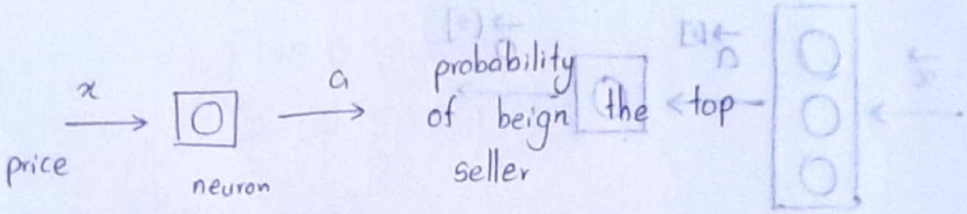
Advanced Learning Algorithms

Demand Prediction

x : price — input

$$a = f(x) = \frac{1}{1+e^{-(wx+b)}}$$ — output

a: activation



Neural network layer

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

$\vec{w}_1^{[1]}, b_1^{[1]}$

$a_1 = g(\vec{w}_1 \cdot \vec{x} + b_1) = 0.3$

$\vec{x}$

$\begin{bmatrix} 197 \\ 184 \\ 136 \\ 214 \end{bmatrix}$ →

$w_2^{[1]}, b_2^{[1]}$

$a_2 = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}) = 0.7$ → $\vec{a}^{[1]} = \begin{bmatrix} 0.3 \\ 0.7 \\ 0.2 \end{bmatrix}$

$w_3^{[1]}, b_3^{[1]}$

$a_3 = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]}) = 0.2$

$\vec{a}^{[1]}$ → $\vec{w}_1^{[2]}, b_1^{[2]}$

$a_1 = g(\vec{w}_1 \cdot \vec{a}^{[1]} + b_1)$

$\begin{bmatrix} 0.3 \\ 0.7 \\ 0.2 \end{bmatrix}$ → $a^{[2]} = 0.84$

if $[a^{[2]}] \geq 0.5$

yes → $\hat{y} = 1$

no → $\hat{y} = 0$

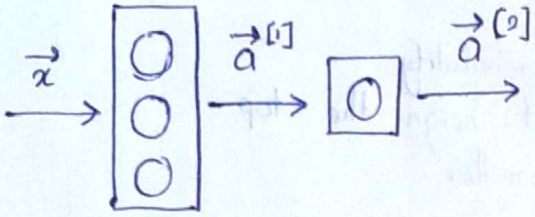
$$a_j^{[l]} = g\left(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]}\right)$$

layer $\rightarrow l$

unit $\rightarrow j$

input $\vec{x} = a^{[0]}$

Inference in Code




```
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
```

x = np.array([[200, 17]]) ⟶ [200 17] ← 2D array

x = np.array([200, 17]) ⟶ no row or column ← 1D vector ~~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 = 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)

model.predict(x_new)
```

Vectorization

loops vs vectorization

```
x = np.array([200, 17])
W = np.array([[-1, -3, 5],
              [-2, 4, -6]])
b = np.array([-1, 1, 2])
units = w.shape[1]
def dense(a_in, W, b):
    a_out = np.zero(units)
    for j in range(units)
        w = W[:, j]
        z = np.dot(w, x) + b[j]
        a[j] = g(z)
    return a.
```

```
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^T, W) + B
    A_out = g(z)
    return A_out
```

AT = A.T

Transpose

z = np.matmul (AT, w)

(z = AT @ w)

AT = np.array ([[200, 17]])

w = np.array ([[1, -3, 5],
[-2, 4, -6]])

b = np.array ([[-1, 1, 2]])

z = np.matmul (AT, w) + b

a_out = g(z)

Train a NN in Tensor Flow

