Abstract:

In this assignment, we compared the accuracy of the multilayer perceptron (MLP) algorithm with that of the ConvNet algorithm. We used the Fashion-MNIST dataset and trained both of our models on 60 000 images and tested them on 10 000 images. We found that using L2 regression as a weight penalty gave us better results, and that Leaky-ReLU was better than the other activation functions for our dataset. We also used 128 units and 2 hidden layers, which gave us a peak accuracy of 79%. In comparison, we trained a ConvNet model with 2 convolutional layers and 2 fully connected layers, and we found an accuracy of 92.2%. We found that ReLU and LeakyReLU work better with lower learning rates as we have more layers, and that Tanh can support much higher learning rates. We started observing overfitting at 2 layers when using larger learning rates. When testing different parameters for the ConvNet model, we did not observe any noticeable differences in accuracy or training time.

Introduction:

We were tasked in this project to classify image data using a multilayer perceptron that we implemented from scratch, and to compare its performance with that of a convolutional neural network. The dataset used for both part of the experiment was the same and consisted of 70 000 28x28 images of Zalando's clothing articles.

A deep learning algorithm learns the pattern of the data given as input and learn how to represent it with features they extract. They then combine everything they have extrapolated form the data and combine it to make a more abstract and high-level representation of the data. This method is very hands-off and allows for a fast interpretation of the data without much human intervention (Bento, 2021). Neural networks are inspired by the structure of the brain and are used in perceptrons and multilayer perceptrons to discover patterns in the data (Bento, 2021). The algorithm can learn the weights progressively to generate an output that optimizes the training accuracy. A multilayer perceptron is a neural network that has an input and an output layer with as many hidden layers as we want in between, and can use any activation function, which is important in determining whether the "neuron" (or the node), will fire and send its information to other layers or not. In this project, we experimented with three activation functions: ReLU, tanh and Leaky-ReLU, and we have found that Leaky ReLU gives us better results compared to the others. The algorithm learns through backpropagation and allows the algorithm to adjust the weights iteratively to minimize the cost function. One important requirement for backpropagation is that the function used must have a bounded derivative as the derivative of the activation function for example is used to calculate the gradient (Bento, 2021). One more way to optimize the accuracy is to use a regularization term which acts as a penalty and helps avoid overfitting the model by favorizing smaller weights (Pedregosa et al., 2011). In our experiments, we have observed that using L2 regularization increases the accuracy on the testing set. We have thus combined L2 regularization in our model with the use of Leaky-ReLU as the activation function, 128 units, 2 hidden and a learning rate of 0.07 layers as it gave us an accuracy of 79 %.

We implemented the convolutional neural network using Keras and TensorFlow libraries. A Convolutional Neural Network (ConvNet) is a deep learning algorithm that takes images as input and can classify them based on weights and biases that it assigns to certain features. While a multilayer perceptron is fed a "flattened" image (1x784 instead of 1X28x28 for a single image), ConvNet keeps the image as a 3D object and is able to extrapolate spatial dependencies that are extremely important in determining the classification of the data. This is why we see consistently in our experiments that the accuracy hovers around 70% in MLP, while it goes up to 92% in ConvNet because the algorithm takes into account where the pixels are placed and their relationship with one another (Saha, 2018).

Datasets

The Fashion-MNIST dataset consists of 70 000 images of Zalando's clothing articles separated into 60 000 images for the training set and 10 000 images for the testing set. Each one is a 28x28 grayscale image that can be part of one of 10 classes, which are: t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. When working with the MLP, the images were kept with their shape of (1, 784) for each image (so the training dataset had a shape of (60 000, 784)) since that is how the model can make operations on the data. In the ConvNet algorithm, the images were resized to (60 000, 28, 28) since ConvNet is able to take into account the spatial relationships between the pixels, which ultimately results in better accuracy. We also divided the data into mini batches for both algorithms due to memory limitations, which could have resulted in some loss in information, particularly for the MLP model.

