# Logistic Regression and Neural Networks

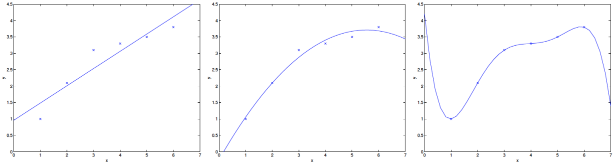
José Eduardo Ochoa Luna
Dr. Ciencias - Universidade de São Paulo

Maestría C.C. Universidad Católica San Pablo
Sistemas Inteligentes

18 de octubre 2018

## Overfitting

Consider the problem of predicting $y$ from $x \in \mathbb{R}$



- 1) $y = \theta_0 + \theta_1 x$
- 2) $y = \theta_0 + \theta_1 x + \theta_2 x^2$
- 3) $y = \sum_{j=0}^{5} \theta_j x^j$

## Underfitting and Overfitting

- 1) underfitting : data shows structure not captured by the model

## Underfitting and Overfitting

- 1) underfitting : data shows structure not captured by the model
- 3) overfitting: the fitted curve passes through the data perfectly

## Underfitting and Overfitting

- 1) underfitting : data shows structure not captured by the model
- 3) overfitting: the fitted curve passes through the data perfectly
- We would not expect this to be a very good predictor of, housing prices ($y$) for different living areas ($x$)

## Logistic Regression

Logistic Regression

## Classification

- This is just like the regression problem, except that the values $y$ take on only a small number of discrete values

## Classification

- This is just like the regression problem, except that the values $y$ take on only a small number of discrete values
- Focus on the binary classification problem in which $y$ can take on only two values, 0 and 1.

## Classification

- This is just like the regression problem, except that the values $y$ take on only a small number of discrete values
- Focus on the binary classification problem in which $y$ can take on only two values, 0 and 1.
- spam classifier: $x^{(i)}$ may be some features of a piece of email, $y$ may be 1 if it is a piece of spam mail, and 0 otherwise

## Classification

- This is just like the regression problem, except that the values $y$ take on only a small number of discrete values
- Focus on the binary classification problem in which $y$ can take on only two values, 0 and 1.
- spam classifier: $x^{(i)}$ may be some features of a piece of email, $y$ may be 1 if it is a piece of spam mail, and 0 otherwise
- Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the label for the training example.

## Logistic Regression

- We could use linear regression algorithm

## Logistic Regression

- We could use linear regression algorithm
- It is easy to construct example where this method performs very poorly

## Logistic Regression

- We could use linear regression algorithm
- It is easy to construct example where this method performs very poorly
- It also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$

## Logistic Regression

- We could use linear regression algorithm
- It is easy to construct example where this method performs very poorly
- It also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$
- Thus, the hypotheses $h_\theta(x)$ change:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

## Logistic Regression

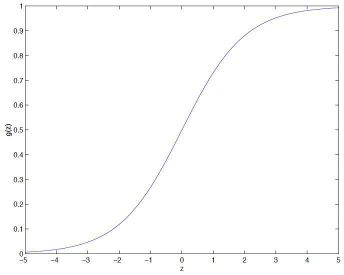
- We could use linear regression algorithm
- It is easy to construct example where this method performs very poorly
- It also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$
- Thus, the hypotheses $h_\theta(x)$ change:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

- where g is called the logistic or the sigmoid function

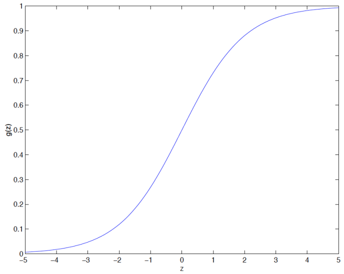
$$g(z) = \frac{1}{1 + e^{-z}}$$

## Sigmoid Function



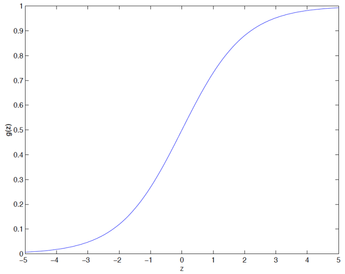
- $g(z)$ tends towards 1 as $z \to \infty$

