COMS E6998 section 012
PRACT DEEP LEARNING SYS PERF
Instructor: Parijat Dube

Homework 2

Due: October 13, 2022

Problem 1 - *Perceptron* 15 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. You need to implement the perceptron algorithm without regularization (don't use any existing implementation of perceptron), 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. You need to have your own implementation of the perceptron algorithm. (6)
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. (5)
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 - *Weight Initialization, Dead Neurons, Leaky ReLU* 30 points

Read the two blogs, one by Andre Pernunicic and other by Daniel Godoy 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 gradients phenomenon using standard normalization with different values of standard deviation and tanh and sigmoid activation functions. Then show how *Xavier (aka Glorot normal) initialization* of 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. (10)
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. ? (10)
3. Instead of ReLU consider Leaky ReLU activation as defined below:

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

Homework 2

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 ? (10)

References:

- Andre Perunicic. Understand neural network weight initialization. Available at https://intoli.com/blog/neural-network-initialization/
- Daniel Godoy. Hyper-parameters in Action Part II — Weight Initializers.
- Initializers - Keras documentation. https://keras.io/initializers/.
- Lu Lu et al. Dying ReLU and Initialization: Theory and Numerical Examples .

Problem 3 - *Batch Normalization, Dropout, MNIST* 25 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., http://yann.lecun.com/exdb/lenet/). 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.* (5)
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 ? (5)
3. Next instead of standard normalization use batch normalization for input layer also and train the network. Plot the distribution of learned batch norm parameters for each layer (including input) using violin plots. Compare the train/test accuracy and loss for the two cases ? Did batch normalization for input layer improve performance ? (5)
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. (5)
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 ? (5)

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.

Homework 2

Problem 4 - *Universal Approximators: Depth Vs. Width* 30 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. We 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. (20)

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 ? (10)

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.