Deep Learning: A first look

Exebit 2018, IITM

Sachin Sridhar, Department of CSE

April 15, 2018

Indian Institute of Technology Madras

Table of contents

Machine Learning: Some basics

What is Machine Learning?

- *"The science of getting computers to act without being explicitly programmed."* - Andrew Ng

What is Machine Learning?

- *"The science of getting computers to act without being explicitly programmed."* - Andrew Ng
- Uses statistical procedures to get computer systems to be able to "learn" and "predict" an output given an input

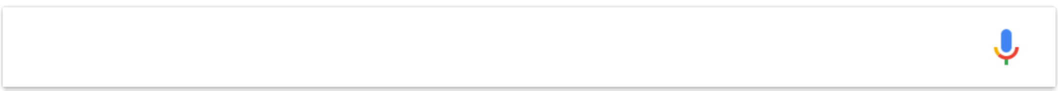
What is Machine Learning?

- *"The science of getting computers to act without being explicitly programmed."* - Andrew Ng
- Uses statistical procedures to get computer systems to be able to "learn" and "predict" an output given an input
- Machine learning algorithms are different from conventional algorithms; they are statistical in nature and often have no correctness guarantees

What is Machine Learning?

- *"The science of getting computers to act without being explicitly programmed."* - Andrew Ng
- Uses statistical procedures to get computer systems to be able to "learn" and "predict" an output given an input
- Machine learning algorithms are different from conventional algorithms; they are statistical in nature and often have no correctness guarantees
- Seen in daily life!

ML in daily life



How does it work?

- Predicting an output given an input is mathematically formulated as a probability query $P(Y = y|X = x)$.

How does it work?

- Predicting an output given an input is mathematically formulated as a probability query $P(Y = y|X = x)$.
- We need a measure of "how good" our estimate is. So we try to minimize a *loss function* (mean squared error, negative log-likelihood, etc)

How does it work?

- Predicting an output given an input is mathematically formulated as a probability query $P(Y = y|X = x)$.
- We need a measure of "how good" our estimate is. So we try to minimize a *loss function* (mean squared error, negative log-likelihood, etc)
- This can be optimized using techniques from calculus, convex analysis, etc. A popular example of such a technique is gradient descent.

Example 1: Coin bias

Consider the following sequence of coin tosses:

H T T H H H T T T T

Example 1: Coin bias

Consider the following sequence of coin tosses:

H T T H H H T T T T

What could the bias of the coin be?

Example 1: Coin bias

Consider the following sequence of coin tosses:

H T T H H H T T T T

What could the bias of the coin be? (*We can never say for sure*)

Example 1: Coin bias

Consider the following sequence of coin tosses:

H T T H H H T T T T

What could the bias of the coin be? (*We can never say for sure*)

What is a good guess for the bias of the coin?

Example 1: Coin bias

Consider the following sequence of coin tosses:

H T T H H H T T T T

What could the bias of the coin be? (*We can never say for sure*)

What is a good guess for the bias of the coin?

Intuitively we would say $\frac{\text{number of heads}}{\text{total number of coin tosses}} = \frac{4}{10}$

Example 1: Coin bias

Consider the following sequence of coin tosses:

H T T H H H T T T T

What could the bias of the coin be? (*We can never say for sure*)

What is a good guess for the bias of the coin?

Intuitively we would say $\frac{\text{number of heads}}{\text{total number of coin tosses}} = \frac{4}{10}$

We can use this for predicting further coin tosses!

Coin bias: The math

Let D be the given data, and θ be a random variable representing the bias of the coin.

Definition

The *likelihood function* $L(\theta)$ is given by $L(\theta) = P(D|\theta)$.

Here we have $L(\theta) = \binom{10}{4}\theta^4(1-\theta)^6$ (assuming i.i.d coin tosses).

Setting the derivative wrt θ to zero to maximize the likelihood, we get:

$$\frac{dL(\theta)}{d\theta} = \binom{10}{4}(\theta^4 \cdot 6(1-\theta)^5 + 4\theta^3 \cdot (1-\theta)^6) = 0$$

$$\implies \boxed{\theta = 0.4}$$

In some sense, this is the best estimate we can get!

Example 2: Firm profits

Suppose we want to predict the profit of a firm given some simple data:

Units produced (x)	Cost of production (y)
10	592
20	1090
30	1604
40	2122
50	2620
70	?

We can assume that there is some fixed cost for setting up production (a), plus a marginal cost (the cost per unit) required to produce the items (b):

$y = f(x) = a + bx$ (This is a linear model)

Example 2: Firm profits

We require a good estimate for a and b.

Example 2: Firm profits

We require a good estimate for a and b.

Definition

Consider a dataset with n input and output pairs, denoted (x_i, y_i). The *mean squared error* (MSE) is defined as $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$.

