ECE-GY 9143
INTRO HIGH PERF DEEP LEARNING
Instructor: Parijat Dube
Due: March 25, 2022

Lab 3

Problem 1 - *Weight Initialization, Dead Neurons, Leaky ReLU* 20 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. (8)
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 to 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 on page 13 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. ? (8)
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 keep everything else the same. Again calculate the fraction of simulations that resulted in neural network collapse. Did Leaky ReLU help in preventing dying neurons? (4)

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 2 - *Batch Normalization, Dropout, MNIST* 20 points

Batch normalization and Dropout are used as effective regularization techniques. However, it is 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 networks developed for image classification and its implementation in all major frameworks is available.

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 the 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 the 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 the input layer improve performance? (4)

4. Train the network without batch normalization but this time use dropout. For hidden layers use a 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 parts 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 3 - *Learning Rate, Batch Size, FashionMNIST* 15 points

Recall the cyclical learning rate policy discussed in the class. The learning rate changes in a 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 the FashionMNIST dataset and MiniGoogLeNet (described in reference).

1. 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 the learning rate. You should see a curve like Figure 3 in the reference below. From that figure identify the values of lr_{min} and lr_{max}. (2)

2. 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)

3. We want to test if increasing batch size for a fixed learning rate has the same effect as decreasing learning rate for fixed batch size. Fix learning rate to lr_{max} and train your network starting with batch size 32 and incrementally going up to 16384 (in increments of a factor of 2; like 32, 64...). You can choose a step size (in terms of the number of iterations) to increment the batch size. If your GPU cannot handle large batch sizes, you need to employ an effective batch size approach as discussed in Lecture 3 to simulate large batches. Plot the training loss. Is the generalization of your final model similar or different from than cyclical learning rate policy? (10)

Lab 3

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/.

Problem 4 - *Adaptive Learning Rate Methods, CIFAR-10* 20 points

We will consider five methods, AdaGrad, RMSProp, RMSProp+Nesterov, AdaDelta, Adam, and study their convergence using the CIFAR-10 dataset. We will use a multi-layer neural network model with two fully connected hidden layers with 1000 hidden units each and ReLU activation with a minibatch size of 128.

1. Write the weight update equations for the five adaptive learning rate methods. Explain each term clearly. What are the hyperparameters in each policy? Explain how AdaDelta and Adam are different from RMSProp. (5+1)
2. Train the neural network using all the five methods with L_2-regularization for 200 epochs each and plot the training loss vs the number of epochs. Which method performs best (lowest training loss)? (5)
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). (5)
4. Compare test accuracy of the trained model for all the five methods from part 2 and part 3. Note that to calculate test accuracy of the model trained using dropout you need to appropriately scale the weights (by the dropout probability). (4)

References:

- The CIFAR-10 Dataset.

Problem 5 - *Convolutional Neural Networks Architectures* 25 points

In this problem, we will study and compare different convolutional neural network architectures. We will calculate the number of parameters (weights, to be learned) and the memory requirement of each network. We will also analyze inception modules and understand their design.

1. Calculate the number of parameters in Alexnet. You will have to show calculations for each layer and then sum it to obtain the total number of parameters in Alexnet. When calculating you will need to account for all the filters (size, strides, padding) at each layer. Look at Sec. 3.5 and Figure 2 in Alexnet paper (see reference). Points will only be given when explicit calculations are shown for each layer. (4)
2. VGG (Simonyan et al.) has an extremely homogeneous architecture that only performs 3x3 convolutions with stride 1 and pad 1 and 2x2 max pooling with stride 2 (and no padding) from the beginning to the end. However VGGNet is very expensive to evaluate and uses a lot more memory and parameters. Refer to VGG19 architecture on page 3 in Table 1 of the paper by Simonyan et al. You need to complete Table 1 below for calculating activation units and parameters at each layer in VGG19 (without counting biases). Its been partially filled for you. (6)