Results

MLP:

For all experiments, when initializing the weights for the layers, we multiplied a random normal distribution by 0.5. Firstly, we tested the base implementation of the model by running it with no hidden layers. Doing so allowed us to have a baseline for all further tests we were going to do. We ran this test with a learning rate of 0.3, 10 maximum iterations, and 5 minibatches. *Figure 1* shows that cross-entropy loss lowered to around 2 by the end of the fitting, and we observed 71.1% accuracy on the test set.

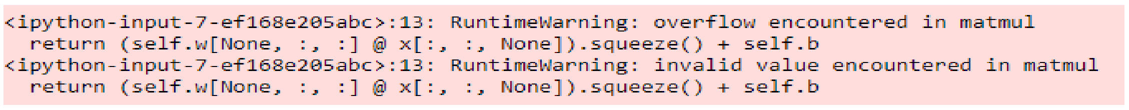
As we can see in figure 2, we then tested another model, this time with a hidden layer consisting of 128 hidden units using ReLU activation. This model took considerably more time than the previous model to finish fitting. Furthermore, the cross-entropy graph doesn't look as smooth as with no hidden layer. Both phenomena are to be expected as this model has many more parameters to train and many more gradients to account for. This also means there are many local minima on the landscape of cross-entropy losses it can attempt to optimize for in any given iteration. We decided to run this test with a 0.1 learning rate and observed better results. The cross-entropy is higher, which makes sense as there are more units in the network, but it still decreases over time as shown in figure 2. Since we have more minibatches this time, we also have more epochs. In this test, we observed 75.4% accuracy on the test set.

We then increased the number of hidden layers to 2, each with 128 hidden units, still using ReLU as the activation function. Once again, this test was much slower than the test with 1 hidden layer to fit to the training data, which is to be expected. In this graph, however, we clearly see the moment at which the code changes minibatches, which is a jagged edge in the cross-entropy graph. We reach the same cross-entropy as the previous model (around 2.5 as shown in *figure 3*), but this time we only observe an accuracy of 66.8% on the test data when using a learning rate of 0.05, indicating that we are observing overfitting. With a learning rate of 0.1, the accuracy is considerably worst, at around 34%. We also attempted to train this model on the unnormalized image dataset and observed a peculiar cross entropy graph as shown below. We see it drop very rapidly at around epoch 35 and stay at around 2.5, except when it spikes twice (*figure 4*). This model has an accuracy of only 10% on the test set despite its relatively low cross entropy most of the time.

Next, we tried a 2 hidden layer with 128 hidden units each model using Tanh activation function. In this case, we observed that we can reach a very high learning rate, up to 0.4, and still maintain decent accuracy on the test set, namely 71.4% .This can be explained by looking at the Tanh function and noticing that it's gradient will not be arbitrarily big, instead we see small gradients when the function saturates, which could explain why a large learning rate is not causing problems (figure 5). Interestingly, this model has far less overfitting when compared with the model containing the same number of layers and hidden units but using ReLU activation instead. Although the learning rate is high, the cross entropy stayed at around 1 for most of the fitting. The same model with a learning rate of 0.2, 0.35, and 0.5 showed a similar trend, with cross entropy stabilizing rapidly at some value for each minibatch, and increasing accuracies with higher learning rates (67.7%, 71.0%, and 71.8% respectively, and figures 6 to 8). However, none of these models outperformed the ReLU model with a single hidden layer.

We then moved on to a 2 layer, 128 hidden unit model with LeakyReLU activation. Generally, we observed similar trends as with ReLU but with slightly increased accuracies. We tested the model with a learning rate of 0.05 and 0.1, obtaining 73.4% and 75.5% accuracy on the test sets respectively (figure 9 and 10).

