Eindopdracht

Mohammed Al Hor

2023-05-13

Exercise 1. Hands-on experience [20 pt]

(a) Report your accuracy score (provided to you by the web application).

My accuracy score is 0.1440543.

(b) While accuracy is important, the goal is to gain hands-on experience. Summarize your actions, how did you start? what did you try?

I did this exercise in Python, because I wanted to play around with KerasTuner (https://keras.io/keras_tuner/). I started off with a simple model. 3 layers (different number of neurons ranging form 8 to 256), 'relu' activation function and 'Adam' optimizer. I also tried different epochs. After spending a lot of time manually tuning the model I couldn't get anywhere near the scores I saw fellow students post in the Whatsapp group. I struggled with heuristic nature of getting my model to perform better. I then learned about KerasTuner, and this is what I used to get the final model with a score of approximately 0.14. I dit the following to obtain these results:

1. I wanted to use 3 layers, based on existing research on similar datasets
2. Each layer should consists of at least 32 neurons and should increase stepwise by 32, with a max of 512.
3. The model should use the 'relu' activation function.
4. Adam optimizer.
5. I also implemented a dropout rate after every hidden layer to counter overfitting. Minimum value of 0.1, maximum value of 0.5.
6. Lastly, I manually added early stopping, because this seemed to increase accuracy.

The hyperparamters of the final model where as follows.

Number of epochs: 100
Batch size: 2^6
Hidden layers: 3
Neurons per layer: 369, 269, 81
Activation function: Relu
Regularization: Early stopping with patience of 5
Optimizer: Adam optimizer

Increasing the number of neurons per layer, the choice in optimizer had the most significant impact on performance of the model. I used the RMSL metric on the test data to

compare the different models. What I took away from this exercise is that getting the perfect model is kind off messy, you have to try lots of different hyperparameters, while keeping others constant. It is very fun, once you get the hang of it and you get a grasp for which nobs to turn to get better results. Fun exercise!

Exercise 2. Training deep learning models [20 pt]

(a) Explain in your own words what do $g(\cdot)(x)$ represent?

I like to think of this as a function that takes the input values on the left, does some mathematical operations on them and extracts useful patterns from the input data (by tweaking the weights of the inputs etc.). The specific transformation performed by this neuron is determined by the activation function associated with the neuron (relu, sigmoid etc.). These activation functions introduce non-linearity to the computations of the neuron and by doing this we can capture even the most complex of relationships between inputs and the output.

(b) Explain in your own words what does $f(g(x))$ represent?

In the most simplest of terms; this obviously represents the final output of the neural network, as a result of applying a transformation function f() to the outputs of the hidden layers represented by g(x). To go in to more detail; the hidden layer performs computations on the input data to extract meaningful features. Each neuron in this hidden layer g(x) contributes to this. The specific computations performed by each neron are determined by their own activation functions. Once the inputs have been transformed by the hidden layer, these are passed on to the 'output layer'. In this layer, the function f() is applied to the outputs of the hidden layer and produces a final output of this layer. Whatever this function is depends on the problem your solving. It could be sigmoid, softmax, linear activation function etc.

(c) Is this a deep learning model? Why or why not?

I would say this model, with only 1 layer, is not a deel LEARNING model. Generally, a deep learning model involves neural networks with multiple layers. Each layer in this network 'learns' to understand different aspects of the data. Considere the following example; you're trying to recognize certain objects(busses, cats, cars, whatever) in images, the first layer might learn to recognize basic shapes, like lines or edges. The next hidden layer will try to build on this by combining these shapes to recognize more complex patterns, like curves and corners. The deeper you go in to the network, the more complex features a layer learns to recognize. Finally, the last layer (output layer) can use all this information to make predictions about the objects in an image. By using multiple layers, the neural network can learn to understand the data at different levels of complexity.

(d) How is this network trained? Describe the evolution of the accelerated stochastic gradient descent optimization algorithms. Use the following order: Momentum⇒ Adagrad ⇒ Adadelta ⇒ RMSprop ⇒ Adam.

