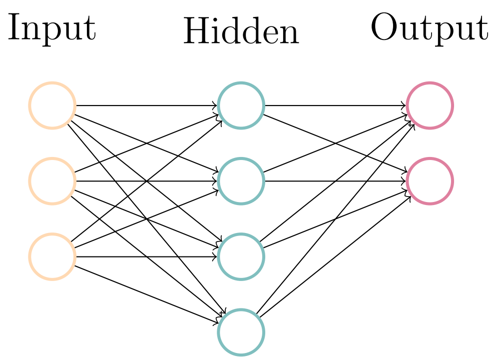
1 Week1

Introduction to Deep Learning.

1.1 What is a Neural Network?

In machine learning, a neural network (also artificial neural network or neural net, abbreviated ANN or NN) is a model inspired by the neuronal organization found in the biological neural networks in animal brains. (from wikipedia)

An artificial neural network is an interconnected group of nodes, inspired by a simplification of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.



1.2 Supervised Learning with Neural Networks

Standard NN, Convolutional NN, Recurrent NN.
Structured Data vs. Unstructured Data.

1.3 Why is Deep Learning taking off?

Scale drives deep learning progress.

1.4 About this Course

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring your Machine Learning project

4. Convolutional NeuralNetworks
5. Natural Language Processing: Building sequence models

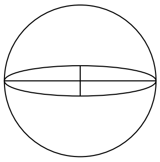
1.5 Outline of this Course

- Week1. Introduction
- Week2. Basics of Neural Network programming
- Week3. One hidden layer Neural Networks
- Week4. Deep Neural Networks

2 Week2

Basics of Neural Network Programming

How do I write an equation in LaTeX?



In 1902, Einstein created this equation: $E = mc^2$
And Newton came up with this one: $\sum F = ma$

$$5 + 5 = 10 \tag{1}$$

$$\begin{aligned} A &= \frac{5\pi r^2}{2} \\ A &= \frac{1}{2}\pi r^2 \end{aligned} \tag{2}$$

2.1 Neural Network Notations

General comments:
superscript (i) will denote the i^{th} training example.
superscript $[l]$ will denote the l^{th} layer.

Sizes:

- m: number of examples in the dataset
- n_x: input size
- n_y: output size
- $n_h^{[l]}$: number of hidden units of the l^th layer.
  In a for loop, it is possible to denote $n_x = n_h^{[0]}$ and $n_y = n_h^{[number of layer+1]}$
- L: number of layers in the network

Objects:

- $X \in \mathbb{R}^{n_x \times m}$ is the input matrix
- $x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th} example represented as a column vector
- $Y \in \mathbb{R}^{n_y \times m}$ is the label matrix
- $y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the i^{th} example
- $W^{[l]} \in \mathbb{R}^{\sharp \text{ of units in next layer} \times \sharp \text{ of units in the previous layer}}$ is the weight matrix, superscript $[l]$ indicates the layer
-

Common forward propagation equation examples:

-
-

Examples of cost functions:

-
-

2.2 Binary Classification

Use matrix without using for loops.
Computation using Forward propagation and Backward propagation.
Logistic regression is an algorithm for binary classification.

m training examples $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$ where $x^{(i)} \in \mathbb{R}^{n_x}$ and $y^{(i)} \in \{0, 1\}$ for $i \in [1, m]$

$$X = \begin{bmatrix} \vdots & \vdots & \vdots \\ X^{(1)} & X^{(1)} & X^{(m)} \\ \vdots & \vdots & \vdots \end{bmatrix} \in \mathbb{R}^{n_x \times m}$$

$$X.shape = (n_x, m)$$

$$Y = [Y^{(1)}, Y^{(2)}, \ldots, Y^{(m)}] \in \mathbb{R}^{1 \times m}$$

$$Y.shape = (1, m)$$

2.3 Logistic Regression

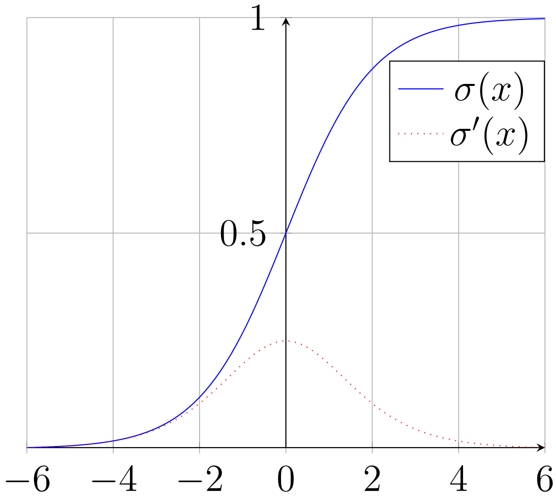
Given x, want $\hat{y} = P(y = 1|x)$ where $x \in \mathbb{R}^{n_x}$

Parameters: $w \in \mathbb{R}^{n_x}$ a n_x dimensional vector, $b \in \mathbb{R}$ a real number.

