Why the perceptron learning procedure cannot be generalised to hidden layers

- The perceptron convergence procedure works by ensuring that every time the weights change, they get closer to every “generously feasible” set of weights.
  - This type of guarantee cannot be extended to more complex networks in which the average of two good solutions may be a bad solution.
- So “multi-layer” neural networks do not use the perceptron learning procedure.
  - They should never have been called multi-layer perceptrons.

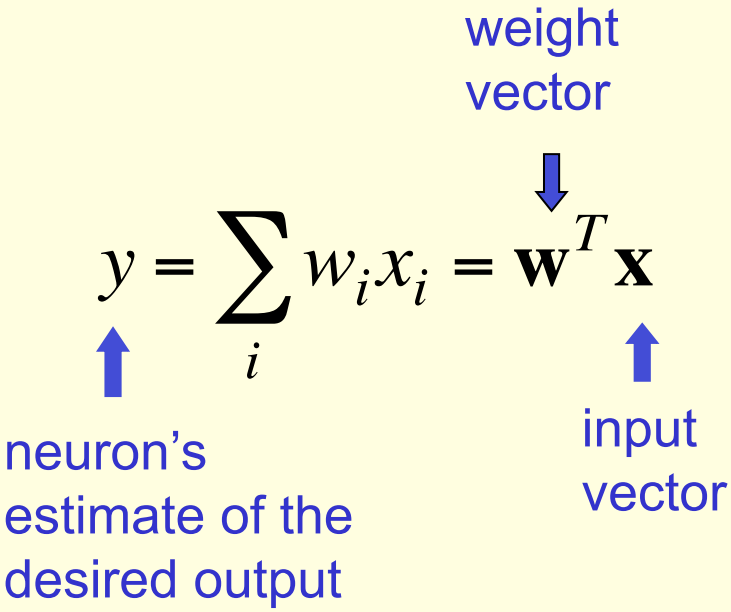
A different way to show that a learning procedure makes progress

- Instead of showing the weights get closer to a good set of weights, show that the actual output values get closer the target values.
  - This can be true even for non-convex problems in which there are many quite different sets of weights that work well and averaging two good sets of weights may give a bad set of weights.
  - It is not true for perceptron learning.

- The simplest example is a linear neuron with a squared error measure.

Linear neurons (also called linear filters)

- The neuron has a real-valued output which is a weighted sum of its inputs
- The aim of learning is to minimize the error summed over all training cases.
  - The error is the squared difference between the desired output and the actual output.



Why don't we solve it analytically?

- It is straight-forward to write down a set of equations, one per training case, and to solve for the best set of weights.
  - This is the standard engineering approach so why don't we use it?
- Scientific answer: We want a method that real neurons could use.
- Engineering answer: We want a method that can be generalized to multi-layer, non-linear neural networks.
  - The analytic solution relies on it being linear and having a squared error measure.
  - Iterative methods are usually less efficient but they are much easier to generalize.

A toy example to illustrate the iterative method

- 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:

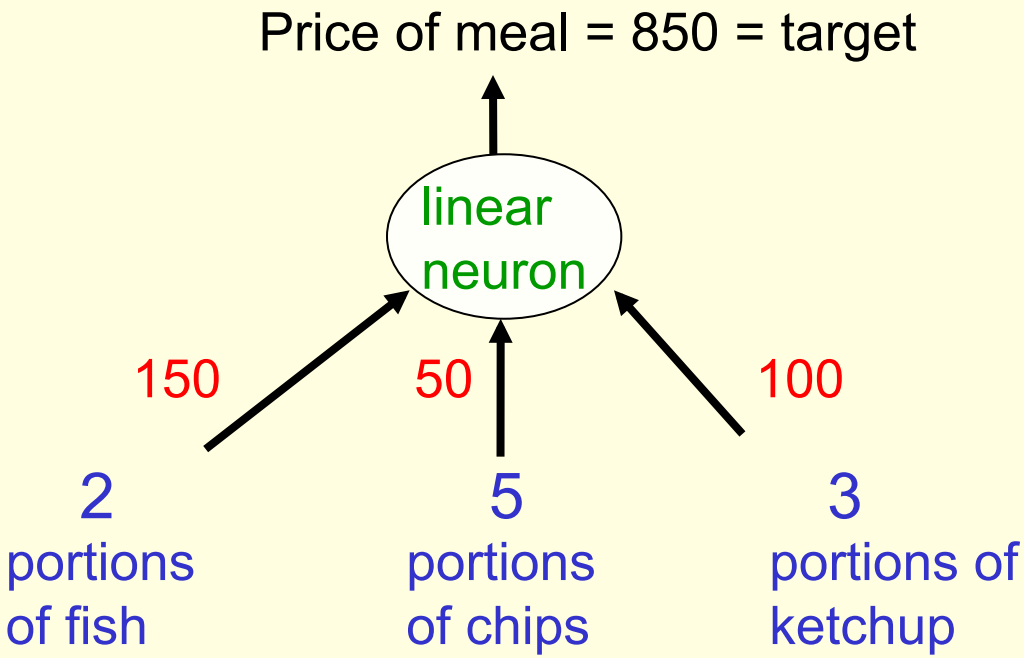
$$price = x_{fish} w_{fish} + x_{chips} w_{chips} + x_{ketchup} w_{ketchup}$$

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

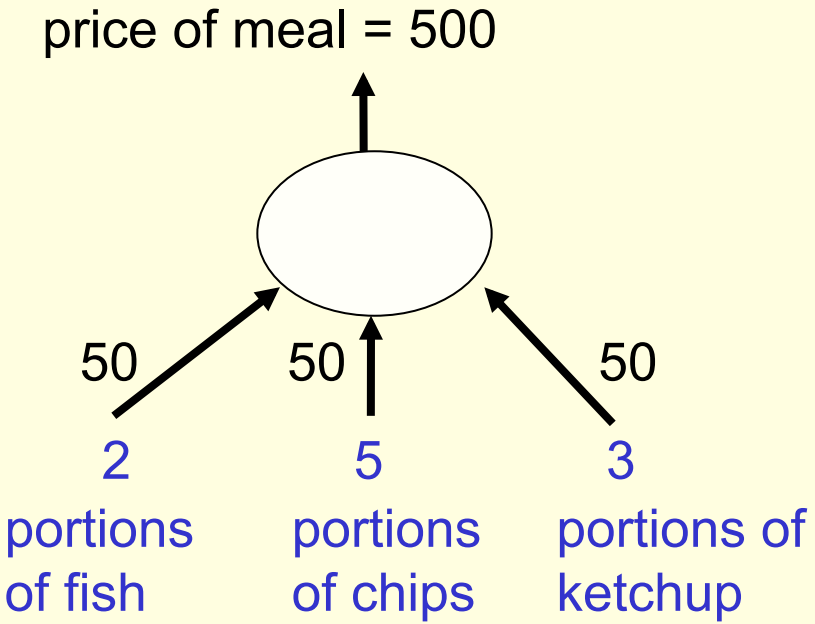
$$\mathbf{w} = (w_{fish}, w_{chips}, w_{ketchup})$$

- 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



A model of the cashier 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!

Deriving the delta rule

- Define the error as the squared residuals summed over all training cases:
- Now differentiate to get error derivatives for weights
- The batch delta rule changes the weights in proportion to their error derivatives summed over all training cases

$$E = \frac{1}{2} \sum_{n \in training} (t^n - y^n)^2$$

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_n \frac{\partial y^n}{\partial w_i} \frac{dE^n}{dy^n}$$

$$= -\sum_n x_i^n (t^n - y^n)$$

$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i} = \sum_n \varepsilon \, x_i^n (t^n - y^n)$$

Behaviour 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.

The relationship between the online delta-rule and the learning rule for perceptrons

- In perceptron learning, we increment or decrement the weight vector by the input vector.
  - But we only change the weights when we make an error.

- In the online version of the delta-rule we increment or decrement the weight vector by the input vector scaled by the residual error and the learning rate.
  - So we have to choose a learning rate. This is annoying.