Homework 2 Report

Havanur Dervişoğlu, No: 504242523

Computer Engineering, ITU, İstanbul, Türkiye. dervisoglu25@itu.edu.tr

Keywords: AutoEncoder, FeedForward Neural Network

Grade	Q1	Q2	Total
Max	5	5	**10 pts**
Expected	4	4	**8 pts**
Given			

The full source code for this homework, including all figures and scripts, can be found in the following GitHub repository:

https://github.com/drvshavva/BLG_527E_hw2

Dataset: Fashion MNIST

- Type: Image classification dataset, 10 clothing categories
- Number of Samples: Training set: 60,000 images, Test set: 10,000 images
- Image Dimensions: 28 x 28 pixels (grayscale)

Q1) Training Autoencoder (AE) to generate new samples for Zalando Fashion-MNIST dataset

An autoencoder is a neural network trained to copy its input to its output through an internal hidden layer h that represents a compressed code (Figure 1). The goal is not perfect copying but learning useful representations by constraining the network's capacity.

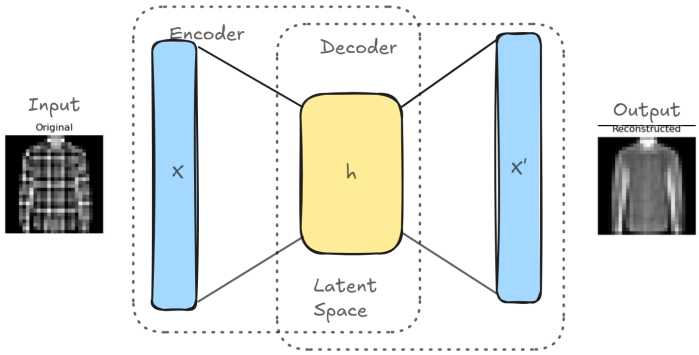


Figure 1: Autoencoder Architecture

An autoencoder is fundamentally an unsupervised learning algorithm. It applies backpropagation with a unique setup: the target values are set to be equal to the inputs. That is, for a training example $x^{(i)}$, the target output $y^{(i)}$ is also $x^{(i)}$. The core task of an autoencoder is to learn a function, $h_{W,b}(x)$, that approximates the identity function, aiming to reconstruct an output $\hat{x}$ that is as similar as possible to the original input x. While learning the identity function might seem trivial, the interesting aspect comes from placing constraints on the network, which forces it to discover structure within the data.

a) Autoencoder's architecture and hyperparameters

In this study, a fully-connected autoencoder is implemented to compress and reconstruct 28×28 grayscale images from the Fashion-MNIST dataset. The model consists of two main components: an Encoder and a Decoder, combined in a single Autoencoder class (Figure 2). Model architecture layers, which are implemented in this study, are defined as below:

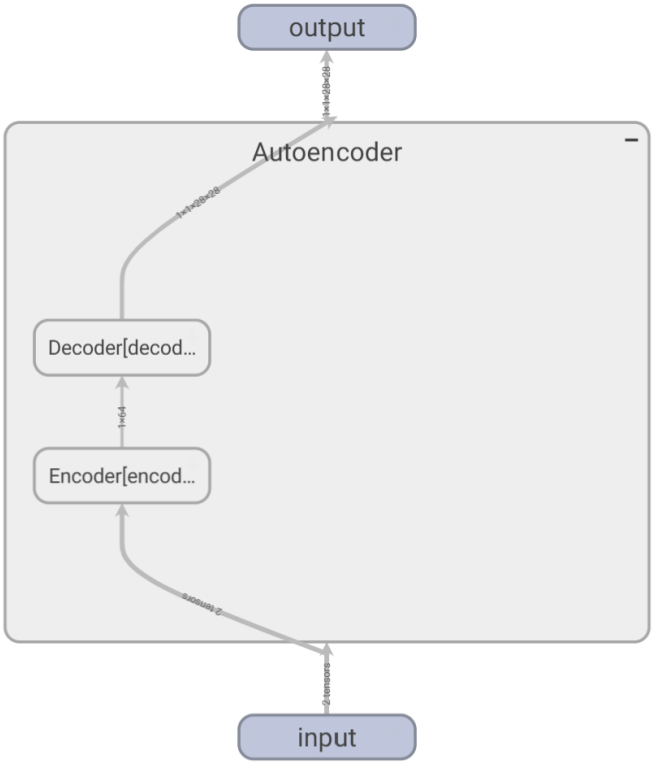


Figure 2: Our Autoencoder Architecture

Encoder: The encoder compresses the input image into a lower-dimensional latent space using a series of fully connected (dense) layers with ReLU activations (Figure 3):

- Input: Flattened image vector of size 784 (28×28)
- Hidden layers:
- Linear (784 → 512) + ReLU
- Linear (512 → 256) + ReLU
- Linear (256 → 128) + ReLU
- Linear (128 → latent_dim)
- Output: Compressed latent vector of size latent_dim (64)

Decoder: The decoder reconstructs the original image from the latent vector. It mirrors the encoder architecture in reverse order (Figure 4):

