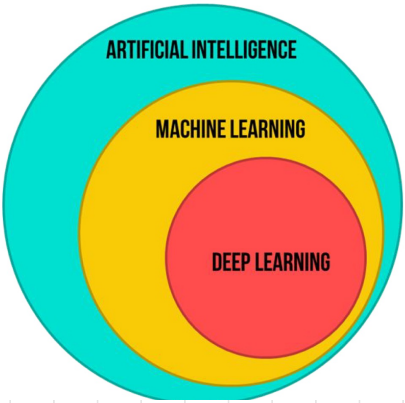
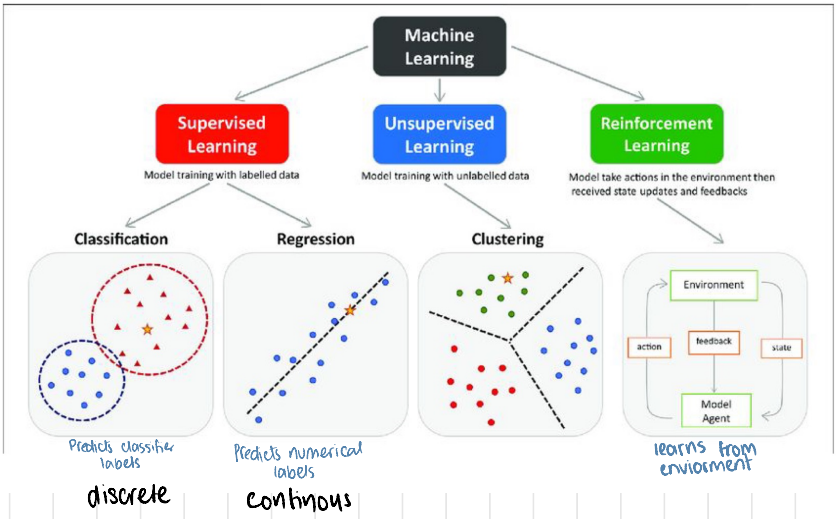
Deep Learning for Perception

MACHINE LEARNING RECAP





D. Is it possible to use ReLU activation in the last layer for (i) classification and (ii) Clustering?
For unsupervised learning, yes. For classification NO

MACHINE LEARNING TERMS

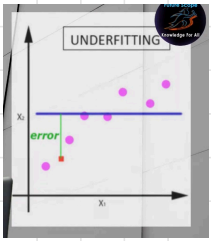
overfitting

↳ memorises noise instead of patterns
↳ learns training data too well
↳ training accuracy high
↳ testing accuracy too low
↳ dosent work well on unseen data

Learns more features than req.
might learn diameter and material of chair
so wont classify other chairs as a chair

underfitting

↳ model too simple to capture underlying trends
↳ training accuracy too low
↳ high training error



might only learn 1 feature, 4 legs
So will classify dog as a chair too as both 4 legs

↳ when less sample size

Training

↳ tuning weights
↳ changing your parameters (weights, biasness)

Validation

↳ used to optimize hyperparameters (epoches, activation function)

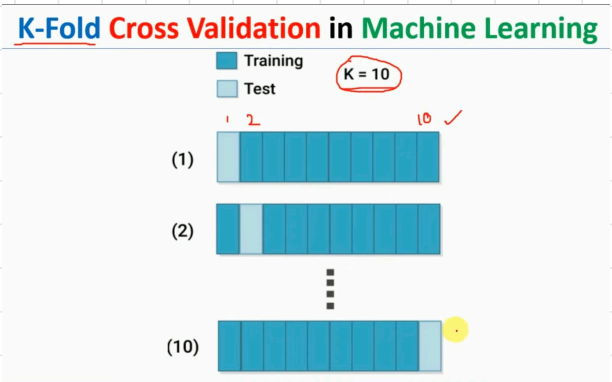
Testing

↳

cross validation

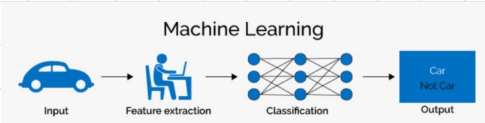
(aka K-Fold)

↳ avoids over and under fitting
↳ tests on different folds
↳ reduces biasnes by working on different splits in data
↳ averages out your accuracy
↳ gives validation accuracy



Machine Learning

steps
- Input
- feature extraction → done manually (eyes, nose, lips) → so can have biasness
- Classification
- Output



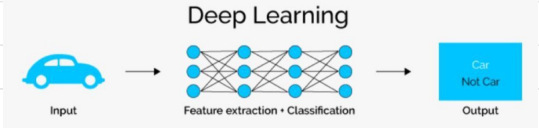
- makes the pic black and white
- then identifies edges
- then fragments → eyes, nose, ears

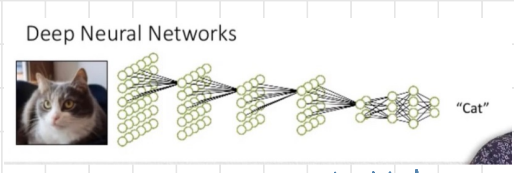
- These are hard engineered features that are
  - time consuming → manually select relevant features
  - brittle → isn't generalized features will fail when data distribution changes
  - not scalable in practice → when dataset grows manually selecting becomes unfeasible

what your eyes can't do
Deep learning model can't do either

Deep Learning

steps
- Input
- feature extraction + Classification → done automatically (might not take eyes as a feature but something else)
- Output

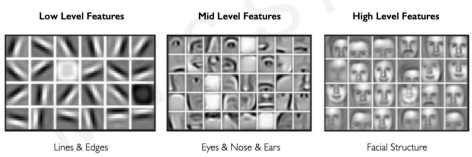


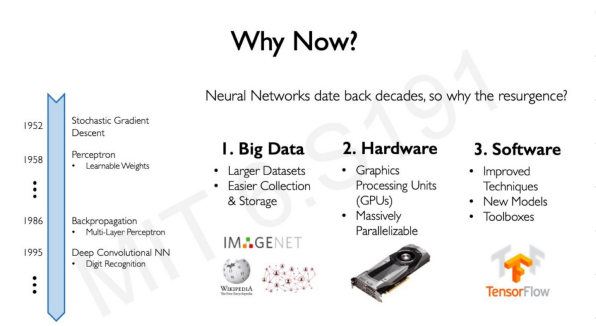


use some cv filers matrix → edge detector → gradient to generate

- It learns underlying features directly from data automatically

HIERARCHICAL FEATURES

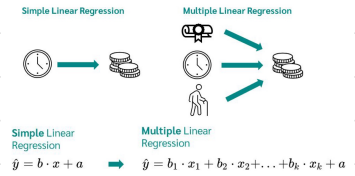




Linear Regression

↳ Output is continous $-\infty \rightarrow +\infty$

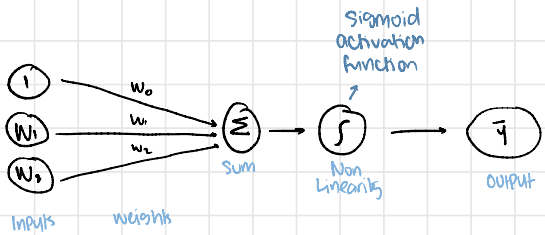
↳ Predicts outcomes by learning weights and outcomes throug Gradient Descent

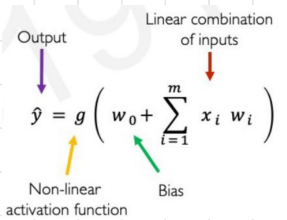


Logistic Regression

↳ Using activation function in Linear Regression

Forward Propogation

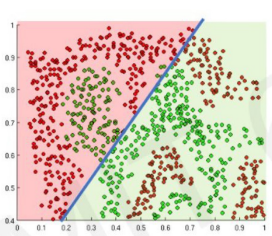




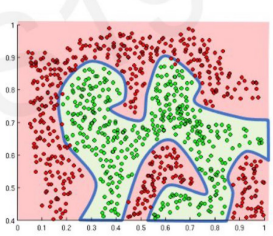
Activation Function

↳ used to introduce non-linearity

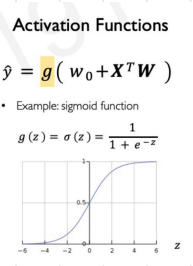
↳ chooses which nodes should be activated



Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions



Common Activation Functions

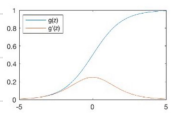
Sigmoid Function

Last layer

$\sigma = \frac{1}{1+e^{-n}}$

$-\infty \to 0$ → class 0
$0 \to 0.5$
$a \to 1$ → class 1

$\delta = a(1-a)$ derivative



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

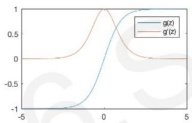
$g'(z) = g(z)(1-g(z))$

`tf.math.sigmoid(z)`

Hyperbolic Tangent

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

$\delta = 1 - f(n)^2$ derivative



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

$g'(z) = 1 - g(z)^2$

`tf.math.tanh(z)`

ReLU

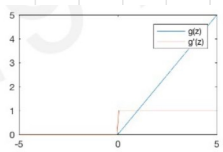
Rectified Linear Unit

Best for first, mid layer

↳ makes -ve value 0

$$f(x) = \begin{cases} 0 \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases} \quad \to \text{derivative}$$



$g(z) = \max(0, z)$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

`tf.nn.relu(z)`

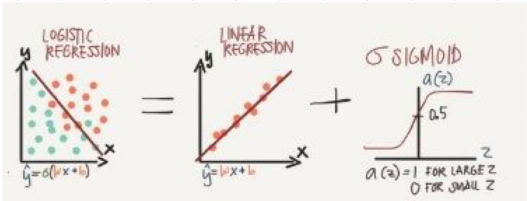
↳ for unsupervised

3. Is it possible to use ReLU in the last layer?
   Ans: For unsupervised learning, yes. For classification NO

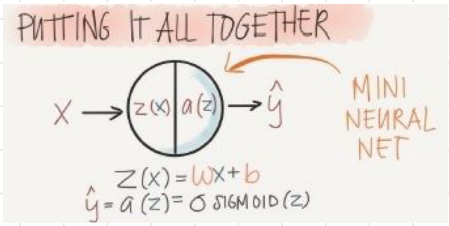
LOGISTIC REGRESSION

↳ its outcome variable is binary

↳ basically activation function in linear regression



C. Briefly discuss the training and testing phase of Logistic Regression.
training: we train the system (specifically the weights w and b) using stochastic gradient descent and the cross-entropy loss.
test: Given a test example x we compute p(y|x) and return the higher probability label y = 1 or y = 0.



