Natural Language Processing (CS-472) Spring-2023

Muhammad Naseer Bajwa

Assistant Professor,
Department of Computing, SEECS
Co-Principal Investigator,
Deep Learning Lab, NCAI
NUST, Islamabad
naseer.bajwa@seecs.edu.pk

NUST
Defining futures
School of Electrical Engineering
& Computer Science

Overview of this week's lecture

Review of Neural Networks

- Recap of fundamentals of neural networks
- Forward and backpropagation in MLPs
- Activation functions
- Improving network training

NUST
Defining futures
School of Electrical Engineering
& Computer Science

Let’s establish some notation for neural networks

- In supervised learning, the training dataset consists of input/output pairs.

$$\{\boldsymbol{x_i}, y_i\}, for\ i = 1, \dots, N$$

  - In NLP, $\boldsymbol{x_i}$ are inputs representing words (indices or vectors), sentences, or documents etc.
  - Inputs have the dimension d, representing number of (raw) features.
  - y_i denotes labels (one type/class of word from C classes).

Let's establish some notation for neural networks

- In supervised learning, the training dataset consists of input/output pairs.

$$\{\boldsymbol{x_i}, y_i\}, for\ i = 1, \dots, N$$

  - In NLP, $\boldsymbol{x_i}$ are inputs representing words (indices or vectors), sentences, or documents etc.
  - Inputs have the dimension d, representing number of (raw) features.
  - y_i denotes labels (one type/class of word from C classes).
- The task of Deep Neural Networks (DNNs) applied to NLP is normally prediction for,
  - Classes: sentiment, Named Entity, buy/sell decision, etc.
  - Other words: Masked Natural Language Understanding.
  - Multi-word sequences: Translation, Question/Answering, etc.

Training data for a binary classification may be visualised on a 2D plane

- For the given $2D$ vector space, learn a decision boundary to separate the two classes.

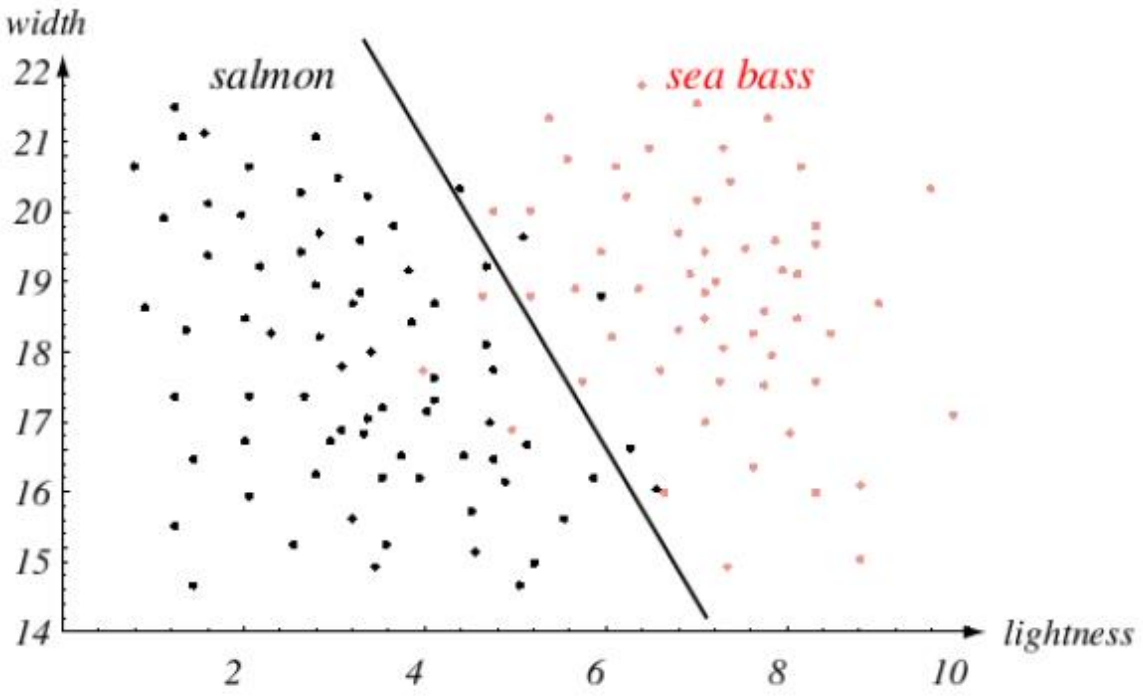


FIGURE 1.4. The two features of lightness and width for sea bass and salmon. The dark line could serve as a decision boundary of our classifier. Overall classification error on the data shown is lower than if we use only one feature as in Fig. 1.3, but there will still be some errors. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

NUST
Defining futures
School of Electrical Engineering
& Computer Science

Training data for a binary classification may be visualised on a 2D plane

- For the given $2D$ vector space, learn a decision boundary to separate the two classes.

- The Learned Line is the Classifier.

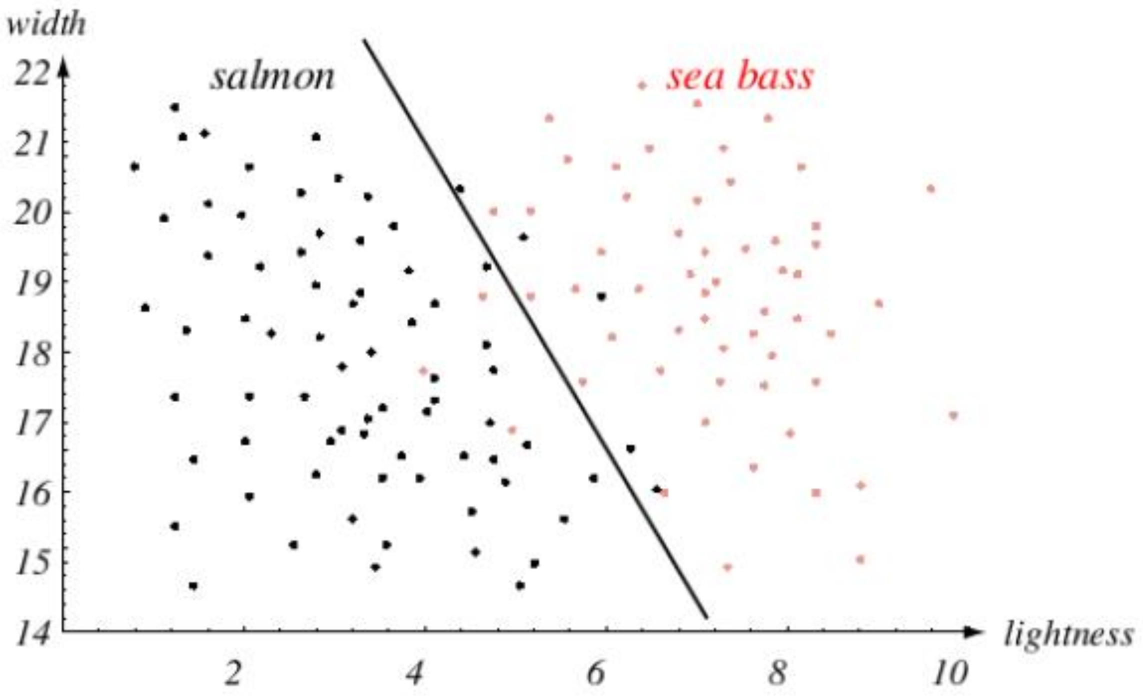


FIGURE 1.4. The two features of lightness and width for sea bass and salmon. The dark line could serve as a decision boundary of our classifier. Overall classification error on the data shown is lower than if we use only one feature as in Fig. 1.3, but there will still be some errors. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Training data for a binary classification may be visualised on a 2D plane

- For the given $2D$ vector space, learn a decision boundary to separate the two classes.

- The Learned Line is the Classifier.

- Approach?

    - Set weights $\boldsymbol{W} \in \mathbb{R}^{c \times d}$ to find a **hyperplane** the fits the data.

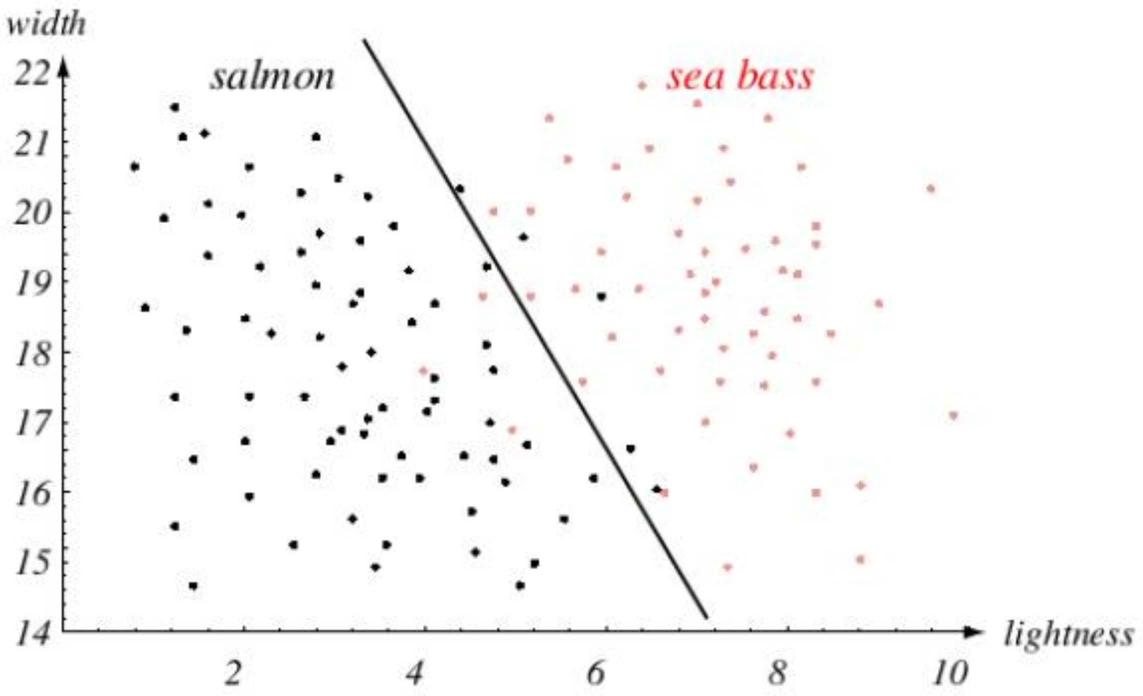


FIGURE 1.4. The two features of lightness and width for sea bass and salmon. The dark line could serve as a decision boundary of our classifier. Overall classification error on the data shown is lower than if we use only one feature as in Fig. 1.3, but there will still be some errors. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Training data for a binary classification may be visualised on a 2D plane

- For the given $2D$ vector space, learn a decision boundary to separate the two classes.

- The Learned Line is the Classifier.

- Approach?

    - Set weights $\boldsymbol{W} \in \mathbb{R}^{c \times d}$ to find a **hyperplane** the fits the data.

- Method?

  For each input x_i, predict $P(y|x)$.

