Lecture 7. Multilayer Perceptron. Backpropagation

COMP90051 Statistical Machine Learning

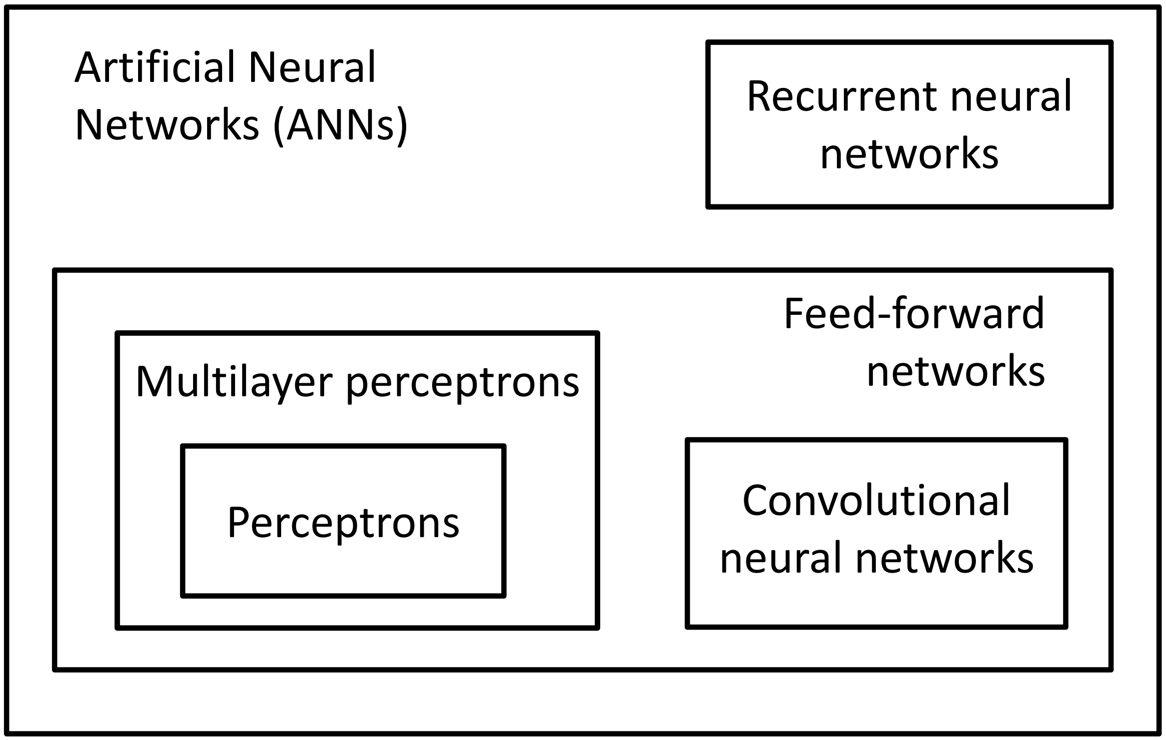
Semester 2, 2018
Lecturer: Ben Rubinstein

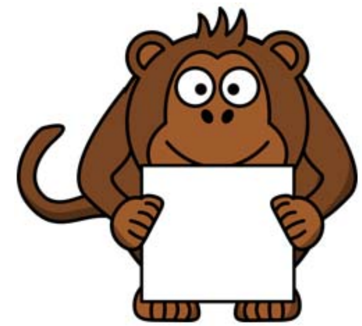
THE UNIVERSITY OF MELBOURNE

This lecture

- Multilayer perceptron
  - Model structure
  - Universal approximation
  - Training preliminaries
- Backpropagation
  - Step-by-step derivation
  - Notes on regularisation

Animals in the zoo





art: OpenClipartVectors at pixabay.com (CC0)

