

Homework 3 (Neural Networks)

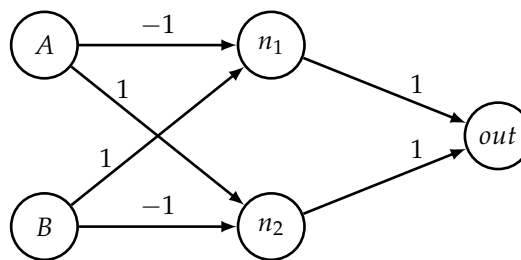
For questions, please refer to Moodle.
 Released on 3 April 2023

GENERAL INSTRUCTIONS

- Submission of solutions is not mandatory but solving the exercises are highly recommended. The master solution will be released next week.
- Part of the exercises are available on Moodle as a quiz. These problems are marked with [🔒].

Exercise 1: Neural Network Learns Boolean Functions

Consider the following neural network with one hidden layer and no bias. The neural network uses the ReLU activation function for the hidden layer, and mean squared error loss. The weights are initialized as follows:



We denote $x = (A, B)^T$, $z = W^{(1)}x$, $v = (n_1, n_2)^T = \sigma(z)$, $out = W^{(2)}v$, where σ is the ReLU activation function.

(a) [🔒] What are weight matrices of layers 1 ($W^{(1)}$) and 2 ($W^{(2)}$)?

1.

$$W^{(1)} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}, W^{(2)} = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

2.

$$W^{(1)} = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}, W^{(2)} = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

3.


$$W^{(1)} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}, W^{(2)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

4.

$$W^{(1)} = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}, W^{(2)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Solution:

$$W^{(1)} \in \mathbb{R}^{2 \times 2} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}, W^{(2)} \in \mathbb{R}^{1 \times 2} = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

(b)  Calculate the forward propagation for the inputs:

- $\mathbf{x}_1 = (A, B)^T = (0, 0)^T$
- $\mathbf{x}_2 = (A, B)^T = (0, 1)^T$
- $\mathbf{x}_3 = (A, B)^T = (1, 0)^T$
- $\mathbf{x}_4 = (A, B)^T = (1, 1)^T$

What Boolean function does this neural network compute?

1. AND
2. OR
3. XOR
4. NAND

Solution:


$$out = W^{(2)} \sigma(W^{(1)} \vec{x})$$

Or in scalar notation:

$$out = \sigma(-A + B) + \sigma(A - B)$$

A	B	Network
0	0	0
0	1	1
1	0	1
1	1	0


The network computes the XOR of A and B.

(c)  What is $\frac{\partial L}{\partial W_{11}^{(2)}}$?

1. $2(y - out) \cdot n_1$
2. $2(out - y) \cdot n_1$
3. $2(y - out) \cdot n_2$
4. $2(out - y) \cdot n_2$

Solution: $\frac{\partial L}{\partial W_{11}^{(2)}}$

$$\begin{aligned}
 \frac{\partial L}{\partial W_{11}^{(2)}} &= \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial W_{11}^{(2)}} \\
 &= \frac{\partial (y - out)^2}{\partial out} \cdot \frac{\partial (W_{11}^{(2)} n_1 + W_{12}^{(2)} n_2)}{\partial W_{11}^{(2)}} \\
 &= 2(out - y) \cdot n_1
 \end{aligned}$$

(d)  What is $\frac{\partial L}{\partial W_{12}^{(1)}}$?

1. $2(out - y) \cdot W_{11}^{(2)} \cdot \sigma'(W_{11}^{(1)} A + W_{12}^{(1)} B) * B$
2. $2(out - y) \cdot W_{12}^{(2)} \cdot \sigma'(W_{11}^{(1)} A + W_{12}^{(1)} B) * B$
3. $2(out - y) \cdot W_{11}^{(2)} \cdot \sigma'(W_{11}^{(1)} A + W_{12}^{(1)} B) * A$
4. $2(out - y) \cdot W_{12}^{(2)} \cdot \sigma'(W_{11}^{(1)} A + W_{12}^{(1)} B) * A$