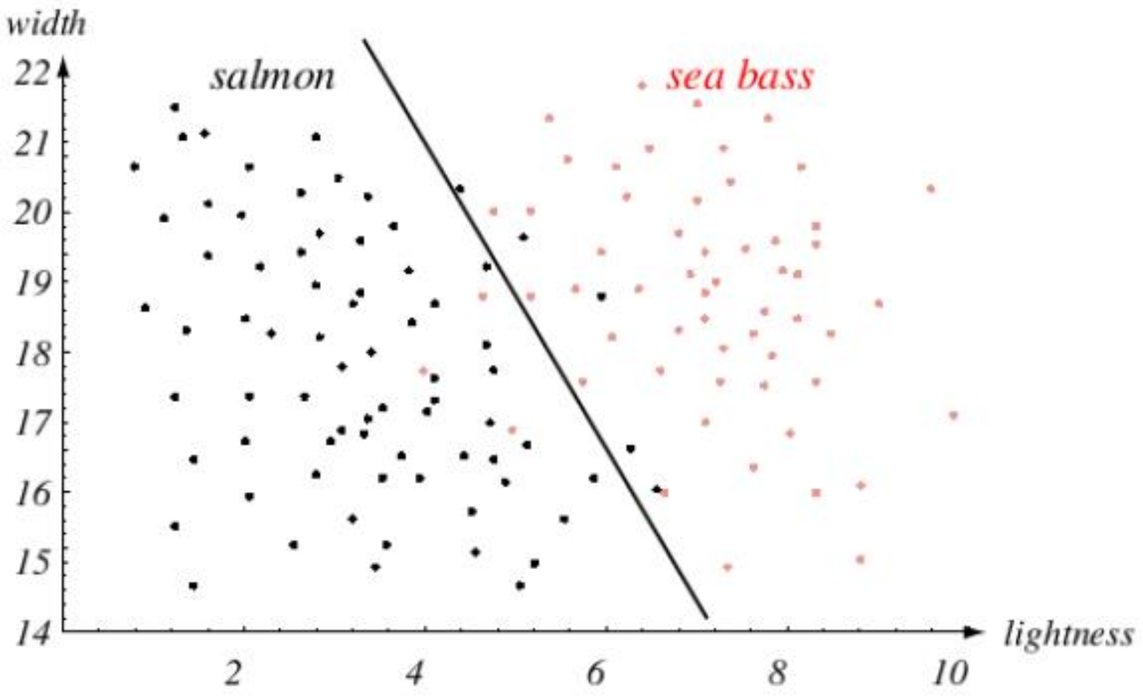
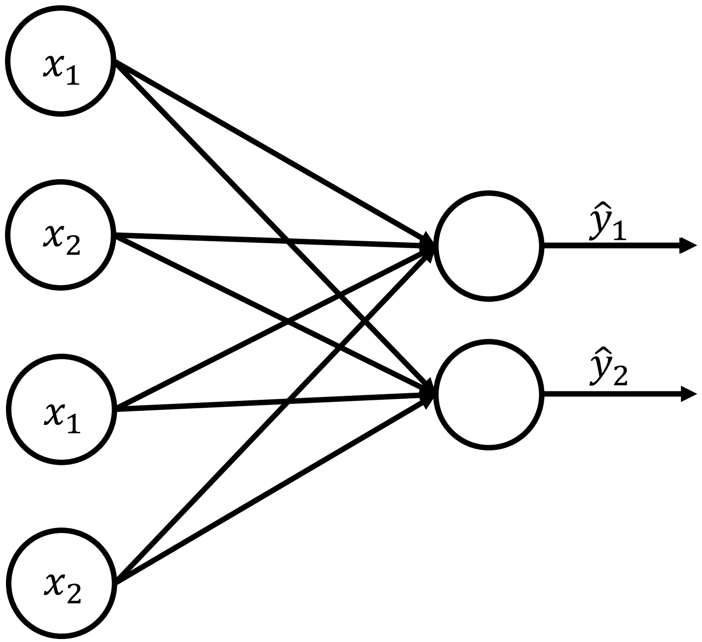


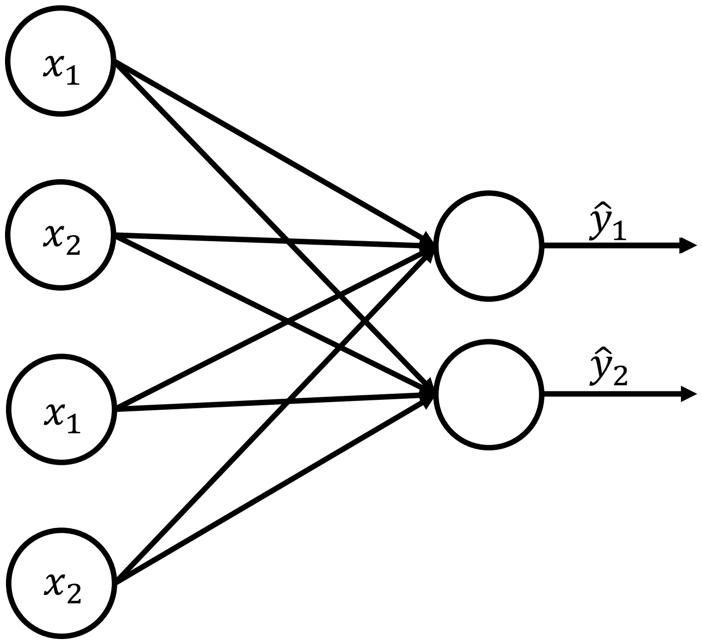
FIGURE 1.4. The two features of lightness and width for sea bass and salmon. The dark line could serve as a decision boundary of our classifier. Overall classification error on the data shown is lower than if we use only one feature as in Fig. 1.3, but there will still be some errors. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

How to calculate $P(y|x)$?



How to calculate $P(y|x)$?

- We have a set of inputs that may assume any real number.
-
    - These real numbers represent **likeness score**.

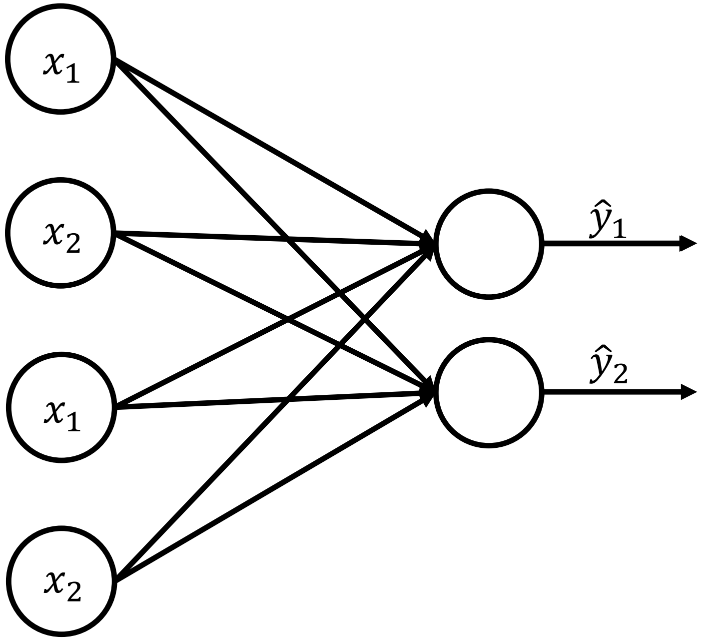


How to calculate $P(y|x)$?

- We have a set of inputs that may assume any real number.
-
    - These real numbers represent **likeness score**.

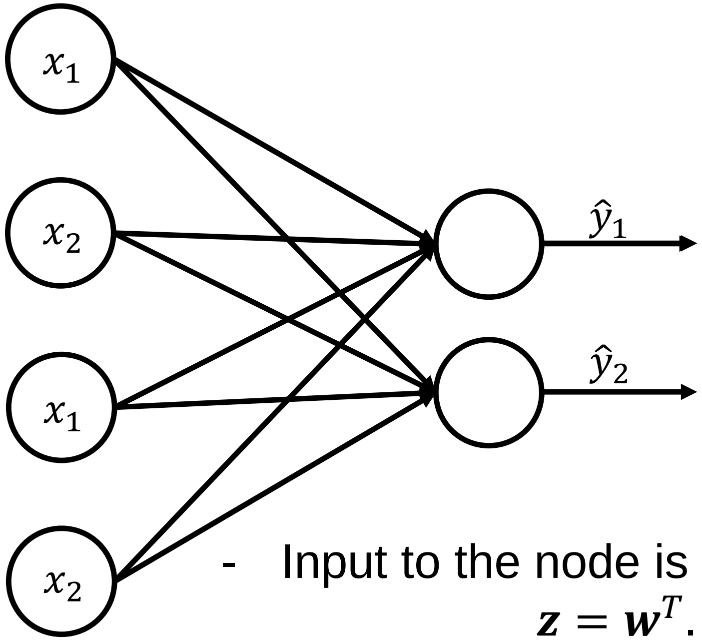
- We want an output that resembles probability.

    - All positive values
    - Within interval (0,1)
    - All values should sum up to 1.



How to calculate $P(y|x)$?

- We have a set of inputs that may assume any real number.
-
    - These real numbers represent **likeness score**.
- We want an output that resembles probability.
    - All positive values
    - Within interval (0,1)
    - All values should sum up to 1.
- First make all values positive.
    - How?



- Input to the node is $\boldsymbol{z}$, a 4×1 vector

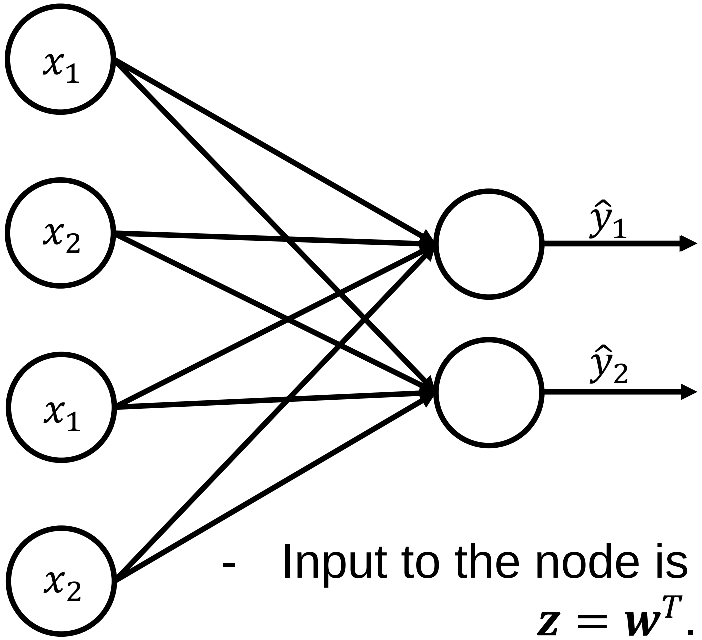
$$\boldsymbol{z} = \boldsymbol{w}^T . \boldsymbol{x}$$

How to calculate $P(y|x)$?

- We have a set of inputs that may assume any real number.
-
    - These real numbers represent **likeness score**.

- We want an output that resembles probability.
    - All positive values
    - Within interval (0,1)
    - All values should sum up to 1.

- First make all values positive.
    - How? Take exponent.



- Input to the node is $\boldsymbol{z}$, a 4×1 vector

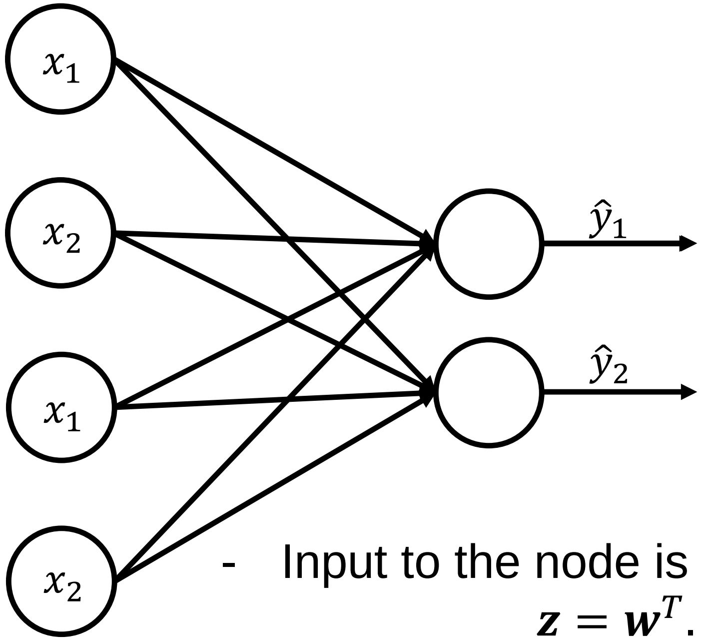
$$\boldsymbol{z} = \boldsymbol{w}^T . \boldsymbol{x}$$

- Take element wise exponent

$$\exp z_i$$

How to calculate $P(y|x)$?

