Due Date: February 12th (11pm), 2019
Instructions

- *For all questions, show your work!*
- *Submit your code ("solution.py" only) and your report (pdf) separately and electronically via the course Gradescope page.*
- *Work off the template and DO NOT modify the name of the file "solution.py" since the code will be automatically graded.*
- ***TAs for this assignment are Jie Fu, Sai Rajeswar, and Akilesh B***

Problem 1

The second part of this assignment will be graded based on your report (including comments and figures). You may create additional scripts aside from the template that we provide and/or reuse the template WITHOUT modifying or renaming it! One way to do it is to create another class object that inherits from the `NN` template.

In the following sub-questions, please specify the *model architecture* (number of hidden units per layer, and the total number of parameters), the *nonlinearity* chosen as neuron activation, *learning rate*, *mini-batch size*. The following tasks should be written as a report and submitted via Gradescope (separately from the code).

Initialization (report) [15] In this sub-question, we consider different initial values for the weight parameters. Set the biases to be zeros, and consider the following settings for the weight parameters:

- **Zero**: all weight parameters are initialized to be zeros (like biases).
- **Normal**: sample the initial weight values from a standard Normal distribution; $w_{i,j} \sim \mathcal{N}(w_{i,j}; 0, 1)$.
- **Glorot**: sample the initial weight values from a uniform distribution; $w^l_{i,j} \sim \mathcal{U}(w^l_{i,j}; -d^l, d^l)$ where $d^l = \sqrt{\frac{6}{h^{l-1}+h^l}}$.

1. Train the model for 10 epochs [1] using the initialization methods above and record the average loss measured on the training data at the end of each epoch (10 values for each setup).

   For these initialization methods, the following hyper-parameters were chosen; a learning rate of 7×10^{-4}, 2 hidden layers of 500 and 400 neurons, and a mini-batch size of 64.

[1] One epoch is one pass through the whole training set.

```
2020-02-08 17:46:31.319436 - Epoch 1 : loss = 2.3025823920240978
2020-02-08 17:46:43.001955 - Epoch 2 : loss = 2.3025796956493596
2020-02-08 17:46:54.548206 - Epoch 3 : loss = 2.30257700386218
2020-02-08 17:47:06.339992 - Epoch 4 : loss = 2.302574316654922
2020-02-08 17:47:18.199736 - Epoch 5 : loss = 2.3025716340199622
2020-02-08 17:47:30.683055 - Epoch 6 : loss = 2.3025689559496856
2020-02-08 17:47:42.533985 - Epoch 7 : loss = 2.3025662824364908
2020-02-08 17:47:53.948825 - Epoch 8 : loss = 2.302563613472792
2020-02-08 17:48:05.408893 - Epoch 9 : loss = 2.3025609490510086
2020-02-08 17:48:16.647286 - Epoch 10 : loss = 2.3025582891635814
zero initialization
Layer 1 - W 1 (784, 500)        b 1 (1, 500)
Layer 2 - W 2 (500, 400)        b 2 (1, 400)
Layer 3 - W 3 (400, 10) b 3 (1, 10)
Total number of parameters: 596910

2020-02-08 17:48:34.012785 - Epoch 1 : loss = 2.0580302500668157
2020-02-08 17:48:50.389234 - Epoch 2 : loss = 1.6435337149232274
2020-02-08 17:49:06.909232 - Epoch 3 : loss = 1.3848512069114924
2020-02-08 17:49:23.824495 - Epoch 4 : loss = 1.2091570947519965
2020-02-08 17:49:40.427536 - Epoch 5 : loss = 1.120734630499463
2020-02-08 17:49:57.303670 - Epoch 6 : loss = 1.0233777498287164
2020-02-08 17:50:14.326337 - Epoch 7 : loss = 0.8985837433154218
2020-02-08 17:50:32.314041 - Epoch 8 : loss = 0.8354616197398361
2020-02-08 17:50:49.955527 - Epoch 9 : loss = 0.8217522399206727
2020-02-08 17:51:07.508274 - Epoch 10 : loss = 0.773342501491482
normal initialization
Layer 1 - W 1 (784, 500)        b 1 (1, 500)
Layer 2 - W 2 (500, 400)        b 2 (1, 400)
Layer 3 - W 3 (400, 10) b 3 (1, 10)
Total number of parameters: 596910

2020-02-08 17:51:21.324762 - Epoch 1 : loss = 1.9345233573819667
2020-02-08 17:51:34.157486 - Epoch 2 : loss = 1.568176365112443
2020-02-08 17:51:46.978180 - Epoch 3 : loss = 1.2365064002272124
2020-02-08 17:51:59.609519 - Epoch 4 : loss = 0.9936469375559156
2020-02-08 17:52:12.223193 - Epoch 5 : loss = 0.8320996049241498
2020-02-08 17:52:25.168385 - Epoch 6 : loss = 0.7241287333693052
2020-02-08 17:52:37.606199 - Epoch 7 : loss = 0.6487440543347597
2020-02-08 17:52:50.466452 - Epoch 8 : loss = 0.593578051134673
2020-02-08 17:53:02.853626 - Epoch 9 : loss = 0.5515687221224399
2020-02-08 17:53:15.542823 - Epoch 10 : loss = 0.5185363217750453
glorot initialization
Layer 1 - W 1 (784, 500)        b 1 (1, 500)
Layer 2 - W 2 (500, 400)        b 2 (1, 400)
Layer 3 - W 3 (400, 10) b 3 (1, 10)
Total number of parameters: 596910
```

