Projects

- **3-4** person groups
- Deliverables: Poster, Report & main code (plus proposal, midterm slide)
- Topics: your own or chose form **suggested topics / kaggle**
- **Week 3 groups** due to TA Nima. Rearrangement might be needed.

- **May 2** proposal due. TAs and Peter can approve.
- Proposal: One page: Title, A large paragraph, data, weblinks, references.
- Something physical and data oriented.
- **May ~16** Midterm slides. Likely presented in 4 subgroups (3TA+Peter).
- **5pm 6 June** Jacobs Hall lobby**,** final poster session. Snacks

- Poster, Report & main code. Report due Saturday 16 June.

Logistic regression (page 205)

When there are only two classes we can model the conditional probability of the positive class as

$$p(C_1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0) \qquad where \quad \sigma(z) = \frac{1}{1+\exp(-z)}$$

If we use the right error function, something nice happens: The gradient of the logistic and the gradient of the error function cancel each other:

$$E(\mathbf{w}) = -\ln p(\mathbf{t} \mid \mathbf{w}), \qquad \nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\, \mathbf{x}_n$$

The natural error function for the logistic

Fitting logistic model using maximum likelihood, requires minimizing the negative log probability of the correct answer summed over the training set.

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

$$= -\sum_{n=1}^{N} \underbrace{t_n \ln y_n}_{\text{if } t=1} + \underbrace{(1-t_n)\ln(1-y_n)}_{\text{if } t=0}$$

error derivative on training case n

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

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

Using the chain rule to get the error derivatives

Logistic regression (Bishop 205)

$p(C_1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x})$

Observations $\{x_n, t_n\}$ $t_n \in [0, 1]$

Likelihood

$y = \sigma(\boldsymbol{w}^T\boldsymbol{x})$

$$p(y|\boldsymbol{x},\boldsymbol{w}) = Bern(y, \mu) = \mu^y(1-\mu)^{1-\mu}$$

$$p(T|\boldsymbol{x},\boldsymbol{w}) = \prod_n^N y_n^{t_n}(1-y_n)^{1-t_n}$$

Log-likelihood

$$E_{\boldsymbol{w}} = -ln(p(T|\boldsymbol{x},\boldsymbol{w})) = -\sum_n^N \left(t_n \ln y_n + (1-t_n)\ln(1-y_n)\right)$$

Minimize –log like

Derivative

$$\nabla_{\boldsymbol{w}} E_{\boldsymbol{w}} = -\sum^N \left[t_n \frac{1}{y_n} + \frac{1-t_n}{1-y_n}\cdot(-1)\right] y_n(1-y_n)\, x_n$$

$$= \sum_n^N (t_n - y_n)\, x_n$$

$$W_{i+1} = w_i - \eta \nabla E_w$$

Softmax function

For the case of $K > 2$ classes, we have

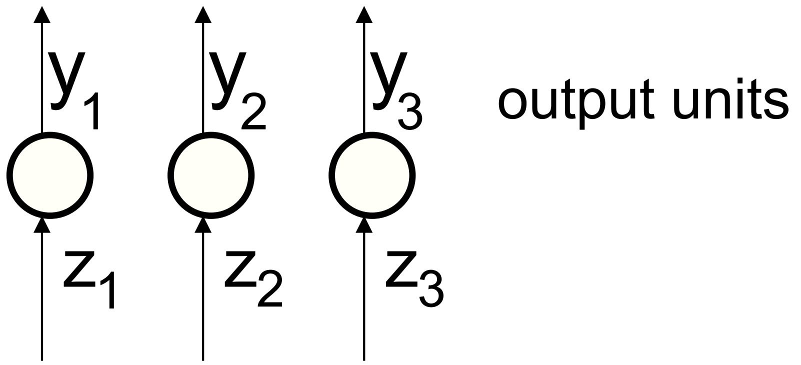
$$\begin{aligned} p(\mathcal{C}_k|\mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} \\ &= \frac{\exp(a_k)}{\sum_j \exp(a_j)} \end{aligned} \tag{4.62}$$

$$a_k = \ln p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k). \tag{4.63}$$

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

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

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



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

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

target value

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

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

A special case of softmax for two classes

$$y_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_0}} = \frac{1}{1 + e^{-(z_1 - z_0)}}$$

So the logistic is just a special case of softmax without redundant parameters:

- Adding the same constant to both z1 and z0 has no effect.
- The over-parameterization of the softmax is because the probabilities must add to 1.