## Sigmoid Function



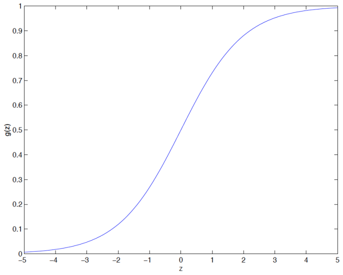
- $g(z)$ tends towards 1 as $z \to \infty$
- $g(z)$ tends towards 0 as $z \to -\infty$

## Sigmoid Function



- $g(z)$ tends towards 1 as $z \to \infty$
- $g(z)$ tends towards 0 as $z \to -\infty$
- $g(z)$, and hence $h(x)$, is always bounded between 0 and 1

## Sigmoid Function



- $g(z)$ tends towards 1 as $z \to \infty$
- $g(z)$ tends towards 0 as $z \to -\infty$
- $g(z)$, and hence $h(x)$, is always bounded between 0 and 1
- $x_0 = 1$, and $\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$

## Derivative of the Sigmoid Function

$$\begin{aligned}
g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\
&= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\
&= g(z)(1-g(z)).
\end{aligned}$$

## Likelihood

Let us assume that

$$p(y = 1|x; \theta) = h_\theta(x)$$

$$p(y = 0|x; \theta) = 1 - h_\theta(x)$$

this can be written more compactly as

$$p(y|x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

## Likelihood II

Assuming that the $m$ training examples were generated independently, the likelihood of the parameters is

$$\begin{aligned} L(\theta) &= p(\vec{y}|X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m}(h_\theta(x^{(i)}))^{(y(i))}(1-h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

It will be easier to maximize the log likelihood:

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1-y^{(i)}) \log(1-h(x^{(i)})) \end{aligned}$$

## Gradient Ascent

To maximize the likelihood, we can use gradient ascent

$$\theta := \theta + \alpha \nabla_\theta l(\theta)$$

Consider on training example $(x, y)$ and take derivatives to derive stochastic gradient ascent rule:

$$\begin{aligned}
\frac{\partial}{\partial(\theta)} l(\theta) &= \left( y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= \left( y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) g(\theta^T x)(1-g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= \left( y(1-g(\theta^T x)) - (1-y) g(\theta^T x) \right) x_j \\
&= (y - h_\theta(x)) x_j
\end{aligned}$$

## Gradient Ascent II

We used the fact that $g'(z) = g(z)(1 - g(z))$, this gives us the stochastic gradient ascent rule

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

- If we compare to the LMS update rule, it looks identical

## Gradient Ascent II

We used the fact that $g^{'}(z) = g(z)(1 - g(z))$, this gives us the stochastic gradient ascent rule

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

- If we compare to the LMS update rule, it looks identical
- but $h_\theta(x^{(i)})$ is defined as a non-linear function of $\theta^T x^{(i)}$

## Model Selection

- Given $h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_k x^k)$

## Model Selection

- Given $h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_k x^k)$
- We wish to decide if $k$ should be $0, 1, \ldots$ or 10

## Model Selection

- Given $h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_k x^k)$
- We wish to decide if $k$ should be $0, 1, \ldots$ or 10
- How can we automatically select a model with a good bias and variance tradeoff?

## Model Selection

- Given $h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_k x^k)$
- We wish to decide if $k$ should be $0, 1, \ldots$ or 10
- How can we automatically select a model with a good bias and variance tradeoff?
- Assume we have a set of models $M = \{M_1, \ldots, M_d\}$ that we are trying to select among.

## First Attempt

Given a training set $S$

- Train each model $M_i$ on $S$, to get some hypothesis $h_i$

## First Attempt

Given a training set $S$

- Train each model $M_i$ on $S$, to get some hypothesis $h_i$
- Pick the hypotheses with the smallest training error

## First Attempt

Given a training set $S$

- Train each model $M_i$ on $S$, to get some hypothesis $h_i$
- Pick the hypotheses with the smallest training error
- This algorithm does not work, why?

## Cross Validation

Given a training set $S$

- Randomly split $S$ into $S_{\text{train}}$ (say, 70% of data) and $S_{\text{CV}}$ (the remainder 30%). $S_{\text{CV}}$ is called the hold-out cross validation set.