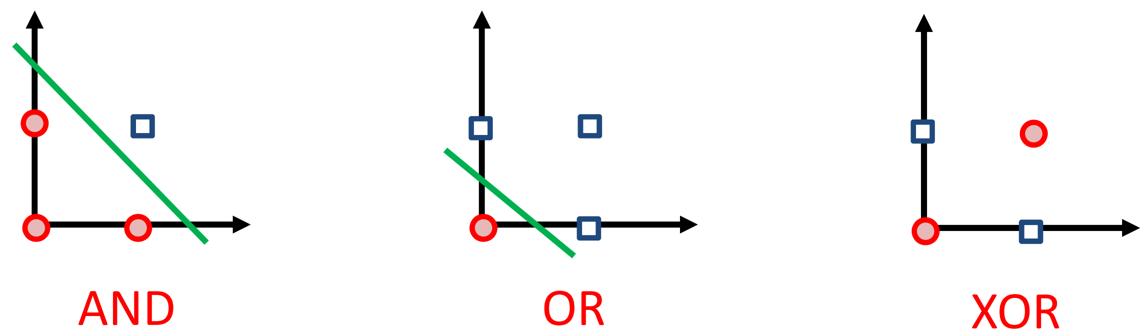
- Recurrent neural networks are not covered in this subject
- An autoencoder is an ANN trained in a specific way.
  - E.g., a multilayer perceptron can be trained as an autoencoder, or a recurrent neural network can be trained as an autoencoder.

Multilayer Perceptron

Modelling non-linearity via function composition

Limitations of linear models

Some problems are linearly separable, but many are not



Possible solution: composition

$x_1 \text{ XOR } x_2 = (x_1 \text{ OR } x_2) \text{ AND not}(x_1 \text{ AND } x_2)$

We are going to compose perceptrons …

Perceptorn is *sort of* a building block for ANN

- ANNs are not restricted to binary classification
- Nodes in ANN can have various activation functions

Step function $$f(s) = \begin{cases} 1, & if\ s \geq 0 \\ 0, & if\ s < 0 \end{cases}$$

Sign function $$f(s) = \begin{cases} 1, & if\ s \geq 0 \\ -1, & if\ s < 0 \end{cases}$$

Logistic function $$f(s) = \frac{1}{1 + e^{-s}}$$

Many others: $tanh$, rectifier, etc.

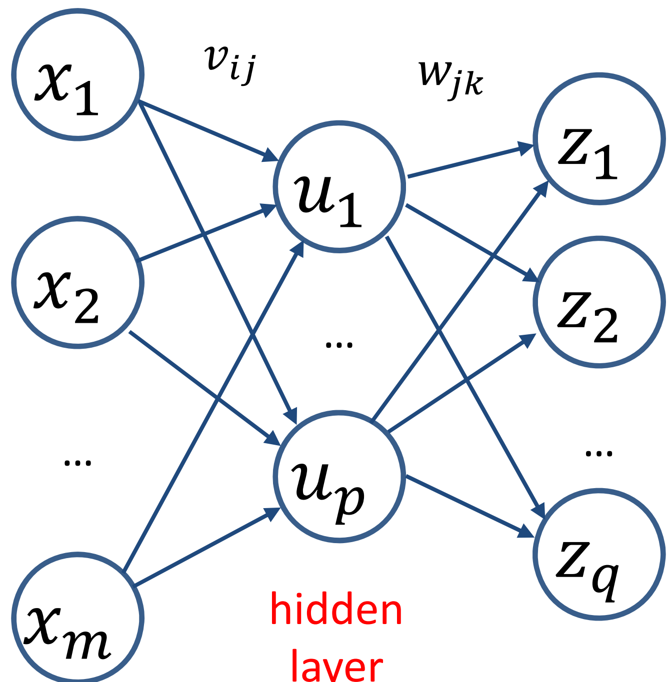
$\boxed{\text{Feed-forward}}$ Artificial Neural Network

