Lecture 11. Neural Network Fundamentals

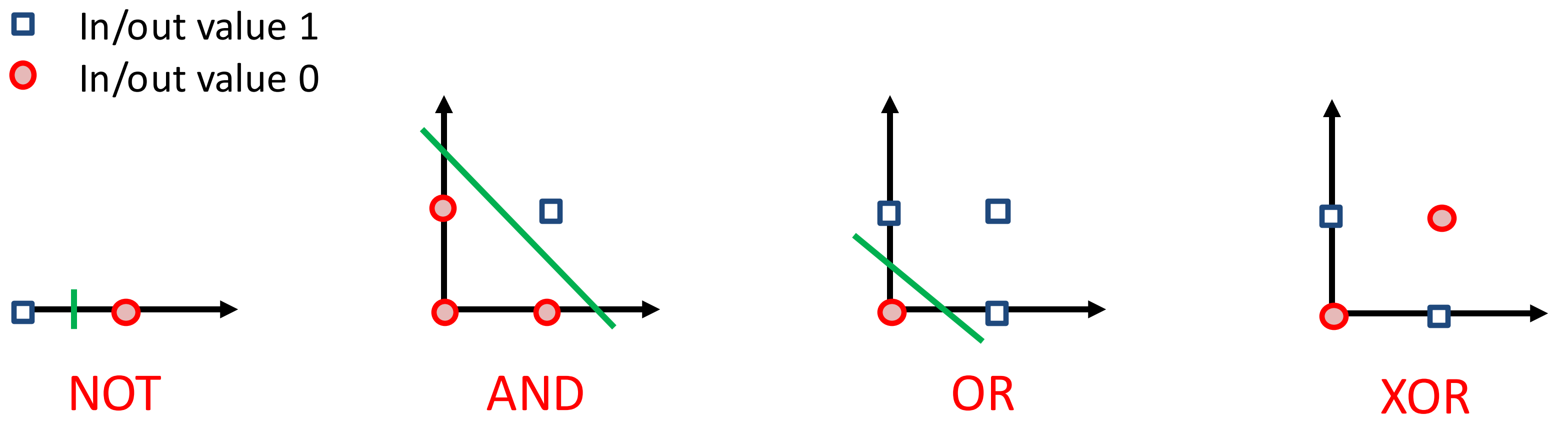
COMP90051 Statistical Machine Learning

Lecturer: Zahra Dasht Bozorgi

THE UNIVERSITY OF MELBOURNE

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 …

Perceptron is *sort of* a building block for ANN

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

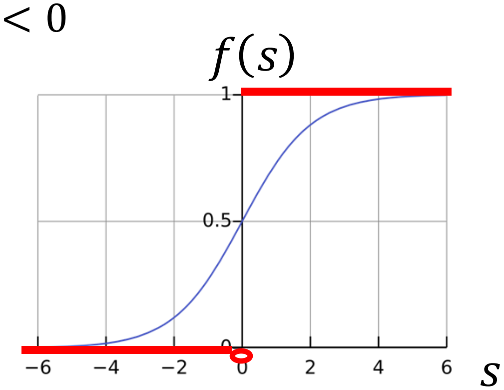
Step function $f(s) = \begin{cases} 1, & if\ s \geq 0 \\ 0, & if\ s < 0 \end{cases}$

