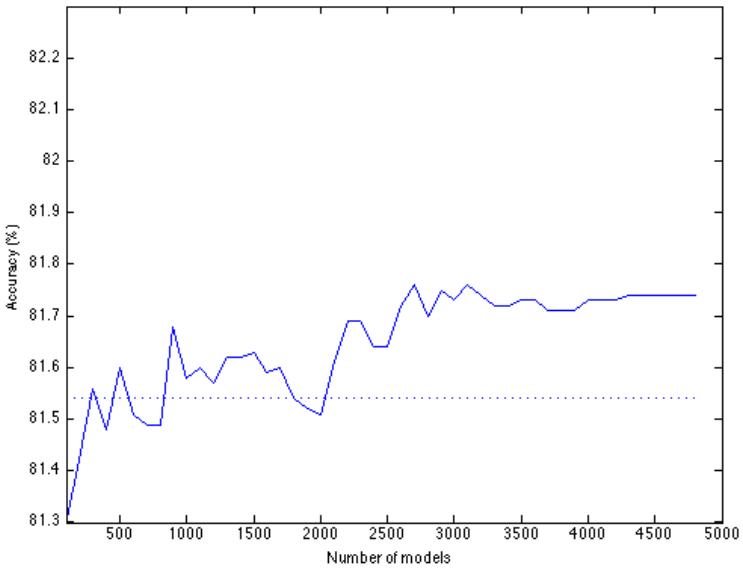
5.2 The role of dropout

The reason of the lower improvement for dropout network is explained in this section. Dropout trains different sub-model at a time, but the weights are shared by all the sub-models. This procedure forces the network to learn more robust features that are useful for many different random sub-models, and after training, the sub-models in the dropout network can give similar results for all the data. This is the eventual goal of the dropout training. In other words, the sub-models in the dropout network are similar to each other, therefore whatever combination of them would not achieve much improvement. On the other hand, the network trained without dropout doesn't have the invariant property, thus the sub-models are quite different to each other, leading to greater improvement by combining them.

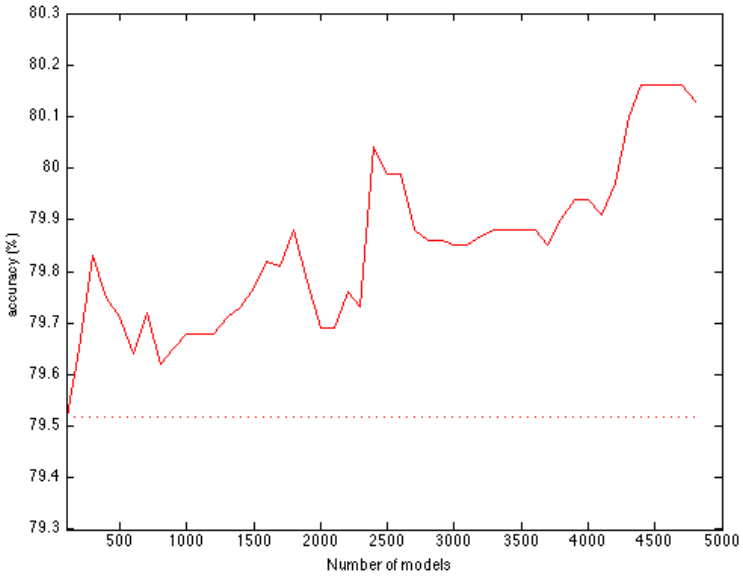
Figure 4 illustrates this invariance effect of dropout training more clearly. Accuracy achieved by each of 4800 sub-models for non-dropout/ dropout network is shown in the two histograms. Clearly, the sub-models of dropout network are much more concentrated than those of the non-dropout network. This shows the larger invariance of the sub-models in non-dropout network.

5.3 Convolutional network with different convex loss functions

For this part, we use Caffe to do the experiments. Caffe is the most popular and fastest deep convolutional network implementation for the time being. It provides the cross-entropy loss. For the other loss functions, we implemented by ourself. We are going to compare cross-entropy(CE), One-Versus-All hinge loss(1vsAll-L1), One-Versus-All square hinge loss(1vsALL-L2), One-Versus-One hinge loss(1vs1-L1), rank loss(RANK).