Steps to learn parameters

1. Loss function/Cost function
   ↳ measures how well models predictions ŷ (σ(wx+b)) match the actual y (0 or 1)
   ↳ gives loss score — that model tries to minimise during training
2. Optimizer algorithm
   ↳ uses loss score to
   ↳ iteratively update weights

1. Forward propogation
   ↳ Calculate ŷ
2. Backward propogation
   ↳ Gradient Descent + update w and b

REPEAT UNTIL IT CONVERGES

Loss Function or Cost Function

Conditional maximum likelihood estimation prefers the correct class labels of the training examples to be more likely.

We choose the parameters w, b that maximize the log probability of the true y labels in the training data given the observations x.

Maximum Likelihood Estimation

$$P(y=1) = \sigma(w \cdot x + b) = \frac{1}{1+e^{-(w \cdot x+b)}}$$

For samples labeled as '1', we try to estimate (w and b) such that the product of all probability p(x) is as close to 1 as possible.

And for samples labeled as '0', we try to estimate (w and b) such that the product of all probability is as close to 0 as possible in other words (1 - p(x)) should be as close to 1 as possible.

$$\text{for samples labelled as } 1: \prod_{x \text{ in } y_i=1} p(x_i)$$

$$\text{for samples labelled as } 0: \prod_{x \text{ in } y_i=0} (1-p(x_i))$$

On combining both , (w and b) parameters should be such that the product of both of these products is maximum over all elements of the dataset.

$$L(\beta) = \prod_{x \text{ in } y_i=1} p(x_i) * \prod_{x \text{ in } y_i=0} (1-p(x_i))$$

This function is the one we need to optimize and is called the likelihood function.

Loss Function

if $y=1 \rightarrow P(y|x) = \hat{y}$

if $y=0 \rightarrow P(y|x) = 1-\hat{y}$

$$L = -[y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

Loss Function Types

1. Cross Entrophy → multiclass classification
2. Binary Cross Entrophy → logistic Regression, Binary classification
3. MSE → regression tasks (numerical labels)

Log Likelihood

for probabilistic models

↳ evaluates how well a model explains observed data

↳ goal is to maximise the log likelihood in training

means minimize loss function like cross entropy

Since there are only two discrete outcomes (1 or 0), this is a Bernoulli distribution. → 2 diff outcomes

p(y|x) for one observation:

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

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

whatever values maximize a probability will also maximize the log of the probability.

p(y|x) for n observation:

$$L(\hat{y}) = \prod_i \hat{y}^y (1-\hat{y})^{1-y}$$

$$l(\hat{y}) = \sum_{i=1}^{n} y\log\hat{y} + (1-y)\log(1-\hat{y})$$

where, $l(\hat{y})$ is log − likelihood

use ln instead of log in calculator

$$\mathcal{L} = y \log(\hat{y}) + (1-y)\log(1-\hat{y})$$

Derived from log likelihood in classification tasks

LOSS FUNCTION TYPES

1. Cross Entropy

for multiclass classification

↳ measures diff b/w 2 probability distributions

True Distribution (actual labels) — Predicted Distribution

$$\mathcal{L} = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

Log likelihood should be maximized.

In order to turn this into loss function (something that we need to minimize), we'll just flip the sign.

negative log likelihood loss = cross-entropy loss.

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$$

$$L_{CE}(w,b) = -[y\log\sigma(w\cdot x+b) + (1-y)\log(1-\sigma(w\cdot x+b))]$$

True label (y)	Predicted Probability ($\hat{y}$)	Loss
1	0.9	$-\log(0.9) = 0.1054$
0	0.2	$-\log(0.8) = 0.223$
1	0.8	$= 0.2231$
0		$= 0.5108$

$0.1054 + 0.223 + 0.2231 + 0.5108$ (sum of loss)

$\frac{1.0624}{4} = 0.2656$

2. Binary Cross Entropy

for Binary classification

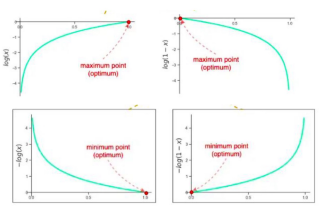
↳ used for binary classification problems

↳ 2 classes → 0, 1

Binary cross-entropy is often calculated as the average cross-entropy across all data examples.

$$\mathcal{L} = \frac{-1}{N}[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$$



3. Mean Squared Error (MSE)

↳ used in regression tasks

↳ measures the avg of squared diff b/w actual and predicted values

$$L = (\text{actual} - \text{predicted})^2$$

↳ if more than 1 take avg

WHY NOT MSE?

↳ struggles with classification involving probabilities

1. Treats all errors same

e.g. if class 1 is cat and
- ↳ model predits 0.01 % confidence → bad prediction
- ↳ model predits 0.99% confidence → not agreed

BUT MSE will treat both errors similarly even though 0.01 is a much worse prediction

1. Actual label: 1, Predicted: 0

MSE: $(1-0)^2 = 1$

log loss: $-(1 \log(0) + 0 \log(1)) = \infty$ loss

2. Actual label: 1, Predicted: 1

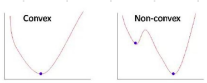
MSE: $(1-1)^2 = 0$

log loss: $-(1 \log(1) + 0 \log(0)) = 0$

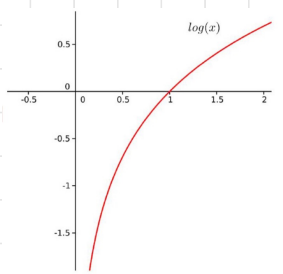
2. Leads to slow and bad convergence

↳ MSE is a non-convex function

↳ cross entropy is a convex function



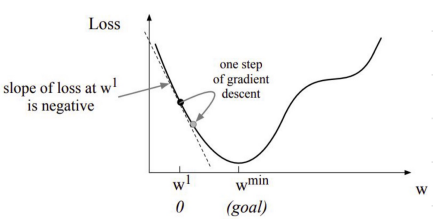
Gradient descent will converge into global minimum only if the function is convex.

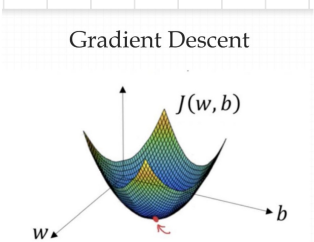


Gradient Descent Algo

↳ The Goal is to

↳ minimises loss function by finding optimal weights → for logistic regression $\theta = \underline{w}, b$

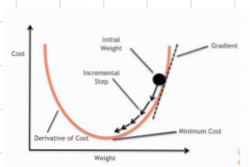




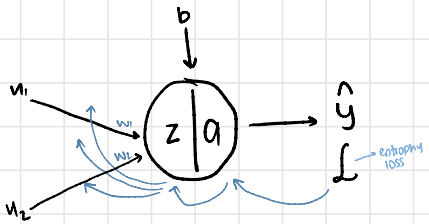
Steps

↳ choose starting point

↳ calculate gradient



Gradient - Backward Propogation



$a = \hat{y}$ → as only 1 neuron

Entrophy loss

$$\mathcal{L} = -[y \log(a) + (1-y)\log(1-a)]$$

$$\frac{\delta \mathcal{L}}{\delta a} = -y \times \frac{1}{a}(1) - (1-y) \times \frac{1}{1-a}(-1)$$

$$\boxed{\frac{\delta \mathcal{L}}{\delta a} = \frac{-y}{a} + \frac{1-y}{1-a}}$$

Activation Function

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

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

$$\boxed{\frac{\delta a}{\delta z} = a(1-a)} \rightarrow \text{derivative given in paper}$$

Forward Propogation

$$z = w_1 n_1 + w_2 n_2 + b$$

$$\boxed{\frac{\delta z}{\delta w_1} = n_1}$$

Updating w_1

$$\frac{\delta \mathcal{L}}{\delta w_1} = \frac{\delta \mathcal{L}}{\delta a} \cdot \frac{\delta a}{\delta z} \cdot \frac{\delta z}{\delta w_1}$$

$$= \left[\frac{-y}{a} + \frac{1-y}{1-a}\right] \times a(1-a) \times n_1$$

$$\therefore \left[\frac{-y(1-a) + a(1-y)}{a(1-a)}\right] \times a(1-a) \times n_1$$

$$\therefore \left[-y + ay + a - ay\right] \times n_1$$

$$\frac{\delta \mathcal{L}}{\delta w_1} = (a-y)n_1$$

$$\boxed{w_1^+ = w_1 - \alpha \frac{\delta \mathcal{L}}{\delta w_1}} \quad (\alpha = \text{learning weights})$$

$$\boxed{b^+ = b - \alpha \frac{\delta \mathcal{L}}{\delta b}}$$

Updating w_2

⇒ same steps only change w_1 to w_2

$$\frac{\delta \mathcal{L}}{\delta w_2} = (a-y)n_2$$

$$\boxed{w_2^* = w_2 - \alpha \frac{\delta \mathcal{L}}{\delta w_2}}$$

Here, $(a-y) = \frac{\delta \mathcal{L}}{\delta z}$

$\frac{\delta z}{\delta w_1} = n_1$ $\quad$ $\frac{\delta z}{\delta w_2} = n_2$

where, $\frac{\delta \mathcal{L}}{\delta b} = (a-y)$

$\frac{\delta z}{\delta b} = 1$

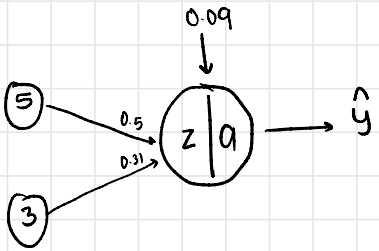
$\frac{\delta a}{\delta z} = a(1-a)$

Question

$w_1 = 0.5$ $x_1 = 5$ $\alpha = 0.01$ (learning rate)