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

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

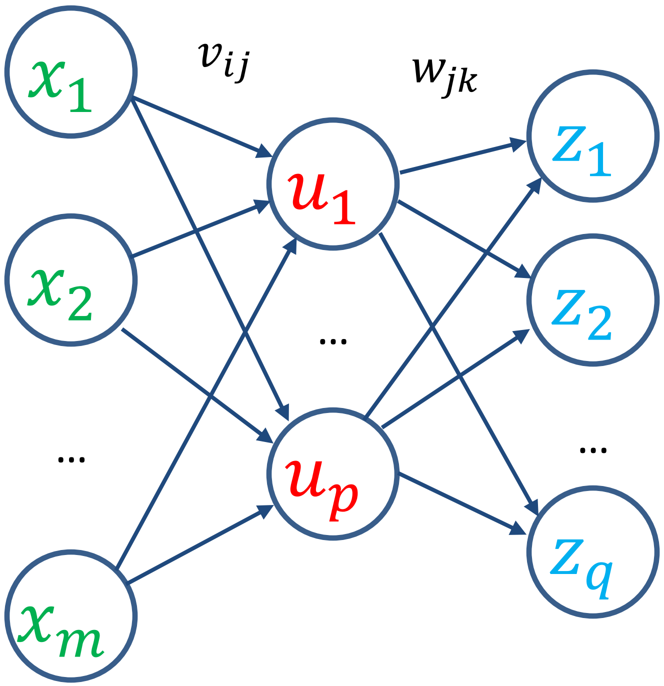
tanh function $f(s) = \dfrac{e^s - e^{-s}}{e^s + e^{-s}}$

Rectified linear unit $f(s) = \max\{0, s\}$



...many others, many variations...

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.)

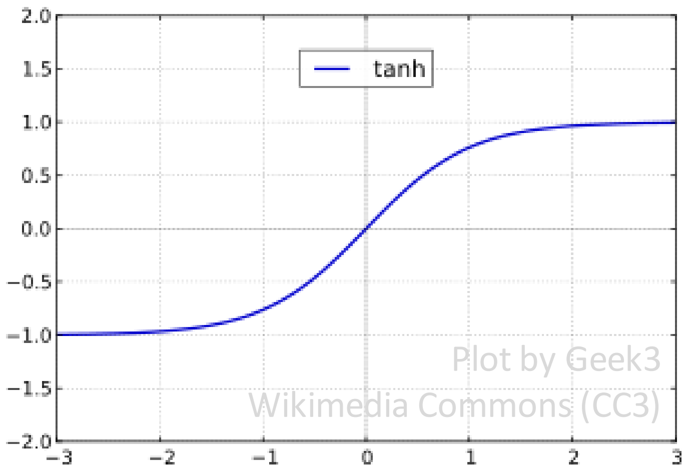
here g, h are activation functions. These can be either same (e.g., both sigmoid) or different

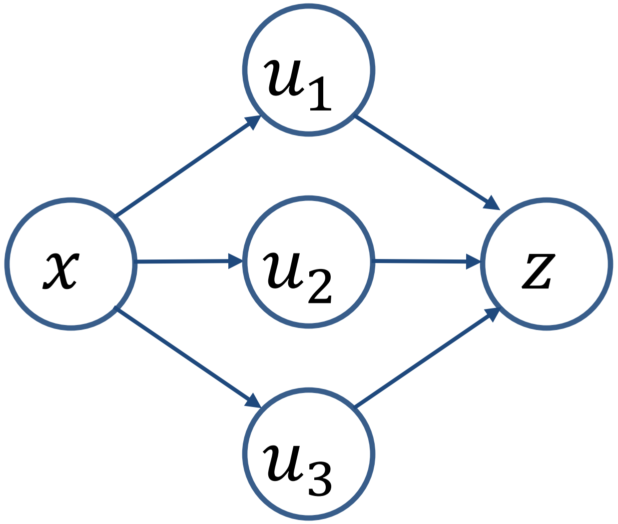
you can add bias node $x_0 = 1$ to simplify equations: $r_j = \sum_{i=0}^{m} x_i v_{ij}$