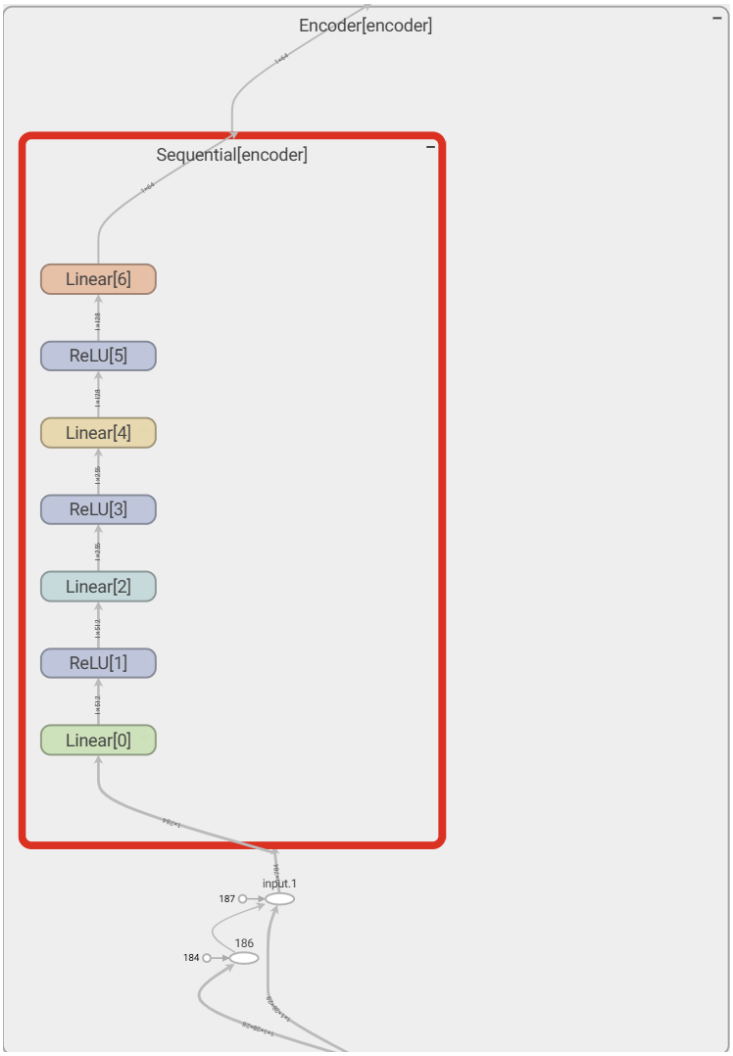


Figure 3: Our Autoencoder Architecture - Encoder

- Input: Latent vector of size latent_dim
- Hidden layers:
- Linear (latent_dim $\rightarrow$ 128) + ReLU
- Linear (128 $\rightarrow$ 256) + ReLU
- Linear (256 $\rightarrow$ 512) + ReLU
- Linear (512 $\rightarrow$ 784) + Sigmoid
- Output: Reconstructed image reshaped to (1, 28, 28)

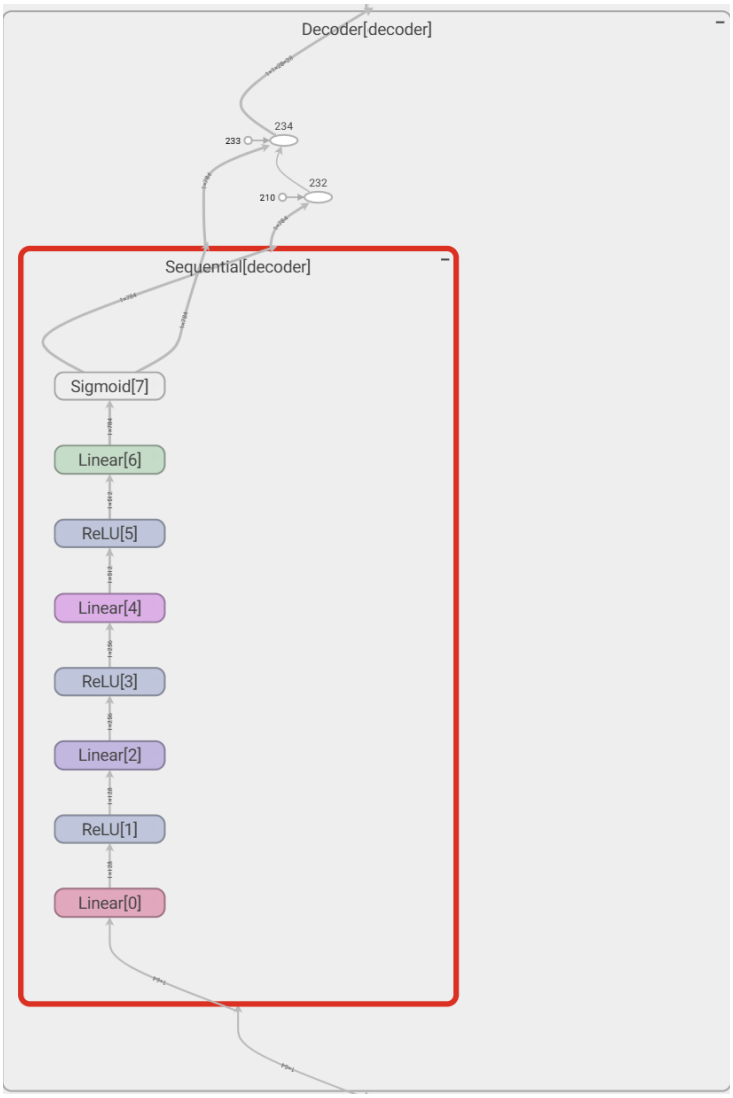


Figure 4: Our Autoencoder Architecture - Decoder

Logging and Visualization: TensorBoard is used to monitor:

- Training and validation loss per epoch
- Batch-wise training loss
- Sample reconstructions every 5 epochs
- Latent space visualization (optional)
- Generated samples from randomly sampled latent vectors

Table 1: Hyperparameter Configuration for Autoencoder Training

Hyperparameter	**Value**	**Description**
Latent Dimension	64	Dimensionality of the compressed representation in latent space
Learning Rate	1×10^{-3}	Step size used by the optimizer for updating weights
Optimizer	Adam	Adaptive learning rate optimizer; generally faster convergence
Loss Function	MSELoss	Mean Squared Error between input and reconstruction
Epochs	20	Number of complete passes over the training dataset
Batch Size	64	Number of samples per mini-batch during training
Activation Functions	ReLU, Sigmoid	ReLU in hidden layers, Sigmoid at output to match pixel intensity scale

b) Training process and final reconstruction loss

The training and validation loss curves demonstrate a successful and stable learning process for the autoencoder model. Throughout the 20 epochs, the training loss steadily decreased by approximately 74%, from 0.0395 to 0.0103, while the validation loss showed a consistent reduction of around 56%, from 0.024 to 0.0106 (Figure 5). This parallel decline indicates that the model not only learned to reconstruct the training data effectively but also maintained good generalization performance on unseen validation data. Importantly, there are no signs of overfitting, as the validation loss continues to decrease alongside the training loss.

