Reading Group on Deep Learning Session 1

Stephane Lathuiliere & Pablo Mesejo

2 June 2016

Contents

Introduction to Artificial Neural Networks to understand, and to be able to efficiently use, the popular and successful techniques known as **Deep Learning**.

Sessions 1 and 2: discussion and analysis of the Chapter 5 about Neural Networks from the book *"Pattern Recognition and Machine Learning"* by Christopher M. Bishop (Springer, 2006)

- Session 1 (2nd of June): 5.1, 5.2, 5.3. and 5.4.
- Session 2 (10th of June): 5.5, 5.6 and 5.7.

"The importance of neural networks is that they offer a very powerful and very general framework for representing non-linear mappings from several input variables to several output variables, where the form of the mapping is governed by a number of adjustable parameters."
Christopher Bishop

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.

Introduction

Given a training data set comprising N observations $\{\mathbf{x}_n\}$, where $n = 1, \ldots, N$, together with corresponding target values $\{t_n\}$, the goal is to predict the value of t for a new value of $\mathbf{x}$:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \tag{3.7}$$

- $y(\mathbf{x}, \mathbf{w})$ is the model ($\mathbf{w}$ are the parameters).
- ϵ is the residual error.

More generally, from a probabilistic perspective, we aim to model the predictive distribution $p(t|\mathbf{x})$ because this expresses our uncertainty about the value of t for each value of $\mathbf{x}$.

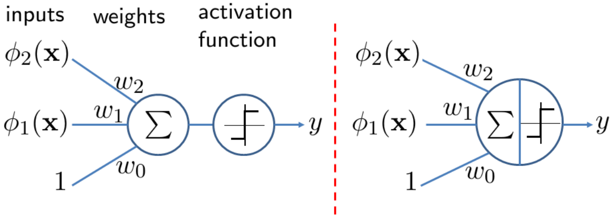
Introduction: the perceptron (Rosenblatt, 1957)

Linear discriminant model (Section 4.1.7.):

$$y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x})) \tag{4.52}$$

- $\phi(\mathbf{x})$ is a feature vector obtained using a fixed nonlinear transformation of input vector $\mathbf{x}$.
- $\mathbf{w} = (w_0, \dots, w_M)^\top$ are the model coefficients, or weights.
- $f(\cdot)$ is a nonlinear activation function (sign function)

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$



$$w_2\phi_2(\mathbf{x}) + w_1\phi_1(\mathbf{x}) + w_0 = 0$$

Introduction: the perceptron (Rosenblatt, 1957)

We are seeking $\mathbf{w}$ s.t. $\mathbf{x}_n$ in class $\mathcal{C}_1$ will have $\mathbf{w}^T\phi(\mathbf{x}_n) > 0$, whereas $\mathbf{x}_n$ in class $\mathcal{C}_2$ have $\mathbf{w}^T\phi(\mathbf{x}_n) < 0$.

Using $t \in \{-1, +1\}$, the perceptron criterion is

$$E_P(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^T\phi(\mathbf{x}_n)t_n \tag{4.54}$$

where $\mathcal{M}$ denotes the set of all misclassified patterns.

We now apply the stochastic gradient descent algorithm to minimize this error function. The change in the weight vector $\mathbf{w}$ is given by

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta\phi(\mathbf{x}_n)t_n \tag{4.55}$$

Introduction: the perceptron (Rosenblatt, 1957)

Perceptron convergence theorem: if the training data set is linearly separable, then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps.

Problems:

- Learning algorithm:
  - Linearly separable: there may be many solutions, and which one is found will depend on the *parameters initialization* and on the *order of presentation* of the data points.
  - Not linearly separable: the algorithm will never converge.
- Does not provide probabilistic outputs.
- Does not generalize readily to $K > 2$ classes.
- It is based on linear combinations of fixed basis functions.

A closely related system called the *adaline* ('adaptive linear element') was also presented by Widrow and Hoff (1960).

