Introduction to Programming with Scientific Applications (Spring 2022)

Final project

Study ID	Name	% contributed
201907685	**Søren Orm Hansen**	**100/3**
201906235	**Pernille Højlund Brams**	**100/3**
201908918	**Anna Hedvig Møller Daugaard**	**100/3**

max 3 students

Briefly state the contributions of each of the group members to the project
This project was written as a group project with all members contributing an equal amount in the production of the project. Thus, it is impossible to state which parts each member contributed with.

Note on plagiarism

Since the evaluation of the project report and code will be part of the final grade in the course, **plagiarism in your project handin will be considered cheating at the exam**. Whenever adopting code or text from elsewhere you must state this and give a reference/link to your source. It is perfectly fine to search information and adopt partial solutions from the internet – actually, this is encouraged – but always state your source in your handin. Also discussing your problems with your project with other students is perfectly fine, but remember each group should handin their own solution. If you are in doubt if you solution will be very similar to another group because you discussed the details, please put a remark that you have discussed your solution with other groups.

For more Aarhus University information on plagiarism, please visit http://library.au.dk/en/students/plagiarism/

Introduction

In this report we aim to implement a neural network to classify the hand-written digits from the MNIST database. We develop the neural network, test its ability to recognize digits, and examine its performance. Furthermore, we discuss our implementation- and design choices as well as possible improvements.

All code and visualizations are available on GitHub: MNIST_IPSA or in this drive folder.

Methods and Implementation

It was essential to the project that packages like Numpy, Keras, Pytorch, Tensorflow etc. were not used, since they provide ready-to-use machine learning frameworks. Instead, the neural network was built from scratch with 'pure' Python, and the modules used the most for this were *matplotlib* (The Matplotlib Development team, 2021) and *random* (Python Software Foundation, 2001). Further, the functions *open* and the built-in package *json* were used to read in labels and images. *Matplotlib* was naturally used for visualizations, but also for debugging along the way, functioning as a health check for the network and its learning.

A central part of the project was permutation (i.e. often also called *shuffling* in the literature (Brandvain, 2021) and will be called as such from here on) and batching, where *random* was used. The task pertaining to this was #16, which dictated shuffling of a list of data before partitioning the data into batches. Shuffling ensures that the train/test sets are representative of the entire dataset, and are not meaningfully organized (e.g. sorted) which would risk the model finding an unwanted meaningful pattern in the order (deepwizAI, 2022). After shuffling, batching was conducted. These together thus serve to improve model fit by mitigating risk of overfitting, ensuring that the model remains general and is not 'stuck' on a suboptimal local minimum (McElreath, 2016), and is exposed to the dataset in iterations. An important design choice made in terms of the model architecture was to shuffle our data after each epoch, which is an established practice in the ML community (*Performance Tips | TensorFlow Datasets*, n.d.). Further, random weights were initialized to help mitigate the model getting stuck during the training.

Normalization of the pixel values in the pictures was conducted, constraining values to be between 0 and 1 (in function #7). Further, the problem statement suggested running the model with 5 *epochs* and a *batch size* of 100. Mean Squared Error (MSE) was applied as a cost measure to evaluate how far off a network's prediction of a label was from the actual label per input, and in the function *update()* the derivative of MSE (formula for MSE below, formula for derivatives appear in the problem statement and in the code) is used to update the weights and biases.

Mean Squared Error formula:

$$\text{cost}(a, y) = \frac{sum_i((a_i - y_i)^2)}{10}$$

When investigating the best network, the code is configured to save a network if it performs with a higher accuracy than in the previous iterations.

In terms of design choices, readability and generalizability of the code, an aim was to be compliant in accordance with PEP-8 as per the guidelines supplied on sites like https://realpython.com/ especially pertaining to guides in block-design, docstrings, commenting and spacing. The questions in the project were given in steps 1.-18. and this structure was kept in our submitted code, resulting in a block-by-block design of the code and in the more extensive functions like *update()* and *learn()*, which depends on previously made functions in preceding blocks.

Note that some questions (e.g. 4 and 14) asked for returning a few images. These can be found in App. B.

Visualizations and Outputs

GIFs for visualizing learning

To get a feeling for how our network learns, we have shown 99 images to our network, let the network learn from the images, and saved the weights of the network after each image. We then collected the images of the weights as 10 separate gifs. We have chosen all 0's for the initial state of the network in order to get clearer images of how the network learns. The gifs can be found here: drive or on github.