Solution:

$$\begin{aligned}\frac{\partial L}{\partial W_{12}^{(1)}} &= \frac{\partial L}{\partial out} \cdot \frac{\partial out}{\partial n_1} \cdot \frac{\partial n_1}{\partial z_1} \cdot \frac{\partial z_1}{W_{12}^{(1)}} \\ &= \frac{\partial (y - out)^2}{\partial out} \cdot \frac{\partial (W_{11}^{(2)} n_1 + W_{12}^{(2)} n_2)}{\partial n_1} \cdot \frac{\partial \sigma(z_1)}{\partial z_1} \cdot \frac{\partial (W_{11}^{(1)} A + W_{12}^{(1)} B)}{W_{12}^{(1)}} \\ &= 2(out - y) \cdot W_{11}^{(2)} \cdot \sigma'(W_{11}^{(1)} A + W_{12}^{(1)} B) * B\end{aligned}$$

- (e) [✓] Now, provide weights that implements the logical OR function $Y = x_1 \vee x_2$. Assume the weights can only take values -1, 0, or 1.

1.

$$W^{(1)} \in \mathbb{R}^{2 \times 2} = \begin{pmatrix} 0 & -1 \\ -1 & 1 \end{pmatrix}, W^{(2)} \in \mathbb{R}^{1 \times 2} = (1 \quad 1)$$

2.

$$W^{(1)} \in \mathbb{R}^{2 \times 2} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}, W^{(2)} \in \mathbb{R}^{1 \times 2} = (1 \quad 1)$$

3.

$$W^{(1)} \in \mathbb{R}^{2 \times 2} = \begin{pmatrix} -1 & -1 \\ 0 & 1 \end{pmatrix}, W^{(2)} \in \mathbb{R}^{1 \times 2} = (1 \quad 1)$$

4.

$$W^{(1)} \in \mathbb{R}^{2 \times 2} = \begin{pmatrix} -1 & 1 \\ -1 & 0 \end{pmatrix}, W^{(2)} \in \mathbb{R}^{1 \times 2} = (1 \quad 1)$$

Solution:

$$W^{(1)} \in \mathbb{R}^{2 \times 2} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}, W^{(2)} \in \mathbb{R}^{1 \times 2} = (1 \quad 1)$$

A	B	Network
0	0	0
0	1	1
1	0	1
1	1	1

Exercise 2: MLP vs CNN

You just took a picture which has $1920 * 1080$ pixels. Each pixel has a red, blue, and green value. You would like to design a neural network for this image.

- (a) [✓] Calculate the number of parameters needed for a multilayer perceptron (MLP) that has only one hidden layer of 256 nodes and an output layer of 10 nodes.

Solution: From input to hidden:

$$\#_{input}^{MLP} = 1920 * 1080 * 3 * 256 + 256 = 1'592'525'056$$

From hidden to output:

$$\#_{output}^{MLP} = 256 * 10 + 10 = 2570$$

The total number of parameters is the sum of both:

$$\#_{tot}^{MLP} = \#_{input}^{MLP} + \#_{output}^{MLP} = 1'592'525'056 + 2570 = 1'592'527'626 \approx 1.6 * 10^9$$

- (b) [✓] Now you would like to use a convolutional neural network (CNN). The network has 10 layers, each layer has 64 4×4 filters for EACH channel. Calculate the number of parameters needed for this CNN.

Solution: The first layer receives 3 channels from the input (the RGB channels of the picture) and outputs 64 channels. There are 64 filters, each of the dimension $4 * 4 * 3$.

$$\#_{first}^{CNN} = (4 * 4 * 3 + 1) * 64 = 3136$$

The rest 9 layers receive 64 channels (the output from the previous layer) and output 64 channels. There are 64 filters, each of the dimension $4 * 4 * 64$.

$$\#_{rest}^{CNN} = (4 * 4 * 64 + 1) * 64 = 65600$$

The total number of parameters is

$$\#_{tot}^{CNN} = \#_{first}^{CNN} + 9 * \#_{rest}^{CNN} = 593'536$$

Note this is independent of the input size of the picture.

- (c) [✓] Now consider a simple CNN with only one layer and one 4×4 filter per channel. What dimensions do the outputs of this layer have, if we choose a stride of 2 and apply 2-pixel padding to the input?

Solution: $1920 * 1080 * 3$ convolved with $4 * 4 * 3$ results in

$$\begin{aligned} & \left(\frac{1920 + 2 * padding - filter}{stride} + 1 \right) * \left(\frac{1080 + 2 * padding - filter}{stride} + 1 \right) \\ &= \left(\frac{1920 + 2 * 2 - 4}{2} + 1 \right) * \left(\frac{1080 + 2 * 2 - 4}{2} + 1 \right) \\ &= 961 * 541 \end{aligned}$$

- (d) Prove that there exists a fully connected linear layer of input size and output size that is functionally equivalent to the described convolutional network.