```
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)
```

↑ number of steps in gradient descent

logistic loss → Binary Crossentropy

Regression (Predicting numbers)

Loss → Mean Square Error ()

Choosing Activation Function

Output layer

Binary classification - Sigmoid

Regression - Linear activation function (+ and -)

activation = 'linear'

Only + - ReLU

activation = 'relu'

Hidden layer

Common - ReLU

Softmax

Softmax regression (N possible outputs)

$y = 1, 2, \ldots, N$

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

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

$a_1 + \ldots + a_N = 1$

for 2 outputs

$a_1 = g(z)$

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

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

$a_1 + a_2 = 1$

Cost

Logistic Regression

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

$$a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$$

$$a_2 = 1 - a_1 = P(y=0|\vec{x})$$

$$\text{loss} = \underbrace{-y \log a_1 - (1-y) \log \underbrace{(1-a_1)}_{a_2}}_{y=0}$$

$J(\vec{w}, b)$ = average loss

Softmax regression

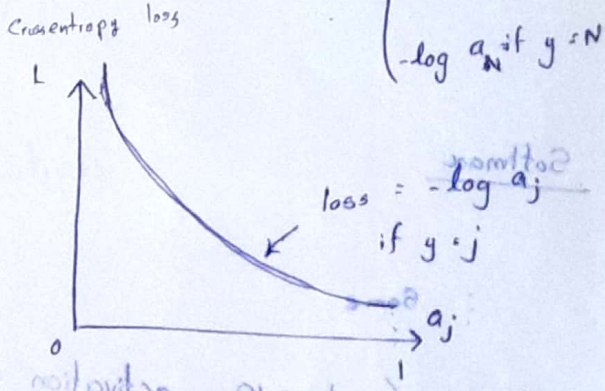
$$a_1 = \frac{e^{z_1}}{e^{z_1} + \dots + e^{z_N}} = P(y=1|\vec{x})$$

$$\vdots$$

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

Crossentropy loss

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



Neural Network with softmax

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

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

from tensorflow.keras.losses import
    SparseCategoricalCrossentropy

model.compile(loss = SparseCategoricalCrossentropy())

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

Improved version of Softmax

Logistic Regression

Numerical Roundoff error

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

model.compile(loss = BinaryCrossEntropy())

model.compile(loss = BinaryCrossEntropy(from_logits = True))
```

logit : z

Softmax

Same

```
Dense(units = 10, activation = 'linear')

model.compile(loss = SparseCrossEntropy(from_logits = True))
```

*) ഇത് output കിട്ടുന്നത് $a_1, \ldots a_{10}$ ആയിരിക്കും.

$z_1, \ldots z_{10}$

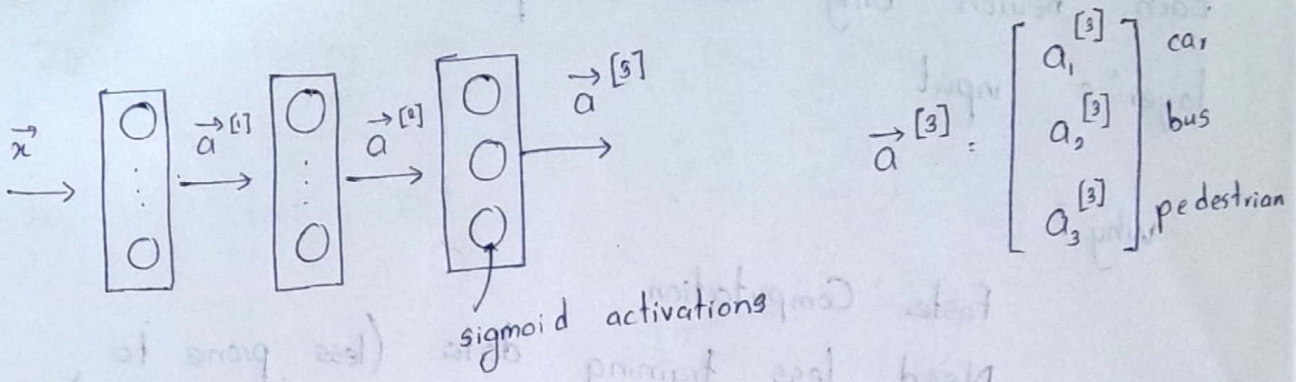
```
predict:  logits = model(x)
          f_x = tf.nn.softmax(logits)
```

for logistic regression

```
logit = model(x)
f_x   = tf.nn.sigmoid(logit)
```

Classification with multiple output

(x) Multi-label classification (Self driving car)