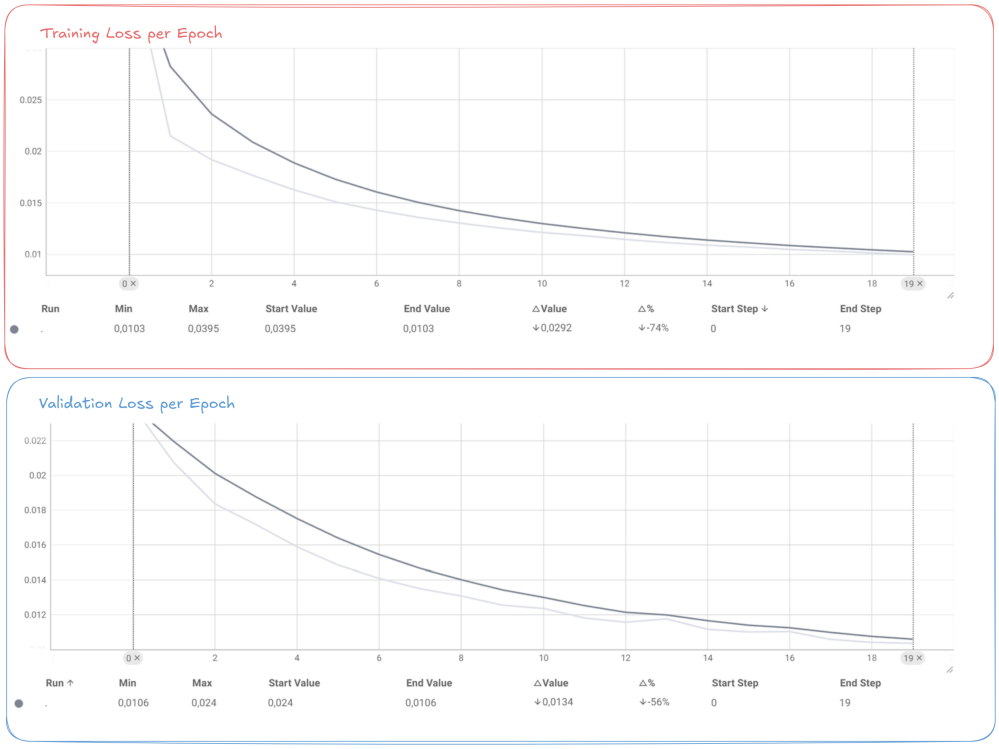


Figure 5: Train and Validation Reconstruction Loss

c) Examples of original vs reconstructed image

Figure 6 illustrates the reconstruction performance of the autoencoder across different training stages. At Epoch 5, the reconstructions capture only coarse outlines and general shapes, with considerable blurring and limited detail. By Epoch 10, the outputs show clearer silhouettes and improved structural fidelity, though finer textures are still missing. At Epoch 15, the reconstructions become sharper and more recognizable, closely resembling the original images in both form and positioning. Finally, by Epoch 20, the autoencoder effectively reconstructs the input images with strong visual similarity and distinguishable class characteristics, such as the outlines of shoes, pants, and jackets. Overall, the qualitative results confirm that the model progressively learns more discriminative features over time, leading to visually accurate reconstructions as training progresses.