flow of computation →

x_i are inputs, i.e., attributes

note: here x_i are components of a single training instance $\boldsymbol{x}$

a training dataset is a set of instances



input layer

hidden layer

output layer

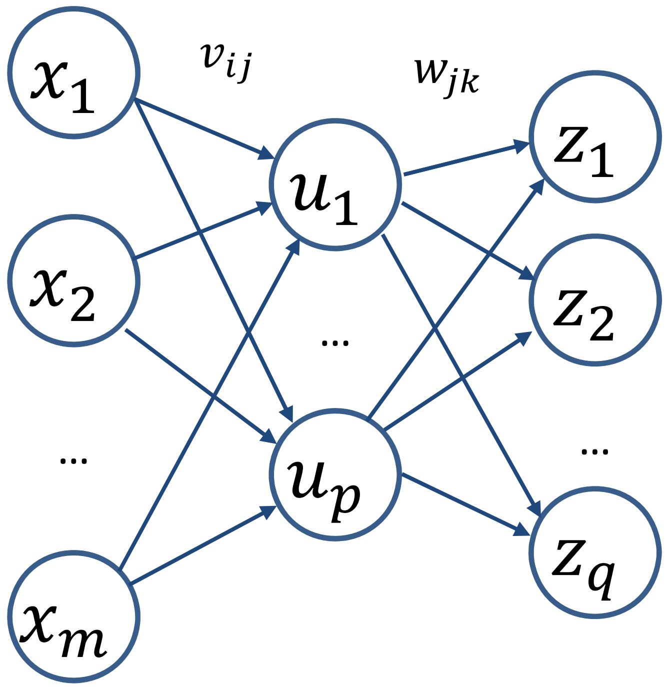
(ANNs can have more than one hidden layer)

z_i are outputs, i.e., predicted labels

note: ANNs naturally handle multidimensional output

e.g., for handwritten digits recognition, each output node can represent the probability of a digit

ANN as function composition



$$u_j = g(r_j)$$

$$r_j = v_{0j} + \sum_{i=1}^{m} x_i v_{ij}$$

$$z_k = h(s_k)$$

$$s_k = w_{0k} + \sum_{j=1}^{p} u_j w_{jk}$$

note that z_k is a function composition (a function applied to the result of another function, etc.)

here g, h are activation functions. These can be either same (e.g., both sigmoid) or different

you can add bias node $x_0 = 1$ to simplify equations: $r_j = \sum_{i=0}^{m} x_i v_{ij}$

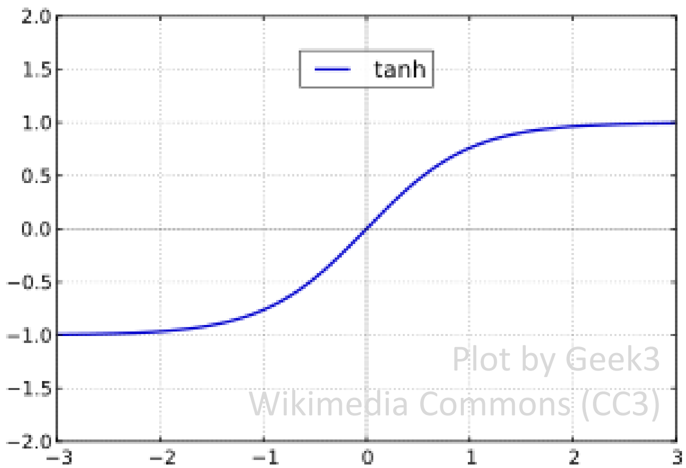
similarly you can add bias node $u_0 = 1$ to simplify equations: $s_k = \sum_{j=0}^{p} u_j w_{jk}$