## Cross Validation

Given a training set $S$

- Randomly split $S$ into $S_{\text{train}}$ (say, 70% of data) and $S_{\text{CV}}$ (the remainder 30%). $S_{\text{CV}}$ is called the hold-out cross validation set.
- Train each model $M_i$ on $S_{\text{train}}$ only, to get some hypothesis $h_i$

## Cross Validation

Given a training set $S$

- Randomly split $S$ into $S_{\text{train}}$ (say, 70% of data) and $S_{\text{CV}}$ (the remainder 30%). $S_{\text{CV}}$ is called the hold-out cross validation set.
- Train each model $M_i$ on $S_{\text{train}}$ only, to get some hypothesis $h_i$
- Select and output the hypothesis $h_i$ that had the smallest error $\hat{E}_{S_{CV}}(h_i)$ on the hold out cross validation set.

## k-fold Cross Validation

k-fold cross validation holds out less data each time:

- Randomly split $S$ into $k$ disjoint subsets of $m/k$ training examples each $(S_1, \ldots, S_k)$

## k-fold Cross Validation

k-fold cross validation holds out less data each time:

- Randomly split $S$ into $k$ disjoint subsets of $m/k$ training examples each $(S_1, \dots, S_k)$
- For each model $M_i$, we evaluate it as follows:
  For $j = 1, \dots, k$
  - Train the model $M_i$ on $S_1 \cup \dots \cup S_{j-1} \cup S_{j+1} \cup \dots \cup S_k$ to get some hypothesis $h_{ij}$
  - Test the hypothesis $h_{ij}$ on $S_j$, to get $\hat{E}_{S_{CV}}(h_{ij})$

  The estimated generalization error of the model $M_i$ is then calculated as the average of the $\hat{E}_{S_{CV}}(h_{ij})$'s

## k-fold Cross Validation

k-fold cross validation holds out less data each time:

- Randomly split $S$ into $k$ disjoint subsets of $m/k$ training examples each $(S_1, \ldots, S_k)$
- For each model $M_i$, we evaluate it as follows:
  For $j = 1, \ldots, k$
  - Train the model $M_i$ on $S_1 \cup \ldots \cup S_{j-1} \cup S_{j+1} \cup \ldots \cup S_k$ to get some hypothesis $h_{ij}$
  - Test the hypothesis $h_{ij}$ on $S_j$, to get $\hat{E}_{S_{CV}}(h_{ij})$

  The estimated generalization error of the model $M_i$ is then calculated as the average of the $\hat{E}_{S_{CV}}(h_{ij})$'s
- Pick the model $M_i$ with the lowest estimated generalization error, and retrain that model on the entire training set $S$. The resulting hypothesis is the output as our final answer.

## k-fold Cross Validation II

- A typical choice for k is 10

## k-fold Cross Validation II

- A typical choice for k is 10
- This procedure may also be more computationally expensive

## k-fold Cross Validation II

- A typical choice for k is 10
- This procedure may also be more computationally expensive
- In problems with data is really scarce, sometimes $k = m$

## k-fold Cross Validation II

- A typical choice for k is 10
- This procedure may also be more computationally expensive
- In problems with data is really scarce, sometimes $k = m$
- This method is called leave-one-out cross validation

## Feature Selection

- Imagine that you have a problem where the number of features $n$ is very large (perhaps $n \gg m$)

## Feature Selection

- Imagine that you have a problem where the number of features $n$ is very large (perhaps $n \gg m$)
- You suspect that there is only a small number of features that are "relevant"

## Feature Selection

- Imagine that you have a problem where the number of features $n$ is very large (perhaps $n \gg m$)
- You suspect that there is only a small number of features that are "relevant"
- You can apply a feature selection algorithm to reduce the number of features

## Feature Selection

- Imagine that you have a problem where the number of features $n$ is very large (perhaps $n \gg m$)
- You suspect that there is only a small number of features that are "relevant"
- You can apply a feature selection algorithm to reduce the number of features
- Given $n$ features, there are $2^n$ posible feature subsets

## Feature Selection

- Imagine that you have a problem where the number of features $n$ is very large (perhaps $n \gg m$)
- You suspect that there is only a small number of features that are "relevant"
- You can apply a feature selection algorithm to reduce the number of features
- Given $n$ features, there are $2^n$ posible feature subsets
- It is usually too expensive to enumerate over and compare all $2^n$ models $\rightarrow$ heuristic search

## Forward Search

1. Initialize $\mathcal{F} = \emptyset$
2. Repeat $\{$
   (a) For $i = 1, \ldots, n$ if $i \notin \mathcal{F}$, let $\mathcal{F}_i = \mathcal{F} \cup \{i\}$, and use some version of cross validation to evaluate features $\mathcal{F}_i$.
   (b) Set $\mathcal{F}$ to be the best feature subset found on step (a).
   $\}$
3. Select and output the best feature subset that was evaluated during the entire search procedure.

