Neural Networks for Machine Learning

Lecture 3a
Learning the weights of a linear neuron

Geoffrey Hinton
with
Nitish Srivastava
Kevin Swersky

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.

weight vector

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

neuron's estimate of the desired output

input vector

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:

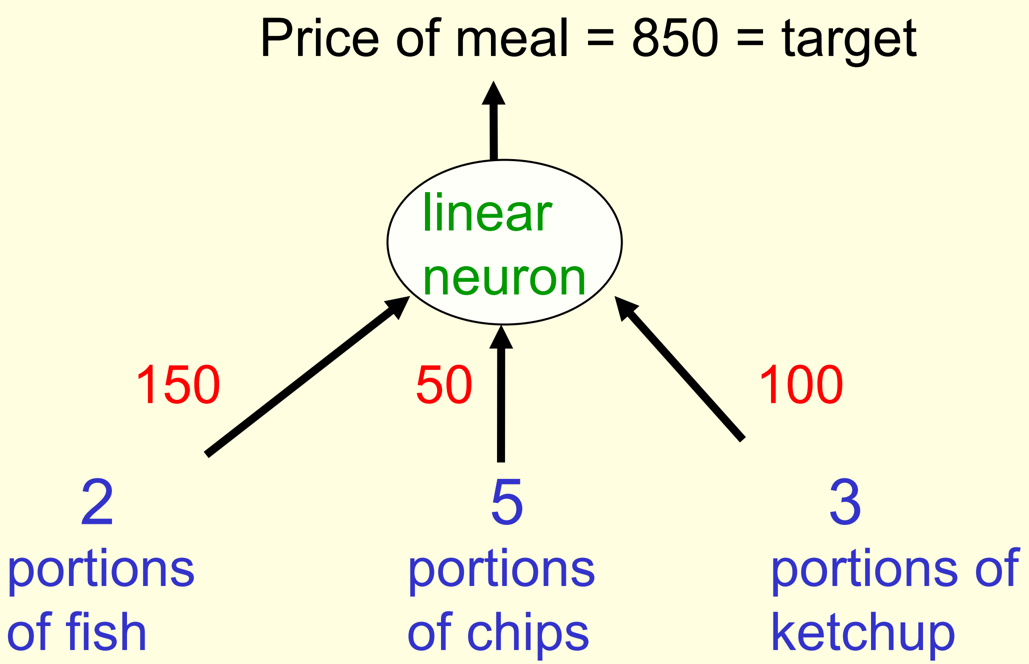
$$price = x_{fish} w_{fish} + x_{chips} w_{chips} + x_{ketchup} w_{ketchup}$$

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

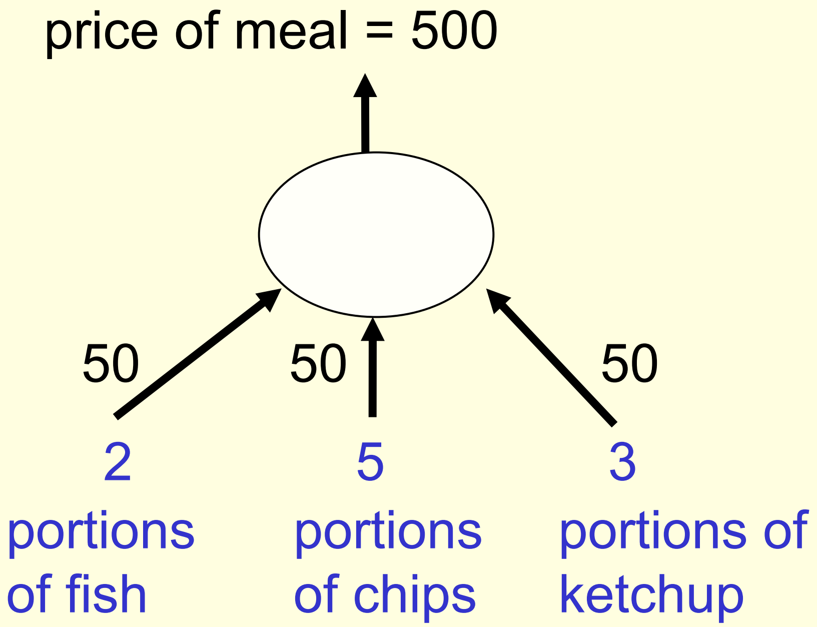
$$\mathbf{w} = (w_{fish}, w_{chips}, w_{ketchup})$$

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

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.