Example 2: Firm profits

We require a good estimate for a and b.

Definition

Consider a dataset with n input and output pairs, denoted (x_i, y_i). The *mean squared error* (MSE) is defined as $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$.

In this case, the (x_i, y_i) pairs are given by $\{(10, 592), (20, 1090), (30, 1604), (40, 2122), (50, 2620)\}$. We look for values of a and b that minimize the mean squared error $L = \frac{1}{5}\sum_{i=1}^{5}((a + bx_i) - y_i)^2$.

Example 2: Firm profits

We require a good estimate for a and b.

Definition

Consider a dataset with n input and output pairs, denoted (x_i, y_i). The *mean squared error* (MSE) is defined as $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$.

In this case, the (x_i, y_i) pairs are given by $\{(10, 592), (20, 1090), (30, 1604), (40, 2122), (50, 2620)\}$. We look for values of a and b that minimize the mean squared error $L = \frac{1}{5}\sum_{i=1}^{5}((a + bx_i) - y_i)^2$.

We can do this by setting $\frac{\partial L}{\partial a} = 0$, $\frac{\partial L}{\partial b} = 0$ to get $\boxed{a = 79.2, b = 50.8}$.

Firm profits: The math

$$L = \frac{1}{5}\sum_{i=1}^{5}(a + bx_i - y_i)^2$$

$$\frac{\partial L}{\partial a} = \cancel{\frac{1}{5}}\sum_{i=1}^{5}\cancel{2}x_i(a + bx_i - y_i) = 0 \qquad (1)$$

$$\frac{\partial L}{\partial b} = \cancel{\frac{1}{5}}\sum_{i=1}^{5}\cancel{2}(a + bx_i - y_i) = 0 \qquad (2)$$

Solving, we get

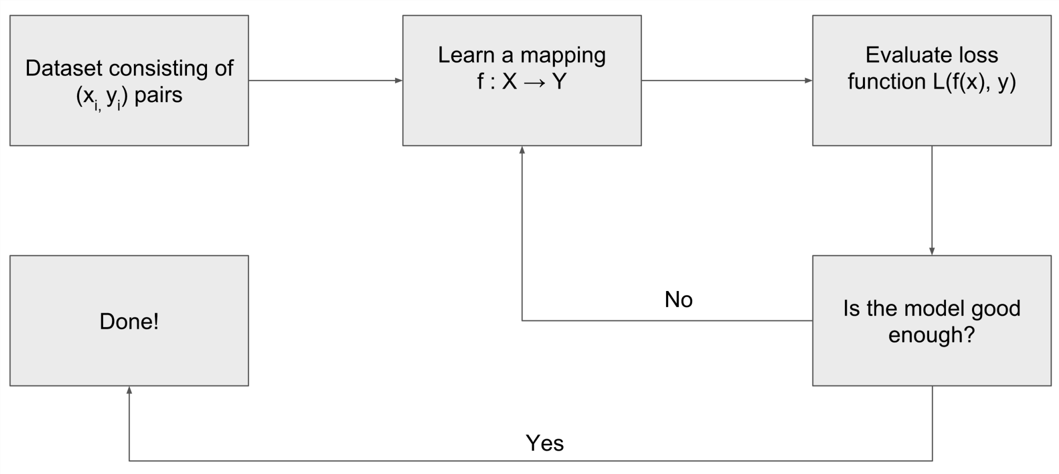
$$\boxed{b = \frac{\sum_{i=1}^{5}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{5}(x_i - \overline{x})^2}, \qquad a = \overline{y} - b\overline{x}}$$

Where $\overline{x} = \frac{1}{5}\sum_{i=1}^{5} x_i$ and $\overline{y} = \frac{1}{5}\sum_{i=1}^{5} y_i$