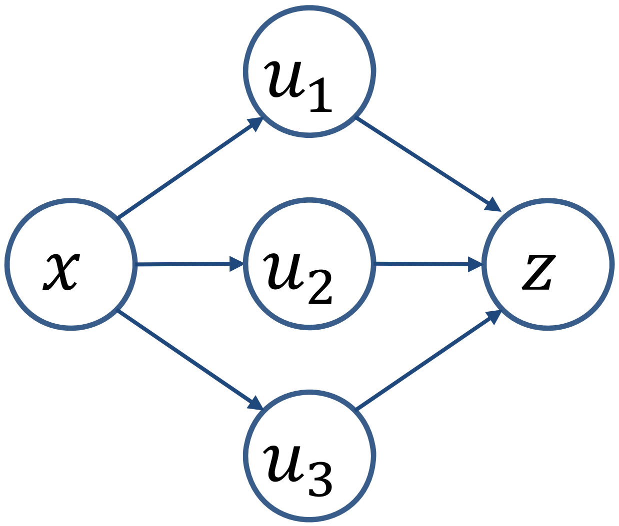
ANN in supervised learning

- ANNs can be naturally adapted to various supervised learning setups, such as univariate and multivariate regression, as well as binary and multilabel classification
- Univariate regression $y = f(\boldsymbol{x})$
  - e.g., linear regression earlier in the course
- Multivariate regression $\boldsymbol{y} = f(\boldsymbol{x})$
  - predicting values for multiple continuous outcomes
- Binary classification
  - e.g., predict whether a patient has type II diabetes
- Multivariate classification
  - e.g., handwritten digits recognition with labels "1", "2", etc.

The power of ANN as a non-linear model

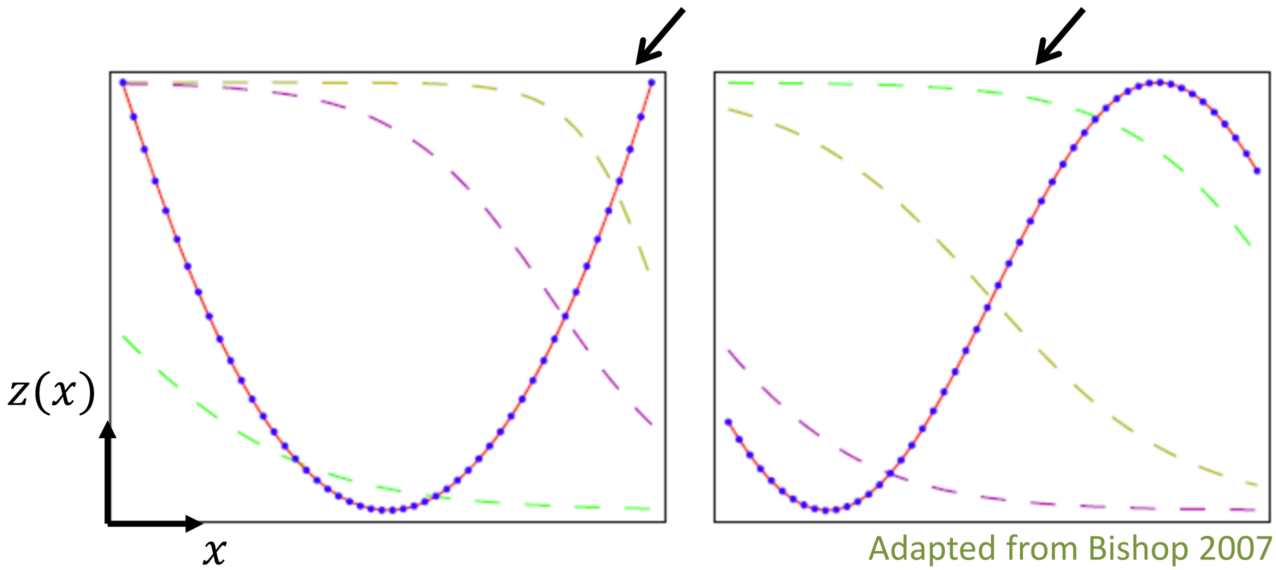
- ANNs are capable of approximating plethora non-linear functions, e.g., $z(x) = x^2$ and $z(x) = \sin x$
- For example, consider the following network. In this example, hidden unit activation functions are tanh





The power of ANN as a non-linear model

- ANNs are capable of approximating various non-linear functions, e.g., $z(x) = x^2$ and $z(x) = \sin x$



Adapted from Bishop 2007

Blue points are the function values evaluated at different x. Red lines are the predictions from the ANN. Dashed lines are outputs of the hidden units

- *Universal approximation theorem* *(Cybenko 1989)*: An ANN with a hidden layer with a finite number of units, and mild assumptions on the activation function, can approximate continuous functions on compact subsets of $\boldsymbol{R}^n$ arbitrarily well

How to train your ~~dragon~~ network?

