CS795/895: Fundamentals of Deep Learning (Spring 2024)
Homework Assignment 3
Name: Sushmitha Halli Sudhakara

1 Problem Description

In this assignment, you will implement a multilayer perceptron, with two hidden layers. A multilayer perceptron is a feed-forward neural network architecture composed of multiple layers of stacked perceptron units. See Figure 1.

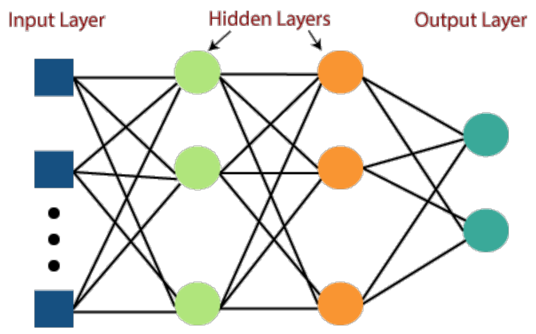

Figure 1: A schematic illustration of a multilayer perceptron.

1.1 Forward Pass

In the forward pass, we compute the activation of all the neurons as:

$$a_j^{(l+1)} = \phi\left(\sum_i w_{ij}^{(l+1)} a_i^{(l)} + b_j^{(l+1)}\right)$$

here, $a^{(l)}$ is the activation of layer l.

For $l = 0$, activations $\left(a^{(0)}\right)$ are the inputs.

$w_{ij}^{(l)}$ is he weight from the i-th neuron in Layer $(l-1)$ to the j-th neuron in Layer l.

ϕ is the non-linear activation function.

In matrix form, it can be written as

$$\mathbf{a}^{(l+1)} = \phi(W^T \mathbf{a}^{(l)} + \mathbf{b}^{(l+1)})$$

Note: To better understand each module, try to divide them into pre and post-activation. To get post-activations, we can use various non-linear activation functions such as Elu, Selu, and LeakyRelu. In this exercise you will use Relu as non-linear activation, thus ϕ can be represented as follows.

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

Note: The neural network will output a set of probabilities or the confidence for each class. One can further explore the importance of each class and create a weighted version of softmax. Remember: Everything should be differentiable when the model is trained using back-propagation, when dealing with the binary classification, sigmoid is the default choice, and in scenarios when you have multi-class you will go with softmax.

Therefore for the the last layer, which is also known as the output layer, the nonlinear function depends on the task at hand. For binary classification, the logistic function σ is applied:

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

For multiclass classification, the Softmax function is used:

$$\phi(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

1.2 Loss Function

Loss function provides us with a measure of the difference between the predicted output of the neural network and the actual output. Loss function is derived by minimizing the negative log-likelihood of the output given the input. The cost function is parameterized by the weights the of neural network. For classification tasks, the function evaluates to cross-entropy:

$$L(W) = -\sum_j y_j \log(h_j)$$

where y_j are the components of vectorized ground truth, and h_j are the corresponding outputs from the neural network.

Similarly, for regression tasks, the cost function evaluates to mean squared error (MSE)

$$L(W) = \frac{1}{n} \sum_j^n (y_j - h_j)^2$$

1.3 Backward Pass

In the backward pass, our aim is to calculate the gradients of the cost function with respect to all the parameters of the neural network.

Consider the forward pass equations:

$$z_j^{(l+1)} = \sum_i w_{ij}^{(l+1)} a_i^{(l)} + b_j^{(l+1)}$$

Taking partial derivative, we get

$$\frac{\partial z_j^{(l+1)}}{\partial w_{ij}^{(l+1)}} = a_i^{(l)}$$

and,

$$a_j^{(l+1)} = \phi\left(z_j^{(l+1)}\right)$$

$$\frac{\partial a_j^{(l+1)}}{\partial w_{ij}^{(l+1)}} = \frac{\partial \phi}{\partial z_j^{(l+1)}} \frac{\partial z_j^{l+1}}{\partial w_{ij}^{l+1}} = a_i^{(l)} \frac{\partial \phi}{\partial z_j^{(l+1)}}$$

This allows us to calculate the partial derivatives with respect to parameters using chain rule across the layers of the neural network.