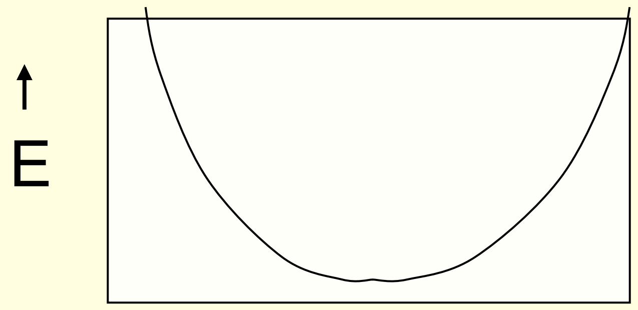
Neural Networks for Machine Learning

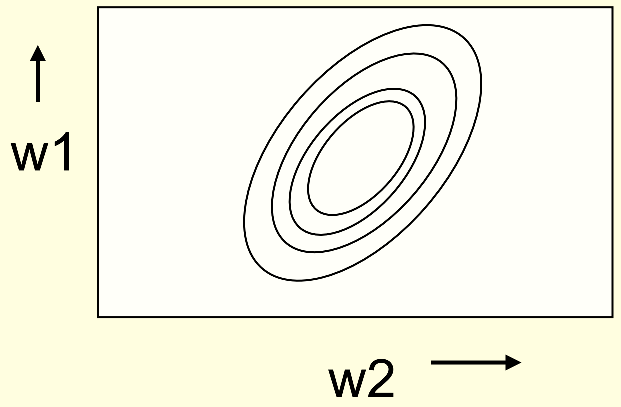
Lecture 3b
The error surface for a linear neuron

Geoffrey Hinton
with
Nitish Srivastava
Kevin Swersky

The error surface in extended weight space

- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
  - For a linear neuron with a squared error, it is a quadratic bowl.
  - Vertical cross-sections are parabolas.
  - Horizontal cross-sections are ellipses.
- For multi-layer, non-linear nets the error surface is much more complicated.

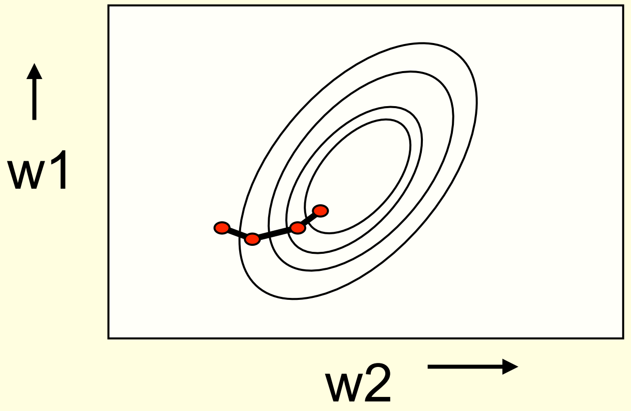


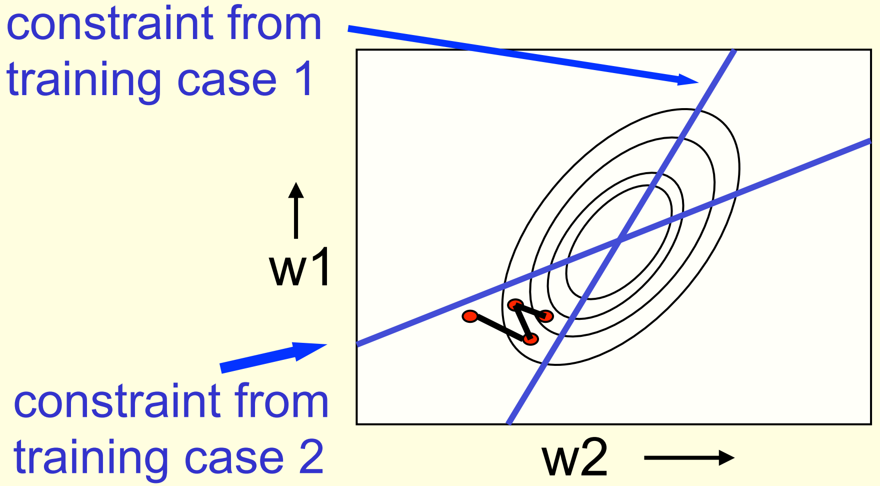


Online versus batch learning

- The simplest kind of batch learning does steepest descent on the error surface.
  - This travels perpendicular to the contour lines.

