Learning from Data: Layered Neural Networks 2

Amos Storkey, School of Informatics

November 9, 2005

http://www.anc.ed.ac.uk/∼amos/lfd/

Layered Neural Networks

- Error model
- Calculating the derivative: the chain rule
- Optimisation

Error Function: Real Case

- Remember there is a correspondence between the error function and the log likelihood up to an additive and multiplicative constant.
- In the real case the output neurons are usually linear.
- The neural network is a deterministic function.
- We presume the output of of the neural network is subject to Gaussian measurement error.
- Remember the Gaussian likelihood produces the sum squared error function.

Sum squared error function

- Remember
- y_j is the desired output of unit j
- f_j is the actual output of unit j

$$E = \sum_j (y_j - f_j)^2$$

Error Function: Binary Class Case

- In the binary class case the output neurons are usually sigmoid.
- The output is interpreted as the probability of class 1.
- Then the logistic likelihood produces the *cross-entropy error.*

$$E = -\sum_j [y_j \log f_j + (1 - y_j) \log(1 - f_j)]$$

Error function: multinomial case

- In the multinomial case (many classes) there is an output neuron per class and the output neurons are usually linear.
- The final class y is interpreted from the outputs f_i using a *softmax* or *logit* model.

$$P(y=c)=\frac{\exp(f_c)}{\sum_i \exp(f_i)}$$

.

- Here the number of output neurons matches the number of classes.

Multinomial error function

- The multinomial error function is therefore

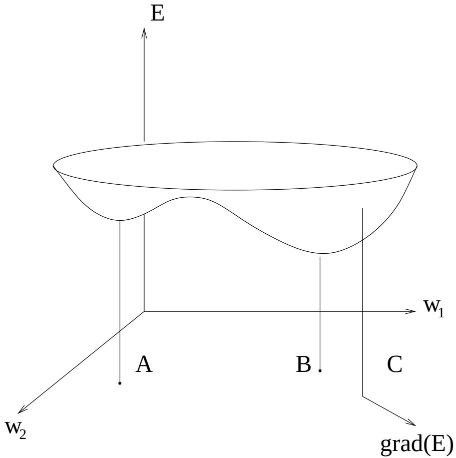
$$-[\log f_y - \log \sum_i \exp(f_i)]$$

- Again subscripts denote the neuron number.

Form of error functions

- The error surface is continuous and differentiable.
- The error surface may have local minima (unlike logistic regression).
- The error surface is generally high dimensional.
- There are many symmetries to the error surface (for a start all the hidden layer neurons are exchangeable).

Error function



- A is a *local* minimum
- B is the *global* minimum
- C is not a minimum, grad(E) $\neq 0$

Learning in a multi-layer network

- Presume a sum squared error function.
- Present an input pattern $\mathbf{x}$ and observe outputs $\mathbf{y}$ of the output nodes. Let $\boldsymbol{\theta}$ denote the vector of parameters of the network.
- $\mathbf{y}$ is the desired output, $\mathbf{f}$ the actual output. Adjust weights to minimise

$$E = \sum_{\mu} (\mathbf{y}^{\mu} - \mathbf{f}^{\mu})^2$$

  where μ labels the particular training item.
- Calculate $\frac{\partial E}{\partial \boldsymbol{\theta}}$ and carry out minimisation.

Regularisation

- Remember regularisation is the approach used to incorporate a prior over weights into the error function.
- This can help prevent overfitting.
- Standard regulariser is $\lambda\boldsymbol{\theta}^T\boldsymbol{\theta}$.
- Add this on to the error function.

The Full Error Function

- We write the MLP with one hidden layer as

$$f(\mathbf{x}, \boldsymbol{\theta}) = r\left(\sum_{i=1}^{K} v_i g(\mathbf{w}_i^T \mathbf{x} + b_i) + b\right)$$

- The full error function in the regression case is

$$E(\boldsymbol{\theta}) = \sum_{\mu=1}^{N} (f(\mathbf{x}^\mu, \boldsymbol{\theta}) - y^\mu)^2 + \lambda \boldsymbol{\theta}^T \boldsymbol{\theta}$$

Calculating Derivatives

- We can calculate the derivatives...

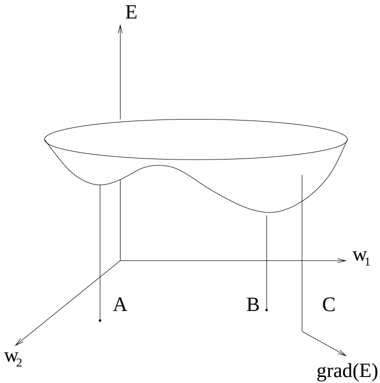
$$\frac{\partial E}{\partial \theta_i} = 2\sum_{\mu=1}^{N}(f(\mathbf{x}^{\mu}, \boldsymbol{\theta}) - y^{\mu})\frac{\partial f(\mathbf{x}^{\mu}, \boldsymbol{\theta})}{\partial \theta_i} + 2\lambda\theta_i$$

- But to do this we need to calculate $\frac{\partial f(\mathbf{x}^{\mu}, \boldsymbol{\theta})}{\partial \theta_i}$.
- This involves the use of the chain rule.
- The use of the chain rule in neural networks has become known as *back-propagation*.

Optimisation

- Gradient descent
- Line search
- Problems with gradient descent
- Second-order information
- Conjugate gradients
- Batch vs online

Optimisation



- Use methods that "go downhill" on the error surface to find a *local* minimum, e.g.
  - gradient descent
  - conjugate gradient

Gradient Descent

- Remember the gradient descent (or ascent) procedure from the lecture on logistic regression.
- Can do the same here.

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$$

- For small η

$$E(\boldsymbol{\theta}^{new}) \simeq E(\boldsymbol{\theta}) - \eta (\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}))^2$$

- Locally, we are modelling the function as a plane.

Gradient Descent Algorithm

Set $\boldsymbol{\theta} = (\mathbf{w}_1^T, \mathbf{w}_2^T, \ldots, \mathbf{w}_K^T, b_1, b_2, \ldots, b_K, v_1, v_2, \ldots, v_K, b)$

Initialise $\boldsymbol{\theta}$

while $E(\boldsymbol{\theta})$ is still changing substantially

$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$

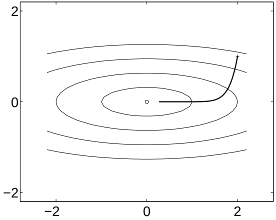
end while

return $\boldsymbol{\theta}$

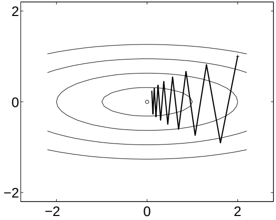
Choosing η

- Too small
  - too slow
- Too big
  - unstable – goes outside region where linear approximation is valid.

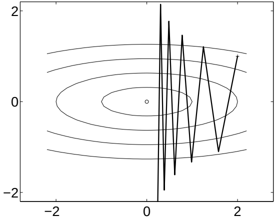
Example



$\eta = 0.01$



$\eta = 0.0952$



$\eta = 0.105$

Summary

- Error functions for various standard problems.
- the full MLP error.
- Calculating the derivatives.
- Gradient ascent.