ELEC 576 Assignment 1

Tsung-Yu Liu

[tsung-yu.liu@rice.edu](mailto:tsung-yu.liu@rice.edu)

1. Backpropagation in a simple Network
2. Dataset  
   一張含有 螢幕擷取畫面, 文字, 鮮豔 的圖片

   自動產生的描述

actFun\_type='tanh':  
一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

actFun\_type='sigmoid'

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

actFun\_type='relu'

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

**tanh**:

* + Produces a **smooth decision boundary**.
  + Best suited for this dataset due to its ability to capture non-linearity.
  + Outputs between -1 and 1, which helps the model learn more flexible patterns.

**sigmoid**:

* + Results in a **rigid and less accurate boundary**.
  + Struggles with non-linear data, due to potential **vanishing gradients** and limited output range (0 to 1).

**relu**:

* + Provides a **sharper, angular boundary**.
  + Efficient and faster to converge, making it ideal for deeper networks, though it may introduce **dead neurons** (neurons that stop learning).

I found the difference between tanh and sigmoid visually quite similar. Later I will increase the hidden units to make it more obvious

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述If we increase the hidden units (nn\_hidden\_dim) and retrain the network using Tanh as the activation function,

一張含有 文字, 螢幕擷取畫面, 軟體, 設計 的圖片

自動產生的描述Show hidden units: on the left = 3, show hidden units on the right = 7

Obviosely, with greater hidden units, we can have more complex decision Boundary, which means that gives model more capabilities to learn complex patterns. And it can potentially fit better to the data, especially when the original number of hidden units was insufficient. However, I also know that this will potentially increase the risk of overfitting, so it is important to analyze the result to see whether you have to increase the hidden units.   
What’s more is that with higher hidden units, absolutely you will need higher computational cost as well.

When increasing hidden units to 10, the difference becomes obvious

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述Tanh:

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述Sigmoid:

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述Relu:

Compare these three pic to those hidden layer ==3, we can tell the difference

Training a Simple Deep Convolutional Network on MNIST

1. **Build and Train a 5-layer DCN**

一張含有 耶誕樹, 螢幕擷取畫面, 設計 的圖片

自動產生的描述

一張含有 螢幕擷取畫面, 文字, 繪圖軟體, 多媒體軟體 的圖片

自動產生的描述

The graphs are showing how the distribution of weights and biases in both first and second convolutional layer is evolving over time (or epochs). We can see a histogram distribution for both the weights and the biases.

一張含有 螢幕擷取畫面, 繪圖軟體, 多媒體軟體, 文字 的圖片

自動產生的描述

一張含有 螢幕擷取畫面, 文字, 耶誕樹, 繪圖軟體 的圖片

自動產生的描述

Layer Function:

* fc1: As the first fully connected layer, fc1 is responsible for learning intermediate representations of the input data (features learned from the convolutional layers).   
  The smooth weight distribution suggests that fc1 is learning and generalizing well.
* fc2: The final fully connected layer (fc2) is responsible for mapping the features learned by fc1 to the final output (10 classes for MNIST digits).   
  The slightly broader weight and bias distribution reflects the final adjustments needed to classify each input correctly.
* Weight stability:

Both fc1 and fc2 show stable and well-behaved weight distributions. The weights are centered around zero with no extreme values, indicating that your network’s training process is stable and the optimizer is working effectively.

* Bias Adjustment:

The biases in both fc1 and fc2 are small and appropriately centered around zero. The slightly wider range of biases in fc2 suggests that the network is making more adjustments in the output layer, which is normal for a classification task like MNIST. This layer is responsible for producing the final class probabilities, so it makes sense for the biases to be more spread out as the network fine-tunes the final decision boundaries.

* Test accuracy

One thing interesting is that I found that with higher epoch, we can have lower accuracy, which is what I was not expected. But after doing some research, I find out this is normal simply due to overfitting or facing instability during training. Here is the chart of accuracy

一張含有 文字, 螢幕擷取畫面, 圖表, 繪圖 的圖片

自動產生的描述

1. **More on Visualizing my Training**

I implemented a simple convolutional neural network (CNN) consisting of two convolutional layers followed by fully connected layer. Below I describe the architecture  
Model Design:

* Conv1: 10 filters, kernel size = 5
* Max Pooling: kernel size = 2
* Conv2: 20 filters, kernel size = 5
* Dropout Layer: Applied after Conv2
* Fully Connected Layer 1 (fc1): 50 units
* Fully Connected Layer 2 (fc2): 10 units (output layer with softmax activation)

I used the ReLU activation function after each convolution and fully connected layer to introduce non-linearity. The dropout layer was used to reduce overfitting.

I trained the network for 10 epochs with the following step

**Training Loop**:

* The model was trained using Cross-Entropy Loss, optimized with the Adam optimizer (learning rate: 0.01).
* After every batch, the training loss was logged in TensorBoard.
* Histograms for weights, biases, net inputs, and activations were recorded at each layer.
* Min, max, mean, and standard deviation for weights and biases were tracked.

**Testing Loop**:

* After each epoch, the model was evaluated on the test set, and the test accuracy and test loss were logged.
* Similar to the training process, we recorded histograms for the test set activations and net inputs.

The result Analysis show in next page

一張含有 文字, 功能表, 螢幕擷取畫面, 圖書 的圖片

自動產生的描述一張含有 文字, 功能表, 螢幕擷取畫面, 字型 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 功能表, 字型 的圖片

自動產生的描述一張含有 文字, 功能表, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Initial Result

With first attempt, initial training, I observed the following:

* Training Loss: The training loss started at 2.30 and showed a gradual decrease with each epoch. However, it fluctuated within a range of 1.8 to 2.3, indicating that the learning process was not fully working.
* Test Accuracy: The test accuracy began at 64% after the first epoch but exhibited high variability across subsequent epochs. It fluctuated between 31% and 55% across the training process, signaling that the model may not be generalizing well.
* Test Loss: The test loss consistently remained around 0.0018 to 0.0023, which might indicate the model is fitting the training data but not effectively learning the features required for accurate predictions on the test set.

Key Observation:

* **Overfitting**: While the test loss remained low, the fluctuations in accuracy across epochs suggest the model might be overfitting to the training data. This is indicated by the accuracy oscillating around 40%–55%.
* **Learning Rate**: The learning rate of 0.01 may be too high for this model, causing the weights to update in a manner that is not allowing for effective learning.
* **Model Complexity**: The current model, while simple, may not have enough capacity to fully capture the complexities in the dataset, leading to the test accuracy plateauing around 55%.

Therefore, I want to fix the issues identified

1. Lowering the Learning Rate: I plan to reduce the learning rate from 0.01 to 0.001 to ensure more stable updates to the weights. This should help the model converge better and reduce the fluctuations in test accuracy.
2. Regularization Techniques: To further address the overfitting issue, I will introduce weight decay (L2 regularization) to the optimizer. This should help prevent the model from overfitting to the training data by penalizing large weights.
3. Increasing Model Complexity: To improve the model’s capacity, I will consider adding an additional fully connected layer or increasing the number of units in the fully connected layers. This should allow the model to better learn the features in the MNIST dataset.
4. Monitoring Metrics: I will continue to monitor the histograms and scalar metrics in TensorBoard for further insights into the learning process

Then I face again warning: Due to Colab's usage limits, you cannot connect to the GPU right now. So I try another day