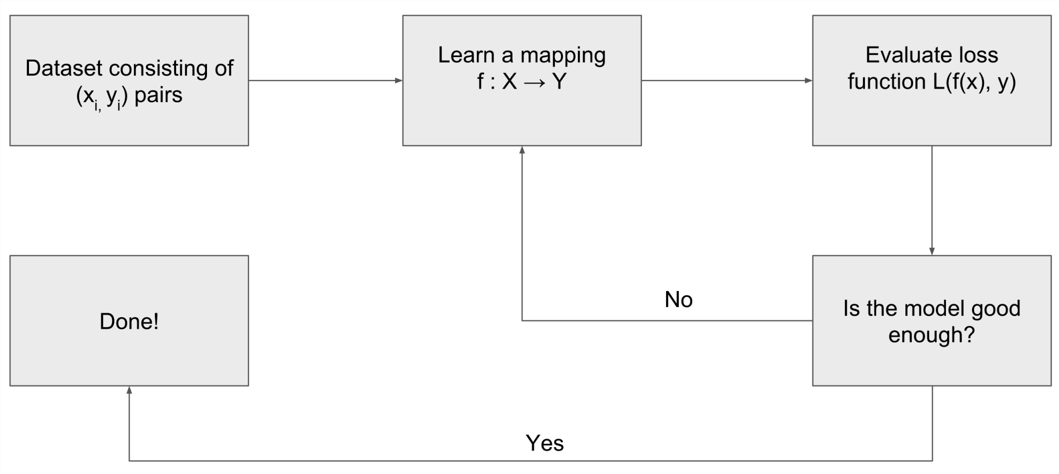
Machine Learning: A general framework

- Tries to predict an output y given an input x by learning a mapping $f : X \rightarrow Y$
- Quality of the estimate defined by a loss function $L(f(x), y)$
- ML algorithms try to minimize the loss function (closed form, gradient descent, etc)

Machine Learning: A general framework



Machine Learning: A general framework



Is it really that simple?

Machine Learning: Some issues

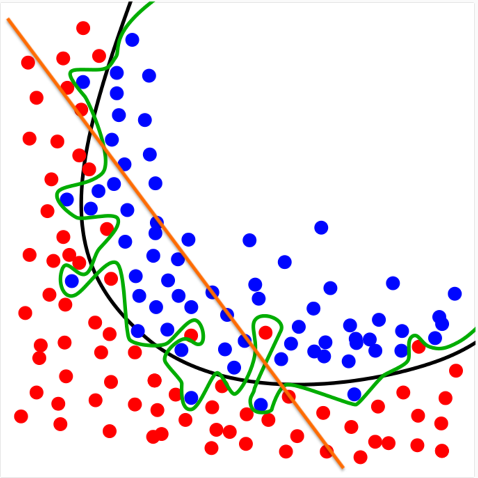
Overfitting: The model (function) fits too strongly to the data.

Underfitting: The model does not fit well enough to the data.

Machine Learning: Some issues

Overfitting: The model (function) fits too strongly to the data.

Underfitting: The model does not fit well enough to the data.



Deep Learning

What is Deep Learning?

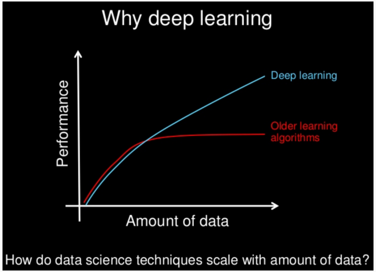
- Subfield of Machine Learning inspired by a crude model of the brain, called *artificial neural networks*

What is Deep Learning?

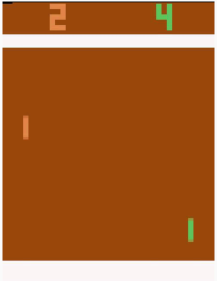
- Subfield of Machine Learning inspired by a crude model of the brain, called *artificial neural networks*
- Typically works with large amounts of data and outperforms other machine learning algorithms

What is Deep Learning?

- Subfield of Machine Learning inspired by a crude model of the brain, called *artificial neural networks*
- Typically works with large amounts of data and outperforms other machine learning algorithms
- Can be used to learn powerful representations and highly non linear functions



What can Deep Learning do?



How did it all start?

- In the 1940s, Frank Rosenblatt developed an artificial neuron model that was capable of learning (the "perceptron")

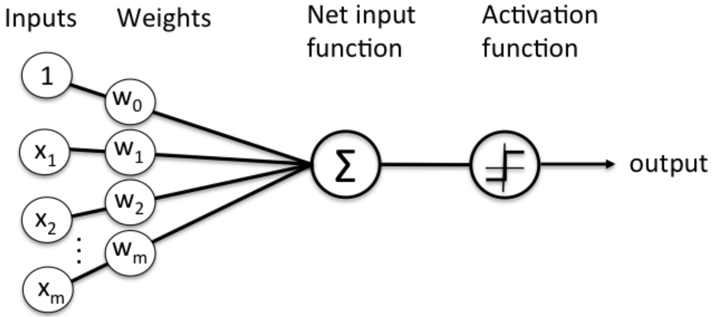
How did it all start?

- In the 1940s, Frank Rosenblatt developed an artificial neuron model that was capable of learning (the “perceptron”)
- Worked well on small problems that were *linearly separable* (what’s that?)

How did it all start?

- In the 1940s, Frank Rosenblatt developed an artificial neuron model that was capable of learning (the "perceptron")
- Worked well on small problems that were *linearly separable* (what's that?)
- Was eventually superseded by support vector machines (SVMs) which were more stable and performed better on real-world data

Perceptrons



Schematic of Rosenblatt's perceptron.

