EECS 598 Homework 1

Chenchen Ma

February 14, 2019

1 Neural network layer implementation

1.1 Fully-connected layer

- Note that

  $$\frac{\partial Y_k}{\partial W_{ij}} = \mathbb{1}\{k=j\}X_i,$$

  and

  $$\frac{\partial L}{\partial W_{ij}} = \sum_k \frac{\partial L}{\partial Y_k}\frac{\partial Y_k}{\partial W_{ij}} = \sum_k \frac{\partial L}{\partial Y_k}\mathbb{1}\{k=j\}X_i = \frac{\partial L}{\partial Y_j}X_i,$$

  we have

  $$\frac{\partial L}{\partial W} = X\left(\frac{\partial L}{\partial Y}\right)^T$$

  .

- Note that

  $$\frac{\partial Y_k}{\partial b_i} = \mathbb{1}\{k=i\},$$

  and

  $$\frac{\partial L}{\partial b_i} = \sum_k \frac{\partial L}{\partial Y_k}\frac{\partial Y_k}{\partial b_i} = \frac{\partial L}{\partial Y_i},$$

  therefore we have

  $$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial Y}.$$

- Note that

  $$Y_k = W_k^T X + b_i,$$

  where W_k is the kth column of W, we have

  $$\frac{\partial Y_k}{\partial X_i} = W_{ik}.$$

  Thus we have

  $$\frac{\partial L}{\partial X_i} = \sum_k \frac{\partial L}{\partial Y_k}\frac{\partial Y_k}{\partial X_i} = \sum_k \frac{\partial L}{\partial Y_k}W_{ik},$$

  and

  $$\frac{\partial L}{\partial X} = W\frac{\partial L}{\partial Y}.$$

1.2 ReLU

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \odot \mathbb{1}\{X>0\}.$$

1.3 Dropout

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \odot M.$$

1.4 Batch Normalization

- Since $\frac{\partial \mu}{\partial X_{ij}} = \mathbb{1}\{k = j\}\frac{1}{n}$, we have $\frac{\partial \mu}{\partial X_i} = \frac{1}{n}$.
- Note that

$$\begin{aligned}\frac{\partial \sigma_k}{\partial X_{ik}} &= \frac{1}{2}\frac{1}{\sigma_k}\frac{2}{n}\sum_j (X_{jk} - \mu_k)\frac{\partial (X_{jk} - \mu_k)}{\partial X_{ik}} \\ &= \frac{1}{n\sigma_k}\sum_j (X_{jk} - \mu_k)(\mathbb{1}\{i = j\} - \frac{1}{n}) \\ &= \frac{1}{n\sigma_k}(X_{ik} - \mu_k),\end{aligned}$$

  and $\frac{\partial \sigma_k}{\partial X_{ij}} = 0$ if $k \neq j$. Therefore, we have

$$\frac{\partial \sigma}{\partial X_i} = \frac{1}{n\sigma}(X_i - \mu).$$

- Note that

$$\frac{\partial Y_i}{\partial X_i} = \frac{\gamma}{\sigma} + \frac{\partial Y_i}{\partial \mu}\frac{\partial \mu}{\partial X_i} + \frac{\partial Y_i}{\partial \sigma}\frac{\partial \sigma}{\partial X_i},$$

  and

$$\frac{\partial Y_i}{\partial X_j} = \frac{\partial Y_i}{\partial \mu}\frac{\partial \mu}{\partial X_j} + \frac{\partial Y_i}{\partial \sigma}\frac{\partial \sigma}{\partial X_j}, \ i \neq j,$$

  we have

$$\begin{aligned}\frac{\partial L}{\partial X_i} &= \sum_j \frac{\partial L}{\partial Y_j}\frac{\partial Y_j}{\partial X_i} \\ &= \frac{\gamma}{\sigma}\frac{\partial L}{\partial Y_i} + \sum_j \frac{\partial L}{\partial Y_j}\frac{\partial Y_j}{\partial \mu}\frac{\partial \mu}{\partial X_i} + \sum_j \frac{\partial L}{\partial Y_j}\frac{\partial Y_j}{\partial \sigma}\frac{\partial \sigma}{\partial X_i} \\ &= \frac{\gamma}{\sigma}\frac{\partial L}{\partial Y_i} - \frac{\gamma}{n\sigma}\sum_j \frac{\partial L}{\partial Y_j} - \frac{\gamma}{\sigma^2}\sum_j \frac{\partial L}{\partial Y_j}(X_j - \mu)\frac{\partial \sigma}{\partial X_i} \\ &= \frac{\gamma}{\sigma}\big[\frac{\partial L}{\partial Y_i} - \frac{1}{n}\sum_j \frac{\partial L}{\partial Y_j} - \frac{1}{n\sigma^2}(X_i - \mu)\sum_j \frac{\partial L}{\partial Y_j}(X_j - \mu)\big].\end{aligned}$$

