NN: activation functions and learning weights

LING 570

Fei Xia

(Some slides are from Geoffrey Hinton's lectures)

Outline

- Neurons
- Learning of linear neurons
- Learning of logistic neurons **

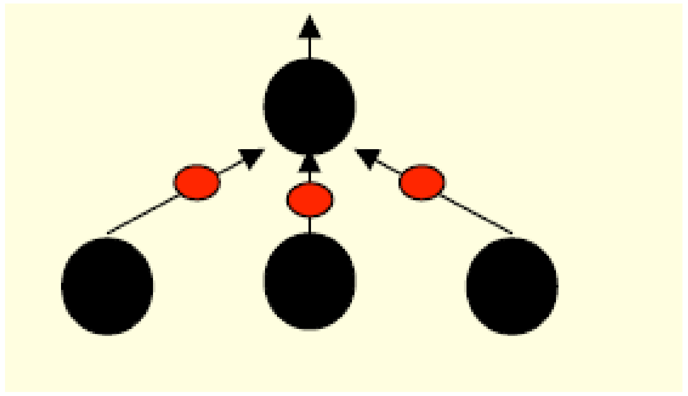
How does the brain work?

- Each neuron receives inputs from other neurons.

- The effect of each input line on the neuron is controlled by a weight.

- The weights adapt so that the whole network learns to perform useful computations.

- Our brain has about 100B neurons, each with about 10K weights

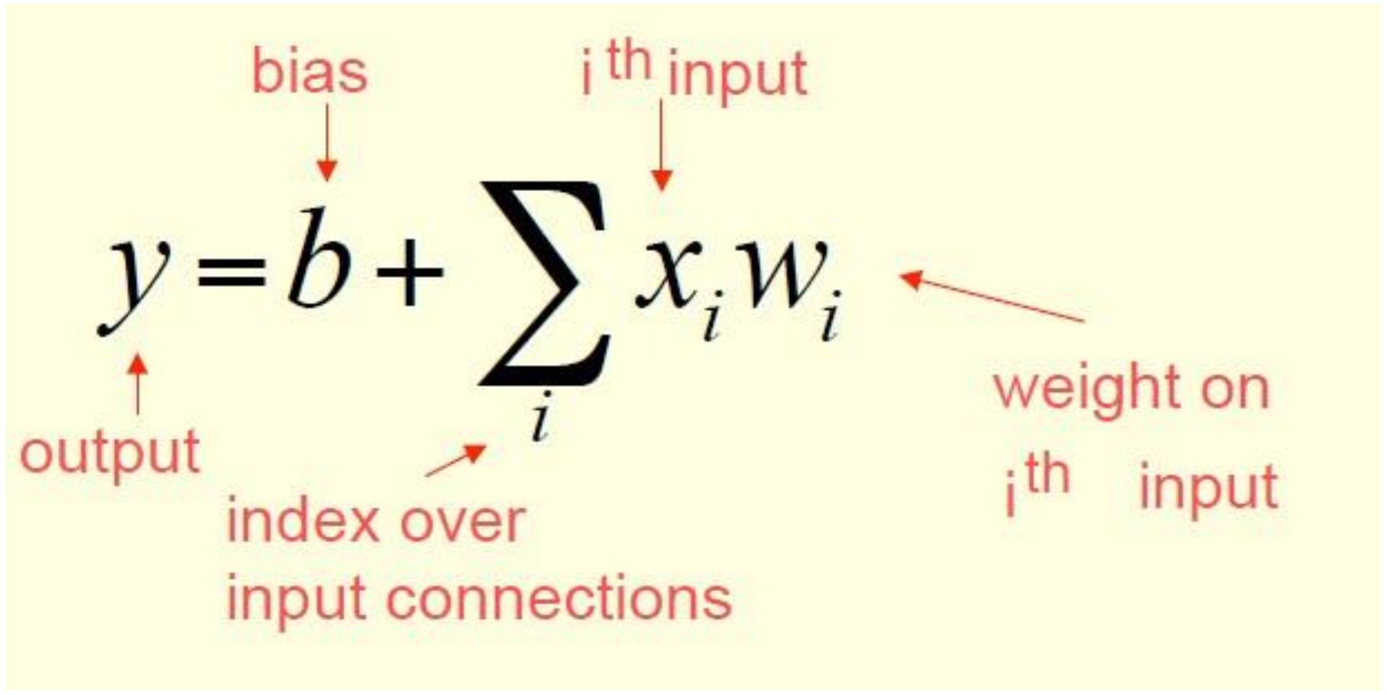


Neural network (NN)

