

CSCI 5561

Homework 4 – Convolutional Neural Network

He Zhou

November 23, 2021

MNIST Data: The goal of this assignment is to implement a single-layer linear perceptron, a single-layer perceptron, a multi-layer perceptron and a convolutional neural network to recognize hand-written digits in the MNIST dataset. The size of each image is $196 = 14 \times 14$ with a label in $\{0, 1, 2, \dots, 9\}$. For each neural network, the stochastic gradient descent method will be implemented to optimize the objective function $\sum_{i=1}^n \ell(y_i, f(x_i; \theta))$, where $\ell()$ is the loss function and $\{f(\cdot; \theta)\}$ is the family of function with coefficient θ .

List of Functions: `get_mini_batch()`: randomly permutes the order of images to build the mini-batch of size `batch_size = 32` for stochastic gradient descent. Each batch of images is a matrix with size $196 \times \text{batch_size}$, and each batch of labels is a matrix with size $10 \times \text{batch_size}$, where the labels are converted to $\{0, 1\}^{10}$ using one-hot encoding.

`fc()` and `fc_backward()`: `fc` is the fully connected layer, i.e., a linear transform of $\mathbf{x} \in \mathbb{R}^{n \times 1}$: $\mathbf{y} = \mathbf{w}\mathbf{x} + \mathbf{b}$, where $\mathbf{w} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^{m \times 1}$. `fc_backward` computes the partial derivative w.r.t. input \mathbf{x} , weights \mathbf{w} , and bias \mathbf{b} using the loss derivative w.r.t. the output \mathbf{y} .

`loss_euclidean()`: Compute the Euclidean distance $L = \|\mathbf{y} - \tilde{\mathbf{y}}\|^2$ and the loss derivative w.r.t. the prediction $\mathbf{y_tilde}$, i.e., $2(\tilde{\mathbf{y}} - \mathbf{y})$.

`loss_cross_entropy_softmax()`: Add a soft-max layer to input \mathbf{x} , i.e., $\tilde{y}_j = e^{x_j} / (\sum_j e^{x_j})$ and compute the cross-entropy between the two distributions $L = -\sum_{j=1}^m y_j \log(\tilde{y}_j)$. The loss derivative w.r.t. the input \mathbf{x} is $\tilde{\mathbf{y}} - \mathbf{y}$.

`relu()` and `relu_backward()`: `relu` is an activation unit, $\max(\cdot, 0)$. `relu_backward` computes the loss derivative w.r.t. the relu input \mathbf{x} : $\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \cdot I\{\mathbf{x} > 0\}$.

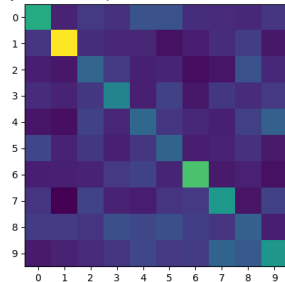
`conv()` and `conv_backward()`: `conv` is a convolutional operation with weights $\mathbf{w_conv} \in \mathbb{R}^{h \times w \times C_1 \times C_2}$ and bias $\mathbf{b_conv} \in \mathbb{R}^{C_2 \times 1}$. Zeros are padded at the boundary of the input image. `conv_backward` computes the loss derivatives w.r.t. the weights and bias. We employ the `im2col` and `col2im`¹ operations that convert the convolution operation into a matrix multiplication to simplify the computation.

`pool2x2()` and `pool2x2_backward()`: 2×2 max-pooling operation and its loss derivative w.r.t. the input.

`flattening()` and `flattening_backward()`: Flattening operation and its loss derivative w.r.t. the input.

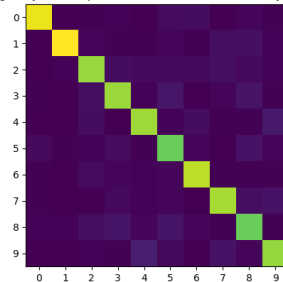
Single-layer Linear Perceptron: Function `train_slp_linear` with functions `fc`, `fc_backward` and `loss_euclidean` implements a single layer linear perceptron with stochastic gradient descent method. The function $f()$ for this single-layer linear perceptron is a fully connected layer: $\sigma_{fc}(\mathbf{x}; \mathbf{w}, \mathbf{b}) = \mathbf{w}\mathbf{x} + \mathbf{b}$. The loss function is the Euclidean loss: $\ell(\mathbf{y}, \tilde{\mathbf{y}}) = \|\mathbf{y} - \tilde{\mathbf{y}}\|^2$. When implementing the stochastic gradient descent method, the learning rate and the decaying rate are tuned as $\gamma = 0.01$, $\lambda = 0.1$, and the maximum number of iterations is $nIters = 2000$. The visualization of confusion matrix is given in Figure (1a) and the accuracy of the prediction is 0.296. We can see that the single-layer linear perceptron does not perform well.