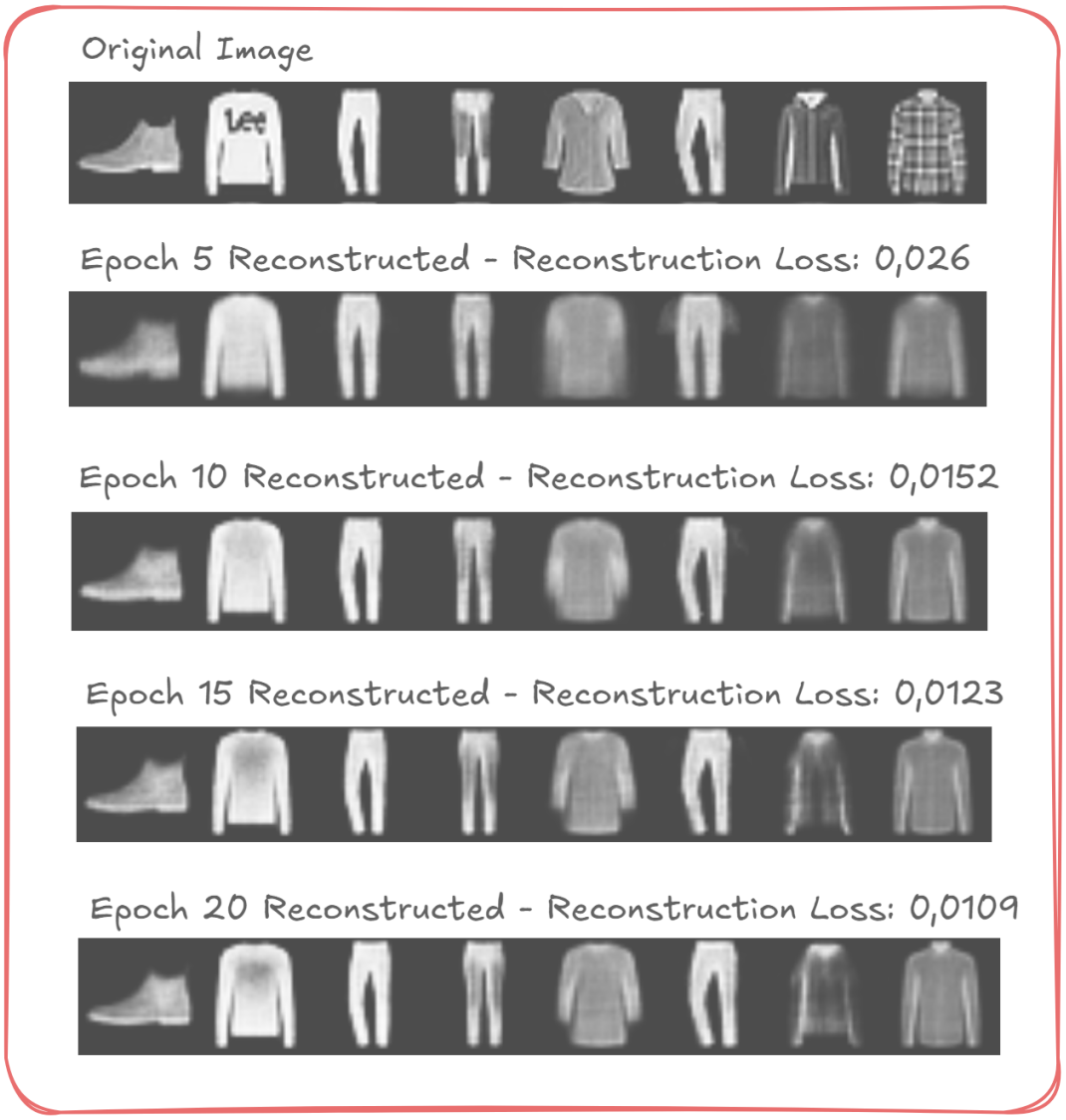


Figure 6: Reconstruction of Images over Epochs

d) Examples of generated samples

Figure 7 shows synthetic images generated by the model using randomly sampled vectors from the latent space. Each image is not generated from training data, but only from the representations learned by the model. When the images are examined, it can be said that basic structures such as shoulder width, arm structure and length of the clothes are clear. We can say that the images are blurry and have low detail, this may be because the latent space has not yet learned enough discriminators. Many examples belong to similar categories, which is probably because some regions in the latent space represent certain classes better.

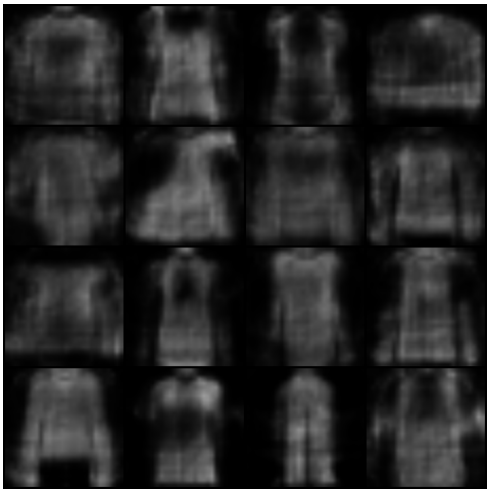


Figure 7: Generation of Images from Latent Space

According to Figure 8, we can observe distinct clusters. This indicates that the model positions similar examples closely in the latent space, demonstrating a discriminative representation. While some classes are quite clearly separated, we see that in some areas the colors are overlapping, that is, some classes are mixed with each other in the latent space. This explains why there is no complete separation in some classes in the results seen in Figure 7. This is because the model cannot fully separate the classes or because there is a natural similarity between the classes.

Figure 8: Latent Space

Q2) Feedforward neural network for classification

Feedforward neural network is one of the advanced types of artificial neural networks. In these networks, information starts from the input nodes, passes through hidden nodes (if any) and moves only in one direction - forward - towards the output nodes (Figure 9). There is no loop or feedback in the network. Feedforward neural networks were one of the first types of artificial neural networks invented and are simpler than other types such as recurrent neural networks or convolutional neural networks. As data moves through the network's artificial network, each layer processes one aspect of the data, filters out outliers, identifies familiar entities, and produces the final output.

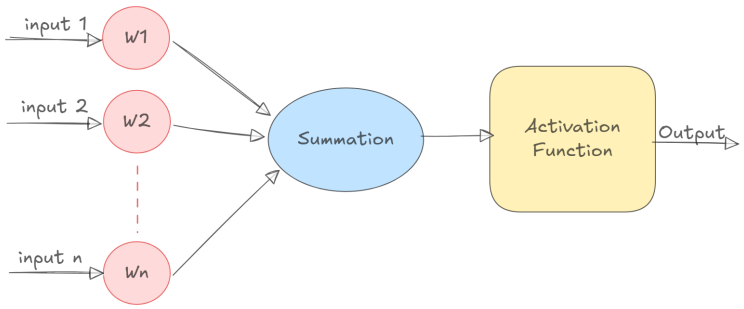


Figure 9: Single Neuron Neural Network

The architecture of a feedforward neural network consists of three types of layers: input layer, hidden layers, and output layer (Figure 10). Each layer consists of units known as neurons, and the layers are connected to each other by weights.

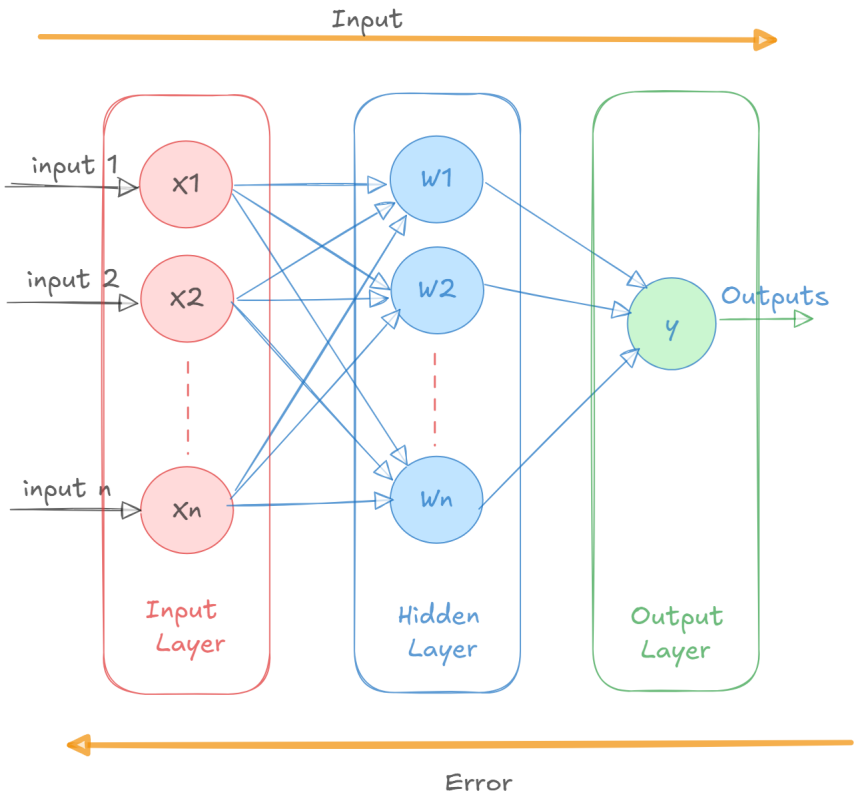


Figure 10: Feedforward Neural Network

- Input Layer: This layer consists of neurons that receive inputs and pass them on to the next layer. The number of neurons in the input layer depends on the dimensions of the input data or the features in the dataset. It is the only layer that passes information from the outside world without any processing.