- A modern NN is a network of small computing units.
- Each unit (a neuron) takes a vector of input values and produce a single output.
- The use of NN is often called deep learning, because modern networks are often deep (having many layers).
- Common NN models:
    - Feedforward networks
    - Recurrent NN (RNN)
    - Recursive NN
    - Encoder-decoder model
    - CNN
    - …

Linear neurons

- They are simple but computationally limited



$$y = b + \sum_i x_i w_i$$

Binary threshold neuron

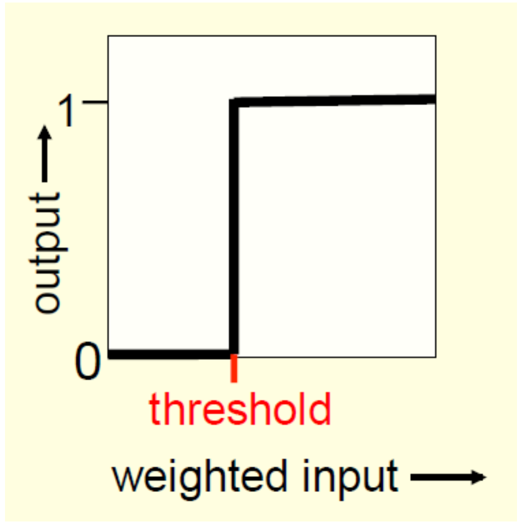
$$z = \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geq \theta \\ 0 \text{ otherwise} \end{cases}$$

$$\Updownarrow \quad \theta = -b$$

$$z = b + \sum_i x_i w_i$$

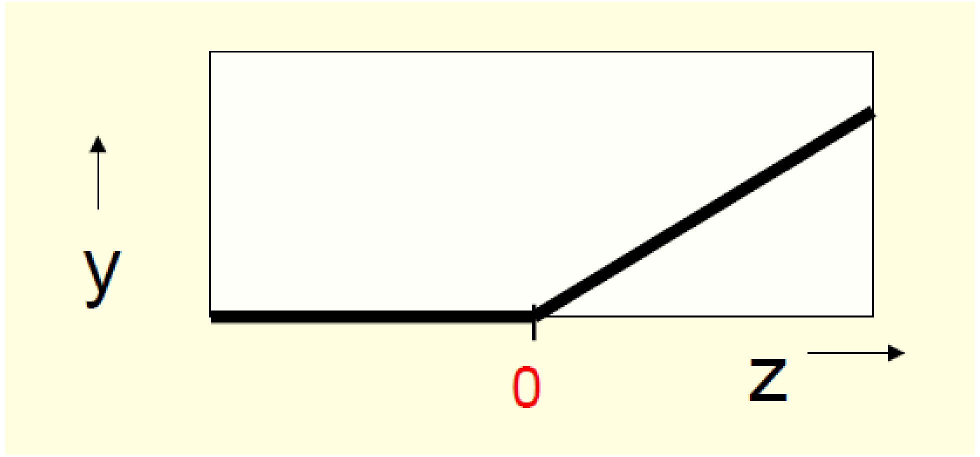
$$y = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$$