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

The power of ANN as a non-linear model

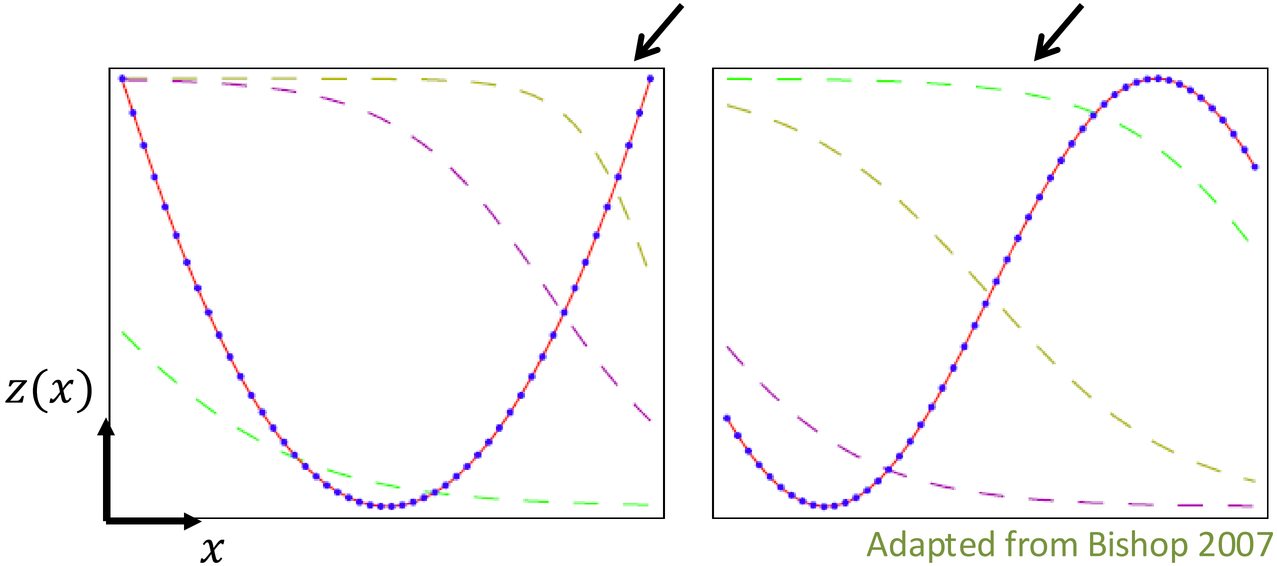
- ANNs are capable of approximating plethora non-linear functions, e.g., $z(x) = x^2$ and $z(x) = \sin x$
- For example, consider the following network. In this example, hidden unit activation functions are tanh





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 $\mathbb{R}^n$ arbitrarily well

Representational capacity

- ANNs with a single hidden layer are universal approximators
- For example, such ANNs can represent any Boolean function

