Course 1

Supervised Learning

25/1/24

Training set: Data used to train the model

Ex:

size in feet^2 (x)	price in \$1000's
2104	400
1416	232
1534	315
852	178
---	---
3210	870

x: "input" variable
feature

y = "output" variable
"target" variable

$(x^{(i)}, y^{(i)})$ = ith training example (x, y) = single training example (i.e, a row from a dataset)

Statistical Decision Theory - Regression

$X \in R^p$ — Input

$Y \in R$ — Output - Regression

$Pr(X, Y)$ $\{(x_1, y_1) \ldots (x_n, y_n)\}$

Linear Regression

$$f(X) : R^p \to R$$

$$\hat{y} = f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \beta_p X_p$$

$$X = (X_1, X_2, X_3, \ldots, X_p)$$

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

$$f(X) = \sum_{j=1}^{p} X_j \beta_j$$

k-Nearest Neighbours (Regressor)

(k-NN)

$$\hat{y}(x) = \frac{1}{k} \sum_{x_i \in N(x)} y_i$$

It will pick some neighbouring points to a data point, i.e, nearest and will average the outputs of those Neighbours and will give that average as $\hat{y}$.

(local average)

Loss function : $L(Y, f(X))$

↳ Square Error : $(Y - f(X))^2$

$$EPE(f) = E\{(Y - f(x))^2\}$$

E → Expected Value

(Expected Prediction Error)

$$= \int (y - f(x))^2 \cdot P_r(dx, dy)$$

Pr → Probability

$$P_r(X, Y) = P_r(Y|X) \cdot P_r(X)$$

$$EPE(f) = E_X E_{Y|X}([Y - f(x)]^2 | X)$$

* f(x) can be any function which can make the best possible prediction for every value of X.

(minimum Error)

$$f(x) = \arg\min_c E_{Y|X}([Y - C]^2 | X = x)$$

c → is the output for f(x)

c should picked in such a way that f(x) gives the smallest Error.

Conditioning on a point.

$$\therefore f(x) = E[Y | X = x]$$

Conditional Expectation (or) Regression function

$$\hat{f}(x) = \text{Ave}(y_i | x_i \in N_k(x))$$

Condition on a region.

Assumption: The output of $\hat{f}(x)$ is a constant over the region $N_k(x)$

$f(x) = x^T \beta$

$$X = \begin{bmatrix} x_{10} & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & \cdots & x_{2p} \\ \vdots & \vdots & & & \\ & x_{N1} & \cdots & \cdots & x_{Np} \end{bmatrix}$$

$$(Y - X\beta)^2 = EPE(f)$$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

* The output is globally linear and constant at a region.

<u>Statistical Decision Theory - Classification</u>

$X \in R^p$	$G \in \mathcal{G}$	$P(X, G)$
input	output Classification	$\{(x_1, g_1) \vdots (x_N, g_N)\}$

$f(x): R^p \to \mathcal{G}$

L is a $K \times K$ matrix, where K is the $K = card(\mathcal{G})$

Zero on the diagonal

$L(k, l)$ cost of classifying k as l

0-1 loss function

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \Rightarrow 3 \text{ categories}$$

$$EPE(f) = E[L(G, \hat{f})]$$

$$EPE(f) = E_X \, E_{G|X} [L(G, \hat{f}) | X]$$

$$= E_X \sum_{k=1}^{K} L(k, \hat{f}(x)) P(k|x)$$

$$\hat{f}(x) = \arg\min_g \sum_{k=1}^{K} L(k, g) \, P(k | X = x)$$

<u>0-1 loss:</u>

$$\hat{f}(x) = \arg\max_g P(g | X = x)$$

Bayes Optimal Classifier

Bias - Variance

$$Y = f(X) + \varepsilon, \quad E[\varepsilon] = 0, \quad Var[\varepsilon] = \sigma^2$$

$$EPE(x_0) = E\left[\left(Y - \hat{f}(x_0)\right)^2 \,\middle|\, X = x_0\right]$$

$$= \underbrace{E\left(E[\hat{f}(x_0)] - \hat{f}(x_0)\right)^2}_{\text{Variance}} + \underbrace{\left(E[\hat{f}(x_0)] - f(x_0)\right)^2}_{\text{Bias}} + \sigma^2$$

$$= \frac{\sigma^2}{k} + \left[f(x_0) - \left\{\frac{1}{k}\sum_{\rho=1}^{k} f(x_{(\rho)})\right\}\right]^2 + \sigma^2$$

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

tek analogy.

1	-1.0	-0.5	-1.947
1	-0.5	0.0	-0.391
1	1	2	6.047
1	2.5	-1.5	1.287

$$\beta_0 = \frac{2327}{9000} \quad \beta_1 = \frac{3173}{2250} \quad \beta_2 = 1.707177778$$

y = 3 7 9

60 00

39

Gym

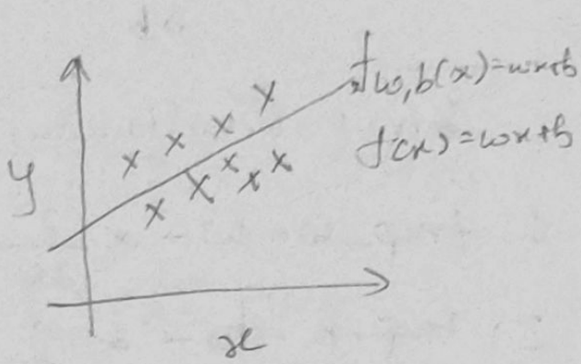
Linear Regression

training set
↓
learning algorithm
↓

$x \longrightarrow \boxed{f} \longrightarrow \hat{y}$

feature Model prediction (estimated y)

target

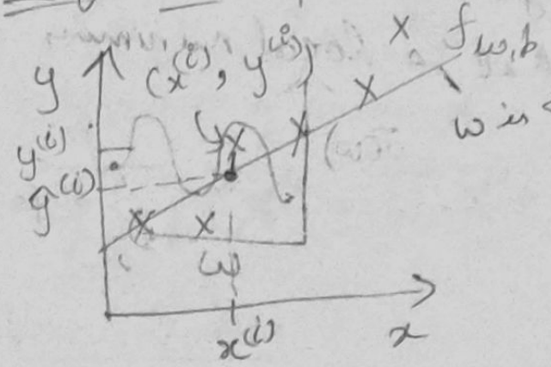


How to represent f?

$$f_{w,b}(x) = wx + b$$

Univariate (one variable) linear regression

Cost function: Squared error cost function



w is the slope of the fitting line

$$J_{(w,b)} = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

m = number of training examples

$$\hat{y}^{(i)} = f_{w,b}(x^{(i)})$$

$$f_{w,b}(x^{(i)}) = wx^{(i)} + b$$

$$\boxed{J_{(w,b)} = \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})^2}$$

Find w, b: (such that)

$\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$.

So, the cost function J depends on the value of w.

Gradient descent algorithm

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b) \qquad \alpha \rightarrow \text{learning rate}$$

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

Correct: Simultaneous Update

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

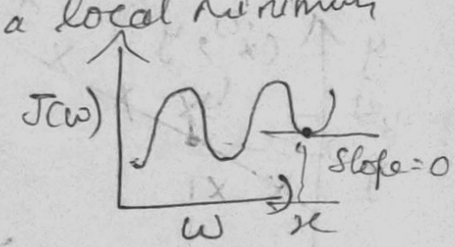
2: $tmp_b = b - \alpha \frac{\partial}{\partial b} J(w,b)$

3: $w = tmp_w$
$b = tmp_b$

4: Repeat until convergence (local minimum)

When the algorithm reaches a local minimum and if we still try to implement gradient descent on the local minima, there would be no changes in the parameters since the slope of the tangent at a local minimum is zero

$$\therefore \frac{d}{dw} J_x(w) = 0$$



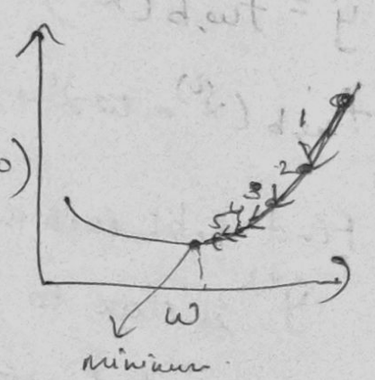
$$w = w - \alpha \frac{d}{dw} J_x(w)$$

$$w = w - \alpha \cdot 0$$

$w = w \Rightarrow$ No change will occur.

→ Near a local min,
- Derivative becomes smaller
- Update steps become smaller

Can reach min without decreasing learning rate α (i.e, α is constant all the time)



Gradient descent for linear regression

Linear regression model

$f_{w,b}(x) = wx + b$

Cost function

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{(w,b)}(x^{(i)}) - y^{(i)}\right)^2$$

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b) \rightarrow \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b) \rightarrow \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)$$

}

Linear Regression with Multivariables (Multiple linear Regression)

If there are n features then the model would be

$$f_{\vec{w},b}(x) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

$$\vec{w} = [w_1, w_2, w_3, \ldots, w_n]$$

$$\vec{x} = [x_1 \; x_2 \; x_3 \; \ldots \; x_n]$$

vector of feature x

$$\therefore f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

Vectorization

$\vec{w} = [w_1 \; w_2 \; w_3]$ $n=3$

b is a number

$\vec{x} = [x_1 \; x_2 \; x_3]$

linear algebra: count from 1

```
w = np.array([1.0, 2.5, -3.3])
b = 4
x = nparray([10, 20, 30])
```

code: count from 0

Without vectorization

$$f_{\vec{w},b} = \left(\sum_{j=1}^{n} w_j x_j\right) + b$$

```
f = 0
for j in range(n):
    f = f + w[j] * x[j]
f = f + b
```

With Vectorization

$f_{\vec{w},b} = \vec{w} \cdot \vec{x} + b$

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

Gradient descent (n feature $n \geq 2$)

Repeat {

$$w_1 = w_1 - \alpha \frac{1}{m} \boxed{\sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_1^{(i)}}$$

$$\hookrightarrow \frac{\partial}{\partial w_1} J(\vec{w}, b)$$

$\vdots$

$$w_n = w_n - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_n^{(i)}$$

~~Simultaneously update~~

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

Simultaneously update

w_j (for $j = 1, \ldots, n$) and b

}

Feature ~~Engineering (Scaling)~~ Scaling

$300 \leq x_1 \leq 2000$

$x_1, \text{scaled} = \frac{x_1}{2000 \text{ (max)}}$

$0.15 \leq x_1, \text{scaled} \leq 1$

$0 \leq x_2 \leq 5$

$x_2, \text{scaled} = \frac{x_2}{5}$

$0 \leq x_2, \text{scaled} \leq 1$

Mean normalization

$\mu_1 = 600$ (avg) $\quad 300 \leq x_1 \leq 2000$

$$x_1 = \frac{x_1 - \mu_2}{2000 - 300}$$

$$-0.18 \leq x_1 \leq 0.82$$

Z-score normalization

Standard deviation σ $\quad 300 \leq x_1 \leq 2000$

$$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$$

→ aim for about $-1 \le x_j \le 1$ for each feature

$-3 \le x_j \le 3$
$-0.3 \le x_j \le 0.3$ } acceptable ranges

$0 \le x_1 \le 3$ → okay, no rescaling is needed (can be performed also)

$-2 \le x_2 \le 0.5$ → okay

$-100 \le x_3 \le 1000$ → too large, should be re-scaled

$-0.001 \le x_4 \le 0.0001$ → too small, rescaling should be performed

* When in doubt, it is better to perform rescaling rather than training the model with given raw data.

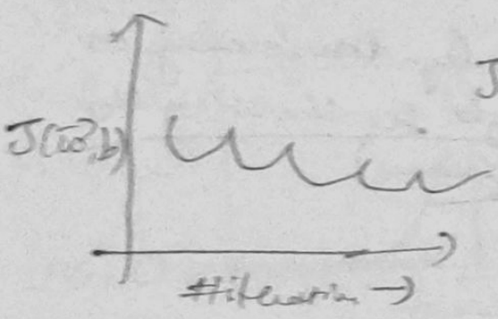
Automatic convergence test

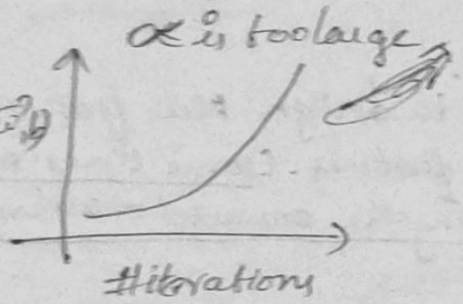
Let ϵ "epsilon" be 10^{-3}

If $J(\vec{w}, b)$ decreased by $\le \epsilon$ in one iteration, declare convergence (ie, find parameters $\vec{w}, b$ to get close to global minimum).