Output $\hat{y} = \sigma(w^T x + b) = \sigma(z)$

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

Drawing a sigmoid function and its derivative in tikz



2.4 Logistic Regression Cost Function

To train the parameter w and b of a Logistic Regression Model, we need a cost function.

$\hat{y} = \sigma(w^T X + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$

$\hat{y}^{(i)} = \sigma(w^T X^{(i)} + b)$ where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

Given $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

Loss(error) function (for a single training Example):
$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log{(1-\hat{y})})$

Cost function (for the entire training Examples):

$$\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log{(1-\hat{y}^{(i)})}]$$

The loss function computes the error for a single training example; the cost function is the average of the loss functions of the entire training set.

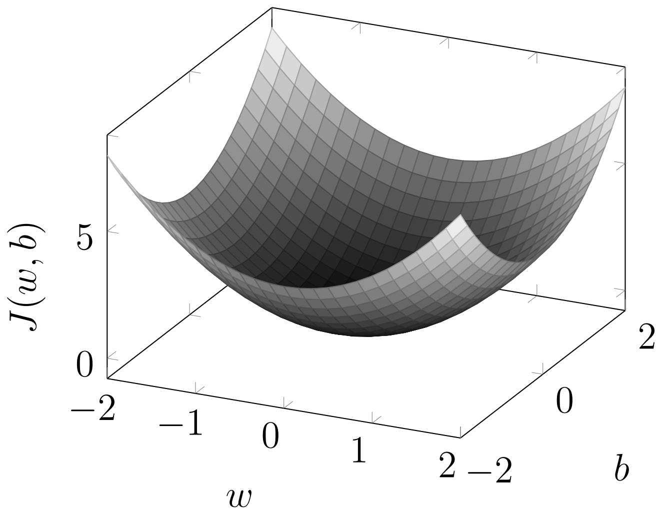
In training logistic regression model, we will try to find w and b such that they minimize the Cost function $\mathcal{J}(w, b)$.

Logistic Regression can be seen as a very small Neural Network.

2.5 Gradient Descent

$$\mathcal{J}(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})]$$

Want to find w and b that minimize the Cost function $\mathcal{J}(w,b)$.



$\mathcal{J}(w,b)$ is a convex function with a single local optimum.
No matter where you initialize the point, you should get to the same point (Global optimum).

Repeat:

$w := w - \alpha\frac{\partial\mathcal{J}(w,b)}{\partial w}$

$w := w - \alpha dw$

$b := b - \alpha\frac{\partial\mathcal{J}(w,b)}{\partial b}$

$b := b - \alpha db$

where α is the learning rate.

2.6 Derivatives

derivatives; slope

Given $f(a) = 3a$

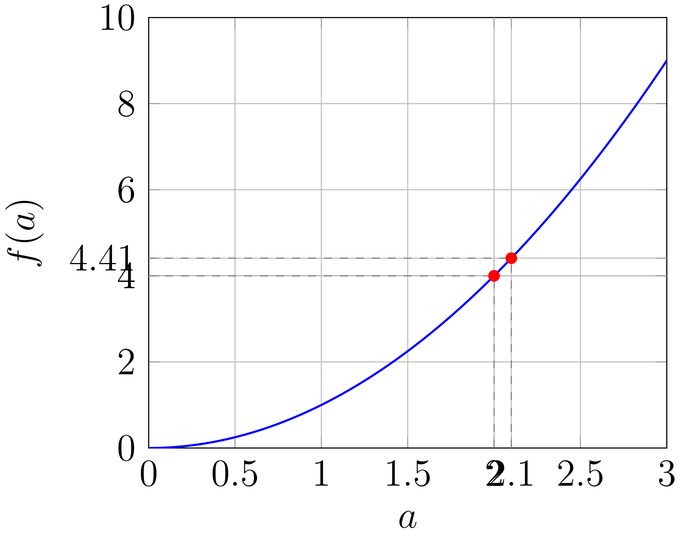
$\epsilon = .001, a = 2 + \epsilon$

$\frac{f(a)-f(a+\epsilon)}{\epsilon}$

make ϵ close to zero $\rightarrow$ derivatives.

