Project 3 - FYS-STK4155

Oda Langrekken

December 18, 2022

Abstract

We classify pieces of clothing using the Fashion MNIST dataset. Our goal is to explore whether neural networks can classify images with higher accuracy than the simpler logistic regression. As an alternative to logistic regression we explore both regular neural networks and convolutional neural networks. We discover that neural networks perform better than logistic regression, as the accuracy for the neural networks are 93% for the convolutional one and 88% for the regular one, while logistic regression achieves an accuracy of 67%. Additionally we discover that the code we have written ourselves is quite slow for larger data sets, and learn to truly appreciate the speed up one can gain from using more batches in stochastic gradient descent and also the computational efficiency of the state-of-the-art python libraries `sklearn`, `tensorflow` and `keras`. Based on this we conclude that, while intimidating in the large number of hyperparameters that can be tuned, neural networks are truly worhty of the hype when we have larger and more complicated data sets.

Github repository: `https://github.com/OdaLangrekken/FYS-STK4155/tree/main/Project3`

1 Introduction

Computers excel at numerical computations, greatly outperforming humans in speed. Thus it is no surprise that machine learning methods are highly performant in tasks such as polynomial fitting, where the goal is to fit the best to curve to numerical data. Image classification is thus a very interesting field in machine learning, as no evident numerical computation is required for humans to classify an object that we see with our eyes. Without any effort we can distinguish a cat from a dog, and tell if a written number is a 1 or a 9. Problems such as image classification inspired the development of neural networks, where the initial goal was to make computers 'think' more like humans. With machine learning methods we have built models that classify images with high accuracy.

The MNIST dataset, which contains images of handwritten digits between 0 and 9, is a popular benchmark dataset for image classification. Several publications, such as [1], have achieved an accuracy of more than 99%. In this project we will spice things up a bit by using the Fashion MNIST [2] dataset. The Fashion MNIST is similiar to MNIST in that it contains images of objects with 10 different output classes, but instead of digits the images show pieces of clothing.

Our goal is to explore whether neural networks outperform simpler methods when classifying images. We will compare a home-made multinomial logistic regression with neural networks from the state-of-the-art python libraries `sklearn` and `tensorflow`. As we consider image data, we will explore the convolution neural network in addition to a regular neural network. As the Fashion MNIST is a larger dataset we will also look into how we can speed up the training time, escpecially for our home-made logistic regression class.

In section 2 we introduce the methods that are used for classification. Logistic regression is the first method that we discuss. Then we move on to neural networks, and give a very brief introduction to convolutional neural networks. Section 3 introduces the Fashion MNIST dataset. All results are presented in section 4, where we study the performance of the logisitc regression and the neural networks. Finally the conclusion in section 5 offers some final remarks, and ideas for further study.

2 Classification Methods

In general we can summarize the goal of supervised machine learning as follows: given some data $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p)$ we wish to find a functional relationship between $\mathbf{X}$ and the output $\mathbf{y}$ by minimizing a loss function $L(\mathbf{y}, f(\mathbf{X}))$. [3] When the output data is discrete, we have a classification task. Among two or more possible classes the classification model should assign the class that is most probable given the input.

An example of a classification problem is the Wisconsin Breast Cancer data, which we studied in the previous project [4]. In this data set there were two possible classes; 0 (no cancer) and 1 (cancer).

A popular error metric for classification the accuracy score

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \tag{1}$$

where $I(t_i = y_i)$ is 1 when $t_i = y_i$ and 0 otherwise.

2.1 Logistic Regression

Logistic regression is a popular method for classification. Let us assume that we have a problem with two possible outcome classes. In such cases we usually code the output such that a value $y_i = 0$ corresponds to the first class and $y_i = 1$ corresponds to the second class. The output of the model is a number between 0 and 1, and gives the probability to belong to the second class. One could e.g. choose a threshold of 0.5, such that all predicted output values $\hat{y}_i \geq 0.5$ are assigned the second class.

The method is similar to linear regression, and the model is trained by fitting coefficients $\boldsymbol{\beta}$. In order to get the probability that the input belongs to the second class, the product $\mathbf{X}\boldsymbol{\beta}$ is transformed to the range [0, 1] by using the sigmoid function

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{2}$$

where $\mathbf{z} = \mathbf{X}\boldsymbol{\beta}$.