Perceptrons: The math

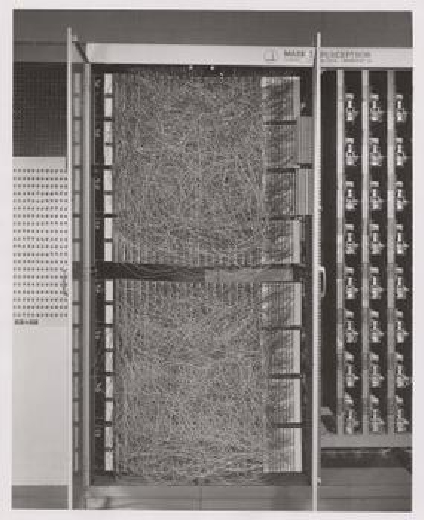
Mathematically, if an input $x = (x_1, ..., x_m)$ is given to the perceptron, the output y is calculated by

$$\hat{y} = f(w_0 + w_1 x_1 + ... + w_m x_m)$$

Where $f(\cdot)$ is the *activation function*, which is a non-linear function of its input. The weights w_i are parameters of the perceptron, and they are *learned* by the perceptron.

Some commonly used activation functions are step, sigmoid, ReLU, tanh, etc.

Perceptrons: In real life



Perceptrons: What do they do?

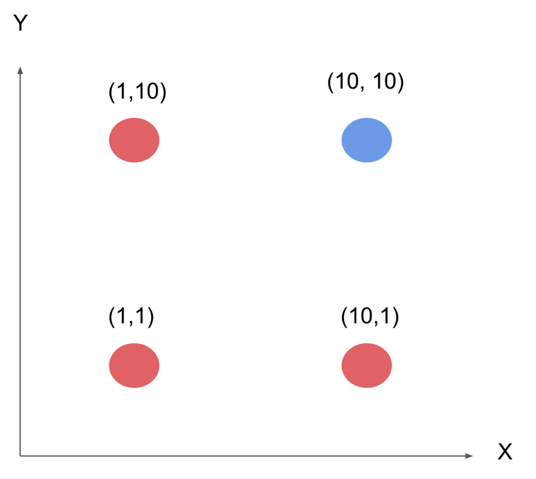
Consider a perceptron that uses a step function for its activation:

$$\hat{y} = f(w_0 + w_1 x_1 + ... + w_m x_m)$$
$$= \begin{cases} 1, & w_0 + w_1 x_1 + ... + w_m x_m > 0 \\ 0, & \text{otherwise} \end{cases}$$

What about the boundary region defined by $w_0 + w_1 x_1 + ... + w_m x_m = 0$?

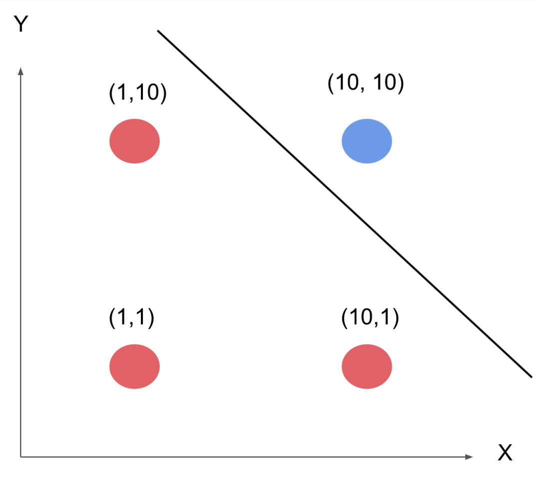
It defines a *plane* in (m-1) dimensions.

Perceptrons: What do they do?



Perceptrons find a *separating plane* to try and classify the data.

Perceptrons: What do they do?



Perceptrons find a *separating plane* to try and classify the data.

Perceptrons: How do they learn?

Consider a dataset $\{(x^1, y^1), (x^2, y^2), ..., (x^n, y^n)\}$. The perceptron learning algorithm is as follows:

Perceptrons: How do they learn?

Consider a dataset $\{(x^1, y^1), (x^2, y^2), ..., (x^n, y^n)\}$. The perceptron learning algorithm is as follows:

1. Initialize all weights w_i to 0

Perceptrons: How do they learn?

Consider a dataset $\{(x^1, y^1), (x^2, y^2), ..., (x^n, y^n)\}$. The perceptron learning algorithm is as follows:

1. Initialize all weights w_i to 0
2. For each training example (x^j, y^j), calculate the output

   $\hat{y}^j = f(w_0 + w_1 x_1^j + +w_2 x_2^j + ..., +w_m x_m^j)$

Perceptrons: How do they learn?