- We have a set of inputs that may assume any real number.
-
    - These real numbers represent **likeness score**.
- We want an output that resembles probability.
    - All positive values
    - Within interval (0,1)
    - All values should sum up to 1.
- First make all values positive.
    - How? Take exponent.
- Normalise by sum of all values.



- Input to the node is $\boldsymbol{z}$, a 4×1 vector

$$\boldsymbol{z} = \boldsymbol{w}^T . \boldsymbol{x}$$

- Take element wise exponent

$$\exp z_i$$

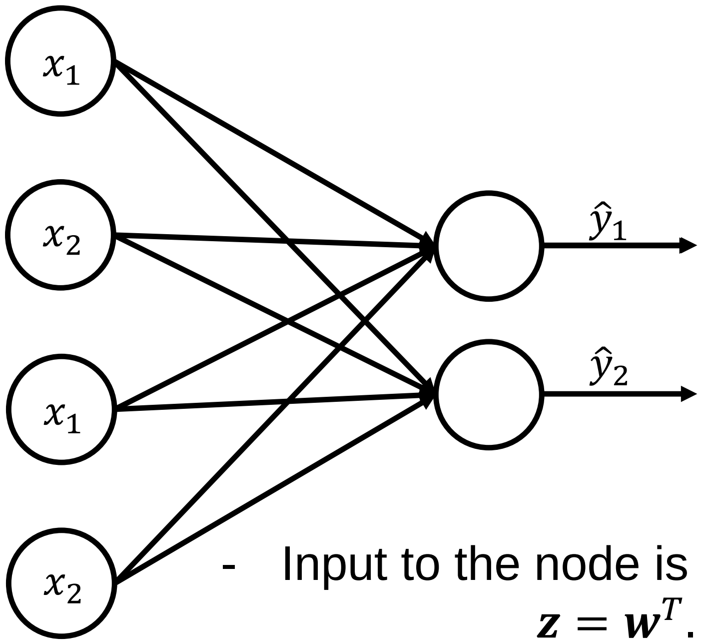
How to calculate $P(y|x)$?

- We have a set of inputs that may assume any real number.
-
  - These real numbers represent **likeness score**.

- We want an output that resembles probability.
  - All positive values
  - Within interval (0,1)
  - All values should sum up to 1.

- First make all values positive.
  - How? Take exponent.

- Normalise by sum of all values.



- Input to the node is $\boldsymbol{z}$, a 4×1 vector

$$\boldsymbol{z} = \boldsymbol{w}^T . \boldsymbol{x}$$

- Take element wise exponent

$$\exp z_i$$

- Normalise

$$= \frac{\exp z_i}{\sum_{c=1}^{C} \exp z_c}$$

NUST
Defining futures
School of Electrical Engineering
& Computer Science

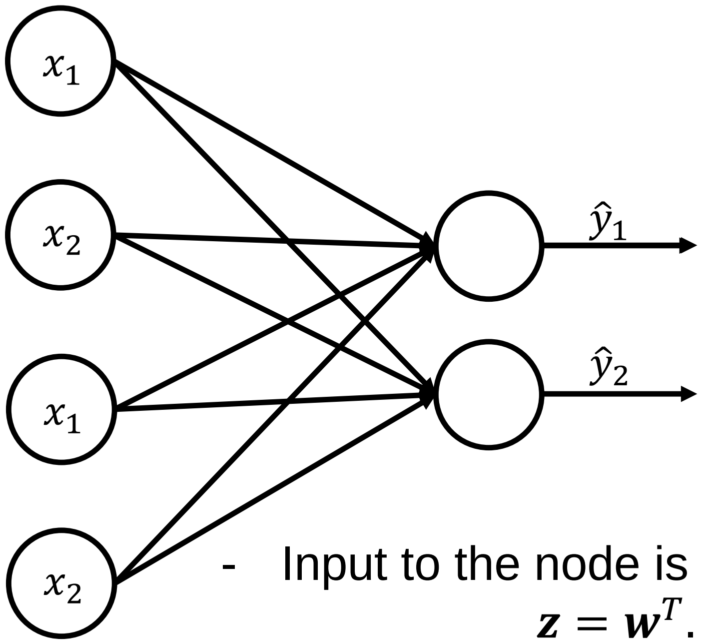
How to calculate $P(y|x)$?

- We have a set of inputs that may assume any real number.
-
    - These real numbers represent **likeness score**.

- We want an output that resembles probability.
    - All positive values
    - Within interval (0,1)
    - All values should sum up to 1.

- First make all values positive.
    - How? Take exponent.

- Normalise by sum of all values.



- Input to the node is $\boldsymbol{z}$, a 4×1 vector

$$\boldsymbol{z} = \boldsymbol{w}^T . \boldsymbol{x}$$

- Take element wise exponent

$$\exp z_i$$

- Normalise

$$softmax = \frac{\exp z_i}{\sum_{c=1}^{C} \exp z_c}$$

A simple $softmax$ classifier is a linear classifier

- Using $softmax$ alone is not very useful.

A simple $softmax$ classifier is a linear classifier

- Using $softmax$ alone is not very useful.

- Linear classifiers have limited capacity to learn complex patterns in the data.

A simple $softmax$ classifier is a linear classifier

- Using $softmax$ alone is not very useful.

- Linear classifiers have limited capacity to learn complex patterns in the data.

- Can you do multiclass classification using linear classifiers?

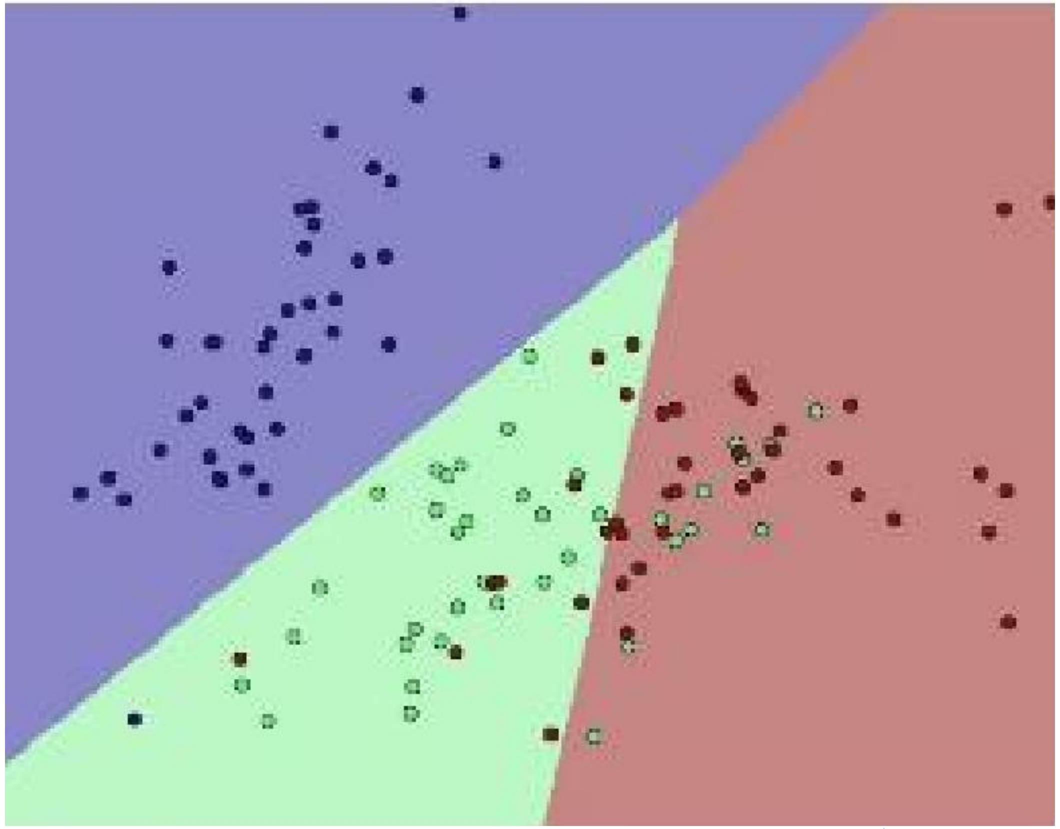
A simple $softmax$ classifier is a linear classifier

- Using $softmax$ alone is not very useful.

- Linear classifiers have limited capacity to learn complex patterns in the data.

- Can you do multiclass classification using linear classifiers?

    - Yes, you can



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

A simple $softmax$ classifier is a linear classifier

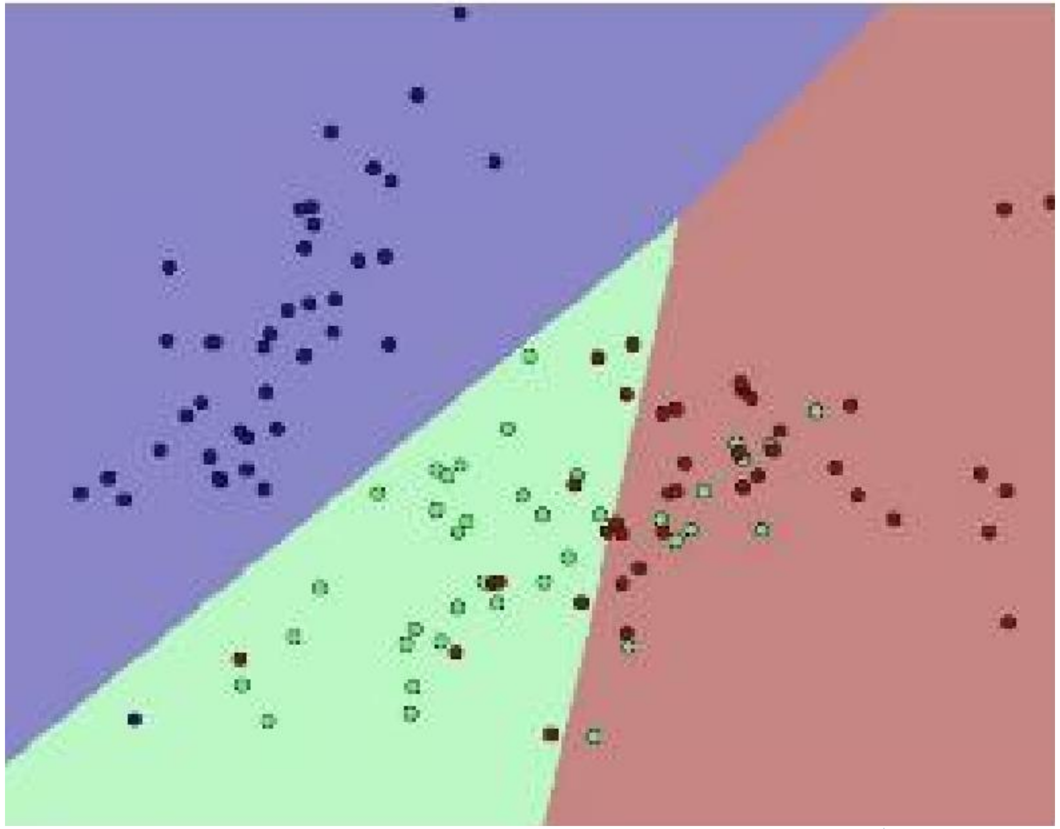
- Using $softmax$ alone is not very useful.