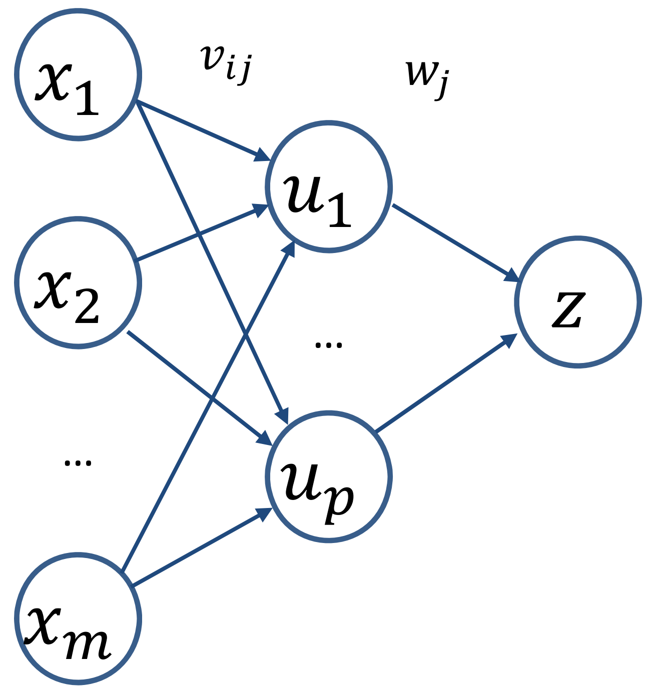
- You know the drill: Define the loss function and find parameters that minimise the loss on training data

Adapted from Movie Poster from
Flickr user jdxyw (CC BY-SA 2.0)

- In the following, we are going to use stochastic gradient descent with a batch size of one. That is, we will process training examples one by one

Training setup: univariate regression

- Consider regression
- Moreover, we'll use identity output activation function $z = h(s) = s = \sum_{j=0}^{p} u_j w_j$
- This will simplify description of backpropagation. In other settings, the training procedure is similar



Training setup: univariate regression

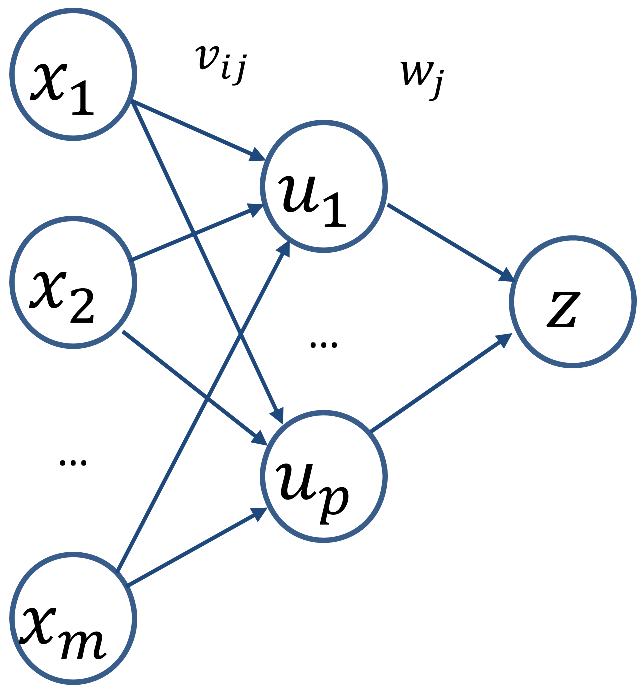
How many parameters does this ANN have? Assume there are bias nodes x0, u0.

$$mp + (p+1)$$

$$(m+2)p + 1$$

$$(m+1)p$$

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**



Loss function for ANN training

- Need loss between training example $\{\boldsymbol{x}, y\}$ & prediction $\hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) = z$, where $\boldsymbol{\theta}$ is parameter vector of v_{ij} and w_j
- As regression, can use squared error

$$L = \frac{1}{2}\left(\hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) - y\right)^2 = \frac{1}{2}(z - y)^2$$

(the constant is used for mathematical convenience, see later)

- Decision-theoretic training: minimise L w.r.t $\boldsymbol{\theta}$
  - Fortunately $L(\boldsymbol{\theta})$ is differentiable
  - Unfortunately no analytic solution in general