Single-layer Linear Perceptron Confusion Matrix, accuracy = 0.296



(1a) Single-layer linear perceptron

Single-layer Perceptron Confusion Matrix, accuracy = 0.850



(1b) Single-layer perceptron

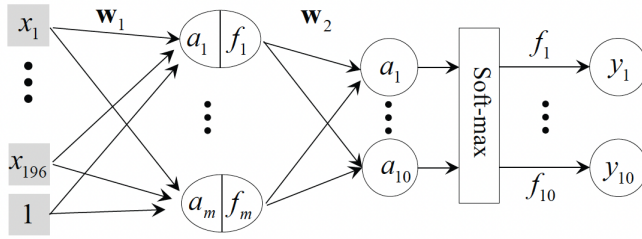
¹https://leonardoaraujosantos.gitbook.io/artificial-intelligence/machine_learning/deep_learning/convolution_layer/making_faster

Single-layer Perceptron: The single-layer perceptron (`train_slp`) adds a soft-max layer to the previous single-layer perceptron and uses the cross-entropy as the loss function: $f(\mathbf{x}; \mathbf{w}, \mathbf{b}) = \sigma_{\text{soft_max}}(\sigma_{\text{fc}}(\mathbf{x}; \mathbf{w}, \mathbf{b}))$, $\ell(\mathbf{y}, \tilde{\mathbf{y}}) = -\sum_{j=1}^m y_j \log(\tilde{y}_j)$. When implementing the stochastic gradient descent method, the learning rate and the decaying rate are tuned as $\gamma = 0.135$, $\lambda = 0.895$, and the maximum number of iterations is $n\text{Iters} = 2500$. The visualization of confusion matrix is given in Figure (1b) and the accuracy of the prediction is 0.850. We can see that the single-layer perceptron performs much better than the single-layer linear perceptron.

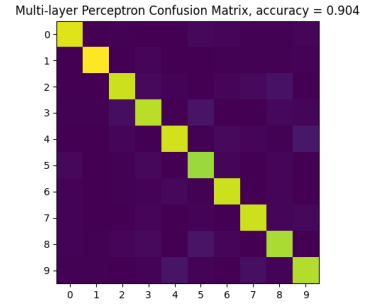
Multi-layer Perceptron: The multi-layer perceptron (`train_mlp`) implements a fully-connected layer, a relu layer, another fully-connected layer, and a soft-max layer sequentially with the cross-entropy loss, shown in Figure (2a).

$$f(\mathbf{x}; \mathbf{w}_1, \mathbf{b}_1, \mathbf{w}_2, \mathbf{b}_2) = \sigma_{\text{soft_max}}(\sigma_{\text{fc}}(\sigma_{\text{relu}}(\sigma_{\text{fc}}(\mathbf{x}; \mathbf{w}_1, \mathbf{b}_1)); \mathbf{w}_2, \mathbf{b}_2)).$$

The back-propagation algorithm is implemented to calculate the partial derivatives w.r.t. the coefficients. When implementing the stochastic gradient descent method, the learning rate and the decaying rate are tuned as $\gamma = 0.49$, $\lambda = 0.905$, and the maximum number of iterations is $n\text{Iters} = 5000$. The visualization of confusion matrix is given in Figure (2b) and the accuracy of the prediction is 0.904. We can see that adding a relu layer and a second fully-connected layer improves the prediction accuracy.



(2a) Multiple-layer perceptron

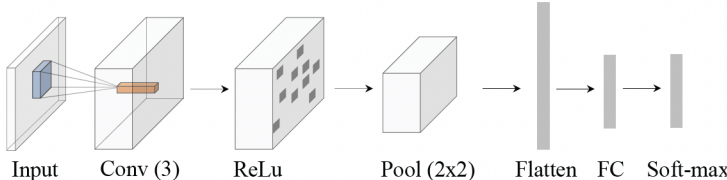


(2b) Confusion matrix with accuracy 0.904

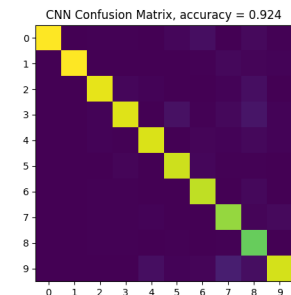
Convolutional Neural Network: The CNN (`train_cnn`) is composed of a single channel input ($14 \times 14 \times 1$) \rightarrow the convolutional layer (3 convolution with 3 channels and stride 1) \rightarrow ReLu layer \rightarrow Max-pooling layer (2×2 with stride 2) \rightarrow Flattening layer \rightarrow Fully-connected layer \rightarrow Soft-max layer, shown in Figure (3a).

$$f(\mathbf{x}; \mathbf{w}_{\text{conv}}, \mathbf{b}_{\text{conv}}, \mathbf{w}_{\text{fc}}, \mathbf{b}_{\text{fc}}) = \sigma_{\text{soft_max}}(\sigma_{\text{fc}}(\sigma_{\text{flat}} \circ \sigma_{\text{pool}} \circ \sigma_{\text{relu}} \circ \sigma_{\text{conv}}(\mathbf{x}; \mathbf{w}_{\text{conv}}, \mathbf{b}_{\text{conv}}); \mathbf{w}_{\text{fc}}, \mathbf{b}_{\text{fc}}))$$

The back-propagation algorithm is implemented to calculate the partial derivatives w.r.t. the coefficients. When implementing the stochastic gradient descent method, the learning rate and the decaying rate are tuned as $\gamma = 0.5$, $\lambda = 0.89$, and the maximum number of iterations is $n\text{Iters} = 1000$. The visualization of confusion matrix is given in Figure (3b) and the accuracy of the prediction is 0.924. We can see that implementing a CNN could further improve the prediction accuracy.



(3a) Convolutional neural network perceptron



(3b) Confusion matrix with accuracy 0.924

References

- S. Lazebnik, C. Schmid, and J. Ponce, “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, vol. 2. IEEE, 2006, pp. 2169–2178.