4.3.3 Iterative reweighted least squares

In the case of the linear regression models discussed in Chapter 3, the maximum likelihood solution, on the assumption of a Gaussian noise model, leads to a closed-form solution. This was a consequence of the quadratic dependence of the log likelihood function on the parameter vector $\mathbf{w}$. For logistic regression, there is no longer a closed-form solution, due to the nonlinearity of the logistic sigmoid function. However, the departure from a quadratic form is not substantial. To be precise, the error function is concave, as we shall see shortly, and hence has a unique minimum. Furthermore, the error function can be minimized by an efficient iterative technique based on the *Newton-Raphson* iterative optimization scheme, which uses a local quadratic approximation to the log likelihood function. The Newton-Raphson update, for minimizing a function $E(\mathbf{w})$, takes the form (Fletcher, 1987; Bishop and Nabney, 2008)

$$\mathbf{w}^{(\mathrm{new})} = \mathbf{w}^{(\mathrm{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w}). \tag{4.92}$$

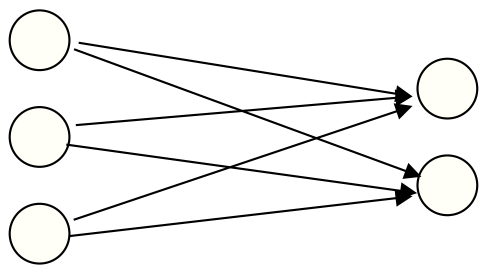
where $\mathbf{H}$ is the Hessian matrix whose elements comprise the second derivatives of $E(\mathbf{w})$ with respect to the components of $\mathbf{w}$.

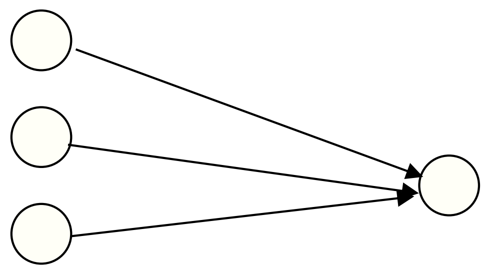
Let us first of all apply the Newton-Raphson method to the linear regression model (3.3) with the sum-of-squares error function (3.12). The gradient and Hessian of this error function are given by

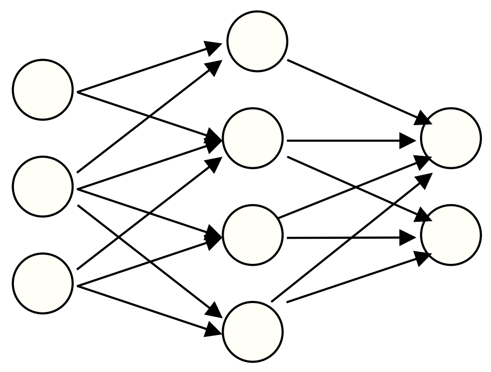
$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}_n - t_n)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\mathbf{w} - \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t} \tag{4.93}$$

$$\mathbf{H} = \nabla\nabla E(\mathbf{w}) = \sum_{n=1}^{N} \boldsymbol{\phi}_n\boldsymbol{\phi}_n^{\mathrm{T}} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} \tag{4.94}$$

Lecture 8: Backpropagation





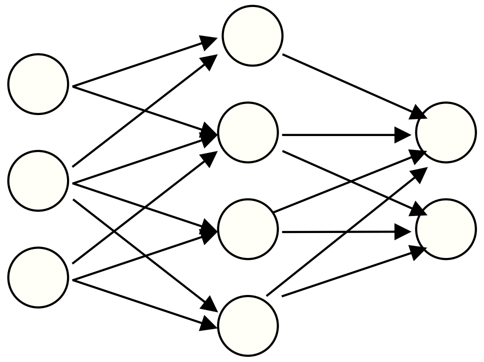


Number of parameters

- $\boldsymbol{t} = \boldsymbol{w}^T\boldsymbol{x}$,N measurement, M parameters
  - How large a **w** can we determine?