The loss function to be minimized is often the cross entropy [5]

$$C(\beta) = \sum_{i=1}^{N} \{y_i \beta^T x_i - \log(1 + \exp(\beta^T x_i)\} \tag{3}$$

The coefficients that minimize the loss function can be found by using gradient methods that move the coefficients in the direction of the greatest decrease in error.

2.1.1 Multinomial logistic regression

When we have multiple possible output classes a possible approach is to train a logistic regression model for each class. Imagine we have C different classes. For each class c the training output is assigned the value 1 if y_i belongs to c and 0 otherwise. A model is trained that finds the probability $P(y_i = c|X)$. In the end we have C different models, each predicting the probability that the input corresponds to one of the possible classes. The prediction is the class with the highest probability among all models.

2.2 Neural Networks

The backbone of a neural network is the node, or artificial neuron [3]

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b_i\right) \tag{4}$$

where y is the output of the neuron, $\mathbf{x}$ are the input data, and $\mathbf{w}$ and $\mathbf{b}$ are the weights and biases of the model respectively. Here f is knows as the activation function, and its purpose is to

alter the input data in some way so as to get the desired output. The sigmoid function defined in equation 2 is an example of such an activation function. Another choice of activation function is the ReLU, defined as

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

A neural network consists of many such nodes. When the network has several layers of node, it is called a multilayer neural network. If the network has many layers, it is known as a deep neural network. Such networks are the basis of deep learning.

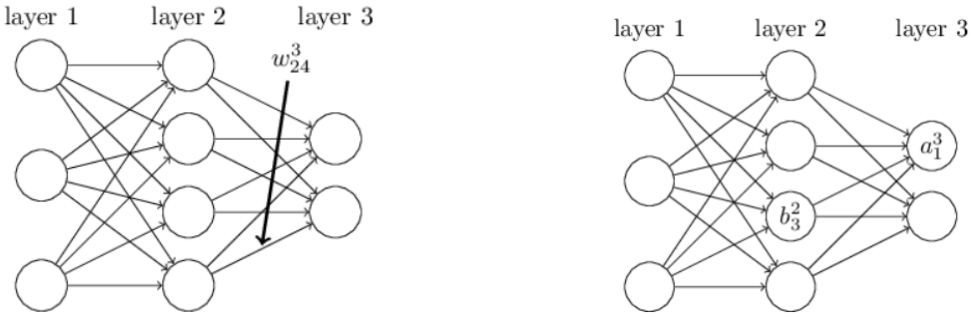

Figure 2.1: Notation and example layout of a neural network, from [6]

Figure 2.1 shows a possible architecture for a neural network. The first layer is knows as the input layer, and corresponds to the input data. The data in the first layer is propagated through the network by calculating the activation for each node:

$$a^l_j = f\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right) \tag{6}$$

The optimal weights and biases are chosen by finding the values that minimize the cost function. As we have multiple layers, the error is propagated backwards in the network in order to update all the weights and biases. This algorithm is known as backpropagation. Let us call the activation function f and C is the cost function. Then the error in the final layer L is [3]

$$\delta^L_j = f'(z^L_j)\frac{\partial C}{\partial(a^L_j)}$$

We can then fid the error for all other layers $l = 1, 2, \cdots, L$ by

$$\delta^l_k = \sum_k \delta^{l+1}_k w^{l+1}_{kj} f'(z^l_j)$$

The weights and biases are updated according to

$$w^l_{jk} \leftarrow w^l_{jk} - \eta \delta^l_j a^{l-1}_k$$

$$b^l_j \leftarrow b^l_j - \eta \delta^l_j$$

where η is the learning rate.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNS) are a type of neural network specifically desgined to classify images. The input to the CNN is 3-dimensional, where the three dimensions are the height of the image, the width of the image and the color. To decrease the number of weights the nodes only output to some of the nodes in the next layer, as shown in figure 2.2.