However, when we then attempted to train the model with a learning rate of 0.2, we ran into numerical instability issues, namely overflow in the matrix multiplication of some hidden layer unit values and the weights. This same problem occurred for a learning rate as small as 0.13 as shown below.



```
<ipython-input-7-ef168e205abc>:13: RuntimeWarning: overflow encountered in matmul
  return (self.w[None, :, :] @ x[:, :, None]).squeeze() + self.b
<ipython-input-7-ef168e205abc>:13: RuntimeWarning: invalid value encountered in matmul
  return (self.w[None, :, :] @ x[:, :, None]).squeeze() + self.b
```

To remedy this problem, we then started experimenting with L2 regularization to avoid this problem as much as possible. We first ran a 2 layer with 128 hidden units using ReLU activation. We started with a learning rate of 0.05 and observed 72.8% accuracy (*figure 11*). We also decided to experiment with a network consisting of 3 hidden layers, but this also encountered the overflow problem as earlier. In an attempt to generate the best model we could, we trained a 2 layer, 128 hidden unit per layer model using LeakyReLU as the activation function, but using L2 regularization as well. With a learning rate of 0.04, we achieved 75.3% accuracy (*figure 12*).

We then trained this same model using a learning rate of 0.07 and double the number of training iterations, and the resulting model was able to achieve 79.0% accuracy on the testing set (*figure 13*). We then wanted to see if this performance could be achieved using 64 hidden units instead of 128, so we trained the same model and observed 77.6% accuracy, which is still better than all of our other models without L2 regularization.

ConvNet:

Finally, we decided to test a convolutional neural network (CNN) to compare to our current MLP networks. To do so, we generated a network consisting of 2 convolutional layers using 8 filters, each followed by max pooling with a pooling size of 2, 25% dropout rate, followed by 2 layers of fully connected layers and finally the output layer. The training of this model over 30 epochs resulted in overfitting, but test accuracy reached 92.3% and stayed there without increasing or decreasing drastically (figure 14), although the model's cross entropy loss started increasing. This generated the confusion matrix in figure 15.

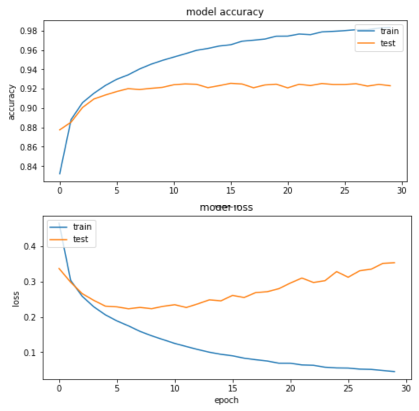


Figure 14

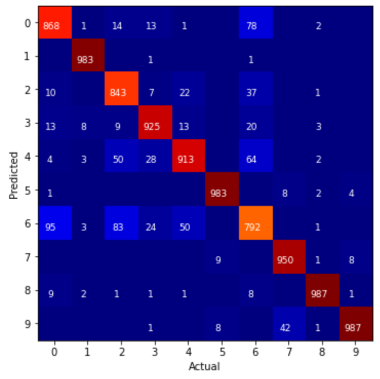


Figure 15

Discussion:

When looking at all the models we trained, the model with no hidden layer performed very well relative to other models when considering the very rapid training time it required. One hidden layer appeared to improve results slightly, but we started seeing overfitting once we reached 2 layers. Overall, we observed that as we add more layers and parameters to the network, we have to use smaller and smaller learning rates to avoid encountering errors such as drastic overfitting or overflow errors. We also observed that normalization is absolutely required for any work on this dataset, as the models were unable to perform adequately on unnormalized images. We also applied L2 regularization to our models and observed some accuracy improvements in the final trained models. Notably, we were less likely to run into overflow errors when using L2 regularization.
We also tested the Tanh activation function which behaved differently than ReLU. We observed much higher learning rates (up to 0.5) without any overfitting or overflow problems. However, we did not observe accuracy or run-time benefits from using Tanh.
LeakyReLU provided slightly better model accuracies than ReLU but suffered from the same overflow problems with

improperly adjusted learning rates. The best fully connected MLP model we were able to obtain was a 2-layer, 128 hidden unit per layer with LeakyReLU as the activation function. Using this model, we were able to reach 79% accuracy. Interestingly, this same model with 64 hidden units per layer was able to reach 77.6% accuracy, making it our second most accurate model.
However, none of these models were able to reach the accuracy of our CNN model with 2 convolutional layers and 2 fully connected layers, at around 92.2% accuracy. Even by tweaking the parameters of this neural network, the accuracy did not change by much. Furthermore, this model was significantly faster to train than our fully connected models.
Although we believe that our fully connected models could potentially be slightly improved by testing more parameters, such as testing 1 layer MLPs, or altering the random weights which are randomly generated at first, we don't believe that a simple fully connected MLP will reach both the same accuracy and efficacy of a CNN.

