Solving Deeper Multilayer Perceptrons (MLPs)

Kate Farrahi

ECS Southampton

February 12, 2019

1

[1] Some of this lecture has been motivated by F. Fleuret's course EE-559 at EPFL.

What is Vectorization?

- Vectorization helps improve the speed of your code by removing for loops

What is Vectorization?

- Vectorization helps improve the speed of your code by removing for loops
- The idea is that we want to apply a function to every element in a vector

What is Vectorization?

- Vectorization helps improve the speed of your code by removing for loops
- The idea is that we want to apply a function to every element in a vector
- The activation of the jth neuron in the lth layer is given by

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \tag{1}$$

What is Vectorization?

- Vectorization helps improve the speed of your code by removing for loops
- The idea is that we want to apply a function to every element in a vector
- The activation of the jth neuron in the lth layer is given by

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \tag{1}$$

- The model parameters can be represented in a matrix form by defining a weight matrix w^l for each layer, l and a bias vector, b^l

What is Vectorization?

- Vectorization helps improve the speed of your code by removing for loops
- The idea is that we want to apply a function to every element in a vector
- The activation of the jth neuron in the lth layer is given by

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \tag{1}$$

- The model parameters can be represented in a matrix form by defining a weight matrix w^l for each layer, l and a bias vector, b^l
- Vectorization can simplify our notation $a^l = \sigma(w^l a^{l-1} + b^l)$

Threshold Logic Unit

The first mathematical model for a neuron was the Threshold Logic Unit, with Boolean inputs and outputs:

$$f(x) = 1_{\{w \sum_i x_i + b \geq 0\}} \tag{2}$$

In particular it can implement

$or(u, v) = 1_{\{u+v-0.5 \geq 0\}} \quad (w = 1, b = -0.5)$
$and(u, v) = 1_{\{u+v-1.5 \geq 0\}} \quad (w = 1, b = -1.5)$
$not(u) = 1_{\{-u+0.5 \geq 0\}} \quad (w = -1, b = 0.5)$ [2]

[2] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115-133, 1943

Perceptron

The perceptron is very similar

$$f(x) = \begin{cases} 1 & \text{if} \quad \sum_i w_i x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

but the inputs are real values and the weights can be different.

This model was originally motivated by biology, with w_i being the synaptic weights, and x_i and f firing rates.

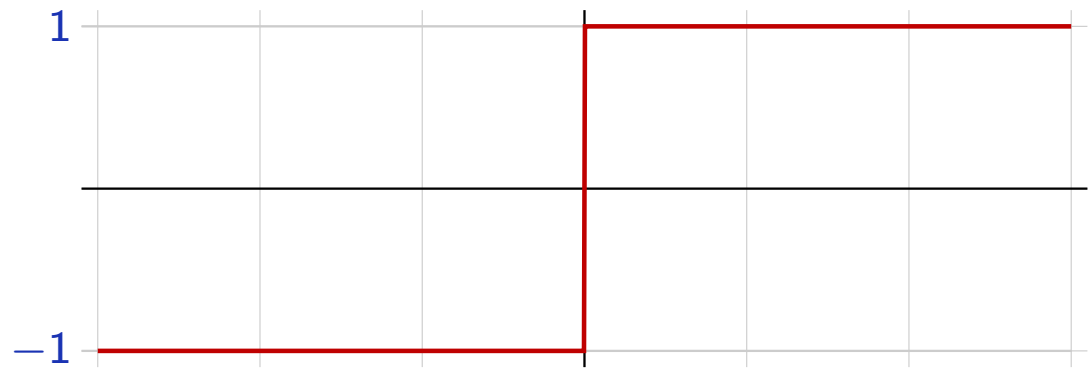
It is a (very) crude biological model. [3]

