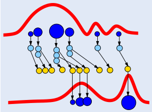


# Lecture 10

## Nonlinear Supervised Learning

Neural networks and optimization

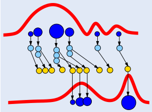
*Machine Learning and Data Mining*
November 2009

Nando de Freitas
UBC

## Gradient

- Searching for a good solution can be interpreted as looking for a minimum of some error (loss) function in parameter space.

⋆



## Gradient

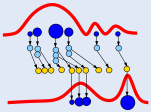
- Searching for a good solution can be interpreted as looking for a minimum of some error (loss) function in parameter space.

⋆

- The gradient is the vector of derivatives:

$$\nabla E(\boldsymbol{\theta}_{1:d}) = \begin{pmatrix} \frac{dE}{d\theta_1} & \cdots & \frac{dE}{d\theta_d} \end{pmatrix}$$

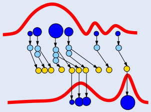
The gradient vector is orthogonal to the contours. Hence, to minimise the error, we follow the gradient (the direction of steepest descent in error).



## Gradient for linear model

- Let's go back to the linear model $\mathbf{Y} = \mathbf{X}\theta$ with quadratic error function $E = (\mathbf{Y} - \mathbf{X}\theta)^T(\mathbf{Y} - \mathbf{X}\theta)$. The gradient for this model is:

⋆



## Gradient for linear model

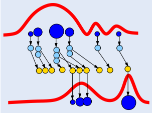
- Let's go back to the linear model $\mathbf{Y} = \mathbf{X}\theta$ with quadratic error function $E = (\mathbf{Y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\theta})$. The gradient for this model is:

⋆

- The gradient descent learning rule, at iteration $t$, is:

$$\begin{aligned}\boldsymbol{\theta}^{(t)} &= \boldsymbol{\theta}^{(t-1)} + \alpha\nabla E \\ &= \boldsymbol{\theta}^{(t-1)} + \alpha\mathbf{X}^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\theta}^{(t-1)})\end{aligned}$$

where $\alpha$ is a user-specified learning rate.

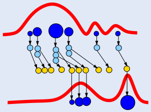


## Online learning

- In some situations, we might want to learn the parameters by going over the data **online**:

$$\theta^{(t)} = \theta^{(t-1)} + \alpha x^{(t)}(y^{(t)} - x^{(t)}\theta^{(t-1)})$$

- This is the **least mean squares** algorithm. This learning rule is a **stochastic approximation** technique also known as the **Robbins-Monro** procedure. It's stochastic because the data is assumed to come from a stochastic process.
- If $\alpha$ decreases with rate $1/n$, one can show that this algorithm converges. If the $\theta$ vary "slowly" with time, it is also possible to obtain convergence proofs with $\alpha$ set to a small constant. There are many tricks, including averaging, momentum and minibatches, to accelerate convergence.



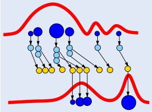
## Hessian of linear model

The **Newton-Raphson** algorithm uses the gradient learning rule, with the inverse Hessian matrix in place of $\alpha$:

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} + \mathbf{H}^{-1} \nabla E$$

$$H = \frac{\partial^2 E}{\partial \boldsymbol{\theta}^2}$$

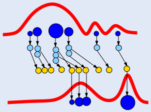
⋆



## Very fast convergence!

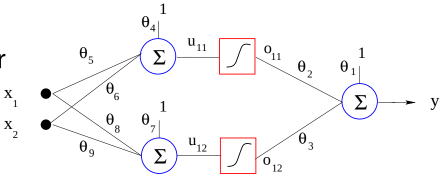
★

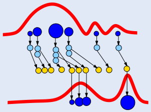
Note that $\alpha$ is a scalar, while $\mathbf{H}$ is a large matrix. So there is a trade-off between speed of convergence and storage.



## Multi-layer perceptrons

- Gradient descent techniques allow us to learn complex, nonlinear supervised "neural networks" known as **multi-layer percetrons**.





