Multi-class Classification with One, Multiple hidden layers and with Convolutional Neural Networks

Neelesh K Bhajantri
Department of Computer Science
University at Buffalo
Buffalo, New York 14260
neeleshb@buffalo.edu

Abstract

In this project we have used 3 different methods to classify the multi class problem. Those three methods are – Neural networks with One Hidden layer (code from scratch), Multi-Layer Neural Network and Convolutional Neural Network. We have the achieved the accuracy of 84.7%, 89.1%, 83% for the respective 3 methods that we have used.

1 Introduction

A feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle. As such, it is different from recurrent neural networks. The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function.

2 Dataset

For training and testing of our classifiers, we will use the Fashion-MNIST dataset. The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.

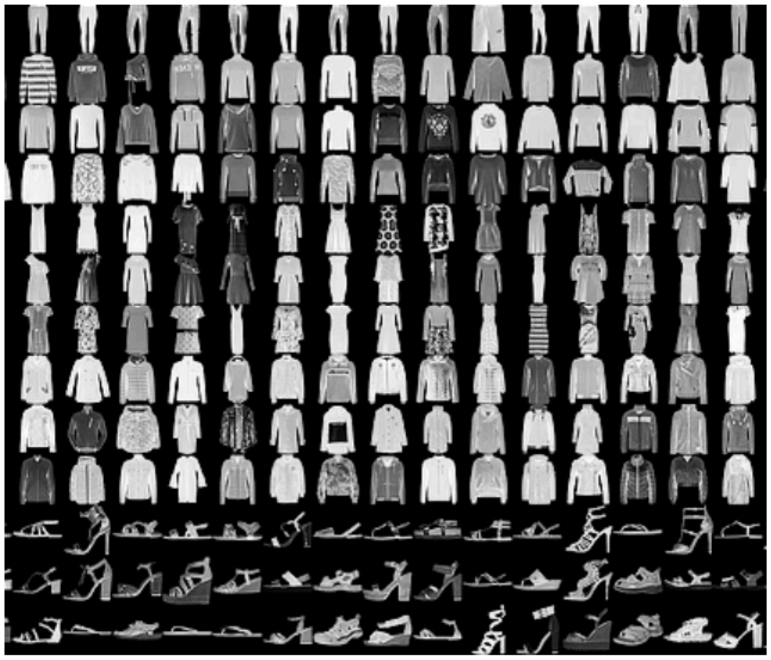


Figure 1: Example of how the data looks like.

The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

3 Preprocessing

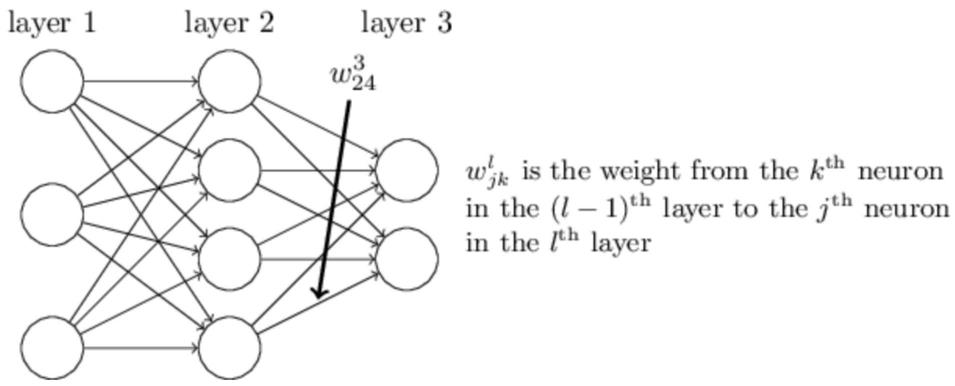
We just had to reshape the data of x train and y train elements so that we can perform dot operations on further processes. Since the data set has a pixel value to simply normalize the values all around, we divide the values by 255. We know that the pixel values for each image in the dataset are unsigned integers in the range between black and white, or 0 and 255.
We do not know the best way to scale the pixel values for modeling, but we know that some scaling will be required. A good starting point is to normalize the pixel values of grayscale images, e.g. rescale them to the range [0,1]. This involves first converting the data type from unsigned integers to floats, then dividing the pixel values by the maximum value. Typically, a neural network's parameters are initialized (i.e. created) as small random numbers. Neural networks often behave poorly when the feature values much larger than parameter values. Furthermore, since an observation's feature values will be combined as they pass through individual units, it is important that all features have the same scale.
For these reasons, it is best practice (although not always necessary, for example when we have all binary features) to standardize each feature such that the feature's values have the mean of 0 and the standard deviation of 1.