[3] F. Rosenblatt. The perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

Perceptron

To make things simpler we take responses ± 1. Let

$$\sigma(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

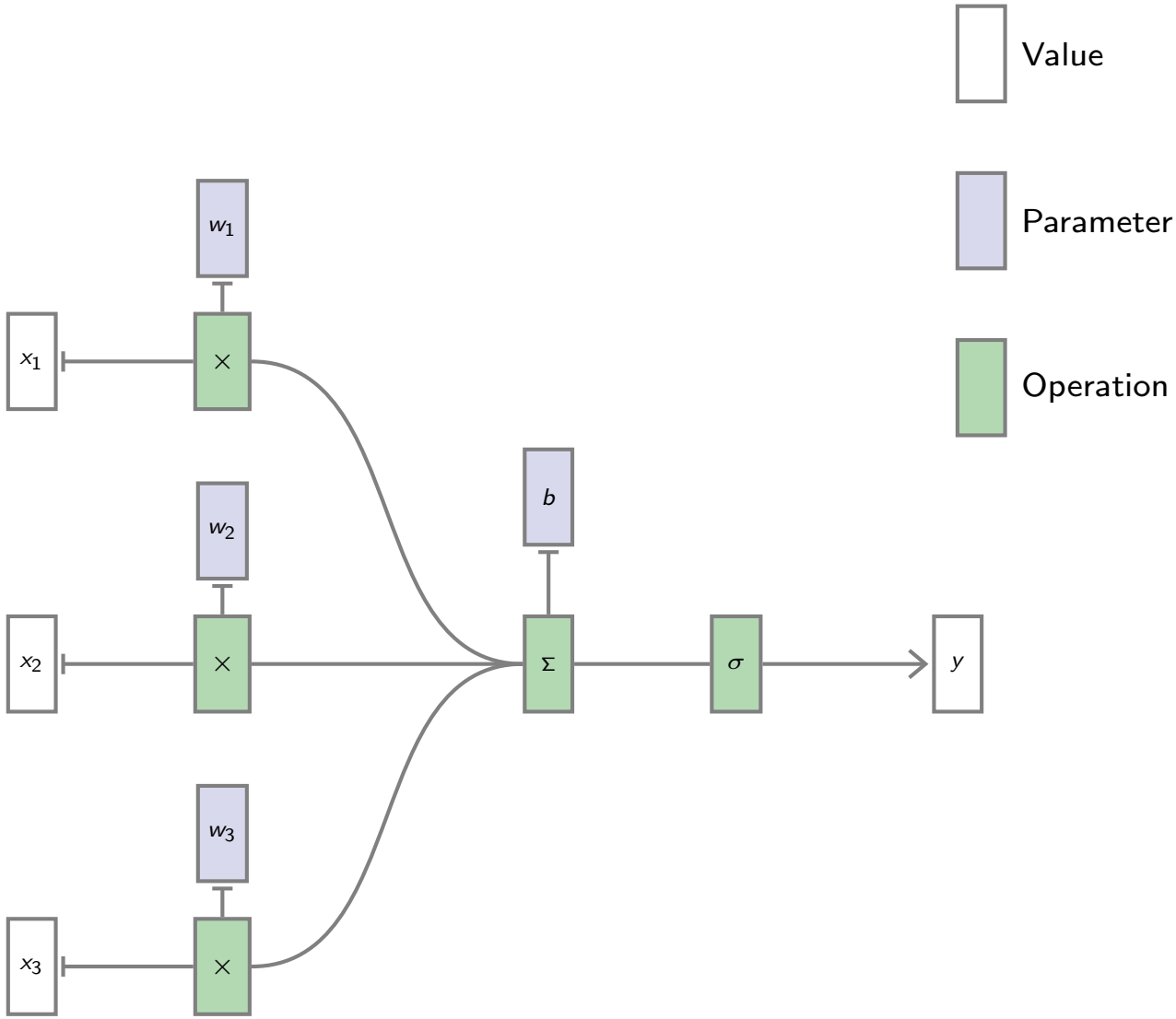


The perceptron classification rule boils down to

$$f(x) = \sigma(w \cdot x + b).$$

Neuron

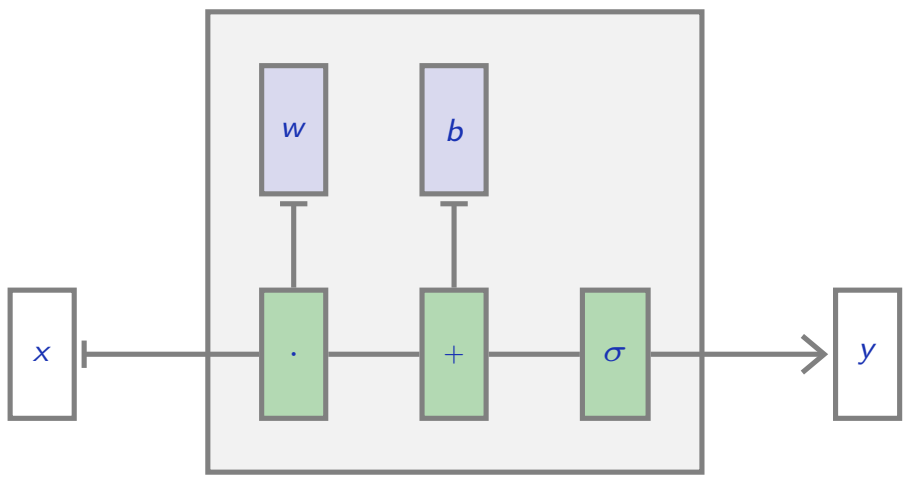
We can represent this "neuron" as follows:



Vectorized Perceptron Model

We can also use tensor operations, as in

$$f(x) = \sigma(w \cdot x + b).$$



Learning Rule

Given a training set

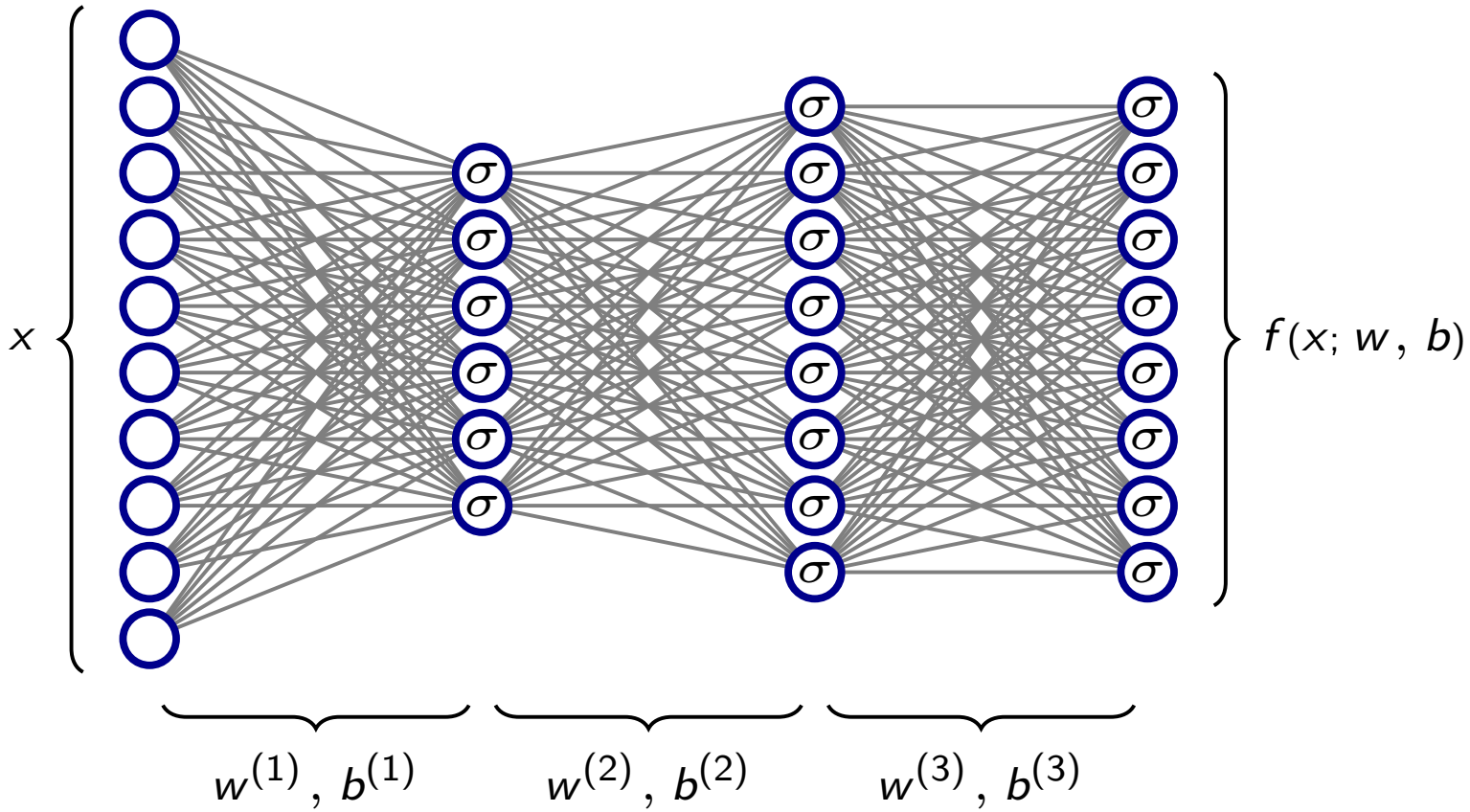
$$(x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}, \quad n = 1, \ldots, N,$$

a very simple scheme to train such a linear operator for classification is the **perceptron algorithm:**

1. Start with $w^0 = 0$,
2. while $\exists n_k$ s.t. $y_{n_k} \left(w^k \cdot x_{n_k} \right) \leq 0$, update $w^{k+1} = w^k + y_{n_k} x_{n_k}$.

The bias b can be introduced as one of the ws by adding a constant component to x equal to 1.

Multilayer Perceptrons



Multiple layers of units

Multilayer Perceptron (MLP)

We can define the MLP formally as,

$$\forall l = 1, 2, \ldots, L, \qquad a^{(l)} = \sigma(w^{(l)} a^{(l-1)} + b^{(l)})$$

where $a^{(0)} = x$, and $f(x; w, b) = a^{(L)} = \hat{y}$

Note, we will define the weighted input term,
$z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)}$, for the backpropagation derivation

Gradient Descent

repeat until convergence:

$$w \leftarrow w - \eta \frac{\partial J}{\partial w} \tag{3}$$

where w is a multidimensional vector representing all of the weights in the model and η is the learning rate.