- Linear classifiers have limited capacity to learn complex patterns in the data.

- Can you do multiclass classification using linear classifiers?

    - Yes, you can

- Naïve Bayes, SVM, Logistic Regression are examples of linear classifiers.



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Cross Entropy loss is commonly used with $softmax$ classifier

- The objective is to maximise the probability of the correct class y or minimise the negative log probability of y.

$$-\log P(y|x) = -\log(\text{softmax})$$

Cross Entropy loss is commonly used with $softmax$ classifier

- The objective is to maximise the probability of the correct class y or minimise the negative log probability of y.

$$-\log P\,(y|x) = -\log\,(\text{softmax})$$

- The loss function here is *cross entropy*.

Cross Entropy loss is commonly used with $softmax$ classifier

- The objective is to maximise the probability of the correct class y or minimise the negative log probability of y.

$$-\log P\,(y|x) = -\log\,(\text{softmax})$$

- The loss function here is *cross entropy*.

  - Let there is a true probability distribution p and our model learns (estimates) a probability distribution q, then

$$cross\ entropy = \mathcal{L}(p,q) = -\sum_{c=1}^{C} p(c)\log q(c)$$

NUST
Defining futures
School of Electrical Engineering
& Computer Science

Cross Entropy loss is commonly used with $softmax$ classifier

- The objective is to maximise the probability of the correct class y or minimise the negative log probability of y.

$$-\log P(y|x) = -\log(\text{softmax})$$

- The loss function here is *cross entropy*.

    - Let there is a true probability distribution p and our model learns (estimates) a probability distribution q, then

$$cross\ entropy = \mathcal{L}(p,q) = -\sum_{c=1}^{C} p(c)\log q(c)$$

    - Assuming a ground truth probability distribution that is 1 at the right class, and 0 elsewhere, or

$$P_{GT} = [0,0,\dots,0,1,0,\dots,0],$$

the only term left is the negative log probability of the true class $(-\log q(c))$

Average loss over all training samples gives cost function

- Cross entropy over the whole dataset gives the cost function.

$$J(\theta) = \sum_{j=1}^{N} -\log\left(\frac{e^{z_{ij}}}{\sum_{c=1}^{C} e^{z_c}}\right)$$

Average loss over all training samples gives cost function

- Cross entropy over the whole dataset gives the cost function.

$$J(\theta) = \sum_{j=1}^{N} -\log\left(\frac{e^{z_{ij}}}{\sum_{c=1}^{C} e^{z_c}}\right)$$

- What if labels are not truly one hot encoded?

$$P_{GT} = [0,0, \dots, 0.7, 0.3, 0, \dots, 0]$$

Optimisation of networks parameters is done by calculating gradients

- Usually, θ only consists of columns of W matrix,

$$\theta = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_d \end{bmatrix} \in \mathbb{R}^{c \times d}$$

Optimisation of networks parameters is done by calculating gradients

- Usually, θ only consists of columns of W matrix,

$$\theta = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_d \end{bmatrix} \in \mathbb{R}^{c \times d}$$

- Update in decision boundary is done via,

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla w_1 \\ \nabla w_2 \\ . \\ . \\ . \\ \nabla w_d \end{bmatrix} \in \mathbb{R}^{c \times d}$$

There are more trainable parameters in NLP than in other NN applications

- What are the parameters of our model?
  - Generally, only weights and biases of the network.

There are more trainable parameters in NLP than in other NN applications

- What are the parameters of our model?
    - Generally, only weights and biases of the network.
- In NLP, the representation of words (word vectors, for instance) may also learnable.

NUST
Defining futures
School of Electrical Engineering
& Computer Science

There are more trainable parameters in NLP than in other NN applications

- What are the parameters of our model?
    - Generally, only weights and biases of the network.
- In NLP, the representation of words (word vectors, for instance) may also learnable.
- We don't only update the weights of the network but also update the values of vectors representing words.
    - Optimise both input and network parameters simultaneously.

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla w_1 \\ \nabla w_2 \\ . \\ . \\ . \\ \nabla w_d \\ \nabla x_1 \\ \nabla x_2 \\ . \\ . \\ . \\ \nabla x_V \end{bmatrix} \in \mathbb{R}^{c\times d+V\times d}$$

There are more trainable parameters in NLP than in other NN applications

- What are the parameters of our model?
  - Generally, only weights and biases of the network.
- In NLP, the representation of words (word vectors, for instance) may also learnable.
- We don't only update the weights of the network but also update the values of vectors representing words.
  - Optimise both input and network parameters simultaneously.
- It's called Representation Learning.

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla w_1 \\ \nabla w_2 \\ . \\ . \\ . \\ \nabla w_d \\ \nabla x_1 \\ \nabla x_2 \\ . \\ . \\ . \\ \nabla x_V \end{bmatrix} \in \mathbb{R}^{c \times d + V \times d}$$

There are more trainable parameters in NLP than in other NN applications

- What are the parameters of our model?
    - Generally, only weights and biases of the network.
- In NLP, the representation of words (word vectors, for instance) may also learnable.
- We don't only update the weights of the network but also update the values of vectors representing words.
    - Optimise both input and network parameters simultaneously.
- It's called Representation Learning.
- One hot encoded word vectors can be considered as another layer in the neural network.

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla w_1 \\ \nabla w_2 \\ . \\ . \\ . \\ \nabla w_d \\ \nabla x_1 \\ \nabla x_2 \\ . \\ . \\ . \\ \nabla x_V \end{bmatrix} \in \mathbb{R}^{c\times d + V\times d}$$

What are the roles of various components in a deep network?

- What is the role of weights?

What are the roles of various components in a deep network?

- What is the role of weights?
  - Decide which input is important.

What are the roles of various components in a deep network?

- What is the role of weights?
  - Decide which input is important.
- What’s the role of biases?

What are the roles of various components in a deep network?

- What is the role of weights?
  - Decide which input is important.
- What’s the role of biases?
  - A tap for outflow.

NUST
Defining futures
School of Electrical Engineering
& Computer Science

What are the roles of various components in a deep network?

- What is the role of weights?
    - Decide which input is important.
- What's the role of biases?
    - A tap for outflow.
- What each node in the network learns?

What are the roles of various components in a deep network?

- What is the role of weights?
  - Decide which input is important.
- What's the role of biases?
  - A tap for outflow.
- What each node in the network learns?
  - Features. Which feature exactly? We don't know.
  - How to judge the usefulness of feature?

What are the roles of various components in a deep network?

- What is the role of weights?
  - Decide which input is important.
- What's the role of biases?
  - A tap for outflow.
- What each node in the network learns?
  - Features. Which feature exactly? We don't know.
  - How to judge the usefulness of feature?
- What's the need for activation function?

What are the roles of various components in a deep network?

- What is the role of weights?
  - Decide which input is important.
- What's the role of biases?
  - A tap for outflow.
- What each node in the network learns?
  - Features. Which feature exactly? We don't know.
  - How to judge the usefulness of feature?
- What's the need for activation function?
  - Introduce non-linearity and scale the activation. Why non-linearity?
  - Which activation function is better?

Do Deep Neural Networks really perform better?

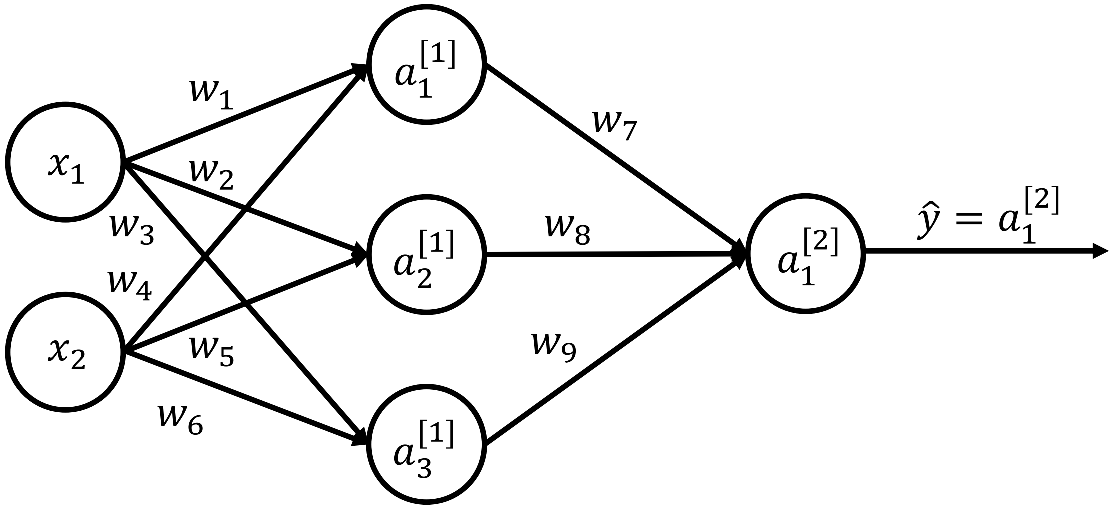
- If you can satiate their appetite.
    - Data for starter
    - Hardware for the main course
    - On the bed of optimised libraries

Do Deep Neural Networks really perform better?

- If you can satiate their appetite.
  - Data for starter
  - Hardware for the main course
  - On the bed of optimised libraries
- Deep Neural Networks are much more to just larger number of layers.
  - What is wide neural network?
  - What happens to gradients in DNNs?
  - Impact of input scale in the DNNs?

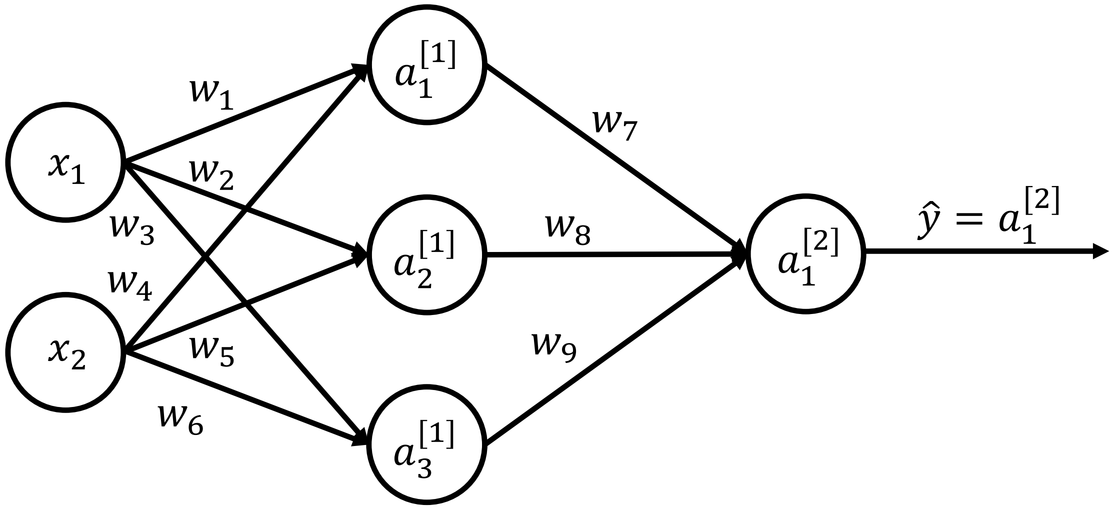
A Multi Layer Perceptron (MLP) is a deep neural network

- Consider the following neural network.
    - It has one input layer of size 2, one hidden layer of size 3, and an output layer of size 1.



