University of Rome "La Sapienza"

Dep. of Computer, Control and Management Engineering A. Ruberti

Introduction to Deep Learning

Valsamis Ntouskos

ALCOR Lab

Overview

- Linear Classification
- Logistic Regression
- Linear Regression
- Deep Feedforward Networks
- Training DFNs
- Activation Function
- Regularization

Linear Models for Classification

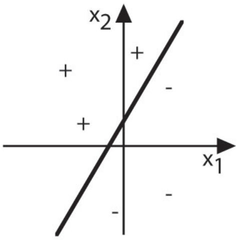
Learning a function $f : X \to Y$, with ...

- $X \subseteq \Re^n$
- $Y = \{C_1, \ldots, C_k\}$

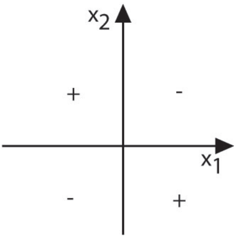
assuming *linearly separable* data.

Linearly separable data

Instances in a data set are *linearly separable* iff it exists a hyperplane that divide the instance space into two regions such that differently classified instances are separated.



(a)



(b)

Discriminant functions

Linear discriminant function

$$y : X \to \{C_1, \ldots, C_K\}$$

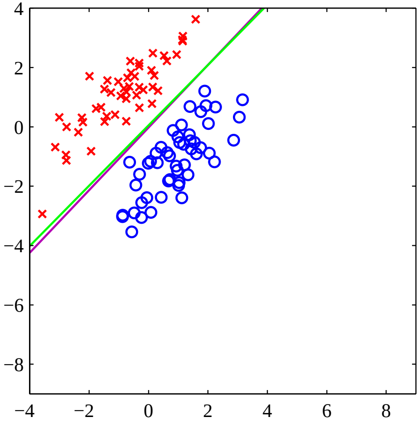
Two classes:

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

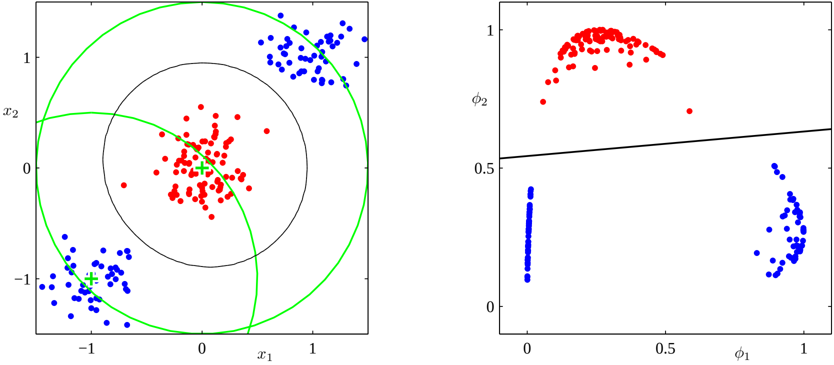
Multi classes:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

Linear Classification



Basis functions



Logistic Regression

Consider first the case of two classes.

Find the conditional probability:

$$P(C_1|\mathbf{x}) = \frac{P(\mathbf{x}|C_1)P(C_1)}{P(\mathbf{x}|C_1)p(C_1) + P(\mathbf{x}|C_2)p(C_2)}$$
$$= \frac{1}{1 + \exp(-\alpha)} = \sigma(\alpha).$$

with:

$$\alpha = \ln \frac{P(\mathbf{x}|C_1)P(C_1)}{P(\mathbf{x}|C_2)P(C_2)}$$

and

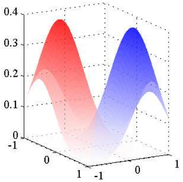
$\sigma(\alpha) = \frac{1}{1+\exp(-\alpha)}$ the *sigmoid function*.

