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.

$$\begin{aligned}
X &= \begin{bmatrix} — & \mathbf{x}_1 & — \\ — & \mathbf{x}_2 & — \\ — & \mathbf{x}_3 & — \end{bmatrix} = [[0.2, -0.4, 0.1, 0.5], [0.8, 1.0, -0.1, 0.3], [-0.5, 0.9, 0.2, -0.8]] \\
\mathbf{t} &= \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = [-0.5, 0.1, 0.8] \\
W^{(1)} &= [[-0.6, 0.7, 0.9, -0.3], [0.4, 0.5, -1.0, 0.1]] \\
\mathbf{b}^{(1)} &= [-0.4, -0.5] \\
\mathbf{w}^{(2)} &= [0.2, 0.6] \\
b^{(2)} &= -0.7
\end{aligned}$$

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

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

1. 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.

2. 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.

1.1 Solution

1.1.1 Loss

Let's start by computing the ReLU function element-wise: $f(x) = max(0, x)$

$$ReLU = f\big(W^{(1)}X^T + \mathbf{b}^{(1)}\big)$$

$$ReLU = f\big(\begin{bmatrix} -0.6 & 0.7 & 0.9 & -0.3 \\ 0.4 & 0.5 & -1.0 & 0.1 \end{bmatrix} \begin{bmatrix} 0.2 & 0.8 & -0.5 \\ -0.4 & 1.0 & 0.9 \\ 0.1 & -0.1 & 0.2 \\ 0.5 & 0.3 & -0.8 \end{bmatrix} + \begin{bmatrix} -0.4 \\ -0.5 \end{bmatrix}\big)$$

$$ReLU = f\big(\begin{bmatrix} -0.86 & -0.36 & 0.95 \\ -0.67 & 0.45 & -0.53 \end{bmatrix}\big)$$

$$ReLU = \begin{bmatrix} 0 & 0 & 0.95 \\ 0 & 0.45 & 0 \end{bmatrix}$$

Now let's compute Y, since we are computing vector Y instead of each component y_k, $b^{(2)}$ is $\begin{bmatrix} -0.7 & -0.7 - 0.7 \end{bmatrix}$

$$Y = \mathbf{w}^{(2)\top} ReLU + b^{(2)}$$

$$Y = \begin{bmatrix} 0.2 & 0.6 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0.95 \\ 0 & 0.45 & 0 \end{bmatrix} + \begin{bmatrix} -0.7 & -0.7 - 0.7 \end{bmatrix}$$

$$Y = \begin{bmatrix} -0.7 & -0.43 & -0.51 \end{bmatrix}$$

Now using the components of Y and t: to solve $L := \frac{1}{2}\sum_{k=1}^{3}(y_k - t_k)^2$.

$$(y_1 - t_1)^2 = (-0.7 - (-0.5))^2 = (-0.2)^2 = 0.04$$

$$(y_2 - t_2)^2 = (-0.43 - 0.1)^2 = (-0.53)^2 = 0.2809$$

$$(y_3 - t_3)^2 = (-0.51 - 0.8)^2 = (-1.31)^2 = 1.7161$$

$$L = \frac{1}{2}(0.04 + 0.2809 + 1.7161) = \frac{1}{2} \cdot 2.037 = 1.0185$$

1.1.2 Gradients

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial f(\cdot)}\frac{\partial f(\cdot)}{\partial W^{(1)}} = \sum_{k=1}^{3}(y_k - t_k)\cdot w^{(2)} \cdot \mathbf{1}_{ReLU_k>0}\cdot x_k^\top$$

$$\frac{\partial L}{\partial W^{(1)}} = \big((\mathbf{w}^{(2)}\cdot \mathbf{1}_{z>0})\cdot (Y^T - t)\big)\cdot X$$

$$\frac{\partial L}{\partial W^{(1)}} = \begin{bmatrix} -0.131 & -0.393 & -0.0524 & 0 \\ -0.159 & -0.477 & -0.0636 & 0 \end{bmatrix}$$

$$\frac{\partial L}{\partial \mathbf{w}^{(2)}} = \frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial w^{(2)}} = \sum_{k=1}^{3}(y_k - t_k) f\big(W^{(1)}x_k + \mathbf{b}^{(1)}\big)$$

$$\frac{\partial L}{\partial \mathbf{w}^{(2)}} = ReLU \cdot (Y^T - t)$$

$$\frac{\partial L}{\partial \mathbf{w}^{(2)}} = \begin{bmatrix} -1.2445 \\ -0.2385 \end{bmatrix}$$