$\frac{df(a)}{da} = \frac{d}{da}f(a)$

2.7 More Derivative Examples



$a = 2,\ f(a) = 4$

$a = 2.001,\ f(a) = 4.004001$

$\frac{d}{da} f(a) = 4$, when $a = 2$

$\frac{d}{da} f(a) = 10$, when $a = 5$

$\frac{d}{da} f(a) = \frac{d}{da} a^2 = 2a$

Given a nudge $\epsilon = 0.001$ to a, the $f(a)$ goes up $2 * a$.

2.8 Computation Graph

- Forward propagation step(forward pulse) : compute output of the network.
- Backward pulse: compute the gradients or derivatives.

$J(a, b, c) = 3(a + bc)$
$u = bc$
$v = a + u$
$J = 3v$

In order to compute derivatices, you go backward propagation.

One step of backward propagation on a computation graph yields derivative of final output variable.

2.9 Computing derivatives.

$u = bc$

$v = a + u$

$J = 3v$

Given $a = 5, b = 3, c = 2$, then $v = 11, J = 33$

We want to see how much J changes if we change the values of a, b, c, u, v for $J = 3v$

If we increase v to 11.001, then $J = 33.003$.

$\frac{dJ}{dv} = 3$

$a = 5 \rightarrow a = 5.001$

$v = 11 \rightarrow a = 11.001$

$J = 33 \rightarrow a = 33.003$

By Chain Rule:

$\frac{dJ}{da} = 3 = \frac{dJ}{dv}\frac{dv}{da} = 3 \times 1$

$\frac{dFinalOutputVar}{dvar}$ where var can be $a, b, c, ...$

We can simply denote $\frac{dJ}{dv} = dv$, $\frac{dJ}{da} = da$, etc with respect to J.

Similarly, $\frac{dJ}{du} = \frac{dJ}{dv}\frac{dv}{du} = 3 \times 1$

$\frac{dJ}{db} = \frac{dJ}{du}\frac{du}{db} = 3 \times 2 = 6$, where $u = bc = 2b$, and $\frac{du}{db} = 2$, given $a = 5, b = 3, c = 2$.

$\frac{dJ}{dc} = \frac{dJ}{du}\frac{du}{dc} = 3 \times 3 = 9$, where $u = bc = 3c$, and $\frac{du}{dc} = 3$, given $a = 5, b = 3, c = 2$.

$$\frac{dJ}{da} = 3$$
$$\frac{dJ}{du} = 3$$
$$\frac{dJ}{db} = 6$$
$$\frac{dJ}{dc} = 9$$

The coding convention $dvar$ represents: The derivative of a final output variable with respect to various intermediate quantities.

2.10 Logistic Regression Gradient Descent

Compute derivatives using Computation Graph(a bit overkill?) to implement/derive gradient descent for Logistic Regression.

Logistic regression recap
$z = w^T x + b$

$\hat{y} = a = \sigma(z)$

$\mathcal{L}(a, y) = -(ylog(a) + (1-y)log(1-a))$

Computation graph:

Given: x_1, w_1, b_1, x_2, w_2, b_2

$\boxed{z = w_1 x_1 + w_2 x_2 + b} \rightarrow \boxed{\hat{y} = a = \sigma(z)} \rightarrow \boxed{\mathcal{L}(a, y)}$

Modify w and b to reduce the loss $\mathcal{L}(a, y)$

In order to find such w and b, we compute the derivatives with respect to ?.

$da = \frac{d\mathcal{L}(a,y)}{da} = -\frac{d}{da}(ylog(a) + (1-y)log(1-a)) = -\frac{y}{a} + \frac{1-y}{1-a}$

$\boxed{\text{dz}} = \frac{d\mathcal{L}(a,y)}{dz} = \frac{d\mathcal{L}(a,y)}{da}\frac{da}{dz} = (-\frac{y}{a} + \frac{1-y}{1-a})\frac{da}{dz} = a - y$
, where $\frac{da}{dz} = \frac{d}{dz}(\frac{1}{1+e^{-z}}) = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}(1 - \frac{1}{1+e^{-z}}) = a(1-a)$

Go backward to compute :

$\boxed{dw_1} = \frac{\partial\mathcal{L}}{\partial w_1} = \frac{d\mathcal{L}(a,y)}{dw_1} = \frac{d\mathcal{L}(a,y)}{dz}\frac{dz}{dw_1} = x_1 dz$