$w_2 = 0.31$ $x_2 = 3$ $y = 1$

$b = 0.09$



$z = w_1x_1 + w_2x_2 + b$

$= 0.5(5) + 0.31(3) + 0.09$

$= 3.52$

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

$= \frac{1}{1+e^{-3.52}}$

$= 0.97$

$L = -y \log(a)$

$= -y \log(0.97)$

$= 0.0305$

$\frac{\delta L}{\delta w_1} = (a-y)x_1$ $= -0.15$

$\frac{\delta L}{\delta w_2} = (a-y)x_2$ $= -0.09$

$\frac{\delta L}{\delta b} = a - y$ $= -0.03$

$w_1^+ = w_1 - \alpha \frac{\delta L}{\delta w_1}$

$w_1^+ = 0.5 - (0.01)(-0.15)$

$= 0.5015$

$w_2^+ = w_2 - \alpha \frac{\delta L}{\delta w_2}$

$w_2^+ = 0.31 - (0.01)(-0.09)$

$= 0.3109$

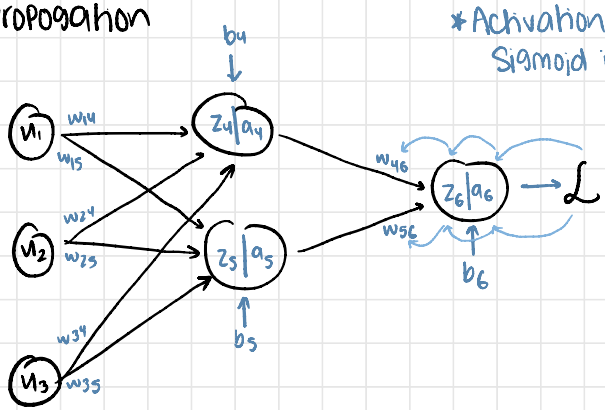
$b^+ = b - \alpha \frac{\delta L}{\delta b}$

$b^+ = 0.09 - (0.01)(-0.03)$

$= 0.0903$

Question → Backpropogation

*Activation function is Sigmoid in all layers



↳ For log loss and cross entropy both

$$\mathcal{L} = -[y \log(a) + (1-y)\log(1-a)]$$

$$\frac{\delta \mathcal{L}}{\delta a_6} = -[y \log(a_6) + (1-y)\log(1-a_6)]$$

$$= -y \times \frac{1}{a_6}(1) - (1-y) \times \frac{1}{1-a_6}(-1)$$

$$= \frac{-y}{a_6} + \frac{(1-y)}{1-a_6}$$

$$z_6 = a_4 w_{46} + a_5 w_{56} + b_6$$

$$\frac{\delta z_6}{w_{46}} = a_4$$

$$\frac{\delta a_6}{\delta z_2} = a_6(1-a_6)$$ → derivative given in paper

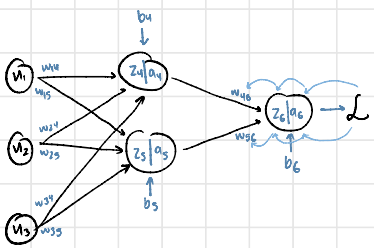
$$\frac{\delta \mathcal{L}}{\delta w_{46}} = \frac{\delta \mathcal{L}}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta w_{46}}$$

$$= \left[\frac{-y}{a_6} + \frac{(1-y)}{1-a_6}\right] \cdot a_6(1-a_6) \cdot a_4$$

$$= \left[\frac{-y(1-a_6) + a_6(1-y)}{a_6(1-a_6)}\right] a_6(1-a_6) \cdot a_4$$

$$= [-y + a_6 y + a_6 - a_6 y]\, a_4$$

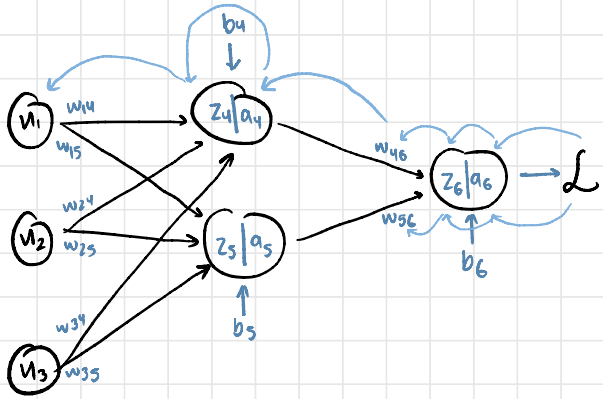
$$\frac{\delta \mathcal{L}}{\delta w_{46}} = (a_6 - y)\, a_4$$



$$= -(y - a_6)\, a_6(1-a_6)\, a_4$$

$$\frac{\delta \mathcal{L}}{w_{56}} = (a_6 - y)\, a_5$$

$$\frac{\delta \mathcal{L}}{\delta b_6} = a_6 - y$$



↳ For hidden layer

$$\frac{\delta L}{\delta w_{14}} = \frac{\delta L}{\delta a_4} \cdot \frac{\delta a_4}{\delta z_4} \cdot \frac{\delta z_4}{\delta w_{14}} \qquad \frac{\delta L}{\delta a_4} = \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta a_4}$$

$$= (a_6 - y) w_{46} \cdot a_4(1-a_4) \cdot n_1 \qquad = (a_6 - y)\, w_{46}$$

Substitute

$$\frac{\delta L}{\delta w_{14}} = (a_6 - y) w_{46} \cdot a_4(1-a_4) \cdot n_1 \qquad \frac{\delta L}{w_{15}} = (a_6 - y) w_{56} \cdot a_5(1-a_5) \cdot n_1$$

$$\frac{\delta L}{\delta w_{24}} = (a_6 - y) w_{46} \cdot a_4(1-a_4) \cdot n_2 \qquad \frac{\delta L}{\delta w_{25}} = (a_6 - y) w_{56} \cdot a_5(1-a_5) \cdot n_2$$

$$\frac{\delta L}{\delta w_{34}} = (a_6 - y) w_{46} \cdot a_4(1-a_4) \cdot n_3 \qquad \frac{\delta L}{\delta w_{35}} = (a_6 - y) w_{56} \cdot a_5(1-a_5) \cdot n_3$$

$$\frac{\delta L}{\delta b_4} = (a_6 - y) w_{46}\, a_4(1-a_4) \qquad \frac{\delta L}{\delta b_5} = (a_6 - y) w_{56} \cdot a_5(1-a_5)$$

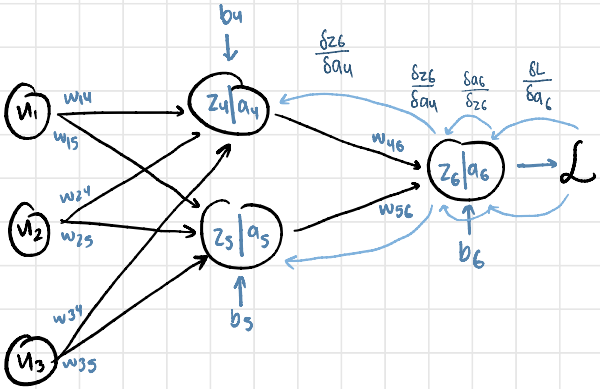
Question → Backpropogation squared

$$\mathcal{L} = \tfrac{1}{2}(y - a_6)^2$$

$$\frac{\delta \mathcal{L}}{\delta a_6} = \frac{1}{2}\left[2(-1)(y - a_6)\right]$$

$$= -1(y - a_6)$$

$$= a_6 - y$$



$$z_4 = w_{14}x_1 + w_{24}x_2 + w_{34}x_3 + b_4$$

a_4 : sigmoid (z_4)

$$z_5 = w_{15}x_1 + w_{25}x_2 + w_{35}x_3 + b_5$$

a_5 : sigmoid (z_5)

$$z_6 = w_{46}a_4 + w_{56}a_5 + b_6$$

a_6 : sigmoid (z_6)

$$\mathcal{L} = \tfrac{1}{2}(y - a_6)^2$$

$$\frac{\delta L}{\delta w_{46}} = \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta w_{46}}$$

$$= -(y - a_6) \cdot a_6(1 - a_6) \cdot a_4$$

$$w_{46}^{+} = w_{46} - \eta \frac{\delta L}{\delta w_{46}}$$

(η: learning rate)

$$\mathcal{L} = \tfrac{1}{2}(y - a_6)^2$$

$$\frac{\delta L}{\delta w_{56}} = \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta w_{56}}$$

$$= -(y - a_6) \cdot a_6(1 - a_6) \cdot a_5$$

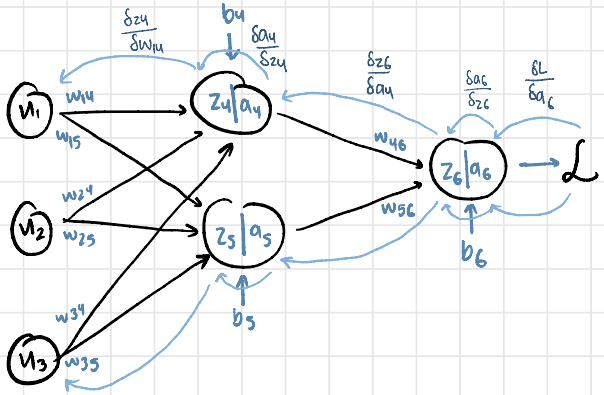
$$w_{56}^{+} = w_{56} - \eta \frac{\delta L}{\delta w_{56}}$$

$$\mathcal{L} = \tfrac{1}{2}(y - a_6)^2$$

$$\frac{\delta L}{\delta b_6} = \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta b_6}$$

$$= -(y - a_6) \cdot a_6(1 - a_6)$$

$$b_6^{+} = b_6 - \eta \frac{\delta L}{\delta b_6}$$



↳ For Hidden Layer

$$\frac{\delta L}{\delta w_{14}} = \frac{\delta L}{\delta a_4} \cdot \frac{\delta a_4}{\delta z_4} \cdot \frac{\delta z_4}{\delta w_{14}} \qquad \frac{\delta L}{\delta a_4} = \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta a_4}$$

$$= -(y-a_6)\, a_6(1-a_6) \cdot w_{46}$$

substitute

$$\frac{\delta L}{\delta w_{14}} = -(y-a_6)\, a_6(1-a_6) \cdot w_{46} \cdot a_4(1-a_4) \cdot n_1 \qquad w_{14}^{+} = w_{14} - \eta \frac{\delta L}{\delta w_{14}}$$