Consider a dataset $\{(x^1, y^1), (x^2, y^2), ..., (x^n, y^n)\}$. The perceptron learning algorithm is as follows:

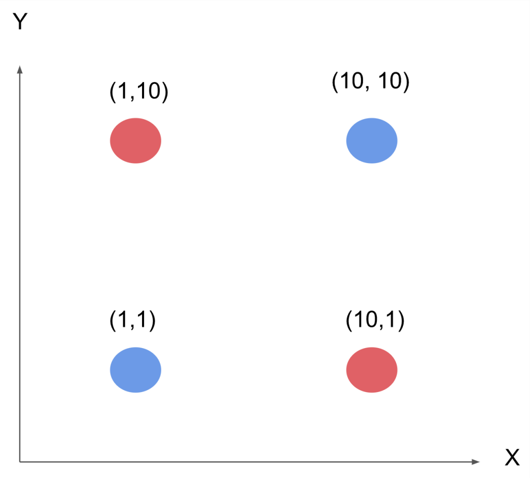
1. Initialize all weights w_i to 0
2. For each training example (x^j, y^j), calculate the output

   $$\hat{y}^j = f(w_0 + w_1 x_1^j + +w_2 x_2^j + ..., +w_m x_m^j)$$

3. Each of the weights w_i are updated as

   $$w_i \rightarrow w_i + (y^j - \hat{y}^j) x_i^j$$

Perceptrons: Can they learn everything?



Can a linear classifier correctly classify all 4 points?

Feed forward networks

What comes next?

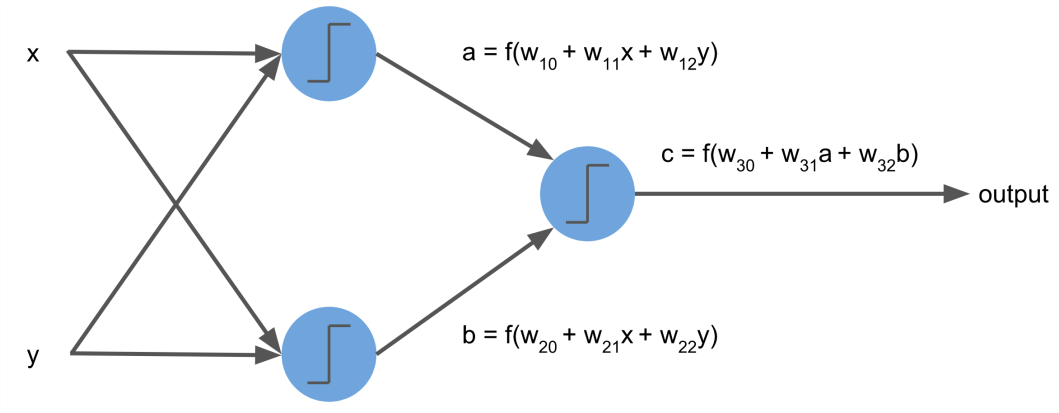
The perceptron is unable to learn non linear functions like XOR. Is there a way to modify it so that it can learn these functions?

What comes next?

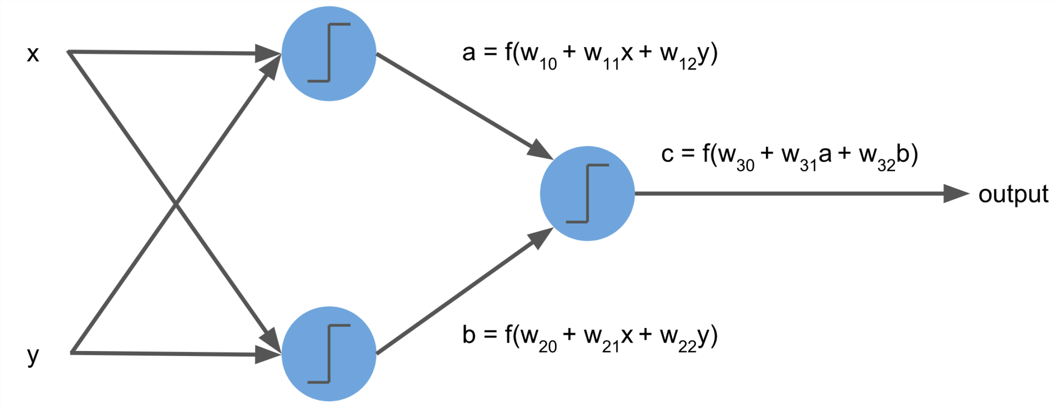
The perceptron is unable to learn non linear functions like XOR. Is there a way to modify it so that it can learn these functions?

The idea is to chain together multiple perceptrons to learn more complex functions. This is done by feeding the output of a perceptron as the input of another perceptron.