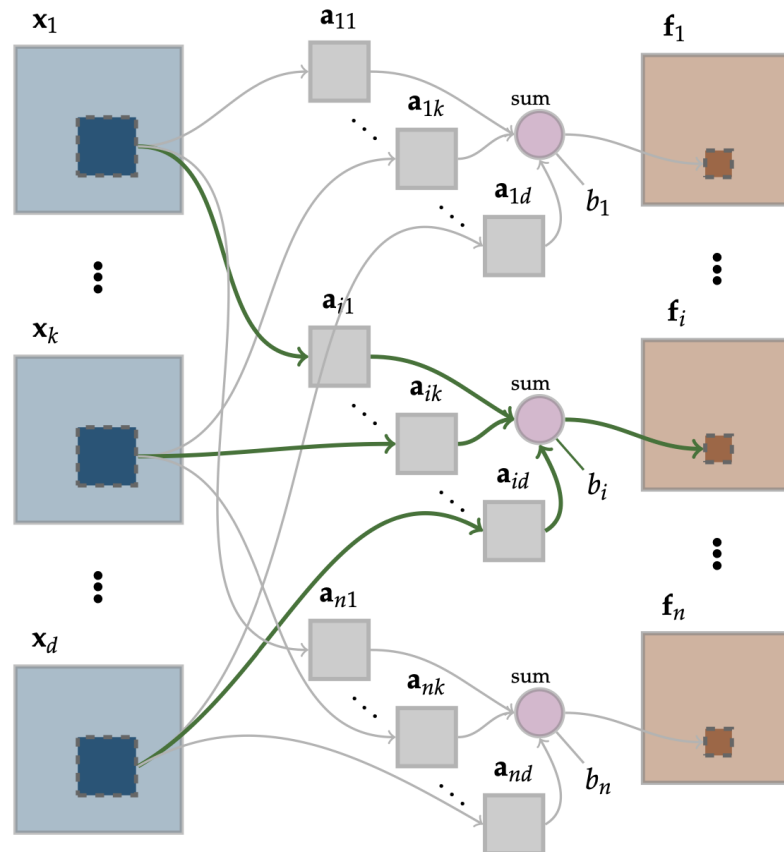
Solution:

Let the input images have size $I_{in} \times I_{in}$ and the output $I_{out} \times I_{out}$. Also, let's say there are d input channels $\mathbf{x}_1, \dots, \mathbf{x}_d$ and n output channels $\mathbf{f}_1, \dots, \mathbf{f}_n$. The convolutional filters $\mathbf{a}_{i1}, \dots, \mathbf{a}_{in}$ and bias b_i correspond to the i -th output channel. Also, consider that each filter has dimension $m \times m$. The idea is to translate both the output and input layers from images to a large flattened vector where each image pixel corresponds to an entry of the flattened vector. Let \mathbf{x} be the vector corresponding to the input and \mathbf{f} the vector of the output. The dimension of \mathbf{x} is d times the input image dimension and \mathbf{f} is n times the output image dimension. Then the convolutional layer can be written equivalently as a fully connected linear layer:

$$\mathbf{f} = \mathbf{Ax} + \mathbf{b}$$

where the i -th row of A consists of zeros and some coefficients corresponding to the convolutional filters that are carefully chosen. In particular, the entry of \mathbf{f} corresponding to the j -th pixel of i -th channel has everywhere zeros except the specific positions where it takes the values $\mathbf{a}_{i1}, \dots, \mathbf{a}_{id}$. Those positions are chosen depending on the padding and stride of the convolution and the output pixel j . Also, the bias vector \mathbf{b} is constructed by consecutively repeating the bias b_i as many times as the output image dimension, for all $i = 1, \dots, n$.

One could also provide a non-constructive proof by simply saying that the convolutional layer is a linear mapping and all linear mappings can be described as linear layers.



- (e) Deduce that the family of functions written as convolutional layers is a subset of those written as fully connected linear layers.