Identifying problem with gradient descent

Bug or learning rate is too large

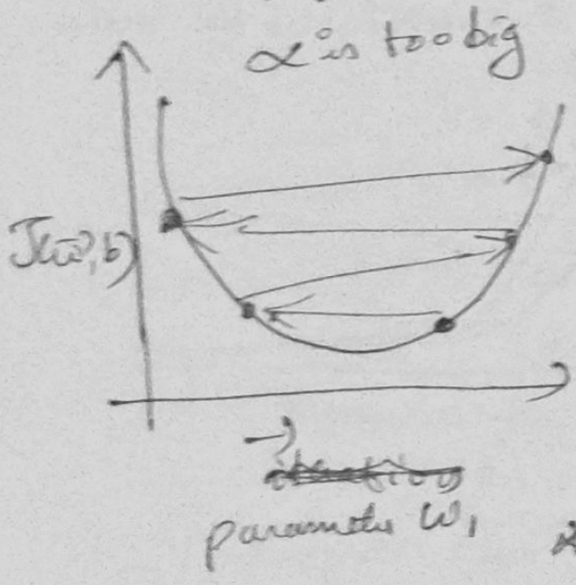


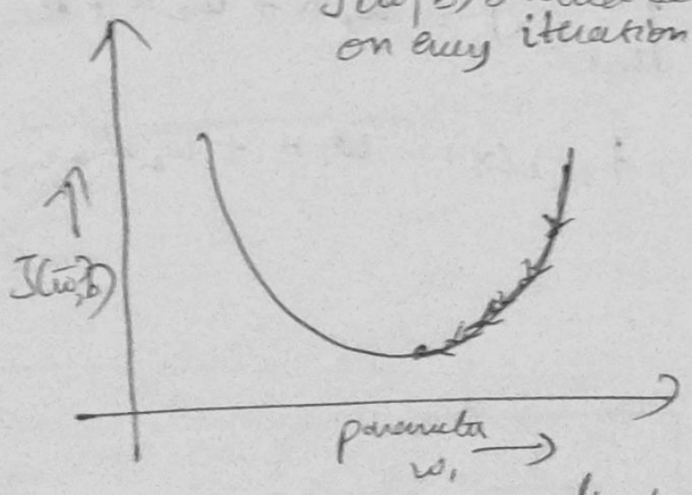


$w_1 = w_1 + \alpha d_1$ ✗
use a minus sign
$w_1 = w_1 - \alpha d_1$ ✓

* With a small enough α, $J(\vec{w}, b)$ should decrease on every iteration.

Adjust learning rate

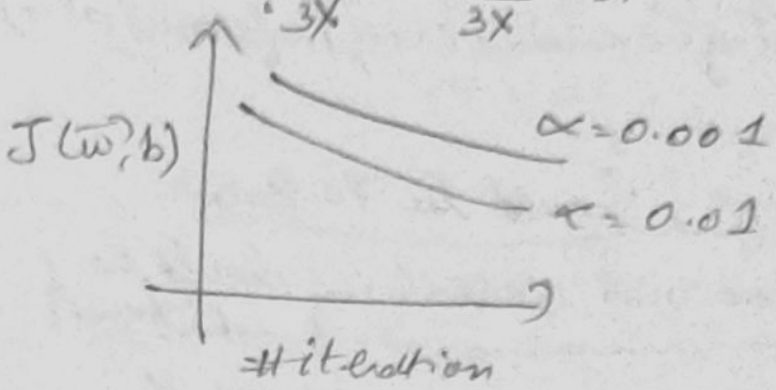




* If α is too small then gradient descent takes a lot more iterations to converge.

Values of α to try:

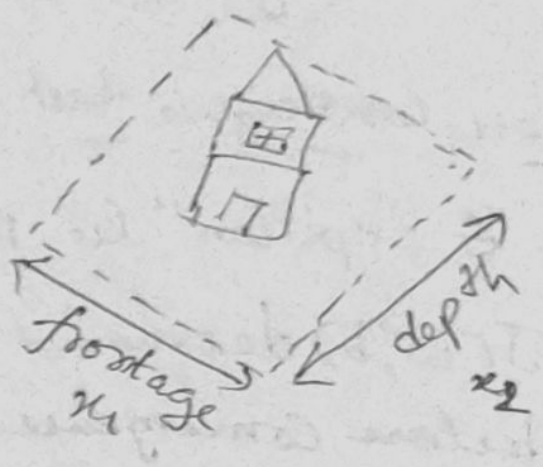
$$\cdots 0.001 \xrightarrow{3\times} 0.003 \xrightarrow{3\times} 0.01 \xrightarrow{3\times} 0.03 \xrightarrow{3\times} 0.1 \xrightarrow{3\times} 0.3 \xrightarrow{3\times} 1 \cdots$$



Feature Engineering

Ex: $f_{\vec{w},b}(\vec{x}) = w_1 \underbrace{x_1}_{\text{frontage}} + w_2 \underbrace{x_2}_{\text{depth}} + b$

area = frontage × depth



$x_3 = x_1 x_2$

new feature

$f(\vec{w},b)$ $f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$

* Feature Engineering:
Using intuition to design new features, by transforming or combining original features. (Some times it might be choosing the right features & leaving the unwanted or useless features)

Polynomial Regression

$$f_{w,b}(x) = w_1 x + w_2 x^2 + b$$

$$f_{w,b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + b$$

Classification

Question	Answer "y"	
Is this email spam?	no	yes
Is this transaction fraudulent?	no	yes
Is the tumor malignant?	no	yes

$\boxed{\text{y can only be one of two values}}$

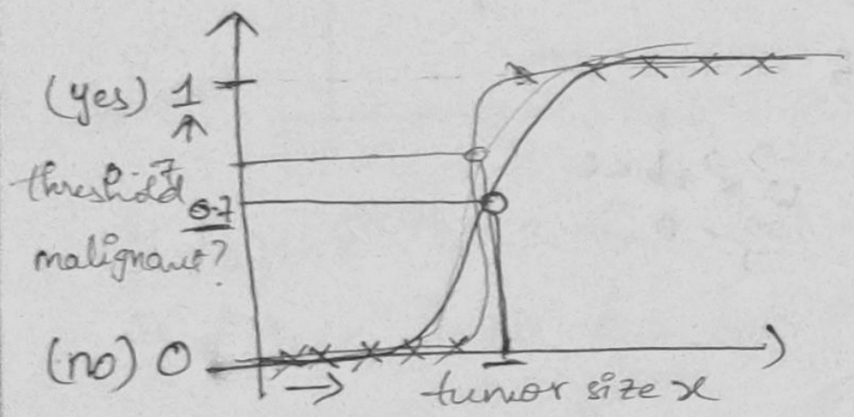
$\boxed{\text{"binary classification"}}$

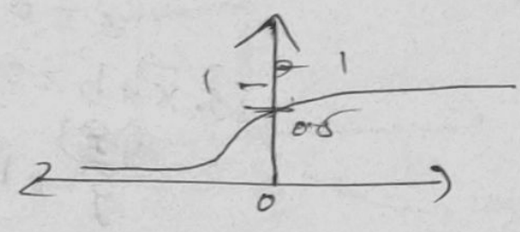
↳ 2 categories or classes

false true

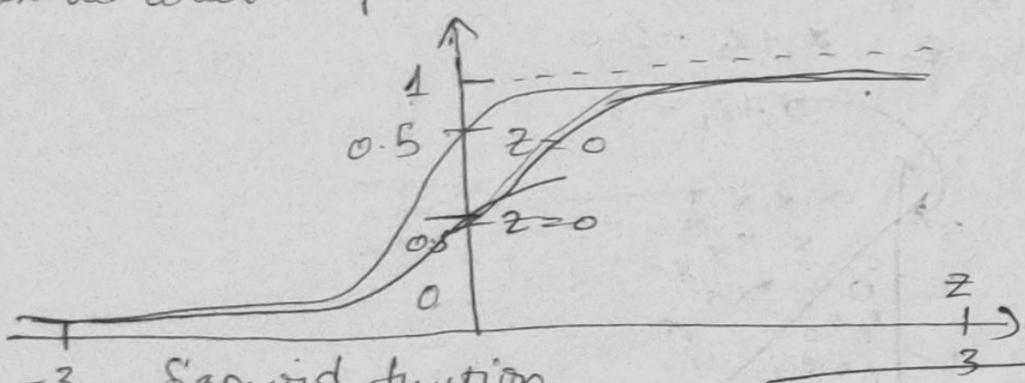
0 1

"negative class" "positive class"





When we want outputs 0 and 1



Sigmoid function

logistic function

outputs b/w 0 and 1

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

$0 < g(z) < 1$

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

$$f_{w,b}(\vec{x})$$

$$\boxed{z = \vec{w}\cdot\vec{x} + b}$$

$$\downarrow$$

$$z$$

$$\downarrow$$

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

logistic Regression.

Decision Boundary

$$f_{\vec{w},b}(\vec{x}) = g(\vec{w}\cdot\vec{x}+b) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

$$= P(y=1 \mid x; \vec{w}, b)$$

Is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?

Yes : $\hat{y} = 1$ No: $\hat{y} = 0$

when is $f_{\vec{w},b}(\vec{x}) \geq 0.5$?

$$g(z) \geq 0.5$$

$$z \geq 0.5$$

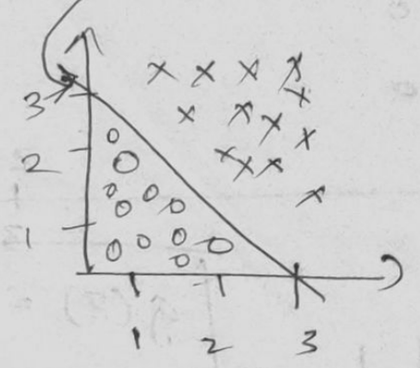
$\vec{w}\cdot\vec{x}+b \geq 0$ $\quad$ $\vec{w}\cdot\vec{x}+b < 0$

$\hat{y} = 1$ $\quad$ $\hat{y} = 0$

Decision Boundary $z = \vec{w}\cdot\vec{x}+b = 0$

$$z = x_1 + x_2 - 3 = 0$$

$$x_1 + x_2 = 3$$

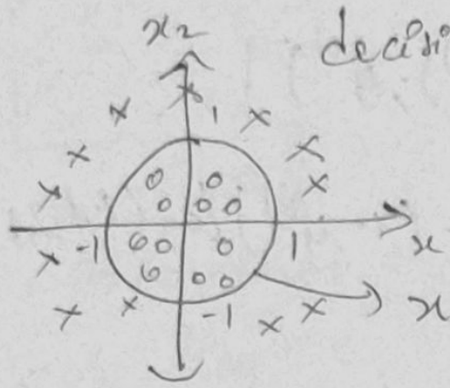


the $x_1 + x_2 = 3$ is the decision boundary

Non-linear decision boundaries

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\overbrace{w_1 x_1^2 + w_2 x_2^2 + b}^{z})$$

$$1 \quad 1 \quad -1$$



decision boundary $z = x_1^2 + x_2 - 1 = 0$

$$x_1^2 + x_2^2 = 1$$

$x_1^2 + x_2^2 \geq 1$ $\quad$ $x_1^2 + x_2^2 \leq 1$

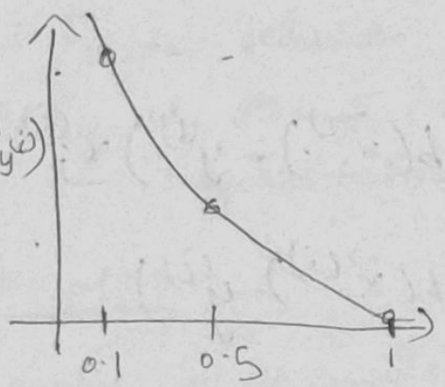
$\hat{y} = 0$ $\quad$ $\hat{y} = 1$

Logistic loss function

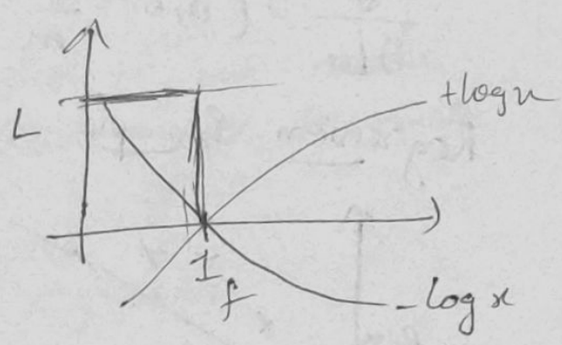
$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$

if $y^{(i)} = 1$



* Loss is lowest when $f_{\vec{w},b}(\vec{x}^{(i)})$ predicts close to true label $y^{(i)}$.