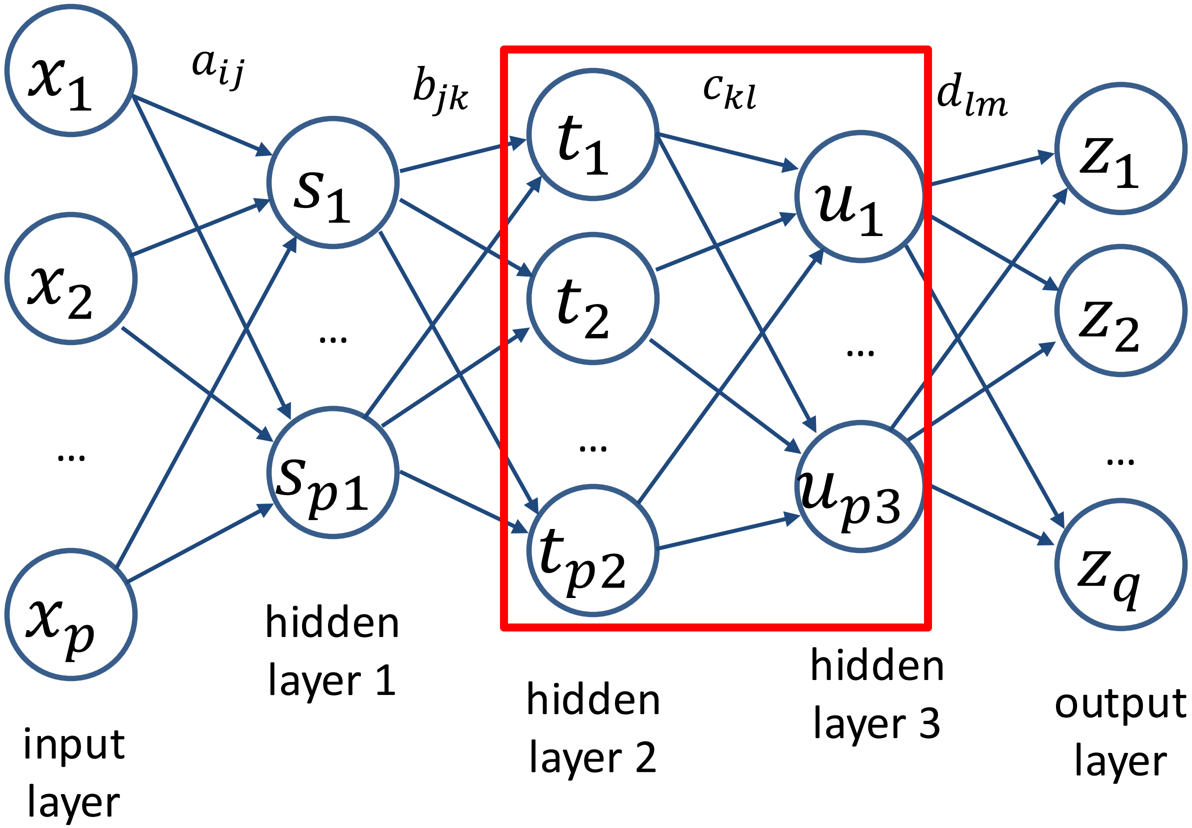
$OR(x_1, x_2)$	$u = g(x_1 + x_2 - 0.5)$
$AND(x_1, x_2)$	$u = g(x_1 + x_2 - 1.5)$
$NOT(x_1)$	$u = g(-x_1)$

$g(r) = 1$ if $r \geq 0$ and $g(r) = 0$ otherwise

- Any Boolean function over m variables can be implemented using a hidden layer with up to 2^m elements
- More *efficient to stack* several hidden layers

"Depth" refers to number of hidden layers

Deep networks



$\boldsymbol{s} = \tanh(\boldsymbol{A}'\boldsymbol{x})$ $\quad$ $\boldsymbol{t} = \tanh(\boldsymbol{B}'\boldsymbol{s})$ $\quad$ $\boldsymbol{u} = \tanh(\boldsymbol{C}'\boldsymbol{t})$ $\quad$ $\boldsymbol{z} = \tanh(\boldsymbol{D}'\boldsymbol{u})$

