CSE 151B/251B Deep Learning
UC San Diego, Spring 2025

Homework 2

Name: Ouyang Yingxuan UID: 10486415

1 Problem 1 Automatic Differentiation

Problem A: Backpropagation and Weight Initialization

i.

Solution A.i: *Both networks have identical architectures with ReLU activation functions, but differ in how their weights are initialized.*

Network A in the first link uses a proper initialization scheme, but network B in the second link uses a poor initialization scheme (weights initialized to very small or zero values).

The ReLU activation function is defined as:

$$\mathit{ReLU}(x) = \max(0, x)$$

It outputs zero for all negative inputs. During backpropagation, this causes the derivative to be zero, and thus the neuron cannot learn anything.

Because many initial weights in a layer are negative or too small, the ReLU units in network B never activate, leading to zero gradients and stagnation during training.

However, with a good initialization strategy, weights are scaled in a way that maintains variance throughout layers. This increases the chance that ReLU units are activated and thus contribute useful gradients during training.

Therefore, after around 250 iterations, network A with better weight initialization performs output with test loss of 0.001 and training loss of 0.001, significantly better than network B output test loss of 0.508 and training loss of 0.497, because network A avoids the vanishing gradient problem and allows faster convergence. It shows that proper initialization is critical in ReLU-based networks to avoid inactive neurons and ensure effective learning.

ii.

Solution A.ii: *We can compare the training behavior of two neural networks with the same architecture but different weight initializations, using sigmoid activation functions instead of ReLU. After resetting and training each model for 4000+ iterations, we observed the following:*

Network A *converged rapidly, with the loss dropping significantly around epoch 300 and reaching very low values (test loss 0.001, training loss 0.001) by epoch 4000.*

Network B *in contrast, showed almost no learning until after epoch 3500, and even by epoch 4083, the loss remained relatively high (test loss 0.412, training loss 0.396).*

The key difference lies in how the sigmoid activation function behaves during backpropagation. The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Its derivative is:

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

This derivative becomes very small when x is either very large or very small, leading to the problem of ***vanishing gradient****. When initial weights are poorly scaled, the activations $\sigma(x)$ saturate near 0 or 1, and their derivatives approach 0, making weight updates very slow or nonexistent.*

Network A *likely used an initialization scheme that kept weights in a range where the sigmoid derivatives remain large, allowing effective learning from the beginning.*

Network B*, however, likely had initial weights that pushed activations into saturated zones of the sigmoid, resulting in very small gradients and delayed learning. Only after thousands of epochs did the network begin to escape this saturation and make progress.*

The experiment show that when using sigmoid activations, proper weight initialization is even more critical than with ReLU, for the risk of vanishing gradients that can stall training.

Problem B: Data Shuffling

Solution B:

When training a fully connected neural network with ReLU activation using stochastic gradient descent (SGD), it is crucial to shuffle the training data. If we loop through all the negative examples (label -1) before the positive ones (label +1), we risk encountering the "dying ReLU" problem.

$$\textit{ReLU}(x) = \max(0, x)$$

Its derivative is 0 for all inputs $x < 0$ and 1 for $x > 0$.

During SGD, the weights are updated based on the current mini-batch. If all early training examples produce large negative inputs to ReLU units like all label -1 samples, then those neurons output 0, and their gradients are also 0. These neurons stop updating as derivative of ReLU is 0 for negative x.

Therefore, by presenting only negative examples early, we may cause a large portion of the network to become inactive before it even sees any positive examples. The model cannot learn the full distribution of the data.

Presenting highly correlated or non-shuffled data, such as all negative examples first, can cause many ReLU units to permanently die during early training, leading to poor performance. It is necessary to shuffling data and avoid this issue by mixing input statistics during training.

2 Problem 2 Convolutional Neural Networks

Problem A: Convolution

i. Number of Parameters

Solution A.i:

Each filter has size $5 \times 5 \times 3 = 75$ weights.

There are 16 filters, the total number of weights is:

$$75 \times 16 = 1200$$

Each filter also has a bias term, so the parameters need to add 16 bias terms:

$$1200 + 16 = 1216 \textit{ parameters}$$

ii. Shape of the Output Tensor

Solution A.ii:

Input size: $32 \times 32 \times 3$

Filter size: $5 \times 5 \times 3$ *Stride = 1, Padding = 0.*

For output dimensions:

$$\text{Output width/height} = \left\lfloor \frac{32-5}{1} \right\rfloor + 1 = 28$$

Number of filters = 16, so output depth = 16.