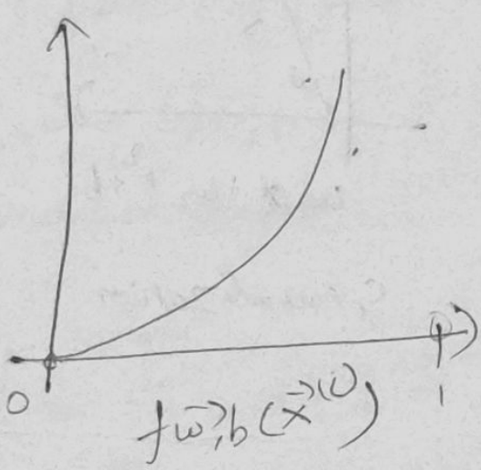


As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to 0$

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$ then loss $\to \infty$

$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$

if $y^{(i)} = 0$



* The farther prediction $f_{\vec{w},b}(\vec{x}^{(i)})$ is from target $y^{(i)}$, the higher the loss.

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to \infty$

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$ the loss $\to 0$

$$\boxed{L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)}\log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1-y^{(i)})\log(1-f_{\vec{w},b}(\vec{x}^{(i)}))}$$

simplified cost function

$$J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left[L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})\right]$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1-y^{(i)})\log(1-f_{\vec{w},b}(\vec{x}^{(i)}))\right]$$

Gradient descent for Logistic Regression

repeat {

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\vec{w},b)$$

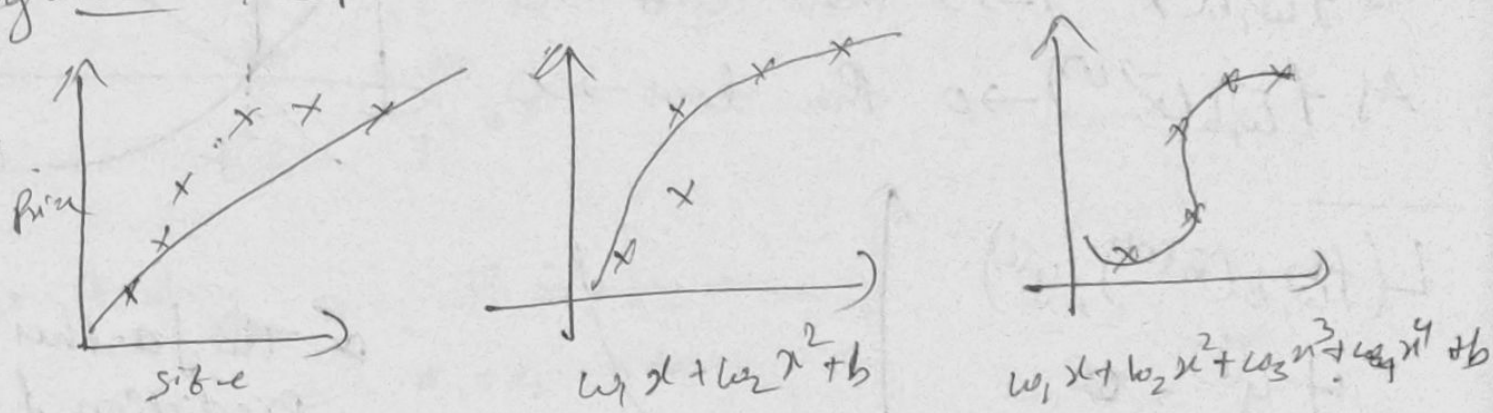
$$b_j = b - \alpha\frac{\partial}{\partial b}J(\vec{w},b)$$

} simultaneous update

$$\frac{\partial}{\partial w_j}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})x_j^{(i)}$$

$$\frac{\partial}{\partial b}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

Regression Example

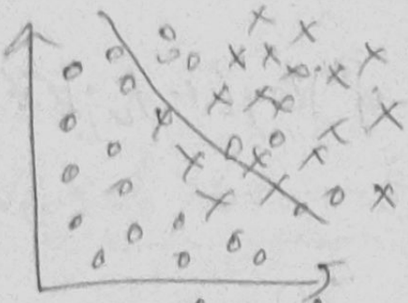


Underfit
high bias

Generalization

Overfit
high variance

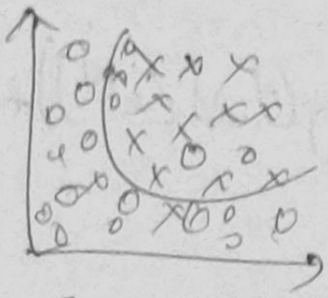
Classification example



Underfit

$z = w_1 x_1 + w_2 x_2 + b_2$

High Bias



$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 + b$

Just right

$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_3 + w_4 x_1^2 x_2^2 + w_5 x_1^2 x_2^3 + w_6 x_1^3 x_2 + \cdots + b$

Overfit

High Variance

Addressing Overfitting

Options

1. Collect more data
2. Select features
   - Feature selection
3. Reduce the size of parameters
   - "Regularization"

Regularization

To make a parameter to regularize we make changes to them in such a way that those parameters are close to 0

Cost function for regularized linear regression

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

Implementing gradient descent

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \Big]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

} Simultaneous update

Regularized Logistic Regression

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)}) \log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) \right]$$

