Machine Learning Assignment 2
Neural Networks

Submission deadline: November 21, 2024

Please submit your solution in PDF format (preferably, but not necessarily, LaTeX—this .tex file can be found on iCorsi). Handwriting and scanned documents are not allowed. In case you need further help, please write on iCorsi or contact me at vincent.herrmann@usi.ch.

1 Problem 1. Calculating Gradients (20 points)

A two-layer neural network is defined by the following equation:

$$y_k := \mathbf{w}^{(2)\top} f\big(W^{(1)}\mathbf{x}_k + \mathbf{b}^{(1)}\big) + b^{(2)}$$

The values of the input vectors $\mathbf{x}_k \in \mathbb{R}^4$ (summarized in the matrix $X \in \mathbb{R}^{3\times 4}$), targets t_k (summarized in the vector $\mathbf{t} \in \mathbb{R}^3$), weights $W^{(1)} \in \mathbb{R}^{2\times 4}$ and $\mathbf{w}^{(2)} \in \mathbb{R}^2$ as well as biases $\mathbf{b}^{(1)} \in \mathbb{R}^2$ and $b^{(2)} \in \mathbb{R}$ are given below.

$$X = \begin{bmatrix} — & \mathbf{x}_1 & — \\ — & \mathbf{x}_2 & — \\ — & \mathbf{x}_3 & — \end{bmatrix} = [[0.2, -0.4, 0.1, 0.5], [0.8, 1.0, -0.1, 0.3], [-0.5, 0.9, 0.2, -0.8]]$$

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = [-0.5, 0.1, 0.8]$$

$$W^{(1)} = [[-0.6, 0.7, 0.9, -0.3], [0.4, 0.5, -1.0, 0.1]]$$

$$\mathbf{b}^{(1)} = [-0.4, -0.5]$$

$$\mathbf{w}^{(2)} = [0.2, 0.6]$$

$$b^{(2)} = -0.7$$

The function f is the ReLU nonlinearity, applied element-wise. The loss of the model is

$$L := \frac{1}{2}\sum_{k=1}^{3}(y_k - t_k)^2.$$

- Numerically calculate the loss and the gradients of L with respect to $W^{(1)}$, $\mathbf{w}^{(2)}$, $\mathbf{b}^{(1)}$ and $b^{(2)}$, and explain your process. You may use a calculator or math software (numpy, matlab, ...), but no auto-differentiation libraries like PyTorch or Jax. Tip: If you use matrix-matrix multiplications involving X, you don't need to do multiple explicit forward and backward passes.

- It is more computationally efficient to calculate gradients in a neural network when we start the chain of derivatives on the side of the loss and then going backward, as opposed to starting on the input side and going forward. Explain why this is the case using the given example.

Problem 2. Gradient Descent (15 points)

If we choose a learning rate that is too high, gradient descent can diverge (even in the case of convex functions). Assume we want to use gradient descent to find the value of $\mathbf{u} = \begin{bmatrix} x \\ y \end{bmatrix}$ that minimizes $f(\mathbf{u}) = \mathbf{u}^T \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix} \mathbf{u}$, or equivalently $f(x, y) = 5x^2 + 2y^2 + 4xy$. The update via gradient descent can be written as $\mathbf{u}_{n+1} = \mathbf{u}_n - \eta \nabla f(\mathbf{u}_n)$, with η being the learning rate. What is the range of values that η can take so that gradient descent eventually converges to the minimum, no matter the starting point? Explain your reasoning–the correct approach is more important than the numerical result.

Problem 3. Convolutional Networks (15 points)

We start with an input image of size 32×32 and three channels. Provide a sequence of alternating convolutional and pooling layers that transform this image into 64 feature maps of size 4×4.

For each convolutional layer, specify the kernel size (e.g., 3×3, 5×5 or 7×7), the number of input channels, and the number of filters. All convolutional layers should have a stride of one in either direction. No padding is used. All pooling layers should be max-pooling layers with a kernel size of 2×2 and a stride of two in each direction. Give the size and the number of the intermediate feature maps (i.e., the outputs of each layer). Calculate the total number of learnable parameters in your network (the number of weights in biases in all layers).

Your answer should look roughly like this:
Architecture 1:

- input feature map: 32×32, 3 feature maps
- conv layer: kernel size ?×?, ? input feature maps, ? output feature maps
- output conv layer: ?×?, ? feature maps
- pooling layer: 2×2
- output pooling layer: ?×?, ? channels/feature maps
- conv layer: ...
- ...
- output pooling/conv layer: 4×4, 64 channels/feature maps

Total number of learnable parameters (weights and biases): ?