Artificial Neural Networks

Definition. An **artificial neural network** (**ANN**) or simply **neural network** (**NN**) is a function $\mathbb{R}^m \to \mathbb{R}^n$ determined by a composition of linear transformations, translations, and non-linear functions. We refer to the non-linear functions as **activation functions**.

Example. Define the functions

$$f_1 : \mathbb{R}^3 \to \mathbb{R}^3,\ f_1\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} \max\{0, x\} \\ \max\{0, y\} \\ \max\{0, z\} \end{bmatrix} \qquad f_2 : \mathbb{R}^2 \to \mathbb{R}^2,\ f_2\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} \dfrac{e^x}{e^x + e^y} \\ \dfrac{e^y}{e^x + e^y} \end{bmatrix},$$

translations

$$S_1 : \mathbb{R}^3 \to \mathbb{R}^3,\ S_1(\mathbf{x}) = \mathbf{x} + \mathbf{b}_1,\ \mathbf{b}_1 = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \qquad S_2 : \mathbb{R}^2 \to \mathbb{R}^2,\ S_2(\mathbf{x}) = \mathbf{x} + \mathbf{b}_2,\ \mathbf{b}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

and linear transformations

$$T_1 : \mathbb{R}^4 \to \mathbb{R}^3,\ T_1(\mathbf{x}) = A_1\mathbf{x},\ A_1 = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & -2 \\ -1 & 0 & 1 & 1 \end{bmatrix} \quad T_2 : \mathbb{R}^3 \to \mathbb{R}^2,\ T_2(\mathbf{x}) = A_2\mathbf{x},\ A_2 = \begin{bmatrix} 0 & 1 & 3 \\ 1 & -1 & 0 \end{bmatrix}$$

Then the function $M : \mathbb{R}^4 \to \mathbb{R}^2$ defined by $M = f_2 \circ S_2 \circ T_2 \circ f_1 \circ S_1 \circ T_1$ is a neural network.

(1) With the neural network M as defined in the previous example, compute $M\left(\begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \end{bmatrix}\right)$.

Note: The activation function f_1 is known as ReLU which is short for rectified linear unit and the activation function f_2 is known as softmax. Both are defined more generally as functions $\mathbb{R}^n \to \mathbb{R}^n$.

(2) Consider the softmax function f_2 in the previous exercise.
   (a) Compute $f_2(\mathbf{x})$ for a couple vectors $\mathbf{x} \in \mathbb{R}^2$. What do you notice about the sum of the components of the vector $f_2(\mathbf{x})$?
   (b) In general, softmax is $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ defined by

$$\sigma\left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} \dfrac{e^{x_1}}{e^{x_1} + \cdots + e^{x_n}} \\ \vdots \\ \dfrac{e^{x_n}}{e^{x_1} + \cdots + e^{x_n}} \end{bmatrix}$$

   Prove the sum of the components of $\sigma(\mathbf{x})$ equals 1 for all $\mathbf{x} \in \mathbb{R}^n$.

Definition. The **loss function** or sometimes **cost function** of a neural network is a function that computes an error value based on a comparison of an observed value and an actual value.

Example. One of the most common loss functions is mean-squared error:

$$\text{MSE}: \ \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{y}_i - \hat{\mathbf{y}}_i||^2$$

Here n is the number of observations, $\mathbf{y}_i$ are the actual values, and $\hat{\mathbf{y}}_i$ are the predicted value.

Note: In the previous example, $||\mathbf{y}_i - \hat{\mathbf{y}}_i||$ refers to the magnitude of the vector $\mathbf{y}_i - \hat{\mathbf{y}}_i$. If y_i and $\hat{y}_i$ are real numbers (vectors with one component), then $||y_i - \hat{y}_i|| = |y_i - \hat{y}_i|$ is the familiar absolute value.

(3) Suppose we have the following dataset:

x	y
1	2
2	4
3	5

Define the neural network $N_0 : \mathbb{R} \to \mathbb{R}$ by $N_0(x) = 2x + 1$. Given an x-value, We will use this model to predict its corresponding y-value.

(a) We can rewrite N_0 as a composition of a linear transformation and translation. What are they?

(b) Compute the MSE for N_0 based on the dataset. That is, $\hat{y}_i = N_0(x_i)$.

(c) Now consider the neural network $N_1 : \mathbb{R} \to \mathbb{R}$ by $N_1(x) = x + \frac{5}{9}$. Compute MSE for N_1.

(d) For a fixed dataset, what does MSE depend on? What values did you change in your computations from the previous two parts?

(e) Formally define the MSE loss function for a general neural network $N(x) = mx + b$ for some real numbers m and b. That is, define a function L by stating its domain, codomain, and how it assigns domain elements to elements in the codomain.

(f) Compute ∇L.

(g) Let the *parameters* for N_0 be an initial guess for the minimum of L. Now complete one step of the gradient descent algorithm with learning rate $l = \frac{1}{6}$. That is, let $(m_0, b_0) = (2, 1)$ and compute

$$(m_1, b_1) = (2, 1) - \frac{1}{6}\nabla L(2, 1)$$

Hint: You should get the *parameters* for N_1.

(h) Complete another step of gradient descent to determine a new model N_2 and compute MSE for that model.

(i) Compare the MSEs of N_0, N_1, and N_2. Which neural network predicts the y-values the best?

Congratulations! You've trained your first neural network!

Observation: To train a neural network, we conduct gradient descent on a loss function which depends solely on the *parameters* of our neural network.

Definition. The **parameters** of a neural network are the collection of components in the matrices and vectors that correspond to the linear transformations and translations of the neural network.

(4) How many parameters do each of the neural networks N_k, $k = 0, 1, 2$ from the previous exercise have?

(5) Recall the model M from the example on page 1.
   (a) How many parameters does M have?
   (b) If we use MSE to train M, carefully describe the loss function.
   (c) Discuss with your group what gradient descent would entail for such a loss function.

(6) Let's revisit the neural network architecture $N(x) = mx + b$ from earlier. Suppose we add an activation function α to this model so that the neural network is now of the form $N(x) = \alpha(mx + b)$. If we use the MSE loss function, how does the activation function affect our gradient descent algorithm? Specifically, what named rule would you need to compute the partial derivatives in the gradient vector?