# ***Assignment 2***

# Backprop, Hyperparameter searching, Setup training, Pytorch

Hui Xue

[hui.xue@nih.gov](mailto:hui.xue@nih.gov)

1. **Multi-choice and short-answer questions**
2. Which of the following activation functions can lead to vanishing gradient problem?
3. ReLU
4. Leaky ReLU
5. Tanh
6. Sigmoid
7. You design a loss function , where **.** , . Compute **:**
8. Backprop is a key step in deep learning training. Which of the following statements about backprop are correct?
9. Backprop starts from the input and computes derivatives through all layers until the loss function.
10. For a given layer with given downstream gradient, its backprop has nothing to do with loss function.
11. If two layers have their downstream gradients ready, their backprop can be computed in parallel.
12. It is almost impossible to manually perform backprop for a complex network.
13. Batch is a subset of training samples. Which of the following statements about batch and batch size are correct?
    * 1. We should always use mini-batch, instead of processing whole dataset.
      2. Batch\_size=1 is a bad idea. You should not use it.
      3. Larger the batch size, smaller the learning rate.
      4. When doing the training, batch\_size is a hyper-parameter.
14. After training the model, you got training accuracy of 97%, validation accuracy 87% and test accuracy 77%. What kind of problem are you facing?
    * 1. Bias problem.
      2. Variance problem.
      3. Both bias and variance problems
      4. No problem, I am happy.
15. Draw the computational graph to compute and perform forward and backward pass to compute function value and ,, **.** Suppose, , .
16. Draw the computational graph to compute , where and . Perform forward and backward pass to compute function value and ,,, **.**
17. It is import to understand what to do after diagnosing the bias or variance in the training. Here are a set of actions. Sort them into three bins: actions to reducing bias, actions to reduce variance, actions to reduce both.

Actions:

1. Increase number of layers
2. Increase number of neurons
3. Increase regularization strength
4. Add more data
5. Train longer
6. Add data augmentation
7. Add early stopping to the training process
8. Choose a different network architecture

Actions to reduce bias:

Actions to reduce variance:

Actions to reduce both:

1. In your language, explain what the learning rate scheduler is? Why do we want this component in the training loop? Compared to use a constant learning rate, what are the advantages to use a variable learning rate? Is there a reason to increase learning rate during training?
2. A local shop runs a website allowing customers to put in the review comments for the shop. The owner of this business learned you can do deep learning and invited you to build a model to classify the customer reviews to be positive, negative or neural. They collected a dataset including N=50,000 reviews with classification.
3. Before building the model, it may be a good idea to estimate the bayes error. How will you get a rough estimation of the best error rate?
4. The goal is to achieve 95% accuracy in this task. How will you split the data to train/dev/test sets?
5. After training the first model, you got 87% accuracy on training set and 82% on dev set. On the test set, you only got 67% accuracy. What will be the next steps to improve the model?
6. You found a big dataset (N=1,000,000) of customer review classification from an online dealer. This online dealer has a much bigger inventory than the local shop. Is it a good idea to add this dataset into your training process? If so, how will you do it?
7. After a few weeks, you finally got a model which achieves 96% accuracy in training and 95% in dev set. On the test set, the accuracy is now 92%. What can you do to close the final gap?
8. Your employer wants to learn how the model works before deploying it. You picked a test sample with “negative” class and ran it through the model. The model gave probabilities of (0.1, 0.85, 0.05). You are trying to make this case show the model is minimizing the loss to make prediction. What will be the CE-loss in this case? You then picked another case with “negative” class as the ground-truth. This time model gave (0.65, 0.25, 0.1). What will be the CE-loss for this case?
9. Pytorch autograd

It is absolutely essential to understand the dynamic computational graph and autograd in Pytorch. Please read the Pytorch tutorial pages:

<https://pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html#>

<https://pytorch.org/docs/stable/notes/autograd.html>

<https://expoundai.wordpress.com/tag/static-vs-dynamic-graphs/>

You can also search and find more tutorials online about autograd and computational graph online.

1. Please explain the dynamic computational graph and static computational graph. While Tensorflow 1.x uses the static graph as the default, in new version, it has changed to use dynamic graph. What are the advantages of dynamic graphs vs. the static graph?
2. Training DL models can be limited by the available amount of system RAM. This limits the batch\_size one can use during training. One useful method to mitigate the problem is the Gradient Accumulation. Read about this idea (e.g. <https://www.quora.com/What-is-gradient-Accumulation-in-deep-learning-1>). Here are two implementation of this idea. Which one is correct? Explain the reason.

Method 1:

dataset = MyDataSet(batch\_size=10, ...)

optimizer.zero\_grad()

for i, (input, target) in enumerate(dataset):

    pred = model(input)

    L = loss(pred, target)

    L.backward()

    if((i+1)%10==0):

        optimizer.step()

        optimizer.zero\_grad()

Method 2:

dataset = MyDataSet(batch\_size=10, ...)