Advanced Optimization

Gradient Descent Optimization

Adam : Adaptive Moment estimation

not just one α

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

$$\vdots$$

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

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

If w_j (or b) keeps moving same direction $\alpha_j \uparrow$

oscillation $\alpha_j \downarrow$

code

compile

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

Additional Layer Types

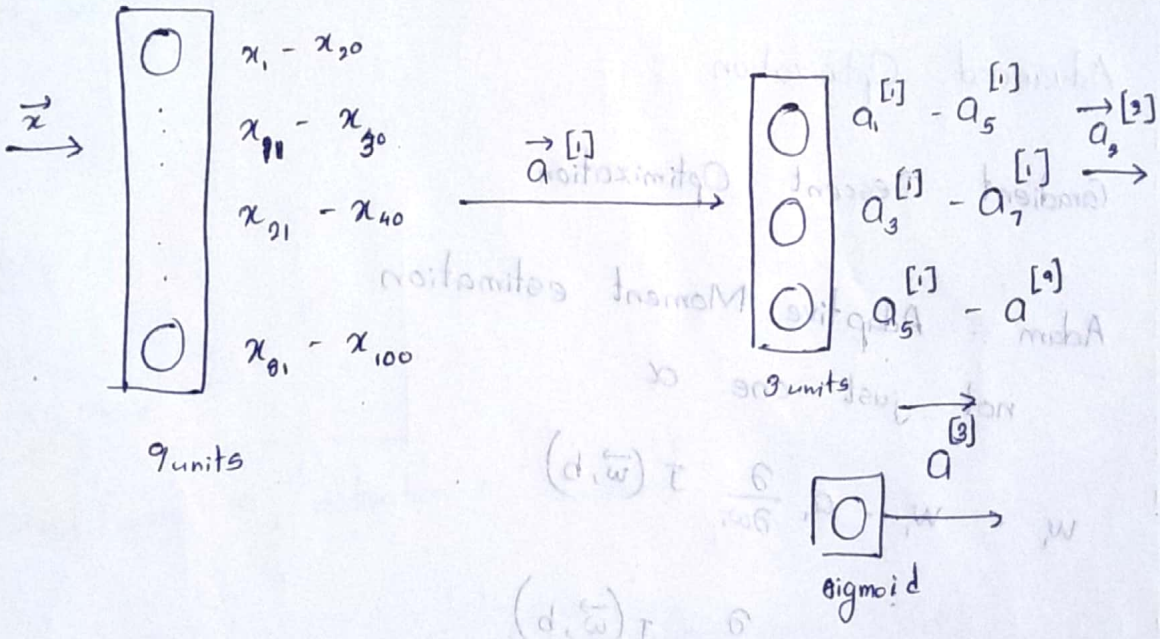
Convolutional Layer

Each neuron only looks at part of the previous layer's input

Why?

Faster Computation

Need less training data (less prone to overfitting)