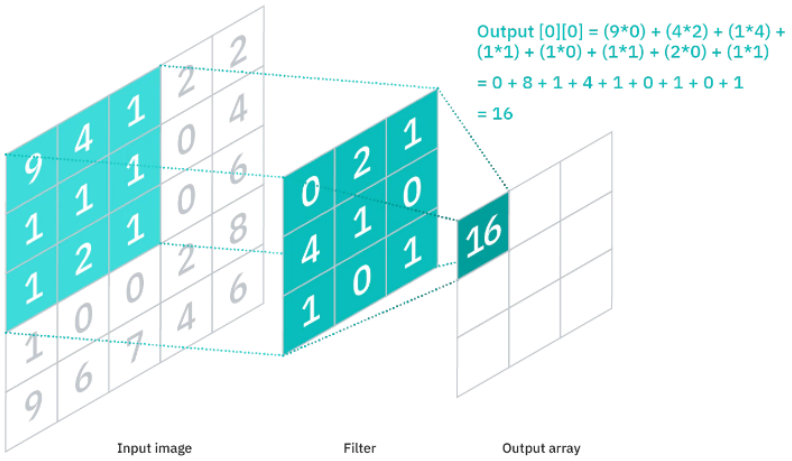

Figure 2.2: A convolutional neural network, taken from [7]

A layer that computes the output for a input region is called a convolutional layer. In addition a CNN can contain pooling layers that reduce the dimensions of the output, layers that flatten the output to one dimension and 'regular' network layers with activation functions such as the ReLU that take the flat output from the flattening layers.

3 Data

We use the Fashion MNIST dataset [2]. The dataset contains 28X28 pixel grayscale images of pieces of clothing from Zalando. Each row in the dataset is one image. The first column is the label of the clothing article. Table 3.1 shows what label is assigned to each piece of clothing.

Label	Clothing article
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Table 3.1: Table that shows how each piece of clothing is assigned an integer number as label in the Fashion MNIST dataset.

The remaining 784 columns are the pixel-values of each pixel in the images. Each pixel has an integer value between 0 and 255. Figure 3.1 shows the first 25 images in the train set. The title of each image is the name of the class.

Figure 3.1: The first 25 images in the Fashion MNIST train set. The title of each image is the class name of the image label, according to table 3.1

4 Results and Discussion

4.1 Logistic Regression

Our first method is logistic regression. There are ten different output classes in the Fashion MNIST data set, which means we train ten different logistic regression models to perform the predictions.

Figure 4.1 shows the train and validation accuracies as functions of the number of epochs. Stochastic gradient descent with 10 minibatches is used for training.

We see that the accuracy seems to converge at about 0.65 (in average) after 50 iterations of gradient descent. As we have 10 classes we would expect an accuracy of 0.10 (assuming classes are equally represented). An accuracy of 0.65 thus shows that the method is able to classify a lot of the images correctly.

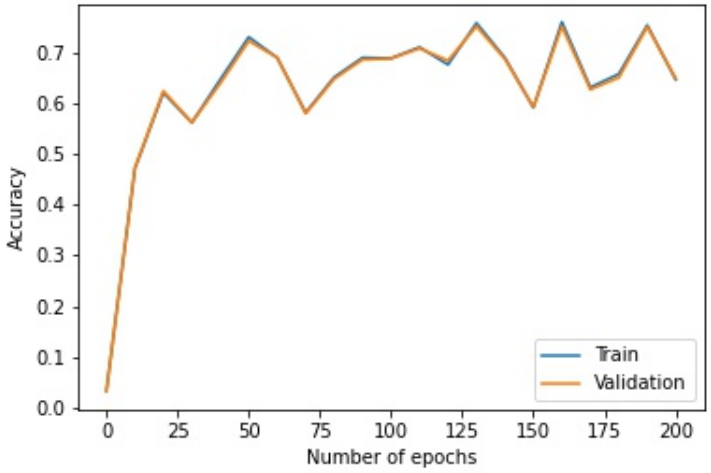

Figure 4.1: Train and validation loss as a function of epochs when using logistic regression for classification. Stochastic gradient descent with 10 minibacthes is used to find the best coefficients.