- Hidden Layers: These layers are not directly exposed to input or output and can be thought of as the computational engine of the neural network. The neurons of each hidden layer take the weighted sum of the outputs of the previous layer, apply an activation function, and pass the result on to the next layer. There can be zero or more hidden layers in the network. These layers apply transformations to the data, and as the network is trained, their weights are updated to become more predictive. It is these intermediate layers that make deep learning possible; they do all the computations and extract the features of the data.

- Output Layer: This layer is the last layer that produces the output or prediction for the given inputs. The number of neurons in the output layer depends on the number of possible outputs the network is designed to produce or the feature being predicted. This layer is where we get the final result.

Every neuron in a layer is connected to every neuron in the next layer, making it a fully connected network. The strength of the connection between neurons is represented by weights. Weights can be compared to the coefficients in a linear regression and are usually initialized with small random values such as 0 and 1. The primary purpose of the weights is to prioritize the features that are most helpful in the learning process.

A feedforward neural network works in two stages: the feedforward stage and the backpropagation stage.

Feedforward Stage: In this stage, the input data is fed into the network and propagated forward through the network. In each hidden layer, the weighted sum of the inputs is calculated and passed through an activation function that gives the model non-linear features. This process continues until the output layer is reached and a prediction is made.

Backpropagation Phase: After a prediction is made, the error (difference between the predicted output and the actual output) is calculated. This error is then propagated back through the network and the weights are adjusted to minimize this error. The process of adjusting the weights is usually done using a gradient descent optimization algorithm. This process is repeated until the network performs satisfactorily on the training data. One round of updating the network for the entire training dataset is called an epoch.

Activation functions play a very important role in feedforward neural networks. They provide the network with nonlinear properties, allowing the model to learn more complex patterns. Common activation functions include sigmoid, tanh, and ReLU (Rectified Linear Unit). Other examples are Unit step, piecewise linear, and Gaussian.

A feedforward neural network is trained by adjusting the weights. This is an iterative process where the dataset is passed through the network multiple times, updating the weights each time to reduce the prediction error. This process is called gradient descent (Figure 11).

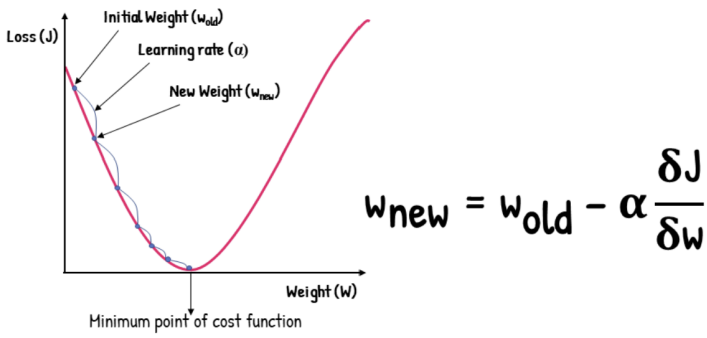


Figure 11: Gradient Descent (image source: `https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/`)

a) Feedforward NN architecture and hyperparameters

Three different networks were used in the study. In this way, the effects of different models on the classification success were observed.

- **Arch. 1:** It is a flexible classification model with 3 hidden layers, batch normalization in each layer, ReLU activation and 30% dropout, and 10 classes at the output (Figure 12). This structure provides sufficient depth and tries to reduce the risk of overfitting with regularization techniques.
- **Arch. 2:** It is a simpler and less computationally demanding model (Figure 13). It has two fully connected layers in total: the first one connects 128 hidden neurons from 784-dimensional input, then ReLU activation and 20% dropout are applied. The second layer directly produces 10-class output.
- **Arch. 3:** It is a deeper neural network with five hidden layers and is regularly enriched with Batch Normalization, ReLU activation and Dropout techniques (Figure 14). It contains layers with 512, 256, 128, 64 and 32 neurons respectively from the input.

