COMSE 6998 section 012
PRACT DEEP LEARNING SYS PERF
Instructor: Parijat Dube

Homework 2

Due: October 15, 2019

Problem 1 - *Perceptron* 10 points

Consider a 2-dimensional data set in which all points with $x_1 > x_2$ belong to the positive class, and all points with $x_1 \leq x_2$ belong to the negative class. Therefore, the true separator of the two classes is linear hyperplane (line) defined by $x_1 - x_2 = 0$. Now create a training data set with 20 points randomly generated inside the unit square in the positive quadrant. Label each point depending on whether or not the first coordinate x_1 is greater than its second coordinate x_2. Now consider the following loss function for training pair $(\bar{X}, y)$ and weight vector $\bar{W}$:

$$L = \max\{0, a - y(\bar{W} \cdot \bar{X})\},$$

where the test instances are predicted as $\hat{y} = \text{sign}\{\bar{W} \cdot \bar{X}\}$. For this problem, $\bar{W} = [w_1, w_2]$, $\bar{X} = [x_1, x_2]$ and $\hat{y} = \text{sign}(w_1 x_1 + w_2 x_2)$. A value of $a = 0$ corresponds to the perceptron criterion and a value of $a = 1$ corresponds to hinge-loss.

1. Implement the perceptron algorithm without regularization, train it on the 20 points above, and test its accuracy on 1000 randomly generated points inside the unit square. Generate the test points using the same procedure as the training points. (4)
2. Change the perceptron criterion to hinge-loss in your implementation for training, and repeat the accuracy computation on the same test points above. Regularization is not used. (2)
3. In which case do you obtain better accuracy and why? (2)
4. In which case do you think that the classification of the same 1000 test instances will not change significantly by using a different set of 20 training points? (2)

Problem 2 - *Softmax Activation Functions* 5 points

Consider the softmax activation function in the output layer, in which real-valued outputs $v_1, \ldots, v_k$ are converted into probabilities as follows:

$$o_i = \frac{\exp(v_i)}{\sum_{j=1}^{k} \exp(v_j)} \quad \forall i \in \{1, \ldots, k\}.$$

1. Show that the value of $\frac{\partial o_i}{\partial v_j}$ is $o_i(1 - o_i)$ when $i = j$ and $-o_i o_j$ when $i \neq j$. (2)
2. Assume that we are using cross-entropy loss $L = -\sum_{i=1}^{k} y_i \log(o_i)$, where $y_i \in \{0, 1\}$ is the one-hot encoded class label over different values of $i \in \{1, \ldots, k\}$. Use the result in part 1 to show the correctness of following equation:
$$\frac{\partial L}{\partial v_i} = o_i - y_i. \text{ (3)}$$

Problem 3 - *Weight Initialization, Dead Neurons* 15 points

Read the blog by A. Pernunicic on weight initialization. You will reuse the code at github repo linked in the blog for explaining vanishing and exploding gradients. You can use the same 5 layer neural network model as in the repo and the same dataset.

1. Explain vanishing and exploding gradients phenomenon using different standard normalization and tanh and sigmoid activation functions. Then show how *Xavier (aka Glorot normal) initialization* of

Homework 2

weights helps in dealing with this problem. Next use ReLU activation and show that instead of Xavier initialization, *He initialization* works better for ReLU activation. You can plot activations at each of the 5 layers to answer this question. (5)

2. The dying ReLU is a kind of vanishing gradient, which refers to a problem when ReLU neurons become inactive and only output 0 for any input. In the worst case of dying ReLU, ReLU neurons at a certain layer are all dead, i.e., the entire network dies and is referred as the dying ReLU neural networks in Lu et al (reference below). A dying ReLU neural network collapses to a constant function. Show this phenomenon using any one of the three 1-dimensional functions in page 11 of Lu et al. Use a 10-layer ReLU network with width 2 (hidden units per layer). Use minibatch of 64 and draw training data uniformly from $[-\sqrt{7}, \sqrt{7}]$. Perform 1000 independent training simulations each with 3,000 training points. Out of these 1000 simulations, what fraction resulted in neural network collapse. Is your answer close to over 90% as was reported in Lu et al. ? (5)

3. Instead of ReLU consider Leaky ReLU activation as defined below:

$$\phi(z) = \begin{cases} z & \text{if } z > 0 \\ 0.01z & \text{if } z \leq 0. \end{cases}$$

Run the 1000 training simulations in part 2 with Leaky ReLU activation and keeping everything else same. Again calculate the fraction of simulations that resulted in neural network collapse. Did Leaky ReLU help in preventing dying neurons ? (5)**[BONUS]**

References:

- Andre Perunicic. Understand neural network weight initialization. Available at https://intoli.com/blog/neural-network-initialization/https://intoli.com/blog/neural-network-initialization/
- Initializers - Keras documentation. https://keras.io/initializers/.
- Lu Lu et al. Dying ReLU and Initialization: Theory and Numerical Examples .