$$+ \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

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

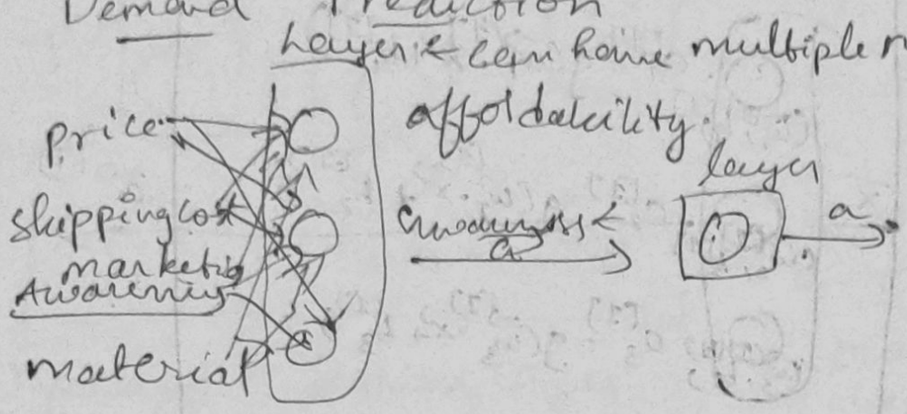
}

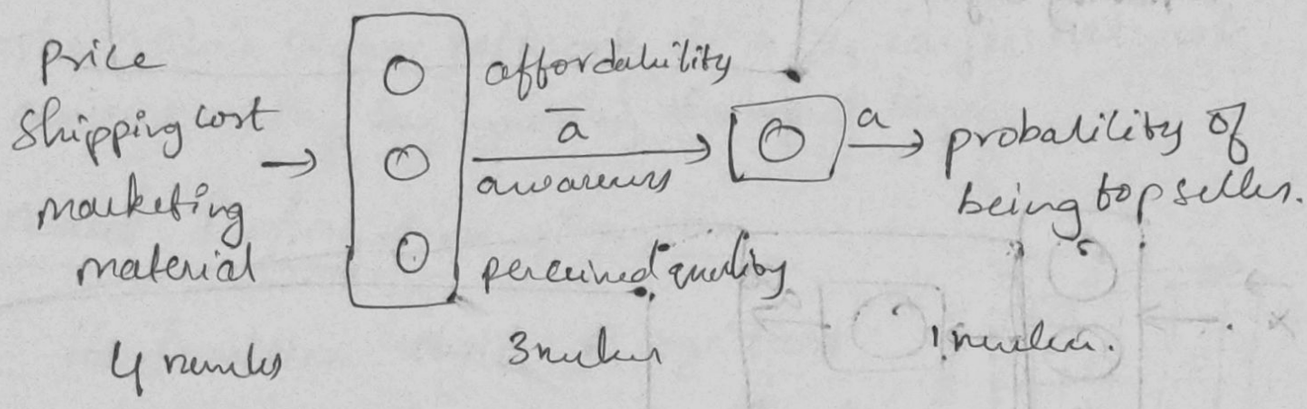
Course-2

Advanced Algorithms

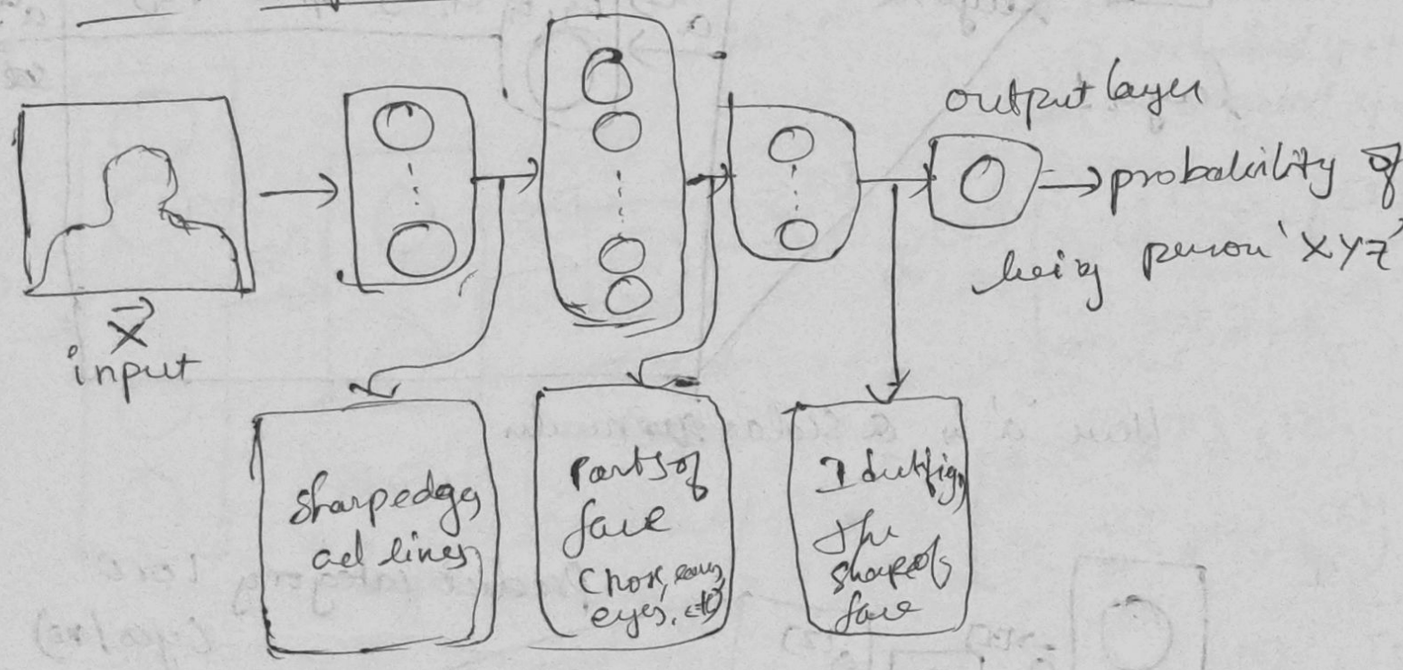
Neural Networks

Demand Prediction

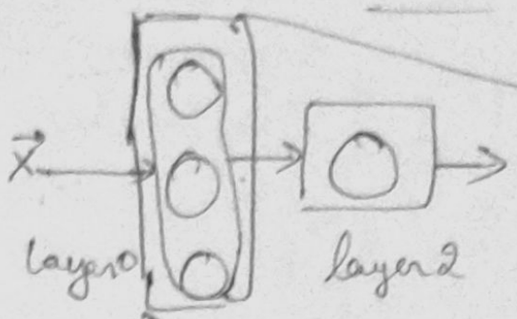




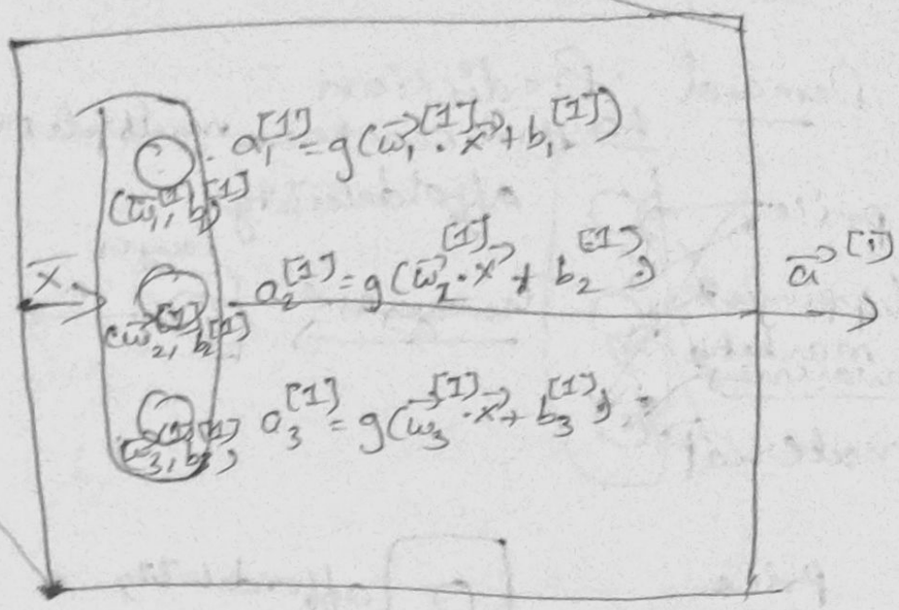
Face Recognition



Neural Network layer



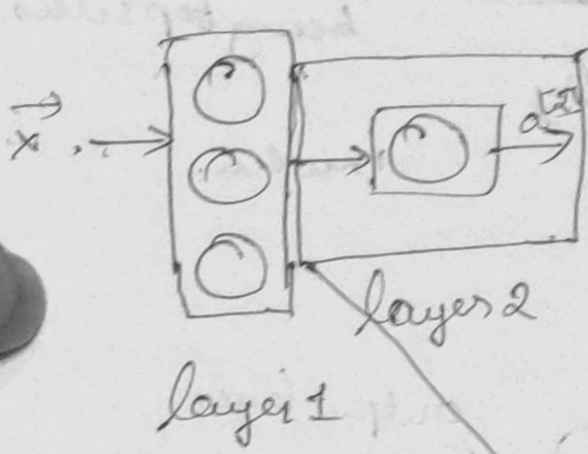
$$g(z) = \frac{1}{1+e^{-(z)}}$$

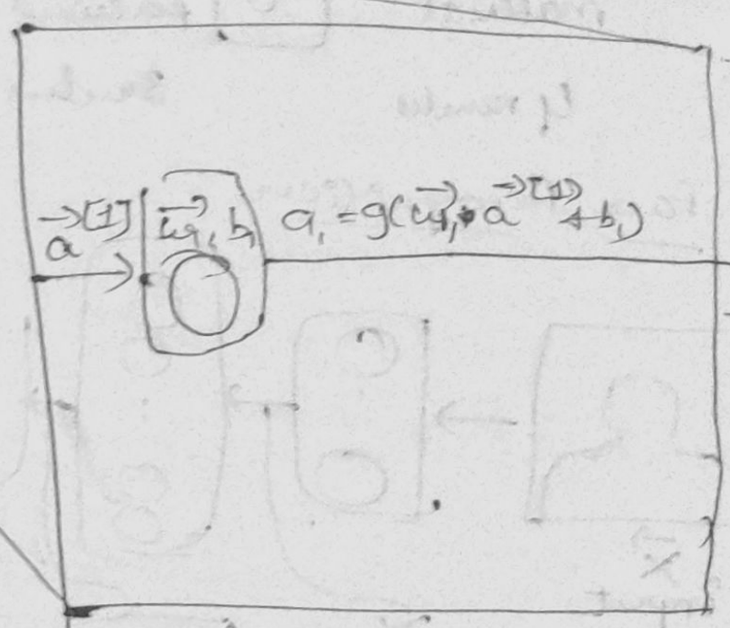


layer 1

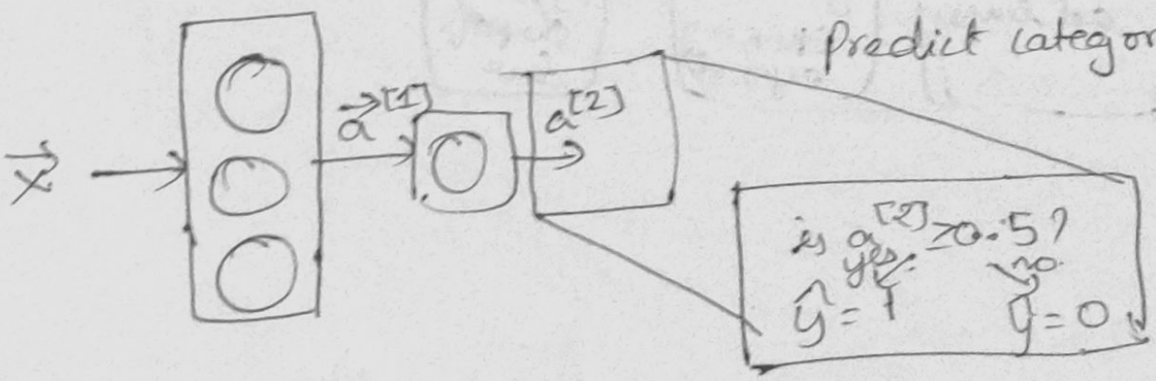
[1] [2]

notation for numbering layer



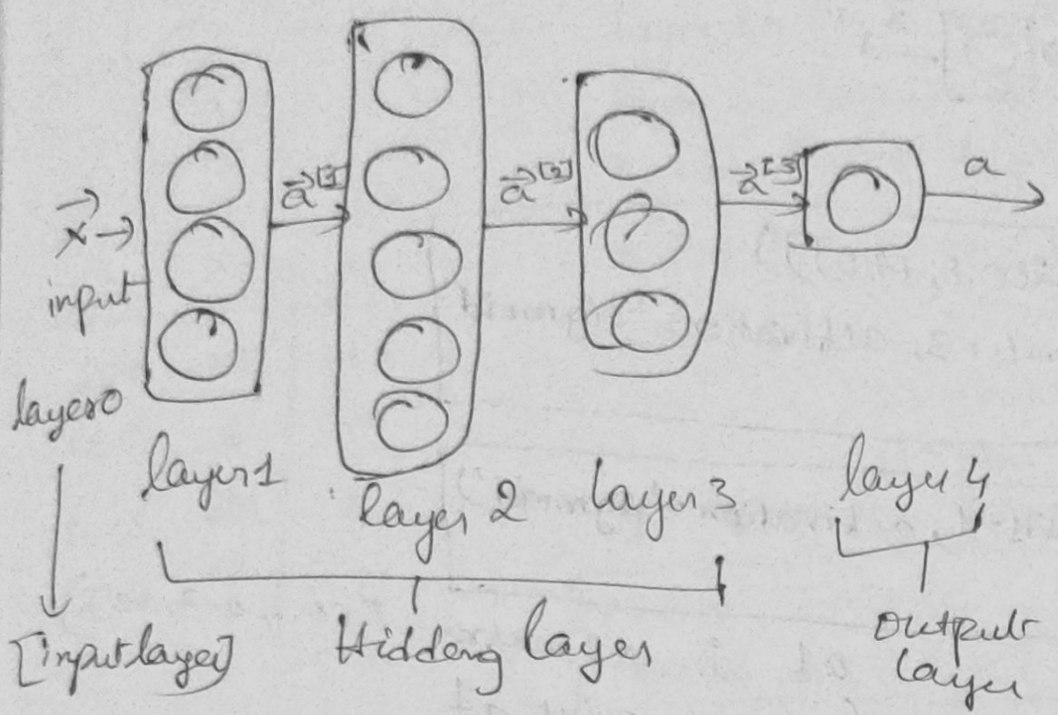


Here 'a' is a scalar ~~no~~ number



Predict category 1 or 0 (yes / no)

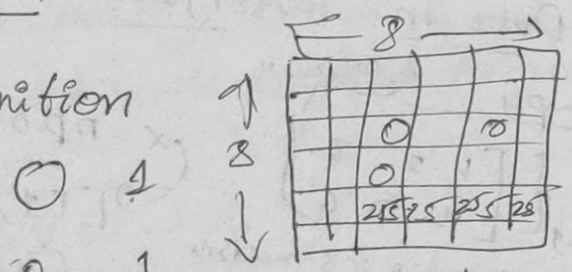
More complex Neural Network



* The taken neural network is a 4 layer network conventionally by ignoring the input layer

Forward Propagation Algorithm

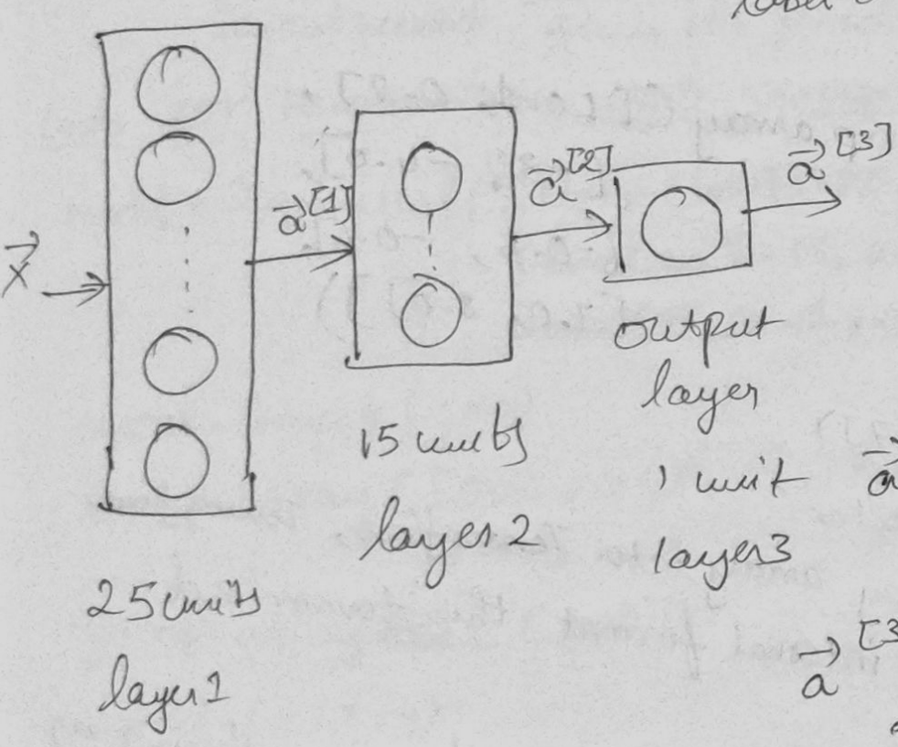
Ex: Handwritten digit recognition



0 → darkest spot

255 → brightest spot.

label 0 1



$$\vec{a}^{[1]} = \begin{bmatrix} g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]}) \\ \vdots \\ g(\vec{w}_{25}^{[1]} \cdot \vec{x} + b_{25}^{[1]}) \end{bmatrix}$$

$$\vec{a}^{[2]} = \begin{bmatrix} g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]}) \\ \vdots \\ g(\vec{w}_{15}^{[2]} \cdot \vec{a}^{[1]} + b_{15}^{[2]}) \end{bmatrix}$$

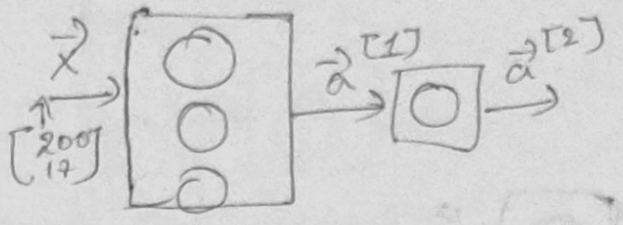
$$\vec{a}^{[3]} = \left[g(\vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]}) \right]$$

is $a^{[3]} \geq 0.5$?

yes → $\hat{y} = 1$ image is digit 1

no → $\hat{y} = 0$ image isn't digit 1.