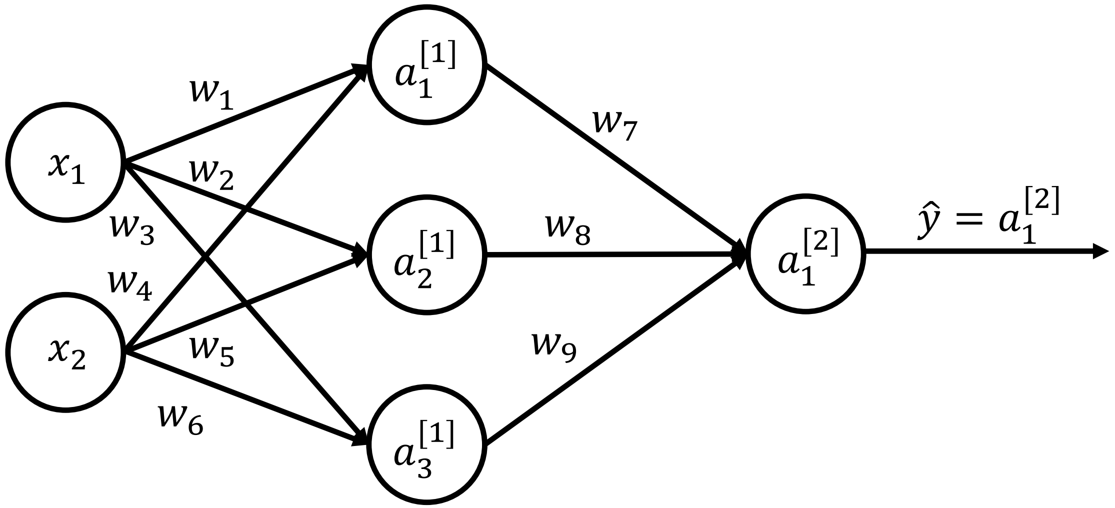
A Multi Layer Perceptron (MLP) is a deep neural network

- Consider the following neural network.
  - It has one input layer of size 2, one hidden layer of size 3, and an output layer of size 1.
- For forward pass:
  - $\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}$
  - $\boldsymbol{a}^{[1]} = \sigma\left(\boldsymbol{z}^{[1]}\right)$
  - $\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$
  - $\boldsymbol{a}^{[2]} = \sigma\left(\boldsymbol{z}^{[2]}\right)$



A Multi Layer Perceptron (MLP) is a deep neural network

- Consider the following neural network.
  - It has one input layer of size 2, one hidden layer of size 3, and an output layer of size 1.
- For forward pass:
  - $\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}$
  - $\boldsymbol{a}^{[1]} = \sigma\,(\boldsymbol{z}^{[1]})$
  - $\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$
  - $\boldsymbol{a}^{[2]} = \sigma\,(\boldsymbol{z}^{[2]})$



- Backward Pass: for update in w_1, for instance.

$$\frac{\partial L}{\partial\, w_1} = \frac{\partial L}{\partial\, a_1^{[2]}} \times \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}} \times \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \times \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \times \frac{\partial z_1^{[1]}}{\partial w_1}$$

NUST Defining futures
School of Electrical Engineering & Computer Science

Activation functions add non-linearity in the neural networks

- Linear neural networks are only suitable for linearly separable data.
    - You might still need non-linearity at the output layer. Why?

Activation functions add non-linearity in the neural networks

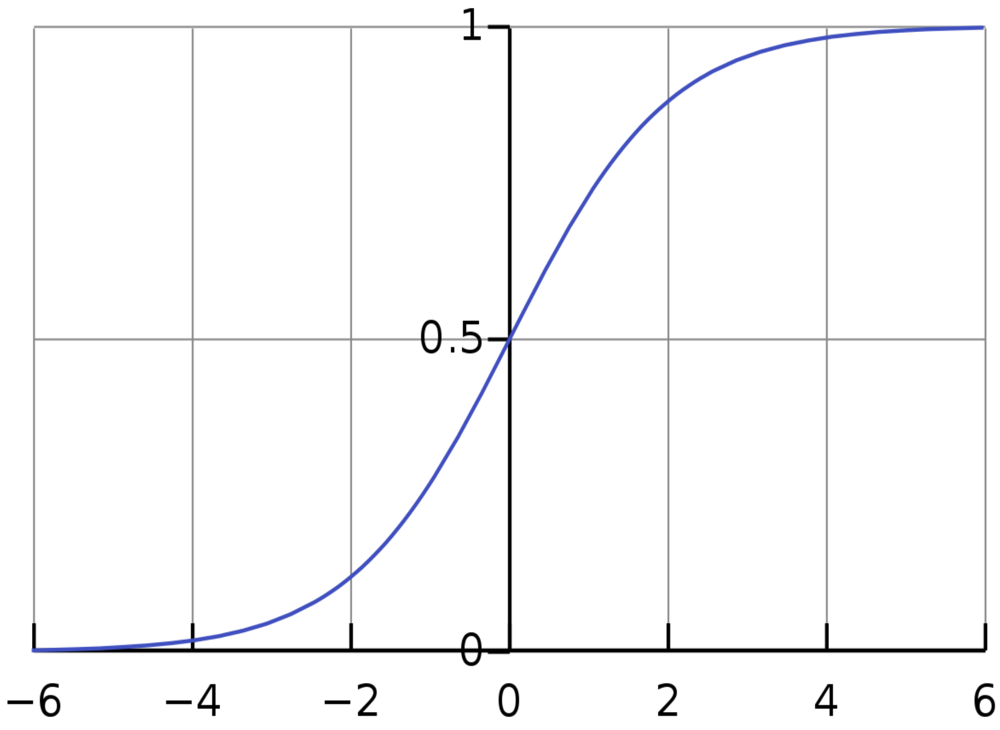
- Linear neural networks are only suitable for linearly separable data.
    - You might still need non-linearity at the output layer. Why?
- The prime objective of activation functions is to introduce non-linearity in neural networks.
    - Learning non-linear functions helps neural networks fit complex data.

Activation functions add non-linearity in the neural networks

- Linear neural networks are only suitable for linearly separable data.
    - You might still need non-linearity at the output layer. Why?
- The prime objective of activation functions is to introduce non-linearity in neural networks.
    - Learning non-linear functions helps neural networks fit complex data.
- Various applications call for different activation functions better suited to the tasks.

Sigmoid function is one of the oldest activation functions

- Suitable for logistic regression.
- Normalises inputs between $(0,1)$ giving psudoprobability.
- Maximum gradient is 0.25 at $x = 0$.



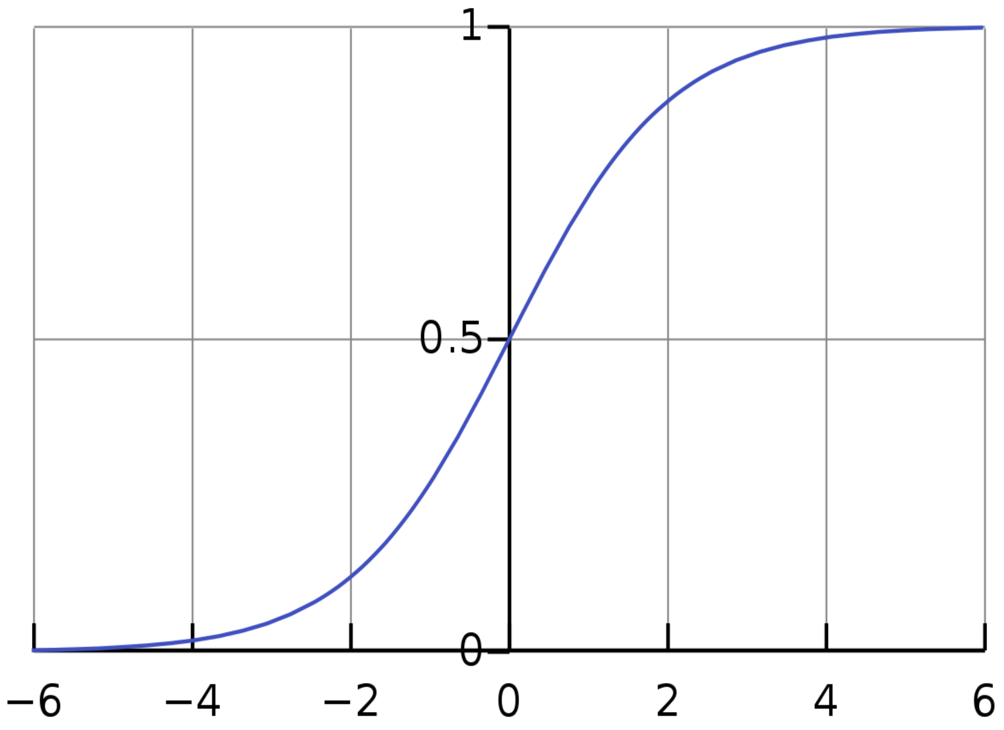
$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$

Sigmoid function is one of the oldest activation functions

- Suitable for logistic regression.
- Normalises inputs between $(0,1)$ giving psudoprobability.
- Maximum gradient is 0.25 at $x = 0$.

  + The function is differentiable over the whole range.
  + Predictions are mostly very clear (close to zero or one).
  + Derivatives are inexpensive to calculate (sigmoid function is already evaluated).



$$f(x) = \frac{1}{1 + e^{-x}}$$

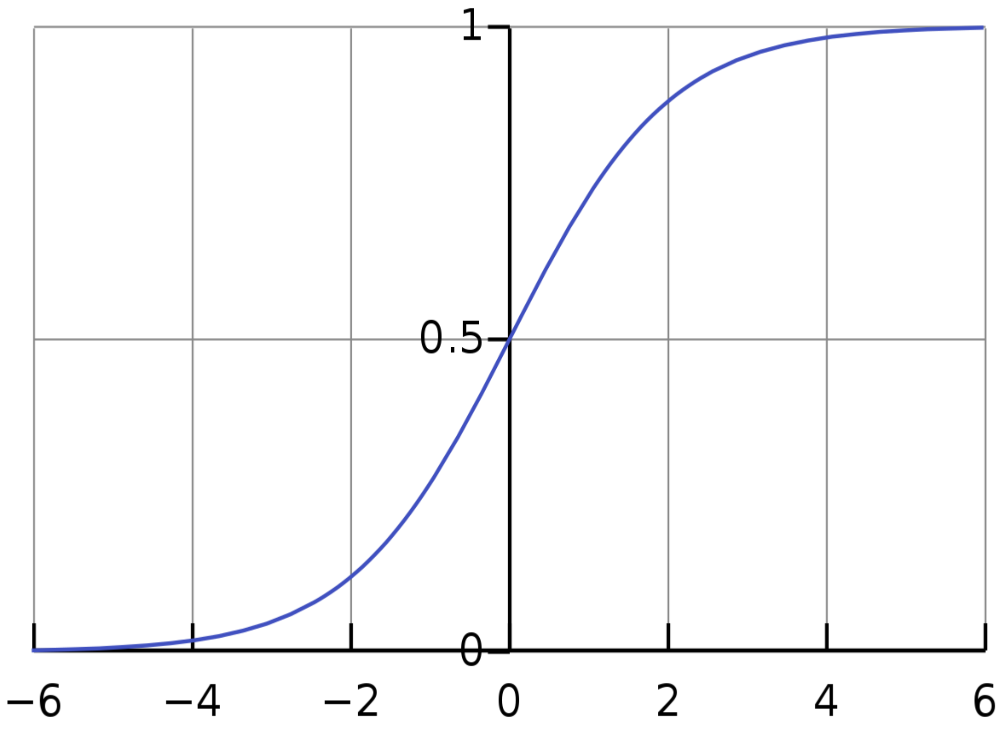
$$f'(x) = f(x)(1 - f(x))$$

Sigmoid function is one of the oldest activation functions

- Suitable for logistic regression.
- Normalises inputs between $(0,1)$ giving psudoprobability.
- Maximum gradient is 0.25 at $x = 0$.