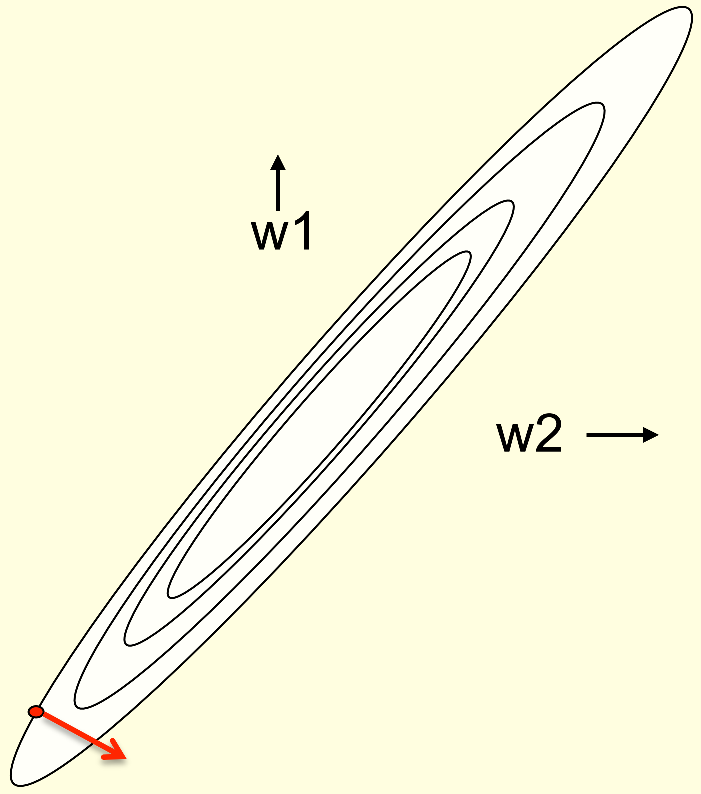
- The simplest kind of online learning zig-zags around the direction of steepest descent:




constraint from training case 1
constraint from training case 2
w1
w2

Why learning can be slow

- If the ellipse is very elongated, the direction of steepest descent is almost perpendicular to the direction towards the minimum!
  - The red gradient vector has a large component along the short axis of the ellipse and a small component along the long axis of the ellipse.
  - This is just the opposite of what we want.



Neural Networks for Machine Learning

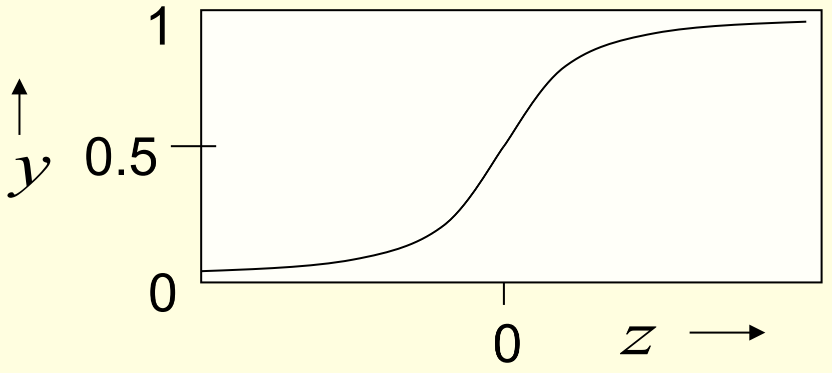
Lecture 3c
Learning the weights of a logistic output neuron

Geoffrey Hinton
with
Nitish Srivastava
Kevin Swersky

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.

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



The derivatives of a logistic neuron

- The derivatives of the logit, z, with respect to the inputs and the weights are very simple:

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

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i$$

- The derivative of the output with respect to the logit is simple if you express it in terms of the output:

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

$$\frac{dy}{dz} = y\,(1 - y)$$

The derivatives of a logistic neuron

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

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

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

Using the chain rule to get the derivatives needed for learning the weights of a logistic unit

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

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

delta-rule

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

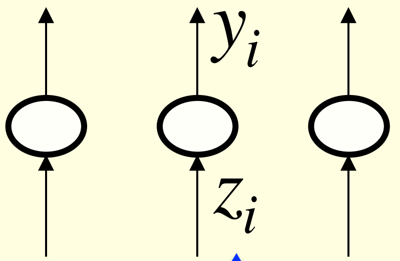
extra term = slope of logistic

Problems with squared error

- The squared error measure has some drawbacks:
  - If the desired output is 1 and the actual output is 0.00000001 there is almost no gradient for a logistic unit to fix up the error.
  - If we are trying to assign probabilities to mutually exclusive class labels, we know that the outputs should sum to 1, but we are depriving the network of this knowledge.
- Is there a different cost function that works better?
  - Yes: Force the outputs to represent a probability distribution across discrete alternatives.

Softmax

The output units in a softmax group use a non-local non-linearity:



softmax group

this is called the "logit"

$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

$$\frac{\partial y_i}{\partial z_i} = y_i \, (1 - y_i)$$

Cross-entropy: the right cost function to use with softmax

- The right cost function is the negative log probability of the right answer.
- C has a very big gradient when the target value is 1 and the output is almost zero.
  - A value of 0.000001 is much better than 0.000000001
  - The steepness of dC/dy exactly balances the flatness of dy/dz

$$C = -\sum_j \underset{\nwarrow}{t_j} \log y_j$$

target value

$$\frac{\partial C}{\partial z_i} = \sum_j \frac{\partial C}{\partial y_j} \frac{\partial y_j}{\partial z_i} = y_i - t_i$$