I did a quick google search of the methods commonly used in the training of neural networks, besides the accelerated SGD, and I got the following. Conjugate Gradient, Limited-memory BFGS, Levenberg-Marquardt, Hessian-Free Optimization, Evolutionary Algorithms. I will not go into detail considering these and will focus on the accelerated SGD algorithms in order.

Momentum: This is an optimization algorithm that accelerates the learning process in order to overcome a local minimum. It builds up velocity and deliberately 'overshoots' in order to overcome 'flat' areas if the loss and converge faster toward the minimum.

Adagrad: Adagrad builds on momentum by adapting the learning rate for each parameter based on its historical gradients. It gives more importance to infrequent features and less importance to frequently occurring features. By individually adjusting the learning rate for each parameter, Adagrad can effectively handle sparse or rare features. It ensures that updates are significant for less frequently occurring parameters and smaller for frequently occurring parameters.

Adadelta: This algorithm improves on Adagrad by introducing a more sophisticated adaptive learning rate scheme. What does this mean? Instead of considering all of the previous gradients, adadelta only keeps track of a fixed window of past gradients. This allows its to adapt more smoothly and takes away the need to manually tune the learning rate in order to combat a stagnating learning rate.

RMSprop: This algorithm builds on Adadelta by introducing an exponentially weighted moving average of the squared gradients. RMSprop helps stabilize the learning process and balances the updates across different parameters. By doing this, it adresses certain situations where some gradients are large and others are small. This ensures a more controlled and effective learning rate for each parameter.

Adam: Adam combines the momentum and RMSprop. Adam adapts the learning rate for each parameter based on the average gradient and the accumulated momentum. It provides a balanced update strategy that adapts to the needs of different parameters during the optimization process.

To summarize the above: Each algorithm adds improvements and refinements to the previous one. Momentum introduces accumulated velocity, Adagrad adapts the learning rate based on historical gradients, Adadelta introduces a more sophisticated adaptive learning rate scheme, RMSprop adds an exponentially weighted moving average of squared gradients for stability, and Adam combines momentum and RMSprop to provide an adaptive and intelligent update strategy. These advancements in the algorithms enhance the optimization process, leading to faster convergence and improved performance in training neural networks.

(e) You are presented with Figure 2. These are two images of two different data: You are asked to recommend a model to help distinguish between the grey classand black class data points, based on their x, y values. You have decided to use neural network. What would you advice in terms of network architecture? Brieflyand clearly outline your considerations.

Left graph:

Looking at the graph there seems to be a linear relationship within the grey and the black points, so a linear model might be sufficient. We can use a single-layer pereceptron, also known as a linear classifier, to model the relationship between the points. This consists of an input layer with two neurons (x,y) and an output layer with one neuron representing the predicted class. The activation function for this output layer can be linear. This allows the model to perform linear transformations and make predictions bases on the 'learned' linear relationship. This is a binary classification task, so we will use the binary cross entropy loss function. As for the optimizer, we can use Adam or RMSprop.

Right graph:

Since this graph shows significant overlap between grey and black points, we cannot get away with a linear model. We have to consider a non linear classifier to achieve better seperation between the classes. We have to use multiple layers to account for this non-linearity, at least 1 hidden layer. These layers will introduce non-linear activation functions and alklow the network to learn complex representations of the data. The architecture will look as follows: the input layer with two neureons (x,y), the number of neurons in the hidden layer must be determined by trial and error. We can use a non-linear activation function like Relu or Tanh. Output layer will have one neuron, for binary classification we can use the sigmoid activation function that produces a probiblity between 0 and 1. For the optimizer we can use Adam or RMSprop.

(f) In Keras fit function. Explain what is the shuffle argument. What should be the default for this argument in your opinion, why?

In short, the shuffle argument determines whether the training data should be shuffled before each epoch during training. The order of the training data is randomly changed. There is lots of arguments for shuffling (generalization, reducing overfitting, breaking dependence) and lots of arguments against (time-series data, reproducibility or results). The choice of this argument highly dependent on the task and the characteristics of the data set. Therefore, I would say this argument should default to False.

(g) A neural network has 11 explanatory variables and two hidden layers, where the first hidden layer has 10 neurons and the second hidden layer has 4 neurons, how many (trainable) parameters are there in the network? You decide to add one more neuron to the second hidden layer hoping for improved accuracy, how many (trainable) parameters are added to the model?