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.

Chapter Structure

5.1. Activations

The basic neural network model can be described as a series of functional transformations.

Construct M linear combinations of the inputs $x_1, \ldots, x_D$:

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{5.2}$$

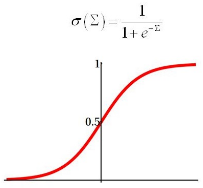
- $a_j^{(1)}$ are the layer one activations, $j = 1, \ldots, M$.
- $w_{ji}^{(1)}$ are the layer one weights, $i = 1 \ldots D$.
- $w_{j0}^{(1)}$ are the layer one biases, that allow for any fixed offset in the data.

Each linear combination $a_j^{(1)}$ is transformed by a (nonlinear, differentiable) activation function:

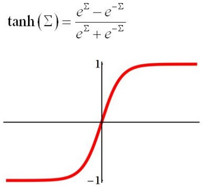
$$z_j = h(a_j^{(1)}) \tag{5.3}$$

5.1. Activations

The nonlinear functions $h(\cdot)$ are generally chosen to be sigmoidal functions such as the logistic sigmoid or the 'tanh'.



logistic (sigmoid, unipolar)



tanh (bipolar)

- "In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU [as activation function]" (Chapter 6, "Deep Learning" by Goodfellow et al, 2016)
  - "Deep convolutional neural networks with ReLUs train several times faster than their equivalents with 'tanh' units." ("ImageNet Classification with Deep Convolutional Neural Networks" by Krizhevsky et al, NIPS 2012)

5.1. Output Activations

The hidden outputs $z_j = h(a_j)$ are linearly combined in layer two:

$$a_k^{(2)} = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{5.4}$$

- $a_k^{(2)}$ are the layer two output activations, $k = 1, \ldots, K$.
- $w_{kj}^{(2)}$ are the layer two weights, $j = 1 \ldots D$.
- $w_{k0}^{(2)}$ are the layer two biases.

The output activations $a_k^{(2)}$ are transformed by the output activation function:

$$y_k = \sigma(a_k^{(2)}) \tag{5.5}$$

- y_k are the final outputs.
- $\sigma(\cdot)$ can be, like $h(\cdot)$, a logistic sigmoid function.

5.1. The Complete Two-Layer Model

The model $y_k = \sigma(a_k)$ is, after substituting the definitions of a_j and a_k:

1..K outputs

$$y_k(\mathbf{x},\mathbf{w}) = \sigma\left(\overbrace{\sum_{j=0}^{M} w_{kj}^{(2)} \underbrace{h\left(\overbrace{\sum_{i=0}^{D} w_{ji}^{(1)} x_i}^{a_j}\right)}_{\phi_j(\mathbf{x})}}^{a_k}\right) \qquad (5.9)$$

0..D inputs

0..M hidden units

- Regression problems, the activation function $\sigma(\cdot)$ is the identity so that $y_k = a_k$.
- Classification problems, each output unit activation maps to a posterior probability (e.g. using a logistic sigmoid function).
- Evaluation of (5.9) is called forward propagation.

5.1. Network Diagram

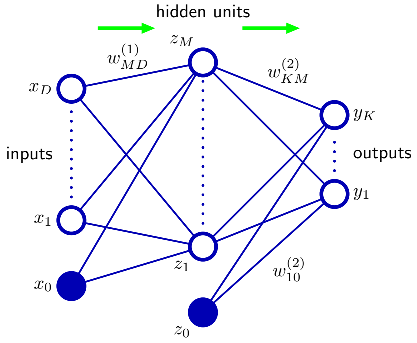


Figure: 5.1

- Nodes are input, hidden and output units. Links are corresponding weights.
- Fig. 5.1 displays a two-layer network: $w^{(1)}$ is the first layer and $w^{(2)}$ is the second one.
- Information propagates 'forwards' from the explanatory variable $\mathbf{x}$ to the estimated response $y_k(\mathbf{x}, \mathbf{w})$.