Learning the XOR function

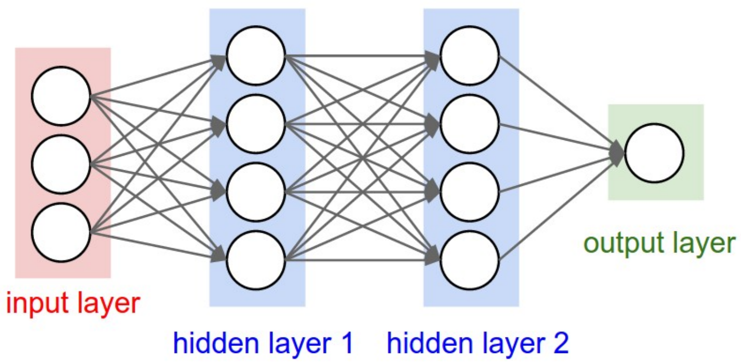


Learning the XOR function



The parameters to be learned are $w_{10}, w_{11}, w_{12}, w_{20}, w_{21}, w_{22}, w_{30}, w_{31}, w_{32}$. This successfully manages to learn the XOR function!

This idea can be extended to many layers and many perceptron units (neurons) in each layer.



This is called a feed forward neural network.

General feed forward networks

Perceptrons can represent linear functions, and a few perceptrons chained together can represent non linear functions like XOR. What could a general feed forward neural network represent?

General feed forward networks

Perceptrons can represent linear functions, and a few perceptrons chained together can represent non linear functions like XOR. What could a general feed forward neural network represent?

Theorem (Universal Approximation)

Any continuous function can be approximated to arbitrary accuracy by a feed forward neural network with the appropriate architecture.

This is great, we can learn anything we need!

General feed forward networks

Perceptrons can represent linear functions, and a few perceptrons chained together can represent non linear functions like XOR. What could a general feed forward neural network represent?

Theorem (Universal Approximation)

Any continuous function can be approximated to arbitrary accuracy by a feed forward neural network with the appropriate architecture.

This is great, we can learn anything we need! (right?)

Learning in feed forward networks

- (Stochastic) gradient descent is used to optimize the loss function for learning.

Learning in feed forward networks

- (Stochastic) gradient descent is used to optimize the loss function for learning.
- The gradient is calculated using the *backpropagation algorithm*, a dynamic programming algorithm that keeps track of various derivatives that are required to construct the final gradient.

Learning in feed forward networks

- (Stochastic) gradient descent is used to optimize the loss function for learning.
- The gradient is calculated using the *backpropagation algorithm*, a dynamic programming algorithm that keeps track of various derivatives that are required to construct the final gradient.
- Need not write code for this, already done for you by software

The backpropagation algorithm (*)

- L: The total number of layers in the network (0 indexed)
- n_l: The number of neurons in the layer l
- x_i^l: The input to the i^{th} neuron in the l^{th} layer
- o_i^l: The output of the i^{th} neuron in the l^{th} layer
- W_{ij}^l: The weight connecting neuron i of layer $l-1$ to neuron j of layer l
- b_i^l: The bias of the neuron i in layer l
- f: The activation function used at each hidden neuron (including the output layer)
- E: The error (loss) function. Here we assume that the squared error is used.

The backpropagation algorithm (*)

Since the input x_j^l only affects neurons after it and x_j^l can be written as a function of W_{ij}^l and other variables, we can write:

$$\frac{\partial E}{\partial W_{ij}^l} = \frac{\partial E}{\partial x_j^l} \frac{\partial x_j^l}{\partial W_{ij}^l}$$

$$\frac{\partial E}{\partial b_j^l} = \frac{\partial E}{\partial x_j^l} \frac{\partial b_j^l}{\partial b_j^l}$$

Defining $\delta_j^l \equiv \frac{\partial E}{\partial x_j^l}$, we have

$$\frac{\partial E}{\partial W_{ij}^l} = \delta_j^l \frac{\partial x_j^l}{\partial W_{ij}^l} \tag{1}$$

$$\frac{\partial E}{b_j^l} = \delta_j^l \frac{\partial x_j^l}{b_j^l} \tag{2}$$

The backpropagation algorithm (*)

By the construction of the network,

$$x_j^l = \left[\sum_{i=1}^{n_{l-1}} W_{ij}^l o_i^{l-1} \right] + b_j^l$$

And so

$$\frac{\partial x_j^l}{\partial W_{ij}^l} = o_i^{l-1} \tag{3}$$

$$\frac{\partial x_j^l}{\partial b_j^l} = 1 \tag{4}$$

The backpropagation algorithm (*)

Now, if l is an output layer we can write

$$\delta_j^l = \frac{\partial E}{\partial x_j^l} = \frac{\partial E}{\partial o_j^l}\frac{\partial o_j^l}{\partial x_j^l}$$