2. Compare the three setups by plotting the losses against the training time (epoch) and comment on the result.

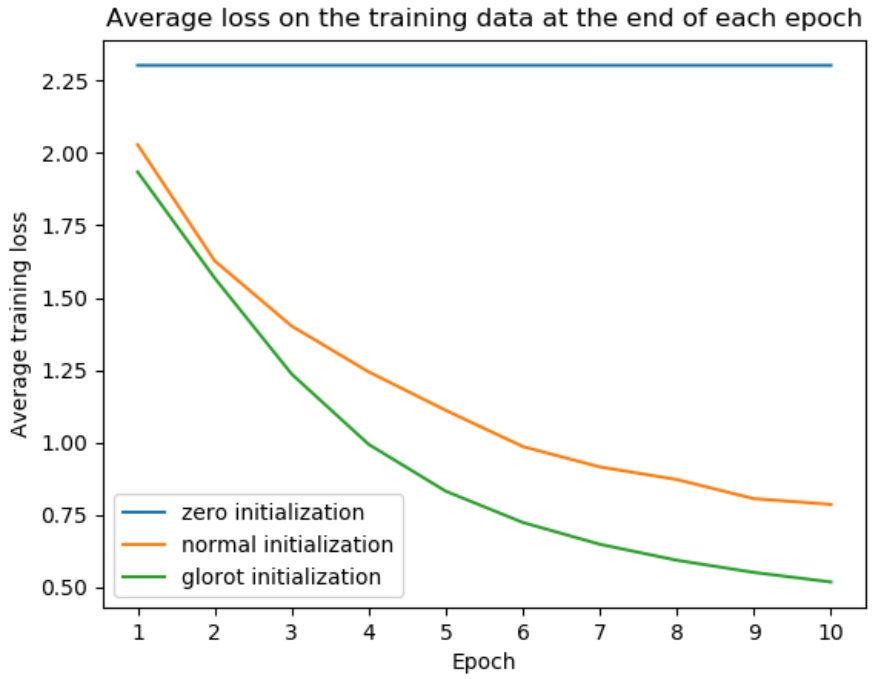


Figure 1: Average training loss for each initialization method

We can observe that the average training loss stays constant for the zero initialization method. Indeed, this initialization method doesn't lead to any update of the weights as all of the gradients are 0 and the model can't learn from backpropagation.

While the normal initialization method can lead to the problem of vanishing and exploding gradients, the fact that the ReLU activation function was used partly addresses this problem as the gradients are less likely to saturate. We can see that as the training proceeds, the loss decreases rather quickly.