4 Architecture

4.1 Gradient Descent

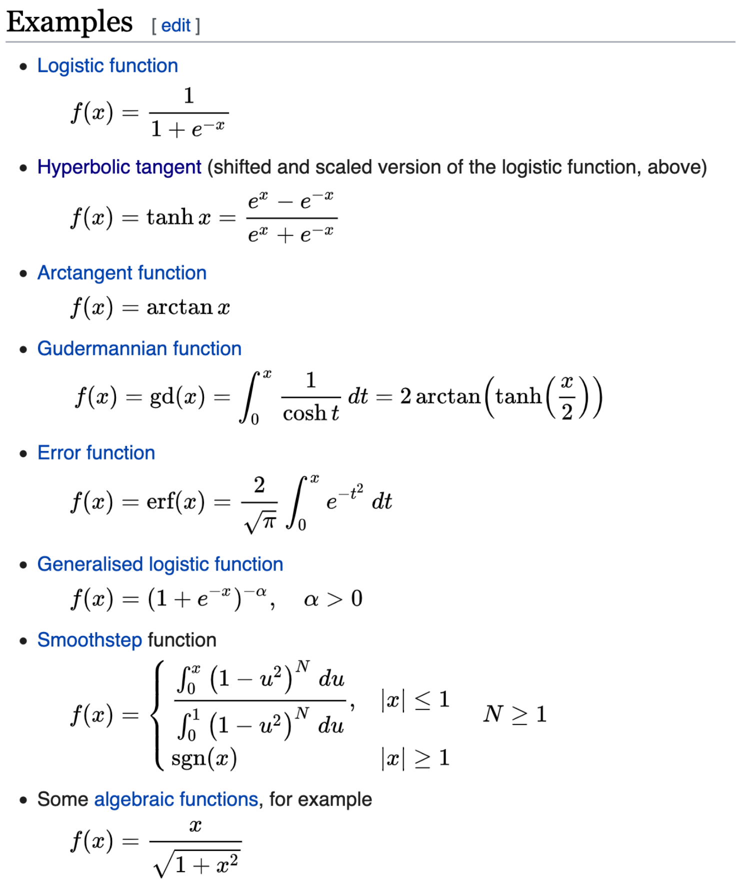
Before discussing backpropagation, let's warm up with a fast matrix-based algorithm to compute the output from a neural network. We actually already briefly saw this algorithm near the end of the last chapter, but I described it quickly, so it's worth revisiting in detail. In particular, this is a good way of getting comfortable with the notation used in backpropagation, in a familiar context.

Let's begin with a notation which lets us refer to weights in the network in an unambiguous way. We'll use w^l_{jk} to denote the weight for the connection from the kth neuron in the (l−1)th layer to the jth neuron in the lth layer. So, for example, the diagram below shows the weight on a connection from the fourth neuron in the second layer to the second neuron in the third layer of a network:


layer 1
layer 2
layer 3
w^3_{24}
w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

4.2 Sigmoid Function

A sigmoid function is a bounded, differentiable, real function that is defined for all real input values and has a non-negative derivative at each point.[1] A sigmoid "function" and a sigmoid "curve" refer to the same object.