$$\frac{\delta L}{\delta w_{24}} = -(y-a_6)\, a_6(1-a_6) \cdot w_{46} \cdot a_4(1-a_4) \cdot n_2 \qquad w_{24}^{+} = w_{24} - \eta \frac{\delta L}{\delta w_{24}}$$

$$\frac{\delta L}{\delta w_{34}} = -(y-a_6)\, a_6(1-a_6) \cdot w_{46} \cdot a_4(1-a_4) \cdot n_3 \qquad w_{34}^{+} = w_{34} - \eta \frac{\delta L}{\delta w_{34}}$$

$$\frac{\delta L}{\delta b_4} = -(y-a_6)\, a_6(1-a_6) \cdot w_{46} \cdot a_4(1-a_4) \qquad b_4^{+} = b_4 - \eta \frac{\delta L}{\delta b_4}$$

$$\frac{\delta L}{\delta w_{15}} = -(y-a_6)\, a_6(1-a_6) \cdot w_{56} \cdot a_5(1-a_5) \cdot n_1 \qquad w_{15}^{+} = w_{15} - \eta \frac{\delta L}{\delta w_{15}}$$

$$\frac{\delta L}{\delta w_{25}} = -(y-a_6)\, a_6(1-a_6) \cdot w_{56} \cdot a_5(1-a_5) \cdot n_2 \qquad w_{25}^{+} = w_{25} - \eta \frac{\delta L}{\delta w_{25}}$$

$$\frac{\delta L}{\delta w_{35}} = -(y-a_6)\, a_6(1-a_6) \cdot w_{56} \cdot a_5(1-a_5) \cdot n_3 \qquad w_{35}^{+} = w_{35} - \eta \frac{\delta L}{\delta w_{35}}$$

$$\frac{\delta L}{\delta b_5} = -(y-a_6)\, a_6(1-a_6) \cdot w_{56} \cdot a_5(1-a_5) \qquad b_5^{+} = b_5 - \eta \frac{\delta L}{\delta b_5}$$

2. Consider a classification problem where we are building a logistic regression classifier. The task is to predict if a given city has a risk of a disease epidemic or not. The data is defined using two input

Practice Problem 6

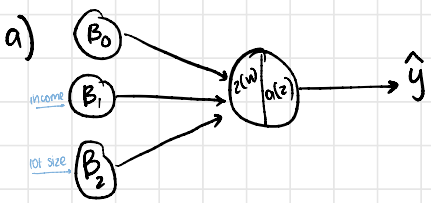
a. Construct a logistic regression model with two predictors for the riding mower example with $\beta_0 = -25.9382, \beta_1 = 0.1109, \beta_2 = 0.9638$, where β_1 and β_2 are for the "Income" and "Lot_Size" variables, respectively.

b. Using the logistic regression model with probability cutoff = 0.75, classify the following 6 customers as "Owner" or "Nonowner" : if $p \geq 0.75$ then the case as a "Owner". Present the results in a classification matrix.

Customer#	Income	Lot_Size	Ownership
1	60.0	18.4	Owner
2	64.8	21.6	Owner
3	84.0	17.6	Nonowner
4	59.4	16.0	Nonowner
5	108.0	17.6	Owner
6	75	19.6	Nonowner

onlv

a)



Predictor Variables

$z = B_0 + B_1(\text{income}) + B_2(\text{Lot Size})$

$z = -25.9382 + 0.1109(\text{Income}) + 0.9638(\text{Lot size})$

$P(Y=1) = \frac{1}{1+e^{-z}}$ → Purchasing

$P(y=0) = \frac{1}{1+e^{-z}}$ → not Purchasing

b) P cutoff = 0.75

$P \geq 0.75$ → Owner

$P < 0.75$ → Not Owner

Customer#	Income	Lot_Size	Ownership	z	$\hat{y}$	
1	60.0	18.4	Owner	-1.55028	0.175	NOT
2	64.8	21.6	Owner	2.0662	0.887	yes
3	84.0	17.6	Nonowner	0.34028	0.584	NOT
4	59.4	16.0	Nonowner	-3.92994	0.019	NOT
5	108.0	17.6	Owner	3.00188	0.9527	Yes
6	75	19.6	Nonowner	1.26978	0.78	yes

DEEP LEARNING TERMS

Input layer
- ↳ first layer of NN
- ↳ has input features

Output layer
- ↳ final layer of NN
- ↳ produces prediction
- ↳ no. of neurons of output layer depends on the type of task
  - ↳ classification with multiple classes

Hidden Layer
- ↳ layers b/w input and output
- ↳ they process data by applying weights, biases, activation functions to learn patterns from input data

Neurons/Nodes
- ↳ each node performs
  - ↳ a weighted sum of inputs
  - ↳ applies an activation function
- ↳ to produce an output

Shallow NN
- ↳ a NN with only 1 or few hidden layers
- ↳ simple networks
- ↳ struggle with complex problems

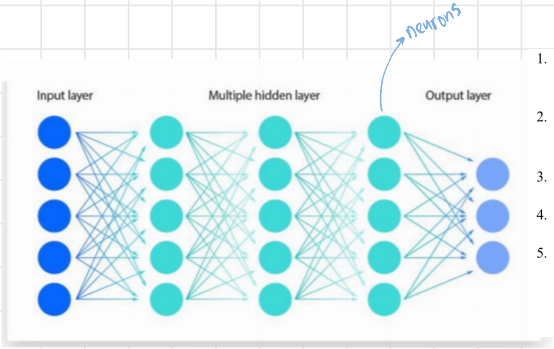
Deep NN
- ↳ a NN with multiple hidden layers
- ↳ good for complex problems
- ↳ are the foundation of deep learning

Epoch
- ↳ 1 complete pass through the entire training set

Parameter
- ↳ Internal values → weights, biases
- ↳ These are learned by model during training

Hyperparameter
- ↳ External settings → learning rate, no of layers, no of neurons
- ↳ These are set before training
- ↳ Controls the learning process

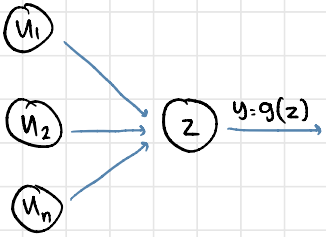
Activation Function
- ↳ applied to the output of neuron to introduce non-linearity
- ↳ helps network learn complex r/s.



1. Is it true that the number of neurons in the output layer should match the number of classes i.e. where is the number of classes are greater than 2.
   Ans: False: Depends upon output coding. For 4 class problem, two output neurons are enough
2. What are hyper-parameters? Name any three.
   Ans: Hyperparameters are configuration settings in machine learning models. Three examples are: Learning Rate, Number of Hidden Layers and Neurons, Batch Size
3. Is it possible to use ReLU in the last layer?
   Ans: For unsupervised learning, yes. For classification NO
4. Write a derivative equation for log loss with L2 regularization.
   Ans: (a-y)/(a(1-a) + (-) lambda w
5. What Is the Difference Between Epoch, Batch, and Iteration in Deep Learning?
   Ans: Epoch - Represents one iteration over the entire dataset (everything put into the training model). Batch - Refers to when we cannot pass the entire dataset into the neural network at once, so we divide the dataset into several batches. Iteration - if we have 10,000 images as data and a batch size of 200. Then an epoch should run 50 iterations (10,000 divided by 50)

$$\frac{a-y}{a(1-a)} - \lambda w$$

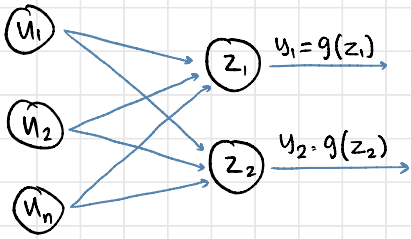
Simple Perceptron



$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

Multi Output Perceptron

↳ As all inputs are densely connected to all outputs these layers called dense layers



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$$

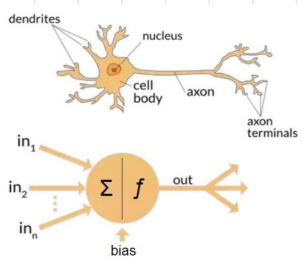
Dense layer from scratch

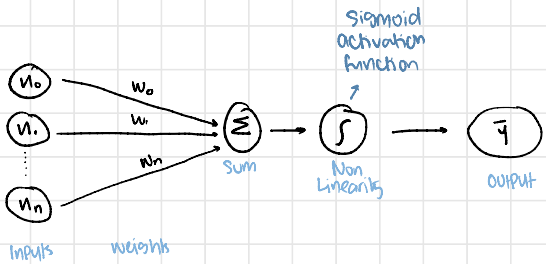
```
class MyDenseLayer(tf.keras.layers.Layer):
  def __init__(self, input_dim, output_dim):
    super(MyDenseLayer, self).__init__()

    # Initialize weights and bias
    self.W = self.add_weight([input_dim, output_dim])
    self.b = self.add_weight([1, output_dim])

  def call(self, inputs):
    # Forward propagate the inputs
    z = tf.matmul(inputs, self.W) + self.b

    # Feed through a non-linear activation
    output = tf.math.sigmoid(z)

    return output
```

NEURAL NETWORKS

Structure of Neuron / Perceptron

→ mathematical model of a neuron





Artificial Neural Network

→ tries to mimic our understanding of a functioning brain

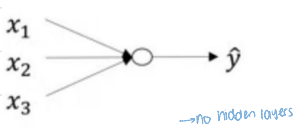
↳ an adaptive system → Parameters can be changed during training to suit the problem

↳ a parallel system which is capable of resolving paradigms that linear computing cant resolve

L Layered NN

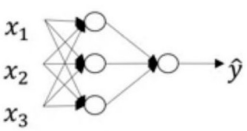
↳ no. of hidden layers + output layer

1 layered NN



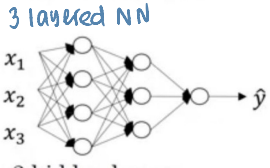
logistic regression

2 layered NN



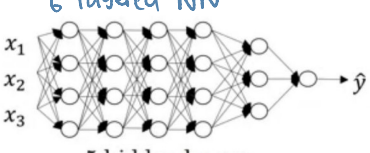
1 hidden layer

3 layered NN



2 hidden layers

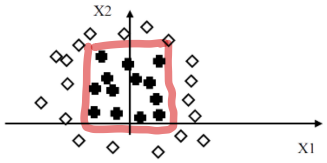
6 layered NN



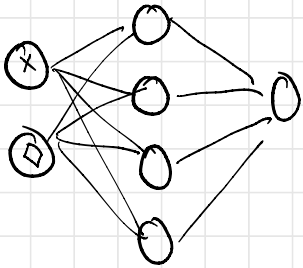
5 hidden layers

Question # 2 **[CLO 1](2 marks)**

Suppose that we want to build a neural network that classifies two-dimensional data (i.e., X = [x1, x2]) into two classes: diamonds and crosses. We have a set of training data that is plotted as follows:



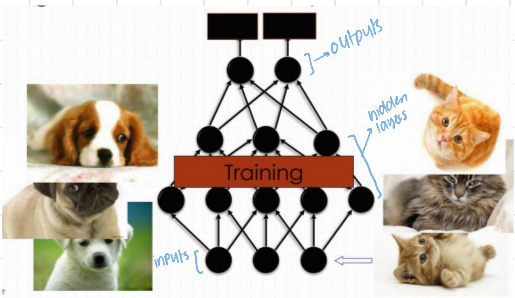
Draw a network that can solve this classification problem. Justify your choice of the number of nodes and the architecture.