2 Training

A training cycle of a neural network involves a forward pass, a backward pass, and a parameter update. The parameters are updated in each iteration as follows:

$$\hat{w}_{ij}^{(l)} = w_{ij}^{(l)} - \alpha \frac{\partial L(W)}{\partial w_{ij}^{(l)}}$$

This is called the Gradient Descent Algorithm as the parameters are updated in the direction maximum negative slope in the loss function.

2.1 Stochastic Gradient Descent

In a traditional Gradient Descent Algorithm, the entire train set is first passed through a forward pass and backward pass, and then parameters are updated using gradients aggregated over the whole train set. This becomes impossible with very large datasets due to memory constraints. Hence we use Stochastic Gradient Descent. Here we divide the training set into smaller batches of data and parameter update is performed for each batch. This does not necessarily follow the most optimum path for optimization but eventually converges. It sometimes provides better results than Gradient Descent as noise introduced due to mini-batches provides an alternate zic-zac path around the global gradient and sometimes avoids problems with diminishing global gradients.

2.2 Early Stopping

Ideally, the training continues until the network converges. In practice, we define a small value ϵ such that, if the loss function does not reduce at least ϵ in consecutive iterations, then we stop the training procedure and consider that the training has converged.

$$L(W)_t - L(W)_{t+1} < \epsilon$$

3 Regularization

Regularization is a way of penalizing the training of neural network model such that we do not end up with extreme parameter values. When the parameters of a neural network take extreme values, the model usually overfits to a certain set of inputs and does not generalize to cover the domain. Two most common types of penalty terms that can be added to cost function are L1 and L2 penalty.

$$L1 := \frac{1}{n}\sum_{i,j,l}^{n} |w_{ij}^{(l)}|$$

$$L2 := \frac{1}{n}\sum_{i,j,l}^{n} \left(w_{ij}^{(l)}\right)^2$$

This adds the following derivatives to the gradients:

$$\frac{\partial L1}{\partial w_{ij}^{(l)}} = \frac{1}{n}\frac{w_{ij}^{(l)}}{|w_{ij}^{(l)}|}$$

$$\frac{L2}{\partial w_{ij}^{(l)}} = \frac{2w_{ij}^{(l)}}{n}$$

L1 penalty makes the parameters sparse as it forces some parameters to take very small values and only allows few others to survive. It makes the model simpler by removing unwanted parameters. L2 penalty stops parameters from taking very large values. Penalty terms are added to cost function during training.

3.1 Dropout

Dropout is another method of regularizing a model. Instead of adding penalty term to cost function, we directly manipulate the parameter values. At each training step, we choose a fraction of parameters from each layer and set them to zero. This regulates the contribution of each neuron in deciding the final output. The forward equations can be updated as:

$$m_{ij}^{(l+1)} \sim Bernoulli(p)$$

$$a_j^{(l+1)} = \phi\left(\sum_j m_{ij}^{(l+1)} w_{ij}^{(l+1)} a_i^{(l)} + b_j^{(l+1)}\right)$$

which updates the derivative calculation as:

$$\frac{\partial z_j^{(l+1)}}{\partial w_{ij}^{l+1}} = m_{ij}^{(l+1)} a_i^{(l)}$$

which implies

$$\frac{\partial a_j^{(l+1)}}{\partial w_{ij}^{(l+1)}} = m_{ij}^{(l+1)} a_i^{(l)} \frac{\partial \phi}{\partial z_j^{(l+1)}}$$

Hence, if a neuron is turned off, all gradients passing through that neuron are also set to 0 while training.

3.2 Normalization

If different input features have values at different scales, e.g. feature 1 has values between 0 and 1 while feature 2 has values between 100 and 10000, then it is good idea to normalize features such that they have 0 mean and unit variance.

$$\mu_i = \frac{1}{m} \sum_{k=0}^{m} x_i^{(k)}$$

$$\sigma_i^2 = \frac{1}{m} \sum_{k=0}^{m} \left(x_i^{(k)} - \mu_i \right)^2$$

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

This idea of normalizing data is sometimes extended to normalizing mini-batches rather than the entire train set. That is called Batch Normalization. Batch Normalization can however lead to instability while training.

Other types of normalization include normalizing the features at the output of each layer. That is called Layer Normalization.