Examples [edit]

- Logistic function

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

- Hyperbolic tangent (shifted and scaled version of the logistic function, above)

$$f(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Arctangent function

$$f(x) = \arctan x$$

- Gudermannian function

$$f(x) = \operatorname{gd}(x) = \int_0^x \frac{1}{\cosh t}\,dt = 2\arctan\left(\tanh\left(\frac{x}{2}\right)\right)$$

- Error function

$$f(x) = \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt$$

- Generalised logistic function

$$f(x) = (1+e^{-x})^{-\alpha}, \quad \alpha > 0$$

- Smoothstep function

$$f(x) = \begin{cases} \dfrac{\int_0^x \left(1-u^2\right)^N du}{\int_0^1 \left(1-u^2\right)^N du}, & |x| \le 1 \\ \operatorname{sgn}(x) & |x| \ge 1 \end{cases} \qquad N \ge 1$$

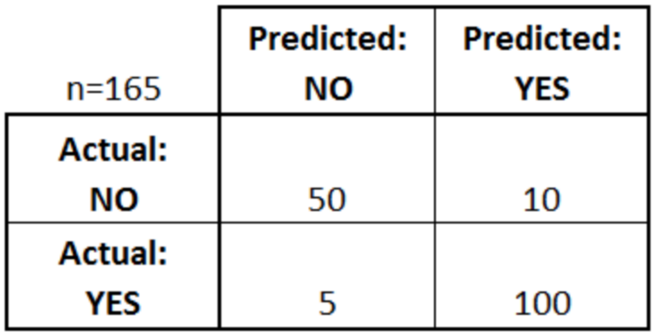
- Some algebraic functions, for example

$$f(x) = \frac{x}{\sqrt{1+x^2}}$$

4.3 Confusion Matrix

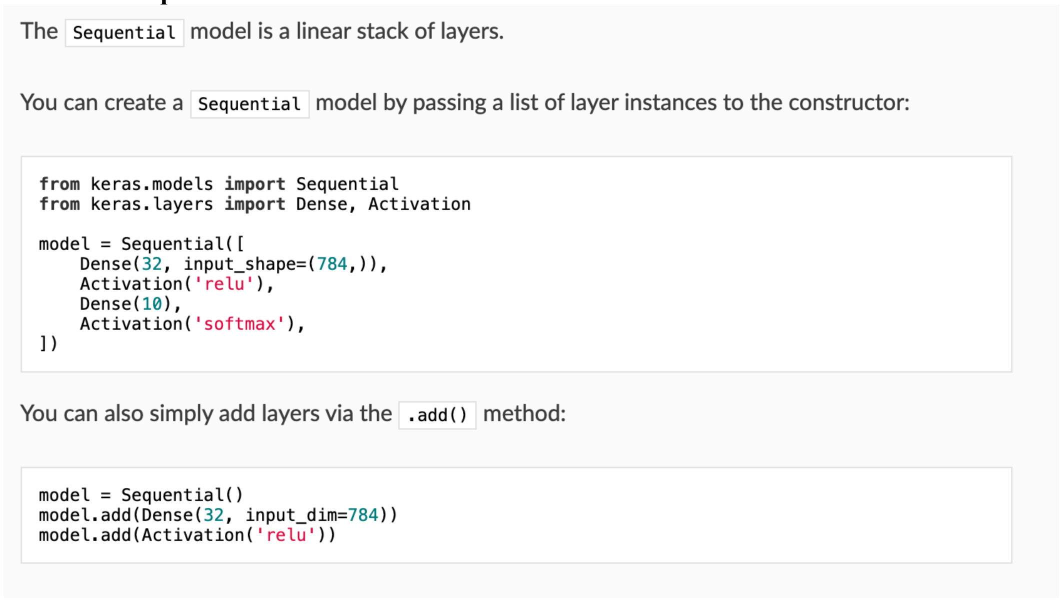
In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix,[5] is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).[2] The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

It is a special kind of contingency table, with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table).



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

4.4 Keras Sequential Model



The `Sequential` model is a linear stack of layers.

You can create a `Sequential` model by passing a list of layer instances to the constructor:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

You can also simply add layers via the `.add()` method:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