5.1. Properties & Generalizations

- If all hidden units have linear $h(\cdot)$ then we can always find an equivalent network without hidden units.
- Individual units do not need to be fully connected to the next layer and their links may skip over one or more subsequent layers.
- Networks with two or more layers are universal approximators:
  - Any continuous function can be uniformly approximated to arbitrary accuracy, given enough hidden units.
  - This is true for many definitions of $h(\cdot)$, but excluding polynomials.
  - The key problem is how to find suitable parameter values given a set of training data.
- There may be symmetries in the weight space, meaning that different choices of $\mathbf{w}$ may define the same mapping from input to output.

Chapter Structure

- Introduction.
- 5.1. Feed-forward Network Functions.
- 5.2. Network Training.
- 5.3. Error Backpropagation.
- 5.4. The Hessian Matrix.
- 5.5. Regularization in Neural Networks.
- 5.6. Mixture Density Networks.
- 5.7. Bayesian Neural Networks.

Chapter Structure

5.2. Network Training

Given a training set comprising a set of input vectors $\{\mathbf{x}_n\}$, where $n = 1, \ldots, N$, together with a corresponding set of target vectors $\{t_n\}$, we minimize the error function

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left\| y(\mathbf{x}_n, \mathbf{w}) - t_n \right\|^2 \tag{5.11}$$

The aim is to minimize the residual error between $y(\mathbf{x}_n, \mathbf{w})$ and t_n.

We can provide a much more general view of network training by first giving a probabilistic interpretation to the network outputs.

5.2. Network Training

A single target variable t has a Gaussian distribution with an $\mathbf{x}$-dependent mean, given by the output of the neural network:

$$p(t|\mathbf{x},\mathbf{w}) = \mathcal{N}\left(t \,\middle|\, y(\mathbf{x},\mathbf{w}), \beta^{-1}\right) \tag{5.12}$$

where β is the precision (inverse variance) of the Gaussian noise.

Given N independent and identically distributed observations, we can construct the corresponding likelihood function

$$p\left(\{t_1,\dots,t_N\}\middle|\{\mathbf{x}_1,\dots,\mathbf{x}_N\},\mathbf{w},\beta\right) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n,\mathbf{w},\beta)$$

Taking the negative logarithm, we obtain the error function

$$\frac{\beta}{2}\sum_{n=1}^{N}\left\{y(\mathbf{x}_n,\mathbf{w}) - t_n\right\}^2 - \frac{N}{2}\log\beta + \frac{N}{2}\log 2\pi \tag{5.13}$$

which can be used to learn the parameters $\mathbf{w}$ and β.

5.2. Maximum Likelihood $\mathbf{w}$

A widely used frequentist estimator is maximum likelihood, in which $\mathbf{w}$ is set to the value that maximizes the likelihood function $p(\mathcal{D}|\mathbf{w})$:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})} \qquad (1.43)$$

$$posterior \propto likelihood \times prior$$

Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function given by

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{y(\mathbf{x}_n, \mathbf{w}) - t_n\right\}^2 \qquad (5.14)$$

where we have discarded additive and multiplicative constants.

The maximum-likelihood estimate of $\mathbf{w}$ can be obtained by minimizing $E(\mathbf{w})$:

$$\mathbf{w}_{\mathsf{ML}} = \min_{\mathbf{w}} \ E(\mathbf{w})$$

5.2. Maximum Likelihood β

Having obtained the ML parameter estimate $\mathbf{w}_{\mathsf{ML}}$, the precision, β can also be estimated. E.g. if the N observations are IID, then their joint probability is

$$p\Big(\{t_1,\ldots,t_N\}\Big|\{\mathbf{x}_1,\ldots,\mathbf{x}_N\},\mathbf{w},\beta\Big) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n,\mathbf{w},\beta)$$