(a) Accuracy-number of sub-models for dropout network

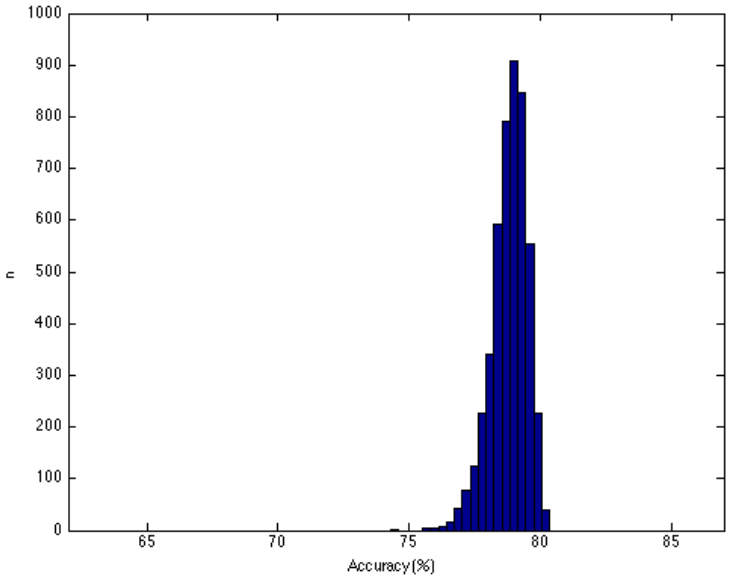


(b) Accuracy v.s. number of sub-models for nondropout network

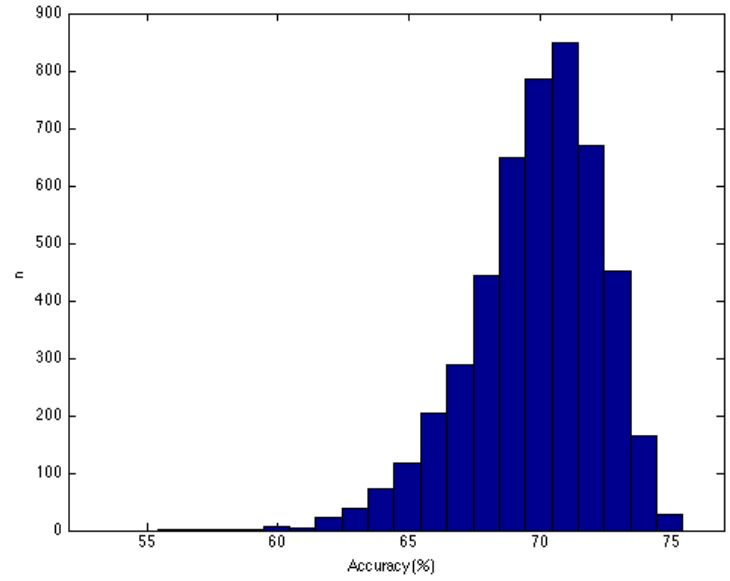
Figure 3: Accuracy v.s. number of sub-models.

Loss	Accuracy on test set (%)	WeightDecay
CE	77.85	750
1vsALL-L1	80.25	10
1vsALL-L2	80.18	100
1vs1-L1	77.80	10
RANK	77.60	100

Table 2: Accuracies with different loss functions and the corresponding weight decay



(a) Histogram of accuracies of sub-models of dropout network



(b) Histogram of accuracies of sub-models of nondropout network

Figure 4: Histograms of sub-models.

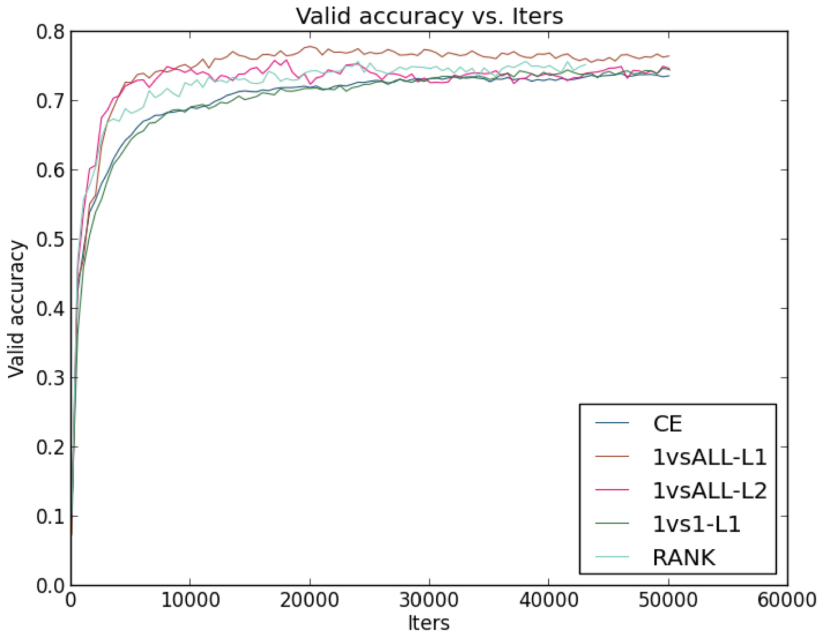


Figure 5: Comparison for different loss functions while training

5.3.1 Classification results

Table 2 shows the results for different loss functions and the corresponding relative weight decay used for the best results. For all of them, we use a similar strategy to train the network. The learning rate is initialized to 0.001. Before reaching the final point, we decrease the learning rate twice(for each time we reduce it by a factor of 10). The results clearly show that One-Versus-All hinge loss and square hinge loss outperform other methods by a large margin. Due to time limit, all the results shown above is the best from the parameters that we have tried.

5.3.2 Convergence comparison

We also plot the accuracy rate on the validation set for the first 50000 iterations while training(see Fig. 5). We can find that under the same learning rate(0.001), the One-Versus-All hinge loss and square hinge loss converge the fastest. Cross-entropy gives the smoothest curve.

Please note that even though in this dataset One-Versus-All hinge loss seems to outperform cross-entropy by a large margin, it doesn't mean that we can use them as the standard loss in the future. By some preliminary experiments, the One-Versus-All hinge loss doesn't work on CIFAR100, which is a 100 class extention of CIFAR10. Cross-entropy is standard because of its stability. Also, for One-Versus-One hinge loss and rank loss, we can not say for sure that they perform worse than the others since the sets of parameters we have tried on them are too limited. This is why many people don't like neural networks. Tuning the parameters is a nightmare. However, we do find that by borrowing ideas from the convex optimization literature, we indeed see some improvements on some certain tasks.

6 Conclusion

References

[1] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. (2012)

[2] Y. LeCun, L. Bottou, G.O., Muller, K.: Efficient backprop. Neural Networks: Tricks of the trade (1998)

[3] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy). (2010)

[4] Heinrich, G.: Parameter estimation for text analysis. Technical report (2004)