Visualization of accuracy and cost

We have plotted the out of sample accuracy and cost of the network, as it evolves over time (see Figure 1 and 2 in App. A). This gives us an insight into how the network learns over time and works to minimize cost.

Output (best network)

Finally, we have found the most accurate version of the network throughout the 5 epochs and 600 batches (with batch size 100), and plotted the weights of each digit of the accurate version of the network (see Figure 3 in App. A). We gained an accuracy of 86.93% in our network and a cost of 0.044.

Discussion and further improvements

The modular structure of the code makes for a flexible design and is a manageable breakdown of the project. The harder parts of the project were the functions *learn()* and *update()*, where we use the previously defined modules to build the network. These functions tested our conceptual understanding of the purpose of the network's workings. It was interesting to see that we could build a neural network using only basic matrix algebra and derivatives.

An improvement of this project would be including tuning of hyperparameters. This would essentially result in finding a more optimal architecture for the model (assuming that one exists) which optimizes performance (Warnes, 2021). Hyperparameters are e.g. no. of epochs (i.e. no. of times model is exposed to the entire dataset), no. of batches (i.e. no. of iterations required to complete an epoch), step size/learning rate, no. of layers in the network, cost function, etc. For example, using different cost functions (e.g. the softmax function could have been applied here instead of MSE) will result in different performances both in terms of accuracy and training time. We remain agnostic as to whether the suggested hyperparameters of 5 epochs and batch size of 100 provided in the problem statement are the result of an optimal-hyperparameter-search, but for a future improvement, automatic hyperparameter tuning could be conducted e.g. through the service Weights & Biases. This would also include having a validation set allowing for early-stopping (Brownlee, 2018), halting the network when improvement has not occured in the last *x* amount of epochs or iterations. This saves unnecessary waiting time, which could be useful in our case, since our network started stagnating (see Figure 1, App. A) in terms of increasing accuracy already around iteration 300-500 in the very 1st epoch of 5. As this is a simple network, we expect that a change in e.g. cost function or step size would have a direct impact on the network's learning.

More complex network structures, e.g. convolutional neural networks (which are extremely proficient in image classification tasks) could result in a better performance. However, the simple network structure allows for a clear interpretation of the result and how the results are derived and thus, though simple, this network serves to build a good intuition of the effect of a network's hyperparameters. Furthermore, complex neural networks are often criticized for compromising interpretability for better accuracy, making it nearly impossible to explain *why* a network made a certain prediction. This can quickly become vital, e.g. if the model is used to diagnose people or decide if prisoners get parole (O'Neil, 2016).

Three different visualizations of the network thought to capture key aspects were chosen. For further improvements, additional visualizations could be heatmaps with predictions and accurate number values from 0-9. This would visualize the range of performance across digits, and highlight problematic areas. It would be interesting to see whether the network has trouble detecting numbers visually close together e.g. 5 and 6 (as figure 5 in App. B hints to) as well as 4 and 9. Another improvement could also be to visualize bias in the network.

A downside to the training set is that it only contains images of handwriting, i.e. only targets. An improvement to the model would be to add images of e.g. animals, random noise or augmentations of the existing dataset, for instance by blurring or stretching the digit-pictures, or including pictures of digits written with a different pen than the one in the MNIST-dataset. This would challenge the network more, likely increasing its ability to classify.

Notably, we have in this project chosen to interpret “best network” as “most accurate”. We could also have chosen to gauge ‘best network’ from the metric of cost, as having a low cost does not necessarily correspond with the highest accuracy. The cost of a network is however a product of the cost function, so comparing networks with different cost functions is to be done with precaution.

When evaluating the best network in terms of accuracy, an improvement for saving computational power could be to have a validation set, keeping track of the best performing network as mentioned above.

Conclusion

In this project we have succeeded in building a neural network for written digit detection. We achieved an accuracy of 86.93% and cost of 0.044 in the best network. We have produced plots visualizing accuracy and cost over time, along with GIFs. We have discussed improvements to both our network and the means for visualizing them, after outlining implementation and design choices. Further, submitted code was written in accordance with PEP-8 guidelines. We conclude - given the simplicity of the network - that we achieve good accuracy for predictions.

Length of report: 9563 characters incl. spaces = 3.98 standard pages.