## Forward Search II

- The outer loop ends either when $\mathcal{F} = \{1, \dots, n\}$ , or when $|\mathcal{F}|$ exceeds a threshold
- The algorithm is one instantiation of wrapper model feature selection
- Backward search starts off with $\mathcal{F} = \{1, \dots, n\}$ and repeatedly deletes features one at time until $\mathcal{F} = \emptyset$
- Wrapper feature selection algorithms often work well, but can be computationally expensive given they need to make many calls to the learning algorithm

## Filter Feature Selection

- The idea is to compute some simple score $S(i)$ that measures how informative each feature $x_i$ is about the class labels $y$
- Then, we simply pick the $k$ features with the largest scores $S(i)$
- $S(i)$ could be the correlation between $x_i$ and $y$, as measured on the training data
- It is more common (particularly for discrete-valued features $x_i$) to choose $S(i)$ to be the mutual information between $x_i$ and $y$:

$$M(x_i, y) = \sum_{x_i \in \{0,1\}} \sum_{y \in \{0,1\}} p(x_i, y) \log \frac{p(x_i, y)}{p(x_i)p(y)}$$

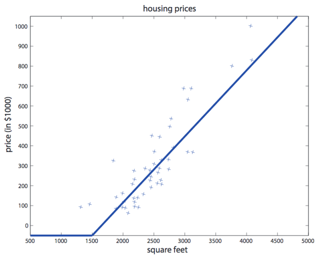
## Filter Feature Selection II

- MI can also be expressed as a Kullback-Leibler (KL) divergence $MI(x_i, y) = KL(p(x_i, y)||p(x_i)p(y))$
- This gives a measure of how different the probability distributions $p(x_i, y)$ and $p(x_i)p(y)$ are.
- If $x_i$ and $y$ are independent random variables then we would have $p(x_i, y) = p(x_i)p(y)$, and the KL-divergence will be zero
- Then $x_i$ is clearly very non-informative about $y$ and thus the score $S(i)$ should be small
- Conversely, if $x_i$ is very informative about $y$, then $MI(x_i, y)$ would be large

Neural Networks

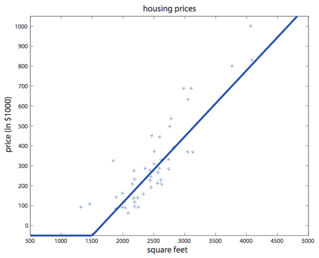
## Neural Networks Example

- Recall the housing price prediction problem



## Neural Networks Example

- Recall the housing price prediction problem
- We wish to prevent negative housing prices by setting the absolute minimum price as zero:



## ReLU Function

- Goal: $f : x \rightarrow y$

## ReLU Function

- Goal: $f : x \to y$
- Simplest possible neural networks as single “neuron” in the network where $f(x) = \max(ax + b, 0)$, for some coefficients $a, b$

## ReLU Function

- Goal: $f : x \rightarrow y$
- Simplest possible neural networks as single "neuron" in the network where $f(x) = \max(ax + b, 0)$, for some coefficients $a, b$
- This function is called a ReLU (rectified linear unit)

## ReLU Function

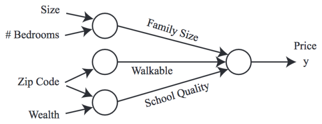
- Goal: $f : x \to y$
- Simplest possible neural networks as single "neuron" in the network where $f(x) = \max(ax + b, 0)$, for some coefficients $a, b$
- This function is called a ReLU (rectified linear unit)
- A more complex neural network may take the single neuron and stack them together such that one neuron passes its output as input into the next neuron

## ReLU Function

- Goal: $f : x \to y$
- Simplest possible neural networks as single “neuron” in the network where $f(x) = \max(ax + b, 0)$, for some coefficients $a, b$
- This function is called a ReLU (rectified linear unit)
- A more complex neural network may take the single neuron and stack them together such that one neuron passes its output as input into the next neuron
- That results in a more complex function

