MASTER OF DATA SCIENCE (SMDS)

Assignment-2

Machine Learning (CSE5ML)

by

SAIKIRAN CHALLA
21356161

(Semester-2 2023)

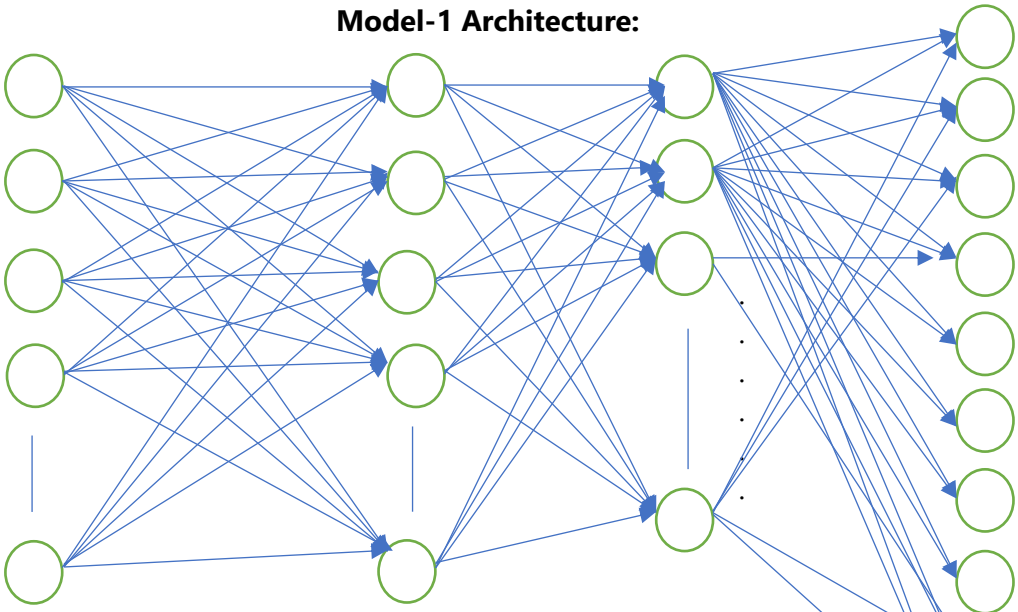
Aim : To Build Neural Networks for supervised learning as a multi-class classification problem for the fashion images.

Task -1 :

Input Layer: An image's flattened pixels are represented by the 28x28 = 784 nodes in the input layer. The intensity value of each pixel is represented by a node in this layer.

1. First Hidden Layer (Dense Layer):
   128 neurons in total.Rectified Linear Unit, or ReLU, is the activation function.
2. Dense Layer, or the Second Hidden Layer:
   There are 64 neurons.**ReLU** is the activation function. Dense layer output:
   Ten neurons, corresponds to the 10 classes (assuming a classification task).
   Softmax is the activation function that is usually utilised in multi-class classification.

Model-1 Architecture:



Input-784 neurons(28 * 28) (128

Hidden layer 1 128 neurons)

Hidden layer2 (64 neurons)

Output layer

Using the well-known deep learning framework Keras, a straightforward neural network model was created. The Fashion MNIST dataset's images are to be classified using this model. The model and the code are broken down as follows:

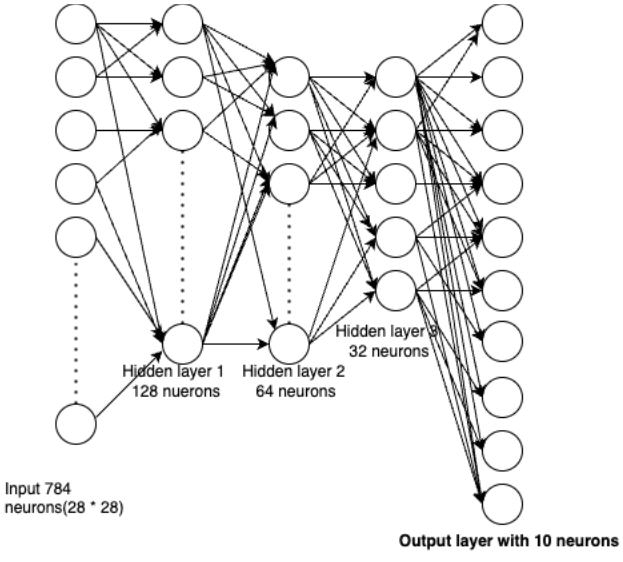
1. **Import Libraries:**
   - Import the required libraries, such as Keras for neural network construction and training and Pandas for data management.
2. **Load Fashion MNIST Data:**
   - For training and testing, load the Fashion MNIST dataset, which includes pictures of apparel items and the labels that go with them.
3. **Data Preprocessing:**
   - To express each image as a 1D array of pixel values, reshape the image data to be one-dimensional (flattened).
   - Divide the pixel values by 255 to normalise them. The pixel values are scaled to the range [0, 1] as a result.
4. **Model Definition:**
   - Construct a sequential model, or a layer-by-layer stack.
   - To the model, add three dense (completely linked) layers:
     - With 128 neurons, the first layer makes use of the ReLU activation function.
     - The ReLU activation function is also used by the 64 neurons in the second layer.
     - Ten neurons make up the final layer, which is equivalent to the ten classes in the Fashion MNIST dataset. For multi-class classification issues, it often employs the softmax activation function.
5. **Model Compilation:**
   - Assemble the model by indicating:
     - 'sparse_categorical_crossentropy' is a popular loss function option for classification jobs involving integer labels.
     - An optimisation algorithm that is frequently employed is called "adam."
     - 'Accuracy' is one of the metrics to keep an eye on during training.
6. **Model Training:**
   - Using the training data, fit (train) the model:
     - Training data: labels from y_train and flattened images from x_train.
     - Ten training epochs will be used, meaning the model will go through the training set ten times.
     - Batch size: 64 — For every training cycle, the data is split up into batches of 64 samples.
7. **Model Evaluation:**
   - Utilising the model, assess how well the trained model performed on the test dataset (held-out data).assess the function.

In Task-1 model-1 we have used 3 layers(1 output layer, 2 hidden layers). For the two hidden layers we are taking activation functions as **relu**. Relu is defined as f(x)=max(0,x), which means that it returns the input value if it's positive and zero if it's negative. ReLU reduces the vanishing gradient issue during training and is computationally efficient. Deep neural networks' hidden layers make extensive use of it. For the output layer, the activation function used is **softmax.** For multi-class classification problems, a neural network's output layer frequently uses the

activation function softmax. It converts a vector of raw scores (logits) over several classes into a probability distribution. The Softmax function normalises the numbers, calculates the exponentials of each score, and gives each class a probability. All probability added together equal one.The loss function **sparse_categorical_crossentropy** is used here**.** It measures how well the model is performing in terms of the difference between its predictions and the actual target values. Here, **sparse** indicates that the target values provided are integers. **Categorical** shows that there are numerous classes (in this case, ten apparel categories) and that the task is a multi-class classification task. **crossentropy** loss function is frequently utilised. The difference between the genuine class labels and the predicted class probabilities is quantified. Now we will train the model based on the training data. Here used epochs=10, One whole run through the whole training dataset is referred to as an epoch. The number of times the model will run through the full dataset during training is specified by this option. The model will be trained for ten epochs in this instance. **Batch_size** is taken as 64 , The dataset is split up into smaller batches for training. The amount of samples used in each weight update iteration depends on the batch size. 64 samples from the training set will be used in each iteration when the batch size is 64. **Verbose=0.2**: This argument controls the level of information displayed during training.