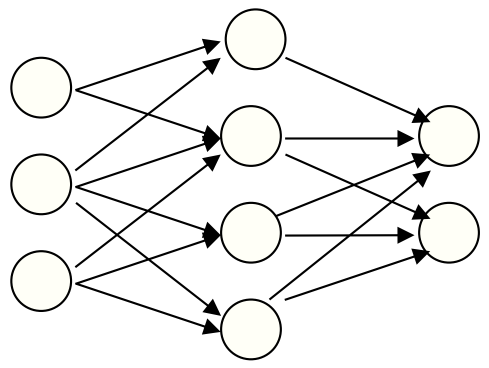
- $\boldsymbol{t} = \varphi(\boldsymbol{w}, \boldsymbol{x})$
  - How large a **w** can we determine?

- Consider a neural network, with one hidden layer, each layer having N=M=100 nodes
  - How large is **W**?
  - How many observations is needed to estimate **W**?



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, giving a universal approximator. But how to 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.



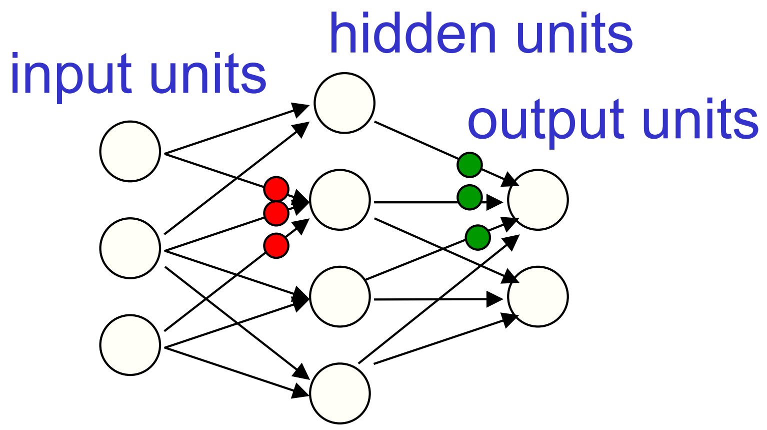
Learning by perturbing weights

Randomly perturb one weight. If it improves performance save the change.

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

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

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

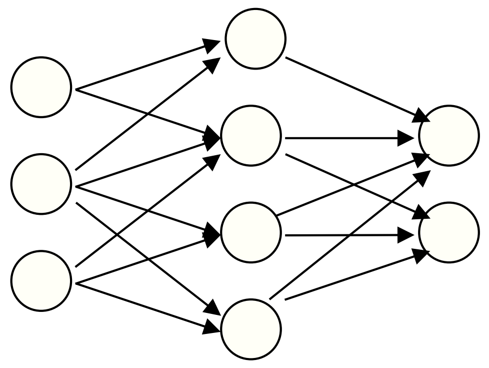


Learning the hidden to output weights is easy. Learning the input to hidden weights is hard.

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 have many separate effects on the error.
- Error derivatives for all the hidden units is computed 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.



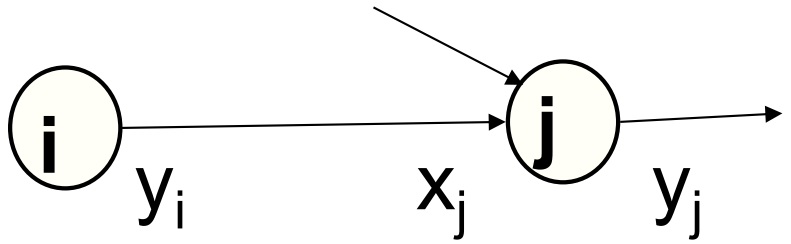
A difference in notation

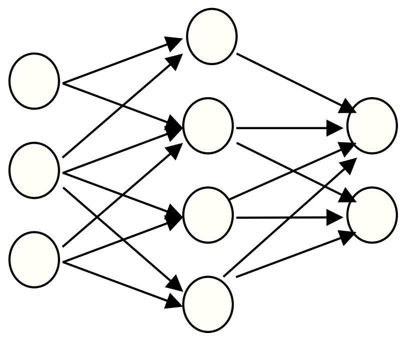
- For networks with multiple hidden layers Bishop uses an explicit extra index to denote the layer.
- The lecture notes use a simpler notation in which the index denotes the layer implicitly.