Tensorflow implementation




```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

```
layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
```

```
if a2 >= 0.5:
    yhat = 1
else
    yhat = 0
```

a1 is 1x3 matrix [[0.2, 0.7, 0.3]]

↳ if you print a1

tf.Tensor([[0.2 0.7 0.3]], shape=(1,3), dtype=float32)

Data in Tensorflow

Numpy

3 columns

2 rows $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

2x3 matrix

```
x = np.array([[1, 2, 3], [4, 5, 6]])
```

↳ [[1, 2, 3],
[4, 5, 6]]

2

4 rows $\begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -.5 & -.6 \\ 7 & 8 \end{bmatrix}$

4x2

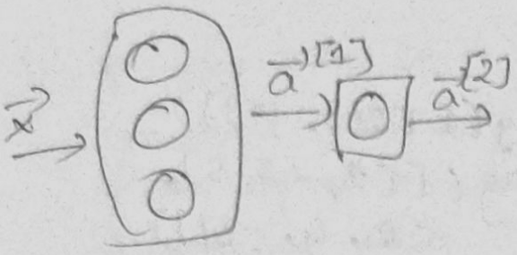
```
x = np.array([[0.1, 0.2],
              [-3.0, -4.0],
              [-0.5, -0.6],
              [7.0, 8.0]])
```

```
x = np.array([200, 17])
```

↳ x becomes a 1D 'vector'

* When you pass a numpy array into Tensorflow, Tensorflow converts it into its own internal format the 'tensor' and performs computations.
* when you want to read the data back out you can keep it as a tensor or convert it into a numpy array.

Building a neural network architecture




```
layer_1 = Dense(units=3, activation="sigmoid")
layer_2 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2])
```

x		y
200	17	1
120	5	0
425	20	0
212	18	1

```
X = np.array([[200.0, 17.0],
              [120.0,  5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
y = np.array([1, 0, 0, 1])
```

```
model.compile(...)
model.fit(X, y)
```

training the model

sequential() → It takes the given layers as input and outputs a neural network by sequentially layering ~~the~~ (string) the given layers.

model.fit() → It takes the given arrays as inputs and trains the given model with the data present in those arrays.

inference on neural network → `model.predict(x_new)` gives the prediction for new data.

Code for Handwritten digit recognition

```
model = Sequential([Dense(units=25, activation="sigmoid"),
                    Dense(units=15, activation="sigmoid"),
                    Dense(units=1, activation="sigmoid")])
model.compile(...)
x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])
y = np.array([1, 0])
model.fit(x, y)
model.predict(x_new)
```

For loops Vs. Vectorization

```
x = np.array([200, 17])
W = np.array([[1, -3, 5],
              [-2, 4, -6]])
b = np.array([-1, 1, 2])

def dense(a_in, W, b):
    units = W.shape[1]
    a_out = np.zeros(units)
    for j in range(units):
        w = W[:, j]
        z = np.dot(w, a_in) + b[j]
        a_out[j] = g(z)
    return a_out

[1, 0, 1]
```

```
X = np.array([[200, 17]])
W = np.array([[1, -3, 5],
              [2, 4, -6]])
B = np.array([[-1, 1, 2]])

def dense(A_in, W, B):
    Z = np.matmul(A_in, W) + B
    A_out = g(Z)
    return A_out

[[1, 0, 1]]
```

Train a Neural Network in Tensorflow

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(units=25, activation='sigmoid'),
    Dense(units=15, activation='sigmoid'),
    Dense(units=1, activation='sigmoid')])

from tensorflow.keras.losses import
BinaryCrossentropy
    model.compile(loss=BinaryCrossentropy())
    model.fit(X, y, epochs=100)
```

↳ no. of steps in gradient descent

Model Training Steps

① Specify how to compute output given input x and parameters w, b (define model)
$f_{\vec{w},b}(\vec{x}) = ?$

logistic regression

```
z = np.dot(w, x) + b
f_x = 1/(1 + np.exp(-z))
```

Neural Network

```
model = Sequential([
    Dense(---)
    Dense(---)
    Dense(---)
])
```

② Specify loss and cost

logistic loss

```
loss = -y * np.log(f_x)
       -(1-y) * np.log(1 - f_x)
```

binary crossentropy

```
model.compile(
    loss = BinaryCrossentropy()
```

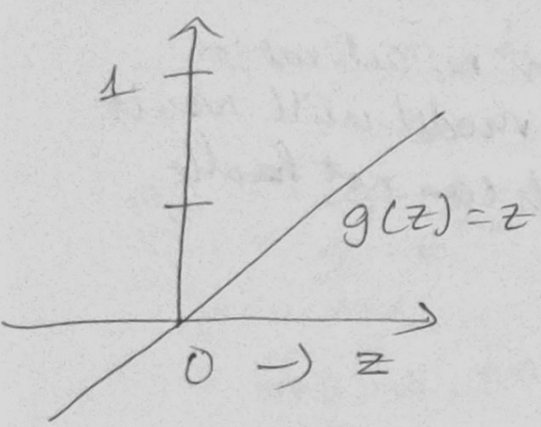
③ Train on data to minimize $J(\vec{w}, b)$

$w = w - alpha * dj_dw$
$b = b - alpha * dj_db$

```
model.fit(
    X, y, epoch=100)
```

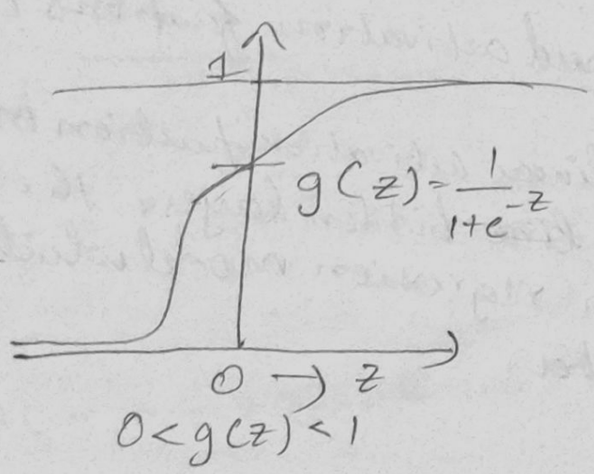
Examples of Activation functions

linear Activation function

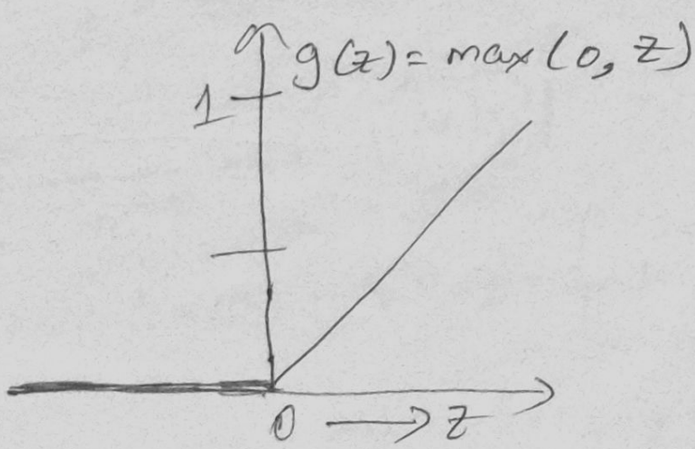


$a = g(z) = \underbrace{\vec{w} \cdot \vec{x} + b}_{z}$

Sigmoid function



ReLU Rectified linear Unit



Activation function for Output layer

Binary Classification ~ Sigmoid

activation='sigmoid'

$y = 0/1$

x_1 x_2	y
	0
	1
	0
	1
	0

Regression - Linear activation function

$y = +/-$

activation='linear'

x_1 x_2	y
	-1.2
	+0.5
	-0.4

Regression - ReLU

$y = 0$ or $+$

activation='relu'

x_1 x_2	y
	+150
	+250
	+230

Activation function for Hidden layers

Best choice is to use ReLU

activation='relu'

Why do we need activation functions?

* If we use linear activation function or no activation function in ~~the~~ hidden layers the model will result in a linear regression model which can not handle complex data.

Multiclass Classification

Multiclass classification problem:
target y can take on more than two possible values.

Softmax Regression (4 possible outputs) $y = 1, 2, 3, 4$

× $z_1 = \vec{w}_1 \cdot \vec{x} + b_1$ $\quad a_1 = \dfrac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=1|\vec{x})$

○ $z_2 = \vec{w}_2 \cdot \vec{x} + b_2$ $\quad a_2 = \dfrac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=2|\vec{x})$

□ $z_3 = \vec{w}_3 \cdot \vec{x} + b_3$ $\quad a_3 = \dfrac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=3|\vec{x})$

△ $z_4 = \vec{w}_4 \cdot \vec{x} + b_4$ $\quad a_4 = \dfrac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=4|\vec{x})$

Softmax Regression ($y = 1, 2, 3, \ldots N$)
(N possible outputs)

$$z_j = \vec{w}_j \cdot \vec{x} + b_j \qquad j = 1, 2, \ldots, N$$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y=j|\vec{x})$$

Note: $a_1 + a_2 + \cdots + a_N = 1$

Cost for Softmax Regression

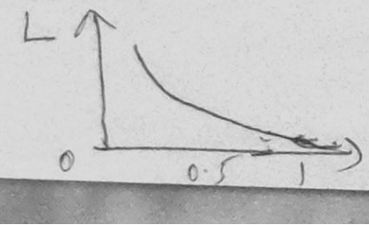
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=1|\vec{x})$$

$$\vdots$$

$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=N|\vec{x})$$

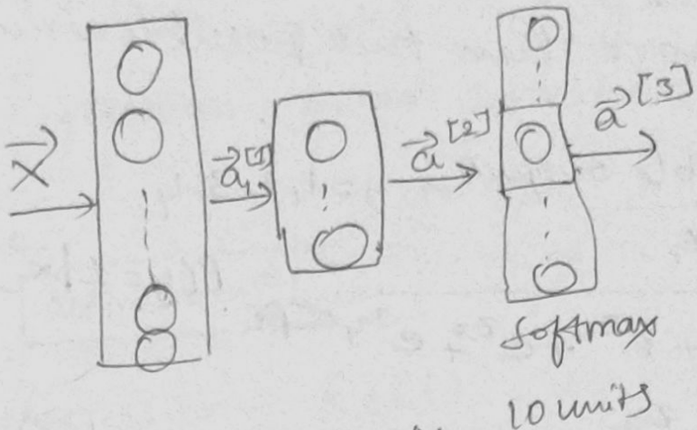
Crossentropy loss

$$\text{Loss}(a_1, \ldots a_N, y) = \begin{cases} -\log a_1 & \text{if } y = 1 \\ -\log a_2 & \text{if } y = 2 \\ \vdots & \\ -\log a_N & \text{if } y = N \end{cases}$$



$$\text{loss} = -\log a_j \quad \text{if } y = j$$

Neural Network with Softmax output



$$z_1^{[3]} = \vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]} = \frac{e^{z_1^{[3]}}}{e^{z_1^{[3]}} + \dots + e^{z_{10}^{[3]}}}$$

$$= P(y=1|\vec{x})$$

$$\vdots$$

$$z_{10}^{[3]} = \vec{w}_{10}^{[3]} \cdot \vec{a}^{[2]} + b_{10}^{[3]} = \frac{e^{z_{10}^{[3]}}}{e^{z_1^{[3]}} + \dots + e^{z_{10}^{[3]}}}$$

$$= P(y=10|\vec{x})$$

logistic regression

$$a_1^{[3]} = g(z_1^{[3]}) \quad a_2^{[3]} = g(z_2^{(3)})$$

softmax

$$\vec{a}^{[3]} = (a_1^{[3]}, \dots a_{10}^{[3]}) = g(z_1^{[3]}, \dots, z_{10}^{[3]})$$

MNIST with Softmax

① Specify the model

```
import tensorflow as tf
from tf.keras import Sequential
from tf.keras.layers import Dense
model = Sequential([
    Dense(units = 25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=10, activation='softmax')
])
```

② Specify loss and cost

```
from tf.keras.losses import
    SparseCategoricalCrossentropy
model.compile(loss=SparseCategoricalCrossentropy())
```

③ Train on data to minimise cost

```
model.fit(X, Y, epochs=100)
```

Note: Don't use this version of code.

More numerically accurate implementation of softmax

Softmax regression

$(a_1, \ldots, a_{10}) = g(z_1, \ldots, z_{10})$

$$\text{Loss} = L(\vec{a}, y) = \begin{cases} -\log a_1 & \text{if } y = 1 \\ \vdots & \\ -\log a_{10} & \text{if } y = 10 \end{cases}$$

More accurate

$$L(\vec{a}, y) = \begin{cases} -\log y \dfrac{e^{z_1}}{e^{z_1} + \cdots + e^{z_{10}}} & \text{if } y = 1 \\ \vdots & \\ -\log y \dfrac{e^{z_{10}}}{e^{z_1} + \cdots + e^{z_{10}}} & \text{if } y = 10 \end{cases}$$

```
model = Sequential [[
    Dense (units=25, activation='relu'),
    Dense (units=15, activation='relu'),
    Dense (units=10, activation='softmax')
    ])

model.compile (loss=SparseCategoricalCrossEntropy())
```

Adam Algorithm Intuition

Adam: Adaptive Moment estimation (not just one α)

$$w_1 = w_1 - \alpha_1 \frac{\partial}{\partial w_1} J(\vec{w}, b)$$

$$\vdots$$

$$w_{10} = w_{10} - \alpha_{10} \frac{\partial}{\partial w_{10}} J(\vec{w}, b)$$

* Adam can adjust α automatically.