1.5 Convolution

- Show $\frac{\partial L}{\partial X_{n,c}} = \sum_f W_{f,c} *_{\text{full}} \left(\frac{\partial L}{\partial Y_{n,f}}\right)$:

  By the chain rule, we know that

$$\frac{\partial L}{\partial X_{n,c,h,w}} = \sum_{f,h',w'} \frac{\partial L}{\partial Y_{n,f,h',w'}}\frac{\partial Y_{n,f,h',w'}}{\partial X_{n,c,h,w}}.$$

  Therefore, we need to calculate $\frac{\partial Y_{n,f,h',w'}}{\partial X_{n,c,h,w}}$.

  Note that

$$Y_{n,f,h',w'} = \sum_{c'} \left(X_{n,c'} *_{\text{valid}} \overline{W_{f,c'}}\ \right)_{h',w'},$$

$$\begin{aligned}\left(X_{n,c} *_{\text{valid}} \overline{W_{f,c}}\right)_{h',w'} &= \sum_{h,w} X_{n,c,h,w} (\overline{W_{f,c}})_{h'-h+H'',w'-w+W''} \\ &= \sum_{h,w} X_{n,c,h,w} W_{f,c,h-h'+1,w-w'+1}.\end{aligned}$$

Thus we have

$$\frac{\partial Y_{n,f,h',w'}}{\partial X_{n,c,h,w}} = W_{f,c,h-h'+1,w-w'+1},$$

and

$$\begin{aligned}\frac{\partial L}{\partial X_{n,c,h,w}} &= \sum_{f,h',w'} \frac{\partial L}{\partial Y_{n,f,h',w'}} \frac{\partial Y_{n,f,h',w'}}{\partial X_{n,c,h,w}} \\ &= \sum_{f,h',w'} \frac{\partial L}{\partial Y_{n,f,h',w'}} W_{f,c,h-h'+1,w-w'+1} \\ &= \sum_{f,h',w'} W_{f,c,h',w'} \frac{\partial L}{\partial Y_{n,f,h-h'+1,w-w'+1}} \\ &= \sum_{f} \Big(W_{f,c} *_{\text{full}} \big(\frac{\partial L}{\partial Y_{n,f}}\big)\Big)_{h,w}.\end{aligned}$$

Therefore we get that $\frac{\partial L}{\partial X_{n,c}} = \sum_f W_{f,c} *_{\text{full}} \left(\frac{\partial L}{\partial Y_{n,f}}\right)$.

- Show $\frac{\partial L}{\partial W_{f,c}} = \sum_n X_{n,c} *_{\text{filt}} \left(\frac{\partial L}{\partial Y_{n,f}}\right)$:

  Similarly by the chain rule, we have

  $$\frac{\partial L}{\partial W_{f,c,h',w'}} = \sum_{n,h'',w''} \frac{\partial L}{\partial Y_{n,f,h'',w''}} \frac{\partial Y_{n,f,h'',w''}}{\partial W_{f,c,h',w'}},$$

  and we need to calculate $\frac{\partial Y_{n,f,h'',w''}}{\partial W_{f,c,h',w'}}$.

  Note that

  $$\begin{aligned}Y_{n,f,h'',w''} &= \sum_{h',w'} X_{n,c,h',w'} W_{f,c,h'-h''+1,w'-w''+1} \\ &= \sum_{h',w'} X_{n,c,h'+h''-1,w'+w''-1} W_{f,c,h',w'},\end{aligned}$$

  we have

  $$\frac{\partial Y_{n,f,h'',w''}}{\partial W_{f,c,h',w'}} = X_{n,c,h'+h''-1,w'+w''-1}.$$

  Therefore we get

  $$\frac{\partial L}{\partial W_{f,c,h',w'}} = \sum_{n,h'',w''} X_{n,c,h'+h''-1,w'+w''-1} \frac{\partial L}{\partial Y_{n,f,h'',w''}},$$

  and thus $\frac{\partial L}{\partial W_{f,c}} = \sum_n X_{n,c} *_{\text{filt}} \left(\frac{\partial L}{\partial Y_{n,f}}\right)$.