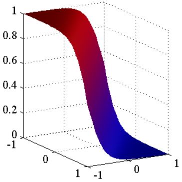
Logistic Regression

Assume $P(\mathbf{x}|C_i) \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma})$ - same covariance matrix

we get:

$$P(C_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0),$$





Multiclass logistic regression

$$p(C_k|\phi) = y_k(\phi) = \underbrace{\frac{exp(a_k)}{\sum_j exp(a_j)}}_{softmax}, \text{with } a_k = \mathbf{w}_k^T\phi$$

Linear Regression

Goal: Estimate the value t of a continuous function at $\mathbf{x}$ based on a dataset $\mathcal{D}$ composed of N observations $\{\mathbf{x}_n\}$, where $n = 1, \dots, N$, together with the corresponding target values $\{t_n\}$.

Ideally:

$$t = y(\mathbf{x}, \mathbf{w})$$

Linear Regression - Model

Linear Basis Function Models

Simplest case:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D = \mathbf{w}^T \mathbf{x}$$

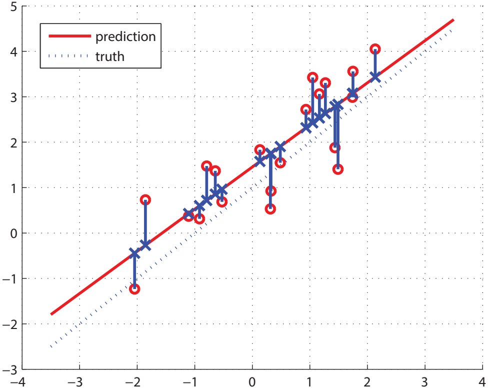
$$\text{with } \mathbf{x} = \begin{bmatrix} 1 \\ \vdots \\ x_D \end{bmatrix} \text{ and } \mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_D \end{bmatrix}$$

Linear both in model parameters **w** and variables **x**.

- Too limiting!

Example - Line fitting

$$y = w_1 x_1 + w_0$$



Linear Regression - Model

Linear Basis Function Models

Using nonlinear functions of input variables:

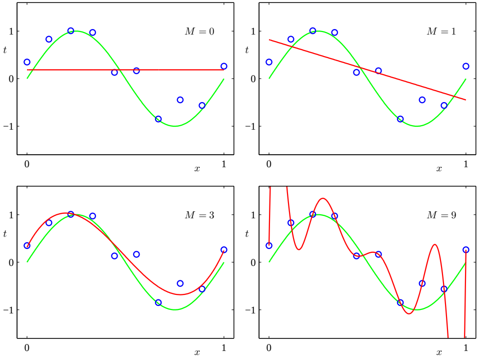
$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi},$$

$$\text{with } \phi_0(\mathbf{x}) = 1 \text{ and } \boldsymbol{\phi} = \begin{bmatrix} \phi_0 \\ \vdots \\ \phi_{M-1} \end{bmatrix}$$

- Still linear in the parameters $\mathbf{w}$!

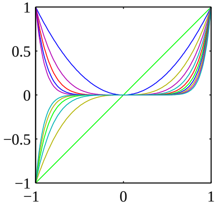
Example - Polynomial curve fitting

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

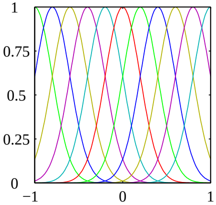


Linear Regression - Model

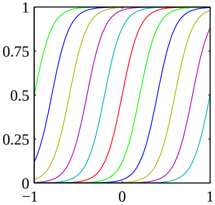
Examples of basis functions



Polynomial



Radial



Sigmoid / Tanh

Deep Feedforward Networks

Alternative names:

- Feedforward Neural Networks
- (Artificial) Neural Networks - (A)NNs
- Multilayer Perceptrons - MLPs

Represent a parametric function

Suitable for tasks described as associating a vector to another vector

Deep Feedforward Networks

Goal: Estimate some function f^*

Examples:

Classification $y = f^*(\mathbf{x})$ with $x \in \mathcal{X}$ and $y \in \{c_1, \ldots, c_K\}$

Regression $y = f^*(\mathbf{x})$ with $x \in \mathcal{X}$ and $y \in \mathbb{R}$

Density estimation $y = f^*(\mathbf{x})$ with $x \in \mathcal{X}$ and $\int_{\mathcal{X}} y = 1$

Framework: Define $y = f(\mathbf{x}, \boldsymbol{\theta})$ and learn parameters $\boldsymbol{\theta}$

Deep Feedforward Networks

Data: target values t_n corresponding to given input variable values $\mathbf{x}_n$ such that $t_n \approx f^*(\mathbf{x}_n)$
We use $\approx$ as the data may be affected by noise.