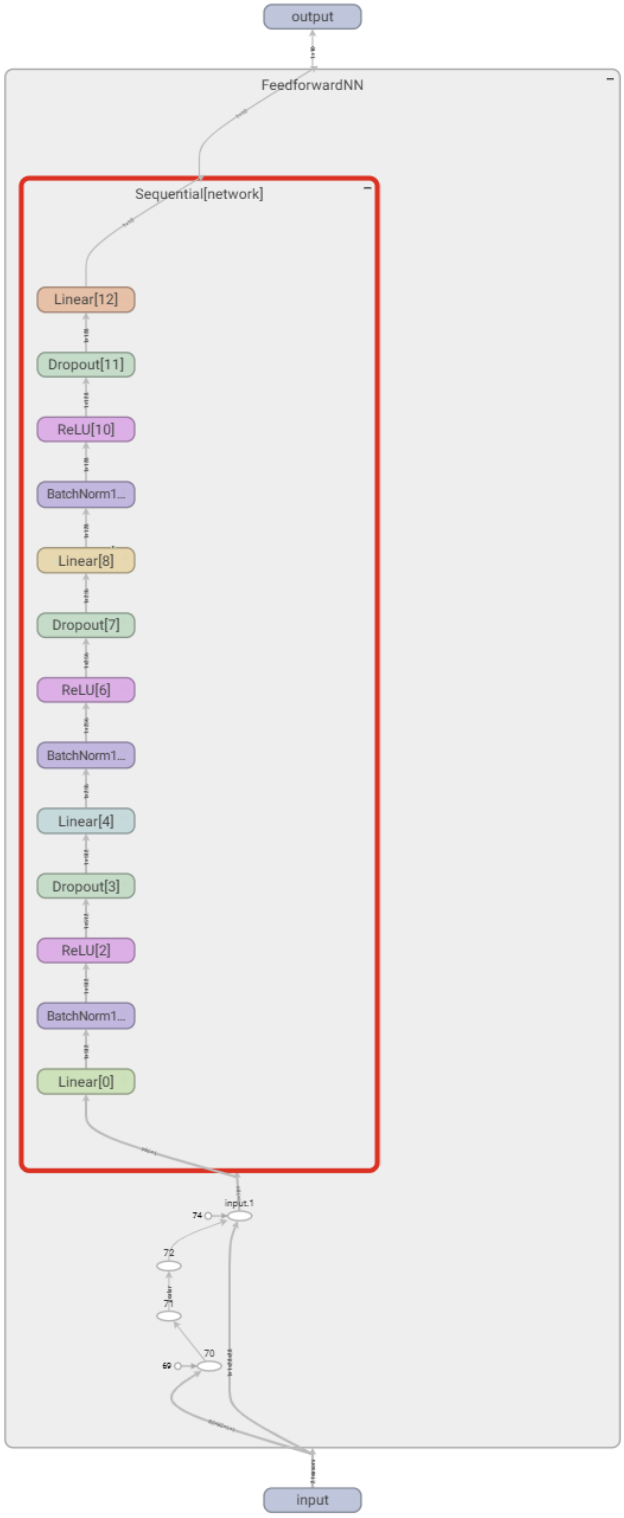


Figure 12: FeedForward Neural Network Architecture 1

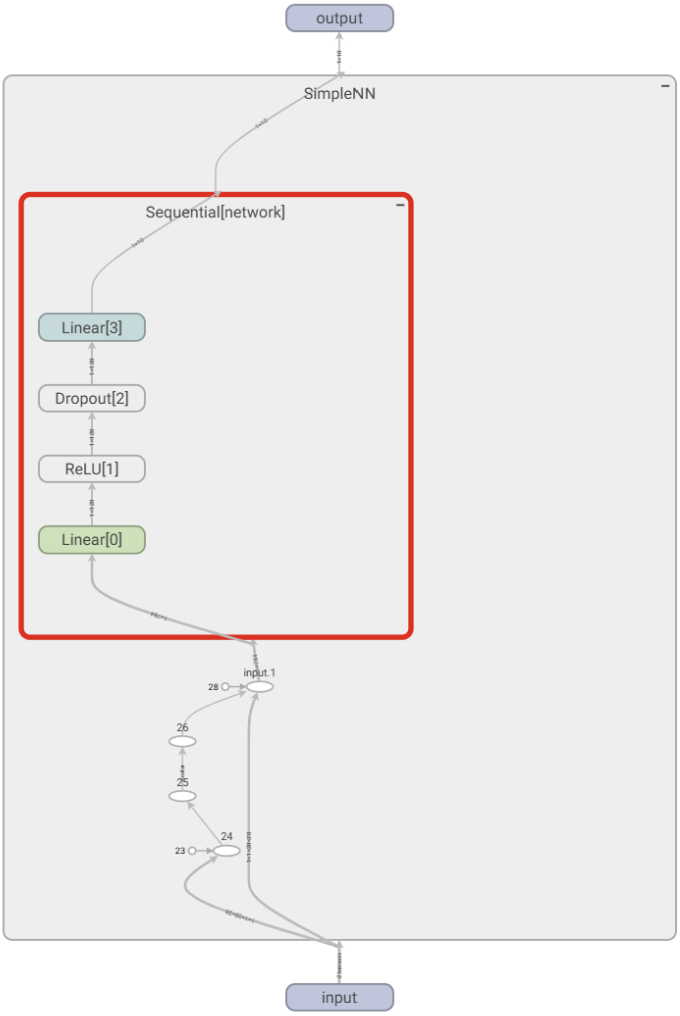
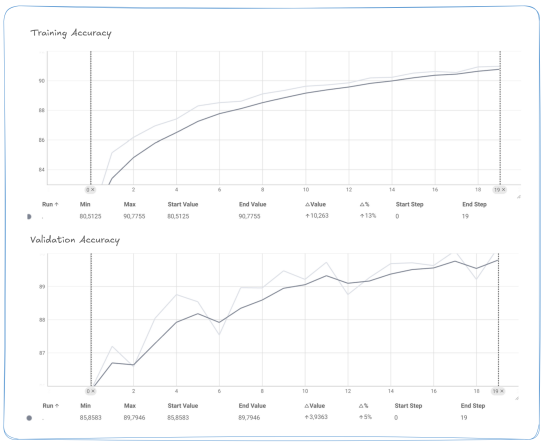


Figure 13: FeedForward Neural Network Architecture 2

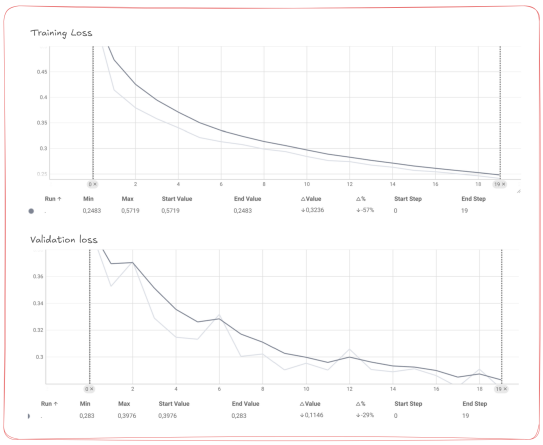
Figure 14: FeedForward Neural Network Architecture 3

b) Training process

For Architecture 1, we can say that the model has learned successfully - both training and validation losses are continuously decreasing (Figure 15(b)). Training accuracy increased from 80.5% to 90.8% (13% increase). Validation accuracy increased from 85.9% to 89.8% (5% increase) (Figure 15(a)). The loss functions show a decreasing trend as expected.