Since we are using the squared loss, $E = \frac{1}{2}(\sum_{j=1}^{n_L}(t - o_j^L)^2)$ where t is the target output. This gives an easy way to calculate the first term. For the second term, note that the derivative of the output of a neuron with respect to the input is just the derivative of the activation function. Putting these two together, we get

$$\delta_j^l = (t - o_j^L)f'(x_j^l), \quad \text{If } l \text{ is an output layer} \tag{5}$$

The backpropagation algorithm (*)

If l is not an output layer, we can determine δ_j^l in terms of the δ_k^{l+1} values. Since changing the input to a neuron affects all inputs of the next layer of neurons, we can write

$$\delta_j^l = \frac{\partial E}{\partial x_j^l} = \sum_{k=0}^{n_{l+1}} \frac{\partial E}{\partial x_k^{l+1}} \frac{\partial x_k^{l+1}}{\partial o_j^l} \frac{\partial o_j^l}{\partial x_j^l}$$

Finally we have $x_k^{l+1} = \sum_{j=1}^{n_l} W_{jk}^{l+1} o_j^l$, so $\frac{\partial x_k^{l+1}}{\partial o_j^l} = W_{jk}^{l+1}$, and

$$\delta_j^l = \sum_{k=0}^{n_{l+1}} \delta_k^{l+1} W_{jk}^{l+1} f'(x_j^l), \quad \text{If } l \text{ is not an output layer} \tag{6}$$

The backpropagation algorithm (*)

Putting together (1), (2), (3), (4), (5), (6), we get

$$\frac{\partial E}{\partial W_{ij}^l} = \delta_j^l o_i^{l-1}$$

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l$$

$$\delta_j^l = (t - o_j^L) f'(x_j^l), \quad \text{If } l \text{ is an output layer}$$

$$\delta_j^l = \sum_{k=0}^{n_{l+1}} \delta_k^{l+1} W_{jk}^{l+1} f'(x_j^l), \quad \text{If } l \text{ is not an output layer}$$

The backpropagation algorithm (*)

This can be written in matrix/vector form as

Backpropagation equations

$$\frac{\partial E}{\partial W^l} = \boldsymbol{\delta}^l (\boldsymbol{o}^{l-1})^T$$

$$\frac{\partial E}{\partial \boldsymbol{b}^l} = \boldsymbol{\delta}^l$$

$$\boldsymbol{\delta}^l = (\boldsymbol{t} - \boldsymbol{o}^L) \circ f'(\boldsymbol{x}^l), \quad \text{If } l \text{ is an output layer}$$

$$\boldsymbol{\delta}^l = (W^{l+1} \boldsymbol{\delta}^{l+1}) \circ f'(\boldsymbol{x}^l), \quad \text{If } l \text{ is not an output layer}$$

Update equation

$$W \leftarrow W - \eta \frac{\partial E}{\partial W}$$

The backpropagation algorithm (*)

This can be written in matrix/vector form as

Backpropagation equations

$$\frac{\partial E}{\partial W^l} = \boldsymbol{\delta}^l(\boldsymbol{o}^{l-1})^T$$
$$\frac{\partial E}{\partial \boldsymbol{b}^l} = \boldsymbol{\delta}^l$$
$$\boldsymbol{\delta}^l = (\boldsymbol{t} - \boldsymbol{o}^L) \circ f'(\boldsymbol{x}^l), \quad \text{If } l \text{ is an output layer}$$
$$\boldsymbol{\delta}^l = (W^{l+1}\boldsymbol{\delta}^{l+1}) \circ f'(\boldsymbol{x}^l), \quad \text{If } l \text{ is not an output layer}$$

Update equation

$$W \leftarrow W - \eta\frac{\partial E}{\partial W}$$

The backpropagation algorithm (*)

Update equation

$$W \leftarrow W - \eta \frac{\partial E}{\partial W}$$

Where η is a parameter called the learning rate.

The backpropagation algorithm (*)

Update equation

$$W \leftarrow W - \eta \frac{\partial E}{\partial W}$$

Where η is a parameter called the learning rate.

This is way too complicated! There has to be an easier way!

The backpropagation algorithm (*)

Update equation

$$W \leftarrow W - \eta \frac{\partial E}{\partial W}$$

Where η is a parameter called the learning rate.

This is way too complicated! There has to be an easier way!

Enter Tensorflow.

Tensorflow

What is Tensorflow?

- An open source framework for ML/DL developed by Google
- Does all the heavy math for you, no need to code gradient descent. Specifying the model architecture is good enough
- Uses a computational graph internally to accomplish all of this. The graph can be manipulated too
- Has a Python library available (that we are going to use)

Placeholders

A placeholder in Tensorflow is an object that allows use to define computations on it without giving it a value. It is initialized as and when required, for example when giving it as input to a function.

