Assignment 1 – Artificial Neural Networks Report

Introduction

This report presents the design, training, and evaluation of an artificial neural network (ANN) used for classifying images from the FashionMNIST dataset. The task involves training a deep learning model to predict one of 10 clothing categories based on 28x28 grayscale images. Key aspects of the report include the chosen network architecture, the preprocessing steps applied to the dataset, the loss function and optimizer used, and the hyperparameter tuning process. The aim of this project is to demonstrate the practical application of neural networks in image classification while addressing challenges such as overfitting, convergence, and model evaluation.

---

1. Network Topology

The architecture chosen for this classification task is a fully connected feedforward artificial neural network (ANN). The network consists of the following layers:

- **Input Layer:** 784 nodes, corresponding to each of the 28x28 grayscale pixels of an input image.
- **First Hidden Layer:** 256 nodes with ReLU activation, followed by dropout (p=0.2).
- **Second Hidden Layer:** 128 nodes with ReLU activation, followed by dropout (p=0.2).
- **Third Hidden Layer:** 64 nodes with ReLU activation.
- **Output Layer:** 10 nodes (one for each FashionMNIST class), with no activation function applied because the CrossEntropyLoss(softmax) handles the classification tasks internally which turn the output into probabilities.

After experimenting with different configurations and balancing model complexity with overfitting concerns. ReLU was selected for its efficiency and performance in training deep networks by mitigating vanishing gradient issues.

This structure balances complexity and generalization. Shallower networks underfit, while deeper ones began overfitting. This 3-layer configuration delivered the most consistent results during validation.

---

2. Preprocessing

The preprocessing steps performed include:

- **Data Normalization:** All image data was converted from uint8 pixel values (0-255) to float and normalized to [0,1] range by dividing by 255.0. This ensures that the gradient descent process remains stable and converges more quickly. Neural networks work best when input values are small and on a consistent scale. Keeping input values in [0,1] ensures faster training, better numerical stability, and often higher model accuracy.

- **Train-Validation Split:** The original training dataset was randomly split, using a subset (same size as test set) as validation data. This allows for unbiased performance evaluation during training.
- **Flattening:** Since the network is fully connected, each 28x28 image was reshaped into a 784-element vector.

These steps help standardize the input data for better convergence and allow us to monitor generalization.

---

3. Loss Function

We used the **CrossEntropyLoss**, which is standard for multi-class classification problems. It measures the distance between the predicted class distribution and the actual class label. Cross-entropy is particularly well-suited here since:

- It integrates the softmax activation implicitly, avoiding manual numerical instability.
- It provides a meaningful loss value that is minimized when the predicted probability distribution is close to the true class.

---

4. Optimizer

The optimizer used was **Adam (Adaptive Moment Estimation)** with a learning rate of 0.001. Adam dynamically adjusts learning rates for each parameter using estimates of first and second moments of the gradients. This makes it well-suited for noisy datasets like images and generally leads to faster convergence.

Alternative options like SGD and RMSprop were tested, but Adam provided the best balance between speed and validation performance. This allows:

- Fast convergence with minimal hyperparameter tuning.
- Stable training across sparse and noisy gradients, which is common in image datasets.

---

5. Training and Validation Procedure

The training was done over 30 epochs using a mini-batch size of 64. Each epoch consisted of:

1. **Training Phase:**
    - Gradients were computed with backpropagation.
    - Model parameters were updated using Adam.
    - Batch-wise accuracy and loss were accumulated.
2. **Validation Phase:**
    - The model was evaluated on the validation set without gradient updates.
    - Accuracy and loss were recorded to monitor overfitting and convergence.

This loop provided real-time feedback on training dynamics, allowing us to adjust hyperparameters if necessary.

---

6. *Hyperparameter Tuning and Architecture Selection*

I used a form of manual grid search, systematically testing combinations of:

- Number of hidden layers (1-4)
- Nodes per layer (64, 128, 256)
- Learning rates (0.01, 0.001)
- Dropout rates (0.1, 0.2, 0.3)
- Optimizers (SGD, Adam)

We monitored validation accuracy and loss to determine the optimal architecture. The best-performing configuration:

- 3 hidden layers (256-128-64)
- ReLU activation
- Epoch = 30
- Dropout = 0.2
- Learning rate = 0.001
- Adam optimizer

This model consistently reached validation accuracy around 88-90%, which was significantly better than shallower or deeper models that either underfit or overfit.

---

7. *Other Details*

- **JPEG Image Inference:** A CLI interface was added post-training to load external 28x28 grayscale JPEG images using torchvision.io.read_image, resized if necessary. Input images are reshaped and normalized before classification.
- **Reproducibility:** Random selection of validation indices ensures some variance in results but reflects realistic training dynamics.
- **No Model Saving/Loading:** The design choice to retrain the model on every run (instead of saving/loading a .pt file) ensures simplicity and avoids dealing with device mismatches (cpu vs cuda) for image inference. This is a trade-off between reproducibility and ease of use.

Conclusion

This assignment involved a structured design, training, and tuning process for an artificial neural network aimed at classifying FashionMNIST images. Through careful architectural choices, preprocessing, and hyperparameter tuning, the final ANN achieved high validation and test accuracy while maintaining good generalization.

References

- https://pytorch.org/tutorials/beginner/nn_tutorial.html
- https://cs231n.github.io/neural-networks-2/#datapre
- https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
- https://pytorch.org/tutorials/beginner/saving_loading_models.html
- FashionMNIST Dataset: https://github.com/zalandoresearch/fashion-mnist