+ The function is differentiable over the whole range.
+ Predictions are mostly very clear (close to zero or one).
+ Derivatives are inexpensive to calculate (sigmoid function is already evaluated).

- Very narrow linear range.
- Severely prone to vanishing gradient resulting in slow learning.
- Its output is not zero-centred. May cause inefficient weight updates
- Exponential operations are computationally expensive.



$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$

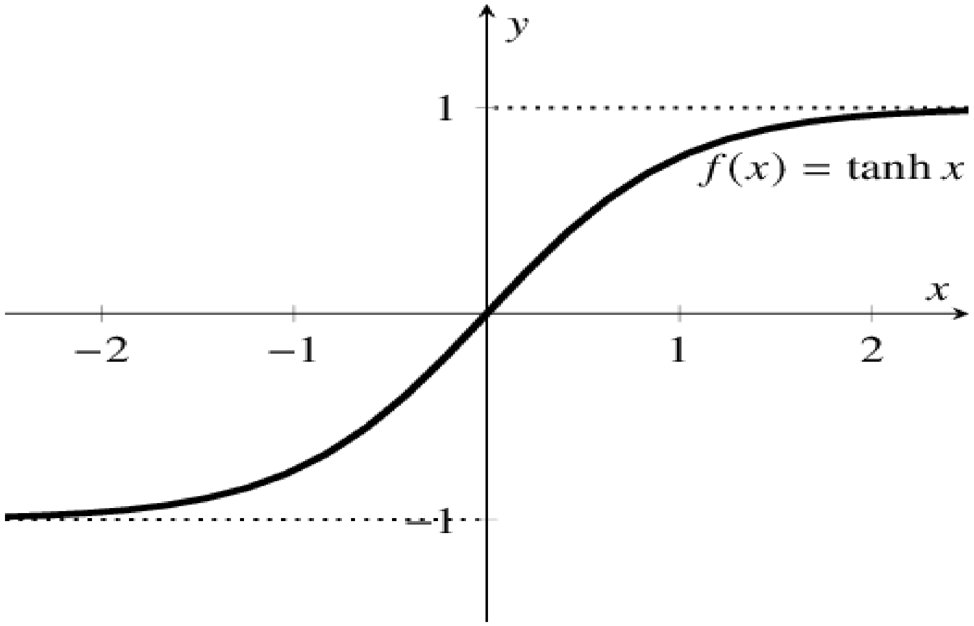
NUST
Defining futures
School of Electrical Engineering
& Computer Science

Hyperbolic Tangent is a better alternative to sigmoid in hidden layers

- A shifted and scaled version of sigmoid.

$$\tanh(z) = 2\,sigmoid(2z) - 1$$

- Most common activation function in RNNs.



$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$f'(x) = 1 - (f(x))^2$$

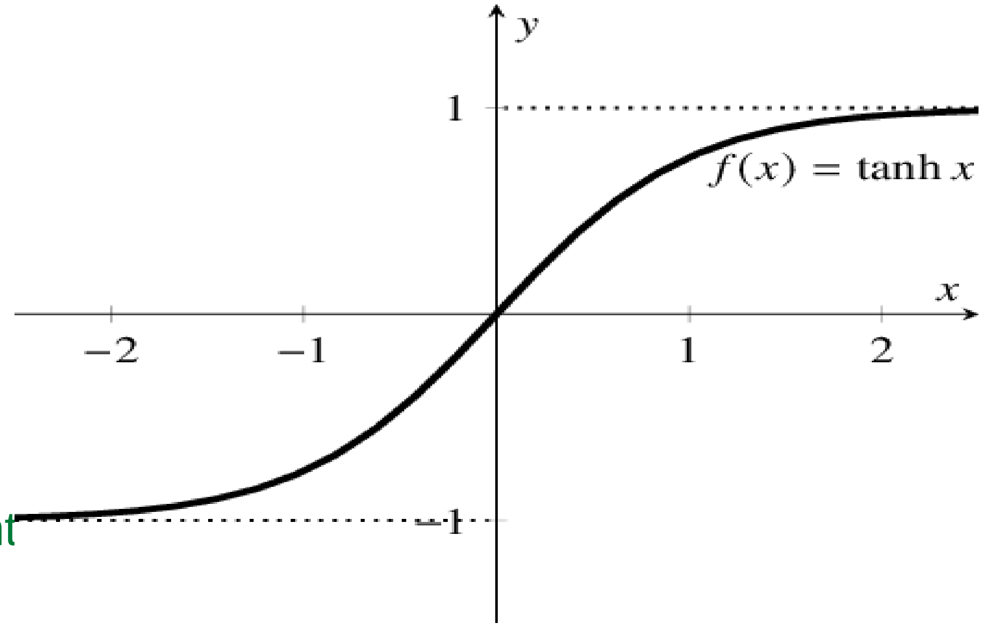
Hyperbolic Tangent is a better alternative to sigmoid in hidden layers

- A shifted and scaled version of sigmoid.

$$\tanh(z) = 2\, sigmoid(2z) - 1$$

- Most common activation function in RNNs.

  + Almost always works better than sigmoid in hidden layers.
  + Due to zero-centred activations, helps in fasting training.
  + More robust to vanishing gradient than sigmoid.
  + Slope in linear regions is higher than sigmoid, better weight updates.



$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$f'(x) = 1 - (f(x))^2$$

Hyperbolic Tangent is a better alternative to sigmoid in hidden layers

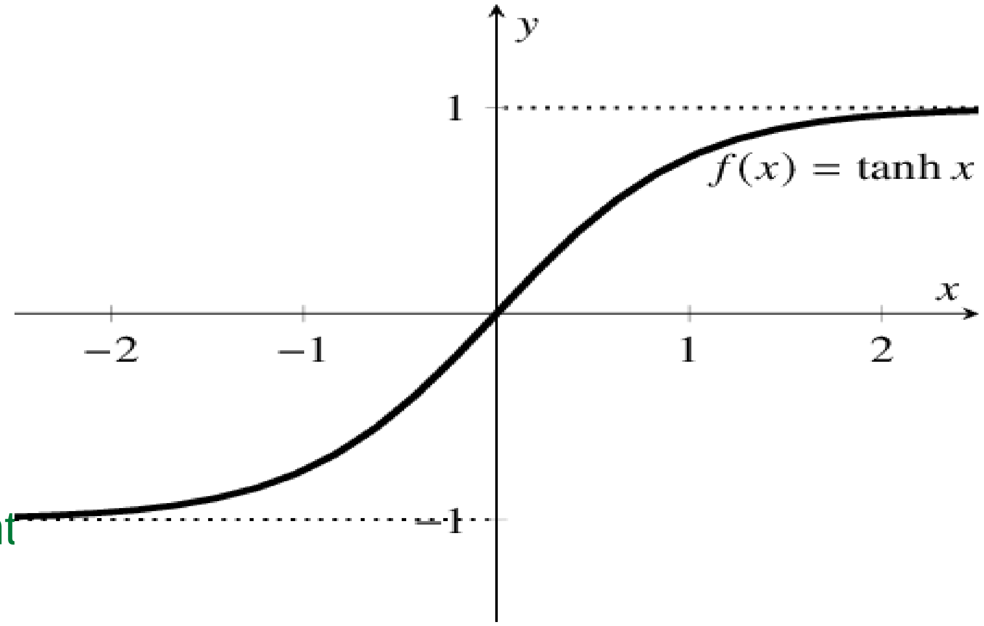
- A shifted and scaled version of sigmoid.

$$\tanh(z) = 2\, sigmoid(2z) - 1$$

- Most common activation function in RNNs.

  + Almost always works better than sigmoid in hidden layers.
  + Due to zero-centred activations, helps in fasting training.
  + More robust to vanishing gradient than sigmoid.
  + Slope in linear regions is higher than sigmoid, better weight updates.

  - Still saturates and may lead to vanishing gradient.
  - More exponential operations, more computationally expensive.



$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f'(x) = 1 - (f(x))^2$$

Hyperbolic Tangent is a better alternative to sigmoid in hidden layers

- A shifted and scaled version of sigmoid.

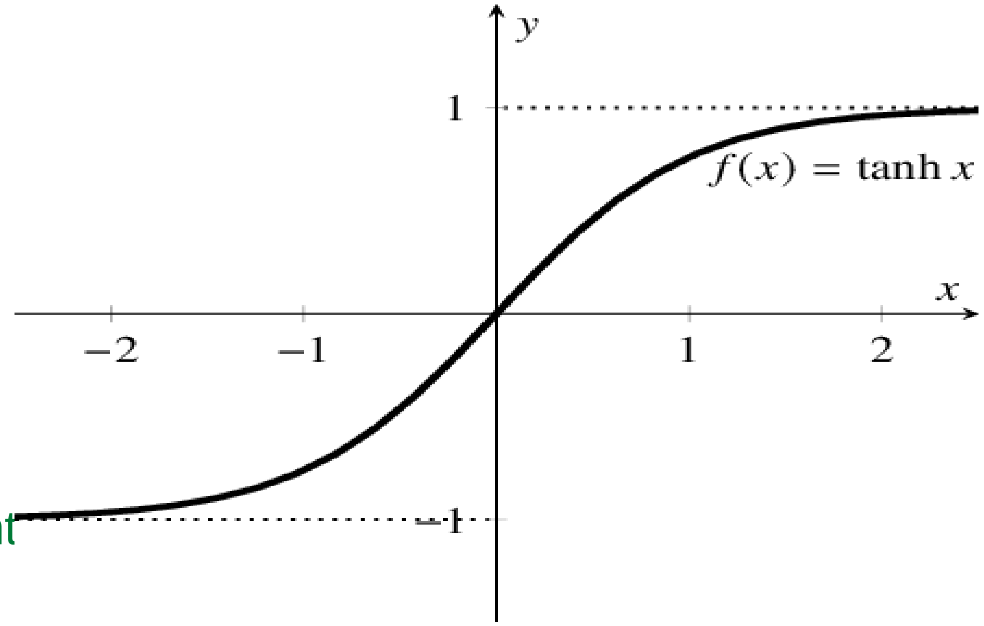
$$\tanh(z) = 2\, sigmoid(2z) - 1$$

- Most common activation function in RNNs.

  + Almost always works better than sigmoid in hidden layers.
  + Due to zero-centred activations, helps in fasting training.
  + More robust to vanishing gradient than sigmoid.
  + Slope in linear regions is higher than sigmoid, better weight updates.

  - Still saturates and may lead to vanishing gradient.
  - More exponential operations, more computationally expensive.

- Variant: Hard tanh.