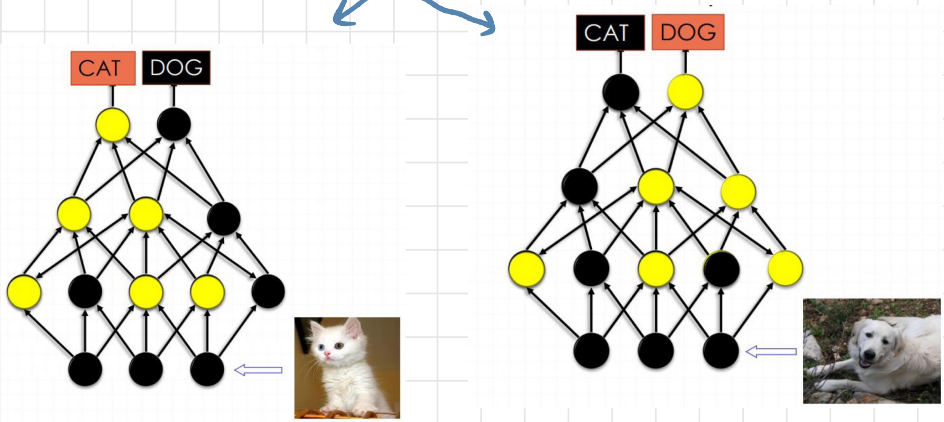


1. **Input Layer:**
   - **2 neurons** (one for each feature x_1 and x_2).
2. **Hidden Layer:**
   - **4 neurons** (to capture the decision boundary).
   - The hidden neurons form **4 linear boundaries**, defining a **square-like decision region** around the crosses.
   - **Activation function: ReLU** (Rectified Linear Unit) is commonly used.
3. **Output Layer:**
   - **1 neuron** (for binary classification).
   - **Activation function: Sigmoid** ($\sigma(x) = \frac{1}{1+e^{-z}}$) to produce a probability output.

Deep Learning

↳ a set of machine learning algos based on multi-layer networks





Shallow NN

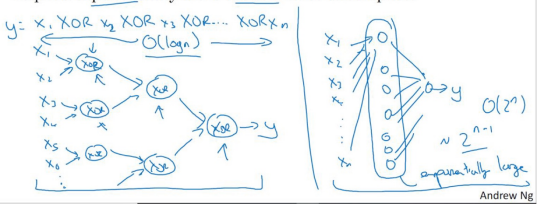
↳ consists of 1 or 2 layer NN

e.g Logistic Regression

↳ it underfits the data

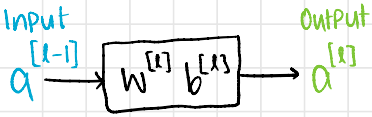
Circuit theory and deep learning

Informally: There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.



Andrew Ng

Forward pass throug layer l

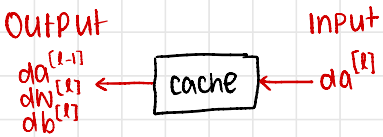


$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

$a^{[l]} = g(z^{[l]})$ an activation function in the layer l

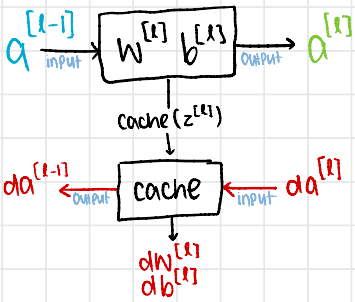
Backward pass

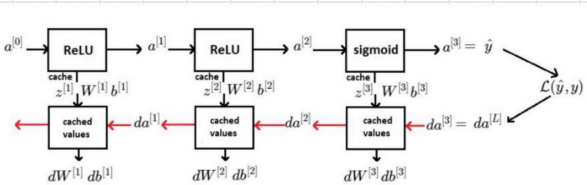
always drawn in red



Hit is good to cache the value of $z^{[l]}$ for calculations in backward pass

Forward pass and Backward pass of Layer L





$$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$$

$$b^{[l]} := b^{[l]} - \alpha db^{[l]}$$

$$da^{[L]} = -\frac{y}{a} + \frac{1-y}{1-a}$$

Multilayer Backpropogation

- NN learning for classification/numerical prediction using backpropogation algo
- Inputs
  - a data set "D"
    - training tuples
    - their associated target values
  - the learning rate "l"
  - a multilayer feed forward network
- Output
  - a trained NN

Weights Initialization

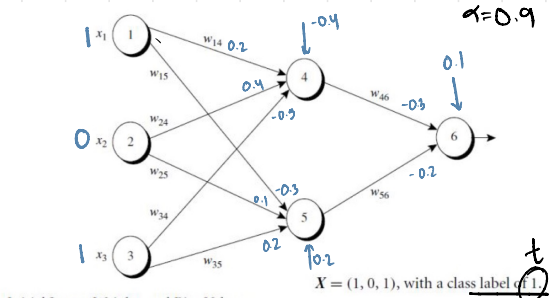
- they are initialized to small random no.
- each unit has a bias associated to it
  - biases also initialized to small random no.

Convergence condition

- Terminating condition
- Training stops when
  - all the Δw_{ij} in the prev epoch are below a specified threshold

  OR
  - The % of tuples misclassified in the prev epoch is below some threshold
  - a prespecified no. of epoch has expired

Worked Example → No loss function Given



Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Output layer

$$error_j = O_j(1-O_j)(t-O_j)$$

Hidden layer

$$error_j = O_j(1-O_j)(e_j)(w_{jk})$$

$$w^t = w + \alpha(e_j \cdot O_i)$$

↳ Feed Forward

	O
$Z_4 = 1(0.2) - 0.5 - 0.4 = -0.7$	0.332
$Z_5 = 1(-0.3) + 1(0.2) + 0.2 = 0.1$	0.525
$Z_6 = 0.332(-0.3) + 0.525(-0.2) + 0.1 = -0.105$	0.474

$\frac{1}{1+e^{-(-0.7)}}$

↳ Calculation of error at each Node

↳ Output layer 6

$error_j = O_j(1-O_j)(t-O_j)$ (O_j: a_6; t: y; O_j: a_6)

$= 0.474(1-0.474)(1-0.474)$

$= 0.1311$

↳ Hidden layer 4

w_{46}

$error_j = O_j(1-O_j)(e_j)(w_{jk})$ (O_j: a_4; e_j: 6; w_{jk}: a_4)

$error_j = 0.332(1-0.332)(0.1311)(-0.3)$

$= -0.0087$

↳ Hidden layer 5

$error_j = O_j(1-O_j)(e_j)(w_{jk})$

$error_j = 0.525(1-0.525)(0.1311)(-0.2)$

$= -0.0065$

↳ Calculation for weight and biasness

$w_{46}^t = -0.3 + 0.9(0.1311)(0.332) = -0.261$

(w_{46}; α; 6; a_4)

3

Calculations for Weight and Bias Updating

Weight or Bias	New Value
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

Classification of unknown datapoint (x)

↳ Input X into the trained NN

↳ Compute the net input and ouput of each unit → no backpropagation needed

  ↳ If multiple output nodes → one per class

    ↳ the node with highest output value determines the predicted class

  ↳ If one output node

    ↳ Output ≥ 0.5 → positive class

    ↳ Output < 0.5 → negative class

Critique of ANN

- **Parameter tuning**: Has a no of parameters that requires empirical determination of parameters like network structure. → trial and error

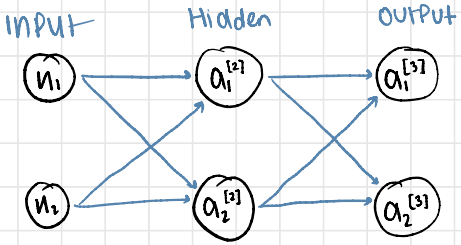
Cons

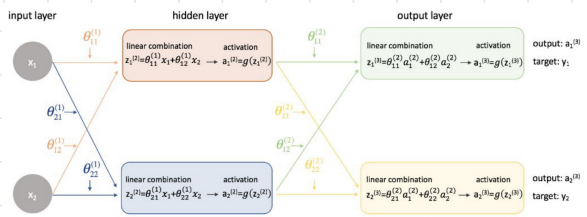
- **Long training times**: Suitable for applications where long training is feasible.
- **Poor interpretability**: Difficult to understand the meaning behind learned weights and hidden units, making them less desirable for data mining initially.

Adv of ANN

- **Noise tolerance**: Can handle noisy data well.
- **Generalization**: Can classify patterns not seen during training.
- **Flexibility**: Useful when relationships between attributes and classes are unknown.
- **Continuous data**: Suitable for continuous-valued inputs and outputs (unlike most decision trees).
- **Real-world success**: Effective in applications like handwritten character recognition, medical diagnosis, and text pronunciation.
- **Parallel processing**: Neural networks are inherently parallel, allowing for faster computation.
- **Non-linear tasks**: Can perform tasks that linear classifiers cannot.
- **Function approximation**: Multilayer feed-forward networks can approximate any function given enough hidden units and training data.