Problem 4 - *Batch Normalization, Dropout, MNIST* 20 points

Batch normalization and Dropout are used as effective regularization techniques. However its not clear which one should be preferred and whether their benefits add up when used in conjunction. In this problem we will compare batch normalization, dropout, and their conjunction using MNIST and LeNet-5 (see e.g., https://engmrk.com/lenet-5-a-classic-cnn-architecture/). LeNet-5 is one of the earliest convolutional neural network developed for image classification and its implementation in all major framework is available. You can refer to Lecture 3 slides for definition of standardization and batch normalization.

1. Explain the terms co-adaptation and internal covariance-shift. Use examples if needed. *You may need to refer to two papers mentioned below to answer this question.* (4)

2. Batch normalization is traditionally used in hidden layers, for input layer standard normalization is used. In standard normalization the mean and standard deviation are calculated using the entire training dataset whereas in batch normalization these statistics are calculated for each mini-batch. Train LeNet-5 with standard normalization of input and batch normalization for hidden layers. What are the learned batch norm parameters for each layer ? (4)

3. Next instead of standard normalization use batch normalization for input layer also and train the network. What are the learned batch norm parameters for each layer (including input) ? Compare the train/test accuracy and loss for the two cases ? Did batch normalization for input layer improve performance ? (4)

Homework 2

4. Train the network without batch normalization but this time use dropout. For hidden layers use dropout probability of 0.5 and for input layer take it to be 0.2 Compare test accuracy using dropout to test accuracy obtained using batch normalization in part 2 and 3. (4)

5. Now train the network using both batch normalization and dropout. How does the performance (test accuracy) of the network compare with the cases with dropout alone and with batch normalization alone ? (4)

References:

- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R.Salakhutdinov . Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Available at at https://www.cs.toronto.edu/ rsalakhu/papers/srivastava14a.pdf.
- S. Ioffe, C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Available at https://arxiv.org/abs/1502.03167.

Problem 5 - *Universal Approximators: Depth Vs. Width* 15 points

Multilayer layer feedforward network, with as little as two layers and sufficiently large hidden units can approximate any arbitrary function. Thus one can tradeoff between deep and shallow networks for the same problem. In this problem we will study this tradeoff using the *Eggholder* function defined as:

$$f(x_1, x_2) = -(x_2 + 47)\sin\sqrt{\left|\frac{x_1}{2} + (x_2 + 47)\right|} - x_1 \sin\sqrt{|x_1 - (x_2 + 47)|}$$

Let $y(x_1, x_2) = f(x_1, x_2) + \mathcal{N}(0, 0.3)$ be the function that we want to learn from a neural network through regression with $-512 \leq x_1 \leq 512$ and $-512 \leq x_2 \leq 512$. Draw a dataset of 100K points from this function (uniformly sampling in the range of x_1 and x_2) and do a 80/20 training/test split.

1. Assume that total budget for number of hidden units we can have in the network is 512. Train a 1, 2, and 3 hidden layers feedforward neural network to learn the regression function. For each neural network you can consider a different number of hidden units per hidden layer so that the total number of hidden units does not exceed 512. I would recommend to work with 16, 32, 64, 128, 256, 512, hidden units per layer. So if there is only one hidden layer you can have at most 512 units in that layer. If there are two hidden layers, you can have any combination of hidden units in each layer, e.g., 16 and 256, 64 and 128, etc. such that the total is less than 512. Plot the RMSE (Root Mean Square Error) on test set for networks with different number of hidden layers as a function of total number of hidden units. If there are more than one network with the same number of hidden units (say a two hidden layer with 16 in first layer and 128 in second layer and another network with 128 in first layer and 16 in second) you will use the average RMSE. So you will have a figure with three curves, one each for 1, 2, and 3 layer networks, with x-axis being the total number of hidden units. Also plot another curve but with the x-axis being the number of parameters (weights) that you need to learn in the network. (10)

2. Comment on the tradeoff between number of parameters and RMSE as you go from deeper (3 hidden layers) to shallow networks (1 hidden layer). Also measure the wall clock time for training each configuration and plot training time vs number of parameters. Do you see a similar tradeoff in training time ? (5)

For networks with 2 and 3 layers you will use batch normalization as regularization. For hidden layers use ReLU activation and for training use SGD with Nesterov momentum. Take a batch size of 1000 and train for 2000 epochs. You can pick other hyperparameter values (momentum, learning rate schedule) or use the default values in the framework implementation.