Contributions:

Both Soumaia and Arien worked on the code for MLP and ConvNet; Soumaia wrote the abstract, introduction and dataset sections of the report, and Arien wrote the results and discussion sections.

Citations:

Bento, Carolina. "Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis." *Medium*, Towards Data Science, 30 Sept. 2021, https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141.

Pedregosa et al. "Varying Regularization in Multi-Layer Perceptron." Scikit-Learn, Journal of Machine Learning Research, 2011, https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mlp_alpha.html.

Saha, Sumit. "A Comprehensive Guide to Convolutional Neural Networks-the eli5 Way." *Medium*, Towards Data Science, 15 Dec. 2018, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Appendix

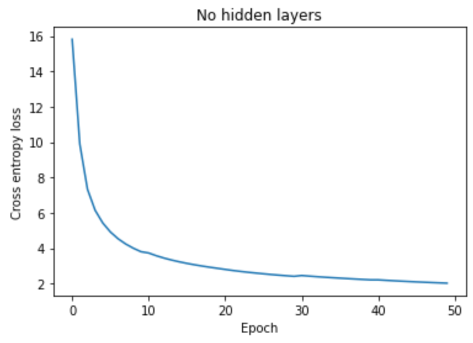


Figure 1. No hidden layers, activation function is ReLU with a learning rate 0.3

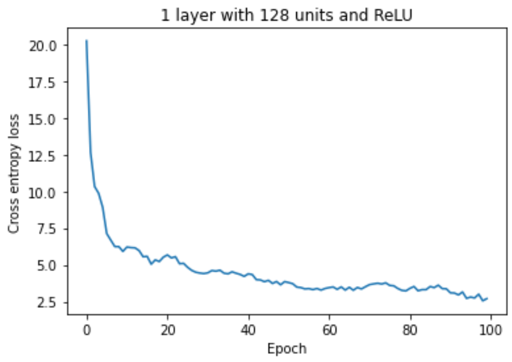


Figure 2. 1 layer with 128 units, a learning rate of 0.05 and ReLU as the activation function.

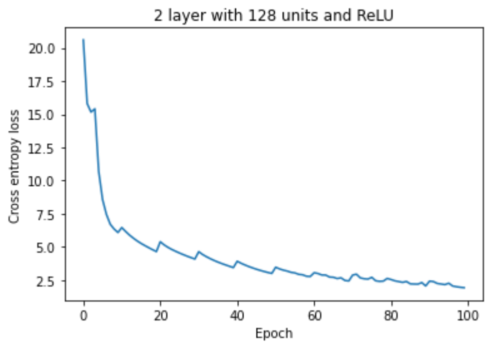


Figure 3.

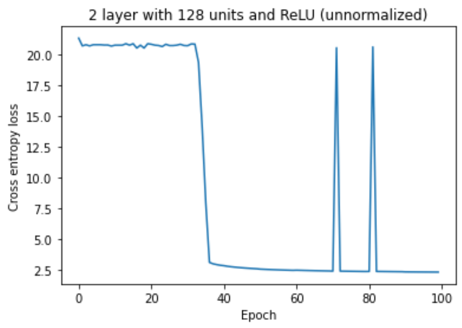


Figure 4.

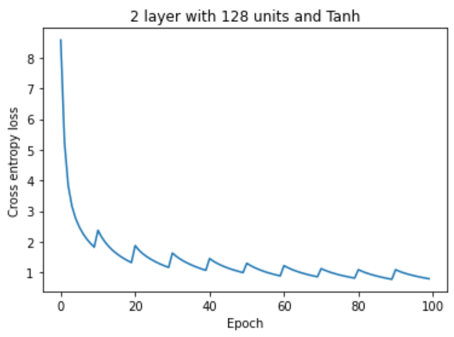


Figure 5.