L = 0

for i, (input, target) in enumerate(dataset):

    pred = model(input)

    L += loss(pred, target)

    if((i+1)%10==0):

        L /= 10

        optimizer.zero\_grad()

        L.backward()

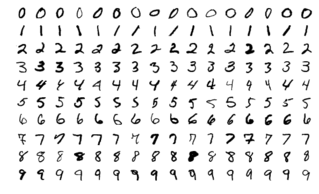
        optimizer.step()

1. Now, with the knowledge of how the auto-differential mechanism works in Pytorch, please implement the autograd functions in a2\_autograd.py. You can test the implementation by running “pytest -v src/a2\_autograd.py”. **Submit the output of this testing run.**

NOTE: Pytorch and Tensorflow have their own autograd engines. They are designed for deep learning purpose, favoring the derivatives to a scalar function. For more general purpose autograd which has much wider utilities in many fields (far beyond DL), you can check JAX (<https://jax.readthedocs.io/en/latest/index.html>).

1. Now it is time to get into the Pytorch. In the first task, you will implement the same 3-layer MLP network to perform the classification. Instead of working on MNIST, we will work on a new dataset:

This dataset included handwritten digits from 0 to 9, with 50,000 images for training and 10,000 for testing. The dataset can be downloaded from <http://yann.lecun.com/exdb/mnist/>. Every image is a 28x28x1 grayscale picture, containing one of the 10 digits:



1. As a multi-class classification problem, given the size of a mini-batch is B, what is the output size of logits of the MLP? After passing the logits through the softmax, what is the size of the probabilities ? Clearly write out the final equation of empirical CE loss. Also, write out the derivative of empirical CE loss to the logits (network outputs before softmax). The answer of the previous question 4d may be useful here.
2. Following the instructions at the course website (https://deeplearningcrashcourse.org/nhlbi2021/) to set up your python development environment. You can use either VSCode(https://code.visualstudio.com/) or PyCharm(https://www.jetbrains.com/pycharm/) for development and debugging.

After setting up the environment, start your python environment and read through the problem py file (*a1\_mlp\_3\_layers.py*). Try to open this python file with VSCode and set break points to step through the code. Please check this video for instructions (<https://www.youtube.com/watch?v=W--_EOzdTHk>). You can also find other tutorials online.

1. The python file *a1\_mlp\_3\_layers.py* contains a 3 layer MLP model, unfinished. Please implement the empirical CE loss function, the forward and backward pass and the gradient descent step. The weights should be initialized from the standard norm distribution and biases are initialized as 0. The training loop has been implemented, including random shuffling the images and performing gradient descent. A portion of the training data was split out as the validation set, to estimate the model performance. After every epoch, the accuracy and loss values are saved.

Please finish the coding and train the model with default parameters (Batch Size, learning rate, number of epochs). **Submit the plots of the accuracy and loss vs. epoch for both training and validation sets.**

One way to debug your model is to let it fit on a single mini-batch with small batch size. Model should be able to fit the data extremely well, getting 100% accuracy and very low loss. Remember to turn off regularization when doing this test. Since the batch size is now small, you need to reduce the learning rate, e.g.

*python3 a1-mlp-3-layers.py --learning\_rate 0.01 --one\_batch\_training True --batch\_size 10 --num\_epochs 1000*

1. Next step is to add L2 regularization to the loss function. Note the L2 norm is computed on the weights only, not the biases. First, write out the empirical loss with L2 regularization as the format of “CE\_loss+. How will you change the gradient computation for weights? Implement the backprog function with regularization and retrain the model. **Submit the plots of the accuracy and loss curves vs. epoch for both training and validation sets**. What do you observe after adding regularization?
2. The python file *a1\_mlp\_N\_layers.py* contains a MLP model with any number of hidden layers, unfinished. Please implement the forward and backward pass and the gradient descent step, with the L2 regularization. Please finish the coding and train the model with default parameters (Batch Size, learning rate, number of epochs). Submit the plots of the accuracy **and loss vs. epoch for both training and validation sets. Compare to the accuracy you got on the 3-layer MLP, what are your findings?**

Now you can change the model architectures to try different number of layers or number of hidden units. Can you get higher accuracy? Explain what you tried.

1. A critical aspect of training models is the reproducibility in terms of model performance. A common tool for python regression test is the pytest package (<https://docs.pytest.org/en/6.2.x/contents.html>). Learn to use pytest to write test cases for your model. A good tutorial can be found at <https://www.youtube.com/watch?v=byaxg00Gf9I>. Finish the test function in the test\_*a1\_mlp\_3\_layers.py and run the test with `pytest -s src`. Submit the output of your test run.*