Grading> Full scale of the letter grade. Grade consist of About 25 % homework, 25% seminar summary, and 50% final-project 4-man teams. Your and my purpose is to lean, so a good effort is sufficient. 10% reduction/day for a delayed homework.
seminar summary Based on one talk at the 3-day workshop Big Data and The Earth Sciences: Grand Challenges Workshop write a one or two page summary. Due at class on 5 June.
Final project Either propose a topic before may 3. Or it will be based on my paper: Niu et al, 2017 on arXiv. We will make teams on May 3. Report due ABOUT June 10.
Howework 1 or 2 homeworks will be graded.

Lecture 5: Backpropagation

Why we need backpropagation

- Networks without hidden units are very limited in the input-output mappings they can model.
  - More layers of linear units do not help. Its still linear.
  - Fixed output non-linearities are not enough
- We need multiple layers of adaptive non-linear hidden units. This gives us a universal approximator. But how can we train such nets?
  - 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.
  - Nobody is telling us directly what hidden units should do.

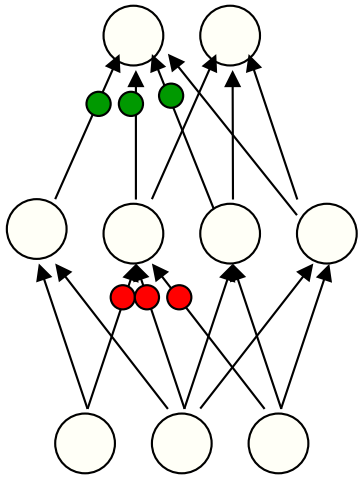
Learning by perturbing weights

Randomly perturb one weight and see if it improves performance. If so, save the change.

- Very inefficient. We need to do multiple forward passes on a representative set of training data just to change one weight.
- Towards the end of learning, large weight perturbations will nearly always make things worse.

Randomly perturb all the weights in parallel and correlate the performance gain with the weight changes.

Not any better because we need lots of trials to "see" the effect of changing one weight through the noise created by all the others.



Learning the hidden to output weights is easy. Learning the input to hidden weights is hard.

The idea behind backpropagation

- Don't know what the hidden units should be, but we can compute how fast the error changes as we change a hidden activity.
  - Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.
  - Each hidden activity affect many output units and therefore have many separate effects on the error. These effects must be combined.
  - We can compute error derivatives for all the hidden units efficiently.
  - Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

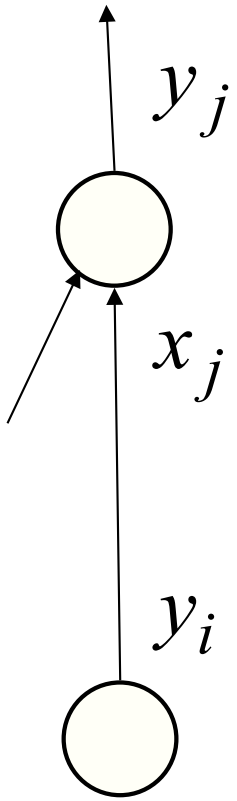
A difference in notation

- For networks with multiple hidden layers Bishop uses an explicit extra index to denote the layer.
- The lecture notes use a simpler notation in which the typical index is used to denote the layer implicitly.