2 Logistic regression and beyond

2.3 A simple logistic classifier

The test accuracy for my best model is 93.6%.

2.4 A 2-layer neural network

The different hidden units numbers I tried and the corresponding validation accuracy are listed below.

Hidden units	Validation Accuracy
50	96.8%
100	96.8%
200	96.8%
500	97.2%
1000	97.6%
1500	97.2%

The values of validation accuracy are pretty close. Based on the validation accuracy, I used 1000 hidden units and the test accuracy is 93.2%.

The implementation for logistic is in the file **logistic.py**. The codes for the model training are in the file **logistic_model.py**.

3 SVM and beyond

3.3 A binary SVM classifier

The test accuracy for my best model is 91.8%.

3.4 A 2-layer neural network with hinge loss

The different hidden units numbers I tried and the corresponding validation accuracy are listed below.

Hidden units	Validation Accuracy
50	96%
100	96.4%
200	96%
500	95.6%
1000	95.2%

Based on the validation accuracy, I used 100 hidden units and the test accuracy is 92.4%.

The implementation for SVM is in the file **svm.py**. The codes for the model training are in the file **svm_model.py**.

4 Softmax regression and beyond

4.3 A softmax multi-class classifier

The test accuracy for my best model is 95.88%.

4.4 A 2-layer neural network with softmax loss

The different hidden units numbers I tried and the corresponding validation accuracy are listed below.

Hidden units	Validation Accuracy
50	98.1333%
100	98.6333%
200	98.6167%
500	98.6333%
1000	98.7333%
1500	98.6833%

The values of validation accuracy are pretty close. I used 1000 hidden units and the test accuracy is 96.88%.

The implementation for softmax is in the file **softmax.py**. The codes for the model training are in the file **softmax_model.py**.

5 CNN for multi-class classification

5.3 Simple CNN

For the interest of time, I only tried two different sets of values of filter size and number of hidden units. One is trained with a small filter size and small number of hidden units, while the other is trained with larger values.

With filter_size = 5, num_filters = 16, and hidden_dim = 16, the best validation accuracy is 97.40%.

With filter_size = 7, num_flters = 32, and hidden_dim = 100, the best validation accuray is 98.22%.

Therefore, I used the latter set of values and got the test accuracy of 98.38%.

5.4 CNN with dropout and batch normalization

For this problem, I also tried two different architectures.

The first architecture is:

conv – relu – max pool – fc – batch norm – relu – dropout (p=0.5) – fc.

The best validation accuracy is 92.07%.

The second architecture is:

conv – batch norm – relu – max pool – dropout (p=0.8) – fc – relu – fc.

The best validation accuracy is 98.63%.

Based on the validation accuracy, I chose to use the second architecture and got the test accuracy of 98.61%.

The implementation for simple CNN is in the file **cnn.py**. The codes for the model training are in the file **cnn_model.py**.

The implementation for CNN with dropout and batch normalization is in the file **cnn_bn_drop.py**. The codes for the model training are in the file **cnn_bn_model.py**.

6 CNN for CIFAR10 image classification

6.2

The classification accuracy on the test set is 68%.

6.3

In this part, I tired several different feature size and tried to add some activation layers. For the best model I've tried, I increase the output channels in each layer two times as large as before, and add a ReLU activation between two fully-connected layer. The test accuracy is 74% now.

The codes for 6.2 are in the file **vgg.py**. The codes for 6.3 are in the file **vgg_v2.py**

7 Short answer questions

7.1

The main difficulty of training deep neural network with the sigmoid non-linearity is gradient vanishing. The gradient of sigmoid will be very close to zero when the neuron's activation is very close to either 0 or 1. More specifically, in backpropogation, we need to multiply the gradient of a neuron with the gradient of its output for the loss function, and if the local gradient is too close to zero, then it will make the whole gradient very small. Moreover, the output of sigmoid is not zero-centered, which will make the gradients on the weights have the same signs (all positive or all negative). This may make the updates of weight zig-zagging.

7.2

During inference, we should multiply the output of the dropout layer by p, since at test time all neurons are always active and we should scale the activations so that for each neuron the output at test time is equal to the expected output at training time. To be more specific, when we do forward propogtion in test mode,

$$y = W(x \times p).$$

7.3

If the training loss goes down quickly and then diverges during training, I would decrease the learning rate and increase the batch size.