

Lecture 10

Nonlinear Supervised Learning

Neural networks and optimization

Machine Learning and Data Mining
November 2009

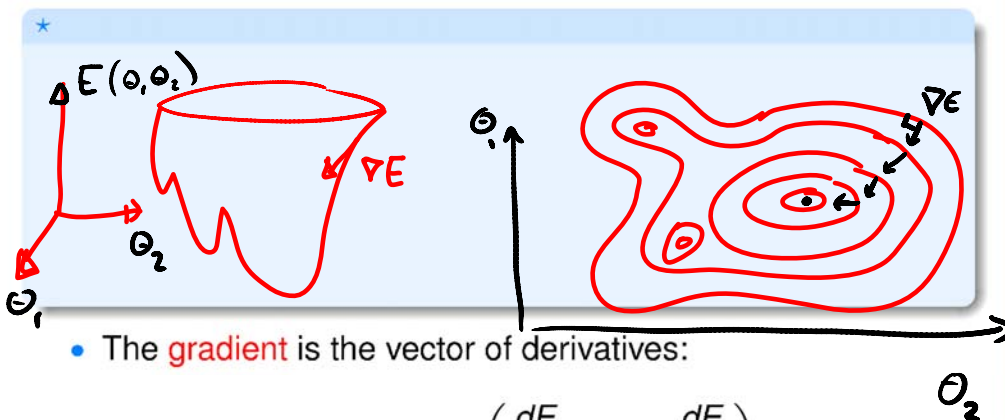
Nando de Freitas
UBC



1

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:

$$\nabla E(\theta_{1:d}) = \left(\frac{dE}{d\theta_1} \quad \dots \quad \frac{dE}{d\theta_d} \right)$$

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



Optimization
Gradient descent
Online learning
Newton's method
Neural Networks

2

Gradient for linear model

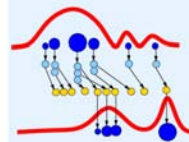
- Let's go back to the linear model $\mathbf{Y} = \mathbf{X}\theta$ with quadratic error function $E = (\mathbf{Y} - \mathbf{X}\theta)^T(\mathbf{Y} - \mathbf{X}\theta)$. The gradient for this model is:

$$\begin{aligned}\nabla E(\theta) &= \frac{\partial}{\partial \theta} E(\theta) \\ &= \frac{\partial}{\partial \theta} (\mathbf{Y}^T \mathbf{Y} - 2 \mathbf{Y}^T \mathbf{X} \theta + \theta^T \mathbf{X}^T \mathbf{X} \theta) \\ &= -2 \mathbf{X}^T \mathbf{Y} + 2 \mathbf{X}^T \mathbf{X} \theta\end{aligned}$$

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

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

where α is a user-specified learning rate.

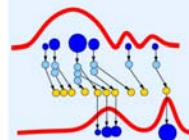


Online learning

- 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 α decreases with rate $1/n$, one can show that this algorithm converges. If the θ vary "slowly" with time, it is also possible to obtain convergence proofs with α set to a small constant. There are many tricks, including averaging, momentum and minibatches, to accelerate convergence.



Hessian of linear model

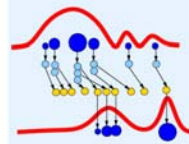
The **Newton-Raphson** algorithm uses the gradient learning rule, with the inverse Hessian matrix in place of α :

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

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

$$H = \frac{\partial}{\partial \theta} (-2X^T y + 2X^T X \theta)$$
$$= 2X^T X$$

$$H^{-1} = \frac{1}{2} (X^T X)^{-1}$$



Very fast convergence!

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \nabla E(\theta^{(t)})$$

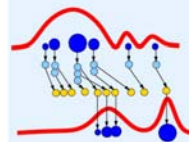
with $\alpha = -H^{-1}$

$$\theta^{(t+1)} = \theta^{(t)} + \frac{1}{2} (X^T X)^{-1} (-2X^T y + X^T X \theta^{(t)})$$

$$= (X^T X)^{-1} X^T y = \hat{\theta}_{ML}$$

in 1 step!

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



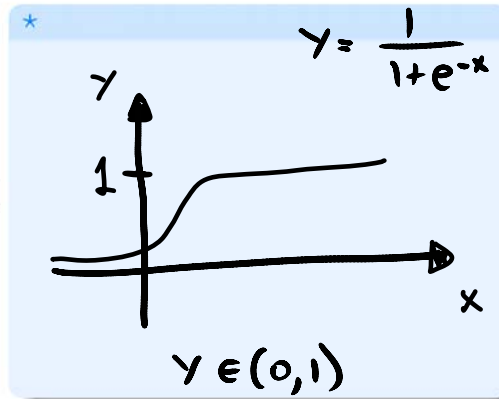
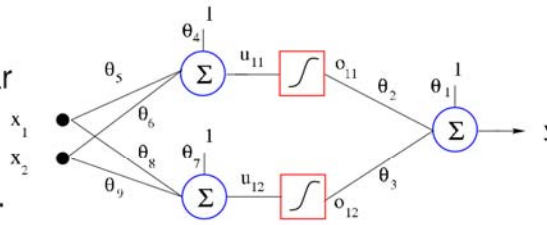
Multi-layer perceptrons

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

$$\hat{y} = \phi_j(\phi_i(\mathbf{X}\theta_j)\theta_i)$$

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

$$\phi_i(\mathbf{X}\theta_j) = \frac{1}{1 + e^{-\mathbf{X}\theta_j}}$$



Nonlinear Supervised Learning
Nando de Freitas

Optimization
Neural Networks
MLPs
Regression
Classification
Backpropagation

7

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 $\hat{\mathbf{Y}} = \mathbf{f}(\mathbf{X}, \theta)$, where θ is obtained by minimizing 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?