First model:

Number of (trainable) parameters = (number of weights in first hidden layer) + (number of biases in first hidden layer) + (number of weights in second hidden layer) + (number of biases in second hidden layer) + (number of weights in output layer) + (number of biases in output layer)

Number of parameters = 110 + 10 + 40 + 4 + 4 + 1 = 169 parameters

Second model:

Number of parameters = (number of weights in first hidden layer) + (number of biases in first hidden layer) + (number of weights in second hidden layer) + (number of biases in second hidden layer) + (number of weights in output layer) + (number of biases in output layer)

Number of parameters = 110 + 10 + 50 + 5 + 5 + 1 = 181 parameters

We've added 12 trainable parameters to the model.

(h) In Figure 3 you are presented with two cost curves (both for the same model): Explain why, while the top curve is decreasing only on average (not at every iteration) the bottom curve is invariably decreasing.

The first cost curve, which sometimes goes up and sometimes goes down, indicates that the model's performance is not consistently improving. During training, the model encounters fluctuations in its ability to minimize the cost. Model might be getting stuck in a local minimum or struggling to find optimal parameters.

The second cost curve decreases in a very smooth way, which would indicate that the performance of the model is consistently improving with each iteration.

The difference between these cost curves can most likely be attributed to either the choice of optimization algorithm or the learning rate.

(i) For the simple network presented in Figure 4, assume the $w(1,1)$ and $w(1,2)$ are estimate to be 4 and -0.5 respectively. $w(2,1)$ is estimated as $w(2,1) = 1$. What is the value of the hidden unit $g(1)(x1, x2)$ for the sample point $x1 = 1$ and $x2 = 6$ (assume that $g(1)$ is the sigmoid activation function)? What is the the value of f $(g(x1, x2))$ when $g(1)$ is the sigmoid activation function and f is also a sigmoid activation function?

The value of the hidden unit $g(1)(x1, x2)$ is determined by the weighted sum of the inputs, passed through the sigmoid function:

g(1)(x1, x2) = sigmoid(w(1,1) * x1 + w(1,2) * x2)

If we fill this in we get the following:

g(1)(1, 6) = sigmoid(4 * 1 + (-0.5) * 6) = sigmoid(4 - 3) = sigmoid(1) = 0.731

Value of the hidden unit is 0.731

The value of $f\ (g(x1, x2))$ when both f and $g(1)$ are the sigmoid activation function can be calculated as follows:

f(g(1)(x1, x2)) = sigmoid(g(1)(x1, x2)) = sigmoid(0.731) = 0.675

(j) For the same values, what is the value of f (g(x1, x2)) when g(1) is the hyperbolic tangent activation function and f , as before, is the sigmoid activation function?

We can use the following formulas to calculate this:

g(1)(x1, x2) = tanh(w(1,1) * x1 + w(1,2) * x2) = tanh(4 * 1 + (-0.5) * 6) = tanh(4 - 3) = tanh(1) = 0.7616

We can impute this into the formula for f(g(1)(x1, x2)), remember the activation function for f is stil sigmoid.

f(g(1)(x1, x2)) = sigmoid(g(1)(x1, x2)) = sigmoid(tanh(1)) = 0.6819

(k) Assuming the tangent activation function is used for the hidden unit, what would be the output value of $f\ (g(x1, x2))$ for a regression problem with f as the linear activation function (recall that $w(2,1) = 1$)?

The output value of $f(g(x1,x2))$ for a regression problem with f as a linear activation function would be equal to g(x1, x2). We previously calculated this:

g(1)(x1, x2) = tanh(w(1,1) * x1 + w(1,2) * x2) = tanh(4 * 1 + (-0.5) * 6) = tanh(4 - 3) = tanh(1) = 0.7616

Exercise 3. Representation learning [20 pt]

(a) Provide two other types of Autoencoder (AE) that differ from the plain vanilla version presented in class. Support your answer with an academic reference that discusses the rationale behind each variant and the unique characteristics that distinguish them from the vanilla AE.