Homework 2

Problem 6 - *Learning Rate, Batch Size, FashionMNIST* 25 points

Recall cyclical learning rate policy discussed in Lecture 3. The learning rate changes in cyclical manner between lr_{min} and lr_{max}, which are hyperparameters that need to be specified. For this problem you first need to read carefully the article referenced below as you will be making use of the code there (in Keras) and modifying it as needed. For those who want to work in Pytorch there are open source implementations of this policy available which you can easily search for and build over them. You will work with FashionMNIST dataset and MiniGoogLeNet (described in reference).

1. Summarize FashionMNIST dataset, total dataset size, training set size, validation set size, number of classes, number of images per class. Show any 3 representative images from any 3 classes in the dataset. (2)

2. Fix batch size to 64 and start with 10 candidate learning rates between 10^{-9} and 10^{1} and train your model for 5 epochs. Plot the training loss as a function of learning rate. You should see a curve like Figure 3 in reference below. From that figure identify the values of lr_{min} and lr_{max}. (4)

3. Use the cyclical learning rate policy (with exponential decay) and train your network using batch size 64 and lr_{min} and lr_{max} values obtained in part 1. Plot train/validation loss and accuracy curve (similar to Figure 4 in reference). (3)

4. Fix learning rate to lr_{min} and train your network starting with batch size 64 and going upto 8192. If your GPU cannot handle large batch sizes, you can employ effective batch size approach as discussed in Lecture 3 to simulate large batches. Plot the training loss as a function of batch size. Do you see a similar behavior of training loss with respect to batch size as seen in part 2 with respect to learning rate ? (5)

5. Can you identify b_{min} and b_{max} from the figure in part 4 for devising a cyclical batch size policy ? Create an algorithm for automatically determining batch size and show its steps in a block diagram as in Figure 1 of reference. (2)

6. Use b_{min} and b_{max} values identified in part 3 and devise a cyclical batch size policy such that the batch size changes in a cyclical manner between b_{min} and b_{max}. In part 3 we did exponential decrease in learning rate as training progress. What should be an analogous trajectory for batch size as training progresses, exponential increase or decrease ? Use cyclical batch size policy (with appropriate trajectory) and train your network using learning rate lr_{min}. (6)

7. Compare the best accuracy from the two cyclical policies. Which policy gives you the best accuracy ? (2)

PS: In part 3 of problem we are doing cyclical learning rate with exponential decay. The code under "Keras Learning Rate Finder" in the blog implements triangular policy, you may need to change it to have exponential decay as mentioned in the first reference below. For part 4 and 6, you will be writing your own python project "Keras Batch Finder".

References:

1. Leslie N. Smith Cyclical Learning Rates for Training Neural Networks. Available at https://arxiv.org/abs/1506.01186.

2. Keras implementation of cyclical learning rate policy. Available at https://www.pyimagesearch.com/2019/08/05/keras-learning-rate-finder/.

Homework 2

Problem 7 - *Adaptive Learning Rate Methods, MNIST* 15 points

We will consider five methods, AdaGrad, RMSProp, RMSProp+Nesterov, AdaDelta, Adam, and study their convergence using MNIST dataset. We will use multi-layer neural network model with two fully connected hidden layers with 1000 hidden units each and ReLU activation with minibatch size of 128.

1. Write the weight update equations for the five adaptive learning rate methods. Explain each term very clearly. What are the hyperparameters in each policy ? Explain how AdaDelta and Adam are different from RMSProp. (5)

2. Train the neural network using all the five methods with L_2-regularization for 200 epochs each and plot the training loss vs number of epochs. Which method performs best (lowest training loss) ? (4)

3. Add dropout (probability 0.2 for input layer and 0.5 for hidden layers) and train the neural network again using all the five methods for 200 epochs. Compare the training loss with that in part 2. Which method performs the best ? For the five methods, compare their training time (to finish 200 epochs with dropout) to the training time in part 2 (to finish 200 epochs without dropout). (4)

4. Compare test accuracy of trained model for all the five methods from part 2 and part 3. Note that to calculate test accuracy of model trained using dropout you need to appropriately scale the weights (by the dropout. probability). (2)