When training the model it is immediately noticeable that the training is rather slow for larger numbers of iterations. We use stochastic gradient descent to find the best coefficients. In stochastic gradient descent the training is only performed on a subset of the data in each iteration. This speeds up training, as well as making the model less sensitive to choice of initial coefficients. The size of the subsets that the model is trained on is determined by the number of mini batches. So far we have used 10 mini batches, which means that the data is split into 10 subsets and that the model is trained on 1/10 of the data at each iteration. Increasing the number of minibatches will of course speed up the training when we have a pre-determined number of iterations, as the training is performed on a smaller set of data at each iteration. We should be careful not to choose a too large number of minibatches though, as training on a too small set of data at each iteration will worsen the model performance.
In table 4.1 we can see the CPU time and the train error for different number of minibatches. As expected the run time of the code decreases when we increase the number of minibatches. Additionally the train accuracy improves when we do not train on the whole dataset. This shows that the model is more robust when we do not include all data points at each iteration.

Number of minibatches	CPU time (s)	Train accuracy
1	227.5	0.59
2	115.6	0.59
10	27.9	0.73
24	14.6	0.68

Table 4.1: CPU time and train accuracy for different numbers of minibatches in stochastic gradient descent.

Based on our previous result we choose a model with 10 minibatches and train it for 100 epochs to get our final test score. The test accuracy for this model is 67%.

Figure 4.2 shows the predicted and true labels for the first nine images in the train data when using the model with 100 epochs and 10 minibatches. Except for the first image, the predictions are all correct.

Figure 4.2: Predicted and true labels when using logistic regression.

4.2 Neural network

Our next model is the neural network. The `MLPClassifier` from `sklearn` was used. For optimization we chose stochastic gradient descent.

When creating a neural network you need to decide on the architecture of the network, i.e. how many layers the network should consist of and how many nodes there should be in each layer. To get a feeling of how the architecture affects the performance of the network, we trained three separate networks. One with 2 hidden layers with 50 nodes in each layer, one with 2 hidden layers with 100 nodes in each layer and one with 3 layers with 100 nodes in each layer. The validation accuracies are plotted in figure 4.3. We see that the difference in accuracy is not substantial between the three networks. The network with three layers seems to reach a high accuracy sooner, which is to be expected as we get more explaining power with a deeper network. They do however all seem to converge to an accuracy of about 87/88%.
The networks use early stopping, which means that they stop training if the increase in validation accuracy is lower than some threshold. Here we see that only the network with 50 nodes continues training for all 100 epochs, so we can conclude that this network has not yet converged.

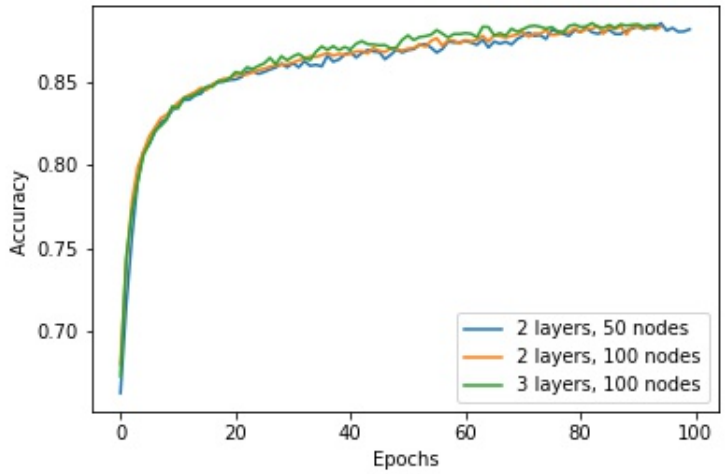

Figure 4.3: Validation accuracy for different network accuracies.

Figure 4.4 shows the confusion matrix, i.e. the true vs predicted labels. Wee see that the several misclassifications are 'understandable', as these are clothing articles that also look similar to the human eye. As an example we see that 152 shirts are classified as T-shirts/top.

	T_shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T_shirt/top	837	1	16	43	2	3	86	0	12	0
Trouser	3	974	3	16	1	2	1	0	0	0
Pullover	16	2	838	11	77	0	50	0	6	0
Dress	27	15	12	912	18	2	14	0	0	0
Coat	2	0	93	36	807	0	59	0	3	0
Sandal	0	0	0	0	0	936	1	42	3	18
Shirt	152	3	94	34	62	0	644	0	11	0
Sneaker	0	0	0	0	0	28	0	931	0	41
Bag	7	0	7	3	4	5	8	5	958	3
Ankle boot	0	0	0	0	0	13	0	42	1	944

Figure 4.4: Confusion matrix for the network with two hidden layers and 100 nodes in each layer.

