CIS 680 Homework #1

Oleh Rybkin
University of Pennsylvania

Introduction

The task of the homework was to experiment with basics of neural networks. I have chosen tensorflow as basis for implementation of all tasks.

Task 1

In this task a single neuron with sigmoid activation was considered. I implemented the neuron in tensorflow, saving the work to compute gradients by myself. The relevant figures are Fig. 1 and Fig. 2.

It is clear from the graphs that cross-entropy (CE) loss better matches the sigmoid activation function than L_2 loss.

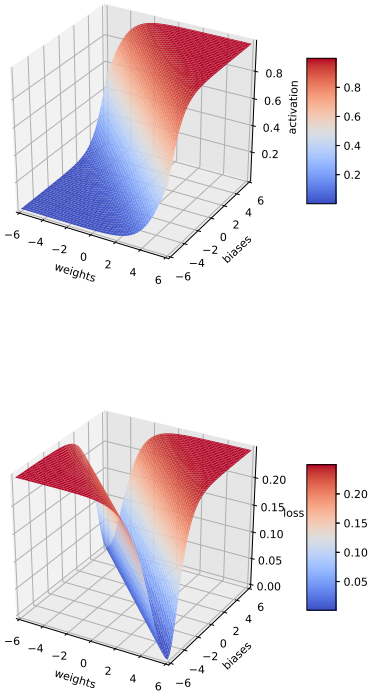

Figure 1: Sigmoid activation. Top: the activation itself, bottom: L_2 loss.

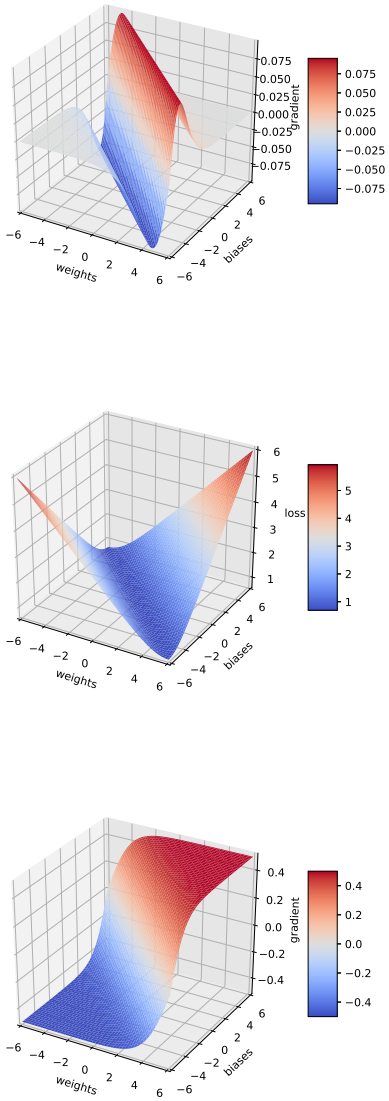

Figure 2: Sigmoid activation. Top: gradient of the L_2 loss with respect to the weight. Middle: cross-entropy loss. Bottom: gradient of the cross-entropy loss with respect to the weight.

The CE loss itself increases rapidly with error, whereas the L_2 loss stops increasing after some moment. Consecutively, the gradient of CE loss with respect to weights remains big as long as the error is present, whereas the gradient of L_2 loss vanishes after some error value. This would prevent the neuron from learning if the error is too big (e.g. bias 5 and weight 5 in our example), as the gradient updates would be diminishingly small. The neuron would be effectively stuck at a bad value of weight vector and would need a prohibitively big amount of time to reach even a local minimum (e.g. bias 0 and weight 0 in our example).

Task 2

In this task I implemented a simple FC neural network using the low-level tensorflow API and "Optimizer" class. The relevant figures are Fig. 3 and 4 and Tab. 1.

With all architectures neural network rapidly memorizes the data due to big number of parameters $n = 73$ and no regularization.

Network settings	Epochs (rounded up)
ReLU + CE	800
ReLU + L_2	2000
Sigmoid + L_2	2000
Sigmoid + CE	4000

Table 1: Networks with different choice of activation and loss function, sorted by the time it takes to train the network.