  y is used for the output of a unit in any layer

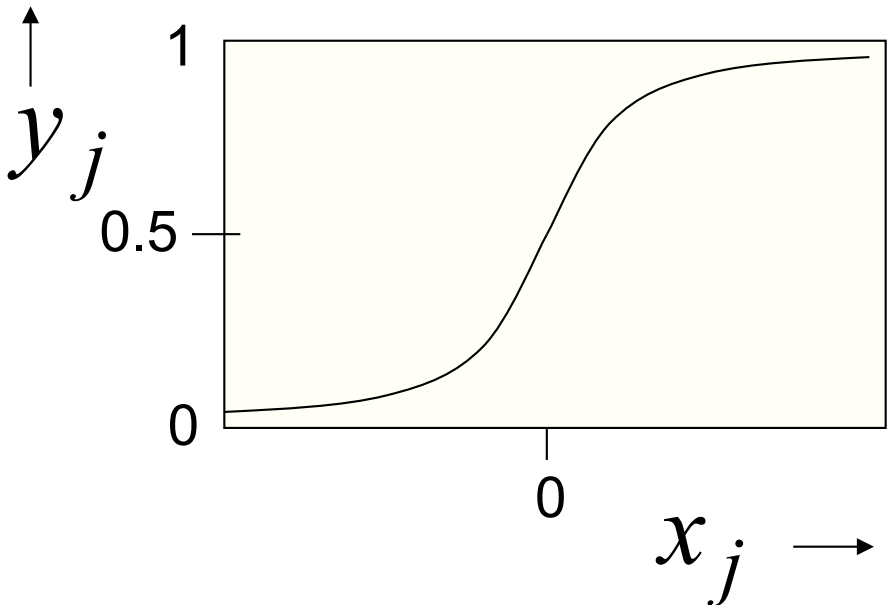
  x is the summed input to a unit in any layer

  The index indicates which layer a unit is in.



Non-linear neurons with smooth derivatives

- For backpropagation, we need neurons that have well-behaved derivatives.
  - Typically they use the logistic function
  - The output is a smooth function of inputs and weights.



$$x_j = b_j + \sum_i y_i w_{ij}$$

$$y_j = \frac{1}{1 + e^{-x_j}}$$

$$\frac{\partial x_j}{\partial w_{ij}} = y_i \qquad \frac{\partial x_j}{\partial y_i} = w_{ij}$$