Stochastic gradient descent for ANN

Choose initial guess $\boldsymbol{\theta}^{(0)}$, $k = 0$

Here $\boldsymbol{\theta}$ is a set of all weights form all layers

For i from 1 to T (epochs)

For j from 1 to N (training examples)

Consider example $\{\boldsymbol{x}_j, y_j\}$

Update: $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \boldsymbol{\nabla} L(\boldsymbol{\theta}^{(i)})$

$$L = \frac{1}{2}\left(z_j - y_j\right)^2$$

Need to compute partial derivatives $\frac{\partial L}{\partial v_{ij}}$ and $\frac{\partial L}{\partial w_j}$

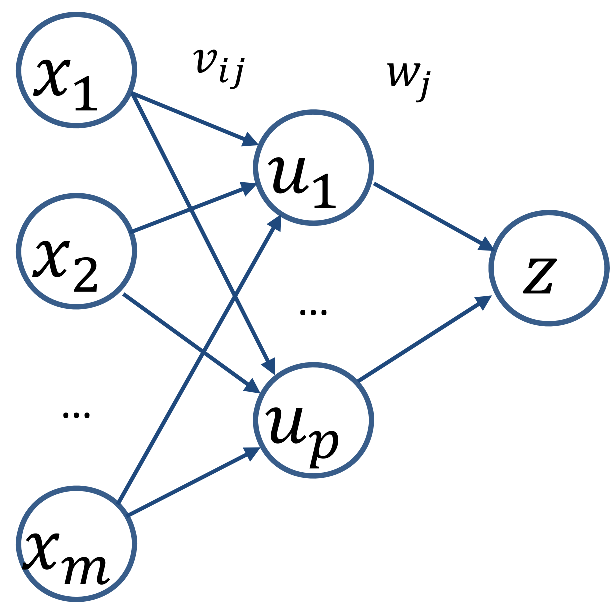
Backpropagation

= “backward propagation of errors”

Calculating the gradient
of loss of a composition

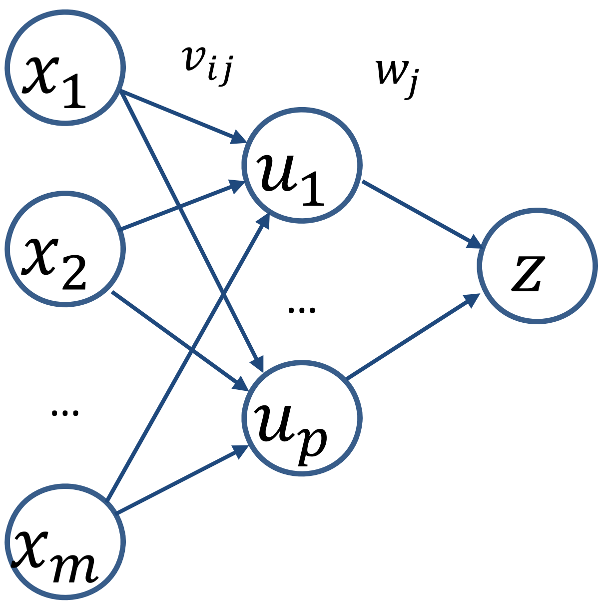
Backpropagation: start with the chain rule

- Recall that the output z of an ANN is a function composition, and hence $L(z)$ is also a composition
  - $L = 0.5(z-y)^2 = 0.5(h(s)-y)^2 = 0.5(s-y)^2$
  - $= 0.5\left(\sum_{j=0}^{p} u_j w_j - y\right)^2 = 0.5\left(\sum_{j=0}^{p} g(r_j) w_j - y\right)^2 = \cdots$
- Backpropagation makes use of this fact by applying the chain rule for derivatives
- $\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial w_j}$
- $\frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}\frac{\partial r_j}{\partial v_{ij}}$



Backpropagation: intermediate step

- Apply the chain rule
- $$\frac{\partial L}{\partial w_j} = \boxed{\frac{\partial L}{\partial z}\frac{\partial z}{\partial s}}\frac{\partial s}{\partial w_j}$$
- $$\frac{\partial L}{\partial v_{ij}} = \boxed{\frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}}\frac{\partial r_j}{\partial v_{ij}}$$