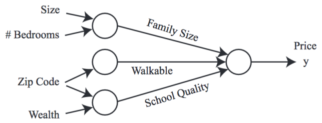
## Housing Prediction Revisited

- features: number of bedrooms, zip code, wealth of the neighborhood



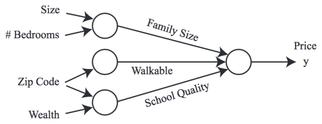
## Housing Prediction Revisited

- features: number of bedrooms, zip code, wealth of the neighborhood
- Building neural networks is analogous to Lego bricks: we take individual neurons and stack them together to create complex neural networks



## Housing Prediction Revisited

- features: number of bedrooms, zip code, wealth of the neighborhood
- Building neural networks is analogous to Lego bricks: we take individual neurons and stack them together to create complex neural networks
- Given three derived features (Family size, walkable school quality), the price of the home depends on these features



## Neural Network Learning

- All you need are the input features $x$ and the output $y$ while the neural network will figure out everything in the middle by itself

## Neural Network Learning

- All you need are the input features $x$ and the output $y$ while the neural network will figure out everything in the middle by itself
- The process of a neural network learning the intermediate features is called end-to-end learning

## Neural Network Learning

- All you need are the input features $x$ and the output $y$ while the neural network will figure out everything in the middle by itself
- The process of a neural network learning the intermediate features is called end-to-end learning
- The input to a NN is a set of input $x_1, x_2, x_3, x_4$.

## Neural Network Learning

- All you need are the input features $x$ and the output $y$ while the neural network will figure out everything in the middle by itself
- The process of a neural network learning the intermediate features is called end-to-end learning
- The input to a NN is a set of input $x_1, x_2, x_3, x_4$.
- We connect these four feature to three neurons (called hidden units)

## Neural Network Learning

- All you need are the input features $x$ and the output $y$ while the neural network will figure out everything in the middle by itself
- The process of a neural network learning the intermediate features is called end-to-end learning
- The input to a NN is a set of input $x_1, x_2, x_3, x_4$.
- We connect these four feature to three neurons (called hidden units)
- Goal NN: automatically determine three relevant features so as to predict the price of a house

## Neural Network Layers

- Black box: it can be difficult to understand the features it has invented

## Neural Network Layers

- Black box: it can be difficult to understand the features it has invented
- input layer (3 units), hidden layer(4 units), output layer (1 unit)

## Neural Network Layers

- Black box: it can be difficult to understand the features it has invented
- input layer (3 units), hidden layer(4 units), output layer (1 unit)
- the first hidden unit requires the input $x_1, x_2, x_3$ and outputs a value denoted by $a_1$

## Neural Network Layers

- Black box: it can be difficult to understand the features it has invented
- input layer (3 units), hidden layer(4 units), output layer (1 unit)
- the first hidden unit requires the input $x_1, x_2, x_3$ and outputs a value denoted by $a_1$
- $a$ refers to the neuron's activation value

## Neural Network Layers

- Black box: it can be difficult to understand the features it has invented
- input layer (3 units), hidden layer(4 units), output layer (1 unit)
- the first hidden unit requires the input $x_1, x_2, x_3$ and outputs a value denoted by $a_1$
- $a$ refers to the neuron's activation value
- $a_1^{[1]}$ denote the output value of the first hidden unit in the first hidden layer

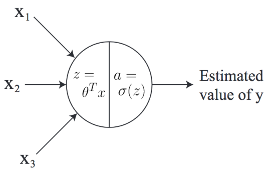
## Neural Network Layers

- Black box: it can be difficult to understand the features it has invented
- input layer (3 units), hidden layer(4 units), output layer (1 unit)
- the first hidden unit requires the input $x_1, x_2, x_3$ and outputs a value denoted by $a_1$
- $a$ refers to the neuron's activation value
- $a_1^{[1]}$ denote the output value of the first hidden unit in the first hidden layer
- The input layer is layer 0, the first hidden layer is 1 and the output is layer 2.

