Mathematics of Image and Data Analysis
Math 5467

Lecture 22: Neural Networks

Instructor: Jeff Calder
Email: jcalder@umn.edu

http://www-users.math.umn.edu/~jwcalder/5467S21

Last time

- Graph-based embeddings (spectral and t-SNE)

Today

- Neural networks

Artificial neural networks

(Artificial) neural networks are parameterized functions made up of simple building blocks: linear functions and simple nonlinearities. The basic building block is a neuron

$$f(x) = \sigma(\omega^T x + b),$$

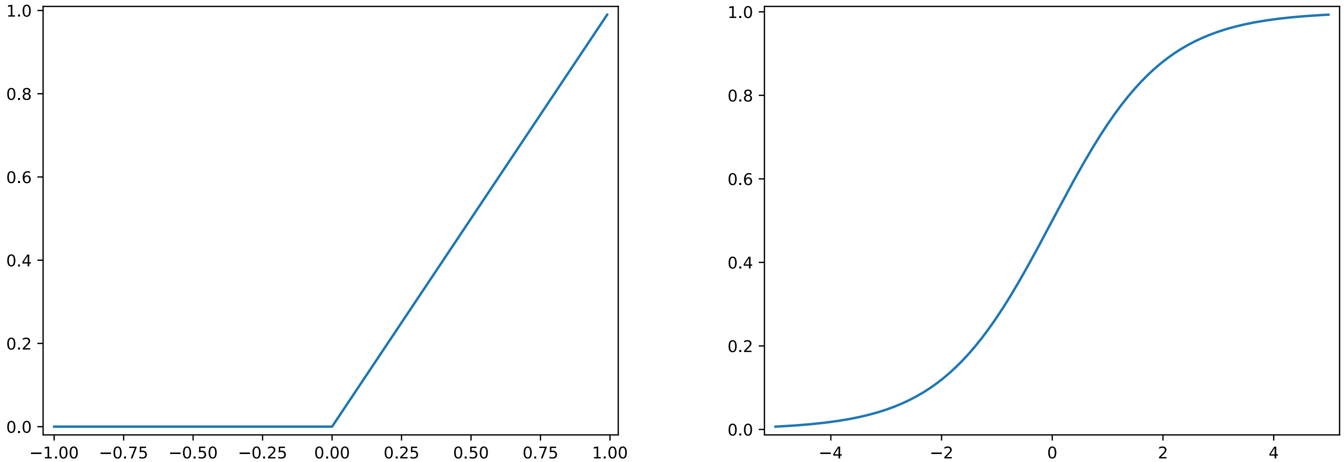
which is a function $f : \mathbb{R}^n \to \mathbb{R}$. Common choices for the activation function σ is the rectified linear unit (ReLU)

$$\sigma(t) = \max\{t, 0\}. \tag{1}$$

and the *sigmoid* activation function

$$\sigma(t) = \frac{1}{1 + e^{-t}}. \tag{2}$$

Common activation functions



(a) ReLU (b) Sigmoid

Figure 1: Plots of the ReLU and Sigmoid activation functions. Both activation functions have the behavior that they give zero, or close to zero, responses when the input is below a certain threshold, and give positive responses above.