$\boxed{dw_2} = \frac{\partial\mathcal{L}}{\partial w_2} = \frac{d\mathcal{L}(a,y)}{dw_2} = \frac{d\mathcal{L}(a,y)}{dz}\frac{dz}{dw_2} = x_2 dz$

$\boxed{\text{db}} = \frac{\partial\mathcal{L}}{\partial b} = \frac{d\mathcal{L}(a,y)}{db} = \frac{d\mathcal{L}(a,y)}{dz}\frac{dz}{db} = dz$

Compute dz to compute dw_1, dw_2, db and do the update with gradient descent:

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db$

2.11 Gradient Descent on m Examples

$\mathcal{J}(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(a^{(i)}, y^{(i)})$

, where $a^{(i)} = \hat{y^{(i)}} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$

$\frac{\partial}{\partial w_1}\mathcal{J}(w,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_1}\mathcal{L}(a^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} dw_1^{(i)}$

$\frac{\partial}{\partial w_2}\mathcal{J}(w,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_2}\mathcal{L}(a^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} dw_2^{(i)}$

$\frac{\partial}{\partial b}\mathcal{J}(w,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b}\mathcal{L}(a^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} db^{(i)}$

Logistic regression on m examples:

$J = 0;\ dw_1 = 0;\ dw_2 = 0;\ db = 0$

for $i = 1$ to m:
$z^{(i)} = w^T x^{(i)} + b$
$a^{(i)} = \sigma(z^{(i)})$
$\mathcal{J} += -[y^{(i)} log(a^{(i)}) + (1 - y^{(i)}) log(1 - a^{(i)})]$
$dz^{(i)} = a^{(i)} - y^{(i)}$

* The value of dw_1, dw_2, db in the code is cumulative:
$dw_1 += x_1^{(i)} dz^{(i)}$
$dw_2 += x_2^{(i)} dz^{(i)}$
$db += dz^{(i)}$

$J /= m,\ dw_1 /= m,\ dw_2 /= m,\ db /= m$

Finally, after finishing calculations for all m examples, we update(implment one step of gradient descent):

$w_1 := w_1 - \alpha dw_1$
$w_2 := w_2 - \alpha dw_2$
$b := b - \alpha db$

We have to multiple steps of above gradient descent.

It has two weaknesses: two for-loops (one for m training examples and another for features: $w_{(i)}$ where i can be big.) $\rightarrow$ $\boxed{\text{Vectorization!}}$

2.12 Vectorization

What is vectorization?

$$z = w^T + b, \text{ where } w \in \mathcal{R}^{n_x} \text{ and } x \in \mathcal{R}^{n_x}$$

```
import numpy as np

z = np.dot(w,x) + b
```

In Jupiter notebook:

```
import time

# 1. Vectorized version
a = np.random.rand(1000000)
b = np.random.rand(1000000)

tic = time.time()
c = np.dot(a,b)
toc = time.time()

print("1. Vectorized version:" + str(1000*(toc-tic))+ "ms")

# 2. For loop
c = 0
tic = time.time()
for i in range(1000000):
    c += a[i]*b[i]
toc = time.time()

print("2. For loop:" + str(1000*(toc-tic))+ "ms")
```

CPU and GPU has SIMD (single instruction multiple data). If you use built-in functions such as numpy's. It enables numpy to take better advantage of parallelization.

2.13 More Vectorization Examples

Whenever possible, avoid explicit for-loops

$u = Av$

$u_i = \sum_j A_{ij} v_j$ for $i = 1, \ldots, n$

1. Non-vectorized:

```
import numpy as np

u = np.zeros((n,1))
for i in range(n):
    for j in range(m):
        u[i] += A[i][j]*v[j]
```

2. Vectorized:

```
import numpy as np

u = np.dot(A,v)
```

Vectors and matrix valued functions.

say you need to apply the exponential operation on every element of a matrix/vector.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

$$u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

```
import numpy as np
u = np.zeros((n,1))

# 1. for-loop
for i in range(n):
    u[i] = math.exp(v[i])

# 2. Vectorized
u = np.exp(v)
u = np.log(v)
u = np.abs(v) # absolute value
u = np.maximum(v,0)
```

```
u = v**2
u = 1/v
```