Deep ANNs as representation learning

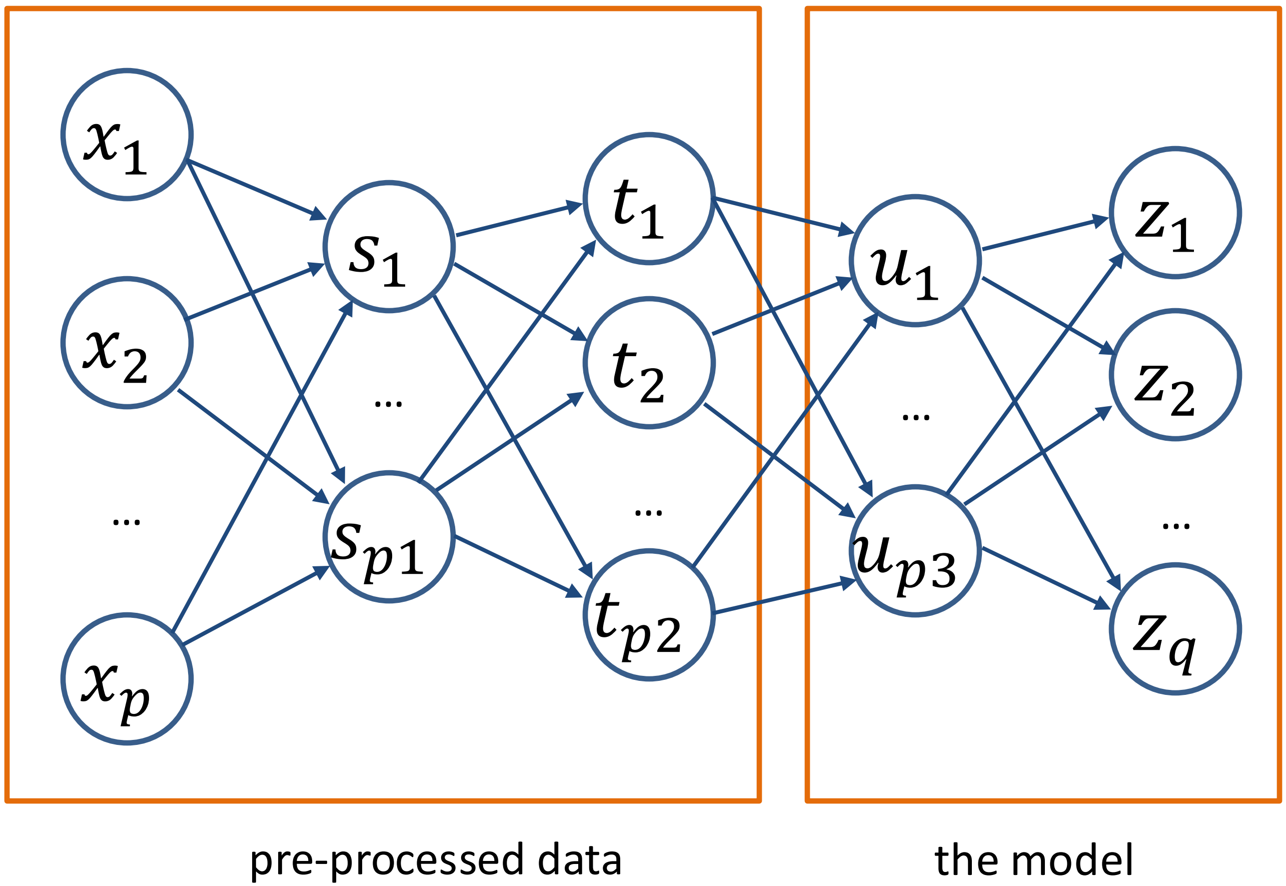
- Consecutive layers form representations of the input of increasing complexity
- An ANN can have a simple *linear* output layer, but using complex *non-linear* representation

$$\boldsymbol{z} = \tanh\left(\boldsymbol{D}'\left(\tanh\left(\boldsymbol{C}'\left(\tanh\left(\boldsymbol{B}'\left(\tanh\left(\boldsymbol{A}'\boldsymbol{x}\right)\right)\right)\right)\right)\right)\right)$$

- Equivalently, a hidden layer can be thought of as the transformed feature space, e.g., $\boldsymbol{u} = \varphi(\boldsymbol{x})$
compare to basis expansion / kernel learning
- Parameters of such a transformation are learned from data

Bias terms are omitted for simplicity

ANN layers as data transformation



Depth vs width

- A single arbitrarily wide layer in theory gives a universal approximator
- However (empirically) depth yields more accurate models

  Biological inspiration from the eye:
  * first detect small edges and color patches;
  * compose these into smaller shapes;
  * building to more complex detectors, of e.g. textures, faces, etc.
- Seek to mimic layered complexity in a network
- However vanishing gradient problem affects learning with very deep models
- Training error decreases but then increases and converges, vanishing gradient and large learning rate

Recap: Stochastic gradient descent training

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 $\left\{\boldsymbol{x}_j, y_j\right\}$