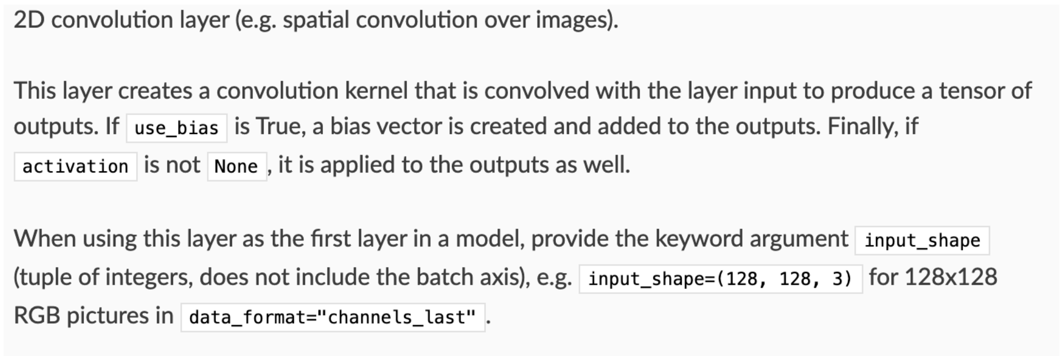
We use the sequential model for multi-layer neural network and achieve accuracy of 83%. RMSprop— is unpublished optimization algorithm designed for neural networks, first proposed by Geoff Hinton in lecture 6 of the online course “Neural Networks for Machine Learning”. RMSprop lies in the realm of adaptive learning rate methods, which have been growing in popularity in recent years, but also getting some criticism. It’s famous for not being published, yet being very well-known; most deep learning framework include the implementation of it out of the box.
There are two ways to introduce RMSprop. First, is to look at it as the adaptation of rprop algorithm for mini-batch learning. It was the initial motivation for developing this algorithm. Another way is to look at its similarities with Adagrad and view RMSprop as a way to deal with its radically diminishing learning rates. I’ll try to hit both points so that it’s clearer why the algorithm works.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\delta C}{\delta w}$$

As you can see from the above equation, we adapt learning rate by dividing by the root of squared gradient, but since we only have the estimate of the gradient on the current mini-batch, we need instead to use the moving average of it. Default value for the moving average parameter that you can use in your projects is 0.9.

4.5 Keras with Conv2D


2D convolution layer (e.g. spatial convolution over images).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the batch axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.

5. Define Model

Next, we need to define a baseline convolutional neural network model for the problem.
The model has two main aspects: the feature extraction front end comprised of convolutional and pooling layers, and the classifier backend that will make a prediction.
For the convolutional front-end, we can start with a single convolutional layer with a small filter size (3,3) and a modest number of filters (32) followed by a max pooling layer. The filter maps can then be flattened to provide features to the classifier.
Given that the problem is a multi-class classification, we know that we will require an output layer with 10 nodes in order to predict the probability distribution of an image belonging to each of the 10 classes. This will also require the use of a softmax activation function.
Between the feature extractor and the output layer, we can add a dense layer to interpret the features, in this case with 100 nodes.
All layers will use the ReLU activation function and the He weight initialization scheme, both best practices.
We will use a conservative configuration for the stochastic gradient descent optimizer with a learning rate of 0.01 and a momentum of 0.9. The categorical cross-entropy loss function will be optimized, suitable for multi-class classification, and we will monitor the classification accuracy metric, which is appropriate given we have the same number of examples in each of the 10 classes.

5.1 Relu

In the context of artificial neural networks, the rectifier is an activation function defined as the positive part of its argument:

{\displaystyle f(x)=x^{+}=\max(0,x),}{\displaystyle f(x)=x^{+}=\max(0,x),} where x is the input to a neuron. This is also known as a ramp function and is analogous to half-wave rectification in electrical engineering.

This activation function was first introduced to a dynamical network by Hahnloser et al. in 2000 with strong biological motivations and mathematical justifications. It has been demonstrated for the first time in 2011 to enable better training of deeper networks, compared to the widely used activation functions prior to 2011, e.g., the logistic sigmoid (which is inspired by probability theory; see logistic regression) and its more practical counterpart, the hyperbolic tangent. The rectifier is, as of 2017, the most popular activation function for deep neural networks.

A unit employing the rectifier is also called a rectified linear unit (ReLU).

5.2 SoftMax

In mathematics, the softmax function, also known as softargmax[1] or normalized exponential function,[2]:198 is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval {\displaystyle (0,1)}(0,1), and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities. Softmax is often used in neural networks, to map the non-normalized output of a network to a probability distribution over predicted output classes.