Neural Network with Multi Output





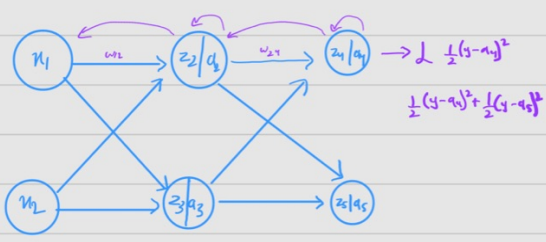
dw_{46}

NN with Multi Output

$$\frac{dL}{dw_{12}} = \frac{dL}{da_4} \cdot \frac{da_4}{dz_4} \cdot \frac{dz_4}{da_2} \cdot \frac{da_2}{dz_2} \cdot \frac{dz_2}{dw_{12}}$$

$$+$$

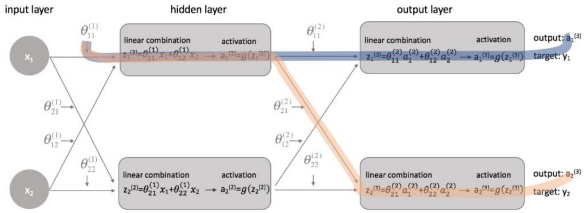
$$\frac{dL}{da_5} \cdot \frac{da_5}{dz_5} \cdot \frac{dz_5}{da_2} \cdot \frac{da_2}{dz_2} \cdot \frac{dz_2}{dw_{12}}$$



$$\frac{\delta L}{\delta w_{14}} = \frac{\delta L}{\delta a_6} \cdot \frac{\delta a_6}{\delta z_6} \cdot \frac{\delta z_6}{\delta a_4} \cdot \frac{\delta a_4}{\delta z_4} \cdot \frac{\delta z_4}{\delta w_{14}}$$

$$\underbrace{-(y-a_6)\,a_6(1-a_6)\,a_4}_{\delta L / w_{46}} \cdot \underbrace{a_4(1-a_4)}_{=} \underbrace{(1)}_{\delta z_4}$$

$$\frac{\delta L}{w_{35}} \cdot -(y-a_6)\,a_6(1-a_6)\,w_{56} \cdot a_5(1-a_5)(n_3)$$



The derivative chain for the blue path is:

$$\left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)$$

The derivative chain for the orange path is:

$$\left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)$$

Combining these, we get the total expression for $\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}}$.

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)$$

Layer 2 Parameters

$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}}\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \theta_{12}^{(2)}}\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \theta_{21}^{(2)}}\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(2)}} = \left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \theta_{22}^{(2)}}\right)$$

Layer 1 Parameters

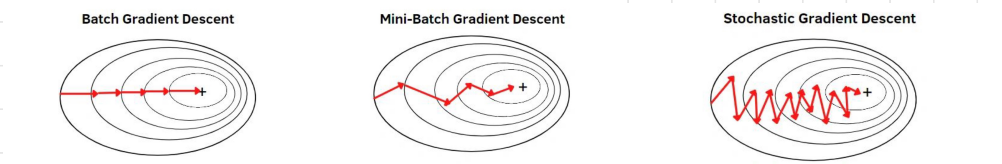
$$\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} = \left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{12}^{(1)}} = \left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}\right)\left(\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}}\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{21}^{(1)}} = \left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial a_2^{(2)}}\right)\left(\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{21}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial a_2^{(2)}}\right)\left(\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{21}^{(1)}}\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_{22}^{(1)}} = \left(\frac{\partial J(\theta)}{\partial a_1^{(3)}}\right)\left(\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial a_2^{(2)}}\right)\left(\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{22}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial a_2^{(3)}}\right)\left(\frac{\partial a_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial a_2^{(2)}}\right)\left(\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{22}^{(1)}}\right)$$

Gradient Descent Variations



1. Batch Gradient Descent

↳ aka vanilla Gradient Descent

↳ It wont update weights
  until every training sample has been assessed

CON
↳ It is slow

↳ entire procedure is called cycle and training epoch

2. Stochastic Gradient Descent (SGD)

↳ doesn't wait for anyone

↳ updates weights after every sample
  eg 100 samples ⟶ 100 updations

PROS
↳ It is faster than batch

↳ Regular updates gives an accurate
  idea of weights improvement

CONS
↳ can produce noisy gradients
  which can cause error rate to fluctuate
  rather than gradually go down

3. Mini Batch Gradient Descent

↳ divides dataset into batches

↳ updates after a batch
  eg updates after every 10 samples

↳ Merges batch and SGD

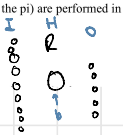
PROS
↳ It strikes a balance b/w
  ↳ Batch's effectiveness
  ↳ SGD's durability

A standard feed-forward neural network is trained by Mini Batch Gradient Descent. The network has 9 input neurons, 5 output neurons and a hidden layer, H. Hidden layer use relu activation while output layer uses softmax activation function. The network is fully connected, but only the hidden layer uses biases. The network has a total of 300 trainable parameters, pi ∈ P. During a particular session, it performs the forward and backward processes of on a small batch size of 40 samples out of total 200 observations.

1. How many nodes are in hidden layer H?
2. How many total internal calculations of gradients of the form ∂L/ ∂pi are performed in one mini batch where pi is any trainable parameter?
3. How many total parameter updates (of the pi) are performed in a single mini batch?

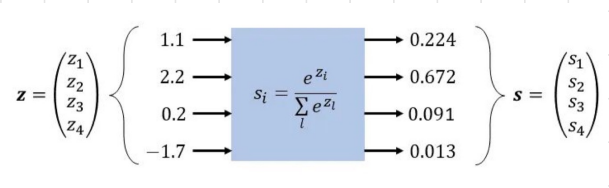
Solution:

1. 9H + H + 5H = 300, so H = 20
2. 300 x 40 = 12,000
3. 300, each pi one time



Softmax Activation Function for Multiclass

$$\frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$



SoftMax Activation Function for MultiClass - Derivative

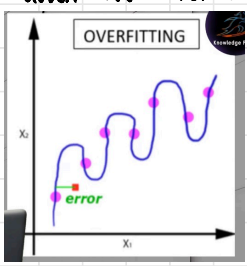
$$J_{softmax} = \begin{pmatrix} \frac{\partial s_1}{\partial z_1} & \frac{\partial s_1}{\partial z_2} & \cdots & \frac{\partial s_1}{\partial z_n} \\ \frac{\partial s_2}{\partial z_1} & \frac{\partial s_2}{\partial z_2} & \cdots & \frac{\partial s_2}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial s_n}{\partial z_1} & \frac{\partial s_n}{\partial z_2} & \cdots & \frac{\partial s_n}{\partial z_n} \end{pmatrix}$$

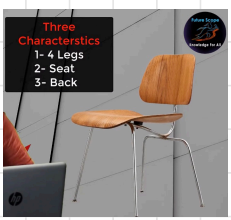
$$J_{softmax} = \begin{pmatrix} s_1 \cdot (1 - s_1) & -s_1 \cdot s_2 & -s_1 \cdot s_3 & -s_1 \cdot s_4 \\ -s_2 \cdot s_1 & s_2 \cdot (1 - s_2) & -s_2 \cdot s_3 & -s_2 \cdot s_4 \\ -s_3 \cdot s_1 & -s_3 \cdot s_2 & s_3 \cdot (1 - s_3) & -s_3 \cdot s_4 \\ -s_4 \cdot s_1 & -s_4 \cdot s_2 & -s_4 \cdot s_3 & s_4 \cdot (1 - s_4) \end{pmatrix}$$

REGULARIZATION

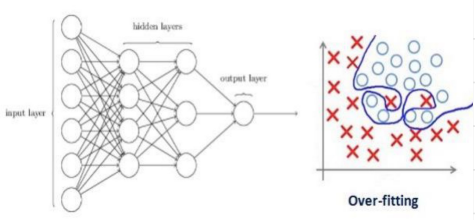
Overfitting

- learns training data too well
- training accuracy high
- testing accuracy too low
- doesnt work well on unseen data



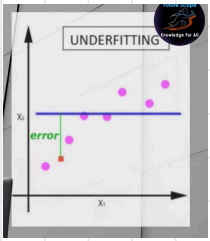


Learns more features than req.
might learn diameter and material of chair
so wont classify other chairs as a chair



Underfitting

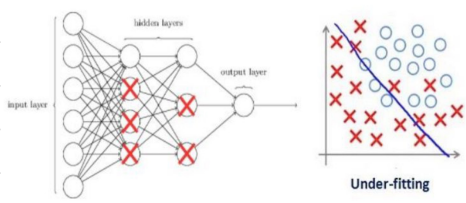
- training accuracy too low
- high training error

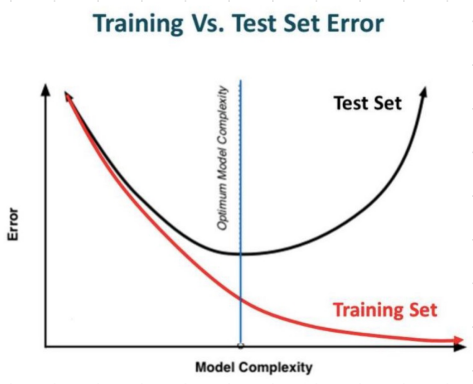


might only learn 1 feature i.e legs
so will classify dog as a chair too

Solution

- increase the length of parameters/weights
- train longer





Regularization

↳ techniques that impove NN's generalization ability by modifying learning algo

↳ improves models performance

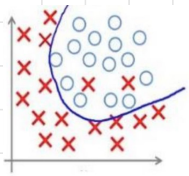
↳ Penalizes parameters that cause overfitting or underfitting

In ML it penalizes co-efficients

In DLP it penalizes weights

Assume that our regularization coefficient is so high that some of the weight matrices are nearly equal to zero

This will result in a much simpler linear network and slight underfitting of the training data.