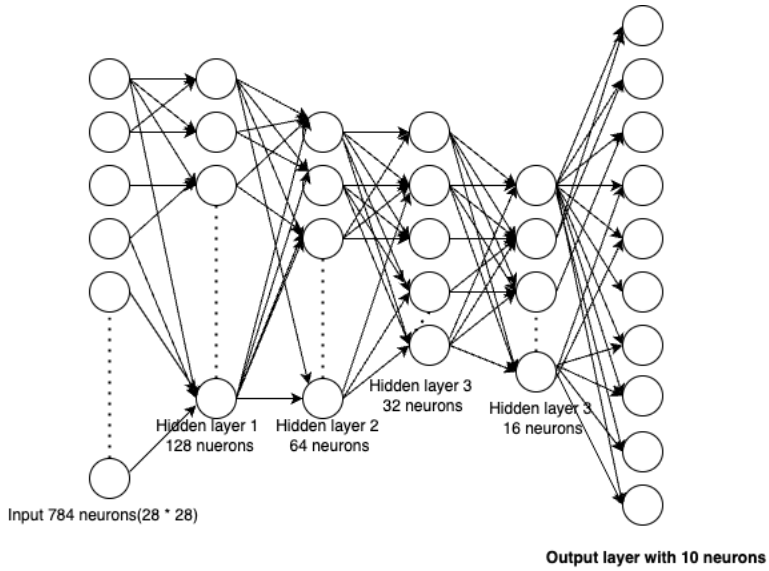
So, for model-1 the accuracy got is 87% with the above mentioned settings. Now in the model-2 to improve the network performance an extra dense layer is added with different set of nodes in each layer as shown in the code. Model-2 has an additional hidden layer with 32 neurons, making it more sophisticated than Model-1. A model that is more complicated may be able to identify more complex patterns in the data, which could result in improved performance. Model-2 is trained for 15 epochs, which is longer than the 10 epochs used for Model-1. The model can adjust its weights and possibly converge to a better answer with more training epochs. The particular features of the dataset may also have an impact on the model's performance. If the dataset has intricate patterns that the extra layer can identify, a more complicated model might function better. And the epochs values as 15. So, the accuracy is higher here i.e., 88%.

Model-2 Architecture



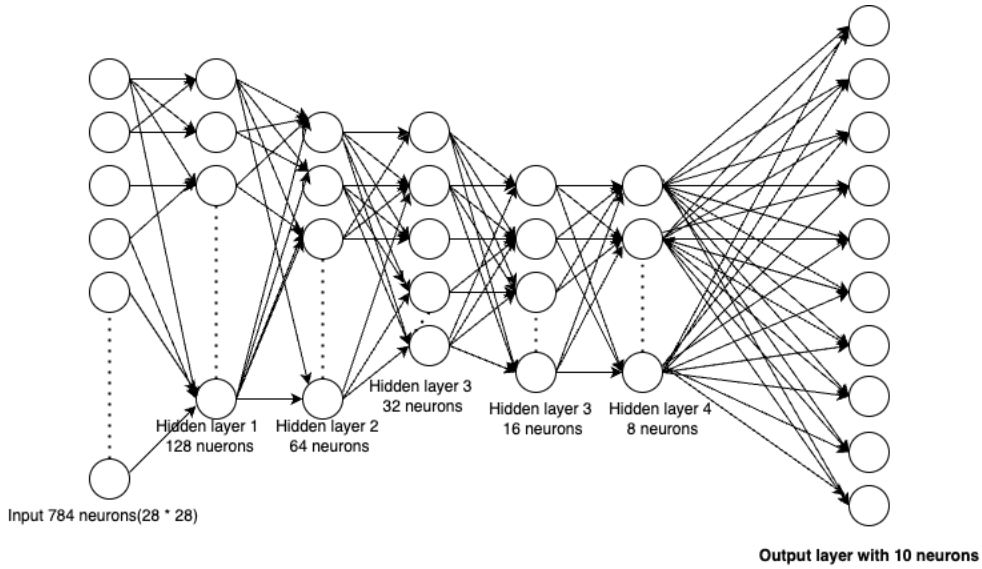
In the same way for the model-3 one more layer is being added with 16 neurons. The accuracy here is 88%. The architecture is as shown below :

Model-3 Architecture



In the same way for the model-4 one more layer is being added with 8 neurons. The accuracy here is 87.7%. An increase in the number of dense layers in a neural network can have a variety of unpredictable effects on performance with the risks of overfitting by which the model4 accuracy is decreased. The architecture is as shown below :

Model-4 Architecture



Task – 2 :

Here using the Fashion MNIST dataset, training a convolutional neural network (CNN) with the Keras toolkit and TensorFlow backend.