Placeholders

A placeholder in Tensorflow is an object that allows use to define computations on it without giving it a value. It is initialized as and when required, for example when giving it as input to a function.

```
import tensorflow as tf
x = tf.placeholder(tf.float32, (None, 1))
```

This declares a placeholder variable x that has size (unknown $\times 1$).

Variables

A Variable in Tensorflow is an object that holds a value, and is typically something that we are optimizing over (like the weights of a neural network). They must be initialized in code.

A Variable in Tensorflow is an object that holds a value, and is typically something that we are optimizing over (like the weights of a neural network). They must be initialized in code.

```
import tensorflow as tf
m = tf.Variable(tf.zeros, ([1]))
```

This declares a Variable *m* that has size 1 (think of it like an array of size 1).

A Variable in Tensorflow is an object that holds a value, and is typically something that we are optimizing over (like the weights of a neural network). They must be initialized in code.

```
import tensorflow as tf
m = tf.Variable(tf.zeros, ([1]))
```

This declares a Variable *m* that has size 1 (think of it like an array of size 1).

In short, placeholders are input data that does not change as the model trains, and Variables are model parameters that change with time.

Sessions

A Session in Tensorflow is where code actually gets executed. Outside of a session, only variables and the computational graph is defined. Code execution requires that a Tensorflow Session be declared.

Sessions

A Session in Tensorflow is where code actually gets executed. Outside of a session, only variables and the computational graph is defined. Code execution requires that a Tensorflow Session be declared.

```
import tensorflow as tf
sess = tf.Session() # Open a session
sess.run(something) # Do stuff
sess.close() # Close the session
```

Or,

```
import tensorflow as tf
with tf.Session() as sess: # Open a session
    sess.run(something) # Do stuff
```

Firm costs revisited

Units produced (x)	Cost of production (y)
10	592
20	1090
30	1604
40	2122
50	2620

How would we find a good linear model for this problem using Tensorflow?

Firm costs revisited

(Download this at goo.gl/5yayn5)

```
import numpy as np
import tensorflow as tf

# Model linear regression y_hat = a + bx

# Placeholder variables for the input data
x = tf.placeholder(tf.float32, [None, 1])
y = tf.placeholder(tf.float32, [None, 1])

# Model parameters are Variables in Tensorflow
a = tf.Variable(tf.zeros([1]))
b = tf.Variable(tf.zeros([1, 1]))

# Calculating the prediction of the linear model
y_hat = a + tf.matmul(x,b)

# Loss function =  sum((y_hat-y)^2)
cost = tf.reduce_mean(tf.square(y_hat - y))

# Training using Gradient Descent to minimize cost
train_step = tf.train.GradientDescentOptimizer(0.000001).minimize(cost)
```

Firm costs revisited

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
steps = 20000
for i in range(steps):
    # Input data for the model
    xs = np.array([[10], [20], [30], [40], [50]])
    ys = np.array([[592], [1090], [1604], [2122], [2620]])

    # Defining a feed dictionary; this is the set of
    # (placeholder : value) pairs for calculations
    feed = { x : xs, y : ys }

    # Running a single training step
    sess.run(train_step, feed_dict = feed)

# Printing the results
print("a: %f" % sess.run(b))
print("b: %f" % sess.run(a))
```

Computational graph

```python
import numpy as np
import tensorflow as tf

# Model linear regression y_hat = a + bx

# Placeholder variables for the input data
x = tf.placeholder(tf.float32, [None, 1])
y = tf.placeholder(tf.float32, [None, 1])

# Model parameters are Variables in Tensorflow
a = tf.Variable(tf.zeros([1]))
b = tf.Variable(tf.zeros([1, 1]))

# Calculating the prediction of the linear model
y_hat = a + tf.matmul(x,b)

# Loss function =  sum((y_hat-y)^2)
cost = tf.reduce_mean(tf.square(y_hat - y))

# Training using Gradient Descent to minimize cost
train_step = tf.train.GradientDescentOptimizer(0.000001).minimize(cost)
```

A few important things about Tensorflow

- *All* computations happen within a session. This includes printing the value of a variable!

A few important things about Tensorflow

- *All* computations happen within a session. This includes printing the value of a variable!
- To build the computational graph, all arithmetic operations for the model must be done by Tensorflow. For example, tf.reduce_sum must be used instead of Python's builtin sum() function.

A few important things about Tensorflow

- *All* computations happen within a session. This includes printing the value of a variable!

- To build the computational graph, all arithmetic operations for the model must be done by Tensorflow. For example, tf.reduce_sum must be used instead of Python's builtin sum() function.

- When declaring a placeholder, if a dimension is "None", it simply means that it is unknown.

Thank you!