Evaluating your model

$(x^{(1)}, y^{(1)})$

$\vdots$

$(x^{(m_{train})}, y^{(m_{train})})$

70% training set

30% test set

$(x^{(1)}_{test}, y^{(1)}_{test})$

$\vdots$

$(x^{(m_{test})}_{test}, y^{(m_{test})}_{test})$

Minimize cost function

for regression

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

Compute test error

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

Training error

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

for Classification

minimize

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

Test error - only testing set

Training error - only training set

Model Selecting & Training

Training . 60%

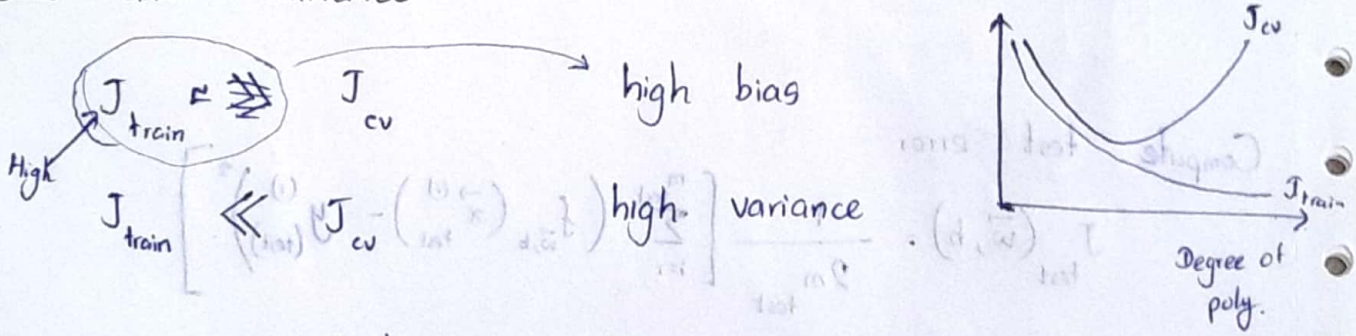
Cross Validation 20% ← Dev set (Development)

Testing 20% :

Select the model which has least CV error.

(x) Take the decision based on cv test or test set.

Bias and Variance



J_{train} high

High bias & High Variance

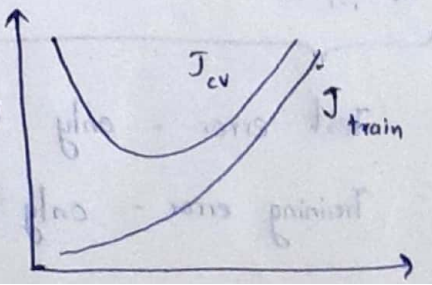
$J_{train} \ll J_{cv}$

Regularization and bias / variance

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

$\lambda = 0 \rightarrow 10$ J_{cv} අගය ලබා ගැනීම මෙහි අරමුණ වේ.

0.01

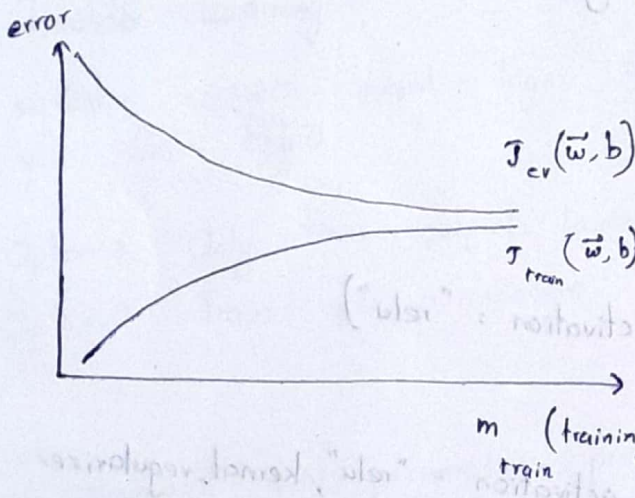


Establishing a baseline level of performance

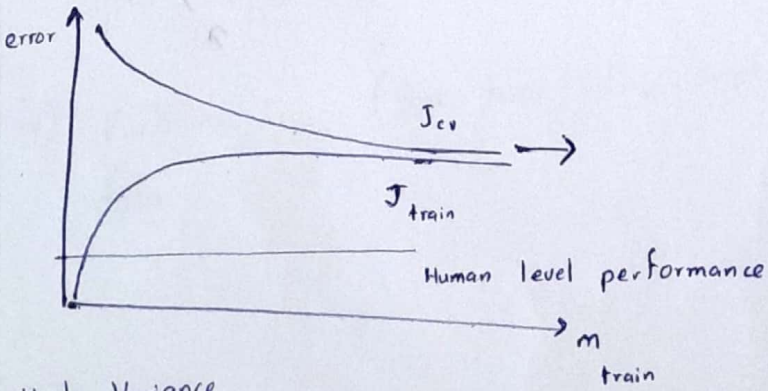
What is the level of error you can reasonably hope to get to ?