  y is 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

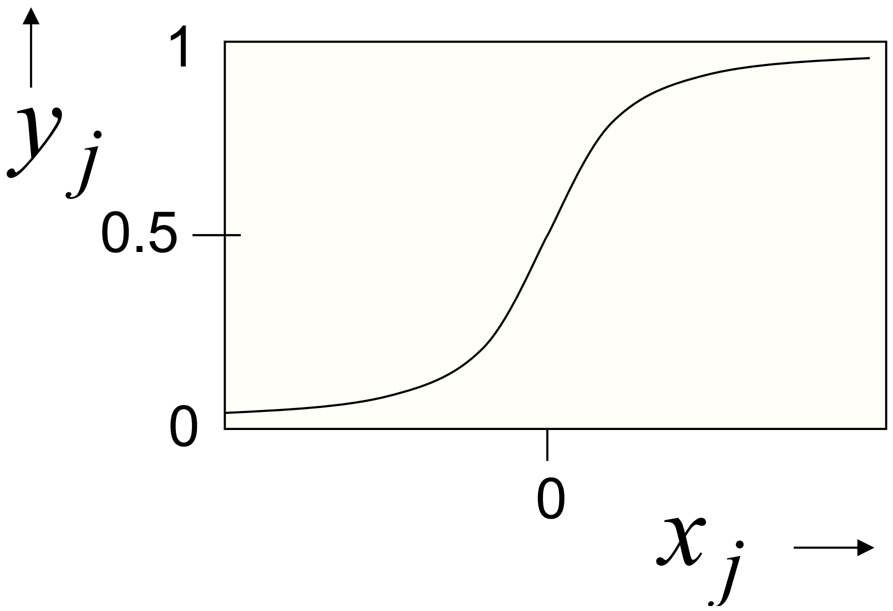
- For backpropagation, we need neurons that have well-behaved derivatives.
  - Typically they use the logistic function
  - The output is a smooth function of inputs and weights.

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

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

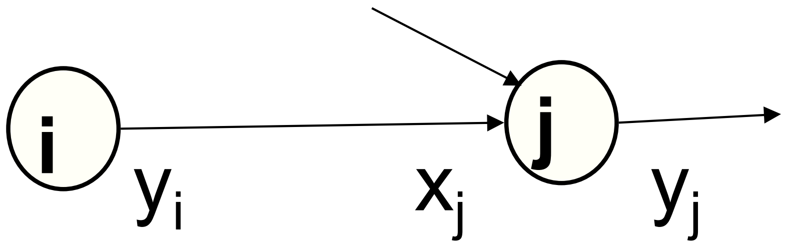
$$\frac{\partial x_j}{\partial w_{ij}} = y_i \qquad \frac{\partial x_j}{\partial y_i} = w_{ij}$$

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



Backpropagation

- J nodes
- Observations t_j
- Predictions y_j
- Energy function $E =$
- $\frac{dE}{dy_j} =$
- $\frac{dE}{dx_j} =$
- $\frac{dE}{dw_{ij}} =$
- $\frac{dE}{dy_i} = \sum_j^J$
- $\frac{dE}{dx_i} =$