Rectified linear neurons

$$z = b + \sum_i x_i w_i$$

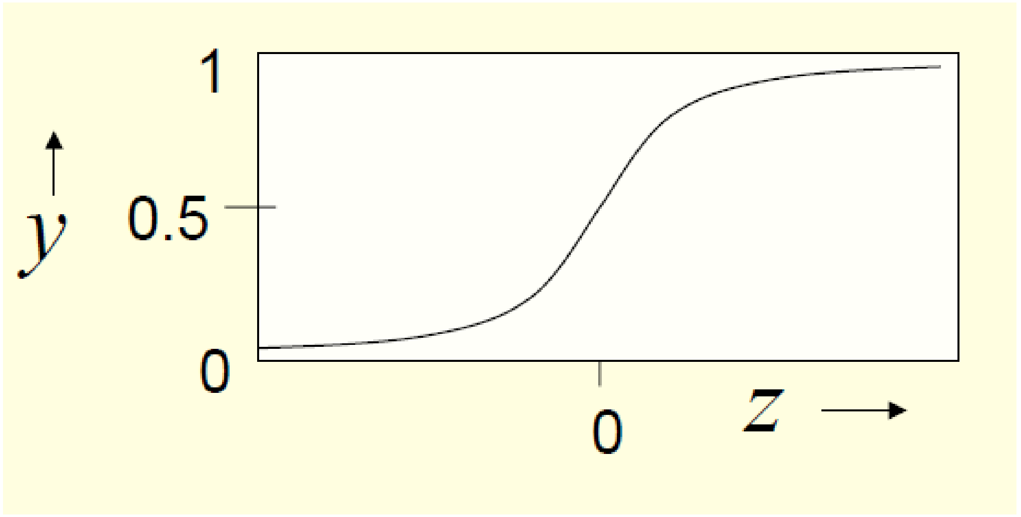
$$y = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$



Sigmoid neurons

- They give a real-valued output that is a smooth and bounded function of their total input:
  - They have nice derivatives which make learning easy.

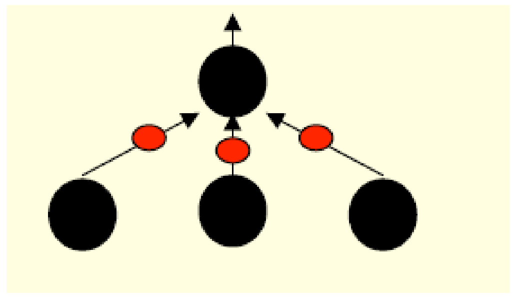
$$z = b + \sum_i x_i w_i \qquad y = \frac{1}{1 + e^{-z}}$$

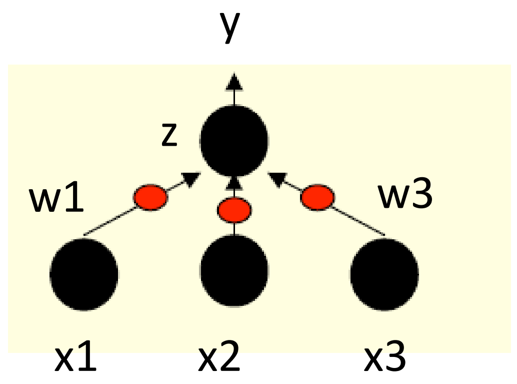


Outline

- Neurons
- Learning of linear neurons
- Learning of logistic neurons **

Learn the weights of linear neuron





bias

i th input

$$y = b + \sum_i x_i w_i$$

output

index over input connections

weight on i th input

weight vector

$$y = \sum_i w_i x_i = \mathbf{w}^T \mathbf{x}$$

neuron's estimate of the desired output

input vector

A toy example

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and ketchup.
  - You get several portions of each.
- The cashier only tells you the total price of the meal
  - After several days, you should be able to figure out the price of each portion.
- The iterative approach: Start with random guesses for the prices and then adjust them to get a better fit to the observed prices of whole meals.