* Human level performance
* Competing algorithms performance
* Guess based on experience.

Learning curve

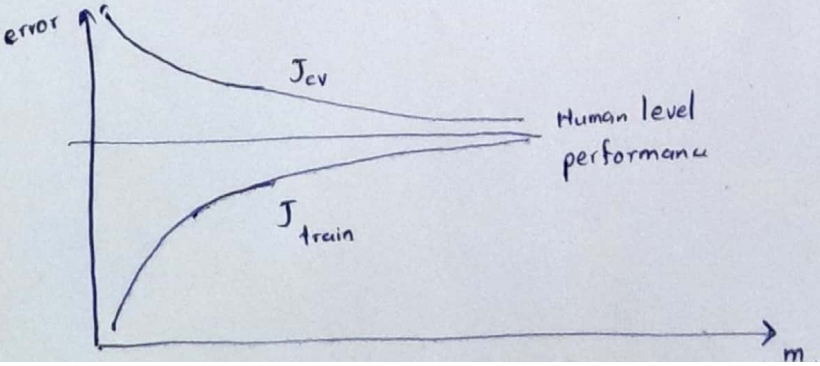


High bias



High bias නම් Data වැඩි කරල වැඩක් නෑ.

High Variance



High variance නම් Data වැඩි කරන්න.

Debugging a learning Algorithm

large error ~~(high bias)~~

Get more training example - fixes high variance

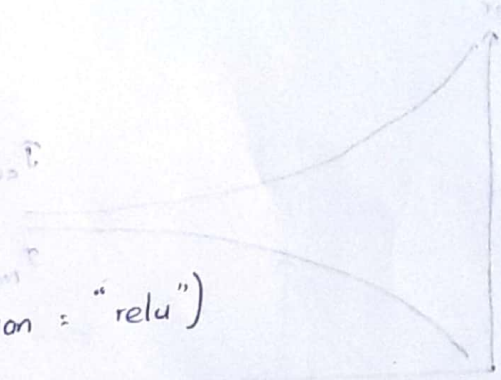
Try smaller sets of features - fixes high variance

Try getting additional features - fixes high bias

Adding polynomial features - fixes high bias

Decrease λ - fix high bias

Increase λ - fix high variance



Bias / Variance in NN

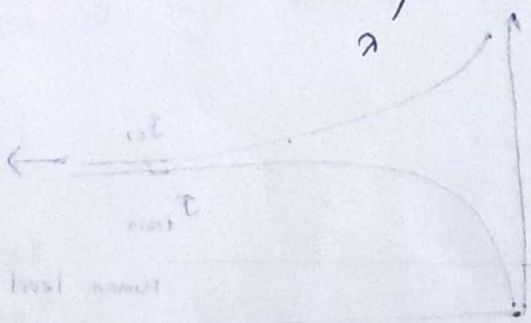
Unregularized MNIST model

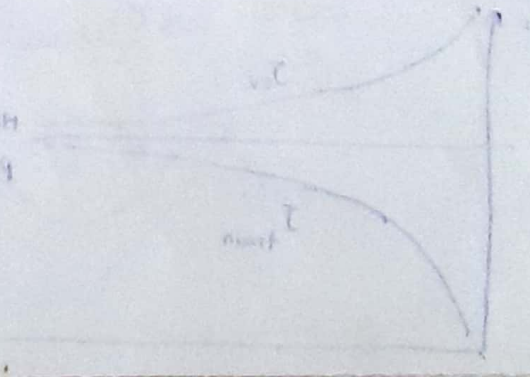
layer_1 = Dense(units = 25, activation = "relu")

Regularized MNIST model

layer_1 = Dense(units = 25, activation = "relu", kernal_regularizer = L2(0.01))