NOT

USED

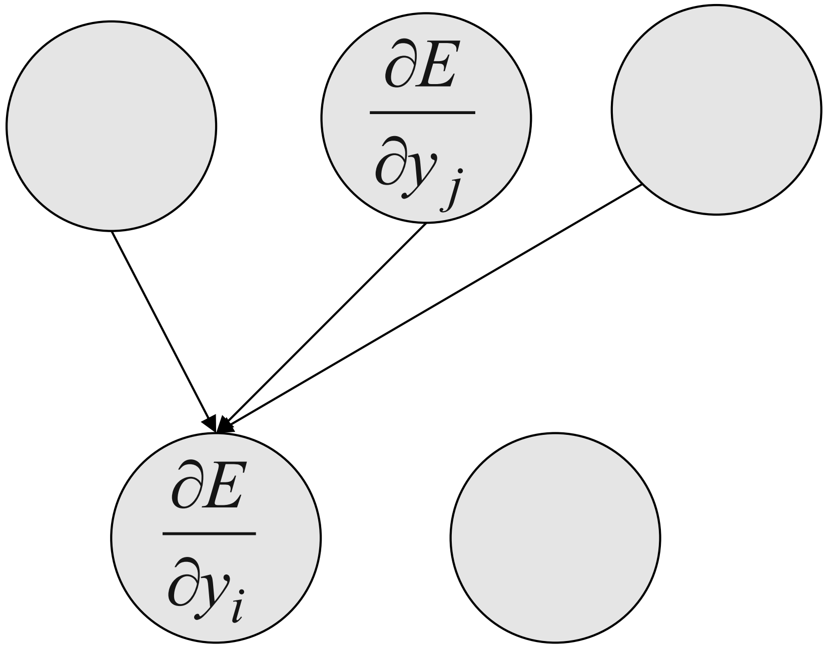
YET

Sketch of backpropagation on a single training case

1. Convert the discrepancy between each output and its target value into an error derivative.
2. Compute error derivatives in each hidden layer from error derivatives in the layer above.
3. Use error derivatives w.r.t. activities to get error derivatives w.r.t. the weights.

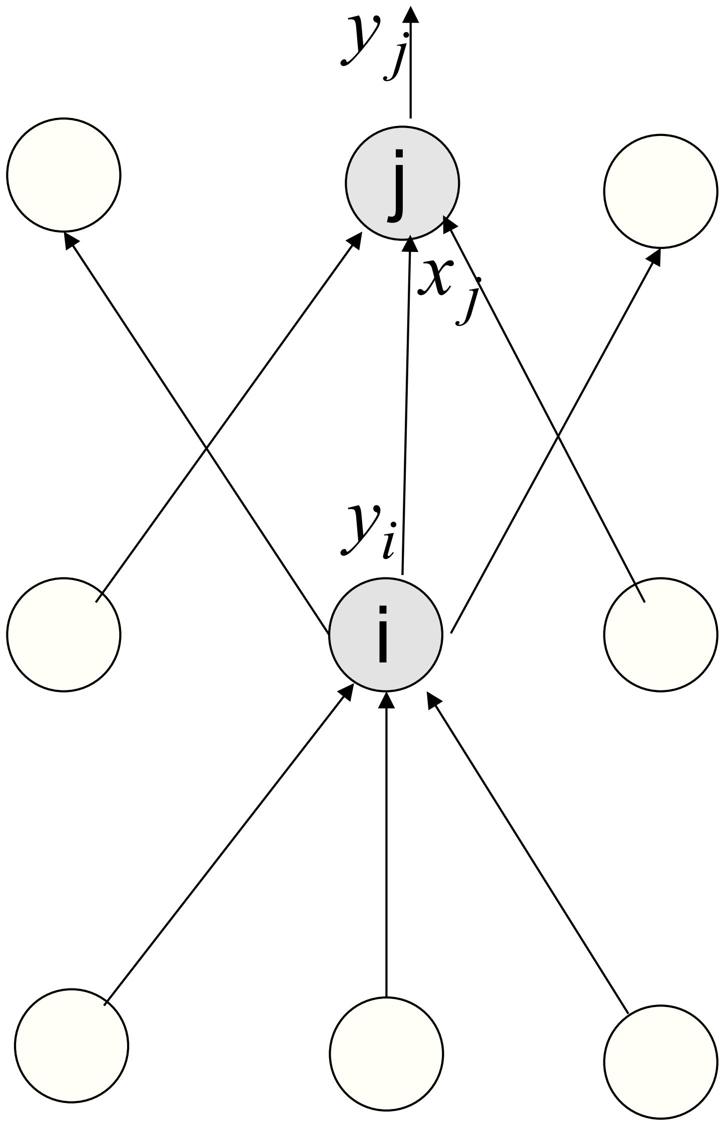
$$E = \sum_j \tfrac{1}{2}(y_j - d_j)^2$$

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



$\frac{\partial E}{\partial y_j}$
$\frac{\partial E}{\partial y_i}$

The derivatives



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

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

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