Variational Autoencoder (VAE):

The Variational Autoencoder (VAE) is a type of autoencoder that incorporates probabilistic modeling and latent variable learning. It differs from the plain vanilla autoencoder by introducing a probabilistic interpretation of the latent space. VAEs aim to generate new data points by sampling from the learned latent space distribution. Kingma, D.P. and Welling, M. (2013) Auto-Encoding Variational Bayes. https://arxiv.org/abs/1312.6114

Contractive Autoencoder:

Contractive Autoencoders incorporate an additional regularization term in the objective function to enforce the model to learn robust representations that are less sensitive to small perturbations in the input data. By adding a penalty term based on the Frobenius norm of the Jacobian of the hidden layer activations with respect to the input, contractive autoencoders explicitly encourage the model to learn stable and invariant representations.

Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In Proceedings of the 28th International Conference on Machine Learning (ICML) (pp. 833-840). http://machinelearning.org/archive/icml2011/papers/618_icmlpaper.pdf

(b) AE falls under the umbrella of unsupervised learning. There are many ways in which we can auto-encode. Suggest at least one way in which we can compare and evaluate two different ways.

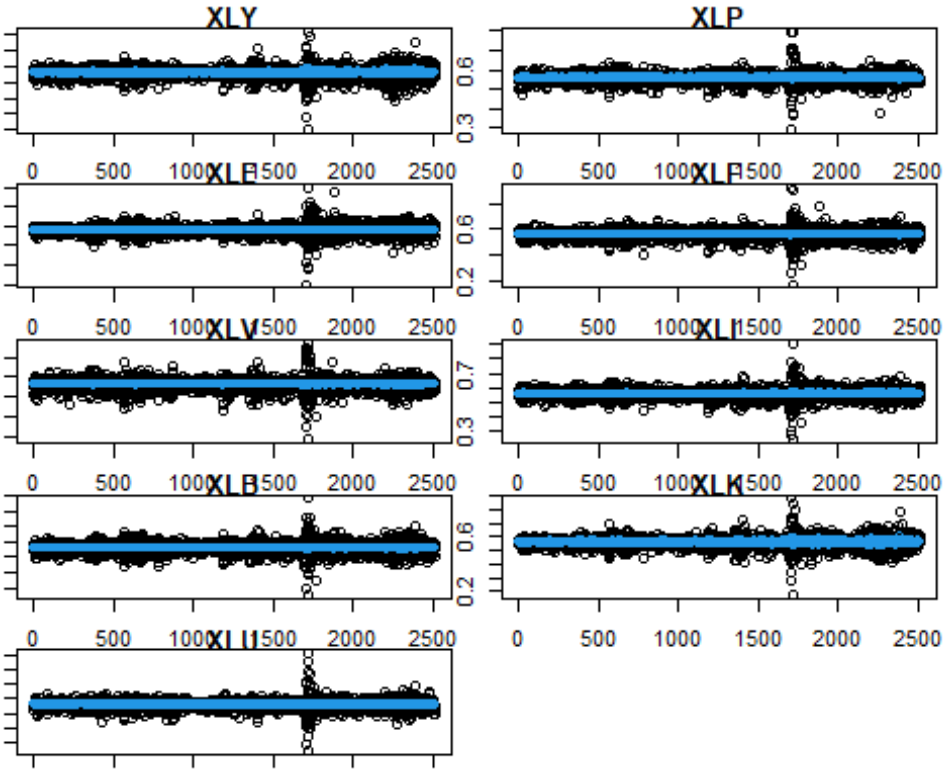
One way to compare different AE methods is by assessing the quality of the reconstructed output. This process is called Data Reconstruction Evaluation. You split the data in a training and test set, you train two AE's using different methods, calculate a reconstruction error metric and finally compare the two. By comparing these you get a sense of how well each method captures and reproduces the input data. There are other ways to do this, one is by letting actual humans evaluate the AE's.

(c) Use the code given in Tutorial 4 to create two different embeddings for the ETF data. Use your embedded spaces to determine which days are outliers (use 20 days). Do the two embeddings agree with each other on the outliers?

Like in the inclass exercise, first we load the data and normalize. We took the in class model and built on this.

