

# 732A99/TDDE01 Machine Learning

## Lecture 3c Block 1: Neural Networks

Jose M. Peña

IDA, Linköping University, Sweden

# Contents

- ▶ Neural Networks
- ▶ Backpropagation Algorithm
- ▶ Regularization
- ▶ Summary

# Literature

- ▶ Main source
  - ▶ Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Sections 5.1-5.3.3 and 5.5.2.
- ▶ Additional source
  - ▶ Hastie, T., Tibshirani, R. and Friedman, J. *The Elements of Statistical Learning*. Springer, 2009. Chapter 11.

- ▶ Consider binary classification with input space  $\mathbb{R}^D$ . Consider a training set  $\{(\mathbf{x}_n, t_n)\}$  where  $t_n \in \{-1, +1\}$ .
- ▶ SVMs classify a new point  $\mathbf{x}$  according to

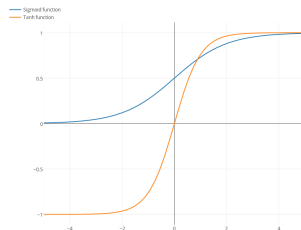
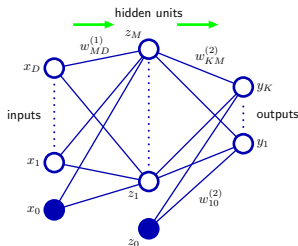
$$y(\mathbf{x}) = \text{sgn} \left( \sum_{m \in S} a_m t_m k(\mathbf{x}, \mathbf{x}_m) + b \right)$$

- ▶ Consider regressing an unidimensional continuous random variable on a  $D$ -dimensional continuous random variable. Consider a training set  $\{(\mathbf{x}_n, t_n)\}$
- ▶ For a new point  $\mathbf{x}$ , SVMs predict

$$y(\mathbf{x}) = \sum_{m \in S} (a_m - \widehat{a}_m) k(\mathbf{x}, \mathbf{x}_m) + b$$

- ▶ SVMs imply **data-selected** **user-defined** basis functions.
- ▶ NNs imply a **user-defined** number of **data-selected** basis functions.

# Neural Networks



- ▶ Activations:  $a_j = \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- ▶ Hidden units and activation function:  $z_j = h(a_j)$
- ▶ Output activations:  $a_k = \sum_j w_{kj}^{(2)} z_j + w_{k0}^{(2)}$
- ▶ Output activation function for regression:  $y_k(\mathbf{x}) = a_k$
- ▶ Output activation function for classification:  $y_k(\mathbf{x}) = \sigma(a_k)$
- ▶ Sigmoid function:  $\sigma(a) = \frac{1}{1+\exp(-a)}$
- ▶ Two-layer NN:

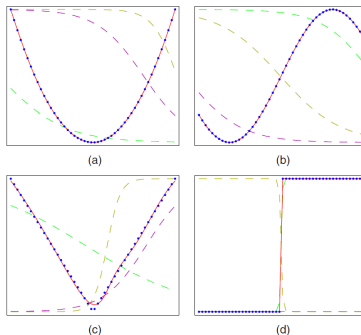
$$y_k(\mathbf{x}) = \sigma\left(\sum_j w_{kj}^{(2)} h\left(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

- ▶ Evaluating the previous expression is known as forward propagation. The NN is said to have a feed-forward architecture.
- ▶ All the previous is, of course, generalizable to more layers.

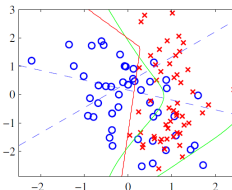
# Neural Networks

- For a large variety of activation functions, the two-layer NN can uniformly approximate any continuous function to arbitrary accuracy **provided enough hidden units**. Easy to fit the parameters ? Overfitting ?!

**Figure 5.3** Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a)  $f(x) = x^2$ , (b)  $f(x) = \sin(x)$ , (c)  $f(x) = |x|$ , and (d)  $f(x) = H(x)$  where  $H(x)$  is the Heaviside step function. In each case,  $N = 50$  data points, shown as blue dots, have been sampled uniformly in  $x$  over the interval  $(-1, 1)$  and the corresponding values of  $f(x)$  evaluated. These data points are then used to train a two-layer network having 3 hidden units with 'tanh' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.

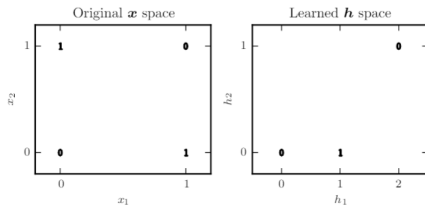
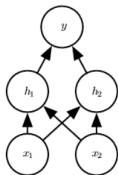


**Figure 5.4** Example of the solution of a simple two-class classification problem involving synthetic data using a neural network having two inputs, two hidden units with 'tanh' activation functions, and a single output having a logistic sigmoid activation function. The dashed blue lines show the  $z = 0.5$  contours for each of the hidden units, and the red line shows the  $y = 0.5$  decision surface for the network. For comparison, the green line denotes the optimal decision boundary computed from the distributions used to generate the data.