The negative log-likelihood, in this case, is

$$-\log p = \beta E(\mathbf{w}_{\mathsf{ML}}) - \frac{N}{2}\log\beta + \frac{N}{2}\log 2\pi \qquad (5.13)$$

The derivative $\mathrm{d}/\mathrm{d}\beta$ is $E(\mathbf{w}_{\mathsf{ML}}) - \frac{N}{2\beta}$ and so

$$\frac{1}{\beta_{\mathsf{ML}}} = \frac{1}{N}2E(\mathbf{w}_{\mathsf{ML}}) = \frac{1}{N}\sum_{n=1}^{N}\Big\{y(\mathbf{x}_n,\mathbf{w}_{\mathsf{ML}}) - t_n\Big\}^2 \qquad (5.15)$$

And $1/\beta_{\mathsf{ML}} = \frac{1}{NK}2E(\mathbf{w}_{\mathsf{ML}})$ for K target variables.

5.2. Pairing of the error function and output units $h(\cdot)$

There is a natural choice of both the output unit activation function and matching error function, according to the type of problem being solved.

- Regression
  - $p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t \,\middle|\, y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$
  - Output activation function: identity

$\Rightarrow$Error function: sum-of-squares error (see Eq. 5.14)

- Binary Classification
  - $p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}$
  - Output activation function: logistic sigmoid, with $0 \leq y(\mathbf{x}, \mathbf{w}) \leq 1$.

$\Rightarrow$Error function: cross-entropy error

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \qquad (5.21)$$

where y_n denotes $y(\mathbf{x}_n, \mathbf{w})$

5.2. Pairing of the error function and output units $h(\cdot)$

- Multiclass Classification
  - $p(t|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k} \{1 - y_k(\mathbf{x}, \mathbf{w})\}^{1-t_k}$
  - Output activation function: softmax (or normalized exponential)

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{exp(a_k^{(2)}(\mathbf{x}, \mathbf{w}))}{\sum_j exp(a_j^{(2)}(\mathbf{x}, \mathbf{w}))} \qquad (5.25)$$

which satisfies $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$.

$\Rightarrow$Error function: multiclass cross-entropy error

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(\mathbf{x}_n, \mathbf{w}) \qquad (5.24)$$

5.2. Error Surface

The residual error $E(\mathbf{w})$ can be visualized as a surface in the weight-space:

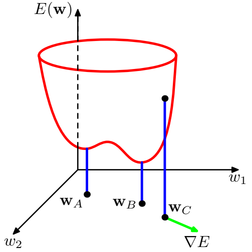


Figure: 5.5

- The error will, in practice, be highly multimodal.
- There will be inequivalent minima (local minima), determined by the particular data and model, as well as equivalent minima, corresponding to weight-space symmetries.

5.2. Parameter Optimization

Our goal:

$$\mathbf{w} = argmin(E(\mathbf{w})).$$

So we want to solve:

$$\nabla E(\mathbf{w}) = 0 \tag{5.26}$$

Iterative search for a local minimum of the error:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{5.27}$$

- τ is the time-step.
- $\Delta\mathbf{w}^{(\tau)}$ is the weight-vector update.
- vector $\nabla E(\mathbf{w})$ points in the direction of greatest rate of increase of the error function.

5.2. Gradient Descent

The simplest approach is to update $\mathbf{w}$ by a displacement in the negative gradient direction.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\big(\mathbf{w}^{(\tau)}\big) \tag{5.41}$$

where $\eta > 0$ is the learning rate.

- This is a batch method, as evaluation of ∇E involves the entire training data set.
- Stochastic methods can be used. Only a part of the training set is used at each iteration.
- Conjugate gradients or quasi-Newton methods may be preferred because they have the property that the error function always decreases at each iteration.
- Many initializations can be tested.

5.2. Optimization Scheme

Each iteration of the descent algorithm has two stages:

- I. Evaluate derivatives of error with respect to weights. An efficient method for the evaluation of $\nabla E(\mathbf{w})$ is needed. It involves backpropagation of error though the network.
- II. Use derivatives to compute adjustments of the weights (e.g. steepest descent).