Objective:
Learn $\boldsymbol{\theta}$ such that $f(\mathbf{x}, \boldsymbol{\theta})$ approximates as much as possible f^*.
Training based on a suitable cost (loss) function

Note: Dataset contains no target values about hidden units!

DFN - Terminology

Feedforward information flows from input to output without any loops
Networks f is a composition of elementary functions in an acyclic graph

Example:

$$f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}, \boldsymbol{\theta}^{(1)}), \boldsymbol{\theta}^{(2)}), \boldsymbol{\theta}^{(3)})$$

where:

$f^{(m)}$ the m-th layer of the network

and

$\boldsymbol{\theta}^{(m)}$ the corresponding parameters

DFN - Terminology

DFNs are **chain structures**

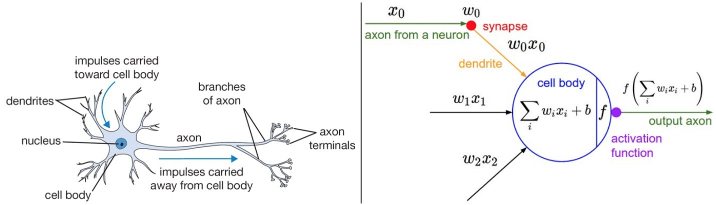
The length of the chain is the **depth** of the network

Final layer also called **output layer**

Name **deep learning** follows from the use of networks with a large number of layers (large depth)

Deep Feedforward Networks

Draw inspiration from brain structures



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Image from Isaac Changhau `https://isaacchanghau.github.io`

Hidden layer output can be seen as an array of **unit** (neuron) activations based on the connections with the previous units

Note: Only use some insights, they are not a model of the brain function!

Deep Feedforward Networks

Why DFNs?

Linear models cannot model interaction between input variables

Kernel methods require the choice of suitable kernels

- use generic kernels e.g. RBF, polynomial, etc. (convex problem)
- use hand-crafted kernels - application specific (convex problem)

Deep leaning:
consider parametric mapping functions ϕ and learn their parameters (non-convex problem)

Model:

$$y = f(\mathbf{x}, \boldsymbol{\theta}, \mathbf{w}) = \phi(\mathbf{x}, \boldsymbol{\theta})^T \mathbf{w}$$

Gradient-based learning

Learning remarks

- Parameters found via gradient-based learning
- Unit saturation can hinder learning
- When units saturate gradient becomes very small
- Suitable cost function and unit nonlinearities help to avoid saturation

Cost function

Model implicitly defines a conditional distribution $p(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta})$

Cost function:
Typically choose the negative log-likelihood - Maximum likelihood principle

$$J(\boldsymbol{\theta}) = -\ln(p(\mathbf{t}|\mathbf{x}))$$

Example:
Assuming additive Gaussian noise we have

$$p(\mathbf{t}|\mathbf{x}) = \mathcal{N}(\mathbf{t}|f(\mathbf{x}, \boldsymbol{\theta}), \beta^{-1}I)$$

and hence

$$J(\boldsymbol{\theta}) = \frac{1}{2}(\mathbf{t} - f(\mathbf{x}, \boldsymbol{\theta}))^2$$

Gradient Computation

Information flows forward through the network when computing network output y from input $\mathbf{x}$

To train the network we need to compute the gradients with respect to the network parameters $\boldsymbol{\theta}$

The **back-propagation** or **backprop** algorithm is used to propagate gradient computation from the cost through the whole network

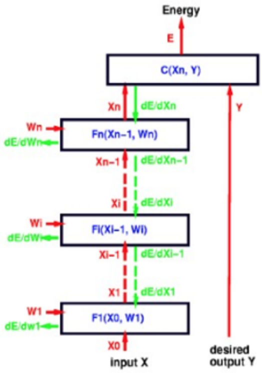


Image by Y. LeCun

Gradient Computation

Goal: Compute the gradient of the cost function w.r.t. the parameters

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Analytic computation of the gradient is straightforward

- simple application of the chain rule
- numerical evaluation can be expensive

Back-propagation is *simple* and *inexpensive*.