Appropriate-fitting

Such a large value of the regularization coefficient is not that useful

We need to optimize the value of regularization coefficient in order to obtain a well-fitted model as shown in the image below

Regulization Techniques

↳ L1 L2 Regulization → Penalizes parameters that cause overfitting → reduce model complexity

↳ Data Augmentation → Get more training data

↳ Early stopping → stops training before performance goes down

↳ Dropout → randomly disables a fraction of neurons during each forward

L1 Regularization

Lasso Regularization

↳ adds absolute value of magnitude of the coefficient to loss function

so if a feature doesn't serve any purpose it penalizes the weights to approx equal to '0'

↳ feature selection

Modified loss function = Loss function + $\lambda \sum_{i=1}^{n} |w_i|$

$$E = \frac{1}{2} * \sum (t_k - o_k)^2 + \lambda * \sum |w_i|$$

squared error | L1 weight penalty

$$\frac{\partial E}{\partial w_{jk}} \text{ gradient}$$

$$\Delta w_{jk} = -1 * \eta * [x_j * (o_k - t_k) * o_k * (1 - o_k)] \pm \lambda]$$

learning rate | signal

$$w_{jk} = w_{jk} + \Delta w_{jk}$$

L2 Regularization

Ridge Regularization

↳ adds square magnitude of the coefficient to loss function

↳ shrinks parameter by spreading errors along all the terms

↳ reduces overfitting

Modified loss function = Loss function + $\lambda \sum_{i=1}^{n} w_i^2$

$$E = \frac{1}{2} * \sum (t_k - o_k)^2 + \frac{\lambda}{2} * \sum w_i^2$$

plain error | weight penalty

elegant math | simple math

$$\frac{\partial E}{\partial w_{jk}} \text{ gradient}$$

$$\Delta w_{jk} = \eta * [x_j * (o_k - t_k) * o_k * (1 - o_k)] + [\lambda * w_{jk}]$$

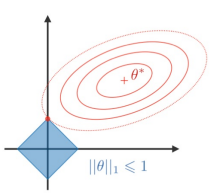
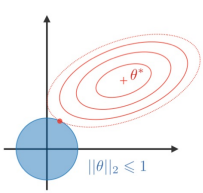
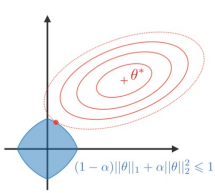
learning rate | signal

L1 L2 Regularization

Elastic Net Regularization

↳ combination of L1 and L2

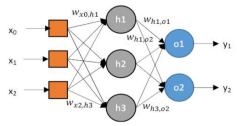
↳ adds an extra hyperparameter that controls ratio of L1 and L2 regularization

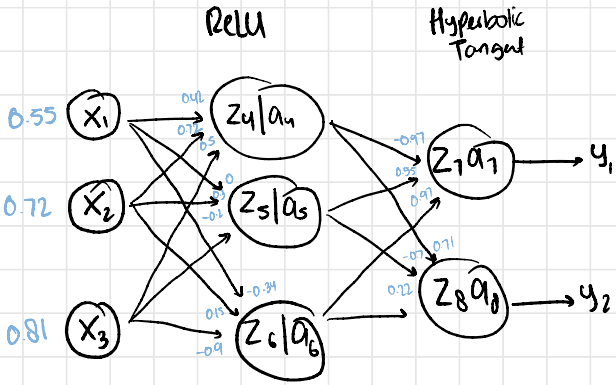
LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$\ldots + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}, \quad \alpha \in [0,1]$

Q3) Learning Rate = 0.5
Regularization Coefficient = 0.03

Q3: [1+1+4+3=9 marks]

Consider the following feed forward neural network:
Consider the given two-layer architecture, where each node in the layer L1 applies ReLU non linearity and L2 uses the Hyperbolic Tangent activation function, for simplicity the bias unit is not considered for the given network.
X = [[0.55], [0.72],[0.81]]
weights of L1
W1 = [[0.42, 0.72,0.5],[0. , 0.3 , -0.2], [-0.34,0.15, -0.9]]
weights of L2
W2 = [[-0.97, 0.55, 0.97],[0.71, -0.7, 0.22]]
Learning Rate = 0.5.
Regularization Coefficient= 0.03





Feed Forward

$Z_4 = (0.55)(0.42) + (0.72)(0.72) + (0.81)(0.5) = 1.1544$ $\quad a_4 = 1.1544$

$Z_5 = (0.55)(0) + (0.72)(0.3) + (0.81)(-0.2) = 0.054$

$Z_6 = (0.55)(-0.34) + (0.72)(0.15) + (0.81)(-0.9) = -0.808$

$Z_7 =$

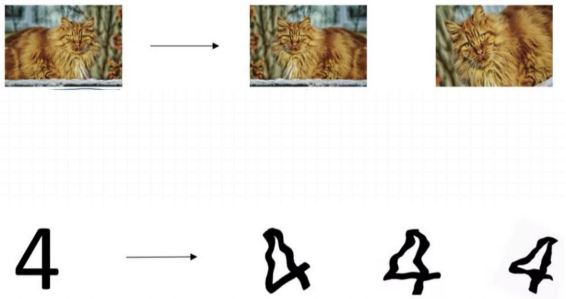
1. Compute the output of each node in the network for forward propagation.
   Z(h1)=1.1544➔ a(h1)=1.1544
   Z(h2)=0.116➔ a(h2)=0.116
   Z(h3)=-0.4848➔ a(h2)=0
   Z(o1)=-1.0559➔ a(o1)=-0.784
   Z(o2)=-0.7384➔ a(o2)=0.63819
2. Compute the error if y= [-0.98, 0.75]
   E= ½ ((-0.784+0.98)^2 + (0.63819-0.75)^2)=0.02545
3. Perform backpropagation, and compute the derivatives i.e. dWh1O1, dWh2O2, dWx0h1. Calculate the updated weights.
   D h1o1=-(-0.98+0.784)(1-0.784^2) 1.1544 = 0.0871
   Wh1o1= -0.97 - 0.5*0.0871=-1.01355
   D h2o2=-(0.75-0.63819)(1-0.63819^2) 0.116= -0.00768
   Wh2o2= -0.7- 0.5*-0.00768=-0.69616
   D X0h1=-(-0.98+0.784)(1-0.784^2) (-0.97)(1)(0.55) + (-(0.75-0.63819)(1-0.63819^2) (0.71)(1)(0.55))
   D X0h1=-0.0662
   W X0h1= 0.42- 0.5*-0.0662=0.4531
4. Perform backpropagation with **L1 regularization** and compute the derivatives for dWh2O1, dWh3O2. Calculate the updated weights.
   D h2o1=-(-0.98+0.784)(1-0.784^2) 0.116 + 0.03 = 0.03876
   Wh2o1= 0. - 0.5*0.03876=-0.01938
   D h3o2=0 + 0.03 = 0.03
   Wh3o2= 0.22 - 0.5*0.03=0.205

$$E = \frac{1}{2}(y - \hat{y})^2 \cdot \lambda$$

$$w_{12}^{+} = w_{12} - \alpha\left(\frac{\delta L}{\delta w_{12}} + \lambda\right)$$

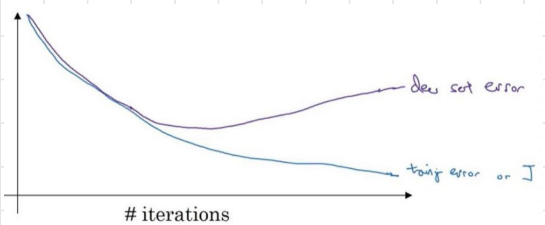
Data Augmentation

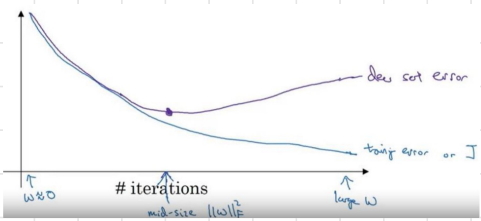
- synthesizing data to increase training data by
  - resiz
  - Horizontal/Vertical flip
  - Rotate
  - Add noise
  - Deform
  - Modify colors



Early stopping

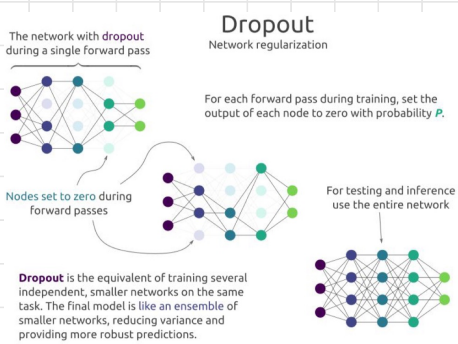
- stops training when performance on the validation set starts to degrade





Dropout

- forgets 20% — 50% of the learning
- randomly disables a fraction of neurons during each forward pass
- on backward pass only update those weights
  - it reduces co-dependence among features



ishma hafeez notes

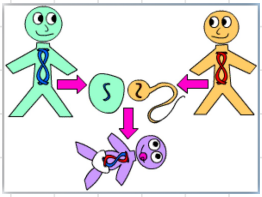
DROPOUT

Motivation

- NN can sometimes overfit
- solution is to use Bayesian Framework
  - but not feasible if network too big
- we need something faster

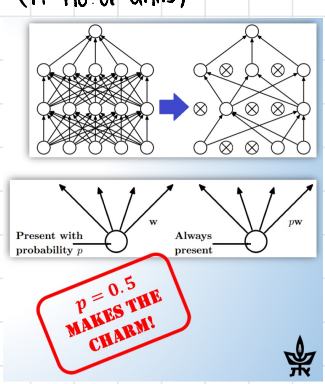
Inspiration

- Genes work well another random set of genes
- Similarly, dropout suggests that each unit should work with a random set of other samples



Dropout

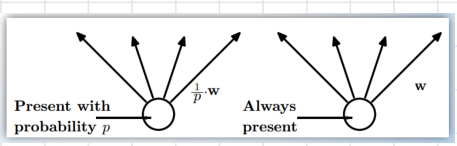
- At each training iteration
  - each unit is retained with a probability p
- At test
  - the network is used as a whole
  - the weights are scaled down by a factor of p
- In practice
  - dropout trains 2^n networks (n - no of units)



Dropout - an equivalent method

- At training
  - weights are scaled up by a factor of $1/p$
- A test
  - no scalling is applied
- This method is used in Tensorflow

  tf.nn.dropout(x, keep_prob=p)



Why apply dropout on units?

↳ with all their in and outgoing arcs

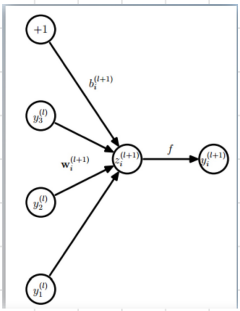
and not just on the arc's themselves

Dropout is applied to entire neurons (units) rather than individual connections (arcs) to prevent co-adaptation, encourage independent feature learning, and improve generalization.
Dropping full neurons forces the network to learn redundant and distributed representations, making it more robust.
It also simplifies implementation and has been empirically shown to work better.