References

https://matplotlib.org/

Brandvain, Y. (2021). *Chapter 17 Shuffling labels to generate a null | Applied Biostats*. https://bookdown.org/ybrandvain/Applied-Biostats/perm1.html

Brownlee, J. (2018, December 9). Use Early Stopping to Halt the Training of Neural Networks At the Right Time. *Machine Learning Mastery*. https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

deepwizAI. (2022). *Why randomly shuffling data improves generalizability in neural networks*. DeepwizAI. https://www.deepwizai.com/simply-deep/why-random-shuffling-improves-generalizability-of-neural-nets

McElreath, R. (2016). *Statistical Rethinking: A Bayesian Course with Examples in R and Stan (1st ed.). Chapman and Hall/CRC.* https://doi.org/10.1201/9781315372495

Myrianthous, G. (2021). Shuffling Rows in Pandas DataFrames. *Towards Data Science*. https://towardsdatascience.com/shuffling-rows-in-pandas-dataframes-eda052275635

O'Neil, C. (2016). *Weapons of math destruction: How big data increases inequality and threatens democracy* (First edition). Crown.

Performance tips | TensorFlow Datasets. (n.d.). Retrieved 13 May 2022, from https://www.tensorflow.org/datasets/performances

Python Software Foundation. (2001, 2022). *Random—Generate pseudo-random numbers—Python 3.10.4 documentation*. https://docs.python.org/3/library/random.html

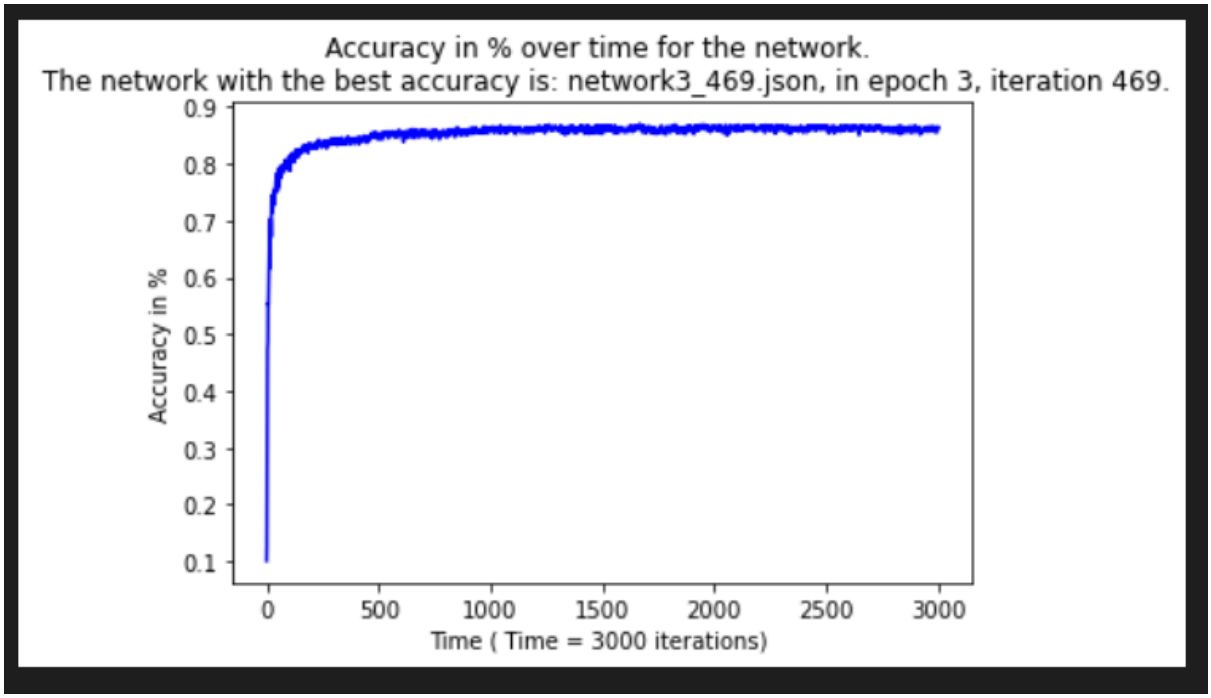
The Matplotlib Development team. (2021). *Matplotlib—Visualization with Python*. https://matplotlib.org/

Warnes, Z. (2021, December 14). *Hyperparameter Tuning—Always Tune your Models*. Medium. https://towardsdatascience.com/hyperparameter-tuning-always-tune-your-models-7db7aeaf47e9