Neural network

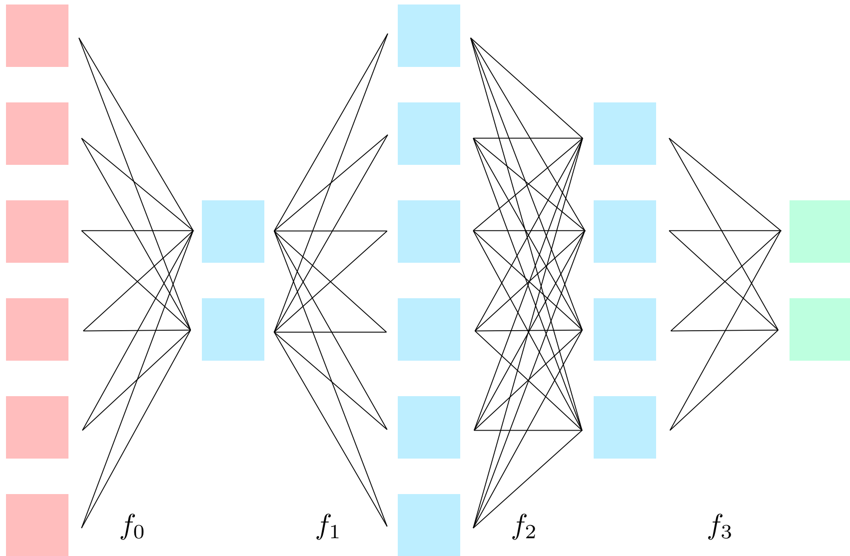


Figure 2: An example of a fully connected neural network with three hidden layers. The blue nodes are the hidden layers, the red is the input, and the green is the output. The hidden layers have width $n_1 = 2$, $n_2 = 6$, and $n_3 = 4$ and the number of input variables is $n_0 = 6$.

Neural network

In more compact notation, we can write a fully connected neural network with L layers recursively as

$$f_k = \sigma_k(W_k f_{k-1} + b_k), \quad k = 1, \ldots, L, \tag{3}$$

where

- $f_0 \in \mathbb{R}^{n_0}$ is the input to the network,
- $f_k \in \mathbb{R}^{n_k}$ for $k = 1, \ldots, L-1$ are the values of the network at the hidden layers,
- f_L is the output of the neural network,
- n_k is the number of hidden nodes in the k^{th} layer,
- The weights $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$ and biases $b_k \in \mathbb{R}^{n_k}$ are the learnable parameters in the neural network.

Loss

The output of the neural network $f_L \in \mathbb{R}^{n_L}$ is typically fed into a loss function

$$\mathcal{L} : \mathbb{R}^{n_L} \to \mathbb{R},$$

which measures the performance of the network for the given learning task.

Typically the loss has the form

$$\mathcal{L}(W_1, b_1, \ldots, W_L, b_L) = \sum_{i=1}^{m} \ell(f_L(x_i), y_i), \tag{4}$$

where (x_i, y_i) for $i = 1, \ldots, m$ are the training data. Here, we write $f_L(x)$ to denote the value of the output of the network f_L given the input is $f_0 = x$.

Neural networks are trained by minimizing the loss function $\mathcal{L}$ with gradient descent.

Gradient descent

Let $\frac{\partial \mathcal{L}}{\partial W_k}$ and $\frac{\partial \mathcal{L}}{\partial b_k}$ denote the gradients of $\mathcal{L}$ with respect to W_k and b_k, respectively.