The Glorot weight initialization method ensures that the variance of the activations is approximately the same across all the layers of the network, which allows for information from each training instance to pass through the network smoothly. Similarly, considering the backward pass, relatively similar variances of the gradients allows information to flow smoothly backwards.[2] This ensures that we do not encounter the problem of vanishing and exploding gradients, thus making our model a better learner. From the graph we can see that the Glorot initialization method lead to a better training loss quicker than the other initialization methods.

Hyperparameter Search (report) [10] From now on, use the Glorot initialization method.

1. Find out a combination of hyper-parameters (model architecture, learning rate, nonlinearity, etc.) such that the average accuracy rate on the validation set ($r^{(valid)}$) is at least 97%.

[2] https://mnsgrg.com/2017/12/21/xavier-initialization/

2. Report the hyper-parameters you tried and the corresponding $r^{(valid)}$.

For the hyperparameter search, a combinations of 72 hyperparameters respecting the range of [0.5M to 1M] parameters were tested. The fixed hyperparameters were the ReLU activation function, the Glorot initialization method, and the number of epochs (15).

Hyperparameters tested			
Learning rate	mini-batch size	number of neurons in 1st hidden layer	number of neurons in 2nd hidden layer
0.1	16	400	400
0.01	32	500	500
0.001	64	600	600

Out of 72 combinations, 17 of them achieved at least 98% accuracy on the validation set. Here are the 10 first combinations tested with their respective accuracy on the validation set (the list contains [learning rate, mini-batch size, # neurons in the 1st hidden layer, # neurons in the 2nd hidden layer]).

```
combination : [0.1, 16, 400, 500, 'relu'] validation accuracy 0.976
combination : [0.1, 16, 400, 600, 'relu'] validation accuracy 0.9834
combination : [0.1, 16, 500, 400, 'relu'] validation accuracy 0.9835
combination : [0.1, 16, 500, 500, 'relu'] validation accuracy 0.9821
combination : [0.1, 16, 500, 600, 'relu'] validation accuracy 0.9792
combination : [0.1, 16, 600, 400, 'relu'] validation accuracy 0.9842
combination : [0.1, 16, 600, 500, 'relu'] validation accuracy 0.9821
combination : [0.1, 16, 600, 600, 'relu'] validation accuracy 0.981
combination : [0.1, 32, 400, 500, 'relu'] validation accuracy 0.9826
combination : [0.1, 32, 400, 600, 'relu'] validation accuracy 0.9814
```

The best combination of hyperparameters found was an architecture with a learning rate of 0.1, mini-batch size of 16, 600 neurons in the first layer, and 400 neurons in the second layer. The model has a total of 715 4010 learnables parameters. This model architecture achieved a validation accuracy of 98.42%.

The learning rate may be the most important hyperparameter to tune as it controls how much we are adjusting the weights of our neural network with respect to the gradient of the loss. In fact, a learning rate that is too large will cause the neural network to converge to a sub-optimal solution too quickly, while a too small learning rate will cause the training process to get stuck. With a learning rate of 0.1, the neural network seems to have found one that is neither too small nor too big.

