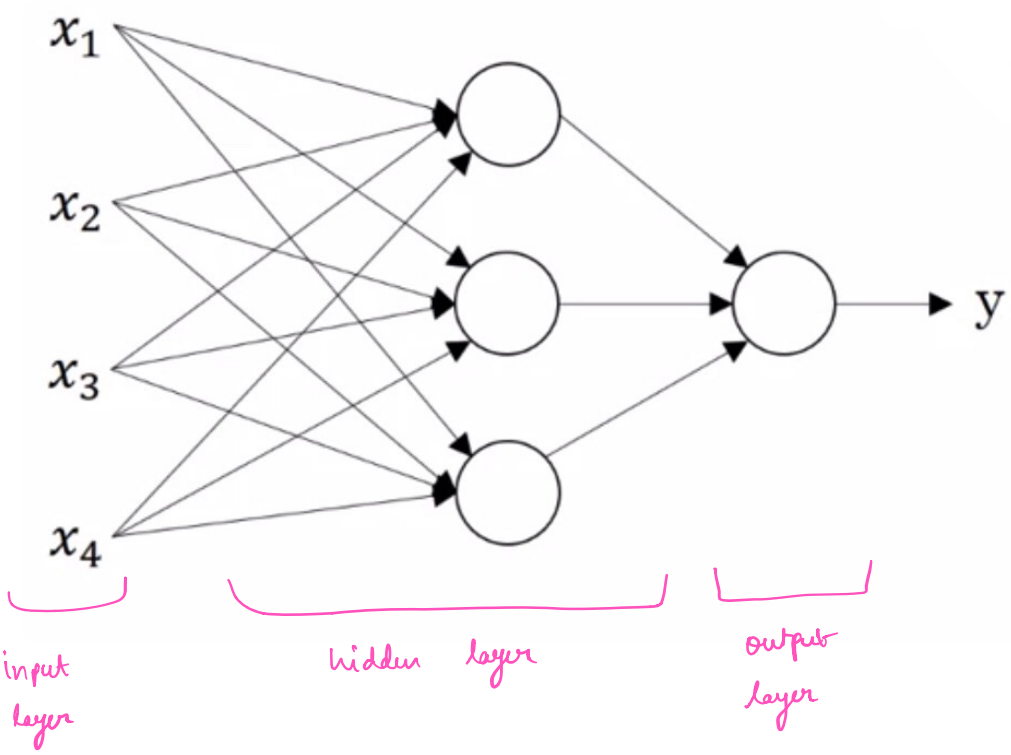
Week 1

Note:

ReLU - Rectified Linear Unit

↳ type of function

Neural Networks



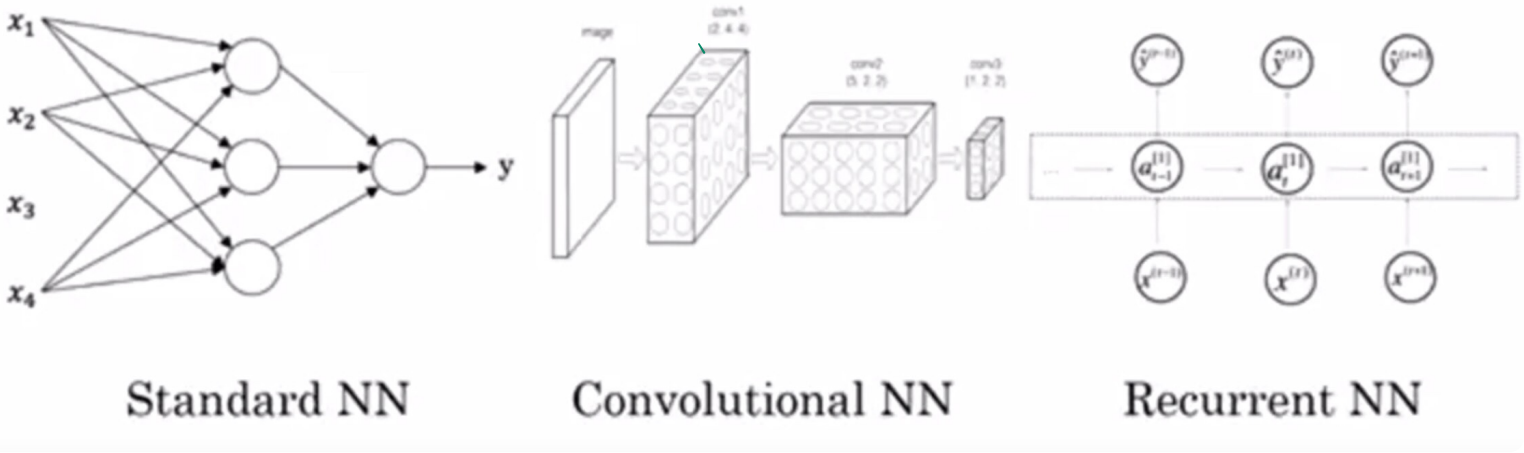
Types of NN

→ Standard NN

→ Convolutional NN: used for image applications

→ Recurrent NN: sequence data / played out over time / 1D time, for audio and language

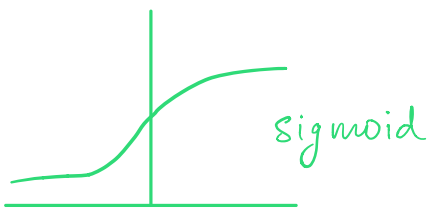
→ Complex Hybrid NN: Autonomous driving

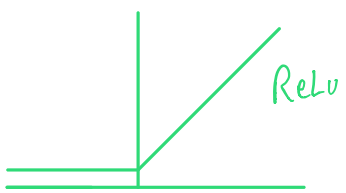


Supervised Learning:

- Structured data
  → each of the features has a defined meaning
- Unstructured data
  → images, audio, text

Note: m = # of training examples





Week 2

Binary Classification

- Yes / No classification

- Image : $\begin{bmatrix} \text{Red pixel values} \\ \vdots \\ \text{Blue pixel values} \\ \vdots \\ \text{Green pixel values} \end{bmatrix}$ $n_x = n = 64 \times 64 \times 3 = 12288$

  n = length of feature vector

- image → Yes / No $\quad x \rightarrow y$

Notation:

- (x, y) $\quad x \in \mathbb{R}^{n_x}$

  $y \in \{0, 1\}$

- m training examples

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}) \ldots\ldots (x^{(m)}, y^{(m)})\}$$

training set

m = # of training samples

- Matrix X

$$X = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \ldots\ldots & x_m \\ | & | & & | \end{bmatrix}$$ n_x (height), m (width)

$X \in \mathbb{R}^{n_x \times m}$

$X.shape = (n_x, m)$

- Matrix Y

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m)} \end{bmatrix} \qquad y \in \mathbb{R}^{1 \times m}$$

Logistic Regression

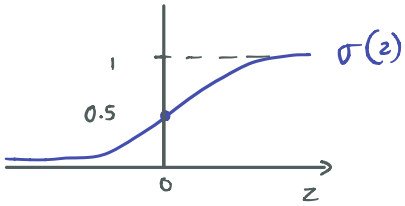
Given x, want $\hat{y} = P(y=1|x)$ → goal $\hat{y} \approx y$

$\hat{y}$: prediction, estimate of y

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

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

Sigmoid σ



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

z is large → $\sigma(z) \approx 1$

z is small → $\sigma(z) \approx 0$

- z

$$z^{(i)} = w^T x^{(i)} + b$$

Loss (error) Function: want as small as possible * on one single training sample

$$\mathcal{L}(\hat{y}, y) = \cancel{\frac{1}{2}(\hat{y}-y)^2} = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$$

$y=1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large

$y=0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log$ large, want $\hat{y}$ small

Cost Function * applied to the whole training set

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

$$= -\frac{1}{m}\sum_{i=1}^{m} \underbrace{\left(y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)})\right)}_{\text{Loss function}}$$