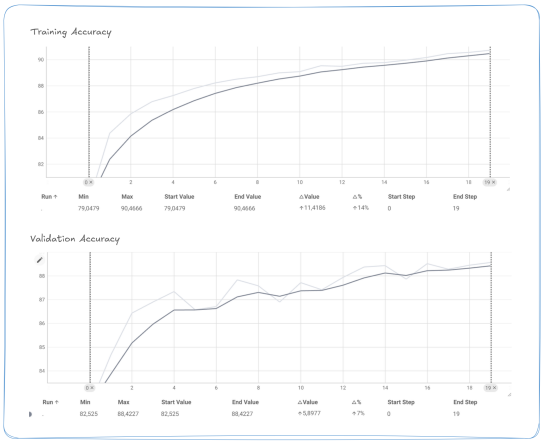
(a) Accuracy over Epochs - Architecture 1



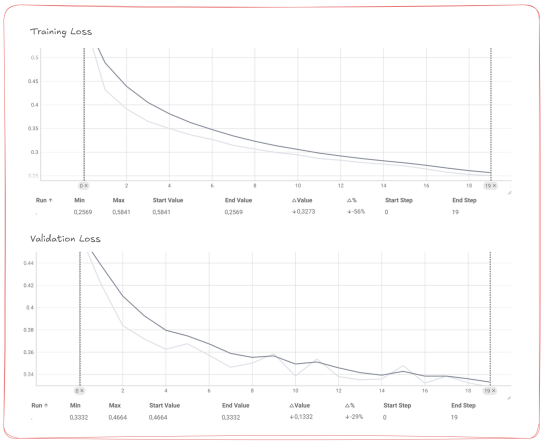
(b) Loss over Epochs - Architecture 1

Figure 15: Training performance metrics for Architecture 2

The results for Architecture 2 show an interesting picture when compared to Architecture 1. Architecture 2 shows a dramatic 56% reduction in training loss (Figure 16(b)), while improving training accuracy from 79% to 90.5% (14% increase) and validation accuracy from 82.5% to 88.4% (7% increase) (Figure 16(a)). Architecture 1 exhibits similar training performance, but slightly lower validation performance (88.4% vs. 89.8%). The most striking difference is that this architecture exhibits much more stable learning curves - there is almost no fluctuation in validation metrics. However, the difference between training and validation accuracy (2.1%) is larger than the previous architecture, indicating a stronger tendency towards overfitting. As a result, this new architecture has a stronger learning capacity and offers a more stable training process, but it needs more regularization techniques in terms of generalization ability.



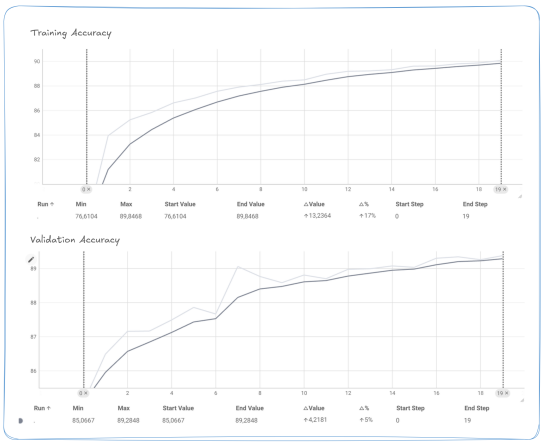
(a) Accuracy over Epochs - Architecture 2



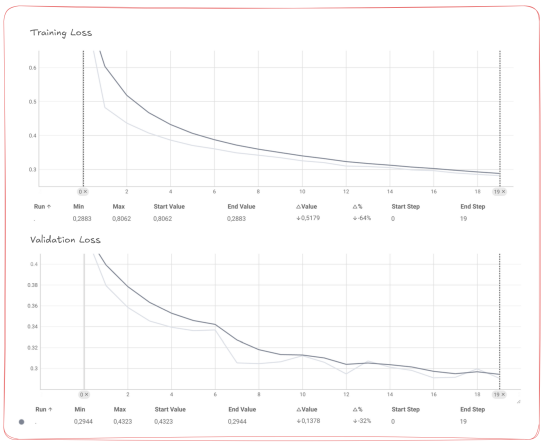
(b) Loss over Epochs - Architecture 2

Figure 16: Training performance metrics for Architecture 2

Architecture 3 shows a 17% increase in training accuracy from 76.6% to 89.8%, while only a 5% improvement in validation accuracy from 85.1% to 89.3% (Figure 17(a)). In the loss metrics, the training loss is reduced by 64%, while the validation loss is reduced by only 32% (Figure 17(b)). This architecture has the lowest initial training accuracy compared to the previous two, but eventually reaches similar levels (89.8% vs. 90.8% for the first model and 90.5% for the second model). The most notable difference is that this architecture shows a dramatic jump in validation performance between steps 6-8 - this spike indicates that the model has made a significant learning leap at this point. In terms of validation accuracy, this model performs the best (89.3%), but it seems to have the best generalization ability, as the difference between training and validation is the lowest of the three architectures at 0.5%. Overall, this model exhibits a slower learning pattern that then accelerates, and is the least prone to overfitting.