Solving the equations iteratively

- Each meal price gives a linear constraint on the prices of the portions:

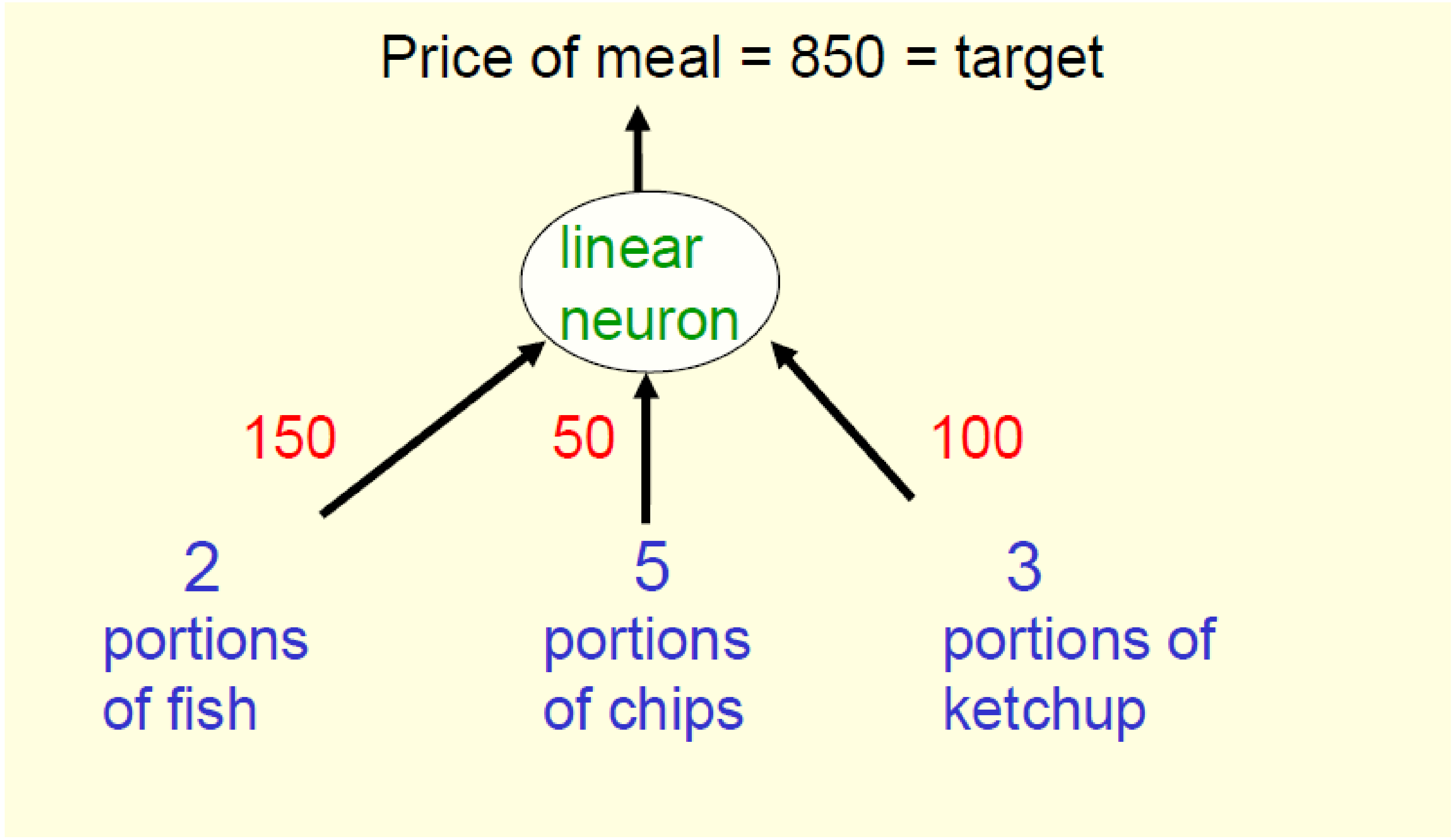
$$price = x_{fish} w_{fish} + x_{chips} w_{chips} + x_{ketchup} w_{ketchup}$$

- The prices of the portions are like the weights in of a linear neuron.