* $\mathbf{y} \in \mathbb{R}^c$

$$P(\mathbf{y} | \mathbf{x}, \theta) = (2\pi\sigma^2)^{-\frac{n}{2}} e^{-\frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{f}(\mathbf{x}, \theta))^T (\mathbf{y} - \mathbf{f}(\mathbf{x}, \theta))}$$

For linear models: $\mathbf{f}(\mathbf{x}, \theta) = \mathbf{x}\theta$
and we recover what we had before.

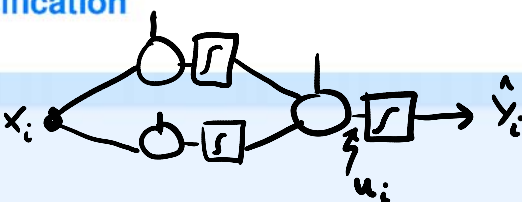
Nonlinear Supervised Learning
Nando de Freitas

Optimization
Neural Networks
MLPs
Regression
Classification
Backpropagation

8

Loss functions for classification

* $y_i \in \{0, 1\}$



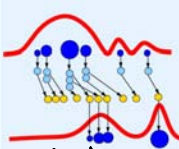
$P(y_i | x_i, \theta) = \underbrace{\left[\frac{1}{1 + e^{-u(x_i; \theta)}} \right]}_{NN(x_i; \theta)}^{\mathbb{I}_1(y_i)} \left[1 - \frac{1}{1 + e^{-u(x_i; \theta)}} \right]^{\mathbb{I}_0(y_i)}$

i.e. Use Bernoulli likelihood and one sigmoidal output neuron. The error function is:

$$E(\theta) = -\mathbb{I}_1(y_i) \log NN(x_i; \theta) - \mathbb{I}_0(y_i) \log [1 - NN(x_i; \theta)]$$

For more outputs, use softmax encoding.

Nonlinear Supervised Learning
Nando de Freitas



Neural Networks
MLPs
Regression
Classification
Backpropagation

9

Backpropagation

The **synaptic weights** θ can be learned by following gradients:

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

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

$$\hat{y} = \theta_1 + \theta_2 o_{11} + \theta_3 o_{12} = \mathbf{f}(x, \theta)$$

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

*
$$\frac{\partial \hat{y}}{\partial \theta_1} = 1$$

$$\frac{\partial \hat{y}}{\partial \theta_2} = o_{11}$$

$$\frac{\partial \hat{y}}{\partial \theta_3} = o_{12}$$

Nonlinear Supervised Learning
Nando de Freitas



Optimization
Neural Networks
MLPs
Regression
Classification
Backpropagation

10

Backpropagation

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

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

Note that

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

★

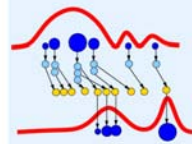
The derivatives with respect to the weights are:

$$\begin{aligned} \frac{\partial f(x, \theta)}{\partial \theta_4} &= \frac{\partial \hat{y}}{\partial \theta_4} = \frac{\partial \hat{y}}{\partial o_{11}} \frac{\partial o_{11}}{\partial u_{11}} \frac{\partial u_{11}}{\partial \theta_4} = \theta_2 o_{11} (1 - o_{11}) 1 \\ \frac{\partial \hat{y}}{\partial \theta_5} &= \frac{\partial \hat{y}}{\partial o_{11}} \frac{\partial o_{11}}{\partial u_{11}} \frac{\partial u_{11}}{\partial \theta_5} = \theta_2 o_{11} (1 - o_{11}) x_1 \\ \frac{\partial \hat{y}}{\partial \theta_6} &= \frac{\partial \hat{y}}{\partial o_{11}} \frac{\partial o_{11}}{\partial u_{11}} \frac{\partial u_{11}}{\partial \theta_6} = \theta_2 o_{11} (1 - o_{11}) x_2 \end{aligned}$$

11

Nonlinear Supervised Learning

Nando de Freitas



Optimization

Neural Networks

MLPs

Regression

Classification

Backpropagation

Backpropagation

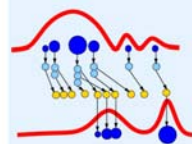
The derivatives with respect to the weights of the other hidden layer neuron can be calculated following the same procedure. Once we have all the derivatives, we can use either steepest descent or Newton-Raphson to update the weights.

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_4 \end{bmatrix}^{(t+1)} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_4 \end{bmatrix}^{(t)} + \alpha (\gamma - f(x, \theta^{(t)})) \begin{bmatrix} \partial f / \partial \theta_1 \\ \partial f / \partial \theta_2 \\ \vdots \\ \partial f / \partial \theta_4 \end{bmatrix}^{(t)}$$

12

Nonlinear Supervised Learning

Nando de Freitas



Optimization

Neural Networks

MLPs

Regression

Classification

Backpropagation