Difference between various activation and loss functions can be explained by the form that the gradient of the weight and the bias takes in each particular case. The gradient should remain big as long as the loss is also big, so as to allow effective training. This is not the case with sigmoid activation, where gradient tends to zero whenever the output

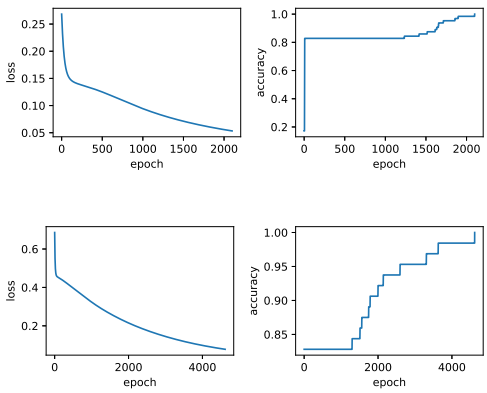

Figure 3: FC network with sigmoid activation performance. Top: using L_2 loss. Bottom: using cross-entropy loss.

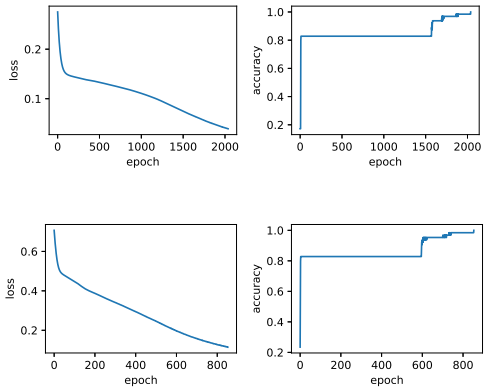

Figure 4: FC network with ReLU activation performance. Top: using L_2 loss. Bottom: using cross-entropy loss.

of the neuron is big, no matter what the loss is. ReLU, in contrast, does not suffer from this problem, which, indeed, makes it outperform sigmoid activation in terms of shorter training time. In ReLU case, the cross-entropy loss outperforms L_2 loss, which is also consistent with this argument (see also discussion in Task 1).

Task 3

In this task I implemented a simple CNN. The relevant figure is Fig. 5.

As the precise form of the network was not specified by

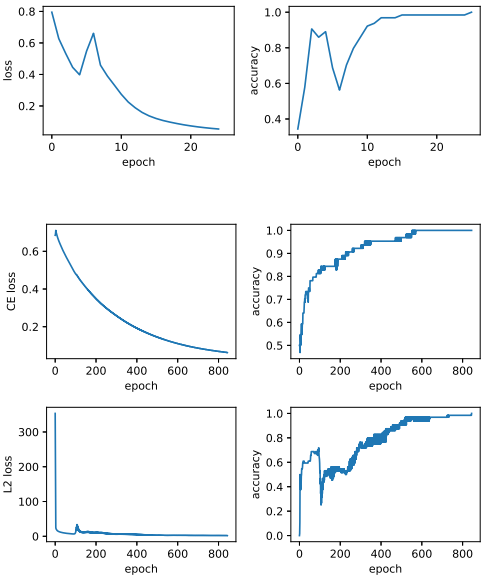

Figure 5: CNN training with standard ReLU. Top: trained on one task. Middle and bottom: trained on two tasks, middle: classification accuracy, bottom: regression accuracy.

the task, I have chosen to use no stride in the convolutional layers. Spatial dimensions shrinking is insured by using no padding.

The classification-only network learns much quicker than the previous one while still achieving absolute performance. This can be attributed to several different causes. Firstly, the size of images is now 16 times bigger, which makes it even easier to memorize the training data. Secondly, the number of parameters of the networks has increased to a few thousand.