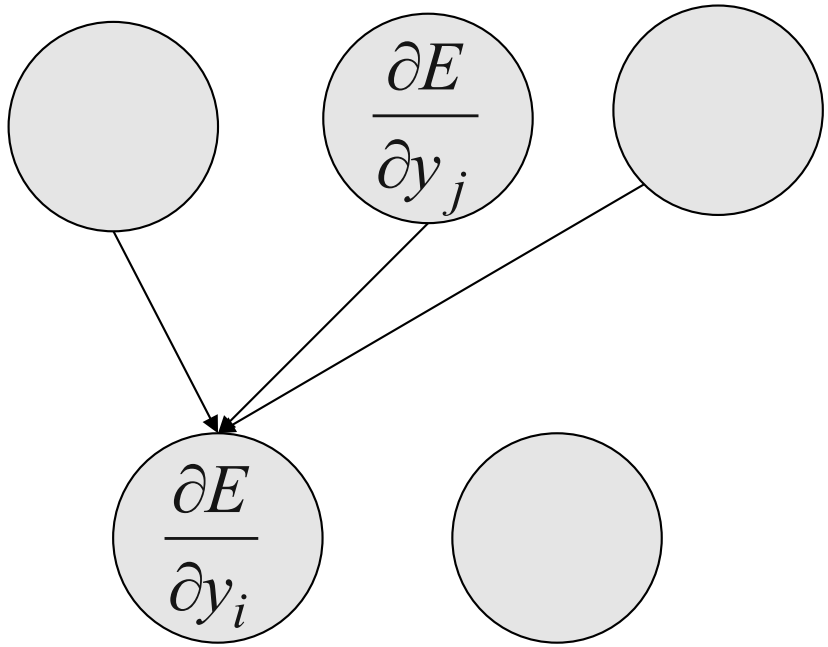
$$\frac{dy_j}{dx_j} = y_j (1 - y_j)$$

Sketch of backpropagation algorithm on a single training case

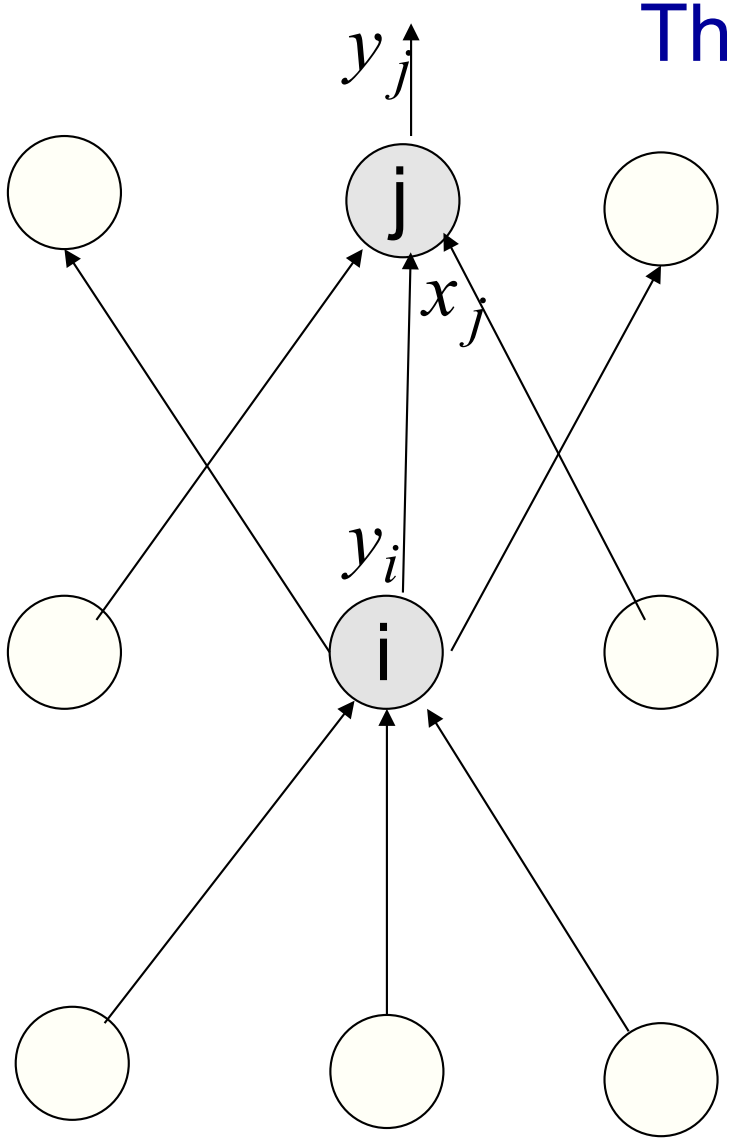
- Convert the discrepancy between each output and its target value into an error derivative.
- Then compute error derivatives in each hidden layer from error derivatives in the layer above.
- Then use error derivatives w.r.t. activities to get error derivatives w.r.t. the weights.

$$E = \sum_j \frac{1}{2}(y_j - d_j)^2$$

$$\frac{\partial E}{\partial y_j} = y_j - d_j$$



The derivatives



$$\frac{\partial E}{\partial x_j} = \frac{dy_j}{dx_j}\frac{\partial E}{\partial y_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial x_j}{\partial w_{ij}}\frac{\partial E}{\partial x_j} = y_i\frac{\partial E}{\partial x_j}$$