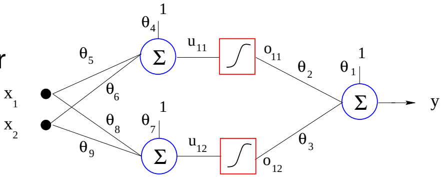
## Multi-layer perceptrons

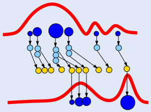
- Gradient descent techniques allow us to learn complex, nonlinear supervised "neural networks" known as **multi-layer percetrons.**
- Mathematically, an MLP is a nonlinear function approximator:

$$\widehat{\mathbf{y}} = \phi_j \left(\phi_i \left(\mathbf{X}\boldsymbol{\theta}_j\right) \boldsymbol{\theta}_i\right)$$

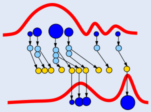
where $\phi(\cdot)$ is the **sigmoidal (logistic) function**:

$$\phi_i \left(\mathbf{X}\boldsymbol{\theta}_j\right) = \frac{1}{1 + e^{-\mathbf{X}\boldsymbol{\theta}_j}}$$





## Multi-layer perceptrons

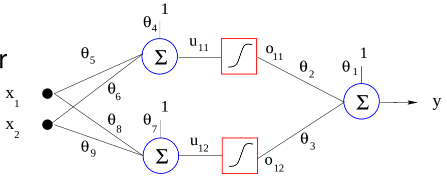


- Gradient descent techniques allow us to learn complex, nonlinear supervised "neural networks" known as **multi-layer percetrons.**
- Mathematically, an MLP is a nonlinear function approximator:

$$\widehat{\mathbf{y}} = \phi_j\left(\phi_i\left(\mathbf{X}\boldsymbol{\theta}_j\right)\boldsymbol{\theta}_i\right)$$

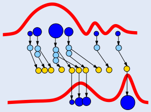
where $\phi(\cdot)$ is the **sigmoidal (logistic) function**:

$$\phi_i\left(\mathbf{X}\boldsymbol{\theta}_j\right) = \frac{1}{1+e^{-\mathbf{X}\boldsymbol{\theta}_j}}$$



★

## Loss functions for regression

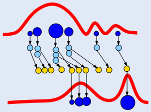


Assume we are given the data $\{\mathbf{X}_{1:n}, \mathbf{Y}_{1:n}\}$ and want to come up with a nonlinear mapping $\widehat{\mathbf{Y}} = \mathbf{f}(\mathbf{X}, \theta)$, where $\theta$ is obtained my minimising a loss function: quadratic $E = (\mathbf{Y} - \mathbf{f}(\mathbf{X}, \theta))^T(\mathbf{Y} - \mathbf{f}(\mathbf{X}, \theta))$ when doing regression. What is the likelihood model?

⋆

# Loss functions for classification

⋆



## Backpropagation

The **synaptic weights** $\theta$ can be learned by following gradients:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \alpha(\mathbf{Y} - \widehat{\mathbf{Y}})\frac{\partial \widehat{\mathbf{Y}}}{\partial \boldsymbol{\theta}^{(t)}}$$

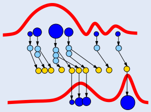
where $\widehat{\mathbf{Y}} = \mathbf{f}(\mathbf{X}, \theta^{(t)})$. The **output layer** mapping for our example is given by:

$$\widehat{y} = \theta_1 + \theta_2 o_{11} + \theta_3 o_{12}$$

and consequently, the derivatives with respect to the weights are given by:

⋆

$$\frac{\partial \widehat{\mathbf{y}}}{\partial \theta_1} =$$
$$\frac{\partial \widehat{\mathbf{y}}}{\partial \theta_2} =$$
$$\frac{\partial \widehat{\mathbf{y}}}{\partial \theta_3} =$$



## Backpropagation

The **hidden layer** mapping for the top neuron is:

$$o_{11} = \frac{1}{1+\exp(-u_{11})} \qquad \text{where } u_{11} = \theta_4 + \theta_5 x_1 + \theta_6 x_2$$

Note that

$$\frac{\partial o_{11}}{\partial u_{11}} = o_{11}(1 - o_{11})$$

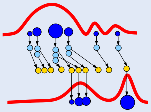
⋆

The derivatives with respect to the weights are:

$$\frac{\partial \widehat{y}}{\partial \theta_4} = \frac{\partial \widehat{y}}{\partial o_{11}} \frac{\partial o_{11}}{\partial u_{11}} \frac{\partial u_{11}}{\partial \theta_4} =$$

$$\frac{\partial \widehat{y}}{\partial \theta_5} =$$

$$\frac{\partial \widehat{y}}{\partial \theta_6} =$$



## Backpropagation

The derivatives with respect to the weights of the other hidden layer neuron can be calculated following the same procedure. Once we have all the derivatives, we can use either steepest descent or Newton-Raphson to update the weights.