In order to get Gradient Descent working in practice, we need to compute $\frac{\partial J}{\partial w}$. For neural networks, there are 2 stages to this computation, (1) the forward pass and (2) the backwards pass.

Backpropagation: The Forward Pass

We need to compute the multilayer perceptron outputs y_k in order to compute the cost function $J(w)$.

$$a^{(1)} = f(w^{(1)}x + b^{(1)}) \tag{4}$$

$$\hat{y} = a^{(2)} = f(w^{(2)}a^{(1)} + b^{(2)}) \tag{5}$$

Backpropagation: The Backwards Pass

In order to update the weights, we need to compute the gradient of the cost function with respect to each of the weights. Let us consider the quadratic cost function as follows:

$$J(w) = \frac{1}{2} \sum_{k=1}^{M} (t_k - a_k^{(2)})^2 \tag{6}$$

To compute the weight updates, we compute the derivative of the cost function with respect to each weight. The derivative of J with respect to the weights at the output layer can be computed as follows:

$$\frac{\partial J}{\partial w_{kj}^{(2)}} = \frac{\partial J}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial w_{kj}^{(2)}} \tag{7}$$

Backpropagation: The Backwards Pass

Let us assume the quadratic cost function and the sigmoid activation function.

$$\frac{\partial J}{\partial a_k^{(2)}} = -(t_k - a_k^{(2)}) \tag{8}$$

If we assume a sigmoid activation function then $a_k^{(2)} = \frac{1}{1+e^{-z_k^{(2)}}}$

$$\frac{\partial a_k^{(2)}}{\partial z_k^{(2)}} = a_k^{(2)}(1 - a_k^{(2)}) \tag{9}$$

$$\frac{\partial z_k(2)}{\partial w_{kj}^{(2)}} = a_k^{(1)} \tag{10}$$

Since $z_k^{(2)} = \sum_j w_{kj}^{(2)} a_k^{(1)}$ therefore,

$$\frac{\partial J}{\partial w_{kj}^{(2)}} = -(t_k - a_k^{(2)}) a_k^{(2)} (1 - a_k^{(2)}) a_k^{(1)} \tag{11}$$

Backpropagation: The Backwards Pass

To compute the weight updates with respect to the input layer:

$$\frac{\partial J}{\partial w_{ji}^{(1)}} = \frac{\partial J}{\partial a_k^{(1)}} \frac{\partial a_k^{(1)}}{\partial z_k^{(1)}} \frac{\partial z_k^{(1)}}{\partial w_{ji}^{(1)}} \tag{12}$$

Backpropagation: The Backwards Pass

Again, this is the derivative of the output of the neuron w.r.t. the sigmoid activation function. Since $a_k^{(1)} = \frac{1}{1+e^{-z_k^{(1)}}}$, therefore

$$\frac{\partial a_k^{(1)}}{\partial z_k^{(1)}} = a_k^{(1)}(1 - a_k^{(1)}) \tag{13}$$

Since $z_k^{(1)} = \sum_i x_i w_{ji}^{(1)}$

$$\frac{\partial z_k^{(1)}}{\partial w_{ji}^{(1)}} = x_i \tag{14}$$

Backpropagation: The Backwards Pass

Again, we can work out these three partial derivatives as follows:

$$\frac{\partial J}{\partial a_k^{(1)}} = \sum_{k=1}^{M} \frac{\partial J_{y_k}}{\partial a_k^{(1)}} = (\sum_{k=1}^{M} \frac{\partial J_{y_k}}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial a_k^{(1)}}) \tag{15}$$

where $\frac{\partial z_k^{(2)}}{\partial a_k^{(1)}} = w_{kj}^{(2)}$ since $z_k^{(2)} = \sum_j w_{kj}^{(2)} a_k^{(1)}$, however, this time we take the derivative with respect to $a_k^{(1)}$ instead of $w_{kj}^{(2)}$.

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{16}$$

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{16}$$

- Then

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{17}$$

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{16}$$

- Then

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{17}$$

- For the case of a MSE cost function:

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{16}$$

- Then

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{17}$$

- For the case of a MSE cost function:
- $\delta^L = (a^L - y) \odot \sigma'(z^L)$

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{16}$$

- Then

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{17}$$

- For the case of a MSE cost function:
- $\delta^L = (a^L - y) \odot \sigma'(z^L)$

Backpropagation in General

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{18}$$

Backpropagation in General

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{18}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{19}$$

Backpropagation in General

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{18}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{19}$$

$$\frac{\partial J}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{20}$$

Backpropagation in General

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{18}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{19}$$

$$\frac{\partial J}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{20}$$

$$\frac{\partial J}{\partial b^l_j} = \delta^l_j \tag{21}$$