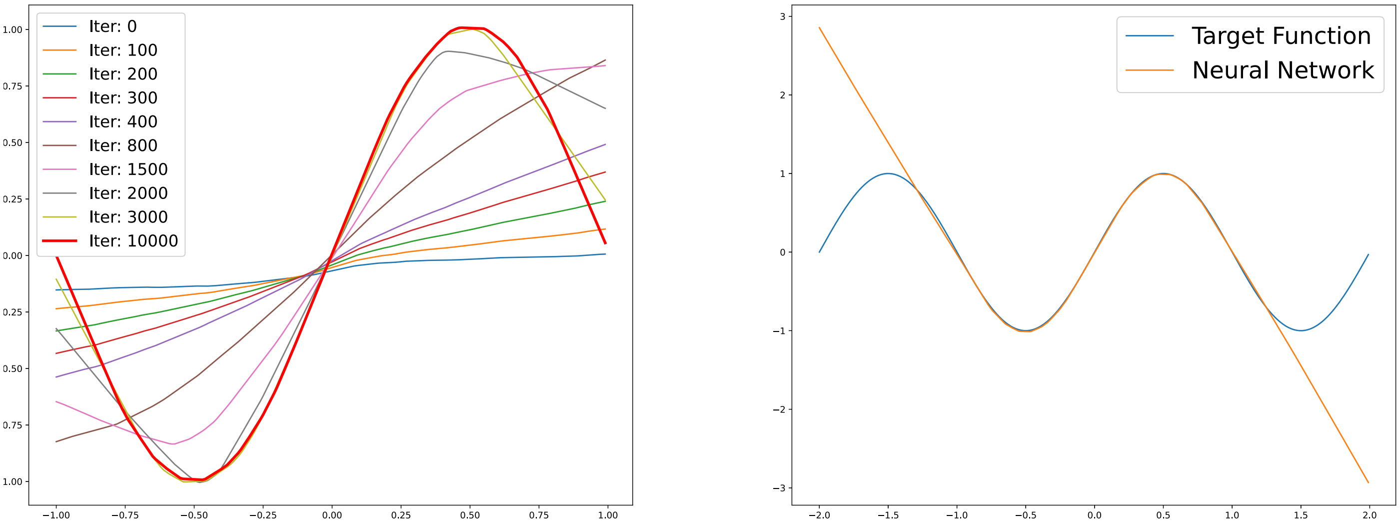
- The gradient $\frac{\partial \mathcal{L}}{\partial W_k}$ is the $n_k \times n_{k-1}$ matrix whose (i,j) entry is the partial derivative of $\mathcal{L}$ in $W_k(i,j)$.
- The gradient $\frac{\partial \mathcal{L}}{\partial b_k} \in \mathbb{R}^{n_k}$ is the vector whose i^{th} entry is the partial derivative of $\mathcal{L}$ in $b_k(i)$.

Gradient descent for minimizing $\mathcal{L}$ corresponds to updating the weights W_k and biases b_k according to

$$W_k^{j+1} = W_k^j - \alpha \frac{\partial \mathcal{L}}{\partial W_k} \quad \text{and} \quad b_k^{j+1} = b_k^j - \alpha \frac{\partial \mathcal{L}}{\partial b_k}, \tag{5}$$

where $\alpha > 0$ is the time step, also called the *learning rate*.

Toy example



(a) Training iterations

(b) Neural network vs target

Figure 3: A toy example of fitting the function $\sin(\pi x)$ with a 2-layer neural network with 100 hidden nodes. The loss is $\mathcal{L} = \sum_{i=1}^{m} |f_L(x_i) - \sin(\pi x_i)|$ for evenly spaced points $-1 = x_1 \leq x_2 \leq \cdots \leq x_m = 1$.

Stochastic Gradient Descent (SGD)

For modern machine learning problems with very large training sets, it is sometimes impractical to compute the full gradients $\frac{\partial \mathcal{L}}{\partial W_k}$ and $\frac{\partial \mathcal{L}}{\partial b_k}$, since the loss involves *all* of the training data.

Stochastic gradient descent (SGD) fixes this by computing the gradient of the loss over a random subset of the training data

$$\widetilde{\mathcal{L}}(W_1, b_1, \ldots, W_L, b_L) = \sum_{i \in I} \ell(f_L(x_i), y_i),$$

where $I \subset \{1, 2, \ldots, n\}$ is a random subset, called a *mini-batch*. The mini-batch changes at each iteration of SGD.

One pass over all the mini-batches in the dataset is called an *epoch*, and training usually proceeds for some number of ephochs, say 100.

Momentum descent

Various other trickes are used in the optimization, such as *momentum*

$$W_k^{j+1} = W_k^j - \alpha \frac{\partial \mathcal{L}}{\partial W_k} + \beta(W_k^j - W_k^{j-1}),$$

and

$$b_k^{j+1} = b_k^j - \alpha \frac{\partial \mathcal{L}}{\partial b_k} + \beta(b_k^j - b_k^{j-1}),$$

where $\beta \in [0, 1]$ is the momentum parameter. Momentum can help to speed up convergence of gradient descent.

We will peform a careful analysis of gradient descent, SGD and momentum descent later.

