Report: Activation Functions and Depth of ANN

Addis Ababa University

Addis Ababa Institute of Technology

School of Information Technology and Engineering

Course Title: Deep Learning (ITSC-1042)

Reported To: Fantahun Bogale(PhD)
Reported By: Mahlet Nigussie

MARCH 27, 2024

Addis Abeba,Ethiopi

INTRODUCTION

This report explores the performance of Deep Neural Networks (DNNs) with different activation functions, focusing on the vanishing gradient problem and ReLU variants. And the performances of ANNs with varying depths using ReLU activation for all layers. MNIST is used, a built-in dataset of handwritten digits often used to train and benchmark machine learning models. A simple DNN architecture is implemented and compare the training behavior of models using sigmoid, tanh, and ReLU activations. Additionally, Investigates the performance differences between ReLU variants like Leaky ReLU and Parametric ReLU.

MATERIALS

DATASET: MINST: A widely used dataset of handwritten digits (0-9) where each image is a 28x28 pixel grayscale image.

TOOLS USED:

NumPy (np): Provides efficient arrays for storing and manipulating the MNIST data.

Matplotlib (plt): Used for visualizing training and validation performance (loss, accuracy) to compare activation functions.

NetworkX (nx): Library for creating and studying complex networks (nodes, edges, analysis). It deals with network analysis.

TensorFlow.keras: The core library for building, training, and evaluating your DNN.

- *Sequential:* Defines the basic structure of the DNN with sequentially stacked layers.
- *Flatten:* Reshapes the flattened MNIST image data into a single vector for feeding into Dense layers.
- *Dense:* Creates fully connected layers, the workhorses of your DNN.
- *Activations (sigmoid, tanh, relu):* Experimenting with these activation functions is the core of your analysis.

- *Adam*: An optimization algorithm that efficiently updates the weights and biases in the DNN during training.
- *SparseCategoricalCrossentropy:* The loss function used to measure the difference between the model's predictions and the true digit labels.
- *mnist:* Loads the MNIST dataset for training and testing your DNN.

DNN ARCHITECTURE: Simple DNN architecture, The input layer size match the input data dimension. The output layer have the number of neurons corresponding to the number of classes in the dataset.

EXPERIMENTS

1. Experiment One: Using Simple DNN with two different Models to implement sigmoid, tanh and Relu.

   Model Architectures:

   Model 1: Multi-Layered Model , Simple Feed-Forward Neural Network

   Layers: [8, 9, 8, 10]
   Input layer: 8 neurons (assuming data has 8 features)
   Two hidden layers: 64 neurons each
   Output layer: 10 neurons (assuming 10 possible classes)
   Model 2: Single-Layered Model (Using create_model function)

   Simple Feed-Forward Neural Network (defined in create_model function)
   Layers: [784, 64, 10]
   Input layer: 784 neurons (assuming MNIST flattened images)
   One hidden layer: 64 neurons
   Output layer: 10 neurons (for digit classification)

Experiment Setup:

- Data Loading and Preprocessing: Load the MNIST dataset. Preprocess the data as necessary.

- Model Building:

  Implement Model 1 with three variations, each using a different activation function (sigmoid, tanh, relu) applied to the hidden layers.

  Use the create_model(activation) function to create three Model 2 variations, each with a different activation function (sigmoid, tanh, relu).

  Use a fixed learning rate and optimizer (e.g., Adam) for all models.

- Training and Monitoring:

  Train each model (across both architectures) for a fixed number of epochs.

  During training, monitor and record the training loss and validation loss for each epoch.

- Evaluation:

  Plot the training and validation loss curves for each combination of model architecture and activation function (6 total plots).

  Compare the loss curves to assess the impact of activation functions on:

  Training speed (how quickly the model learns)

  Model convergence (whether the loss stabilizes)

  Overfitting (if the training loss decreases significantly while validation loss increases)

2. Experiment Two: exploring ReLU variants

   Building upon the insights gained from Experiment One, this experiment investigates the impact of ReLU variants (Leaky ReLU and Parametric ReLU) on the performance of your DNN models. This experiment focuses on ReLU, a popular activation function known for its efficiency and effectiveness. However, ReLU suffers from the "dying ReLU" problem, where neurons become permanently inactive due to negative inputs.

   Leaky ReLU: Introduces a small positive slope for negative inputs, allowing a small gradient to flow through and alleviate the dying ReLU problem (where neurons become permanently inactive due to negative inputs).

Parametric ReLU (PReLU): Learns a separate slope parameter for negative inputs, providing more flexibility in gradient propagation.

Experimental steps used are the same as Experiment one.