Remarks:

- back-propagation is **not** a training algorithm
- back-propagation is only used to compute the gradients
- back-propagation is **not** specific to DFNs

Learning algorithms

- Stochastic Gradient Descent (SGD)
- SGD with momentum
- Algorithms with adaptive learning rates

Stochastic Gradient Descent

Require: Learning rate η
Require: Initial values of $\boldsymbol{\theta}$
$k \leftarrow 1$
while stopping criterion not met **do**
Sample a subset (minibatch) $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ of m examples from the dataset $\mathcal{D}$
Compute gradient estimate: $\mathbf{g} = \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}, \boldsymbol{\theta}), \mathbf{t}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\mathbf{g}$
$k \leftarrow k + 1$
end while

Note: η might change according to some rule through the iterations

Output units activation functions

Network output units determine also the cost function.
Let $h = f(\mathbf{x}, \boldsymbol{\theta})$ the output of the hidden layers.

Regression
Linear units: Identity activation function - no nonlinearity

$$y = W^T \mathbf{h} + \mathbf{b}$$

Used to model a conditional Gaussian distribution

$$p(t|\mathbf{x}) = \mathcal{N}(t|y, \beta^{-1})$$

Maximum likelihood equivalent to minimizing mean squared error

Note: Linear units do not saturate!

Output units activation functions

Binary classification

Sigmoid units: Sigmoid activation function

$$y = \sigma(\mathbf{w}^T \mathbf{h} + b)$$

We have seen that the likelihood corresponds to a Bernoulli distribution

Hence:

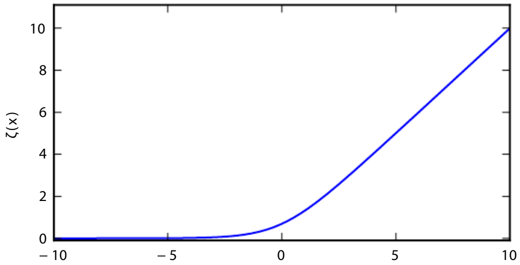
$$\begin{aligned} J(\boldsymbol{\theta}) &= -\ln P(t|\mathbf{x}) \\ &= -\ln \sigma(\alpha)^t (1 - \sigma(\alpha))^{1-t} \\ &= -\ln \sigma((2t - 1)\alpha) \\ &= \mathrm{softplus}((1 - 2t)\alpha), \end{aligned}$$

with $\alpha = \mathbf{w}^T \mathbf{h} + b$.

Note: Unit saturates only when it gives the correct answer.

If α has wrong sign $\mathrm{softplus}((1-2t)\alpha) \approx |\alpha|$ and $\frac{dy}{d\alpha} \approx \mathrm{sign}(\alpha)$.

Output units activation functions



The softplus function

Output units activation functions

Multiclass classification

Softmax units: Softmax activation function

$$y = \mathrm{softmax}(\alpha)_i = \frac{\exp(\alpha_i)}{\sum_j \alpha_j}$$

Likelihood corresponds to a Multinomial distribution

Hence:

$$J(\boldsymbol{\theta})_i = -\ln \mathrm{softmax}(\alpha)_i = \ln \sum_j \exp(\alpha_j) - \alpha_i$$

Note: $\ln \sum_j \exp(\alpha_j) \approx \ln \exp(\max_j(\alpha_j)) = \max_j \alpha_j$.

If α_i corresponds to the correct answer the derivative is small.

Misclassifications give large derivatives.