$$\frac{\partial L}{\partial \mathbf{b}^{(1)}} = \frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial f(\cdot)}\frac{\partial f(\cdot)}{\partial b^{(1)}} = \sum_{k=1}^{3}(y_k - t_k)\cdot w^{(2)} \cdot \mathbf{1}_{ReLU_k>0}$$

$$\frac{\partial L}{\partial \mathbf{b}^{(1)}} = \big(\mathbf{w}^{(2)}\cdot \mathbf{1}_{z>0}\big)\cdot (Y^T - t)$$

$$\frac{\partial L}{\partial \mathbf{b}^{(1)}} = \begin{bmatrix} -0.262 \\ -0.318 \end{bmatrix}$$

$$\frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial b^{(2)}} = \sum_{k=1}^{3}(y_k - t_k)\cdot 1 = -2.04\cdot 1 = -2.04$$

Intermediate Results

$$\frac{\partial L}{\partial y_k} = \sum_{k=1}^{3}(y_k - t_k) = (-0.2) + (-0.53) + (-1.31) = -2.04$$

$$\frac{\partial y_k}{\partial b^{(2)}} = 1$$

$$\frac{\partial y_k}{\partial w^{(2)}} = f\big(W^{(1)}x_k + \mathbf{b}^{(1)}\big)$$

$$\frac{\partial y_k}{\partial f(\cdot)} = w^{(2)}$$

$$\frac{\partial f(\cdot)}{\partial b^{(1)}} = \mathbf{1}_{ReLU_k>0}$$

$$\frac{\partial f(\cdot)}{\partial w^{(1)}} = \mathbf{1}_{ReLU_k>0} \cdot x_k^T$$

$$\mathbf{1}_{ReLU_k>0} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Note: $w^{(2)} \cdot \mathbf{1}_{ReLU_k>0}$ is done element-wise

1.2 Solution

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 because we can reuse intermediate results (partial derivatives that we already computed) to compute new gradients, saving a lot of computation time and power.

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 2. Solution

Since $f(u)$ can be expressed in a quadratic form $f(x) = x^T A x$ i.e

$$f(u) = \mathbf{u}^T \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix} \mathbf{u}$$

and A is symmetric, we know that the gradient is

$$\nabla f(u) = 2Au = 2 \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix} \mathbf{u}$$

and the Hessian Matrix is

$$\nabla^2 f(u) = 2A = 2 \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}$$

we want to converge to the minimum of $f(u)$, to do this we need find the optimal learning rate η at step n. Since the goal is to converge at the minimum of $f(u)$ then we can minimize f respect to u_{n+1} and find the optimal learning rate η at step n.

$$\frac{df(\mathbf{u}_n - \eta \nabla f(\mathbf{u}_n))}{d\eta} = 0$$

$$\frac{df}{d\eta}(\mathbf{u}_n - \eta \nabla f(\mathbf{u}_n))^T A (\mathbf{u}_n - \eta \nabla f(\mathbf{u}_n)) = 0$$

$$-2(\nabla f(\mathbf{u}_n))^T A\mathbf{u}_n + 2\eta(\nabla f(\mathbf{u}_n))^T A(\nabla f(\mathbf{u}_n)) = 0$$
$$-(\nabla f(\mathbf{u}_n))^T A\mathbf{u}_n + \eta(\nabla f(\mathbf{u}_n))^T A(\nabla f(\mathbf{u}_n)) = 0$$
$$\eta = \frac{(\nabla f(\mathbf{u}_n))^T A\mathbf{u}_n}{(\nabla f(\mathbf{u}_n))^T A(\nabla f(\mathbf{u}_n))}$$

OTHER SOLUTION

To converge, the update of f must be stable:

$$u_{n+1} = u_n - \eta\nabla f(u_n)$$

$$u_{n+1} = (I - 2\eta A)u_n$$

For convergence, the eigenvalues of the matrix $(I - 2\eta A)$ must lie within the unit circle:

$$|1 - 2\eta\lambda_i| < 1 \quad \forall i$$

1. $1 - 2\eta\lambda_i > -1$:
$$2\eta\lambda_i < 2 \quad \Rightarrow \quad \eta < \frac{1}{\lambda_i}$$
2. $1 - 2\eta\lambda_i < 1$:
$$-2\eta\lambda_i < 0 \quad \Rightarrow \quad \eta > 0$$