## Notation

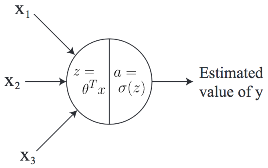
$$\begin{aligned} x_1 &= a_1^{[0]} \\ x_2 &= a_2^{[0]} \\ x_3 &= a_3^{[0]} \end{aligned}$$

## Logistic Regression as Neuron



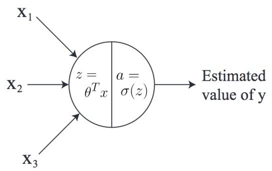
- We can look at at logistic regression $g(x)$ as a single neuron

## Logistic Regression as Neuron



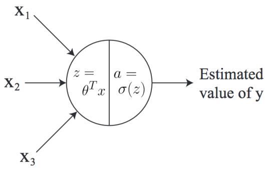
- We can look at at logistic regression $g(x)$ as a single neuron
- The input is three features: $x_1, x_2, x_3$, it outputs an estimated $y$ value

## Logistic Regression as Neuron



- We can look at at logistic regression $g(x)$ as a single neuron
- The input is three features: $x_1, x_2, x_3$, it outputs an estimated $y$ value
- break $g(x)$ two computations: 1) $z = w^T x + b$ and 2) $a = \sigma(z)$, where $\sigma(z) = 1/(1 + e^{-z})$

## Logistic Regression as Neuron



- We can look at at logistic regression $g(x)$ as a single neuron
- The input is three features: $x_1, x_2, x_3$, it outputs an estimated $y$ value
- break $g(x)$ two computations: 1) $z = w^T x + b$ and 2) $a = \sigma(z)$, where $\sigma(z) = 1/(1 + e^{-z})$
- notational difference, before $z = \theta^T x$, now $z = w^T x + b$ ($W$ will denote a matrix)

## Activation functions

$a = g(z)$, where $g(z)$ is some activation function:

$$\begin{aligned} g(z) &= \frac{1}{1+e^{-z}} && \text{(sigmoid)} \\ g(z) &= \max(z, 0) && \text{(ReLU)} \\ g(z) &= \frac{e^z - e^{-z}}{e^z + e^{-z}} && \text{(tanh)} \end{aligned}$$

In general, $g(z)$ is a non-linear function

## NN computation

- The first hidden unit in the fist hidden layer will perform: $z_1^{[1]} = {W_1^{[1]}}^T x + b_1^{[1]}$ and $a_1^{[1]} = g(z_1^{[1]})$

## NN computation

- The first hidden unit in the fist hidden layer will perform: $z_1^{[1]} = {W_1^{[1]}}^T x + b_1^{[1]}$ and $a_1^{[1]} = g(z_1^{[1]})$
- $W_1$ is the first row of $W$, a matrix of parameters

## NN computation

- The first hidden unit in the fist hidden layer will perform: $z_1^{[1]} = {W_1^{[1]}}^T x + b_1^{[1]}$ and $a_1^{[1]} = g(z_1^{[1]})$
- $W_1$ is the first row of $W$, a matrix of parameters
- $W_1^{[1]} \in \mathbb{R}^3$ and $b_1^{[1]} \in \mathbb{R}$

## NN computation

- The first hidden unit in the fist hidden layer will perform: $z_1^{[1]} = W_1^{[1]^T} x + b_1^{[1]}$ and $a_1^{[1]} = g(z_1^{[1]})$
- $W_1$ is the first row of $W$, a matrix of parameters
- $W_1^{[1]} \in \mathbb{R}^3$ and $b_1^{[1]} \in \mathbb{R}$
- second and third hidden units (first hidden layer):

$$\begin{aligned} z_2^{[1]} &= W_2^{[1]^T} x + b_2^{[1]} \text{ and } a_2^{[1]} = g(z_2^{[1]}) \\ z_3^{[1]} &= W_3^{[1]^T} x + b_3^{[1]} \text{ and } a_3^{[1]} = g(z_3^{[1]}) \end{aligned}$$

## NN computation II

The output layer performs:

$$z_1^{[2]} = {W_1^{[2]}}^T a^{[1]} + b_1^{[2]} \text{ and } a_1^{[2]} = g(z_1^{[2]})$$

where $a^{[1]}$ is the concatenation of all first layer activations:

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

For regression tasks one typically does not apply a non-linear function to $a_1^{[2]}$

## Vectorization

One must be careful when using for loops. In order to compute hidden unit activations in the first layer, we must compute:

$$\begin{aligned} z_1^{[1]} &= W_1^{[1]T} x + b_1^{[1]} \quad \text{and} \quad a_1^{[1]} = g(z_1^{[1]}) \\ &\vdots \qquad\qquad\qquad \vdots \qquad\qquad\quad \vdots \\ z_4^{[1]} &= W_4^{[1]T} x + b_4^{[1]} \quad \text{and} \quad a_4^{[1]} = g(z_4^{[1]}) \end{aligned}$$

Deep Learning algorithms have high computational requirements. As a result, code will run very slowly if you use for loops