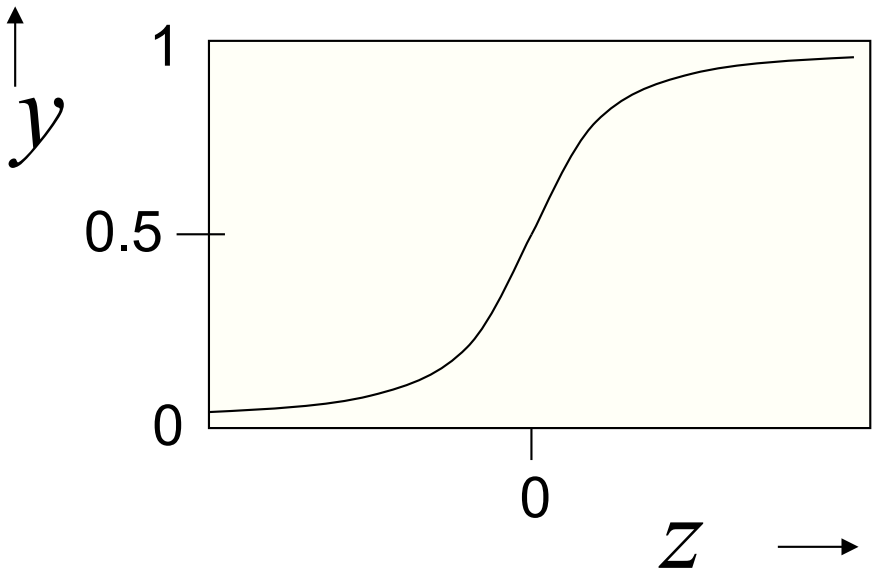
$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dx_j}{dy_i}\frac{\partial E}{\partial x_j} = \sum_j w_{ij}\frac{\partial E}{\partial x_j}$$

The perceptron convergence procedure

- Add an extra component with value 1 to each feature vector. The “bias” weight on this component is minus the threshold. Now we can forget the threshold.
- Pick training cases using any policy that ensures that every training case will keep getting picked
  - If output is correct, leave its weights alone.
  - If output is 0 but should be 1, add the feature vector to the weight vector.
  - If output is 1 but should be 0, subtract the feature vector from the weight vector
- This is guaranteed to find a set of weights that gets the right answer on the whole training set if any such set exists
- There is no need to choose a learning rate.

The logistic function

The output is a smooth function of the inputs and the weights.