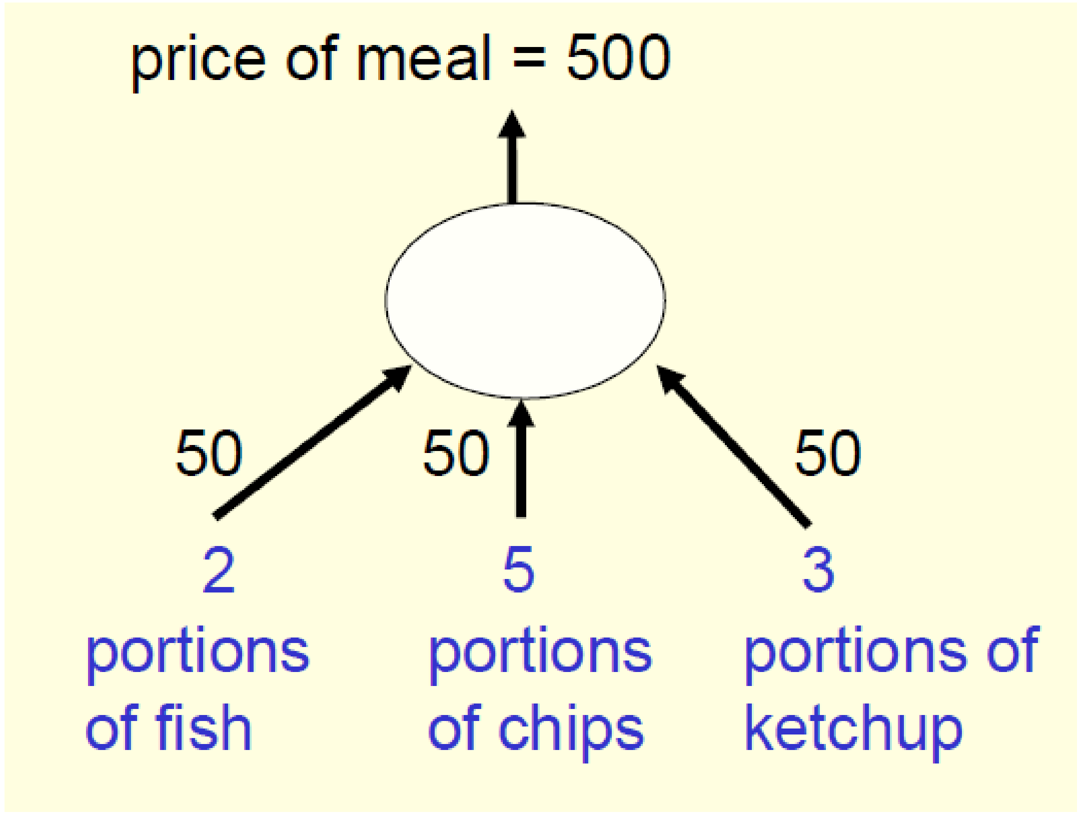
$$\mathbf{w} = (w_{fish}, w_{chips}, w_{ketchup})$$

- We will start with guesses for the weights and then adjust the guesses slightly to give a better fit to the prices given by the cashier.

The true weights used by the cashier



Start with arbitrary initial weights



- Residual error = 350
- The "delta-rule" for learning is:
  $\Delta w_i = \varepsilon \, x_i \, (t - y)$
- With a learning rate ε of 1/35, the weight changes are +20, +50, +30
- This gives new weights of 70, 100, 80.
  - Notice that the weight for chips got worse!

