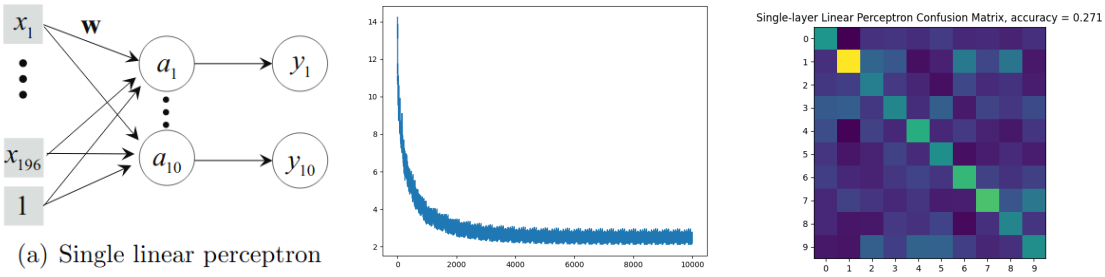
CSci 5561 Assignment 4

Luis Guzman

Friday November 27, 2020

This assignment focused on the task of handwritten digit recognition by implementing four different classification methods: a single layer linear perceptron (SLP), a single-layer perceptron using softmax cross-entropy loss (SLP-CE), a multi-layer perceptron (MLP), and a convolutional neural network (CNN). The SLP model achieved 27% accuracy, the SLP with cross entropy achieved 88% accuracy, the MPL was around 89% accurate, and the CNN was around 87% accurate. I will highlight my methods and elaborate on these results in the following paragraphs.

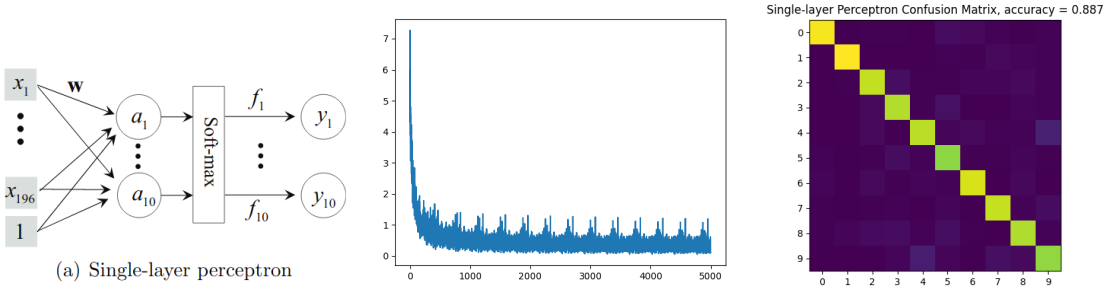
The Single Layer Perceptron consisted of a single fully-connected layer with 10 output nodes (one for each label). The fully connected layer implements the linear function $y = wx + b$, where $w \in \mathbb{R}^{n\times m}, x \in \mathbb{R}^{m\times 1}$, and $b \in \mathbb{R}^{n\times 1}$. This model uses euclidean loss $l = ||\tilde{y} - y||_2$ and the gradient is $\frac{dl}{dy} = 2(\tilde{y} - y)^T$. I implemented back-propagation using the equations $\frac{dl}{dx} = \frac{dy}{dy} \cdot w$, $\frac{dl}{dw} = x \cdot \frac{dy}{dy}$, and $\frac{dl}{db} = \frac{dy}{dy}$. Lastly, I generated mini batches of size 32 from the dataset and performed stochastic gradient descent (SGD) to train the network. Figure 1 shows my results. The low accuracy is to be expected because the dataset is not easily classified by a linear model.



(a) Single linear perceptron

Figure 1: The network architecture, loss per iteration and the resulting confusion matrix for the SLP

The next algorithm uses cross entropy loss to greatly improve the accuracy of the single-layer perceptron. The softmax cross-entropy loss is defined as $l = \sum_i^m y_i \log \tilde{y}_i$ where $\tilde{y}_i = \frac{\exp x_i}{\sum_i \exp x_i}$. I then calculated the gradient to be $\frac{dl}{dy} = (\tilde{y} - y)^T$. This model performed much better, with an accuracy of 88%



(a) Single-layer perceptron

Figure 2: The network architecture, loss per iteration and the resulting confusion matrix for the SLP-CE

Next, I added an additional hidden layer to form a Multi-Layer Peceptron. This hidden layer used the ReLU activation function, which is defined as $y = \max(x, 0)$ This activation function essentially forces all negative values of x to not contribute to the gradient, so the back propagation function must take this into account: $\frac{dl}{dx} = \frac{dl}{dy} \odot \mathbb{1}_{x \geq 0}$ where $\odot$ indicates element-wise matrix multiplication and $\mathbb{1}_{x \geq 0}$ is the matrix where each element i is 1 if $x_i \geq 0$ and 0 otherwise. This method had a marginal improvement over the SLP-CE, with a final accuracy of 89%. I think further hyperparameter tuning could result in the $\geq$90% accuracy mentioned in the problem statement, but I could not break the 90% mark despite trying many different values for the learning and decay rates.