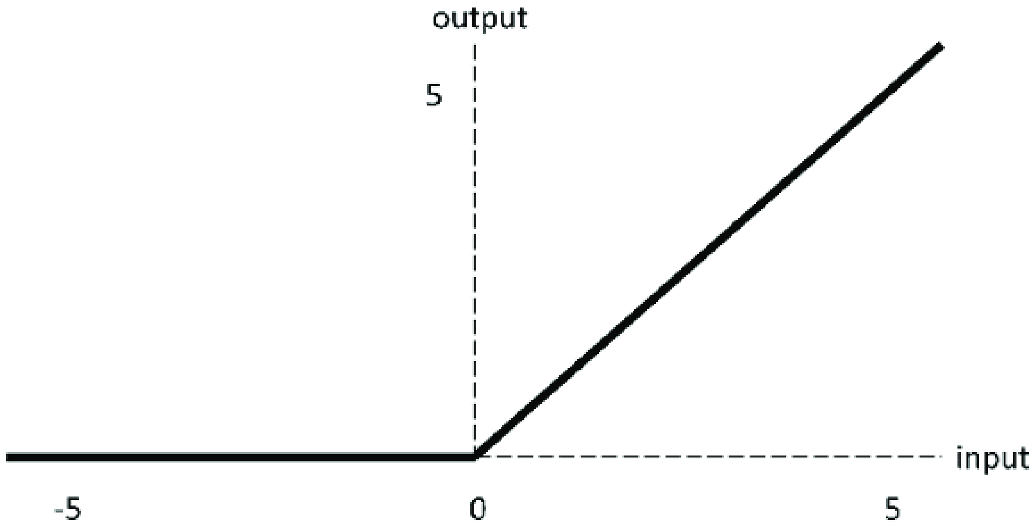


$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f'(x) = 1 - (f(x))^2$$

Rectified Linear Unit is the most popular activation function in hidden layers

- Better substitute for $\tanh$ and $sigmoid$ functions for hidden layers.
- May also be used in output layer.
- Probability of having x exactly equal to 0 is very small.



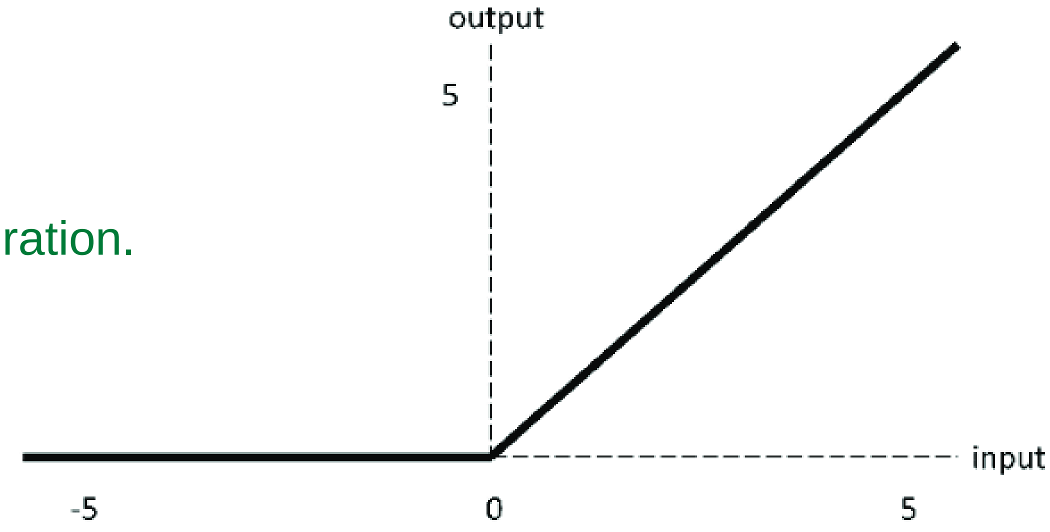
$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \\ undefined, & x = 0 \end{cases}$$

Rectified Linear Unit is the most popular activation function in hidden layers

- Better substitute for $\tanh$ and $sigmoid$ functions for hidden layers.
- May also be used in output layer.
- Probability of having x exactly equal to 0 is very small.

  + Cheap and efficient in both forward and backward passes.
  + Fares well in most training scenarios without gradient saturation.
  + Training is fast because of mostly non-negative gradient.



$$f(x) = \max(0, x)$$

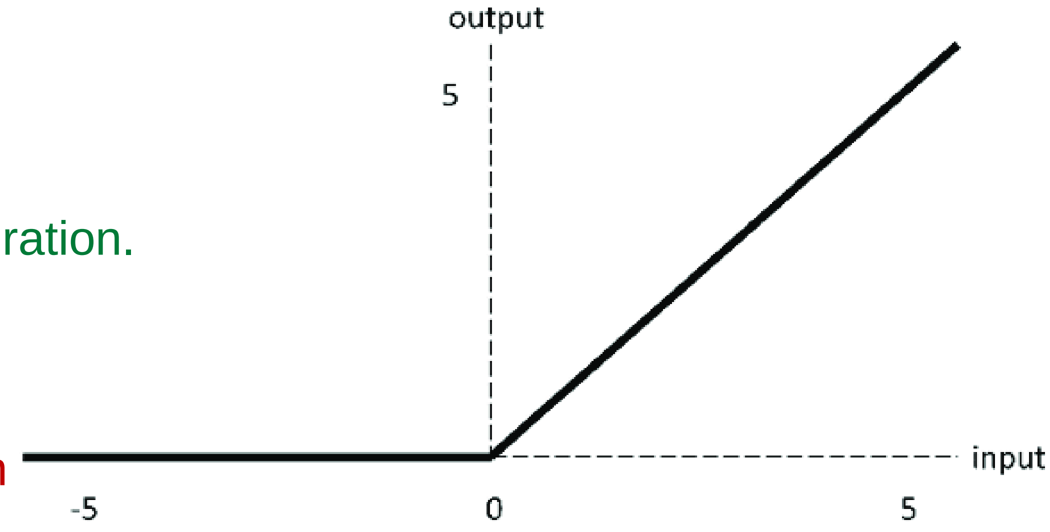
$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \\ undefined, & x = 0 \end{cases}$$

Rectified Linear Unit is the most popular activation function in hidden layers

- Better substitute for $\tanh$ and $sigmoid$ functions for hidden layers.
- May also be used in output layer.
- Probability of having x exactly equal to 0 is very small.

+ Cheap and efficient in both forward and backward passes.
+ Fares well in most training scenarios without gradient saturation.
+ Training is fast because of mostly non-negative gradient.

- Has a discontinuity at $x = 0$. Work around?
- Dead ReLU. (Practically works fine since there are enough positive parameters)
- Mean of activations is not zero-centred.



$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \\ undefined, & x = 0 \end{cases}$$

NUST
Defining futures
School of Electrical Engineering
& Computer Science

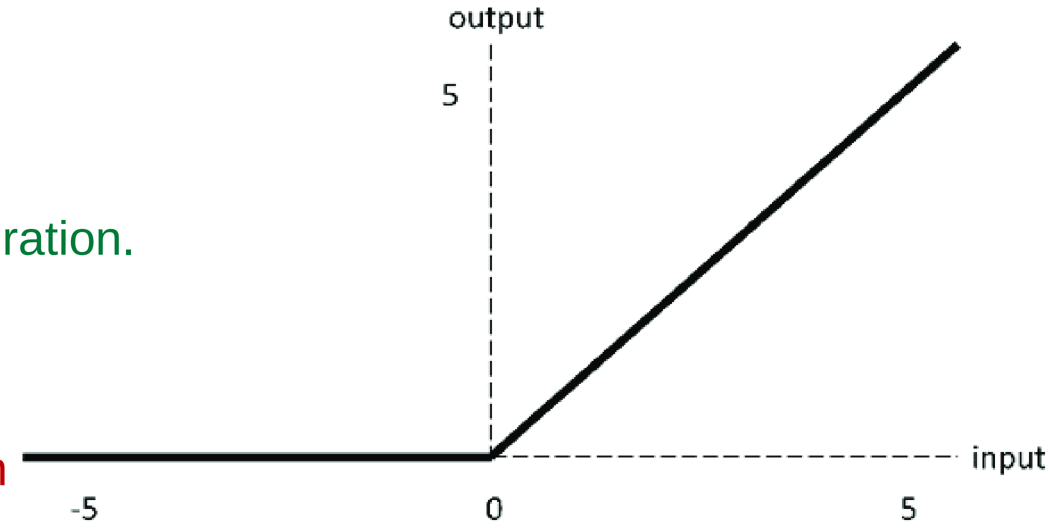
Rectified Linear Unit is the most popular activation function in hidden layers

- Better substitute for $\tanh$ and $sigmoid$ functions for hidden layers.
- May also be used in output layer.
- Probability of having x exactly equal to 0 is very small.

  + Cheap and efficient in both forward and backward passes.
  + Fares well in most training scenarios without gradient saturation.
  + Training is fast because of mostly non-negative gradient.

  - Has a discontinuity at $x = 0$. Work around?
  - Dead ReLU. (Practically works fine since there are enough positive parameters)
  - Mean of activations is not zero-centred.

- Variants: Leaky ReLU, Parametric ReLU, Swish



$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \\ undefined, & x = 0 \end{cases}$$

How to initialise network parameters?

- Initialise all weights and biases with zero?
  - For biases, initialising with zero is fine but not for weights.
  - If weights are initialised with zero, all activations of a layer will be the same.

https://www.deeplearning.ai/ai-notes/initialization/index.html

How to initialise network parameters?

- Initialise all weights and biases with zero?
    - For biases, initialising with zero is fine but not for weights.
    - If weights are initialised with zero, all activations of a layer will be the same.
- Initialise all weights and biases with a constant?
    - Same thing will happen as initialising with all zeros.

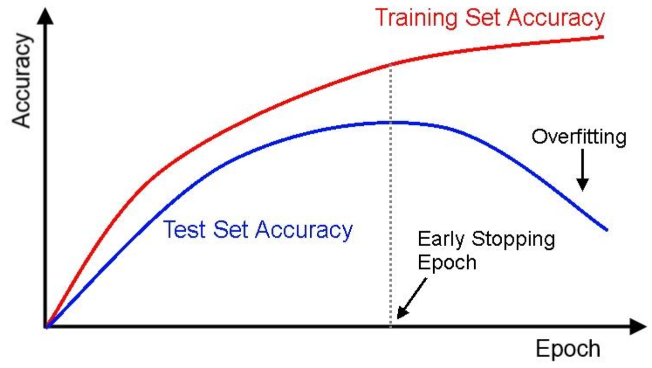
https://www.deeplearning.ai/ai-notes/initialization/index.html

How to initialise network parameters?

