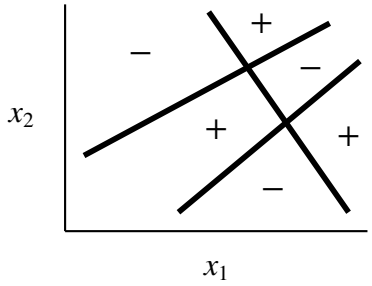
CS 189 Introduction to Machine Learning
Summer 2019 Marc Khoury & Brijen Thananjeyan

# DIS11

## 1 The Multi-layer Perceptron (MLP)

(a) Consider a target function whose + and − regions are illustrated below. Draw the perceptron components, and express $f$ as a Boolean function of its perceptron components.




**Take-away**: a complicated target function, such as the one above, that is composed of perceptrons can be expressed as an `OR` of `AND`s of the component perceptrons.

Let's develop some intuition for why this might be useful.

(b) Over two inputs $x_1$ and $x_2$, how can `OR`, `AND`, and `NOT` each be implemented by a single perceptron? Assume each unit uses a *hard threshold* activation function. Recall the hard-threshold function: for a constant $a$,

$$h(x) = \begin{cases} 1, & \text{if } w^\top x \geq a \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

(c) We have seen that `XOR` is not linearly separable, and hence it cannot be implemented by a single perceptron. Draw a fully connected three unit neural network that has binary inputs $X_1, X_2, \mathbf{1}$ and output $Y$, where $Y$ implements the `XOR` function: $Y = \texttt{XOR}(X_1, X_2)$. Again, assume each unit uses a *hard threshold* activation function.

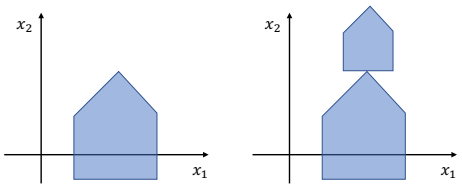
(d) (self-study) Create a neural network with a single hidden layer (of any number of units) that implements the function: (A or not B) XOR (not C or not D).

**Take-away**: If $f$ can be decomposed into perceptrons using an `OR` of `AND`$s$, then it can be implemented by a 3-layer perceptron. A one-hidden-layer MLP is a universal Boolean function. How cool is that?!

## 2 Decision Space

Let's further the intuition about how we can compose arbitrarily complex decision boundaries with a neural network. Consider the images below. For each one, build a network of units with a single output that fires if the input is in the shaded area.




**Take-away:** MLPs can capture any classification boundary. MLPs are universal classifiers. Note that we haven't said anything yet about their ability to generalize.

## 3 What else can a network represent?

So far, we have seen target functions that are exactly decomposable into perceptrons. Even in the case where $f$ is not decomposable into perceptrons, if $f$ is a continuous function then there exists a 3-layer perceptron that approximates $f$ arbitrarily closely. The formal proof for this is beyond the scope of the class. The important point is: how can perceptrons be used to approximate functions with curves? Consider a disc target function (i.e., a circle with positive on the inside and negative on the outside), and illustrate how perceptrons could approximate this $f$.

**Take-away:** Neural networks are universal function approximators. Wow!! Refer to this for an intuitive explanation and visual proof. It's important to note that neural networks are not the only universal function approximators. **Polynomials** were the first provable universal approximators, see the Stone–Weierstrass theorem. In addition, the **Fourier** basis is a popular function approximator in, e.g., signal processing. This Quora post has more details, and a cute picture of the three bases (polynoial, Fourier, and neural network) fitting functions.

We still haven't said anything about the ability to generalize. There is a classic line from John von Neumann on overfitting: With four parameters I can fit an elephant, and with five I can make him wiggle his trunk. Jürgen Mayer took this statement literally in a 2009 publication, *Drawing an elephant with four complex parameters*. You can see the elephant wiggling the trunk here.

Okay, now that neural nets sound amazingly expressive, how can we use them?! A neural network is parameterized by the weights and biases of its units. We use gradient descent to update the network parameters, which means the updates are based on the gradients of the parameters. Backpropagation is a way of computing gradients of expressions in a network through recursive application of the chain rule. Your understanding and intuition of backpropagation will guide you as you design, develop, and debug your neural networks.

## 4 Simple gradients and their interpretations

For each of the functions below, what are the partial derivatives with respect to each variable? For the first three functions only, what is the interpretation? For the first function only, what is $\nabla f$?