Adding small Gaussian noise to gradients can be helpful in assisting training find new optimization path if it get stuck in a shallow local minima or is rolling slowly on path of low gradients. Small noise provides enough energy to explore the surrounding topology of the cost function.

$$\nabla_w \hat{L} = \nabla_w L + N(0, 0.1)$$

4 Bias-Variance Tradeoff

There is no such thing as perfect training. There is always a tradeoff between how general or how specific we need our neural network to be. If it learns all training examples by heart, but struggles on the test test, the model is said to have high variance. On the other hand if the model shows similar performance on train and test set, but is not good on both, then it is said to have high bias. Figure 2 and Figure 3 shows the behavior of models with high bias or high variance.

Our goal is to train a model such that it has acceptable and similar performance on train and test sets. It is said to have low bias and low variance.

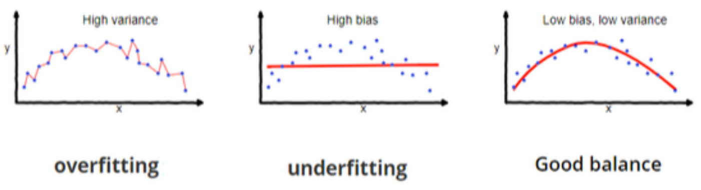

Figure 2: Model fitting.

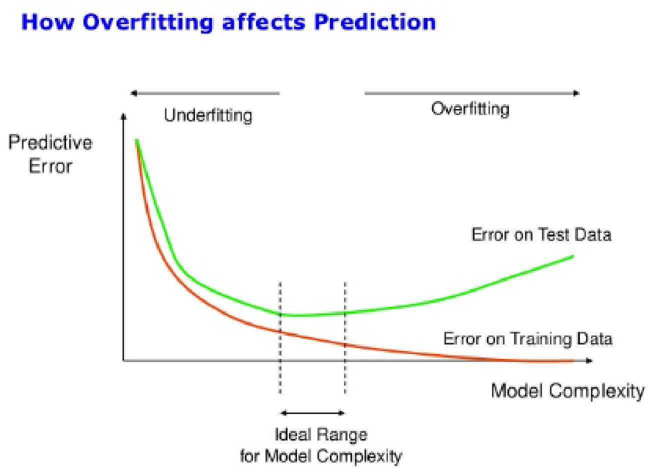

Figure 3: Loss characteristics of underfitting and overfitting model.

5 Things to do in this assignment

1. Design MLP with 2 hidden layers (Input Layer - 2 hidden Layer - Output layer) to classify objects (fashion MNIST) and digits (MNIST). The fashion MNIST dataset can be downloaded from `https://www.kaggle.com/datasets/zalando-research/fashionmnist`. The digits MNIST dataset can be downloaded from `https://www.kaggle.com/code/ngbolin/mnist-dataset-digit-recognizer`.

2. Design regularization approaches, and analyze drop/boost in performance of your model. Report results with at least 2 regularization variants (droput, L1 penalty, L2, L1+L2, Normalization, Noise). Do you see any tradeoff with respect to bias-variance? Report your findings. For more information on the bias-variance trade-off in machine learning, see `https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning`.

   When exploring different neural network setups for handwriting digit recognition, we observe the bias-variance tradeoff through various regularization strategies. Without regularization, models may overfit, seen in inconsistent accuracy across trials. Dropout regularization adds variability, potentially increasing variance. L2 regularization seems to mitigate overfitting with less added variance. Combining dropout and L2 doesn't consistently enhance outcomes, highlighting the complex interplay between regularization methods and the bias-variance tradeoff. This experiment underscores the need to carefully select and balance regularization techniques for optimal model performance.

3. How did you perform hyper-parameter optimization? Report your approach, and also show settings for the best model.

   For hyper-parameter optimization, various strategies were employed, focusing on dropout rates, regularization strength, learning rate adjustments, and early stopping criteria. The best model emerged from experimenting with different settings, particularly emphasizing dropout and L2 regularization to mitigate overfitting. This model underwent extensive training, with its effectiveness attributed to finely tuned hyperparameters such as a keep probability of 0.8, a lambda value of 0.0005, an initial learning rate of 0.10 adjusted as needed, and a calculated approach to early stopping based on performance, culminating in superior validation and test accuracies.

4. Report your results across multiple trials (minimum 10). Besides accuracy, you will also report the standard error and plot a graph showing variance (Tip: Change seeds and run your model (Do inference) K times).