→ If w_j (or b) keeps moving in same direction, increase α_j.

→ If w_j (or b) keeps oscillating reduce α_j.

MNIST ADAM

model

```
model = Sequential([
    tf.keras.layers.Dense(units=25, activation='sigmoid'),
    tf.keras.layers.Dense(units=15, activation='sigmoid'),
    tf.keras.layers.Dense(units=10, activation='linear')
])
```

Compile

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

$\alpha = 10^{-3} = 0.001$

fit

```
model.fit(X, Y, epochs=100)
```

11/7/24

Evaluating your Model

Dataset:

size	price
2104	400
1600	330
⋮	⋮
1427	199
1380	212
⋮	⋮
1494	243

70% — training set: m_{train} = no. training ex

$(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, ⋮, $(x^{(m_{train})}, y^{(m_{train})})$

30% — test set: m_{test} = no. test examples

$(x_{test}^{(1)}, y_{test}^{(1)})$, ⋮, $(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

(3) **Train | test procedure for linear regression** (with squared error cost)

Fit parameters by minimizing cost function $J(\vec{w}, b)$

$$J(\vec{w}, b) = \left[\frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2\right] + \frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2$$

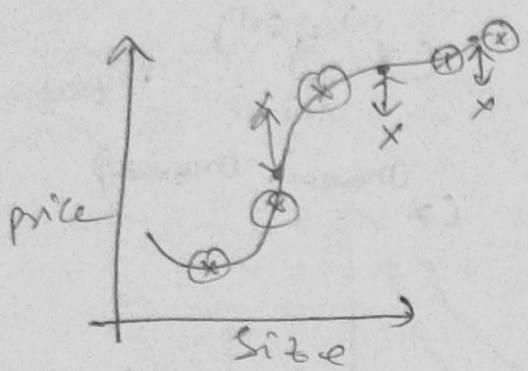
regularization term

Compute test error:

$$J_{test}(\vec{w},b) = \frac{1}{2m_{test}}\left[\sum_{i=1}^{m_{test}}\left(f_{\vec{w},b}(\vec{x}^{(i)}_{test}) - y^{(i)}_{test}\right)^2\right]$$

Compute train error:

$$J_{train}(\vec{w},b) = \frac{1}{2m_{train}}\left[\sum_{i=1}^{m_{train}}\left(f_{\vec{w},b}(\vec{x}^{(i)}_{train}) - y^{(i)}_{train}\right)^2\right]$$



⊗ = train

× = test

$J_{train}(\vec{w},b)$ will be low

* If $J_{test}(\vec{w},b)$ is high, it means that the learning algorithm is not able to generalize new data points that aren't present in the training set.

<u>Train/test</u> <u>procedure</u> for <u>classification</u> <u>problem</u>

→ Fit parameters by minimizing $J(\vec{w},b)$ to find $\vec{w},b$
→ computing training & test errors using logistic loss funt^n } → isn't common for classification problem

* This works okay but there is a better way to do the evaluation.

Count fraction of the test set and the fraction of the test set that the algorithm has misclassified.

$$\hat{y} = \begin{cases} 1 \text{ if } f_{\vec{w},b}(\vec{x}^{(i)}) \geq 0.5 \\ 0 \text{ if } f_{\vec{w},b}(\vec{x}^{(i)}) < 0.5 \end{cases}$$

Count $\hat{y} \neq y$

$J_{test}(\vec{w},b)$ is the fraction of the test set that has been misclassified

$J_{train}(\vec{w},b)$ is the fraction of the train set that has been misclassified.

J_{train} will usually be low.

* If J_{test} is high then the model is not able to do correct categorization, i.e, algorithm isn't working for new data points.

Model Selection

Training / Cross validation / test set

(validation set)
(development set)
(dev set)

Size	price		
2104	400	training set 60%	$(x^{(1)}, y^{(1)})$ ⋮ $(x^{(m_{train})}, y^{(m_{train})})$ m_{train}
1600	330		
2400	369		
1416	232		
3000	540		
1985	300	Cross validation 20%	$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$ m_{cv}
1534	315		
1427	199		
1380	212	test set 20%	$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$ m_{test}
1494	243		

Cross validation error:

$$J_{cv}(\vec{w}, b) = \frac{1}{2m_{cv}} \left[\sum_{i=1}^{m_{cv}} \left(f_{\vec{w},b}(\vec{x}_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2 \right]$$

(validation / dev error)

$d=1$ 1. $f_{\vec{w},b}(\vec{x}) = w_1 x + b$

$d=2$ 2. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + b$

$d=3$ 3. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + w_3 x^3 + b$

⋮

$d=10$ 10. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + \dots + w_{10} x^{10} + b$

Pick $w_1 x + \dots + w_4 x^4 + b$ $(J_{cv}(w^{<4>}, b^{<4>}))$

*Model with lowest dev error should be selected

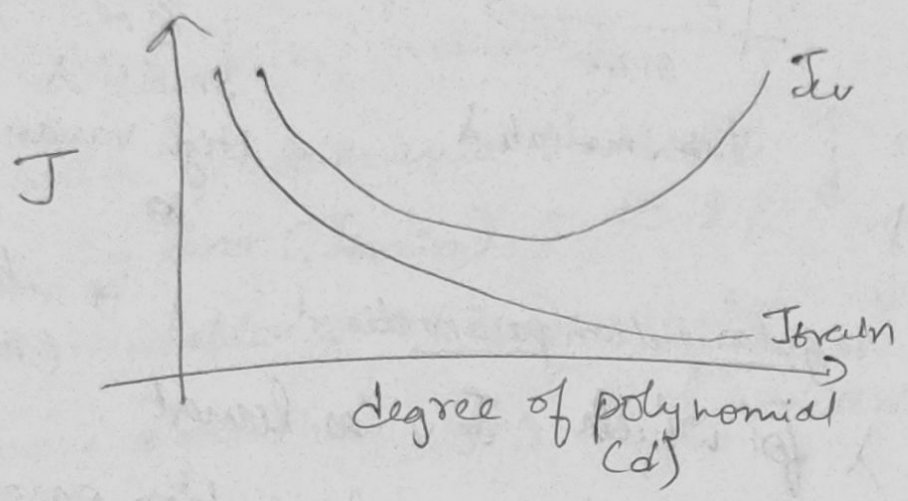
Estimate generalization error using test set: $J_{test}(w^{<4>}, b^{<4>})$

*Reason for using dev set: Since the models have been trained on a subset of the dataset, ie, training set, ~~it is further~~ dev set would be helpful to determine which parameters (model) are best. Then the test set could estimate a fair generalize

on new data points not present in either ~~test~~ set & train set.

Diagnosing Bias/Variance

high Bias (underfitting)	"Just Right"	high Variance (overfitting)
J_{train} is high	J_{train} is low	J_{train} is low
J_{cv} is high	J_{cv} is low	J_{cv} is high



High bias (underfit)
J_{train} will be high
($J_{train} \approx J_{cv}$)

High variance (Overfit)
$J_{cv} >> J_{train}$
(J_{train} may be low)

High bias & high variance
J_{train} will be high
& $J_{cv} >> J_{train}$

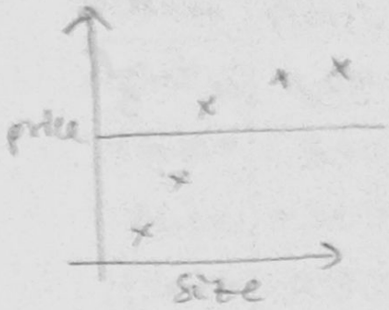
* If ~~J_{train}~~ there is high Bias it means model isn't doing well ~~good~~ on training set & High variance means it does much worse on the cross-validation set.

Linear Regression with Regularization

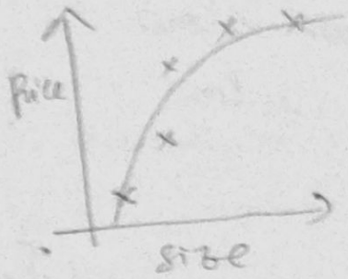
Model: $f_{\vec{w},b} = w_1 x_1 + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$

$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

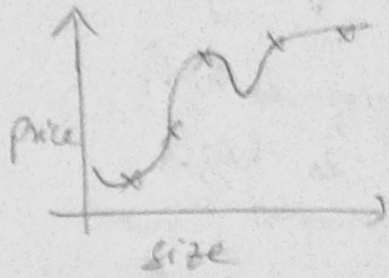
Regularization term



large λ
High bias (underfit)



Intermediate λ



small λ
High Variance (overfit)

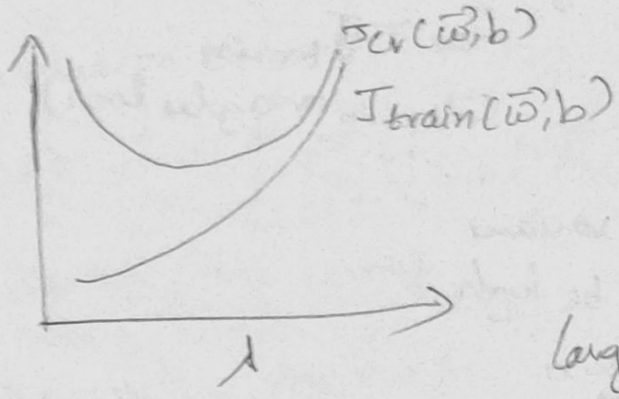
* underfitting (high bias)

Choosing the regularization parameter λ

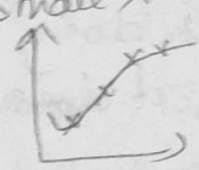
→ choose the λ for which J_{cv} is lowest

Bias Variance as a funct^n of regularization parameter λ

* Overfitting (high variance)

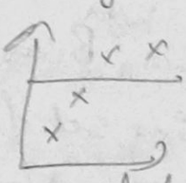


small λ



overfitting

large λ



underfitting

Establishing baseline level of performance

↳ what is the level of error you can reasonably hope to get

- Human level performance
- competing algorithms performance with well performed algorithms.
- Guess based on experience

Recognising the problem with algorithms

(high bias/variance)

Ex1: Baseline performance : 10.6% ↕ 0.2%

Training error (J_{train}) : 10.8% ↕ 4.0%

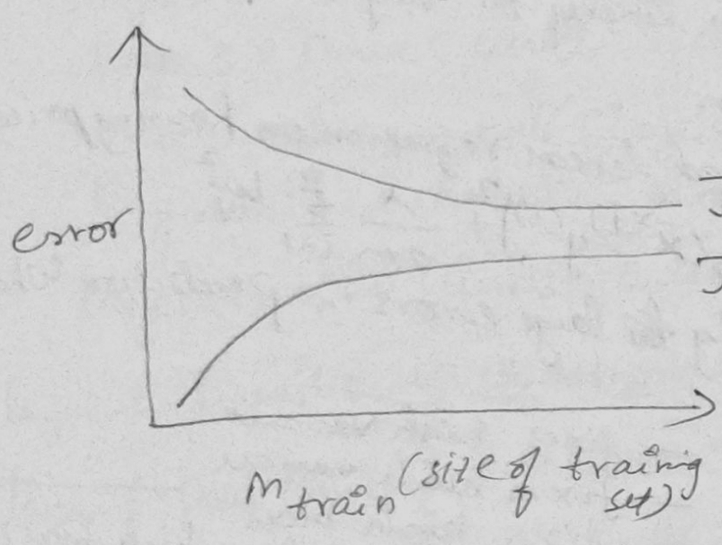
Cross validation error (J_{cv}): 14.8%

The above algorithm has high variance problem

* If the differe b/w baseline performance and training error is high then there is a high bias problem
* If the differe b/w Training error and cross validation set is high then there is a high variance problem.