- Now define
$$\delta \equiv \frac{\partial L}{\partial s} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}$$
$$\varepsilon_j \equiv \frac{\partial L}{\partial r_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}$$
- Here $L = 0.5(z-y)^2$ and $z = s$

  Thus $\boxed{\delta = (z-y)}$
- Here $s = \sum_{j=0}^{p} u_j w_j$ and $u_j = h(r_j)$

  Thus $\boxed{\varepsilon_j = \delta w_j h'(r_j)}$

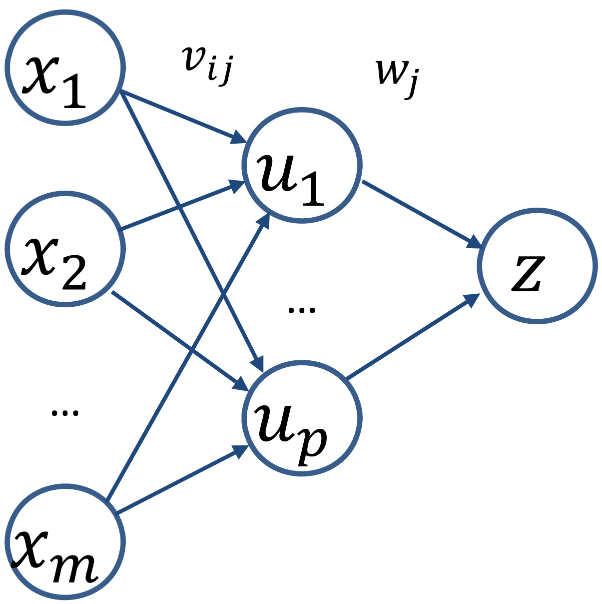
Backpropagation equations

- We have
  - $\frac{\partial L}{\partial w_j} = \delta \boxed{\frac{\partial s}{\partial w_j}}$
  - $\frac{\partial L}{\partial v_{ij}} = \varepsilon_j \boxed{\frac{\partial r_j}{\partial v_{ij}}}$

… where

- $\delta = \frac{\partial L}{\partial s} = (z - y)$
- $\varepsilon_j = \frac{\partial L}{\partial r_j} = \delta w_j h'(r_j)$

- Recall that
  - $s = \sum_{j=0}^{p} u_j w_j$
  - $r_j = \sum_{i=0}^{m} x_i v_{ij}$

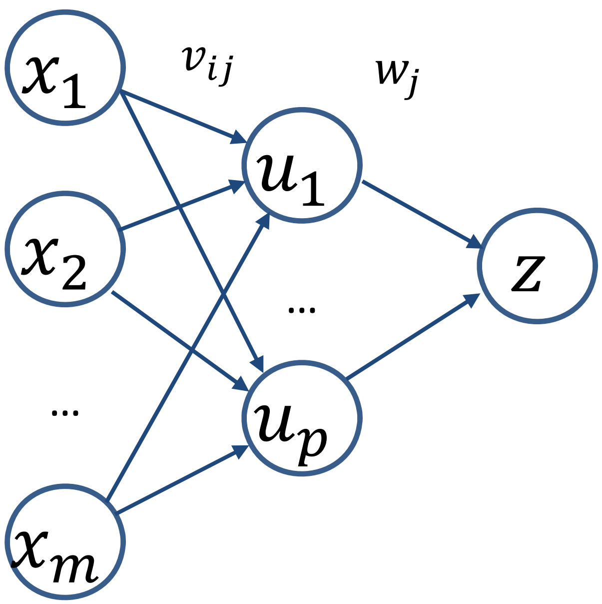


- So $\frac{\partial s}{\partial w_j} = u_j$ and $\frac{\partial r_j}{\partial v_{ij}} = x_i$

- We have
  - $\frac{\partial L}{\partial w_j} = \delta u_j = (z - y)u_j$
  - $\frac{\partial L}{\partial v_{ij}} = \varepsilon_j x_i = \delta w_j h'(r_j) x_i$