Appendix A

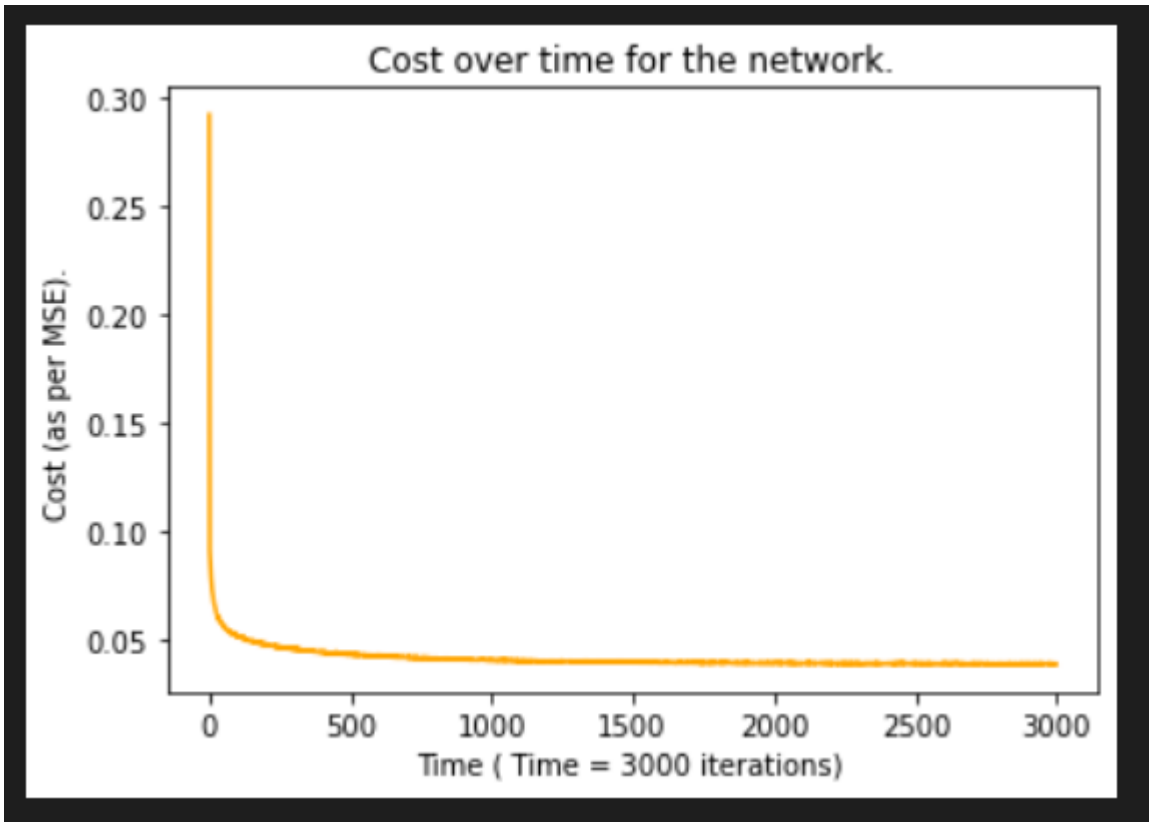
Plots pertaining to the optional 'Question 20.: *Optional. Visualize the changing weights, cost, and accuracy during the learning.'*

Figure 1: Accuracy in % during learning



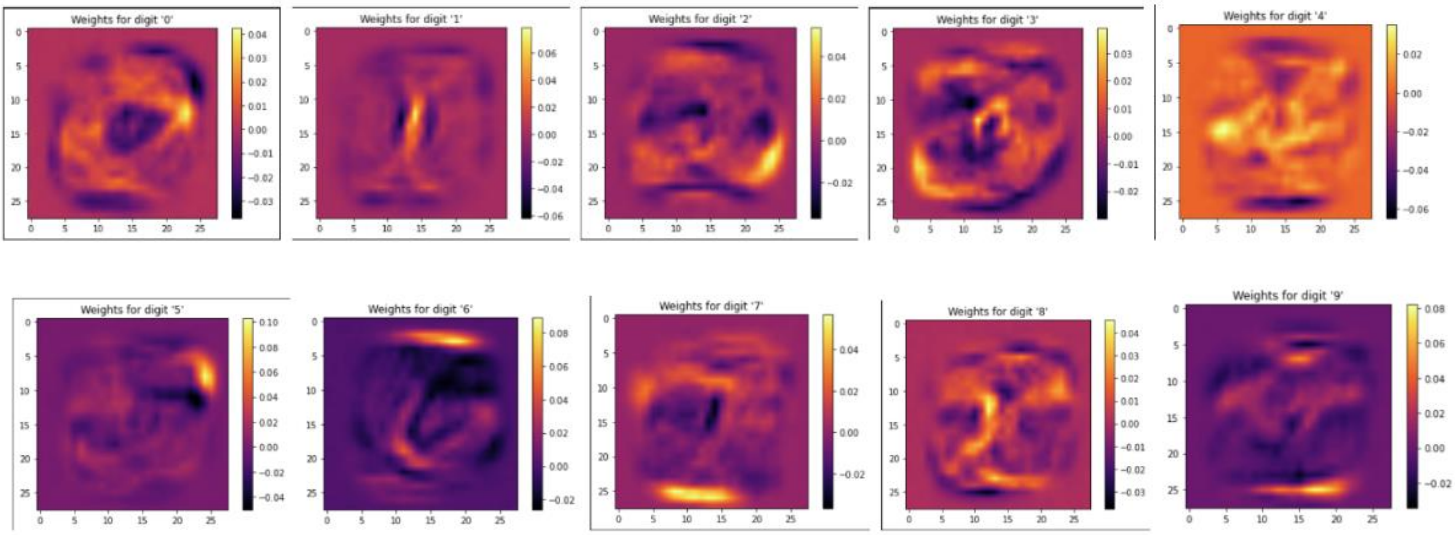
This figure shows accuracy in % over time, i.e. during the iterations over which the network is learning. A visible stagnation starts around the 300-600th iteration, meaning in the very 1st epoch, from where no more substantial increments occur in terms of accuracy.