λ





Engineering the data used by your system

AI = Code (Algorithm / model) + Data

Code: Conventional model - centric

Data: Data - centric

Transfer Learning

තොඩෙක් වෙලාවට output layer එක වෙනස් කරනවා අවට මිනා විදිහට

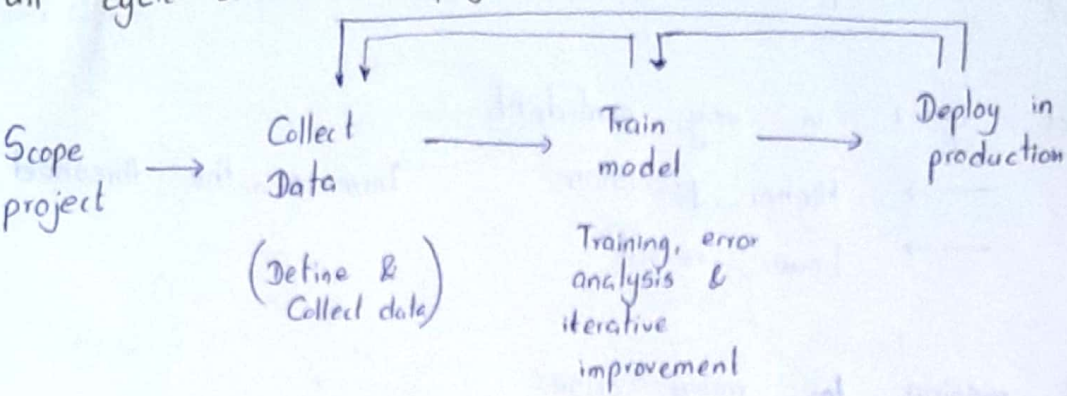
Option 1: Only train output layer parameters. (Less data)

Option 2: Train all parameters (Data set එක shape එකේ ලොකු නම්)

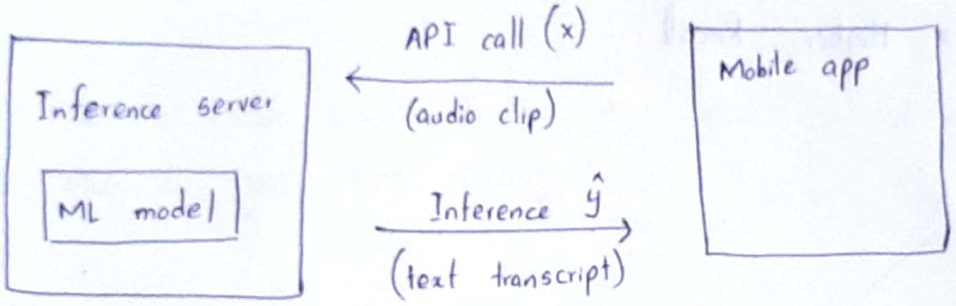
01) Download NN parameters pretrained on a large dataset with same input type (eg: images, audio, text) as your application (or train your own)

02) Further train (fine tune) the network on your own data.

Full cycle of ML project



Deplogment



Fairness, bias & Ethics

Follow the guidelines.

Precision / Recall

Predicted class	Actual class: 1	Actual class: 0
1	True Positive	False Positive
0	False Negative	True Negative

$$\text{Precision} = \frac{TP}{\text{Predicted Positive}} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{\text{Actual Positive}} = \frac{TP}{TP + FN}$$

Trading off precision & Recall

predict $y = 1$ in very confident

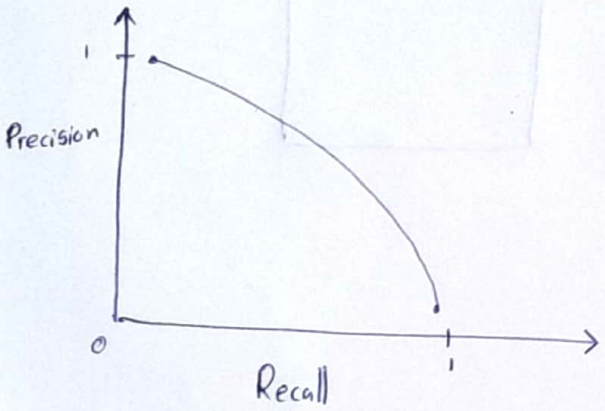
→ Higher precision — Increase the threshold