(a) $f(x, y) = xy$

(b) $f(x, y) = x + y$

(c) $f(x, y) = \max(x, y)$

(d) (self-study) $\sigma(x) = \frac{1}{1+e^{-x}}$

(e) (self-study) $\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

# 5 Backrop in practice: staged computation

For the function $f(x, y, z) = (x + y)z$:

(a) Decompose $f$ into two simpler functions.

(b) Draw the network that represents the computation of $f$.

(c) Write the forward pass and backward pass (backpropagation) in the network.

(d) Update your network drawing with the intermediate values in the forward and backward pass. Use the following inputs $x = -2$, $y = 5$, and $z = -4$.

# 6 More backrop in practice: staged computation (self-study)

For the function $f(w, x) = \frac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$:

(a) Decompose $f$ into two simpler functions.

(b) Draw the network that represents the computation of $f$.

(c) Write the forward pass and backward pass (backpropagation) in the network.

(d) Update your network drawing with the intermediate values in the forward and backward pass. Use the following weight initialization and inputs $w = [2, -3, -3]$ and $x = [-1, -2]$.

(e) Now consider the function $f(x, y) = \frac{x + \sigma(y)}{\sigma(x) + (x+y)^2}$. What are the forward and backward passes?

# 7 Backpropagation Practice (self-study)

(a) Chain rule of multiple variables: Assume that you have a function given by $f(x_1, x_2, \ldots, x_n)$, and that $g_i(w) = x_i$ for a scalar variable $w$. How would you compute $\frac{\mathrm{d}}{\mathrm{d}w} f(g_1(w), g_2(w), \ldots, g_n(w))$? What is its computation graph?

(b) Let $w_1, w_2, \ldots, w_n \in \mathbb{R}^d$, and we refer to these variables together as $W \in \mathbb{R}^{n \times d}$. We also have $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$. Consider the function

$$f(W, x, y) = \left( y - \sum_{i=1}^{n} \phi(w_i^\top x + b_i) \right)^2 .$$

Write out the function computation graph (also sometimes referred to as a pictorial representation of the network). This is a directed graph of decomposed function computations, with the function at one end, and the variables $W, b, x, y$ at the other end, where $b = [b_1, \cdots, b_n]$.

(c) Suppose $\phi(x) = \sigma(x)$ (from problem 1a). Compute the partial derivatives $\frac{\partial f}{\partial w_i}$ and $\frac{\partial f}{\partial b_i}$. Use the computational graph you drew in the previous part to guide you.

(d) Write down a single gradient descent update for $w_i^{(t+1)}$ and $b_i^{(t+1)}$, assuming step size $\eta$. You answer should be in terms of $w_i^{(t)}$, $b_i^{(t)}$, $x$, and $y$.

(e) (optional) Define the cost function

$$\ell(x) = \frac{1}{2}\|W^{(2)}\Phi\left(W^{(1)}x + b\right) - y\|_2^2, \tag{2}$$

where $W^{(1)} \in \mathbb{R}^{d\times d}$, $W^{(2)} \in \mathbb{R}^{d\times d}$, and $\Phi : \mathbb{R}^d \to \mathbb{R}^d$ is some nonlinear transformation. Compute the partial derivatives $\frac{\partial \ell}{\partial x}, \frac{\partial \ell}{\partial W^{(1)}}, \frac{\partial \ell}{\partial W^{(2)}}$, and $\frac{\partial \ell}{\partial b}$.

(f) (optional) Suppose $\Phi$ is the identity map. Write down a single gradient descent update for $W_{t+1}^{(1)}$ and $W_{t+1}^{(2)}$ assuming step size $\eta$. Your answer should be in terms of $W_t^{(1)}, W_t^{(2)}, b_t$ and $x, y$.

# 8 Model Intuition (self-study)

(a) What can go wrong if you just initialize all the weights in a neural network to exactly zero? What about to the same value?

(b) Adding nodes in the hidden layer gives the neural network more approximation ability, because you are adding more parameters. How many weight parameters are there in a neural network with architecture specified by $d = \left[d^{(0)}, d^{(1)}, ..., d^{(N)}\right]$, a vector giving the number of nodes in each of the $N$ layers? Evaluate your formula for a 2 hidden layer network with 10 nodes in each hidden layer, an input of size 8, and an output of size 3.

(c) Consider the two networks in the image below, where the added layer in going from Network A to Network B has 10 units with linear activation. Give one advantage of Network A over Network B, and one advantage of Network B over Network A.