## Vectorizing the Output Computation

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} — & {W_1^{[1]}}^T & — \\ — & {W_2^{[1]}}^T & — \\ & \vdots & \\ — & {W_4^{[1]}}^T & — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}$$

- matrix notation: $z^{[1]} = W^{[1]}x + b^{[1]}$

## Vectorizing the Output Computation

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} — W_1^{[1]T} — \\ — W_2^{[1]T} — \\ \vdots \\ — W_4^{[1]T} — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}$$

- matrix notation: $z^{[1]} = W^{[1]}x + b^{[1]}$
- $a^{[1]} = g(z^{[1]})$, use vectorized libraries (element-wise operations, e.g. ./ element-wise division Octave)

## Summary Vectorizing

- Given an input $x \in \mathbb{R}^3$,

## Summary Vectorizing

- Given an input $x \in \mathbb{R}^3$,
- We compute the hidden layer's activations: $z^{[1]} = W^{[1]}x + b^{[1]}$ and $a^{[1]} = g(z^{[1]})$

## Summary Vectorizing

- Given an input $x \in \mathbb{R}^3$,
- We compute the hidden layer's activations:
  $z^{[1]} = W^{[1]}x + b^{[1]}$ and $a^{[1]} = g(z^{[1]})$
- To compute the output layer's activations:

$$\underbrace{z^{[2]}}_{1\times1} = \underbrace{W^{[2]}}_{1\times4}\underbrace{a^{[1]}}_{4\times1} + \underbrace{b^{[2]}}_{1\times1} \quad \text{and} \quad \underbrace{a^{[2]}}_{1\times1} = g(\underbrace{z^{[2]}}_{1\times1})$$

## Non-linear Activation Functions

Why not use $g(z) = z$?. Assume that $b^{[1]}$ and $b^{[2]}$ are zeros:

$$\begin{aligned} z^{[2]} &= W^{[2]}a^{[1]} \\ &= W^{[2]}g(z^{[1]}) \\ &= W^{[2]}z^{[1]} \\ &= W^{[2]}W^{[1]}x \\ &= \hat{W}x \end{aligned}$$

- Appying a linear function to another linear function will result in a linear function over the original input
- Without non-linear activation functions, the NN will simply perform linear regression

## Vectorization Over Training Examples

Training set with three examples:

$$\begin{aligned} z^{[1](1)} &= W^{[1]}x^{(1)} + b^{[1]} \\ z^{[1](2)} &= W^{[1]}x^{(2)} + b^{[1]} \\ z^{[1](3)} &= W^{[1]}x^{(3)} + b^{[1]} \end{aligned}$$

Vectorization

$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix} = W^{[1]}X + b^{[1]}$$

## Putting it together

Suppose we have a training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$, where $x^{(i)}$ is a picture and $y^{(i)}$ is a binary label for whether the picture contains a cat or not

- Initialize the parameters $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$ to small random numbers

## Putting it together

Suppose we have a training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$, where $x^{(i)}$ is a picture and $y^{(i)}$ is a binary label for whether the picture contains a cat or not

- Initialize the parameters $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$ to small random numbers
- For each example, we compute the output "probability" from the sigmoid function $a^{[2](i)}$

## Putting it together

Suppose we have a training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$, where $x^{(i)}$ is a picture and $y^{(i)}$ is a binary label for whether the picture contains a cat or not

- Initialize the parameters $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$ to small random numbers
- For each example, we compute the output "probability" from the sigmoid function $a^{[2](i)}$
- Using the logistic regression log likelihood: $\sum_{i=1}^{m}(y^{(i)} \log a^{[2](i)} + (1 - y^{(i)}) \log(1 - a^{[2](i)}))$, we maximize this function using gradient ascent

## Putting it together

Suppose we have a training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$, where $x^{(i)}$ is a picture and $y^{(i)}$ is a binary label for whether the picture contains a cat or not

- Initialize the parameters $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$ to small random numbers
- For each example, we compute the output "probability" from the sigmoid function $a^{[2](i)}$
- Using the logistic regression log likelihood: $\sum_{i=1}^{m}(y^{(i)} \log a^{[2](i)} + (1 - y^{(i)}) \log(1 - a^{[2](i)}))$, we maximize this function using gradient ascent
- This maximization procedure corresponds to training the neural network

## Tarea

Implementar ejercicios indicados en:
logisticRegressionSemana2.ipynb