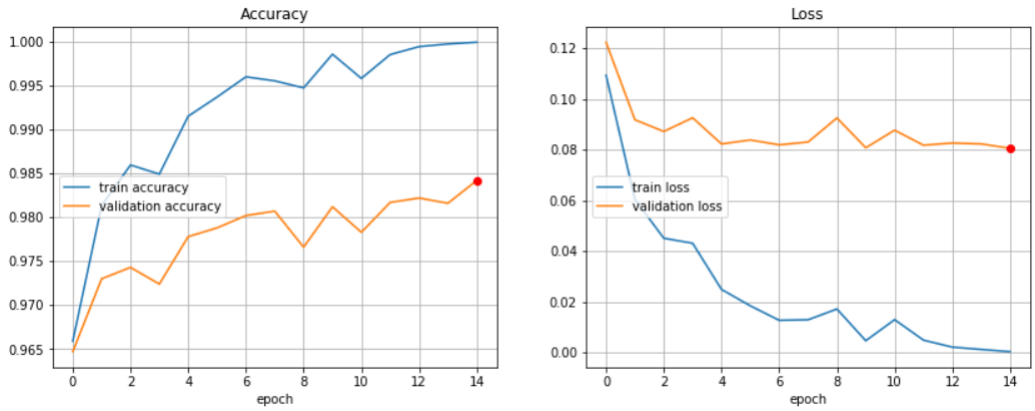


Figure 2: Accuracy and loss of the best neural network architecture found

Validate Gradients using Finite Difference (report) [15] The finite difference gradient approximation of a scalar function $x \in \mathbb{R} \mapsto f(x) \in \mathbb{R}$, of precision ϵ, is defined as $\frac{f(x+\epsilon)-f(x-\epsilon)}{2\epsilon}$. Consider the second layer weights of the MLP you built in the previous section, as a vector $\theta = (\theta_1, \ldots, \theta_m)$. We are interested in approximating the gradient of the loss function L, evaluated using **one** training sample, at the end of training, with respect to $\theta_{1:p}$, the first $p = \min(10, m)$ elements of θ, using finite differences.

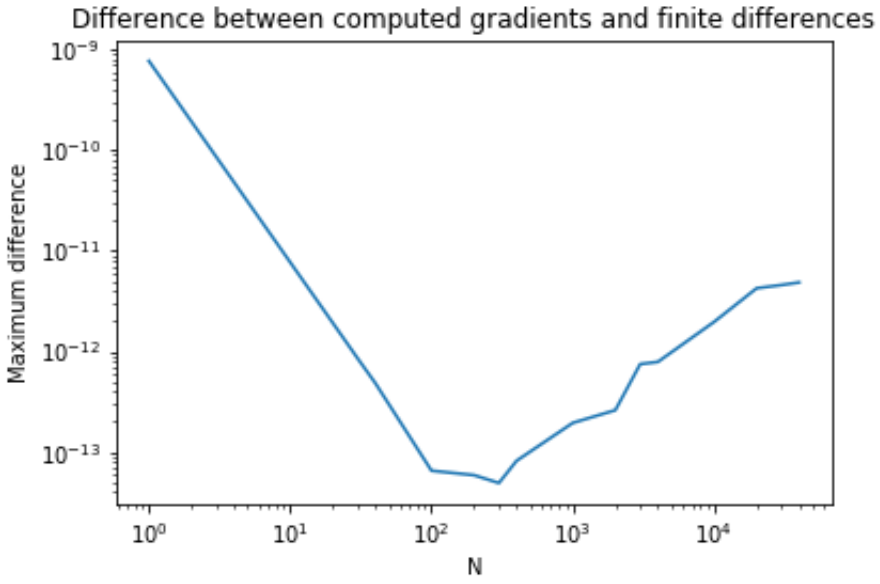
1. Evaluate the finite difference gradients $\nabla^N \in \mathbb{R}^p$ using $\epsilon = \frac{1}{N}$ for different values of N

$$\nabla_i^N = \frac{L(\theta_1, \ldots, \theta_{i-1}, \theta_i + \epsilon, \theta_{i+1}, \ldots, \theta_p) - L(\theta_1, \ldots, \theta_{i-1}, \theta_i - \epsilon, \theta_{i+1}, \ldots, \theta_p)}{2\epsilon}$$

   Use at least 5 values of N from the set $\{k10^i : \ i \in \{0, \ldots, 5\}, \ k \in \{1, 5\}\}$.

   The difference was calculated using $\max_{1 \leq i \leq p} |\nabla_i^N - \frac{\partial L}{\partial \theta_i}|$. This was done by implementing the finite difference method in the NN class.

2. Plot the maximum difference between the true gradient and the finite difference gradient ($\max_{1 \leq i \leq p} |\nabla_i^N - \frac{\partial L}{\partial \theta_i}|$) as a function of N. Comment on the result.



The curve of the maximum difference of the gradients of N is shown above. As N increases, the ϵ decreases up to a certain point, which leads to a smaller difference between the gradients calculated by the backward method and those calculated by the finite difference method. The increase in the curve might be explained by the fact that we become numerically unstable for small values of ϵ.

The minimal difference between the estimated gradients and the real ones are when $\epsilon = 0.003$.

Convolutional Neural Networks (report) [25] Many techniques correspond to incorporating certain prior knowledge of the structure of the data into the parameterization of the model. Convolution operation, for example, was originally designed for visual imagery. For this part of the assignment we will train a convolutional network on MNIST for 10 epochs using PyTorch. Plot the train and valid errors at the end of each epoch for the model.