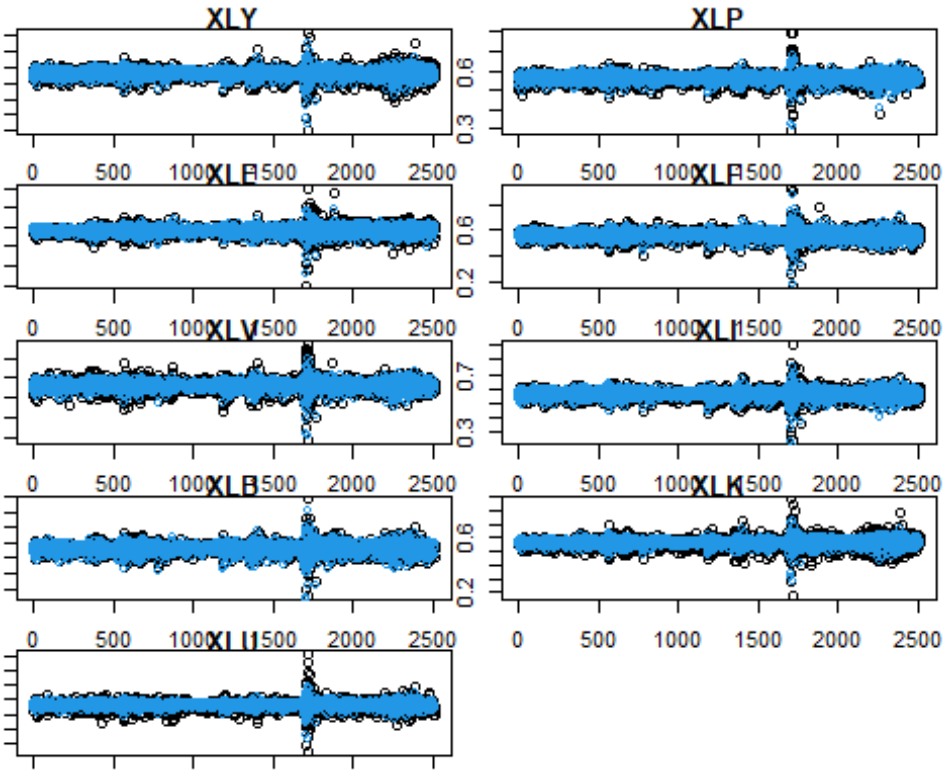
First embedding:

I increased the number of features in the hidden layer of the autoencoder to 5 . This allows the model to learn a more complex representation of the input data. We can adjust this value further to experiment with different dimensions of the latent space. I also increased the number of epochs to 200 and the batch size to 256. These changes can potentially improve the training of the model by allowing it to see the data more times during training (more epochs) and processing more samples in each batch.



Second embedding:

Alternative activation function: Instead of using “relu” in the hidden layer, I used “tanh”. The hyperbolic tangent function can produce different output ranges compared to “relu” and might be suitable for capturing different types of patterns in the data. Alternative optimizer: Instead of using “adam”, I used “RMSprop”. RMSprop is another popular optimizer that adapts the learning rate based on the gradient’s moving average. Different optimizers can affect the convergence speed and performance of the model.



12 out of the 20 dates overlap between the two embeddings.

Exercise 4. Convolution [20 pt]

(a) Load the image to your workspace. You can use (but you don't have to) the magick package.

Image is loaded by using magick package. See markdown file for code.

(b) Pad the image with a 255 value (white color). Use a pad size of 1.