Running Trials for Configuration

Without Regularization

- Reduced learning rate to 0.0010.
- Early stopping triggered at various epochs.
- Trial accuracies ranged from 20.00% to 81.85% for validation accuracy and similar ranges for test accuracy.
- The model's performance varied significantly, indicating the impact of hyperparameter tuning on the outcomes.

With Dropout Regularization

- Early stopping and learning rate adjustments were applied.
- Validation and test accuracies showed an improvement with dropout regularization, emphasizing the method's effectiveness in preventing overfitting.

With L2 Regularization

- L2 Regularization showed similar trends with early stopping and learning rate adjustments leading to varied accuracies.
- It highlights L2's role in controlling model complexity.

With Both Dropout and L2 Regularization

- Combining dropout and L2 regularization offered the best results, demonstrating the benefits of using both techniques together.
- The highest mean accuracies were observed with this configuration, showcasing its efficacy.

Analysis

The trials demonstrate the significant impact of regularization techniques on the neural network's performance, with the combination of dropout and L2 regularization yielding the best results. This suggests a balanced approach to managing overfitting and model complexity is crucial for optimal performance.

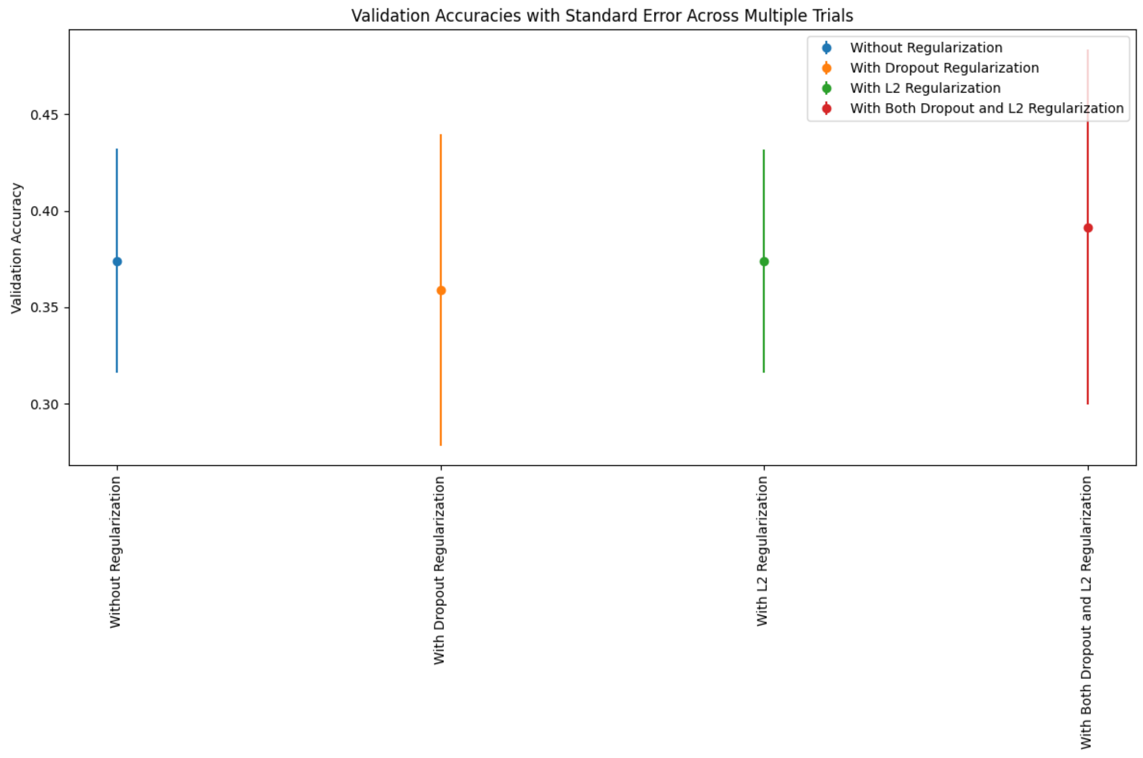

Figure 4: Enter Caption

5. Work on the problem on Google Colab and share it with `j1wu@odu.edu`. Submit a detailed report with graphs, findings, and tables in a single PDF document. Add the Google Colab link to your submission.

   Colab Notebook