To get a final estimate of the prediction error, we chose the network with 2 hidden layers and 100 nodes. For this network the test accuracy is 88%, which is significantly better than the 67% we achieved with logistic regression.

4.3 Convolutional neural network

An image is 2-dimensional. Logistic regression and regular neural networks take 1-dimensional input, so we flattened out the images to 1-dimensional vectors. Consequently information is lost, as neighboring pixels are no longer close to each other. Convolutional neural networks are specifically desgined to analyze images, and information about the closeness of pixels is not lost.

To build a CNN we used `keras` and `tensorflow`. The data was prepared by reshaping it so that it is a 28x28 matrix rather than a vector with 784 elements.

In figure 4.5 we see the train and validation accuracies for the CNN. The network seems to overfit, as the train accuracy is higher than the validation accuracy.

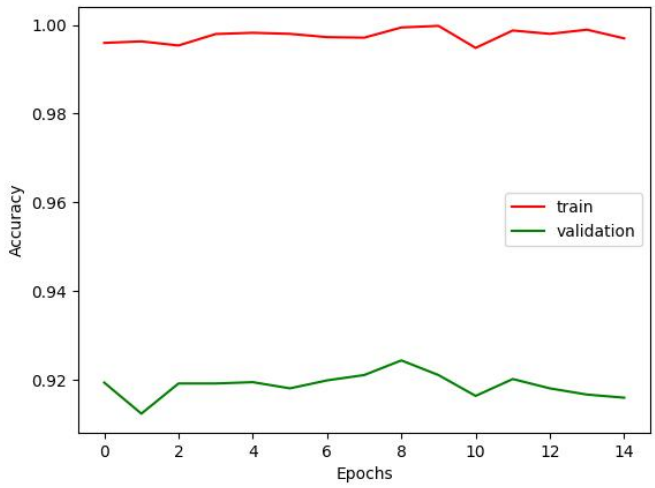

Figure 4.5: Train and validation accuracies for CNN.

For the CNN we achieved a test accuracy of 93%.

5 Conclusion

We have compared the performance of logistic regression to that of neural networks and convolutional neural networks when the task is to classify images in the Fashion MNIST data set. In addition we explored how to speed up the logistic regresison when we have a lot of data.

We discovered that neural networks, and especially convolutional neural networks, outperform logistic regression when classifying images. The test accuracies were 67%, 88% and 93% for logistic regression, neural network and convolutional neural network respectively. Thus we conclude that more complex methods outperform simpler methods when used on complex data, provided that we take care in choosing the methods that are best suited to our data. As convolutional neural networks are designed to classify images, they were a good choice of method for this problem, and it is not surprising that they outperformed the other models. Very little hyperparameter tuning was performed for the neural networks, and with more effort put into maximizing their performance we could probably reach an even higher accuracy.

Furthermore we saw how using stochastic gradient descent instead of regular gradient descent make the training of logistic regression faster. It would be very interesting to try to optimize the code even further. When analyzing data you can often encounter large data sets. Being able to speed up model training and write efficient code is a must in the 'Big Data' age.

So far in this course I have been impressed my how well the methods I have coded myself have performed. On this data set we clearly saw that they are not as efficient as sklearn. Regardless I have greatly enjoyed getting to code some of the famous methods myself, and I believe I have learned a lot from it.

References

[1] G. Jha and H. Cecotti, "Data augmentation for handwritten digit recognition using generative adversarial networks," *Multimedia Tools and Applications*, vol. 79, 12 2020.

[2] Kaggle, "Fashion mnist." `https://www.kaggle.com/datasets/zalando-research/fashionmnist`. Accessed: 2022-12-05.

[3] M. Hjorth-Jensen, "Applied data analysis and machine learning." `https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html`, 2021.

[4] O. Langrekken, "Project 2." `https://github.com/OdaLangrekken/FYS-STK4155/blob/main/Project2/report/FYS_STK4155___Project2.pdf`, 2022.

[5] J. F. Trevor Hastie, Robert Tibshirani, *The Elements of Statistical Learning*. Springer New York, NY, 2 ed., 2009.

[6] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[7] IBM, "Convolutional neural networks." https://www.ibm.com/cloud/learn/convolutional-neural-networks. Accessed: 2022-12-18.