Lab 3

Layer	Number of Activations (Memory)	Parameters (Compute)
Input	224*224*3=150K	0
CONV3-64	224*224*64=3.2M	(3*3*3)*64 = 1,728
CONV3-64	224*224*64=3.2M	(3*3*64)*64 = 36,864
POOL2	112*112*64=800K	0
CONV3-128		
CONV3-128		
POOL2	56*56*128=400K	0
CONV3-256		
CONV3-256	56*56*256=800K	(3*3*256)*256 = 589,824
CONV3-256		
CONV3-256		
POOL2		0
CONV3-512	28*28*512=400K	(3*3*256)*512 = 1,179,648
CONV3-512		
CONV3-512	28*28*512=400K	
CONV3-512		
POOL2		0
CONV3-512		
CONV3-512		
CONV3-512		
CONV3-512		
POOL2		0
FC	4096	
FC	4096	4096*4096 = 16,777,216
FC	1000	
TOTAL		

Table 1: VGG19 memory and weights

Lab 3

3. VGG architectures have smaller filters but deeper networks compared to Alexnet (3x3 compared to 11x11 or 5x5). Show that a stack of N convolution layers each of filter size $F \times F$ has the same receptive field as one convolution layer with filter of size $(NF - N + 1) \times (NF - N + 1)$. Use this to calculate the receptive field of 3 filters of size 5x5. (3)

4. The original Googlenet paper (Szegedy et al.) proposes two architectures for Inception module, shown in Figure 2 on page 5 of the paper, referred to as naive and dimensionality reduction respectively.

   (a) What is the general idea behind designing an inception module (parallel convolutional filters of different sizes with a pooling followed by concatenation) in a convolutional neural network ? (2)

   (b) Assuming the input to inception module (referred to as "previous layer" in Figure 2 of the paper) has size 32x32x256, calculate the output size after filter concatenation for the naive and dimensionality reduction inception architectures with number of filters given in Figure 1. (3)

   (c) Next calculate the total number of convolutional operations for each of the two inception architecture again assuming the input to the module has dimensions 32x32x256 and number of filters given in Figure 1. (3)

   (d) Based on the calculations in part (c) explain the problem with naive architecture and how dimensionality reduction architecture helps (*Hint: compare computational complexity*). How much is the computational saving ? (2+2)

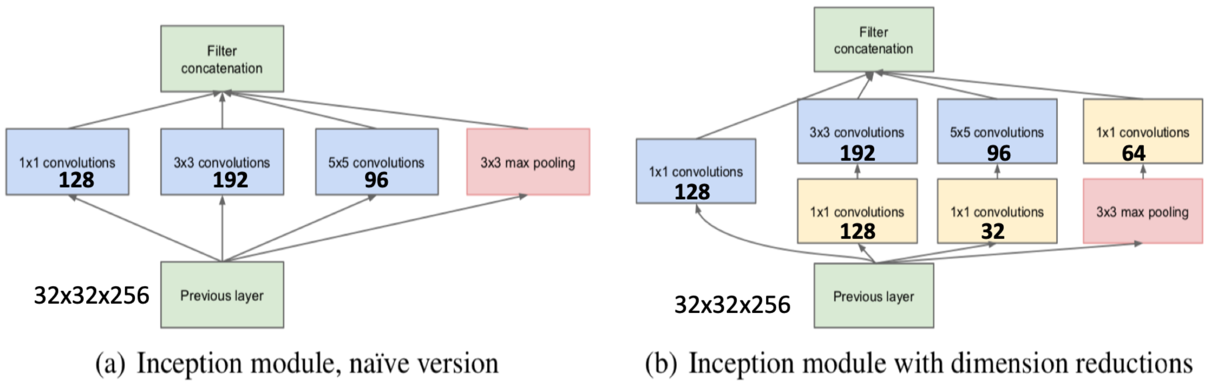

Figure 1: Two types of inception module with number of filters and input size for calculation in Question 3.4(b) and 3.4(c).

References:

- (Alexnet) Alex Krizhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks. Paper available at https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- (VGG) Karen Simonyan et al. Very Deep Convolutional Networks for Large-scale Image Recognition. Paper available at https://arxiv.org/pdf/1409.1556.pdf
- (Googlenet) Christian Szegedy et al. Going deeper with convolutions. Paper available at https://arxiv.org/pdf/1409.4842.pdf