Forward propagation

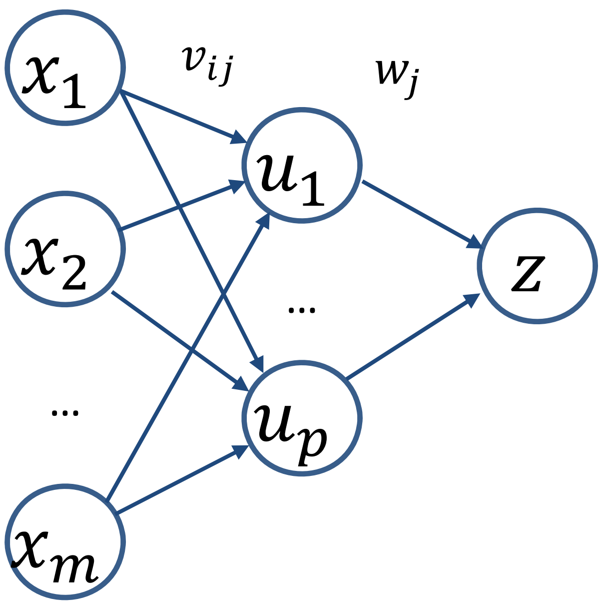
- Use current estimates of v_{ij} and w_j
- Calculate r_j, u_j, s and z



- Backpropagation equations
  - $\frac{\partial L}{\partial w_j} = \delta u_j = (z - y)u_j$
  - $\frac{\partial L}{\partial v_{ij}} = \varepsilon_j x_i = \delta w_j h'(r_j) x_i$

Backward propagation of errors

$$\frac{\partial L}{\partial v_{ij}} = \varepsilon_j x_i \Leftarrow \varepsilon_j = \delta w_j h'(r_j) \Leftarrow \frac{\partial L}{\partial w_j} = \delta u_j \Leftarrow \delta = (z - y)$$



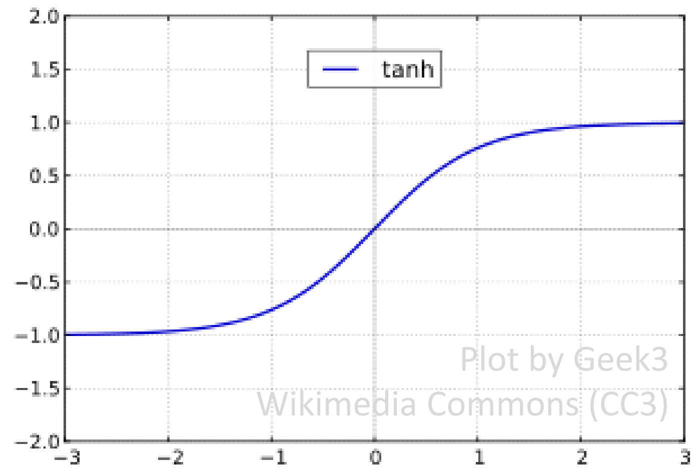
- Backpropagation equations
  - $\frac{\partial L}{\partial w_j} = \delta u_j = (z - y) u_j$
  - $\frac{\partial L}{\partial v_{ij}} = \varepsilon_j x_i = \delta w_j h'(r_j) x_i$

Some further notes on ANN training

- ANN's are flexible (recall universal approximation theorem), but the flipside is over-parameterisation, hence tendency to overfitting

- Starting weights usually random distributed about zero

- Implicit regularisation: early stopping
  - With some activation functions, this shrinks the ANN towards a linear model (why?)



Explicit regularisation

- Alternatively, an explicit regularisation can be used, much like in ridge regression
- Instead of minimising the loss L, minimise regularised function $L + \lambda\left(\sum_{i=0}^{m}\sum_{j=1}^{p} v_{ij}^2 + \sum_{j=0}^{p} w_j^2\right)$
- This will simply add $2\lambda v_{ij}$ and $2\lambda w_j$ terms to the partial derivatives
- With some activation functions this also shrinks the ANN towards a linear model

This lecture

- Multilayer perceptron
  - Model structure
  - Universal approximation
  - Training preliminaries
- Backpropagation
  - Step-by-step derivation
  - Notes on regularisation
- Next lecture: DNNs, CNNs, autoencoders