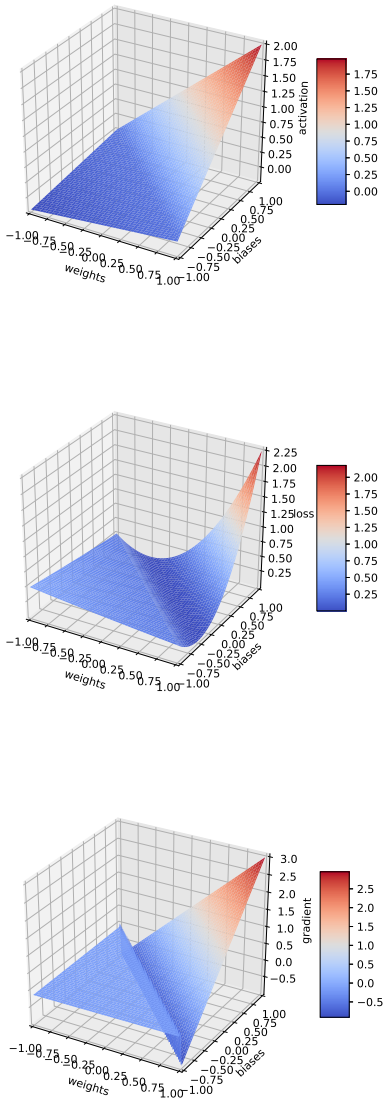

Figure 6: Leaky ReLU activation. Top: the activation itself. Middle: L_2 loss. Bottom: gradient of L_2 loss with respect to the weight.

The detection network, however, takes much more time to train as the simultaneous regression and classification task is more challenging. Conditioned by multi-task learning, the network is forced to learn more meaningful patterns. Similarly to findings of [1], the network experiences a boost in accuracy and loss compliance at the beginning, where it probably learns the easiest and most generalizable patterns. After that it continues to learn slowly until every piece of data is memorized.

One problem encountered during training the network was exploding gradients. When feeding sum of losses in a batch instead of their mean, the network tuned to the sum of losses experienced big perturbations in gradient descent. The learning rate was effectively 64 times bigger than expected. This caused network weights to diverge to huge (1e+30) values and the network became numerically unstable, occasionally producing NaN values at the output. A similar problem occurred when the variance of initial weights distribution was set to 1 instead of 0.1. In both cases, however, in some trials the network was able to train itself successfully in a small number of steps.

Task 4

In this section I evaluate the leaky ReLU introduced in [2] as an alternative activation function to standard ReLU. The relevant figures are Fig. 6 and 7. The leaky ReLU activation is defined as follows:

$$\text{LeakyReLU}(x) = \max(0.1x, x).$$

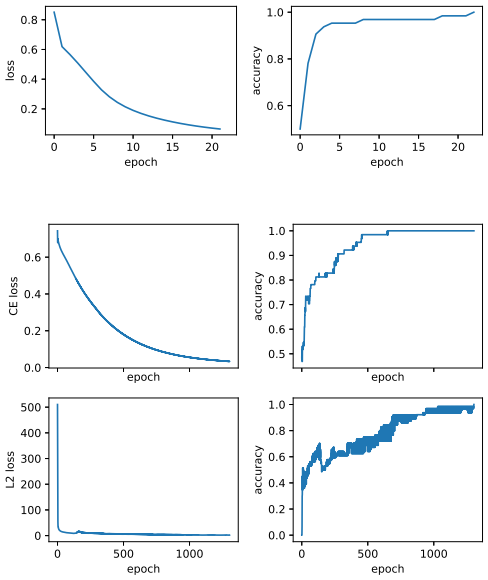

Figure 7: CNN training with leaky ReLU. Top: trained on one task. Middle and bottom: trained on two tasks, middle: classification accuracy, botton: regression accuracy.

With the standard ReLU activation there might occur a situation when a neuron has negative response for most input data and, after the ReLU, zero activation. Such a neuron would not learn as its gradient is always zero. The leaky ReLU has an advantage over standard ReLU in that it allows for such "dead" neuron to learn by allowing some small activation even with negative neuron output.

The leaky ReLU performs more or less the same as standard ReLU in this case. The differences can be as well explained by random perturbations caused by different random weight initialization. The lack of significant change might be due to the fact that the network already has more parameters that needed for the problem and dying neurons are not a problem. Moreover, it was shown elsewhere that the sparse representation that standard ReLU tends to produce is beneficial for the accuracy of the network, which might explain the slightly longer training time with leaky ReLU.

Closing remarks

I learned basics of the tensorflow package and got a hands-on experience with neural networks and issues such as exploding or vanishing gradient problem.

I would like to thank Nikolaos Kolotouros with whom I discussed parts of the task and the style of tensorflow coding.

References

[1] D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien. A Closer Look at Memorization in Deep Networks. *ArXiv e-prints*, June 2017.

[2] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.