So, the output shape is:

$$\boxed{28 \times 28 \times 16}$$

Problem B: Paddling

Solution B:

Benefit: *Zero-padding allows the convolution to preserve spatial dimensions and enable filters to process edge pixels. Then the network will be able to learn features at the borders of the image, which would otherwise be ignored if only non-padded convolutions were used.*

Drawback: *Padding introduces artificial values (zeros) that do not exist in the original image, which leads to artifacts or distortions near the image boundaries and less reliable feature learning at the edges.*

Problem C: Pooling

i. Apply 2×2 average pooling with stride 2

Solution C.i:

For each patch, we compute the average of the four values.

Matrix 1:

$$\begin{bmatrix} \frac{1+1+1+1}{4} & \frac{1+0+1+0}{4} \\ \frac{1+1+0+0}{4} & \frac{0+1+0+0}{4} \end{bmatrix} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 0.25 \end{bmatrix}$$

Matrix 2:

$$\begin{bmatrix} \frac{0+1+0+1}{4} & \frac{1+1+1+1}{4} \\ \frac{0+1+0+0}{4} & \frac{1+1+0+0}{4} \end{bmatrix} = \begin{bmatrix} 0.5 & 1 \\ 0.25 & 0.5 \end{bmatrix}$$

Matrix 3:

$$\begin{bmatrix} \frac{0+0+0+1}{4} & \frac{0+0+1+1}{4} \\ \frac{0+1+0+1}{4} & \frac{1+1+1+1}{4} \end{bmatrix} = \begin{bmatrix} 0.25 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

Matrix 4:

$$\begin{bmatrix} \frac{0+0+1+1}{4} & \frac{0+0+1+0}{4} \\ \frac{1+1+1+1}{4} & \frac{1+1+0+0}{4} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.25 \\ 1 & 0.5 \end{bmatrix}$$

ii. Apply 2×2 max pooling with stride 2

Solution C.ii:

For each patch, we compute the maximum of the four values.

Matrix 1:

$$\begin{bmatrix} \max(1,1,1,1) & \max(1,0,1,0) \\ \max(1,1,0,0) & \max(1,0,0,0) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Matrix 2:

$$\begin{bmatrix} \max(0,1,0,1) & \max(1,1,1,1) \\ \max(0,1,0,0) & \max(1,1,0,0) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Matrix 3:

$$\begin{bmatrix} \max(0,0,0,1) & \max(0,0,1,1) \\ \max(0,1,0,1) & \max(1,1,1,1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Matrix 4:

$$\begin{bmatrix} \max(0,0,1,1) & \max(0,0,1,0) \\ \max(1,1,1,1) & \max(1,0,1,0) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Problem D: Pytorch implementation

i. Training Results

Solution D.i: *We trained 10 models using dropout probabilities* $p = 0.0, 0.1, \ldots, 0.9$ *on the MNIST dataset for 1 epoch each. The final test accuracies observed were:*

- $p = 0.0$: *0.7846*
- $p = 0.1$: *0.6937*
- $p = 0.2$: *0.1009*
- $p = 0.3$: *0.9744*
- $p = 0.4$: *0.8720*
- $p = 0.5$: *0.9763*
- $p = 0.6$: *0.9704*
- $p = 0.7$: *0.9570*
- $p = 0.8$: *0.9297*
- $p = 0.9$: *0.7302*

The best performance was observed at $p = 0.5$*, and we used this setting to train the final model for 10 epochs. The final test accuracy achieved was* ***0.9763****.*

ii. Effective strategies for CNN

Solution D.ii:

The most effective strategies included:

Using multiple convolutional layers with small kernel sizes to extract hierarchical features.

Applying Batch Normalization after each convolution to stabilize training and accelerate convergence. Using Dropout with $p = 0.5$ *as the optimal regularization strength; it balanced generalization and convergence well.*

Among regularization methods, ***Batch Normalization combined with Dropout*** *was the most effective for achieving high accuracy while avoiding overfitting.*

iii. Issues with hyperparameter validation

Solution D.iii:

One problem is that when validating dropout probabilities using only 1 epoch of training, it may not reflect long-term generalization performance. Some models converge more slowly and might appear worse in early epochs. Performing hyperparameter selection on the test set leads to data leakage and overestimates true generalization. A better strategy is to use a validation set which is split from the training data.