Back Propagation

For notational simplicity, we will write

$$z_k = W_k f_{k-1} + b_k, \tag{6}$$

so that $f_k = \sigma_k(z_k)$. Let $\frac{\partial \mathcal{L}}{\partial z_k} \in \mathbb{R}^{n_k}$ denote the gradient of $\mathcal{L}$ with respect to z_k. We also let D_k be the diagonal $n_k \times n_k$ matrix with diagonal entries given by the vector $\sigma_k'(z_k)$. That is

$$D_k = \mathrm{diag}(\sigma_k'(z_k)).$$

Theorem 1 (Back propagation)**.** *For $k = 2, \ldots, L$ we have*

$$\boxed{\frac{\partial \mathcal{L}}{\partial z_{k-1}} = D_{k-1} W_k^T \frac{\partial \mathcal{L}}{\partial z_k},} \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial z_k} f_{k-1}^T, \quad \textit{and} \quad \frac{\partial \mathcal{L}}{\partial b_k} = \frac{\partial \mathcal{L}}{\partial z_k}. \tag{8}$$

$$z_k = W_k f_{k-1} + b_k$$

$$\boxed{\begin{aligned} &W_k \sim n_k \times n_{k-1} \\ &b_k \in \mathbb{R}^{n_k} \end{aligned}}$$

$$z_k(i) = \sum_{\ell=1}^{n_{k-1}} W_k(i,\ell) f_{k-1}(\ell) + b_k(i)$$

$$\frac{\partial L}{\partial W_k(i,j)} = \frac{\partial L}{\partial z_k(i)} \cdot \frac{\partial z_k(i)}{\partial W_k(i,j)}$$

chain rule

$$= \frac{\partial L}{\partial z_k(i)} f_{k-1}(j)$$

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial z_k} f_{k-1}^T$$

rank-one matrix

$$\frac{\partial L}{\partial b_k(i)} = \frac{\partial L}{\partial z_k(i)} \underbrace{\frac{\partial z_k(i)}{\partial b_k(i)}}_{=1} = \frac{\partial L}{\partial z_k(i)}$$

$$\frac{\partial L}{\partial b_k} = \frac{\partial L}{\partial z_k}.$$ This proves (8)

For (7) note $f_{k-1} = \sigma(z_{k-1})$

$$z_k = w_k f_{k-1} + b_k = w_k \sigma(z_{k-1}) + b_k$$

$$\boxed{z_k(i) = \sum_{\ell=1}^{n_{k-1}} w_k(i,\ell)\, \sigma(z_{k-1}(\ell)) + b_k(i)}$$

$$\frac{\partial z_k(i)}{\partial z_{k-1}(j)} = W_k(i,j)\,\sigma'(z_{k-1}(j))$$

$$\frac{\partial L}{\partial z_{k-1}(j)} = \sum_{i=1}^{n_k} \frac{\partial L}{\partial z_k(i)} \frac{\partial z_k(i)}{\partial z_{k-1}(j)}$$

$$= \sum_{i=1}^{n_k} \frac{\partial L}{\partial z_k(i)} W_k(i,j)\,\sigma'(z_{k-1}(j))$$

$$= \sigma'(z_{k-1}(j)) \sum_{i=1}^{n_k} \frac{\partial L}{\partial z_k(i)} W_k(i,j)$$

$$= \sigma'(z_{k-1}(j)) \left[W_k^T \frac{\partial L}{\partial z_k} \right](j)$$

$$= \Big[\underbrace{\mathrm{diag}\left(\sigma'(z_{k-1})\right)}_{D_{k-1}} W_k^T \frac{\partial L}{\partial z_k}\Big](j)$$

$$= \left(D_{k-1} W_k^T \frac{\partial L}{\partial z_k}\right)(j)$$

$$\Rightarrow \quad \frac{\partial L}{\partial z_{k-1}} = D_{k-1} W_k^T \frac{\partial L}{\partial z_k} \qquad \blacksquare$$

Intro to Pytorch (.ipynb)