Backpropagation is a general principle, which can be applied to many types of network and error function.

5.3. Simple Backpropagation

The error function is, typically, a sum over the data points $E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$. For example, consider a linear model (1 layer).

$$y_k = \sum_i w_{ki} x_i \tag{5.45}$$

The error function, for an individual input $\mathbf{x}_n$, is

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2, \quad \text{where} \quad y_{nk} = y_k(\mathbf{x}_n, \mathbf{w}). \tag{5.46}$$

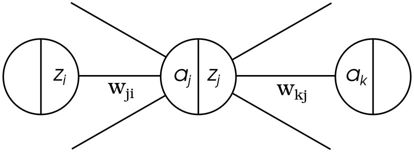
The gradient with respect to a weight w_{ji} is

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})\, x_{ni} \tag{5.47}$$

5.3. General Backpropagation

Recall that, in general, each unit computes a weighted sum:

$$a_j = \sum_i w_{ji} z_i \quad \text{with activation} \quad z_j = h(a_j). \qquad (5.48,5.49)$$



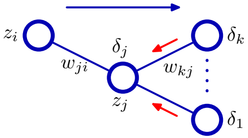
For each error-term: $$\frac{\partial E_n}{\partial w_{ji}} = \underbrace{\frac{\partial E_n}{\partial a_j}}_{\equiv \delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i} = \delta_j z_i \qquad (5.50,5.53)$$

In the network: $$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \quad \text{where } j \to \{k\} \qquad (5.55)$$

Algorithm: $$\delta_j = \sum_k \delta_k w_{kj} h'(a_j) \quad \text{as} \quad \frac{\partial a_k}{\partial a_j} = \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \qquad (5.56)$$

5.3. Backpropagation Algorithm

- Forward pass: Apply input $\mathbf{x}$, and forward propagate to find all the a_i and z_i
- Back propagate the δ's to obtain a δ_j for each hidden unit. To do so, we first evaluate δ_k directly for the output units and then propagate them with $\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$.



- Evaluate the derivatives $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$.
- Update $\mathbf{w}$ by a displacement in the negative gradient direction:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\big(\mathbf{w}^{(\tau)}\big) \qquad (5.41)$$

5.3. Computational Efficiency

The back-propagation algorithm is computationally more efficient than standard numerical minimization of E_n. Suppose that W is the total number of weights and biases in the network.

- Backpropagation: The evaluation is $O(W)$ for large W, as there are many more weights than units.
- Standard approach: Perturb each weight, and compute $E_n(w) - E\big(w + (0, \ldots, 0, \Delta w_{ij}, 0, \ldots, 0,)\big)$. This requires $W \times O(W)$ computations, so the total complexity is $O(W^2)$.

5.3. Jacobian Matrix

The properties of the network can be investigated via the Jacobian

$$J_{ki} = \frac{\partial y_k}{\partial x_i} \quad (5.70)$$

For example, (small) errors can be propagated through the trained network:

$$\Delta y_k \simeq \frac{\partial y_k}{\partial x_i} \Delta x_i \quad (5.72)$$

This is useful, but costly, as J_{ki} itself depends on $\mathbf{x}$. However, note that

$$\frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial x_i} = \sum_j w_{ji} \underbrace{\frac{\partial y_k}{\partial a_j}}_{\tilde{\delta}_{kj}} \quad (5.74)$$

As before, we can show $\tilde{\delta}_{kj} = h'(a_j) \sum_l w_{lj} \tilde{\delta}_{kl}$

5.4. Hessian Matrix

$$\mathbf{H} = \left(\frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}} \right) \tag{5.78}$$

Backpropagation principle can be used to get:

- Diagonal approximation of H in $O(W)$
- Approximation of the Hessian in $O(W^2)$
- Exact computation of the Hessian in $O(W^2)$
- Compute $v^T H$ in $O(W)$