→ Lower recall

Avoid missing too many cases

→ Lower precision — Lower the threshold

→ Higher Recall



F1 Score $= \frac{1}{\frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right)} = \frac{2PR}{P+R}$

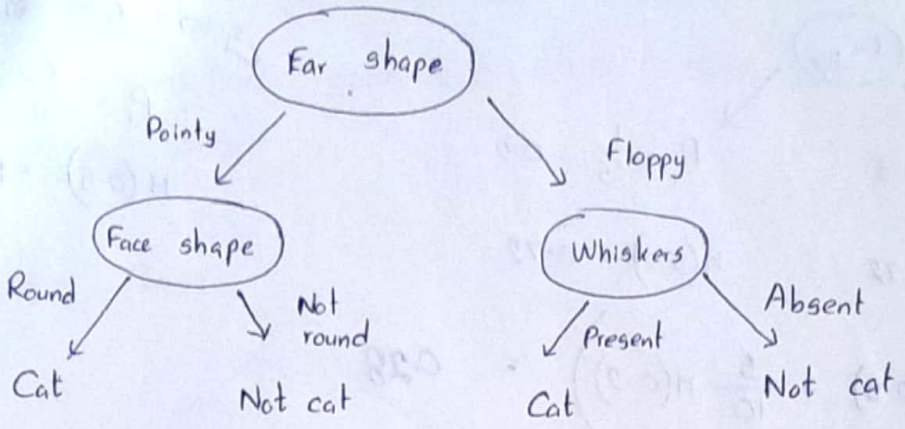
F1 Score $= \frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right)$

F1 Score එක වැඩි වෙන්න ඕන.

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
- When improvements in purity score are below a threshold
- When number of examples in a node below a threshold

Decision Trees

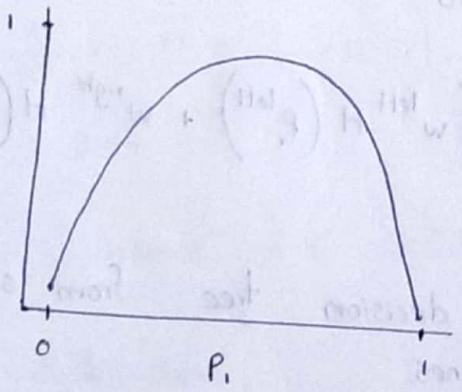


01). How to choose what feature to split on at each node?

02). When do you stop splitting?

Measuring purity

P_1 = fraction of examples that are cats



$H(P_1)$ ← Entropy of impurity ← measure of impurity

$$P_0 = 1 - P_1$$

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

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

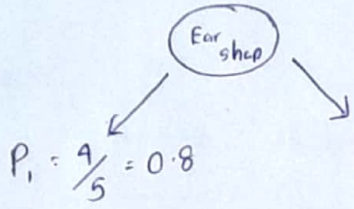
Note. $0 \log(0) = 0$

Choosing a split : Information Gain

මුලදී කැල්ලෙ 5යි පූනය 5ම තිබිය.

$\therefore P_1 = \frac{5}{10} = 0.5$

$H(0.5) = 1$



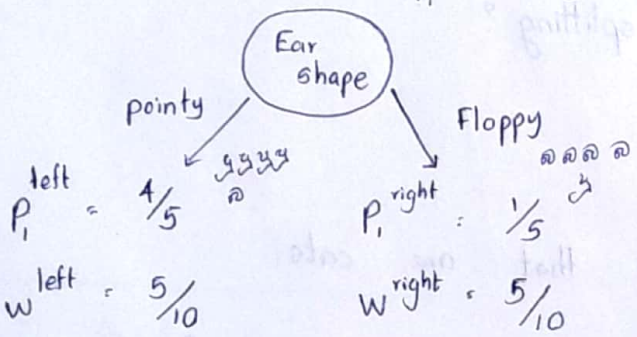
$P_1 = \frac{4}{5} = 0.8$ $\quad P_1 = \frac{1}{5} = 0.2$

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

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

මේ calculation එකෙන් එකම value එක තමයි.