1. **Importing Libraries:**
   In order to build and train neural networks, the code first imports the required Python libraries and modules, such as NumPy for numerical operations and TensorFlow and Keras for neural network construction. Additionally, certain Keras components—such as layers, optimizers, and utilities—are imported by the code.
2. **Loading the DataSet:**
   The fashion_mnist.load_data() function is used to load the Fashion MNIST dataset. The grayscale photos in this dataset show various articles of apparel, including as shirts, trousers and shoes, and they are categorised into ten classes.
3. **Data Pre-processing:**
   Pre-processing of the input data takes multiple steps:

   Reshaping: To indicate one channel (grayscale), the data is reshaped from 28x28 images to 28x28x1 arrays. CNN input requires this.
   Data Type Conversion: To aid in better convergence during training, the pixel values are first transformed to floating-point numbers and then normalised to a range between 0 and 1.
   One-Hot Encoding: Class labels are converted to binary vectors via one-hot encoding. For problems involving multi-class classification, this encoding is crucial.
4. **Model Definition:**
   A CNN model is defined using the Sequential model from Keras. The model architecture consists of the following layers:
   - **Convolutional Layer :** ReLU activation functions utilise convolution operations to learn characteristics of an image using two 2D convolutional layers. The feature maps are additionally down sampled using max-pooling layers.
   - **Flatten Layer:** In order to prepare the data for fully connected layers, a flatten layer transforms the 2D feature maps into a 1D vector following feature extraction.
   - **Dense Layers:** In order to prepare the data for fully connected layers, a flatten layer transforms the 2D feature maps into a 1D vector following feature extraction.
   - **Output Layer:** Ten neurons with a softmax activation function, one for each class, make up the final dense layer. To anticipate the class label, Softmax gives each class a probability.

5. **Model Compilation:**
   The model is compiled with the following settings:
   - **Loss Function:** Many multi-class classification tasks use the categorical cross-entropy loss. The difference between the genuine class labels and the predicted class probabilities is quantified.
   - **Optimizer :** Stochastic Gradient Descent (SGD) is chosen as the optimizer. It is configured with a learning rate, momentum, decay, and nesterov (a variant of SGD).
   - **Metrics :** The metric used to evaluate the model's performance is accuracy.
6. **Model Training:**
   Using the training data, the model is trained via the model.fit method. The parameters used for the training are Number of Epochs, Batch Size, Veebosity.

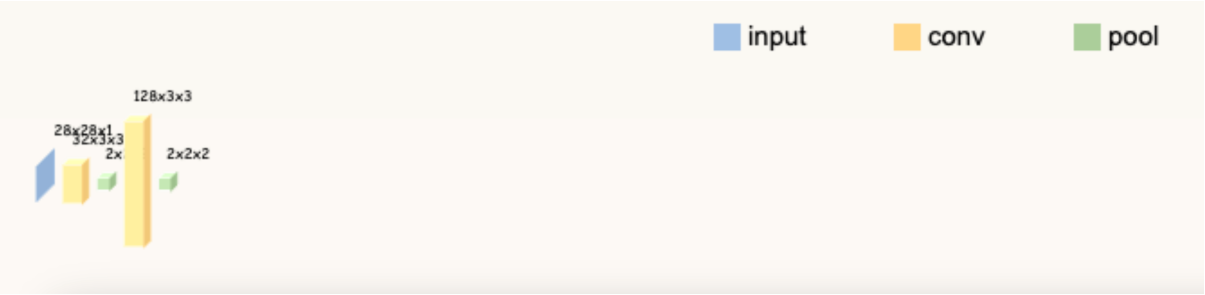
The code finishes the model training, which involves iterating across the network both forward and backward to update the weights and reduce loss.

To put it briefly, the code reads the Fashion MNIST dataset, pre-process it, creates a CNN model, compiles it, and then uses SGD as the optimizer to train the model. Accuracy is used to evaluate the model, and training progress is shown. The basis for building and training CNNs for image classification tasks is provided by this code.

Optimizer :
A well-liked optimisation technique for neural network training, stochastic gradient descent (SGD) is the optimizer utilised in the provided code. The step size at which the optimizer modifies the model's weights during training is based on the learning rate. Because the learning rate in this code is set to 0.002, each optimisation step's weights are modified by a factor of 0.002. One hyperparameter that adds a velocity component to the weight updates is called momentum. It facilitates the optimizer's more steady progress towards the steepest decline. Momentum is given a value of 0.7, which helps to keep the optimisation process stable. Over time, the learning rate is progressively decreased by the learning rate decay, sometimes referred to as a learning rate schedule. In the later phases of training, this can help with convergence and prevent overshooting. By dividing the initial learning rate (lrate) by the total number of training epochs (epochs), one may compute the decay. Model compilation is the context in which the optimizer is utilised. The selection of 'categorical_crossentropy' as the loss function is typical for jobs involving multi-class classification. The difference between the genuine class labels and the predicted class probabilities is quantified. Accuracy is the performance indicator used to assess the model; it quantifies the percentage of correctly identified samples.