➔ Repeat until the residual error is small enough

Deriving the delta rule

- Define the error as the squared residuals summed over all training cases: $\rightarrow$ $$E = \frac{1}{2} \sum_{n \in training} (t^n - y^n)^2$$
- Now differentiate to get error derivatives for weights $\rightarrow$ $$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_n \frac{\partial y^n}{\partial w_i} \frac{dE^n}{dy^n}$$ $$= -\sum_n x_i^n (t^n - y^n)$$
- The batch delta rule changes the weights in proportion to their error derivatives summed over all training cases $\rightarrow$ $$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i} = \sum_n \varepsilon\, x_i^n (t^n - y^n)$$

Behavior of the iterative learning procedure

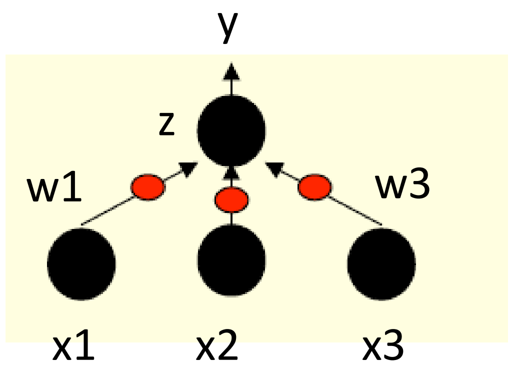
- Does the learning procedure eventually get the right answer?
  - There may be no perfect answer.
  - By making the learning rate small enough we can get as close as we desire to the best answer.

- How quickly do the weights converge to their correct values?
  - It can be very slow if two input dimensions are highly correlated. If you almost always have the same number of portions of ketchup and chips, it is hard to decide how to divide the price between ketchup and chips.

Outline

- Neurons
- Learning of linear neurons
- Learning of logistic neurons **

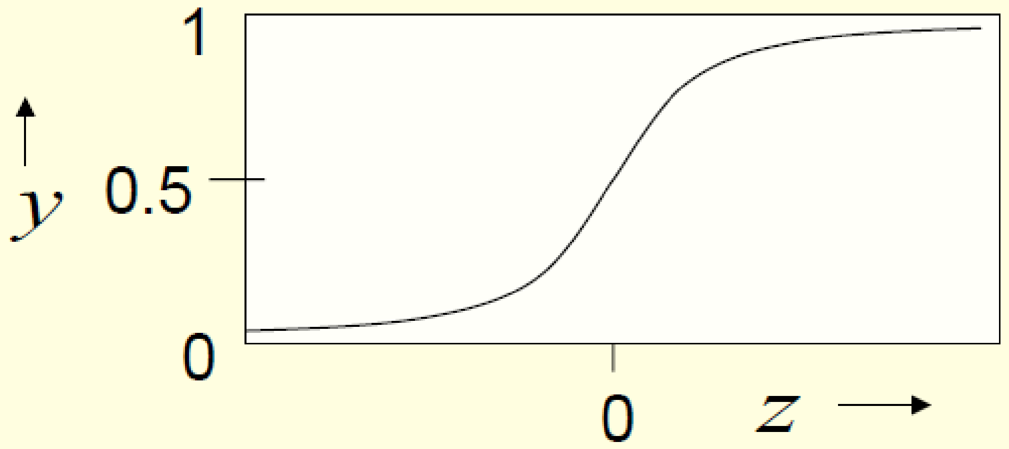
Logistic neurons



- These give a real-valued output that is a smooth and bounded function of their total input.
  - They have nice derivatives which make learning easy.

$$z = b + \sum_i x_i w_i$$

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



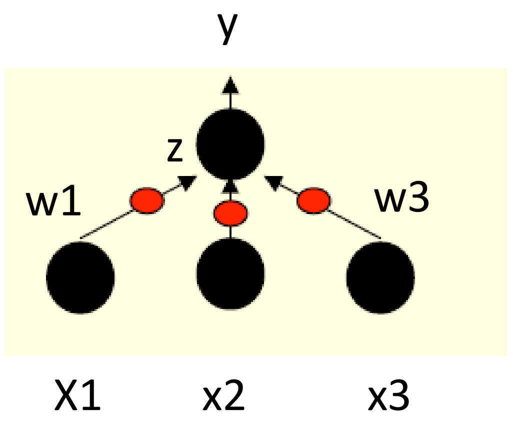
The derivatives of a logistic function

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

$$\frac{dy}{dz} = \frac{-1(-e^{-z})}{(1+e^{-z})^2} = \left(\frac{1}{1+e^{-z}}\right)\left(\frac{e^{-z}}{1+e^{-z}}\right) = y(1-y)$$

because $\dfrac{e^{-z}}{1+e^{-z}} = \dfrac{(1+e^{-z})-1}{1+e^{-z}} = \dfrac{(1+e^{-z})}{1+e^{-z}}\,\dfrac{-1}{1+e^{-z}} = 1-y$