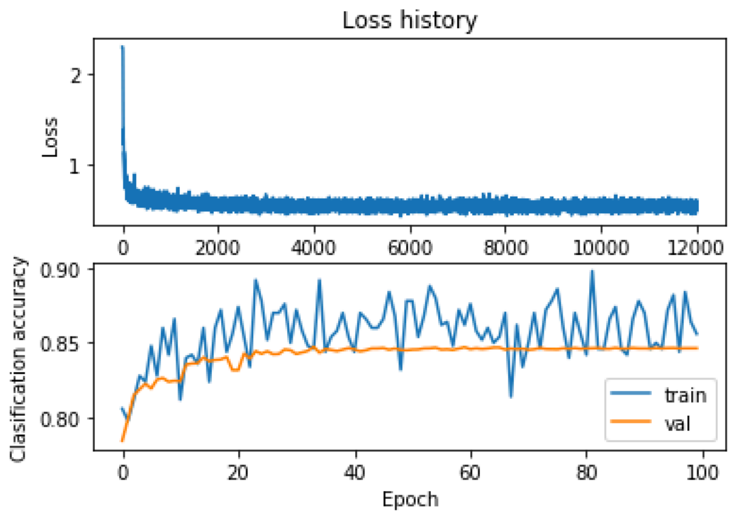
The standard (unit) softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is defined by the formula

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

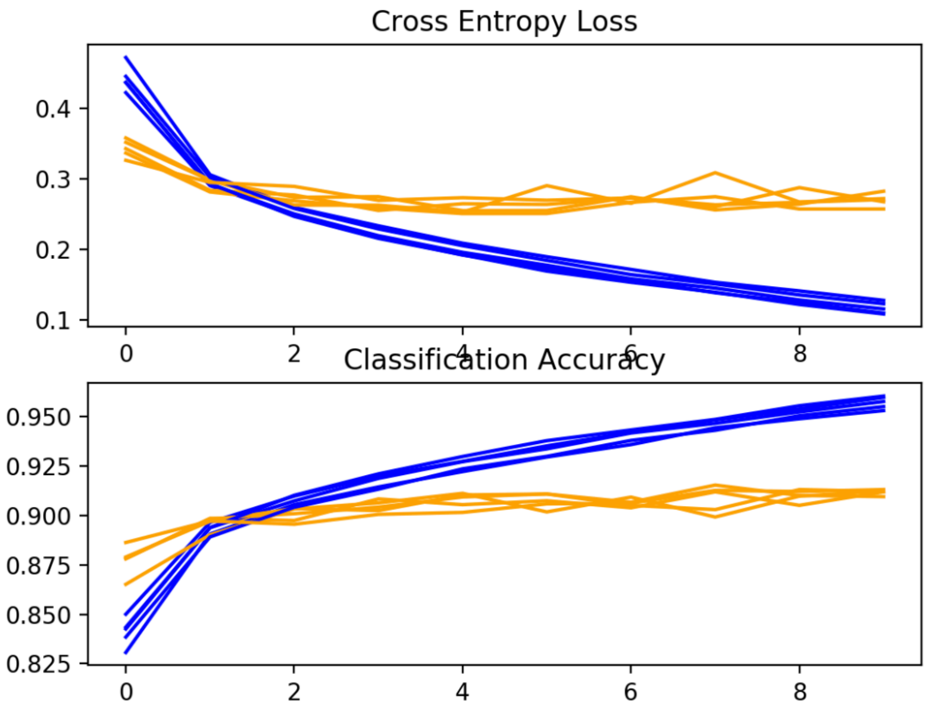
6 Results

We analyze that the convolutional neural network model works very well when compared to the time and the inputs that are given to the model. Because of the way CNN is designed it will form layers and compress the layers to its own redundancy and decreases as the weights and the biases go on.

Here is the graph that is plotted by the Single Hidden layer Network



Here is the graph that we achieved while using Convolutional Neural Networks



7 Conclusion

By observing the results from the different methods used in developing a multi class classification model we can easily decide that CNN has an upper hand in developing a model. As the iterations progress in training the model the weights and biases are replaced over and over depending on the loss value that we are receiving. This helps a lot in changing the learning rate and number of epochs. The CNN model achieves 83% accuracy in just 30 iterations where are the single hidden layer has taken 1000 iterations to achieve a good accuracy.