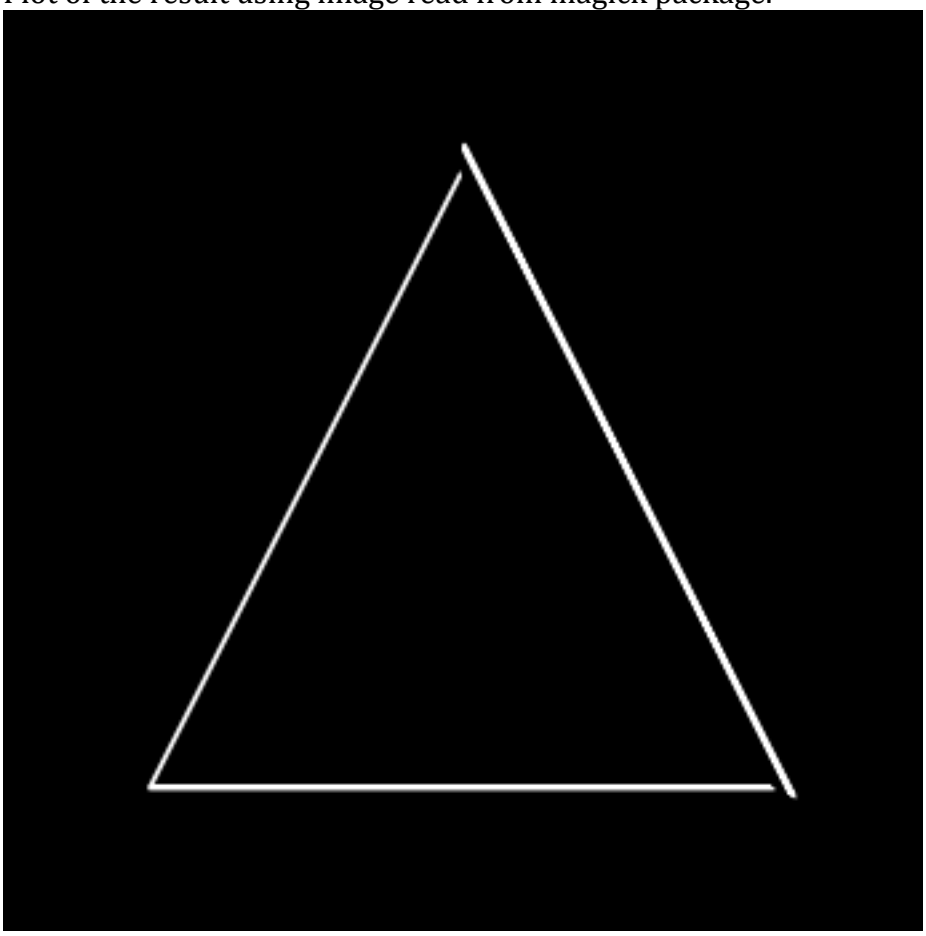
1 layer, white padding added to the image. See markdown file for code

(c) The Prewitt operator is particularly useful for edge detection. Apply that operator to your image. Do not use any functions outside of your base package/module.

Prewitt operator is used and applied to previously loaded image. See markdown for code.

(d) Plot the result. You can use the image_read function from the magick package.

Plot of the result using image read from magick package.



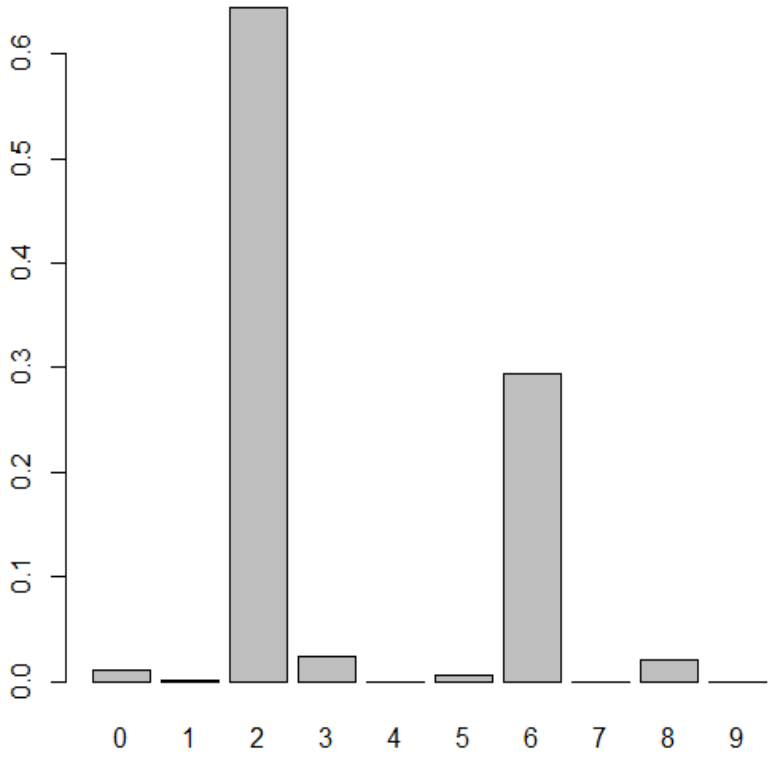
Exercise 5. Model architecture [20 pt]

(a) Categorical cross entropy loss function is often used for deep learning classification models. Describe the difference between the accuracy metric and categorical cross entropy loss metric.