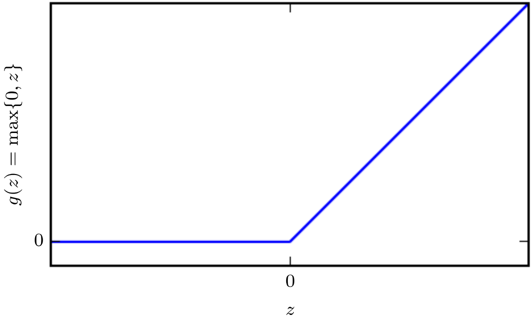
Hidden units activation functions

Rectified Linear Units:

$$g(\alpha) = \max\{0, \alpha\}.$$

- Easy to optimize - similar to linear units
- Not differentiable at 0 - does not cause problems in practice

Hidden unit activation functions



Hidden unit activation functions

Sigmoid and hyperbolic tangent:

$$g(\alpha) = \sigma(\alpha)$$

and

$$g(\alpha) = \tanh(\alpha)$$

Closely related as $\tanh(\alpha) = 2\sigma(2\alpha) - 1$.

Remarks:

- No logarithm at the output, the units saturate easily.
- Gradient based learning is very slow.
- Hyperbolic tangent gives larger gradients with respect to the sigmoid.

Activation functions overview

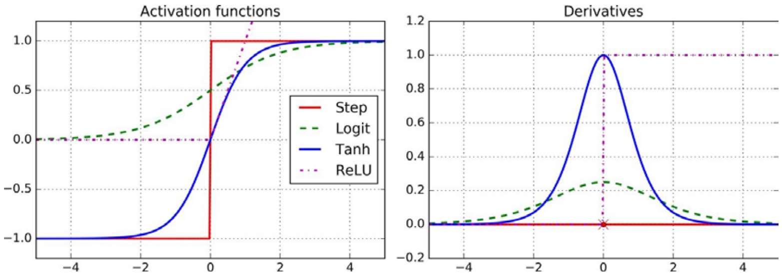
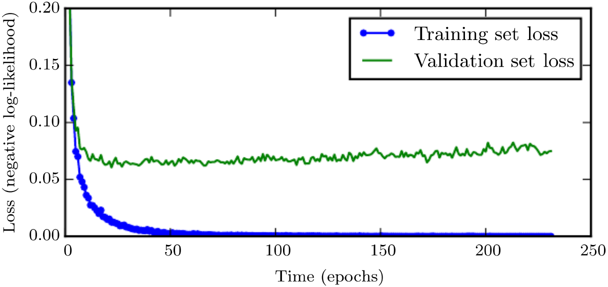


Image from Geron A. "Hands-On Machine Learning with Scikit-Learn and TensorFlow", O'Reilly 2017

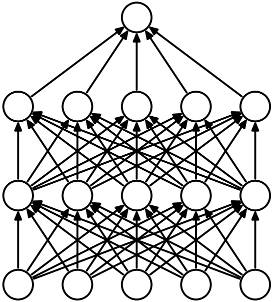
Regularization

Early stopping:
Stop iterations early to avoid overfitting to the training set of data

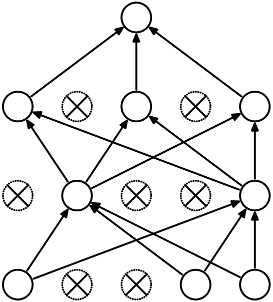


Regularization

Dropout: Randomly remove network units with some probability α



(a) Standard Neural Net



(b) After applying dropout.

Image from Srivastava *et al.*. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"

Regularization

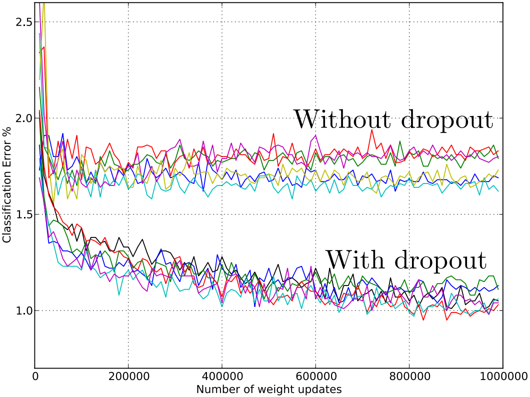


Image from Srivastava *et al.*. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"