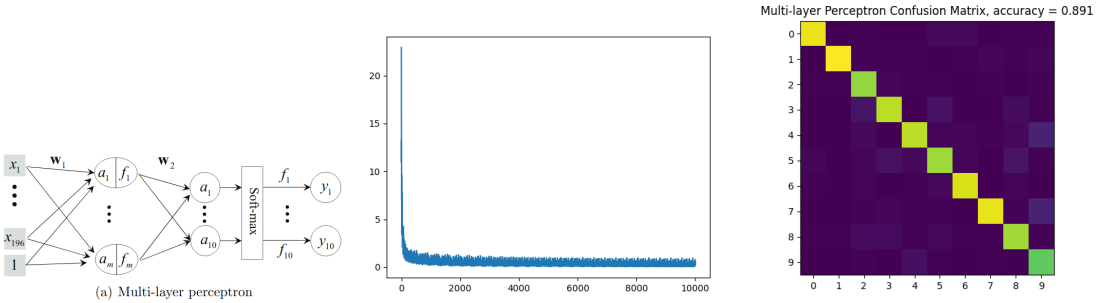


Figure 3: The network architecture, loss per iteration and the resulting confusion matrix for the MLP

Lastly, I implemented a convolutional neural network. I used the im2col function to restate the convolution operation as matrix multiplication. I used 3 filters of size 3×3, for a total w matrix of size $3 \times 3 \times 1 \times 3$. I used same-padding and a stride of 1, so the input of the convolution layer was of size $14 \times 14 \times 1$ and the output was $14 \times 14 \times 3$ (for the three separate filters). Next, I implemented a max-pooling layer. Using a pooling size of 2×2 and a stride of 2, the output of this layer was $7 \times 7 \times 3$. The input was flattened before the fully-connected layer and softmax activation.

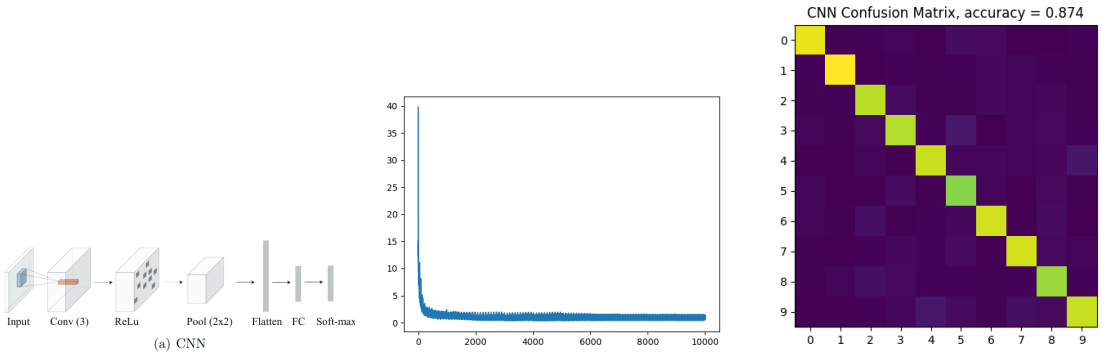


Figure 4: The network architecture, loss per iteration and the resulting confusion matrix for the CNN

For the CNN backpropagation, I started with the pooling layer. Here, $(\frac{dl}{dx})_i = (\frac{dl}{dy})_i$ if the input x_i is the maximum of its 2×2 neighbors, and 0 otherwise. The flattening back-propagation is just a simple reshape, and the ReLU backprop could be reused from the MLP. Lastly, the backpropagation for the convolutional layer was calculated as follows: $(\frac{dl}{db})_k = \sum_i \sum_j (\frac{dl}{dy})_{ijk}$ and

```
dl_dw = dl_dy.reshape((3,-1)).dot(xcol).reshape(w_conv.shape)
```

where xcol is the result from im2col. My CNN had a testing accuracy of 87%. Similar to the MLP, I think this is due to hyperparameter tuning, but even after running a grid search overnight I could not get my CNN to beat the MLP accuracy. The values I tested were [0.05, 0.1, 0.3] for the learning rate, [0.5, 0.75, 0.9] for the decay rate, and [16, 32, 64, 128] for the batch sizes. Results of the my grid search are included in the zip file submission.