Learning Curves

$f_{w,b}(x) = w_1 x + w_2 x^2 + b$

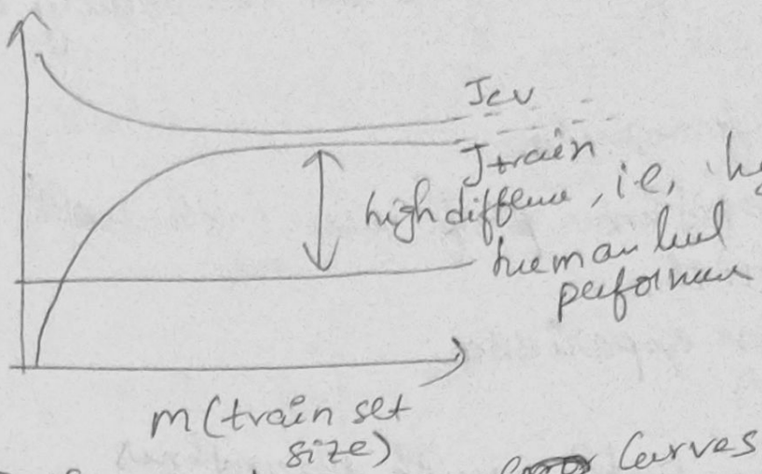


* J_{cv} decreases as data set size increases
* J_{train} increases as data set size increases as it become hard to fit more data using the model

High bias

$f_{\vec{w},b}(\vec{x}) = w_1 x + b$

(underfitting)

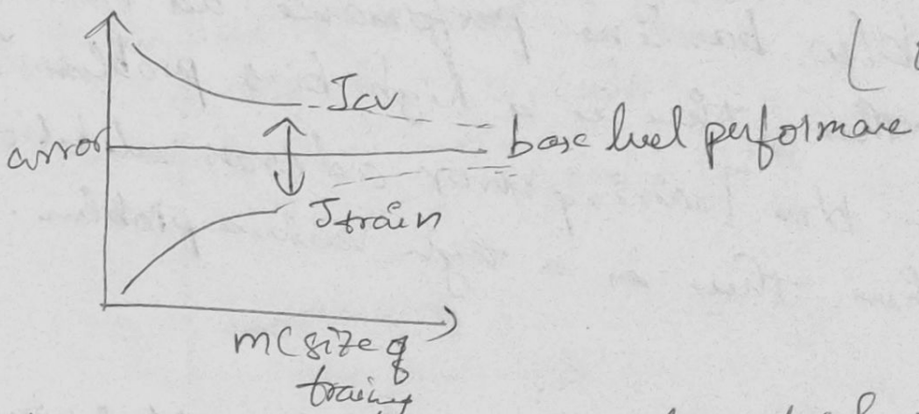


* Both Jcv & Jtrain curves get flattened because after a while as the no. of examples are high the fitting of the straight line doesn't change much.
* If a learning algo suffers with high bias, getting more training data will not (by itself) will not help much.

High Variance

$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$
(with small λ)

(overfitting)



* If a learning algo suffers from high variance, getting more training data is likely to help.

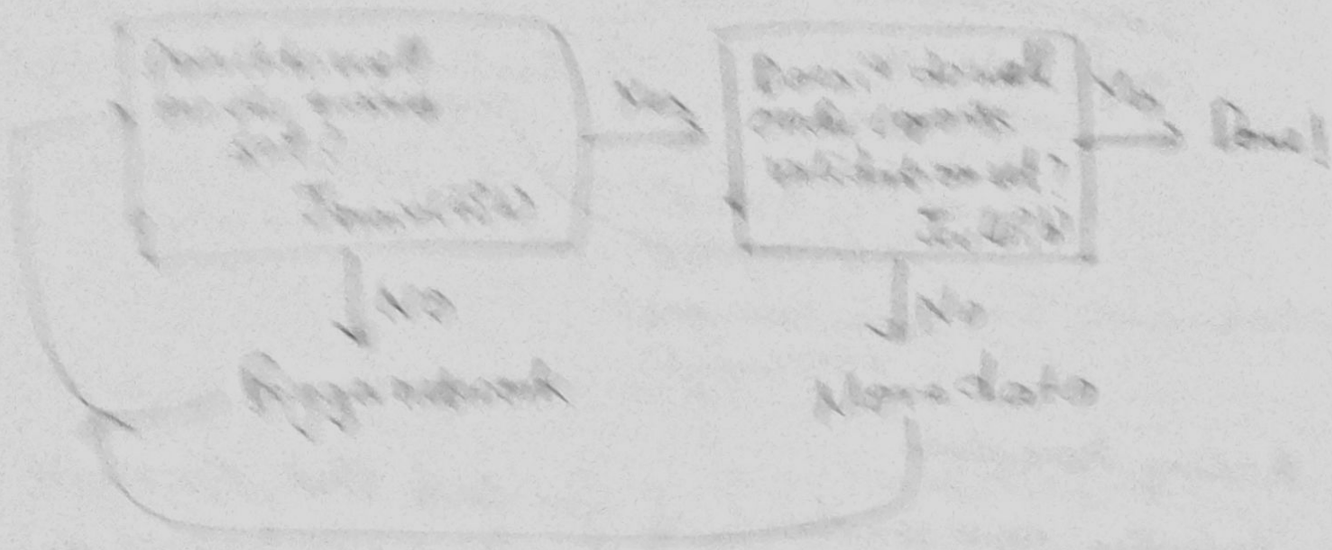
Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{i=1}^{m} w_j^2$$

But it makes unacceptably large errors in prediction. What do you try next

- Get more training examples — fixes high variance
- Try smaller sets of features — fixes high variance
- Try getting additional features — fixes high bias
- Try adding polynomial features (x_1^2, x_2^2, $x_1 x_2$, etc) — fixes high bias
- Try decreasing λ — fixes high bias
- Try increasing λ — fixes high variance.

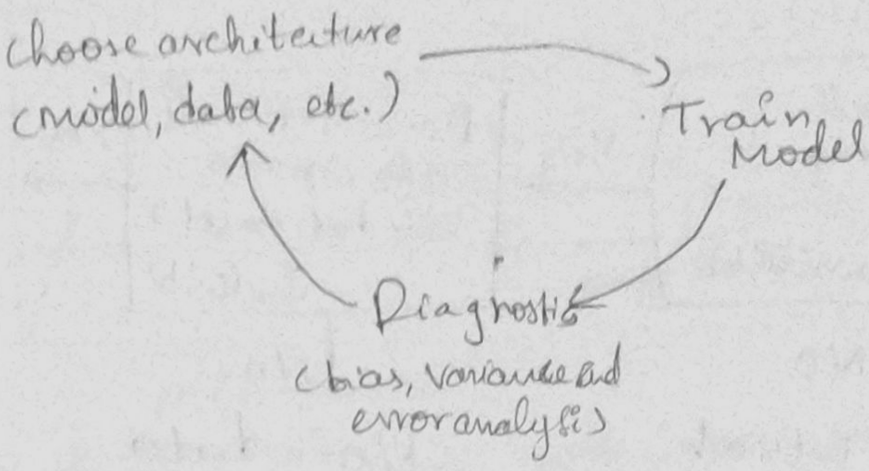


Neural Networks and Regularization

- A large neural network will usually difficult or less than a smaller one so long as regularization is chosen appropriately.
- (The disadvantage of using large networks is that it increases the cost of computing and prone to the problem of overfitting and it should choose the training)

```
Regularized MNIST Model
model
layer_1 = Dense(units=30, activation="relu")
layer_2 = Dense(units=30, activation="relu")
layer_3 = Dense(units=10, activation="softmax")
model = Sequential([layer_1, layer_2, layer_3])
Regularized MNIST model
layer_1 = Dense(units=30, activation="relu", kernel_regularizer
                                    = L2(0.01))
layer_2 = Dense(units=30, activation="relu", kernel_regularizer=L2(0.01))
layer_3 = Dense(units=10, activation="softmax", kernel_regularizer=L2(0.01))
model = Sequential([layer_1, layer_2, layer_3])
```

Iterative loop of ML development

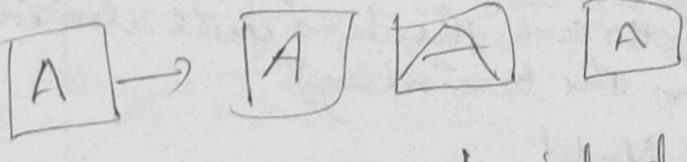


Adding More data

↳ During error analysis if you find that the model is performing poorly on a particular subset of the data, then adding just a few more examples of such kind will boost the model's performance, rather than adding more new examples for the total dataset.

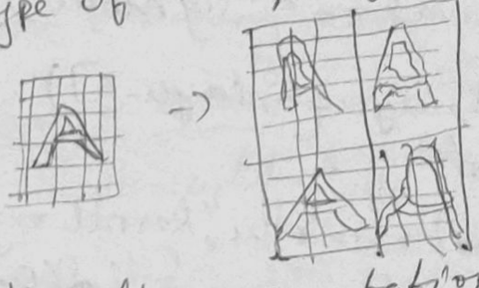
Data augmentation

Augmentation is modifying an existing training example to create a new training example.



Data augmentation by introducing distortions

Distortion introduced should be representation of the type of noise/distortions in the test set.



[Audio: Background noise, bad cellphone connection

* Usually augmentation does not help to add purely random/meaningless noise to your data.

Data Synthesis

Synthesis: using artificial data inputs to create a new training example.

Engineering the data used by your system

Conventional Model-centric approach: AI = (Code) + Data

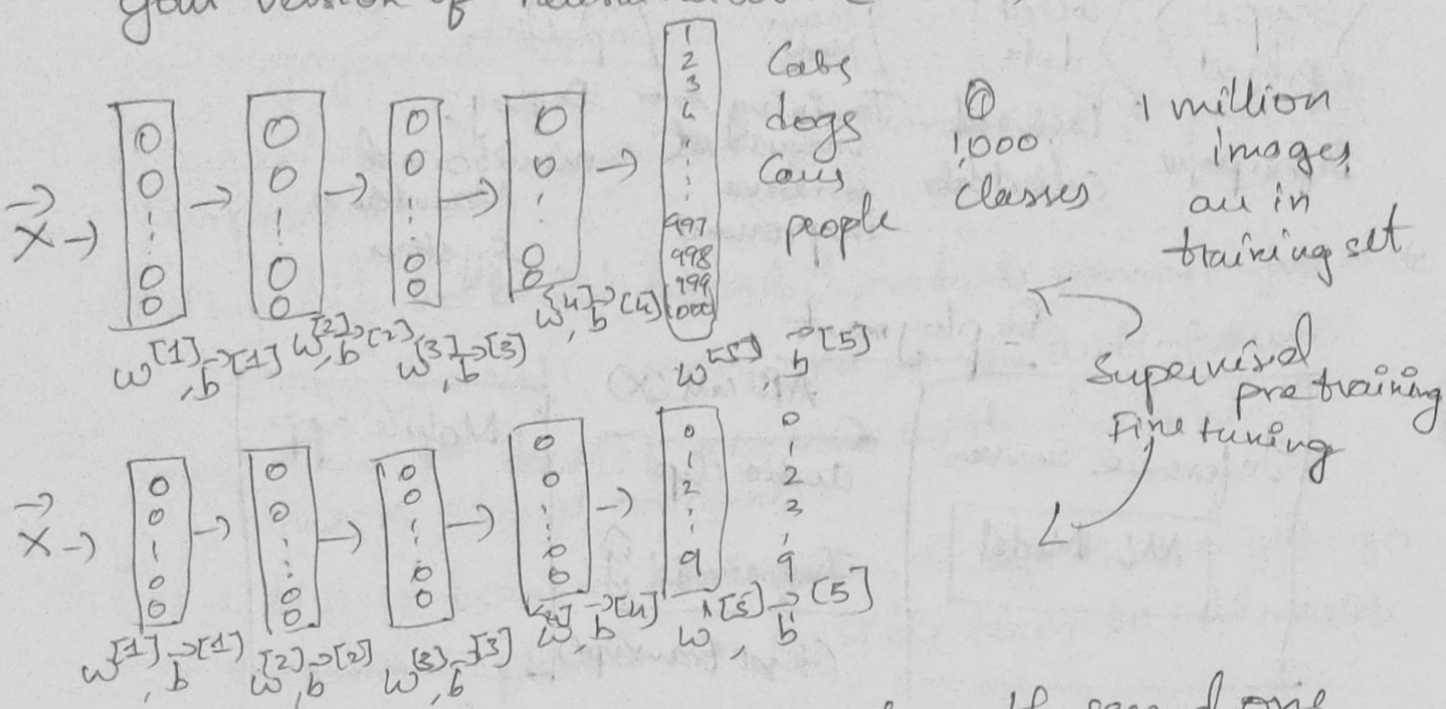
earlier concentration

Data-Centric Approach : AI = Code + (Data)

present concentration

Transfer learning

It is using the power of a large model that is being trained on a huge amount of data to fine tune your version of neural network (model).