Gradient Descent

- Want to find w, b to minimize $J(w,b)$

repeat {

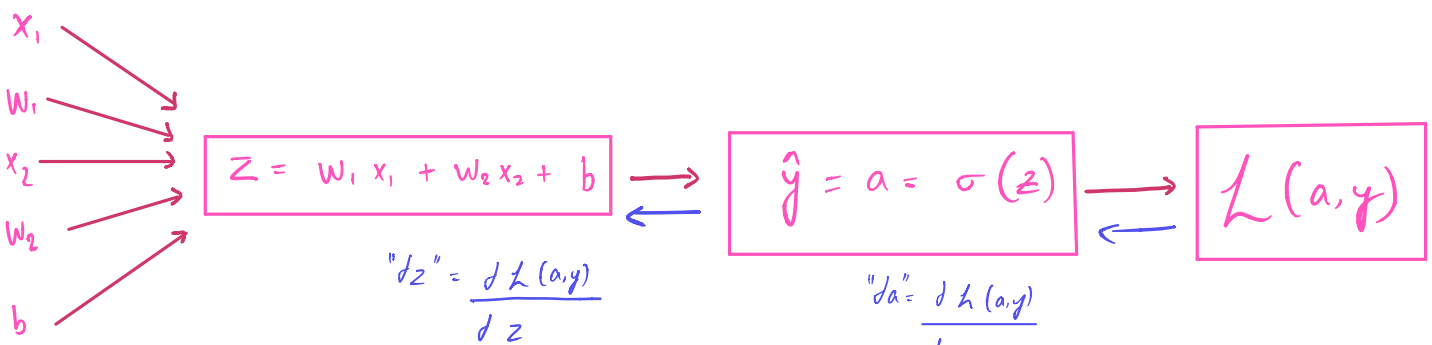
$$w := w - \alpha \frac{dJ(w)}{dw}$$ ← change in value

(learning rate)

$$b := b - \alpha \frac{dJ(w,b)}{db}$$

dw, db } in code

}



"dz" $= \frac{d\mathcal{L}(a,y)}{dz} = a - y = \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}$

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

$\frac{dL}{dw} =$ "dw_1" $= x \cdot dz$

$db = dz$

$$w := w - \alpha \underbrace{\frac{dJ(w)}{dw}}$$ → cost function

$$\frac{dJ(w)}{dw} = \frac{1}{m} \sum_{i=1}^{m} \frac{d}{dw_i} \overbrace{L(a^{(i)}, y^{(i)})}^{\text{lost function}}$$

Logistic Regression on m examples

→ getting rid of for loop in vectorization

(For i=1 to m)

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J = -\left[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log (1-a^{(i)})\right]$$
$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$
$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$ ↕ n=2
$$db \mathrel{+}= dz^{(i)}$$

$J /= m$

$dw_1 /= m$; $dw_2 /= m$; $db /= m$

$$dw_1 = \frac{dJ}{dw_1}$$

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

Vectorization

$$z = w^T x + b$$

Non-vectorized

```
z = 0
for i in range(n-x):
    z += w[i] * x[i]
z += b
```

vectorized

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

($w^T x$)

Using Numpy to avoid for loops

$dw = np.zeros(n, 1)$

For $i=1$ to m

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J = -\left[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log (1-a^{(i)})\right]$$
$$dz^{(i)} = a^{(i)} - y^{(i)}$$

For $j=1 \ldots n$ (i.e. $dw_j += \ldots$)

$dw_1 += x_1^{(i)} dz^{(i)}$

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

$n=2$, required loop

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

$J /= m$

$dw_1 /= m$; $dw_2 /= m$; $db /= m$

$$dw_1 = \frac{dJ}{dw_1}$$

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

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

$dw /= m$

using numpy

Vectorizing Logistic Regression

$$Z = w^T x + b$$
$$A = \sigma(Z)$$

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

n rows, m columns

$$\begin{bmatrix} z^{(1)} & z^{(2)} & \cdots & z^{(m)} \end{bmatrix} = w^T X + \underbrace{\begin{bmatrix} b & b & \cdots & b \end{bmatrix}}_{1 \times m} = \underbrace{\begin{bmatrix} w^T x^{(1)} + b & w^T x^{(2)} + b & \cdots & w^T x^{(m)} + b \end{bmatrix}}_{1 \times m}$$

$Z = np.dot(w.T, x) + b$ → adjusts automatically

$$A = [a^{(1)} \; a^{(2)} \; \dots \; a^{(m)}] = \sigma(z) \qquad Y = [y^{(1)} \; y^{(2)} \; \dots \; y^{(m)}]$$

$$dz = [dz^{(1)} \; dz^{(2)} \; \dots \; dz^{(m)}] = [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(2)} \; \dots] = A - Y$$

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

$$dw = \frac{1}{m} X \, dz^{T} = \frac{1}{m} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix} = \frac{1}{m} \underbrace{\left[x^{(1)} dz^{(1)} + \dots + x^{(m)} dz^{(m)} \right]}_{n \times 1}$$

For $i=1$ to m

$$z^{(i)} = w^{T} x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J = -\left[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log (1 - a^{(i)}) \right]$$
$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$
$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)} \qquad n = 2$$
$$db \mathrel{+}= dz^{(i)}$$