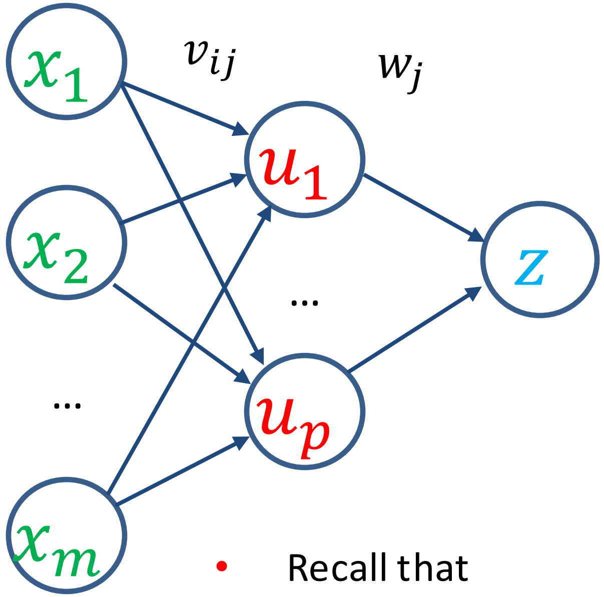
<u>Update</u>: $\boldsymbol{\theta}^{(\boldsymbol{k}+1)} = \boldsymbol{\theta}^{(\boldsymbol{k})} - \eta\nabla L\left(\boldsymbol{\theta}^{(\boldsymbol{k})}\right)$; $k \leftarrow k+1$

For regression we might use $L = \frac{1}{2}\left(z_j - y_j\right)^2$

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

Backpropagation: 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
- $\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial w_j}$
- $\frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}\frac{\partial r_j}{\partial v_{ij}}$



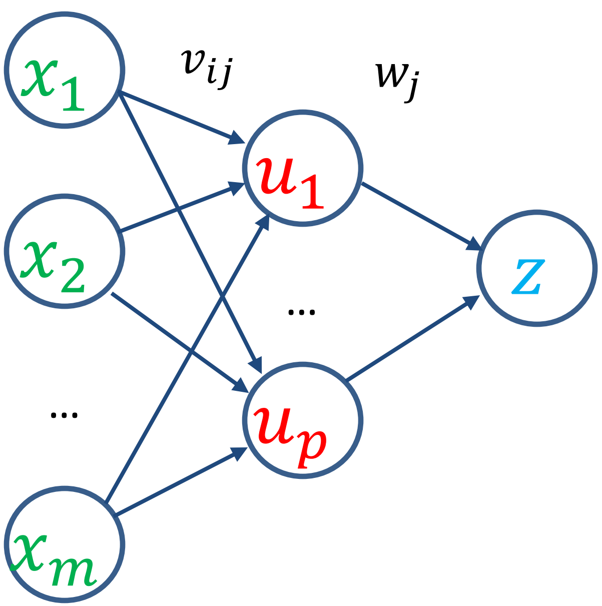
- Recall that
  - $s = \sum_{j=0}^{p} u_j w_j$
  - $r_j = \sum_{i=0}^{m} x_i v_{ij}$

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

$$\boxed{\delta \equiv \frac{\partial L}{\partial s} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}}$$

$$\boxed{\varepsilon_j \equiv \frac{\partial L}{\partial r_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}}$$