- Initialise all weights and biases with zero?
  - For biases, initialising with zero is fine but not for weights.
  - If weights are initialised with zero, all activations of a layer will be the same.
- Initialise all weights and biases with a constant?
  - Same thing will happen as initialising with all zeros.
- Initialise all weights and biases randomly. Following what distribution?
  - Uniform distribution $[0, 1]$
  - Uniform distribution $[-0.5, 0.5]$
  - Gaussian distribution with mean 0 and small variance. Why?

https://www.deeplearning.ai/ai-notes/initialization/index.html

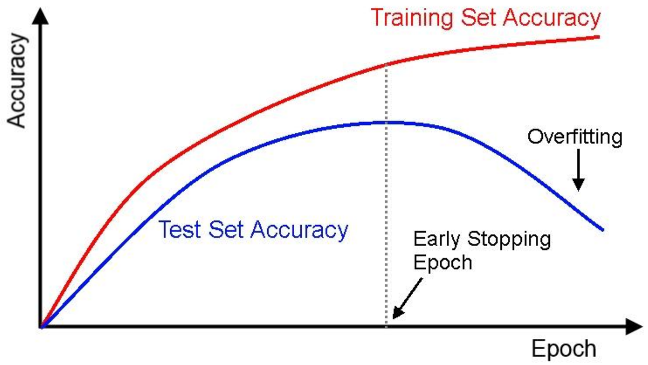
Regularisation prevents overfitting



Regularisation prevents overfitting

- L2 Regularisation is a common and useful method to prevent overfitting.

$$J(w,b) = \frac{1}{m}\sum_{i=0}^{m}\mathcal{L}(y^i,\hat{y}^i) + \frac{\lambda}{2m}\sum_{j=0}^{L}\left\|w^{[l]}\right\|^2$$



NUST
Defining futures
School of Electrical Engineering & Computer Science

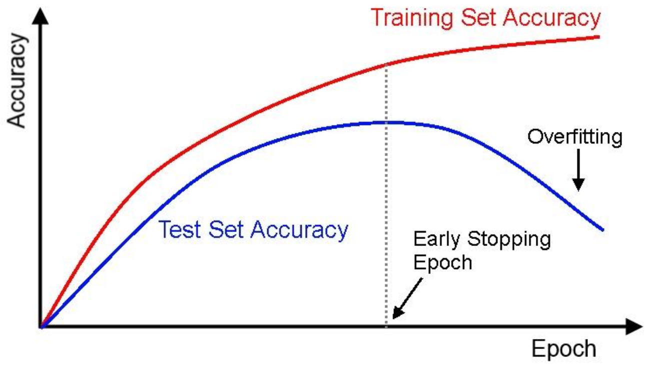
Regularisation prevents overfitting

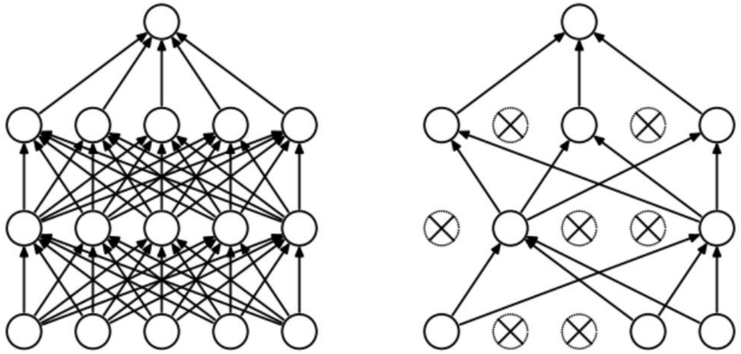
- L2 Regularisation is a common and useful method to prevent overfitting.

$$J(w,b) = \frac{1}{m}\sum_{i=0}^{m}\mathcal{L}(y^i,\hat{y}^i) + \frac{\lambda}{2m}\sum_{j=0}^{L}\left\|w^{[l]}\right\|^2$$

- Dropouts are used only at training time.

$$r^{[l-1]} \sim Bernoulli\ (p)$$
$$\tilde{a}^{[l-1]} = r^{[l-1]} * a^{[l-1]}$$
$$z^{[l]} = w^{[l]}\tilde{a}^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g(z^{[l]})$$





(a) Standard Neural Net

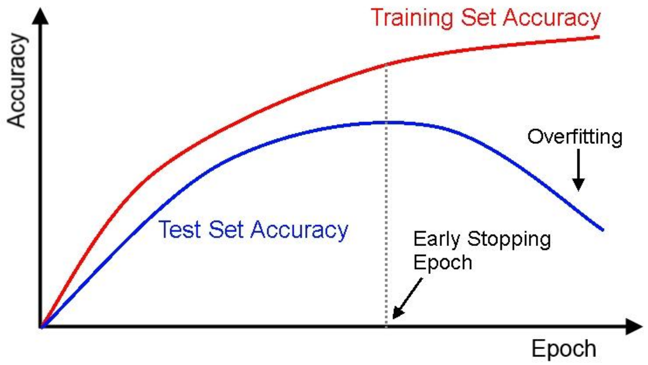
(b) After applying dropout.

NUST
Defining futures
School of Electrical Engineering
& Computer Science

Regularisation prevents overfitting

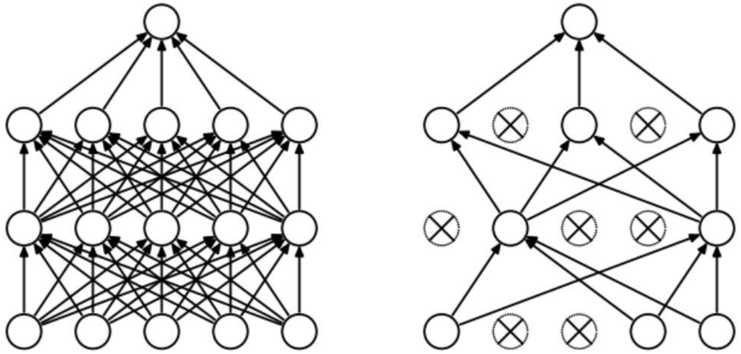
- L2 Regularisation is a common and useful method to prevent overfitting.

$$J(w,b) = \frac{1}{m}\sum_{i=0}^{m}\mathcal{L}(y^i,\hat{y}^i) + \frac{\lambda}{2m}\sum_{j=0}^{L}\left\|w^{[l]}\right\|^2$$



- Dropouts are used only at training time.

$$\begin{gathered} r^{[l-1]} \sim Bernoulli\ (p) \\ \tilde{a}^{[l-1]} = r^{[l-1]} * a^{[l-1]} \\ z^{[l]} = w^{[l]}\tilde{a}^{[l-1]} + b^{[l]} \\ a^{[l]} = g(z^{[l]}) \end{gathered}$$



(a) Standard Neural Net (b) After applying dropout.

- Early Stopping is not a regularisation technique per se but helps *evade* overfitting.

Learning rate decay helps adjust speed of convergence

- Manual Decay: Can only work with smaller models.

- Discrete Staircase Decay: Divide learning rate by a constant every fixed number of epochs.

Learning rate decay helps adjust speed of convergence

- Manual Decay: Can only work with smaller models.

- Discrete Staircase Decay: Divide learning rate by a constant every fixed number of epochs.

- Gradual Decay

$$\alpha = \frac{\alpha_o}{1 + decayFactor \ * epoch}$$

- Exponential Decay

$$\alpha = \ \mathrm{decayFactor}^{epoch} \ * \alpha_o$$

Learning rate decay helps adjust speed of convergence

- Manual Decay: Can only work with smaller models.

- Discrete Staircase Decay: Divide learning rate by a constant every fixed number of epochs.

- Gradual Decay

$$\alpha = \frac{\alpha_o}{1 + decayFactor\ * epoch}$$

- Exponential Decay

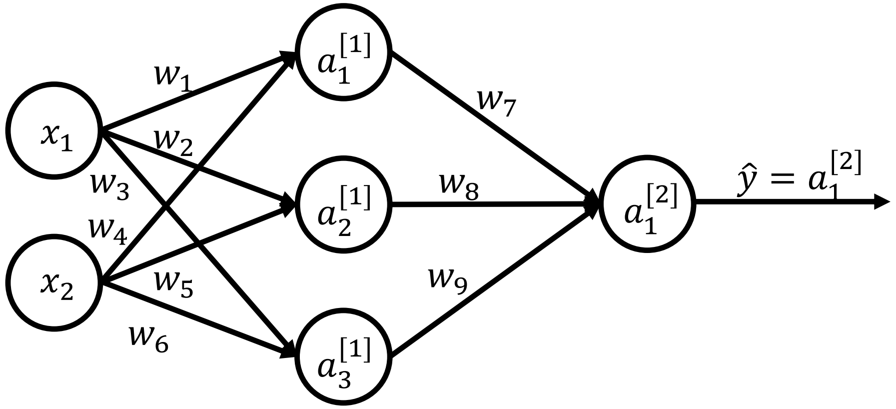
$$\alpha = \mathrm{decayFactor}^{epoch}\ * \alpha_o$$

- Conditional Decay

  - Reduce α only if and when necessary.

Batch Normalisation helps with training stability and convergence

- Normalises the hidden activations.



Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. PMLR, 2015.

Batch Normalisation helps with training stability and convergence

- Normalises the hidden activations.

- Batch Norm takes two steps.

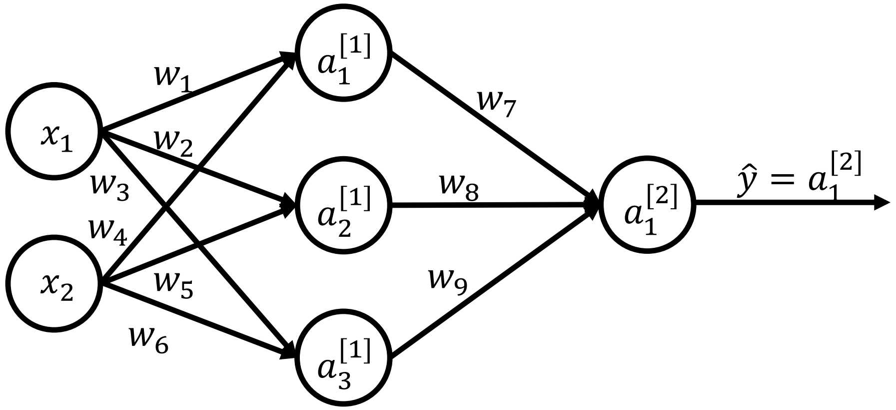
  - Standardise the activations.

$$z'^{(i)}_j = \frac{z^{(i)}_j - \mu}{\sigma}, i \in \{1, \dots, n\}, j \in \{1, \dots, d\}$$

    Here n is minibatch size, and d is the size of the layer.

  - Pre-activation scaling of net inputs.

$$\tilde{z}^{(i)}_j = \gamma_j . z'^{(i)}_j + \beta_j$$



Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. PMLR, 2015.

Batch Normalisation helps with training stability and convergence

- Normalises the hidden activations.

- Batch Norm takes two steps.

  - Standardise the activations.

$$z'^{(i)}_j = \frac{z^{(i)}_j - \mu}{\sigma}, i \in \{1, \dots, n\}, j \in \{1, \dots, d\}$$

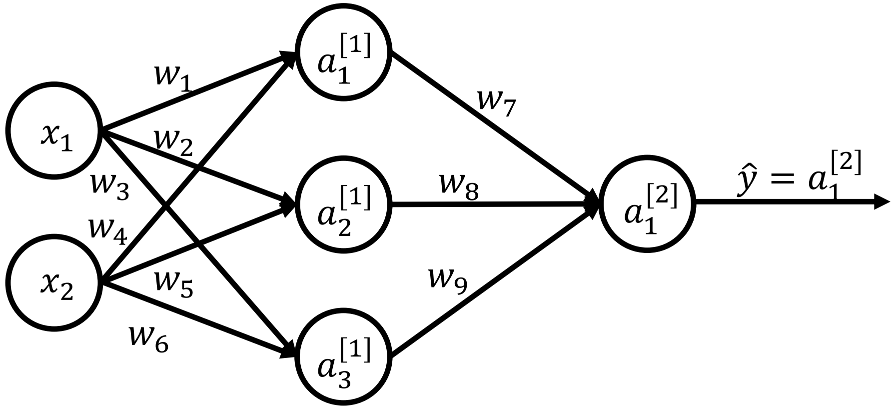
    Here n is minibatch size, and d is the size of the layer.

  - Pre-activation scaling of net inputs.

$$\tilde{z}^{(i)}_j = \gamma_j . z'^{(i)}_j + \beta_j$$

- Batch Norm parameters have the same size as the bias vector.

- Batch Normalisation of layer l helps in faster training of layer $l + 1$.



Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. PMLR, 2015.

Do you have any question?

Some material (images, tables, text etc.) in this presentation has been borrowed from different books, lecture notes, and the web. The original contents solely belong to their owners, and are used in this presentation only for clarifying various educational concepts. Any copyright infringement is ***not*** intended.