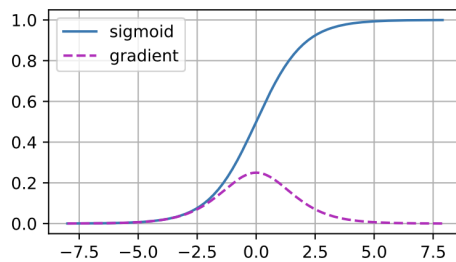
Solution: On our previous statement we proved that every convolutional layer can be written equivalently as a fully connected linear layer. In essence, that means that every function that can be expressed with a convolutional layer, can also be expressed with a linear layer.

Linear layers might be more expressive which means that they can approximate more complicated functions. However, in practice they have many more parameters than the convolutional layers which make them harder to train. On the other hand the convolutional filters can be very useful when it comes to feature extraction of images as they operate like pattern matching detectors, which means that they can detect common patterns in images.

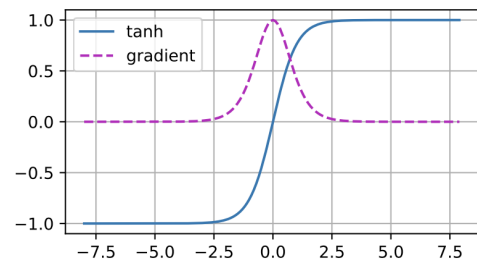
Of course, it is important to mention that there is no rule that convolutional layers always perform better, as it always depend on the application we want to implement.

Exercise 3: Activation Functions

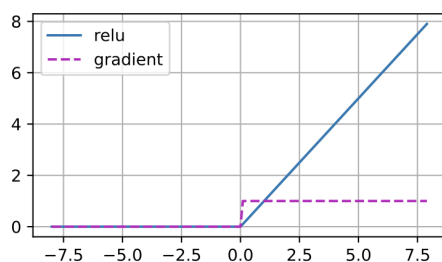
Take a look at the graphs of 4 activation functions and their derivatives:



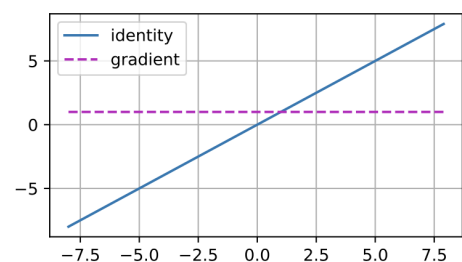
(a) sigmoid



(b) tanh



(c) relu



(d) identity

(a) Which activation function(s) are more prone to vanishing gradients?

1. sigmoid
2. tanh
3. ReLU
4. identity

Solution: (a) sigmoid and (b) tanh

Their derivatives tend to become very small for large and small inputs. This can cause gradients to vanish during backpropagation, as the product of many small derivatives can lead to a gradient that is too small to be useful for training.

(b) Which activation function(s) are differentiable?

1. sigmoid
2. tanh
3. ReLU
4. identity

Solution: All of them are differentiable apart from (c) ReLU, which is not differentiable at $x=0$.

The sigmoid function and the tanh function are both differentiable everywhere, as they are both smooth and continuous functions. The identity function, is also differentiable everywhere with a derivative of 1.

The ReLU function is differentiable everywhere except at exactly 0, where its derivative is undefined. However, in practice, the ReLU function is treated as if it has a derivative of 0 at exactly 0 during backpropagation.

(c) [📄] Which activation function(s) are non-linear?

1. sigmoid
2. tanh
3. ReLU
4. identity

Solution: (a) sigmoid, (b) tanh, and (c) ReLU

The sigmoid and tanh functions are both smooth, S-shaped curves that are non-linear.

The ReLU function is a piecewise linear function, which is also non-linear. The identity function, which simply returns the input value as its output, is a *linear* function.

(d) [📄] Which activation function(s) are zero-centered?