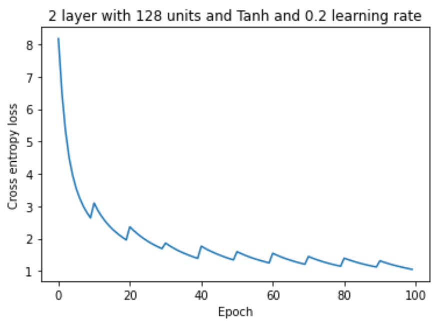
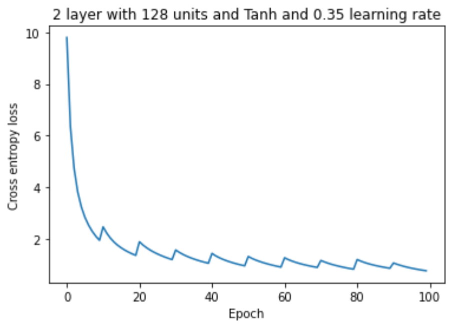


Figure 6.



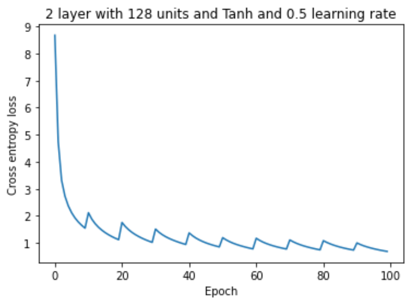
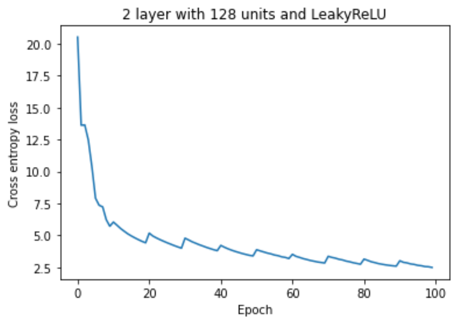


Figure 7.

Figure 8.



..

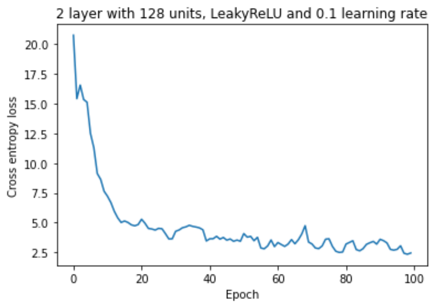
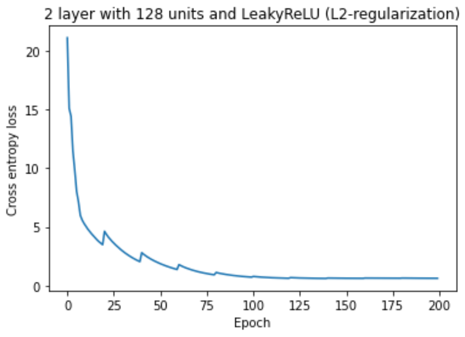


Figure 9.

Figure 10.



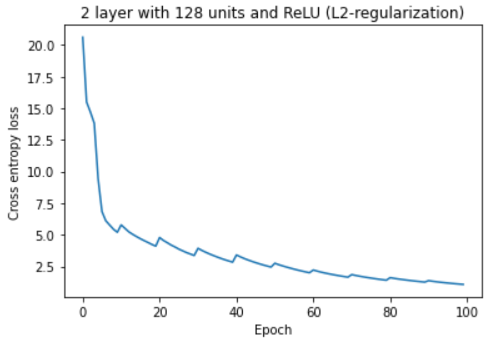


Figure 12.

Figure 11.

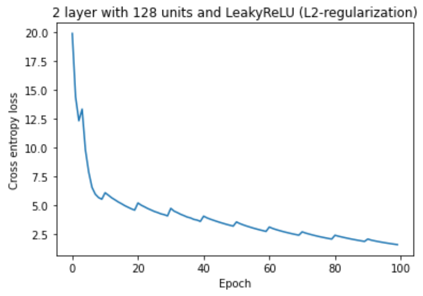


Figure 13.