Logistic regression derivatives

$J = 0$; $dw_1 = 0$; $dw_2 = 0$; $db = 0$

for $i = 1$ to m:
$z^{(i)} = w^T x^{(i)} + b$
$a^{(i)} = \sigma(z^{(i)})$
$\mathcal{J} += -[y^{(i)} log(a^{(i)}) + (1 - y^{(i)}) log(1 - a^{(i)})]$
$dz^{(i)} = a^{(i)} - y^{(i)}$

* The value of dw_1, dw_2, db in the code is cumulative:
$dw_1 += x_1^{(i)} dz^{(i)}$
$dw_2 += x_2^{(i)} dz^{(i)}$
$\ldots$
$dw_{n_x} += x_{n_x}^{(i)} dz^{(i)}$

$dw_2 += x_2^{(i)} dz^{(i)}$
$db += dz^{(i)}$

$J/ = m$, $dw_1/ = m$, $dw_2/ = m$, $db/ = m$

2.14 Vectorizing Logistic Regression

$z^{(i)} = w^T x^{(i)} + b$
$a^{(i)} = \sigma(z^{(i)})$ for $i = 1, \dots, m$

$$X = \begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ \vdots & \vdots & \dots & \vdots \end{bmatrix} \in \mathbb{R}^{n_x \times m}$$

$$w \in \mathbb{R}^{n_x \times 1}$$

$$b = \begin{bmatrix} b & b & \dots & b \end{bmatrix}$$

$$Z = w^T X + b = \begin{bmatrix} z^{(1)} & z^{(2)} & \dots & z^{(m)} \end{bmatrix} = \begin{bmatrix} w^T x^{(1)} + b & w^T x^{(2)} + b & \dots & w^T x^{(m)} + b \end{bmatrix}$$

, where $Z \in \mathbb{R}^{1 \times m}$

$$A = \begin{bmatrix} a^{(1)} & a^{(2)} & \dots & a^{(m)} \end{bmatrix} = \begin{bmatrix} \sigma(z^{(1)}) & \sigma(z^{(2)}) & \dots & \sigma(z^{(m)}) \end{bmatrix} = \sigma(Z)$$

```python
import numpy as np

# Broadcasting: even though b is in 1xR, it spans as a vector
Z = np.dot(w.T, x) + b
```

2.15 Vectorizing Logistic Regression's Gradient Output

$dz^{(i)} = a^{(i)} - y^{(i)}$
$dZ = \left[dz^{(1)}, \ \dots \ , dz^{(m)}\right]$

$$A = \left[a^{(1)}, \ \dots \ , a^{(m)}\right]$$

$$Y = \left[y^{(1)}, \ \dots \ , y^{(m)}\right]$$

$$dZ = A - Y = \left[a^{(1)} - y^{(1)}, \ \dots \ , a^{(m)} - y^{(m)}\right]$$

$J = 0;\ dw_1 = 0;\ dw_2 = 0;\ db = 0$

for $i = 1$ to m:
$z^{(i)} = w^T x^{(i)} + b$
$a^{(i)} = \sigma(z^{(i)})$
$\mathcal{J} += -[y^{(i)} log(a^{(i)}) + (1 - y^{(i)}) log(1 - a^{(i)})]$
$dz^{(i)} = a^{(i)} - y^{(i)}$

* The value of dw_1, dw_2, db in the code is cumulative:
$dw_1 += x_1^{(i)} dz^{(i)}$
$dw_2 += x_2^{(i)} dz^{(i)}$
...
$dw_{n_x} += x_{n_x}^{(i)} dz^{(i)}$

$dw_2 += x_2^{(i)} dz^{(i)}$
$db += dz^{(i)}$

$J /= m,\ dw_1 /= m,\ dw_2 /= m,\ db /= m$

$$db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)} = \frac{1}{m}\text{np.sum}(dz)$$

$$dw = \frac{1}{m} X dz^T = \frac{1}{m}\begin{bmatrix} \vdots & \dots & \vdots \\ X^{(1)} & \dots & X^{(m)} \\ \vdots & \dots & \vdots \end{bmatrix}\begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix} = \frac{1}{m}\left[X^{(1)} dz^{(1)} + \dots + X^{(m)} dz^{(m)}\right]$$

Vectorize:

$$dw += x^{(i)} dz^{(i)}$$

$$Z = w^T + b = np.dot(w.T, X) + b$$

$A = \sigma(Z)$

$dZ = A - Y$

$dw = \frac{1}{m} X dZ^T$

$db = \frac{1}{m}\ \text{np.sum}(dZ)$

$w := w - \alpha dw$

$b := b - \alpha db$

2.16