Figure 2: Cost over time during learning



This figure shows the cost over time for the network during learning. We see an early drop in cost which after the 300-500th iteration (meaning in the 1st epoch) is already below 0.05.

Figure 3: Final weights (for visualization of weights during learning, see GIFs)

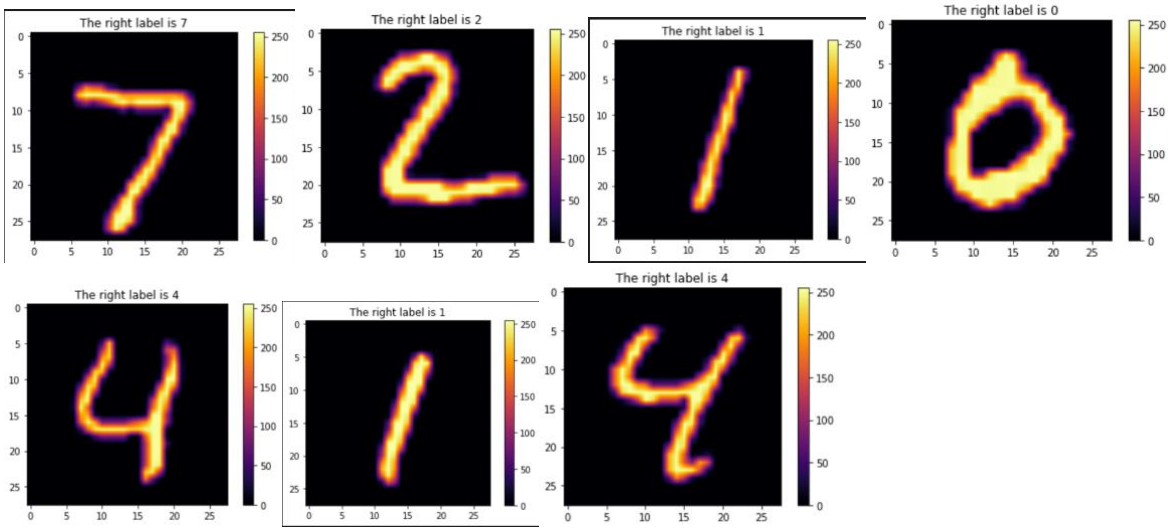


Appendix B

Pictures from question 4: 'Plot images: Make a function plot_images(images, labels) to show a set of images and their corresponding labels as titles using imshow from matplotlib.pyplot. Show the first few images from t10k-images.idx3-ubyte.gz with their labels from t10k-labels.idx1-ubyte.gz as titles. Remember to select an appropriate colormap for imshow.**'**

Here, we visualize the first 7 images. As a colormap for imshow, we chose **inferno**:

Figure 4: Plot_images output a)



Pictures from question 14: 'Extend plot images: Extend plot_images to take an optional argument prediction that is a list of predicted labels for the images, and visualizes if the prediction is correct or wrong. Test it on a set of images from t10k-images and their correct labels from t10k-labels.**'**

Here, the function version configuring a grid (plot_images_grid()) was used for a nicer visual, but functionally it returns the same as plot_images() as configured in 14. We inserted in the title of every image whether the digit was predicted or not. Nr 9 was predicted faulty as can be seen below:

Figure 5: Plot_images output b)