Dropout-Model Description

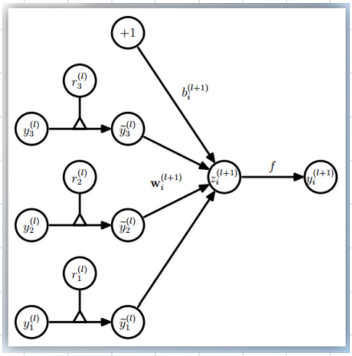
Feed Forward w/o Dropout

$$z_i^{(l+1)} = w_i^{(l+1)} y^{(l)} + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right)$$



Feed Forward with Dropout

$$r_i^{(l)} \sim \text{Bernoulli}(p)$$
$$\tilde{y}^{(l)} = r^{(l)} \cdot y^{(l)}$$
$$z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^{(l)} + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right)$$



Backpropogation with Dropout

↳ is only performed on thinned network

for each training iteration

↳ At training

↳ a case which does not use a parameter

contributes a gradient of '0' for that parameter

↳ A test

↳ the weights are scaled as:

$$w_{test}^{(l)} = p \cdot w^{(l)}$$

Experimental Results

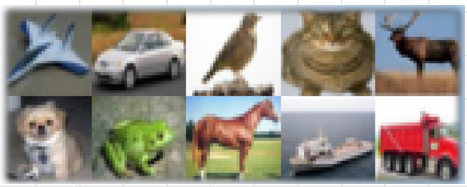
1. Street View House Numbers (SVHN)

- Dropout is also applied to convolution layers
- all hidden units are ReLUs

Method	Error %
Binary Features (WDCH) (Netzer et al., 2011)	36.7
HOG (Netzer et al., 2011)	15.0
Stacked Sparse Autoencoders (Netzer et al., 2011)	10.3
KMeans (Netzer et al., 2011)	9.4
Multi-stage Conv Net with average pooling (Sermanet et al., 2012)	9.06
Multi-stage Conv Net + L2 pooling (Sermanet et al., 2012)	5.36
Multi-stage Conv Net + L4 pooling + padding (Sermanet et al., 2012)	4.90
Conv Net + max-pooling	3.95
Conv Net + max pooling + dropout in fully connected layers	3.02
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	2.80
Conv Net + max pooling + dropout in all layers	2.55
Conv Net + maxout (Goodfellow et al., 2013)	**2.47**
Human Performance	2.0

2. CIFAR-10 & CIFAR-100

- images drawn from 10 and 100 categories respectively

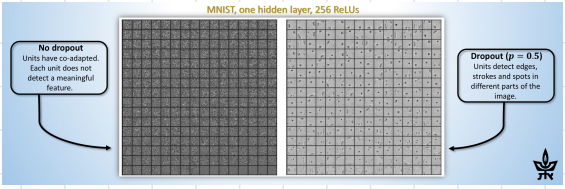


Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

Effect of Dropout on

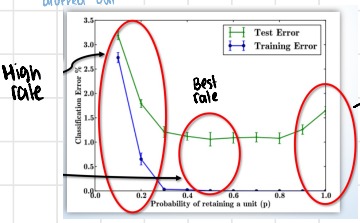
1. Learned Features

- w/o dropout
  - units tend to compensate for mistakes of other units
  - leads to overfitting
    - as these co-adaption do not generalize to unseen data
- with dropout
  - reduced overfitting
    - as co-adaption are prevented by making presence of other hidden layer unreliable



2. Dropout rate P

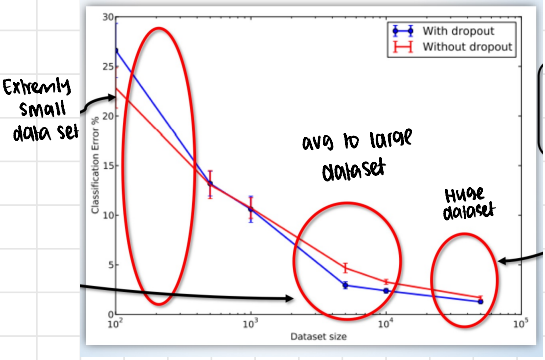
- An architecture of ..... is used on MINST dataset
- The dropout rate P is changed from small numbers (most units are dropped out) to 1.0 (no dropout)



- High rate of dropout (P<0.3)
  - Training error high → Underfitting
  - very few units are turned on during training
- Best rate of dropout (P = 0.5)
  - Training error low
  - Test error is low
  - → Best
- NO dropout (P = 1.0)
  - Training error low
  - Test error is high → Overfitting

3. Dataset Size

- An architecture of ..... is used on MINST dataset
- Extremly small data set
  - dropout does not improve error rate
    - makes it worse sometimes
- Average to large dataset
  - dropout improves error rate
- Huge dataset
  - dropout barely improves error rate
  - dataset is big enough so overfitting isn't an issue



Weight Decay

→ Pushes weights towards smaller values to prevent overfitting

↳ a regularization technique
↳ limiting the growth of weight in the network
↳ a term is added to the original loss function penalizing large weights

$$L_{new}(w) = L_{orig}(w) + \frac{1}{2}\lambda \|w\|_2^2$$

↳ An architecture of ..... is used on MINST dataset with diff regularizations

Method	Test Classification error %
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

Dropout vs Weign Decay

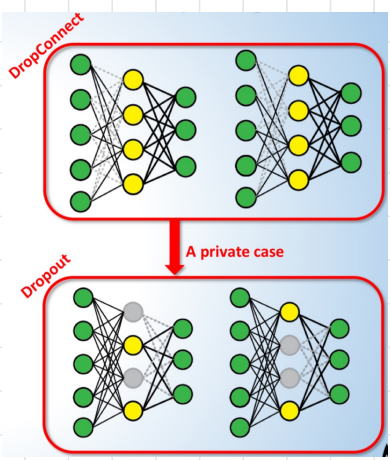
→ Its better

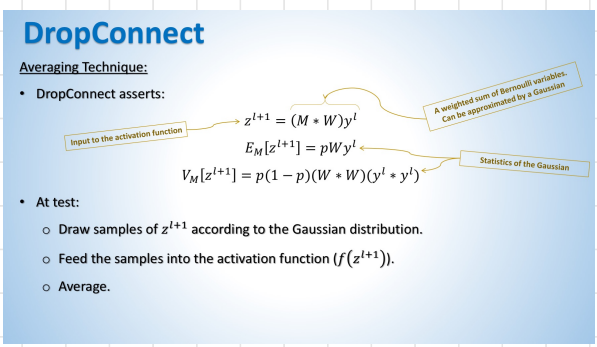
↳ Dropout Adv
↳ it is scale free
↳ it doesn't penalize the use of large weights when needed
↳ is invariant to parameter scaling
↳ It is unaffected if
↳ weights in a certain layer are upscaled by a constant C
↳ and weights in another layer are downscaled by a constant c
↳ implying dropout does not have a isolated minima

DropConnect

→ Similar to dropout but instead of dropping neuron activations it drops weights connections

↳ a regularization technique
↳ It generalizes dropout by suggesting dropping out weights
↳ In Dropout P=0.5
↳ usually gives the best result





Cons

↳ training is slower
↳ implementation is more complicated than Dropout's
↳ In their paper, it has been proven to work mostly when using more than 1 network

COMPARISION OF

1. NO-drop
2. Dropout
3. DropConnect

MNIST dataset

↳ 2 layers → 800 neurons each

MNIST

neuron	model	error(%) 5 network	voting error(%)
relu	No-Drop	1.62 ± 0.037	1.40
	Dropout	1.28 ± 0.040	1.20
	DropConnect	1.20 ± 0.034	**1.12**
sigmoid	No-Drop	1.78 ± 0.037	1.74
	Dropout	1.38 ± 0.039	**1.36**
	DropConnect	1.55 ± 0.046	1.48
tanh	No-Drop	1.65 ± 0.026	1.49
	Dropout	1.58 ± 0.053	1.55
	DropConnect	1.36 ± 0.054	**1.35**

↳ In DropConnects paper on MINST dataset

↳ DropConnet achives the lowest error rate recorded so far in MINST

↳ results are equivalent for

↳ Drop connect and no dropout

crop	rotation scaling	model	error(%) 5 network	voting error(%)
no	no	No-Drop	0.77±0.051	0.67
		Dropout	0.59±0.039	**0.52**
		DropConnect	0.63±0.035	0.57
yes	no	No-Drop	0.50±0.098	0.38
		Dropout	0.39±0.039	0.35
		DropConnect	0.39±0.047	**0.32**
yes	yes	No-Drop	0.30±0.035	**0.21**
		Dropout	0.28±0.016	0.27
		DropConnect	0.28±0.032	**0.21**

CIFAR-10

↳ 4 layers

↳ Dropout & DropConnect applied only on final layer

CIFAR-10

model	error(%)
No-Drop	23.5
Dropout	19.7
DropConnect	**18.7**

SVHN

↳ Same architecture as CIFAR-10

↳ due to large training set size all methods nearly achieve nearly same performance

SVHN

model	error(%) 5 network	voting error(%)
No-Drop	2.26 ± 0.072	**1.94**
Dropout	2.25 ± 0.034	1.96
DropConnect	2.23 ± 0.039	**1.94**

Dropout

pros

- good and fast regularization method
- if dataset is avg-large
  - it excels
- achives better results than Weight Decay

cons

- slow to train (2-3 times slower w/o dropout)
- if dataset is big enough
  - it doesn't help much

DropConnet

- is a generalization of dropout

cons

- training is slower than dropout
- it's superiority over dropout is unclear
- has complications in implementation

SUMMARY

- use a small dropout value (20% - 50% of neurons)
- a probability too
  - low → has minimal effect
  - high → under-learning by network
- use a larger network → dropout than given better performance
- use dropout on incoming (visible) and visible units
- use of dropout at each layer → has good results
- use large learning rate with decay (increase by factor of 10 - 100) and large learning momentum (0.9/0.99)
- constrain the size of network weights
  - large learning rate → large network weights
- imposing a constraint on the size of network weights → improves results