Use the chain rule



- To learn the weights we need the derivative of the output with respect to each weight:

$$\frac{\partial y}{\partial w_i} = \frac{\partial z}{\partial w_i}\frac{dy}{dz} = x_i\, y\,(1-y)$$

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^n}{\partial w_i}\frac{\partial E}{\partial y^n} = -\sum_n \underbrace{\boxed{x_i^n}}_{} \boxed{y^n\ (1-y^n)} \boxed{(t^n - y^n)}$$

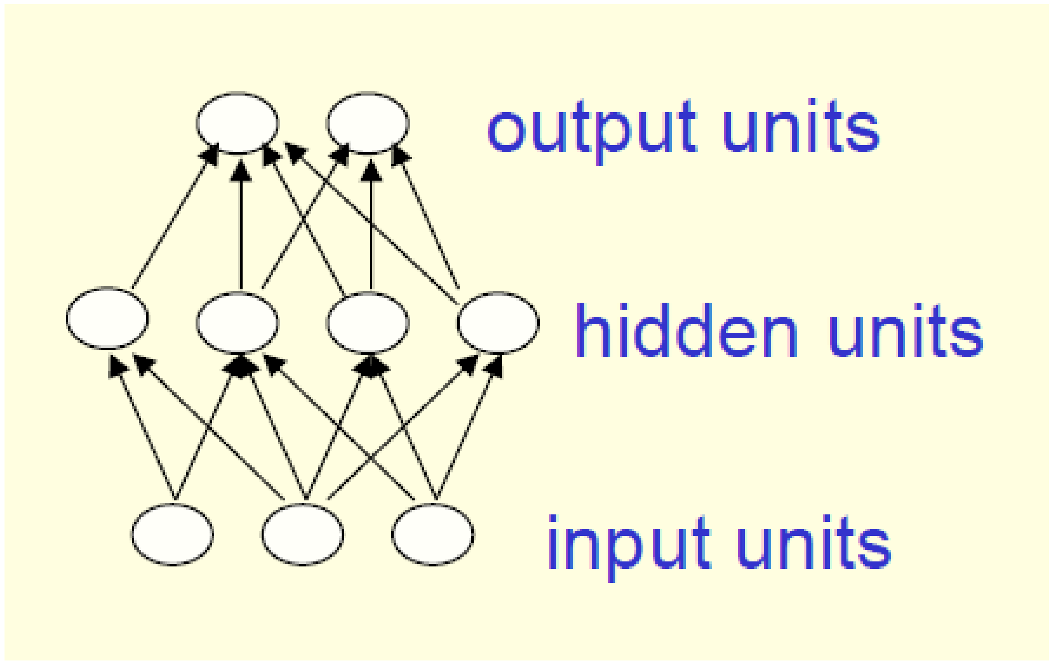
delta-rule

extra term = slope of logistic

Feed-forward neural network

- This is the simplest type of NN:
  - The first layer is the input and the last layer is the output
  - If there is more than one hidden layer, we call them deep NN

- Training: learn the weights on the arcs, using back propagation.



Learning with hidden units

- Networks without hidden units are very limited in the input-output mappings they can learn to model.
- More layers of linear units do not help. It’s still linear.
- We need multiple layers of adaptive, non-linear hidden units:
  - We need an efficient way of adapting all the weights, not just the last layer.
  - Learning the weights going into hidden units is equivalent to learning features.
  - This can be difficult because nobody is telling us directly what the hidden units should do.

Summary

- There are many kinds of neurons (i.e., using different activation functions).
- One important aspect of NNs is its non-linear activation functions.
- Learn the weights of NN using back propagation.