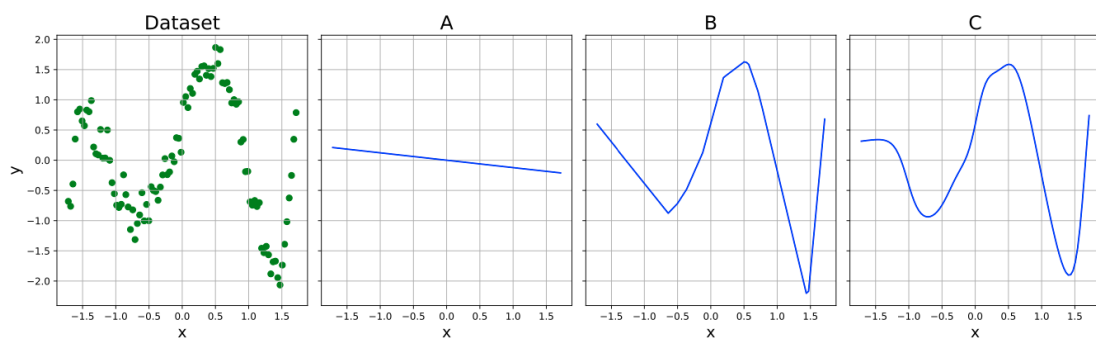
1. sigmoid
2. tanh
3. ReLU
4. identity

Solution: (b) tanh and (d) identity

The term "zero-centered" means that the output of the activation function has a mean of zero when the input is zero-mean (i.e., the mean of the input is zero). This is important for training neural networks with techniques that rely on having zero-centered data, such as batch normalization.

The sigmoid and ReLU functions are not zero-centered. The output of the sigmoid function ranges from 0 to 1, with the midpoint being at 0.5. Similarly, the output of the ReLU function is always positive and starts at 0 for negative inputs.

(e) [📄] The figure below displays a training dataset and the learned function of 3 different neural networks that are trained on the dataset. The dataset consists of 100 scalar input-output pairs. All neural networks have one hidden layer with 20 units but differ in the choice of the activation function that are either the sigmoid, ReLU or identity function. Match the learned output function with the activation function that is used in the corresponding neural network.



1. (A, B, C) = (sigmoid, ReLU, identity)
2. (A, B, C) = (ReLU, identity, sigmoid)
3. (A, B, C) = (identity, ReLU, sigmoid)
4. (A, B, C) = (identity, sigmoid, ReLU)

Solution: (A, B, C) = (identity, ReLU, sigmoid)

With only the identity function, the network can not learn any non-linear relationship, thus network A uses the identity function. Network C has a smoother output function compared to B, which means that network C uses sigmoid and B uses ReLU.


Exercise 4: Gradient Descent

Recall the following Gradient Descent Methods:

(Batch) Gradient Descent: uses all examples in the training data.


Stochastic Gradient Descent: uses one randomly chosen example.

Mark the following as (T) rue or (F)alse:

- (a)  Batch Gradient Descent is suited for large number of training data. ☐ True ☒ False

Solution: Batch Gradient Descent computes the gradient of the cost function with respect to the model parameters using the entire training set at once, and then takes a step in the direction of the negative gradient to update the parameters.


While Batch Gradient Descent can be effective for small datasets or problems with simple models, it is often not well-suited for large datasets due to its high computational cost. Computing the gradient over the entire training set can become very slow and memory-intensive for large datasets, making it impractical or infeasible to use.

- (b)  Batch Gradient Descent gives the global optimal solution given sufficient time. ☐ True ☒ False


Solution: Batch Gradient Descent *can* converge to the global optimal solution of the cost function, given sufficient time and under certain conditions, such as the cost function being convex and the learning rate being properly chosen. However, in general and especially for large datasets and complex models, the cost function is non-convex.

- (c)  Stochastic Gradient Descent tends to escape local minima. ☒ True ☐ False

Solution: Stochastic Gradient Descent (SGD) has a higher likelihood of escaping local minima compared to Batch Gradient Descent. This is because SGD updates the model parameters more frequently using small batches of randomly sampled training examples. These random samples introduce noise into the update process, which can cause the optimization process to jump out of local minima and move toward the global minimum.

- (d)  Stochastic Gradient Descent reaches convergence faster than Batch Gradient Descent. ☒ True ☐ False

Solution: Stochastic Gradient Descent (SGD) generally reaches convergence faster than Batch Gradient Descent, especially for large datasets. This is because SGD updates the model parameters more frequently using small batches of randomly sampled training examples, which allows it to make progress more quickly toward the optimal solution.

- (e)  Batch Gradient Descent is faster and less computationally expensive.

☐ True ☒ False

Solution: Batch Gradient Descent is generally slower and more computationally expensive than Stochastic Gradient Descent (SGD) because it requires computing the gradients on the entire training set at each iteration. This can be especially problematic for large datasets or complex models, where each iteration can be very slow and memory-intensive. In contrast, SGD only requires computing the gradients on a small subset of the training set at each iteration, which can be much faster and more memory-efficient.