$J \mathrel{/}= m$

$dw_1 \mathrel{/}= m \; ; \; dw_2 \mathrel{/}= m \; ; \; db \mathrel{/}= m$

loop this for multiple gradient descent

$$z = w^{T} X + b = np.dot(w.T, x) + b$$
$$A = \sigma(z)$$
$$dz = A - Y$$
$$dw = \frac{1}{m} X \, dz^{T}$$
$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} = np.sum(dz)$$

vectorized : without for loops

* single iteration of gradient descent

Explanation of Logistic Regression Cost Function

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

$\hat{y} = p(y=1|x)$

If $y=1$: $p(y|x) = \hat{y}$

If $y=0$: $p(y|x) = 1-\hat{y}$

$\Big\} \; p(y|x)$

$a*b + a*c - b+c$

$a*(b+c) - b+c$

$(b+c)(a*-1)$

$p(y|x) = \hat{y}^{y} (1-\hat{y})^{(1-y)}$

If $y=1$: $\hat{y}^{1} \underbrace{(1-\hat{y})^{0}}_{=1} = \hat{y}$

If $y=0$: $\underbrace{\hat{y}^{0}}_{=1} (1-\hat{y})^{1} = 1-\hat{y} = 1 \times (1-\hat{y}) = (1-\hat{y})$

$\log p(y|x) = \log \hat{y}^{y} (1-\hat{y})^{(1-y)} = y \log \hat{y} + (1-y) \log (1-\hat{y})$

$= -\mathcal{L}(\hat{y}, y)$ ↓ minimize

Cost on m examples

$p(\text{labels in training set}) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)})$

$\log p(\ldots) = \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) = -\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

$= -\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

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

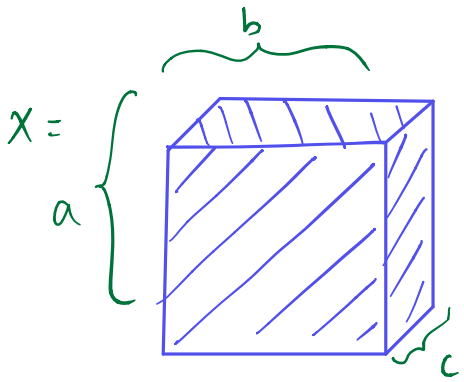
Numpy notes

$np.exp(x) = e^x$

x.shape → get the dimension of the matrix

x.reshape → shape x to the given dimension

x.shape[0] = a
x.shape[1] = b
x.shape[2] = c



PA

- training set m_train images labeled
- test set to be labeled
- image shape (num_px, num_px, 3)

Shapes of matrices

initial:

→ training set x (209, 64, 64, 3) training set y (1, 209)

→ test set x (50, 64, 64, 3) test set y (1, 50)

flattened:

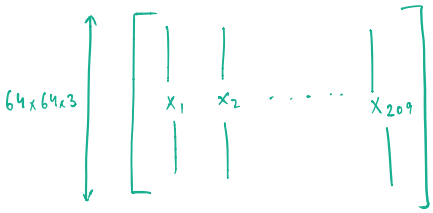
→ training set x (64*64*3, 209)

→ test set x (64*64*3, 50)

training set y (1, 209)

test set y (1,50)

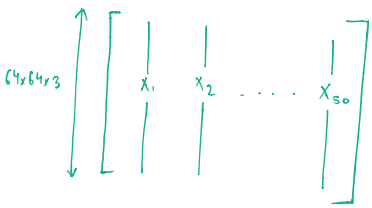
train x:



train y:

$[y^1 \; y^2 \; \ldots \; y^{209}]$

test x:



test y:

$[y^1 \; y^2 \; \ldots \; y^{50}]$

What you need to remember:

Common steps for pre-processing a new dataset are:

- Figure out the dimensions and shapes of the problem (m_train, m_test, num_px, ...)
- Reshape the datasets such that each example is now a vector of size (num_px * num_px * 3, 1)
- "Standardize" the data

Visualizing the PA

Sigmoid (z) $\quad z = w^T X + b$

Propagate

$A = z = [z^1 \; z^2 \; \ldots \; z^m]$

$$J(\text{cost}) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log (1-a^{(i)})$$

$$dw = \frac{dJ}{dw} \qquad db = \frac{dJ}{db}$$

Optimize

for {

propagate

$w: \; w - \alpha \, dw$

$b: \; b - \alpha \, db$

}