

# EECS 498/598 Deep Learning - Homework 1

January 22th, 2019

## Instructions

- This homework is Due February 8th at 5pm. Late submission policies apply.
- You will submit a write-up and your code for this homework.
- Submission instructions will be updated soon.

## 1 [50 points] Neural network layer implementation.

In this problem, you will implement various layers.  $X, Y$  will represent the input and the output of the layers, respectively.  $L$  is a scalar valued function.

For each layer, in addition to the following computations, implement the corresponding forward and backward passes in `layers.py`.

### 1. Fully-connected layer

Let  $X \in \mathbb{R}^{d_1 \times \dots \times d_k}$ . Consider a dense layer with parameters  $W \in \mathbb{R}^{d_k \times d_o}, b \in \mathbb{R}^{d_o}$ . The layer outputs tensor  $Y \in \mathbb{R}^{d_1 \times \dots \times d_{k-1} \times d_o}$  where  $Y_{i_1, \dots, i_{k-1}}$  is given by  $W^T X_{i_1, \dots, i_{k-1}} + b$  for all valid indices  $i_1, \dots, i_{k-1}$ .

Compute the partial derivatives  $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}, \frac{\partial L}{\partial X}$  in terms of  $\frac{\partial L}{\partial Y}$ .

### 2. ReLU

Let  $X$  be a tensor and  $Y = \text{ReLU}(X)$ . Express  $\frac{\partial L}{\partial X}$  in terms of  $\frac{\partial L}{\partial Y}$ .

### 3. Dropout

Given a dropout mask  $M$ , let  $Y = X \odot M$ , where  $\odot$  represents element-wise multiplication. Express  $\frac{\partial L}{\partial X}$  in terms of  $\frac{\partial L}{\partial Y}$ .

Note: For the forward pass implementation, you will have to consider train and test scenarios.

### 4. Batch Normalization

Let  $X$  be a 2D tensor of input instances. The output of the layer  $Y$  is given by  $Y_i = \frac{X_i - \mu}{\sigma}$  where  $\mu = \frac{1}{n} \sum_j X_j$  and  $\sigma = \sqrt{\sum_j \|X_j - \mu\|^2 + \epsilon}$

Derive expressions for  $\frac{\partial \mu}{\partial X_i}$  and  $\frac{\partial \sigma}{\partial X_i}$ .

Based on this, derive an expression for  $\frac{\partial L}{\partial X_i}$  in terms of  $\frac{\partial L}{\partial Y}$ .

Note: For the forward pass implementation, you will have to consider train and test scenarios.

## 5. Convolution

Given 2-d tensors  $\mathbf{a} \in \mathbb{R}^{H \times W}$  and  $\mathbf{b} \in \mathbb{R}^{H' \times W'}$ , the full convolution  $*_{\text{full}}$  is defined by

$$(\mathbf{a} *_{\text{full}} \mathbf{b})_{i,j} = \sum_{m,n} a_{m,n} b_{i-m+1, j-n+1}$$

The convolution output has size  $\mathbf{a} *_{\text{full}} \mathbf{b} \in \mathbb{R}^{H+H'-1 \times W+W'-1}$ .

It is also useful to consider the filtering operation  $*_{\text{filt}}$ , defined by

$$(\mathbf{a} *_{\text{filt}} \mathbf{b})_{i,j} = \sum_{m,n} a_{i+m-1, j+n-1} b_{m,n}$$

The filtering output has size  $\mathbf{a} *_{\text{filt}} \mathbf{b} \in \mathbb{R}^{H-H'+1 \times W-W'+1}$ .

Assume the input to the layer is given by  $X \in \mathbb{R}^{N \times C \times H \times W}$ . Consider a convolutional kernel  $W \in \mathbb{R}^{F \times C \times H' \times W'}$ . The output of the layer is given by  $Y \in \mathbb{R}^{N \times F \times H'' \times W''}$  where  $H'' = H - H' + 1, W'' = W - W' + 1$ . Convolution produces  $F$  output feature maps  $Y_{n,f} = \sum_c X_{n,c} *_{\text{full}} W_{f,c}$ .