1. Come up with a CNN architecture with more or less similar number of parameters as MLP trained in Problem 1 and describe it.

   The model architecture that was chosen is the following;

   - Convolutional layer 1 : 20 kernels, kernel size = 5, stride = 1
     Output shape : 20x12x12
     Number of learnable parameters : 20x5x5 + 20 = 520

   - Max pooling layer : kernel size = 2, stride = 2
     Output shape : 20x12x12

   - Convolutional layer 2 : 30 kernels, kernel size = 4, stride = 1
     Output shape : 30x9x9
     Number of learnable parameters : 20x4x4x30 + 30 = 9630

- Convolutional layer 3 : 40 kernels, kernel size = 3, stride = 1
  Output shape : 40x7x7
  Number of learnable parameters : 30x3x3x40 + 40 = 10840

- Max pooling layer : kernel size = 2, stride = 2
  Output shape : 40x3x3

- Fully connected layer 1 : number of input features = 40x3x3, number of output features = 500
  Number of learnable parameters : 40x3x3x500 + 500 = 180500

- Fully connected layer 2 : number of input features = 500, number of output features = 10
  Number of learnable parameters : 500x10 + 10 = 5010

In total, there are 206 500 learnable parameters to this convolutional neural network.

2. Compare the performances of CNN vs MLP. Plot the training loss and validation loss curve.

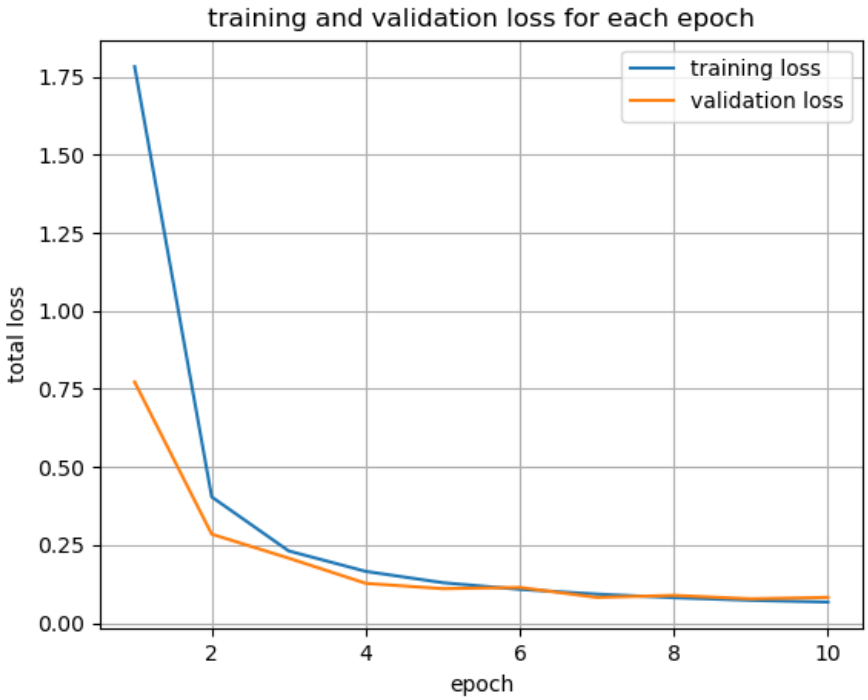


Figure 3: Training and validation loss over the number of epochs

For the same number of epoch, we can see that the vanilla CNN achieves a smaller training loss than the vanilla MLP with the Glorot initialization method (see figure 1). Also, the vanilla CNN has a bit smaller validation loss than the best performing multilayer perceptron with hyperparameter search. Indeed, the validation loss at 10 epochs for the best performing MLP is around 9% while the validation loss of the vanilla CNN is around 8%. Both models have an

accuracy of around 98% on the validation set. Even if both of them have similar validation accuracy, the CNN achieves this feat with a smaller number of learnables parameters (206 500 for the CNN and 715 410 for the MLP) and within smaller epochs (10 for the CNN and 15 for the MLP).