3. Experiment Three: Depth comparison

Training models with varying depths, evaluating their performance on the test set, and visualizing the results through plots, the code effectively analyzes the impact of depth on model performance (test accuracy and loss)

- Testing Different Depths: The code defines a list depths = [3, 7, 10, 15, 20] which iterates through five different model depths (3, 7,10, 15, and 20 hidden layers).
- Training Models with Varying Depths: Inside the loop, the code creates a new neural network for each depth, constructing the specific architecture based on the current depth value.
- Evaluating Performance: After training each model, the code uses the model.evaluate function to assess its performance on the unseen test set. It captures both test accuracy and test loss, providing valuable metrics for comparison.
- Visualization with Plots
- Test Accuracy vs Depth: visually depicts how test accuracy changes as the model depth increases. It helps identify potential trends, such as whether accuracy improves with deeper models or reaches a plateau (no significant benefit from further depth).
- Test Loss vs Depth: Shows the relationship between test loss and model depth.

RESULTS

1. Experiment One

Model One: Sigmoid: Training loss might be higher, and training speed might be slower compared to other functions.

Tanh: This activation function achieved better training performance (lower training loss and potentially faster convergence) compared to ReLU in your experiment. This is an interesting finding as tanh doesn't necessarily guarantee faster training than ReLU.

Possible explanations for this observation:

- The specific dataset or problem tackling might be well-suited for the centered output range of tanh (-1 to 1) compared to ReLU's non-negative output.
- The chosen learning rate or other hyperparameters might have worked better in conjunction with tanh for this specific model architecture.

  ReLU: While ReLU is generally known for faster training, the data suggests tanh performed better in this case.

Model Two: The results indicate that ReLU generally achieved the lowest training loss (0.066) compared to sigmoid (0.114) and tanh (0.078). This suggests that ReLU's ability to avoid vanishing gradients facilitated faster and potentially more effective learning.

The DNN with ReLU loss curve exhibit a steeper initial descent and reach a lower plateau compared to sigmoid and tanh models, indicating faster convergence.

While ReLU achieved the highest training accuracy (0.979), tanh closely followed (0.977). However, ReLU also delivered the highest validation accuracy (0.975), suggesting better generalization to unseen data. Examining the accuracy trends over epochs could reveal if the ReLU model maintained its advantage throughout training.

ReLU tackles the vanishing gradient problem better than sigmoid/tanh because: Unlike sigmoid/tanh, ReLU outputs the input directly for positive values (ReLU(x) = x for x > 0). During backpropagation with positive inputs, the gradient isn't reduced, allowing it to flow through the network without vanishing. In essence, ReLU avoids the constant reduction of gradients seen in sigmoid/tanh, enabling better learning in deeper architectures.

1. Experiment Two

The provided results for Leaky ReLU and PReLU suggest they might not have offered significant performance gains over ReLU in this specific experiment. Leaky ReLU's training and

validation accuracy were slightly lower than ReLU, while PReLU achieved marginally better performance.

Train DNN models with ReLU, Leaky ReLU, and PReLU activations. Compare the training and validation performance. Analyze if Leaky ReLU or PReLU offers any significant advantage in terms of training speed or final accuracy.

2. Experiment Three:

The test accuracy starts high (around 0.97) for a model with 3 hidden layers (depth 3) and gradually decreases as the depth increases (7, 10, 15, 20 layers). This suggests that deeper models might not necessarily lead to better performance on this specific task (MNIST handwritten digit classification).

The test loss follows a similar trend as test accuracy. It starts low (around 0.09) for the 3-layer model and increases with increasing depth (0.11, 0.12, 0.16, 0.21). This reinforces the point that deeper models in this case might not be learning better representations of the data.

While deeper models can theoretically learn more complex features from data, these results demonstrate that simply adding more layers doesn't always translate to better performance. In this specific case:

- The model with 3 hidden layers seems to achieve a good balance between capturing essential features and avoiding overfitting the training data, resulting in the highest test accuracy (0.97) and a relatively low test loss (0.09).

Increasing the depth beyond 3 layers leads to diminishing returns or potentially overfitting, where the model performs well on the training data but generalizes poorly to unseen data (test set). This is evident from the decrease in test accuracy and increase in test loss as the number of layers increases.

Possible Explanations:

- The MNIST dataset might not be complex enough to require a deep architecture. A 3-layer model might be sufficient to capture the necessary features for classifying handwritten digits.

- Overfitting with Deeper Models: Deeper models with more parameters can become prone to overfitting if they're not trained properly with techniques like regularization or dropout. This might explain the decrease in test accuracy and increase in test loss for models with more layers.

FUTURE WORK

Experiment with deeper network architectures to further examine the impact of activation functions and Explore other activation functions designed to address specific challenges which will help to investigate techniques like gradient clipping or weight initialization to mitigate vanishing gradients.

CONCLUSION

The experiment confirms that the choice of activation function can significantly impact the training dynamics and performance of a DNN. ReLU effectively addressed the vanishing gradient problem, leading to faster convergence and potentially better generalization as evidenced by the higher validation accuracy. While sigmoid and tanh achieved reasonable performance, their susceptibility to vanishing gradients hindered their learning efficiency. Leaky ReLU and PReLU, designed to address limitations in ReLU, didn't show a clear advantage in this specific case. However, they may be beneficial in other network architectures or datasets.

While deeper models have the potential for better performance, The provided results highlight the importance of carefully considering depth in neural network design.