Accuracy is a metric that measures the proportion of correctly classified instances out of total number of instances. This is pretty straightforward. Cross Categorical Entropy Loss, on the other hand, is a loss function that quantifies the dissimilarity between predicted class probabilities and true class probabilities. So, while accuracy focusses on correctness, the CCE loss accounts for the confidence of predictions. Both metrics are important in evaluating and improving classification models.

(b) Use the code provided in the third tutorial to generate predictions for the MNIST dataset using multinomial regression.

Load the data and run the MN model. See bar plot of single prediction below for the number 2:



Below the confusion matrix for this MN model:

```
          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0  678    0   20    6    7   18   24    3   13   16
         1    0 1086   10    0    0    8    3   10   12    6
         2    5    1  843    8    3    4    3   23    7    5
         3   14   29   39  949   15   97    2   39   66   41
         4    0    0   14    0  879   11    8   10    8   40
         5  249    0    6   18    1  612    7    0   24    6
         6   19    4   28    5   14   26  905    0   12    1
         7    3    1   20    8    1    9    0  890   13   14
         8   11   14   41   12   11   91    6    3  800    2
         9    1    0   11    4   51   16    0   50   19  878
```

Accuracy **0.852**

(c) In all of the network architectures we use the sigmoid transfer function for all of the hidden layers. I've selected a common loss function called categorical cross entropy. I've selected one of the simplest optimization algorithms: Stochastic Gradient Descent (SGD). Our output layer also uses a special activation function called softmax. This normalizes the values from the ten output nodes such that: all the values are between 0 and 1, andthe sum of all ten values is 1.

MODEL 1:

The first model has a single hidden layer with 32 nodes using the sigmoid activation function with a batch size of 128 and 5 epochs.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 32)                25120
 dense_4 (Dense)             (None, 10)                330
=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0

      loss   accuracy        mse
0.41175404 0.89639997 0.01753548
```

Accuracy of **0.8957**, this model outperforms the multinomial regression model. Let's see if we can do even better.

MODEL 2:

In this model we keep a couple of hyperparameters fixed, while we play around with the number of nodes. These fixed hyperparameters are: SGD, 10 epochs and the categorical cross entropy loss function. First, we increased the number of epochs to 10, the accuracy increased slightly to 0.91, keeping everything else the same. Then, we increased the number of nodes in the layer (32 (0.9119), 64 (0.9147), 128 (0.9133), 256 (0.9127), 512 (0.9093), 1024 (0.9121)). The sweet spot seems to be with 64 nodes with an accuracy of 0.9147. Lastly, we tried to add some width our neural network. We tried repeated layers of 32 nodes, 64 nodes which seemed to degrade the overall performance. We also tried to increase the depth with each layer (64, 128, 256). These models performed worse than the single layer network.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 64)                50240
 dense_6 (Dense)             (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0

      loss   accuracy        mse
0.31074172 0.91320002 0.01351361
```

Our final second model is a model with 1 layer with 64 neurons, using stochastic gradient descent and 10 epochs. The final accuracy of this model is **0.9147.**

MODEL 3:

In this model we use a different optimization procedure. We try Adam optimization. Adam optimization is well-suited for cases with large amounts of data and parameters. We try this for our simple single layer network with 64 neurons The accuracy increases significantly from 0.9147 to 0.9720. We try to add layers(128, 256), using the Adam optimization procedure, but the accuracy does not increase (0.9673). We try a regularization technique by adding a dropout layer after each layer, which also yield worse results than our single layer (64) model (0.9712).