In the above two neural networks, the second one has the same first four layers of the first neural net except the output layer, i.e, the parameter found while training the first model are used in the second model except the output layer's parameters. This will reduce the time for training and increases efficiency.

* By using transfer learning you can use one model's power in an application far different from the one that model has been made for.

Options to implement transfer learning

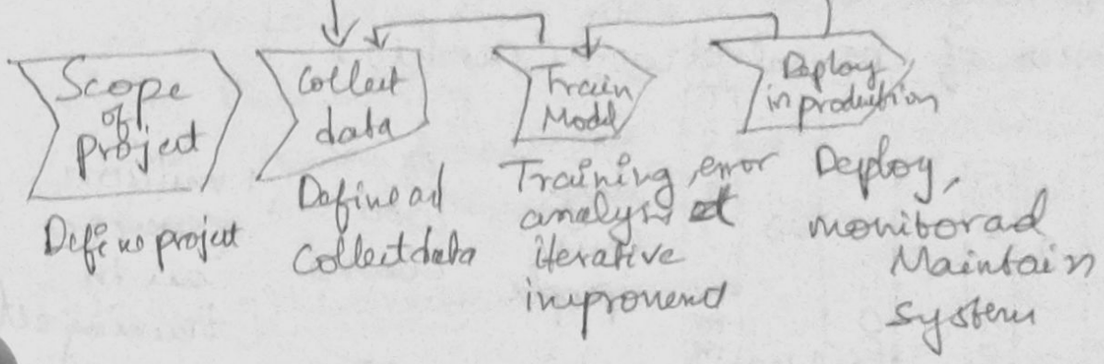
Option 1: only train output layer's parameters

Option 2: train all parameters (start off with the one's found during first Model's training)

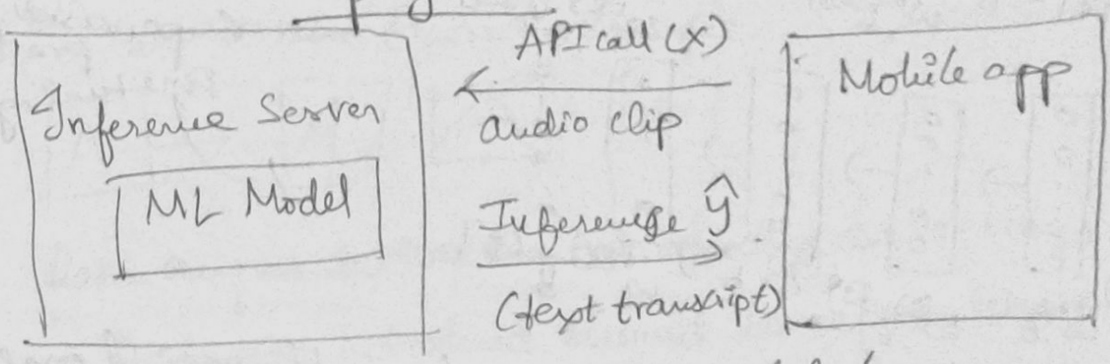
Transfer learning Summary

1. Download neural network parameters pretrained on a large dataset with same input type (e.g., images, audio, text) as your application (or train your own).
2. Further train (fine tune) the network on your own data.

Full cycle of a Machine learning Project



Deployment



→ Software Engineering may be needed for:

Ensure reliable and efficient predictions
Scaling
Logging
System Monitoring
Model updates

MLOps

ML operations

Ethics Bias

Historical examples:

Hiring tool that discriminates against women

Facial recognition system matching dark skinned individuals to criminal Mugshots.

Biased bank loan approvals

Toxic effect of reinforcing negative stereotypes

Adverse Use Cases

Deepfakes

Spreading toxic/incendiary speech through optimizing for engagement

Generating fake content for commercial or political purposes.

Using ML to build harmful products, commit fraud etc.

Spam vs anti-spam: fraud vs. anti-fraud.

Guidelines to develop systems with less bias and more ethical concerns

→ Get a diverse team to brainstorm things that might go wrong, with emphasis on possible harm to vulnerable groups.

→ Carry out literature search on standards/guidelines for your industry.

→ Audit systems against possible harm prior to deployment.

→ Develop mitigation plan (if applicable), and after deployment monitor for possible harm.

Precision/recall

$y = 1$ in presence of rare class we want to detect

		Actual class 1	Actual class 0
Predicted class	1	True +ves 15	False -ves 5
	0	False -ves 10	True -ves 10

Precision:
(of all patients where we predicted $y = 1$, what fraction actually have the rare disease?)

$$\frac{\text{True +ves}}{\text{\# predicted +ve}} = \frac{\text{True +ves}}{\text{True +ve + False +ve}}$$

$$= \frac{15}{15+5} = \frac{15}{20} = 0.75$$

Recall:
(of all patients that actually have rare disease, what fraction did we correctly detect as having it?)

$$\frac{\text{True +ves}}{\text{\# actual +ve}} = \frac{\text{True +ves}}{\text{True +ve + False -ve}} = \frac{15}{15+10} = \frac{15}{25} = 0.6$$

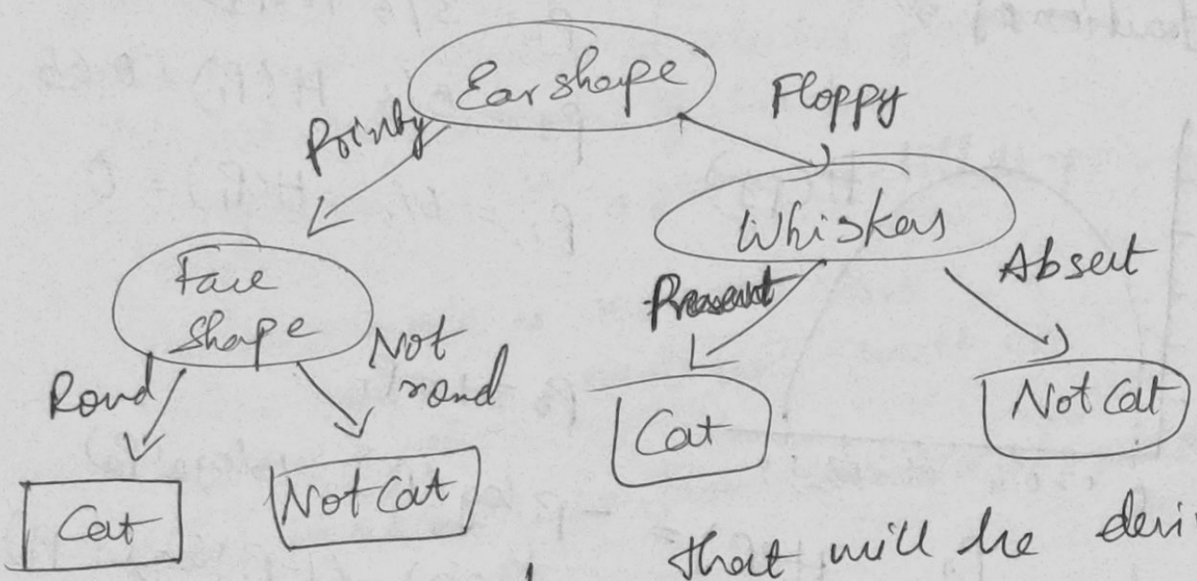
* If an algo has zero precision and zero recall then it is considered useless.

Decision Trees

Cat classification Example

Ear shape (x_1)	Face shape (x_2)	Whiskers	Cat
Pointy	Round	Present	1
Floppy	Not Round	Present	1
Floppy	Round	Absent	0
Pointy	Not Round	Present	0
Pointy	Round	Present	1
Pointy	Round	Absent	1
Floppy	Not round	Absent	0
Pointy	Round	Absent	1
Floppy	Round	Absent	0
Floppy	Round	Absent	0

x y



This the decision tree that will be derived by training the model with the above dataset.

Decision Tree learning

Decision 1: How to choose what feature to split on at each node?

Maximize Purity (or minimize impurity)

Decision 2: When do you stop splitting?

- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth.
- Reasons to keep limit to depth of tree
  - ↳ To make sure it doesn't gets too big a dum mode.
  - ↳ There will be less chances of overfitting.
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold.

Entropy as a measure of impurity

p_1 = fraction of ex that are cats

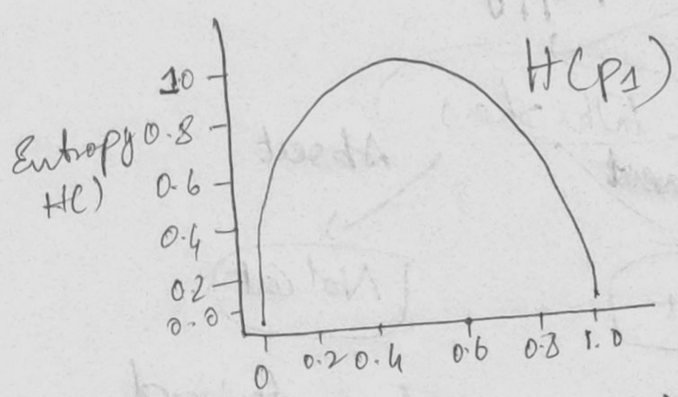
$P_1 = 0 \quad H(P_1) = 0$

$P_1 = 1/6 \quad H(P_1) = 0.92$

$P_1 = 3/6 \quad H(P_1) = 1$

$P_1 = 5/6 \quad H(P_1) = 0.65$

$P_1 = 6/6 \quad H(P_1) = 0$



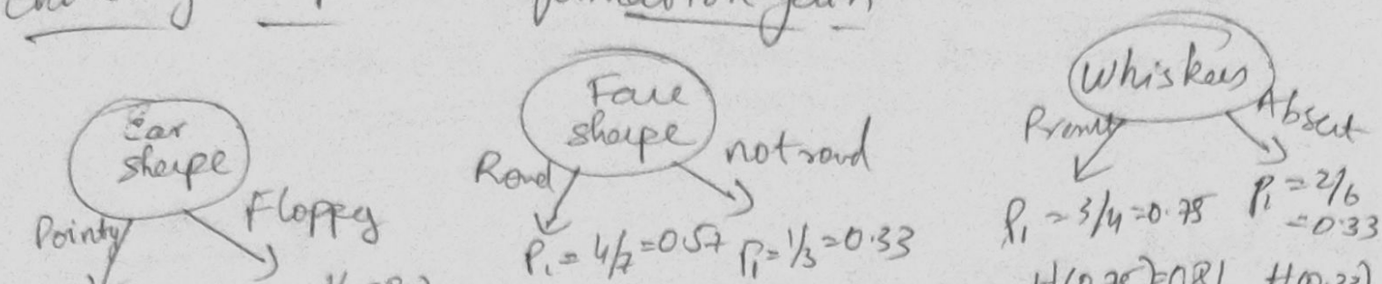
$P_0 = 1 - P_1$

$$H(P_1) = -P_1 \log(P_1) - P_0 \log_2(P_0)$$

$$= -P_1 \log_2(P_1) - (1-P_1)\log_2(1-P_1)$$

$$\boxed{\text{"}0\log(0)\text{"} = 0}$$

Choosing a split: Information gain



$P_1 = 4/5 = 0.8$ $P_1 = 1/5 = 0.2$

$H(0.8) = 0.72$ $H(0.2) = 0.72$

$P_1 = 4/7 = 0.57$ $P_1 = 1/3 = 0.33$

$H(0.57) = 0.99$ $H(0.33) = 0.92$

$P_1 = 3/4 = 0.75$ $P_1 = 2/6 = 0.33$

$H(0.75) = 0.81$ $H(0.33) = 0.92$

* ~~Data set~~ Model with more entropy with more no. of examples is worse than ~~dataset with~~ Model with high entropy and less examples..

$$H(0.5) - \left(\frac{5}{10} H(0.8) + \frac{5}{10} H(0.2)\right) = 0.28$$

$$H(0.5) - \left(\frac{7}{10} H(0.57) + \frac{3}{10} H(0.33)\right) = 0.03$$

$$H(0.5) - \left(\frac{4}{10} H(0.75) + \frac{6}{10} H(0.33)\right) = 0.12$$

Information gain

- Start with all examples at the root node.
- Calculate information gain for all possible features, and pick the one with the highest information gain.
- Split dataset according to selected feature, and create left and right branches of the tree.
- Keep repeating splitting process untill stopping criteria is met:
  - When a node is 100% one class
  - When splitting a node will result in the tree exceeding a max depth
  - Information gain from additional splits is less than threshold.
  - When number of examples in a node below a threshold.