$$z = \mathbf{w}^T \mathbf{x} + w_0$$

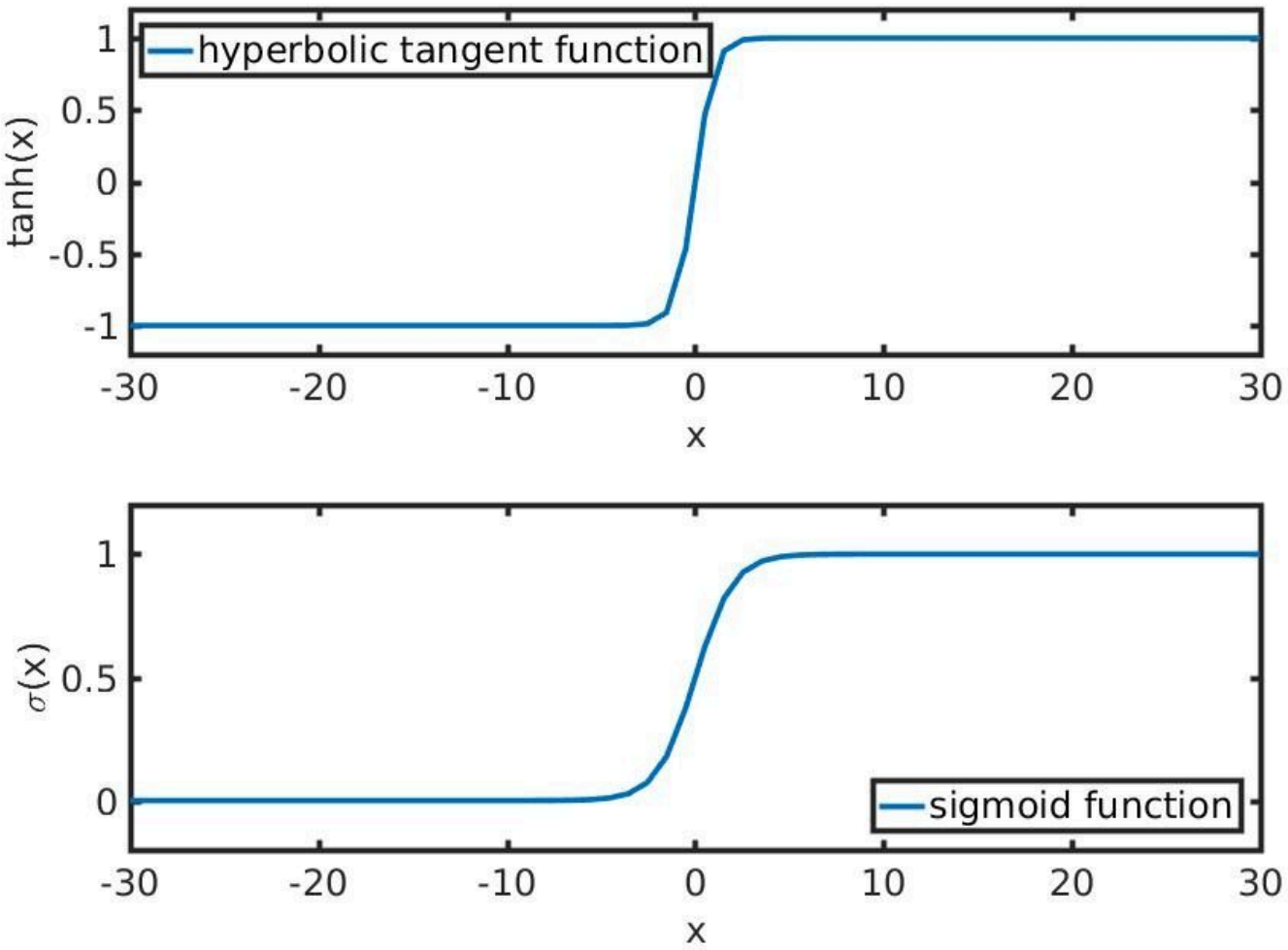
$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$

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

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

Its odd to express it in terms of y.

Tanh vs sigmoid



The natural error function for the logistic

To fit a logistic model using maximum likelihood, we need to minimize the negative log probability of the correct answer summed over the training set.

$$E = -\sum_{n=1}^{N} \ln p(t_n \mid y_n)$$

$$= -\sum_{n=1}^{N} t_n \ln y_n + (1 - t_n) \ln (1 - y_n)$$

if t =1

if t =0

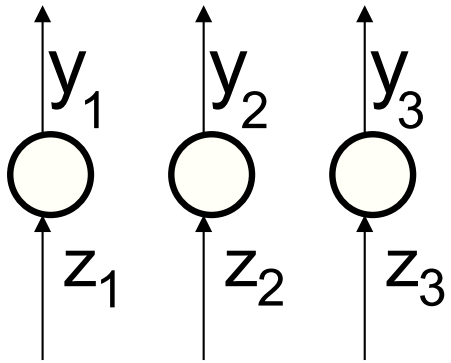
$$\frac{\partial E_n}{\partial y_n} = -\frac{t_n}{y_n} + \frac{1 - t_n}{1 - y_n}$$

$$= \frac{y_n - t_n}{y_n (1 - y_n)}$$

error derivative on training case n

Cross-entropy or "softmax" error function for multi-class classification

The output units use a non-local non-linearity:



output units

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

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

The natural cost function is the negative log prob of the right answer

target value

$$E = -\sum_j t_j \ln y_j$$

The steepness of E exactly balances the flatness of the softmax.

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

Le Net

- Yann LeCun and others developed a really good recognizer for handwritten digits by using backpropagation in a feedforward net with:
  - Many hidden layers
  - Many pools of replicated units in each layer.
  - Averaging of the outputs of nearby replicated units.
  - A wide net that can cope with several characters at once even if they overlap.
- Look at all of the demos of LENET at http://yann.lecun.com