$P_1^{root} = \frac{5}{10} = 0.5$



$P_1^{left} = \frac{4}{5}$ $\quad P_1^{right} = \frac{1}{5}$

$w^{left} = \frac{5}{10}$ $\quad w^{right} = \frac{5}{10}$

$$\text{Information gain} = H\left(P_1^{root}\right) - \left(w^{left} H\left(P_1^{left}\right) + w^{right} H\left(P_1^{right}\right)\right)$$

Recursive algorithm : Build decision tree from small small ones.

One hot encoding

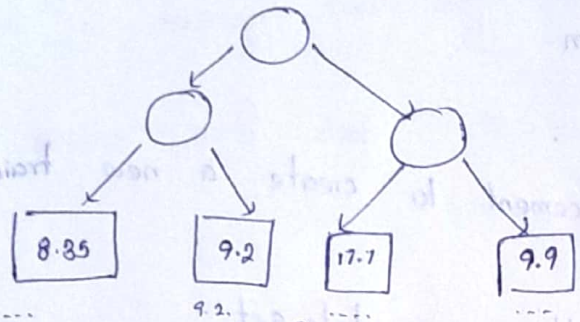
Categorical features තමයි මේක.

Pointy ears	Floppy ear	Round ear	Face shape	Whiskers
1	0	0	~~Round~~ 1	~~Present~~ 1

Continuous valued feature

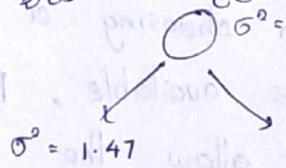
Information Gain එක හොයන්න threshold value එකකට. වැඩිම Information gain එක තියන value එක threshold එක විදියට ගන්න.

Regression Trees



leaf node එක වෙනකන් මෙදුල. Avg. value එක ගන්නෙ.

මේකේදී පළවෙනියට split කරන එවා හොයද්දී variance එක ගන්නෙ.



$$20.51 - \left(\frac{5}{10} \times 1.47 + \frac{5}{10} \times 21.87\right)$$

$$= 8.84$$

Reduction variance එක වැඩිම එක ගන්න ඕන.

Using multiple trees.

Tree ensemble : Trees ගොඩක් හොයල / හදල වැඩියෙන්ම දෙන output එක result එක දෙනවා.

Sampling with replacement

Data set එකක් හදුනාගෙන ඒකෙන් data randomly replace කර කර

Random Forest Algorithm

- × Sampling & Replacement Data Set
- × Create Decision trees

මෙහෙම trees ගොඩක් හදල prediction එක ගන්නවා

Given training set of size m

For b = 1 to B

Use sampling ~~whie~~ with replacement to create a new training set of size m

Train a decision tree on the new data set

At each node, when choosing a feature to use to split, if n features are available, pick a random subset of $k < n$ features and allow the algorithm to only choose from that subset of features

$$k = \sqrt{n}$$

XGBoost

Given training set of size m

For b = 1 to B

Use sampling with replacement to create a new training set of size m

Train a decision tree on the new dataset.

හොඳට වෙන්නෙ නැති එකට Decision tree හදනවා.

Classification

```
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

Regression

```
from xgboost import XGBRegressor
model = XGBRegressor()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

When to use Decision Trees

Decision Trees and Tree ensembles

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast development
- Small decision trees may be human interpretable

Neural Networks

- Works well on all types of data, including tabular (structured) and unstructured data.
- May be slower than a decision tree
- Works with transfer learning
- When building a system of multiple models working together, it might be easier to string together multiple NNs.