Show that

$$\frac{\partial L}{\partial X_{n,c}} = \sum_f \frac{\partial L}{\partial Y_{n,f}} *_{\text{filt}} W_{f,c}$$

and

$$\frac{\partial L}{\partial W_{f,c}} = \sum_n \frac{\partial L}{\partial Y_{n,f}} *_{\text{filt}} X_{n,c}$$

To answer question 2-5, please read through `solver.py` and familiarize yourself with the API. To build the models, please use the intermediate layers you implemented in question 1. After doing so, use a `Solver` instance to train the models.

For questions 2 and 3, use the data file `data.pkl` provided in Canvas for training and evaluating models. The data is provided as a tuple (Input features, Targets). Use the first 500 instances for training and 250 instances each for validation and testing, respectively.

## 2 [8 points] Logistic regression and beyond: Binary classification with a sigmoid output layer

1. Implement the logistic loss layer `logistic_loss` in `layers.py`.
2. Implement a logistic classifier using the starter code provided in `logistic.py`.
3. Train a simple logistic classifier (with no hidden units). Report the test accuracy for the best model you identified.
4. Train a 2 layer neural network with logistic regression as the output layer. Identify and report an appropriate number of hidden units based on the validation set. Report the test accuracy for your best model.

### **3 [8 points] SVM and beyond: binary classification with a hinge-loss output layer**

1. Implement the hinge-loss layer `svm_loss` in `layers.py`.
2. Implement a binary SVM classifier using the starter code provided in `svm.py`.
3. Train a binary SVM classifier (with no hidden units). Report the test accuracy for the best model you identified.
4. Train a 2 layer neural network with hinge-loss as the output layer. Identify and report an appropriate number of hidden units based on the validation set. Report the best test accuracy for your best model.

### **4 [8 points] Softmax regression and beyond: multi-class classification with a softmax output layer**

1. Implement softmax loss layer `softmax_loss` in `layers.py`
2. Implement softmax multi-class classification using the starter code provided in `softmax.py`.
3. Train a softmax multi-class classifier (with no hidden units) on MNIST dataset. Report the test accuracy.
4. Train a 2 layer neural network with softmax-loss as the output layer on MNIST dataset. Identify and report an appropriate number of hidden units based on the validation set (you can leave 1/10 data samples in the train dataset as validation dataset). Report the best test accuracy for your best model.

### **5 [8 points] Convolutional Neural Network for multi-class classification**

1. Implement the forward and backward passes of max-pooling layer in `layers.py`
2. Implement CNN for softmax multi-class classification using the starter code provided in `cnn.py`.
3. Train the CNN multi-class classifier on MNIST dataset. Identify and report an appropriate filter size of convolutional layer and appropriate number of hidden units in fully-connected layer based on the validation set (you can leave 1/10 data samples in the train dataset as validation dataset). Report the best test accuracy for your best model.
4. Train the CNN multi-class classifier with dropout and batch normalization on MNIST dataset. Identify and report an appropriate architecture and appropriate rate of dropout based on the validation set (you can leave 1/10 data samples in the train dataset as validation dataset). Report the best test accuracy for your best model.

## 6 [12 points] Convolutional Neural Networks for CIFAR10 image classification.

You will implement and train VGG11 model in this question using `vgg.py`. In this question, you can use the layer in PyTorch.

1. Implement the VGGNet in `VGG` class and train the model in `train` function. Please follow the description about architecture in comments to build the network.
2. Test the model in `test` function and report the classification accuracy on the test set.
3. You are encouraged to try different optimizer, image preprocessing, activation function, convolutional feature size, learning rate strategy, architecture of the network to make the classification accuracy better on test set.

## 7 [6 points] Short answer questions

1. Describe the main difficulty in training deep neural networks with the sigmoid non-linearity.
2. Consider a linear layer  $W$ . During training, dropout with probability  $p$  is applied to the layer input  $x$  and the output is given by  $y = W(x \odot m)$  where  $m$  represents the dropout mask. How would you use this layer during inference to reduce the train/test mismatch (i.e., what is the input/output relationship).
3. If the training loss goes down quickly and then diverges during training, what hyperparameters would you modify ?