- 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 $u_j = g(r_j)$

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

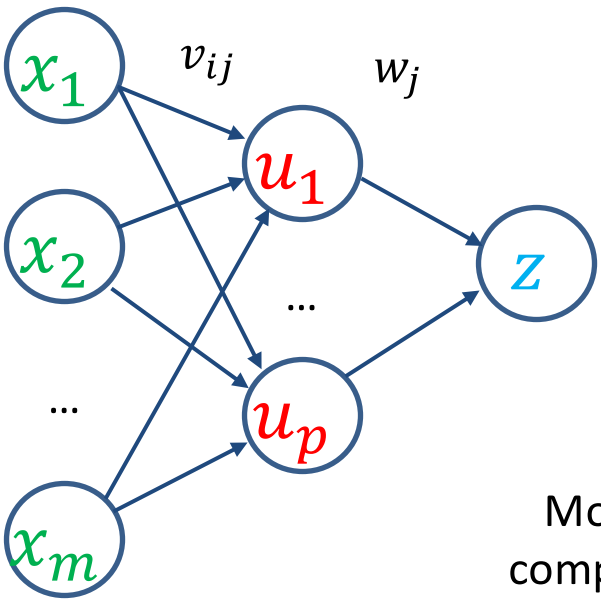
Backpropagation equations

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

… where

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

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



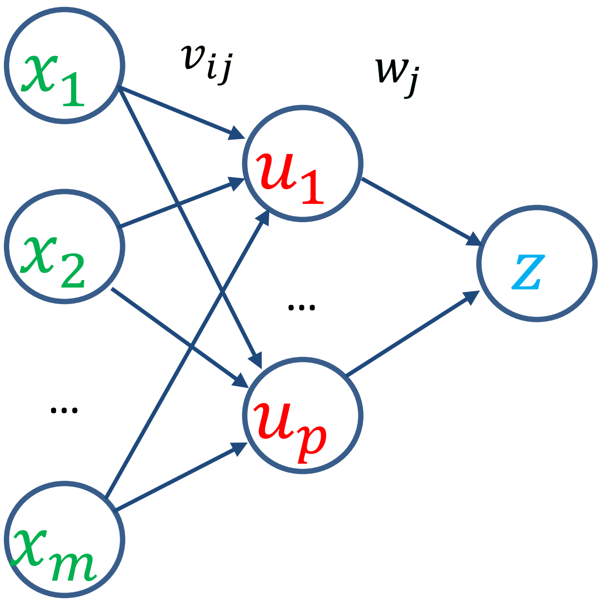
- So $\frac{\partial s}{\partial w_j} = u_j$ and $\frac{\partial r_j}{\partial v_{ij}} = x_i$
- We have
  - $\frac{\partial L}{\partial w_j} = \delta u_j = \boxed{(z - y)} u_j$
  - $\frac{\partial L}{\partial v_{ij}} = \varepsilon_j x_i = \boxed{\delta w_j g'(r_j)} x_i$

Modern DNN libraries have active derivatives pre-computed; autodiff computes derivatives efficiently

Forward propagation

- Use current estimates of v_{ij} and w_j

- Calculate r_j, u_j, s and z

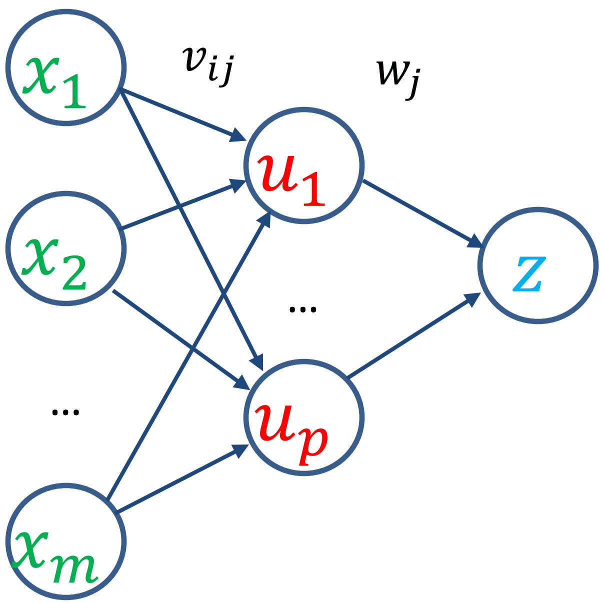


- Backpropagation equations
  * $\frac{\partial L}{\partial w_j} = \delta u_j = \boxed{(z-y)} u_j$
  * $\frac{\partial L}{\partial v_{ij}} = \varepsilon_j x_i = \boxed{\delta w_j g'(r_j)} x_i$

Backward propagation of errors

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

Notice how intermediate values get reused.



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