Together are:

$$0 < \eta < \frac{1}{\lambda_i} \quad \forall i$$

The biggest eigenvalue will limit the condition:

$$0 < \eta < \frac{1}{2\lambda_{\max}}$$

Eigenvalues of matrix A:

$$A = \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}$$

The eigenvalues are:

$$\lambda_1 = 6, \quad \lambda_2 = 1$$

The largest eigenvalue is $\lambda_{\max} = 6$.

$$0 < \eta < \frac{1}{2 \cdot 6} = \frac{1}{12}$$

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 $?\times?$, ? input feature maps, ? output feature maps
- output conv layer: $?\times?$, ? feature maps
- pooling layer: 2×2
- output pooling layer: $?\times?$, ? channels/feature maps
- conv layer: ...
- ...
- output pooling/conv layer: 4×4, 64 channels/feature maps

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

1.3 Problem 3 Solution

Solution: Convolutional Network Design

We aim to design a convolutional network that reduces the input 32×32 image with 3 channels into 4×4 feature maps with 64 channels. Below is the architecture, layer-by-layer computations, and the total number of learnable parameters.

Architecture

1. **Input feature map:** 32×32, 3 feature maps.
2. **Conv Layer 1:**
   - Kernel size: 3×3
   - Input channels: 3
   - Output channels: 16

- Output size:

$$\text{Output size} = (32 - 3 + 1) \times (32 - 3 + 1) = 30 \times 30$$

- Output feature maps: 30×30, 16 feature maps.
- Number of parameters:

$$\text{Weights: } 3 \times 3 \times 3 \times 16 = 432, \quad \text{Biases: } 16$$

$$\text{Total: } 432 + 16 = 448$$

3. **Max-Pooling Layer 1:**
   - Pooling size: 2×2
   - Stride: 2
   - Output size:

$$\text{Output size} = \frac{30}{2} \times \frac{30}{2} = 15 \times 15$$

   - Output feature maps: 15×15, 16 feature maps.

4. **Conv Layer 2:**
   - Kernel size: 3×3
   - Input channels: 16
   - Output channels: 32
   - Output size:

$$\text{Output size} = (15 - 3 + 1) \times (15 - 3 + 1) = 13 \times 13$$

   - Output feature maps: 13×13, 32 feature maps.
   - Number of parameters:

$$\text{Weights: } 3 \times 3 \times 16 \times 32 = 4608, \quad \text{Biases: } 32$$

$$\text{Total: } 4608 + 32 = 4640$$

5. **Max-Pooling Layer 2:**
   - Pooling size: 2×2
   - Stride: 2
   - Output size:

$$\text{Output size} = \frac{13}{2} \times \frac{13}{2} = 6 \times 6$$

   - Output feature maps: 6×6, 32 feature maps.

6. **Conv Layer 3:**
   - Kernel size: 3×3
   - Input channels: 32

- Output channels: 64
- Output size:

$$\text{Output size} = (6-3+1) \times (6-3+1) = 4 \times 4$$

- Output feature maps: 4×4, 64 feature maps.
- Number of parameters:

$$\text{Weights: } 3 \times 3 \times 32 \times 64 = 18432, \quad \text{Biases: } 64$$

$$\text{Total: } 18432 + 64 = 18496$$

Final Architecture Summary

- Input feature map: 32×32, 3 feature maps.
- Conv Layer 1: 3×3 kernel, 3 input channels, 16 output channels.
  - Output: 30×30, 16 feature maps.
- Max-Pooling Layer 1: 2×2 pooling.
  - Output: 15×15, 16 feature maps.
- Conv Layer 2: 3×3 kernel, 16 input channels, 32 output channels.
  - Output: 13×13, 32 feature maps.
- Max-Pooling Layer 2: 2×2 pooling.
  - Output: 6×6, 32 feature maps.
- Conv Layer 3: 3×3 kernel, 32 input channels, 64 output channels.
  - Output: 4×4, 64 feature maps.

Total Number of Learnable Parameters

- Conv Layer 1: 448
- Conv Layer 2: 4640
- Conv Layer 3: 18496

$$\text{Total Parameters} = 448 + 4640 + 18496 = 23584$$

Conclusion

The designed convolutional network outputs 4×4 feature maps with 64 channels and has a total of 23,584 learnable parameters.