Even if both models achieve a high performance on the validation set, CNNs are more adapted to the task of image classification for the following reasons;

- Local connectivity : When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume. Instead, each neuron is connected to only a local region of the input volume. By doing so, we highly reduce the number of learnable parameters.

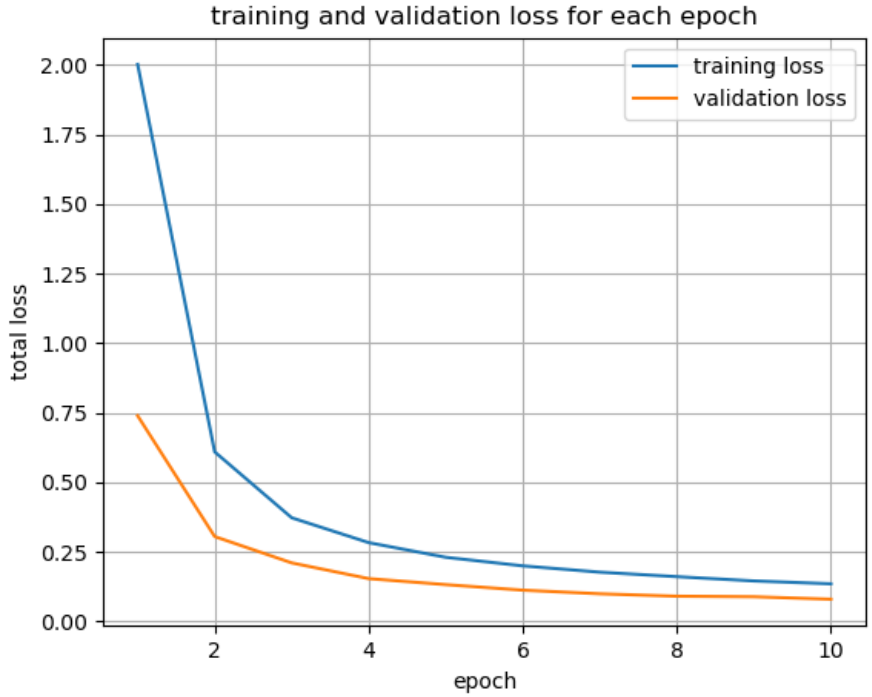
- Parameter sharing : Each member of the kernel is used at every position of the input. This means that rather than learning a separate set of parameters for every location, we learn only one set. Again, this idea greatly reduces the number of learnables parameters compared to a multi-layer perceptron. It also enables us to extract the same features at every position (the assumption that if one feature is useful to compute at some spatial position, it might also be useful to compute at a different position).

- Pooling : 2 layers of max pooling were used. It consists of taking the maximum value within a region (in general non-overlapping). By doing so, we end up with a small number of hidden units in the hidden layers and it introduces some local invariance to translation. This is a way of encoding our prior knowledge about images that is not present in MLPs.

3. Explore *one* regularization technique you've seen in class (e.g. early stopping, weight decay, dropout, noise injection, etc.), and compare the performance with a vanila CNN model without regularization.

You could take inspiration from the architecture mentioned here (hyperlink).

I explored the dropout regularization technique. Dropout consists of randomly dropping some number of neurons during training. It has the effect of making the training process noisy, forcing neurons within a layer to probabilistically take on more or less responsibility for the inputs. I used dropout after the 3rd convolutional layer and after the first fully-connected layer.



As we can see, the validation loss for the regularized CNN looks quite similar to the vanilla CNN (see figure 3). In fact, the validation loss for the regularized CNN sits at 7.4%, while the vanilla CNN one sits at around 8%. Also, both model achieve an accuracy of 98% on the validation As this paper [3] suggests, dropout is less effective for convolutional layers. The authors argue that for convolutional layers, features are correlated spacially. Thus, even with dropout information about the input can still be sent to the next layer, which can cause the network to overfit. They then suggest DropBlock, where features in a block, i.e, contiguous region of a feature map are dropped together instead of being dropped randomly across the feature map.

[3] https://papers.nips.cc/paper/8271-dropblock-a-regularization-method-for-convolutional-networks.pdf