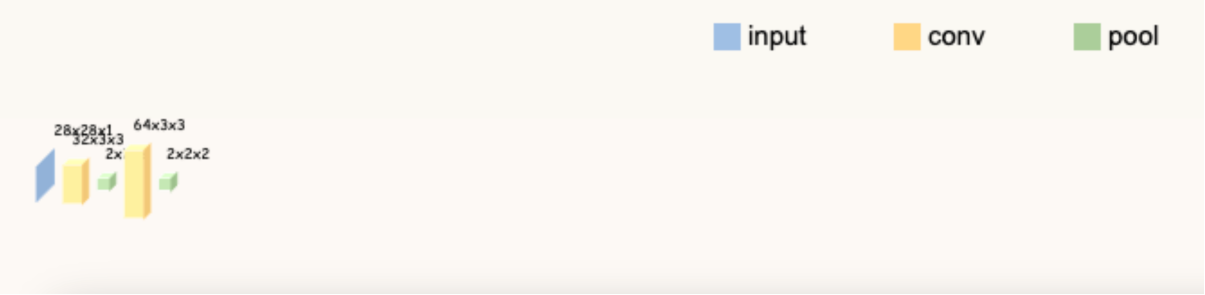
The accuracy for model-1 after training and testing the data is 77.77%.

As per the convolution layers added in the model-1 the architecture is as shown below:



Similarly, The accuracy for model-2 after adding the number of convolutional layers, the number of filters, or activation function in convolutional layers, is 79.37%.

As per the convolution layers added in the model-2 the architecture is as shown below:



Task – 3 :

In this task, the optimizer used in Task-1 model-2 was changed and the same number number of dense layers with same neurons were used. Here, **sgd_optimizer** is being used. By using this optimizer, the accuracy is decreased. The accuracy is 84% in Task-3 with different optimizer.

Similarly, the optimizer used in task-3 considering the Task-2 is **RMSprop**, with learning rate=0.001. The optimisation technique called RMSprop is frequently used to train neural networks, especially deep learning models. Some of the problems with the original stochastic gradient descent (SGD) algorithm are intended to be addressed by it. RMSprop has a reputation for being able to individually adjust the learning rates for each weight parameter, which can result in enhanced training stability and quicker convergence. The key features of RMSprop are Adaptive Learning rates, Damping factor, Running average of squares, Calculation if weight updates, Learning rate schedule. In the code upon instantiation, RMSprop has a learning rate of 0.001. One can modify the learning rate as a hyperparameter according to the dataset and particular situation which are facing. The optimizer is then employed in the compilation of the model, where it is in charge of making updates to the model's weights throughout the training process.

RMSprop is a popular and reliable optimizer that performs well in a range of deep learning applications. Because of its reputation for handling noisy and non-stationary gradients, it is a useful option for efficiently training neural networks. Here the same convolutional layers were being used as that of in model-2. Then the accuracy is increased from 79.39% to 90.72%.

Accuracy	Model-1	Model-2	Model-3	Model-4
Task-1	87.8%	88.2%	88%	87.7%
Task-2	77.77%	79.39%		

Accuracy	w.r.t Task-1	w.r.t task-2
Task-3	84.6%	90.72%

Ranking of the Neural network performances :

1. Task-3- CNN(Task-2)
2. Task-1 Model-2 NN
3. Task-1 Model-3 NN
4. Task-1 Model-1 NN
5. Task-1 Model-4 NN
6. Task-3-NN(Task-1)
7. Task-2 Model-2 CNN
8. Task-2 Model-2 CNN

References :

1. **Online Courses and Tutorials:**
   - Deep Learning Specialization on Coursera by Andrew Ng.
   - Stanford University's CS231n: Convolutional Neural Networks for Visual Recognition.
   - Youtube : Deeplizard Deep learning courses. (https://www.youtube.com/@deeplizard)
2. **Website and Online resources:**
   - TensorFlow and Keras Official Documentation.
   - arXiv.org