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 64)                50240
 dense_8 (Dense)             (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0

      loss   accuracy        mse
0.09565865 0.97200000 0.00438644
```

Our final model is a single layer model, with 64 neurons, using the Adam optimization procedure. Using this model we achieve an accuracy of **0.9720**. We are content with these results.

(d) How many hyperparameter are in your model? Describe them briefly.

Our final model uses the following hyperparameters:

Number of epochs: 10
Batch size: 32(defualt)
Optimization algorithm: Adam
Loss function: Categorical Cross Entropy
Activation function: Sigmoid and Softmax
Number of layers: 1 (64 neurons)

(e) Did you manage to beat the multinomial model in terms of accuracy? by how much?

We did manage to beat the multinomials model. The accuracy of this model was **0.852**, whereas our final model boasts an accuracy of **0.9720**.

(f) Create a confusion table. How many images were correctly classified by the DL model which were incorrectly classified by the multinomial model. Argue for the reason (in your opinion).

First the confusion matrix for our final **NN model**:

	true_labels									
predicted_labels	0	1	2	3	4	5	6	7	8	9
0	956	0	15	3	1	15	20	4	10	16
1	0	1105	13	1	3	9	3	25	14	7
2	2	2	889	19	6	5	8	25	10	6
3	2	4	18	905	0	59	2	3	37	12
4	0	1	18	1	905	13	15	11	12	48
5	5	2	2	30	2	719	20	1	30	15
6	8	4	18	4	13	20	884	0	20	1
7	1	1	22	25	1	9	1	919	12	27
8	6	16	32	16	4	34	5	3	811	6
9	0	0	5	6	47	9	0	37	18	871

The final NN model performs quite well overall. Observe that most of the values on the diagonal are high. The number 0 has the highest count of correct predictions, followed by 1. There is some misclassification here and there, the model seems to struggle the most with the numbers 3 and 5.

Confusion matrix of the MN model:

	Reference									
Prediction	0	1	2	3	4	5	6	7	8	9
0	678	0	20	6	7	18	24	3	13	16
1	0	1086	10	0	0	8	3	10	12	6
2	5	1	843	8	3	4	3	23	7	5
3	14	29	39	949	15	97	2	39	66	41
4	0	0	14	0	879	11	8	10	8	40
5	249	0	6	18	1	612	7	0	24	6
6	19	4	28	5	14	26	905	0	12	1
7	3	1	20	8	1	9	0	890	13	14
8	11	14	41	12	11	91	6	3	800	2
9	1	0	11	4	51	16	0	50	19	878

Let's compare the results for our final model and the multinomial model by using a confusion matrix. Each cell in the matrix represents the difference between the predicted labels of the models and the true labels.

	true_labels									
predicted_labels	0	1	2	3	4	5	6	7	8	9
0	271	0	-6	-1	-6	-4	-6	1	-8	-3
1	0	12	6	1	4	-3	0	13	0	1
2	-3	-1	44	16	3	3	9	8	5	4
3	-12	-23	-19	-56	-15	-33	-1	-35	-36	-33
4	0	1	6	2	16	3	10	-3	4	9
5	-236	2	-5	16	1	104	12	1	10	12
6	-10	0	-10	-3	0	-8	-22	0	2	-1
7	-2	0	-3	12	1	-1	1	23	-4	18
8	-7	9	-5	11	-5	-54	-3	0	29	5
9	-1	0	-8	2	1	-7	0	-8	-2	-12

The positive values indicate that the NN model predicted those numbers more than the MN model, while negative values indicate that the MN model made more predictions for those numbers. The values in the diagonal (top-left to bottom-right) represent the correct predictions for each class.

The NN model generally had higher correct prediction counts across most numbers, with relatively few misclassifications. Numbers 0 and 1 had the highest correct prediction counts, while numbers 3 and 5 had relatively higher misclassifications with other numbers

The MN model had lower correct prediction counts for several numbers compared to the first model. It also had a higher number of misclassifications, particularly in numbers 3, 4, and 8. However, it performed well in numbers 0, 1, and 6, with higher correct prediction counts.

Overall, the NN model appeared to have better performance in terms of overall accuracy and lower misclassification rates, while the MN model had some specific classes where it performed better. Further analysis and evaluation metrics would be required to make a more comprehensive assessment of the models' performances.