(a) Accuracy over Epochs - Architecture 3

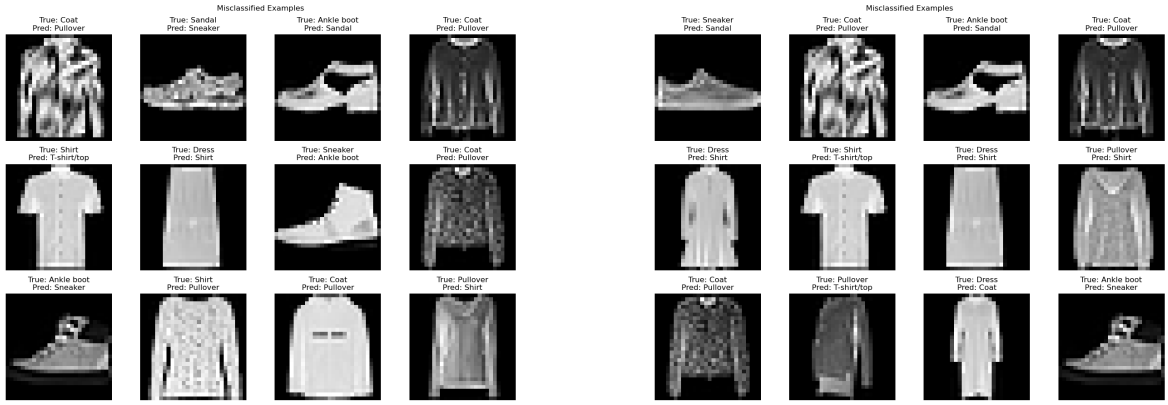


(b) Loss over Epochs - Architecture 3

Figure 17: Training performance metrics for Architecture 3

c) Performance of the model

The comparison in Figure 18, which shows examples of misclassifications by two different architectures on the dataset, shows that the models exhibit different error patterns. The most frequent errors in Architecture 1 occur between visually similar categories - for example, Coat is classified as Pullover, Shirt as T-shirt, and Sneakers as Ankle boot (Figure 18(a)). The model is particularly confused between upper body garments (Coat, Pullover, Shirt) and has difficulty distinguishing shoe types (Sneaker-Ankle boot). Architecture 3, on the other hand, shows similar error types but with some differences - for example, Pullover is classified as T-shirt/Shirt and Dress is recognized as Coat (Figure 18(b)). Both architectures make errors especially on products with similar geometric shapes and textural features, indicating that the models have difficulty capturing fine details. In general, these errors reflect the inherent difficulty level of Fashion-MNIST, as some categories are very similar and can be difficult to distinguish even for the human eye. The fact that both architectures achieve around 89% accuracy suggests that it is reasonable to make errors in such borderline cases.



(a) Missclassified Examples - Architecture 1

(b) Missclassified Examples - Architecture 3

Figure 18: Missclassified Examples

Class	**Precision**	**Recall**	**F1-Score**	**Support**
T-shirt/top	0.84	0.85	0.85	1000
Trouser	0.98	0.98	0.98	1000
Pullover	0.78	0.83	0.80	1000
Dress	0.90	0.89	0.89	1000
Coat	0.77	0.85	0.81	1000
Sandal	0.97	0.96	0.96	1000
Shirt	0.77	0.64	0.70	1000
Sneaker	0.94	0.95	0.95	1000
Bag	0.98	0.97	0.98	1000
Ankle boot	0.95	0.96	0.95	1000
Accuracy			**0.89**	10000
Macro avg	0.89	0.89	0.89	10000
Weighted avg	0.89	0.89	0.89	10000

Table 2: Architecture 1 - Classification report for Fashion-MNIST test set

Architecture 1 (89% accuracy) shows excellent performance especially in Bag (98% F1-Score) and Trouser (98% F1-Score), but struggles in Shirt (70% F1-Score) and Coat (81% F1-Score) (Table 2). Architecture 2 (88% accuracy) shows excellent performance (1.00 precision) in Trouser, but performs slightly lower than Architecture 1 overall, especially weak in Shirt (68% F1-Score) and Pullover (79% F1-Score) (Table 3). Architecture 3 (88% accuracy) shows strong performance in Bag (97% F1-Score) and Trouser (98% F1-Score), but struggles similarly in Shirt (71% F1-Score) and Coat (81% F1-Score) (Table 4). In general, all three architectures are successful in capturing products with distinct shapes such as footwear (Sneakers, Ankle boots) and bags/sandals, but struggle to capture subtle differences between upper body garments (especially Shirt, Coat, Pullover). This pattern reflects the natural difficulty distribution of Fashion-MNIST and shows that the architectures face similar classification challenges.

Class	Precision	Recall	F1-Score	Support
T-shirt/top	0.81	0.85	0.83	1000
Trouser	1.00	0.96	0.98	1000
Pullover	0.81	0.78	0.79	1000
Dress	0.88	0.89	0.89	1000
Coat	0.77	0.86	0.81	1000
Sandal	0.97	0.94	0.96	1000
Shirt	0.72	0.65	0.68	1000
Sneaker	0.91	0.97	0.94	1000
Bag	0.98	0.96	0.97	1000
Ankle boot	0.97	0.94	0.95	1000
Accuracy			0.88	10000
Macro avg	0.88	0.88	0.88	10000
Weighted avg	0.88	0.88	0.88	10000

Table 3: Architecture 2 - Classification report for Fashion-MNIST test set

Class	Precision	Recall	F1-Score	Support
T-shirt/top	0.85	0.83	0.84	1000
Trouser	0.97	0.98	0.98	1000
Pullover	0.82	0.77	0.80	1000
Dress	0.90	0.86	0.88	1000
Coat	0.76	0.87	0.81	1000
Sandal	0.97	0.96	0.96	1000
Shirt	0.71	0.70	0.71	1000
Sneaker	0.92	0.96	0.94	1000
Bag	0.97	0.97	0.97	1000
Ankle boot	0.97	0.94	0.96	1000
Accuracy			0.88	10000
Macro avg	0.89	0.88	0.88	10000
Weighted avg	0.89	0.88	0.88	10000

Table 4: Architecture 3 - Classification report for Fashion-MNIST test set

Table 5 shows that the performances of the three architectures on the Fashion-MNIST test set are quite close. Architecture 1 is the leader with 88.80% accuracy, followed by Architecture 2 with 88.00% and Architecture 3 with 88.40% accuracy. The most interesting point is that Architecture 3 (0.8841) has the highest score in terms of F1-Score - indicating that it is the best in terms of balanced performance. When the precision and recall values are examined, it is seen that Architecture 1 exhibits higher precision (0.8882), while Architecture 3 is superior in recall (0.8841) and overall F1-Score. These results show that all three architectures have similar learning capacity, but Architecture 3 exhibits a more balanced performance between classes. Although the performance differences are minimal (0.8% range), Architecture 3's superiority in F1-Score shows that it establishes a better balance between precision and recall in different classes and may be a more reliable model for practical applications.

Architecture	**F1-Score**	**Precision**	**Recall**	**Accuracy**
Architecture 1	0.8871	0.8882	0.8880	0.8880
Architecture 2	0.8800	0.8800	0.8800	0.8800
Architecture 3	0.8841	0.8854	0.8841	0.8840

Table 5: Overall performance metrics of different architectures on the Fashion-MNIST test set