# Neural Networks

- ▶ Solving the XOR problem with NNs.
- ▶ No line shatters the points in the original space.
- ▶ The NN represents a mapping of the input space to an alternative space where a line can shatter the points. Note that the points (0,1) and (1,0) are mapped both to the point (1,0).
- ▶ **It resembles SVMs.**



$$w_{11}^{(1)} = w_{12}^{(1)} = w_{21}^{(1)} = w_{22}^{(1)} = 1$$

$$w_{10}^{(1)} = 0, w_{20}^{(1)} = -1$$

$$h_j = z_j = h(a_j) = \max\{0, a_j\}$$

$$w_{11}^{(2)} = 1, w_{12}^{(2)} = -2$$

$$w_{10}^{(2)} = 0$$

$$y = y_k = a_k$$

## Backpropagation Algorithm

- ▶ Consider regressing an  $K$ -dimensional continuous random variable on a  $D$ -dimensional continuous random variable.
- ▶ Consider a training set  $\{(\mathbf{x}_n, \mathbf{t}_n)\}$ . Consider minimizing the sum-of-squares error function

$$E(\mathbf{w}) = \sum_n E_n(\mathbf{w}) = \sum_n \frac{1}{2} \|\mathbf{y}(\mathbf{x}_n) - \mathbf{t}_n\|^2 = \sum_n \sum_k \frac{1}{2} (y_k(\mathbf{x}_n) - t_{nk})^2$$

- ▶ This error function can be justified from a maximum likelihood approach to learning  $\mathbf{w}$ . To see it, **assume** that

$$p(t_k | \mathbf{x}, \mathbf{w}, \sigma) = \mathcal{N}(t_k | y_k(\mathbf{x}), \sigma)$$

- ▶ Then, the likelihood function is

$$p(\{\mathbf{t}_n\} | \{\mathbf{x}_n\}, \mathbf{w}, \sigma) = \prod_n \prod_k \mathcal{N}(t_{nk} | y_k(\mathbf{x}_n), \sigma) = \prod_n \prod_k \frac{1}{(2\pi\sigma^2)^{1/2}} e^{-\frac{1}{2\sigma^2} (t_{nk} - y_k(\mathbf{x}_n))^2}$$

and thus

$$-\ln p(\{\mathbf{t}_n\} | \{\mathbf{x}_n\}, \mathbf{w}, \sigma) = \sum_n \sum_k \frac{1}{2\sigma^2} (t_{nk} - y_k(\mathbf{x}_n))^2 + \frac{NK}{2} \ln \sigma^2 + \frac{NK}{2} \ln 2\pi$$

which is equivalent to the sum-of-squares error function for a given  $\sigma$ .

- ▶ If  $\sigma$  is not given, then we can find the ML estimates of  $\mathbf{w}$ , plug them into the log likelihood function, and maximize it with respect to  $\sigma$ .

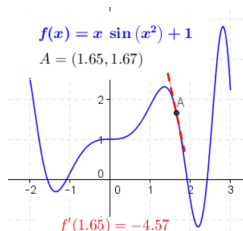
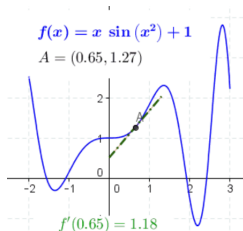


## Backpropagation Algorithm

- ▶ The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.
- ▶ **Batch** gradient descent

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \nabla E(\mathbf{w}^t)$$

where  $\eta_t > 0$  is the learning rate, and  $\nabla E(\mathbf{w}^t)$  is a vector whose components are the partial derivatives of  $E(\mathbf{w}^t)$ .

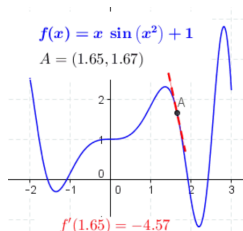
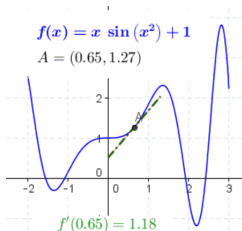


- ▶ To ensure that gradient descent converges, we need to check that

$$\sum_{t=1}^{\infty} \eta_t = \infty \text{ and } \sum_{t=1}^{\infty} \eta_t^2 < \infty$$

that is, not too small to avoid slow convergence and not too big to avoid jumping around the optimum, e.g.  $\eta_t = 1/t$ .

# Backpropagation Algorithm



- **Sequential, stochastic** or online gradient descent

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \nabla E_n(\mathbf{w}^t)$$

where  $n$  is chosen randomly or sequentially.

- Sequential gradient descent is less affected by the multimodality problem, as a local minimum of the whole data will not be generally a local minimum of each individual point.

## Backpropagation Algorithm

- ▶ Since  $E_n$  depends on  $w_{ji}$  only via  $a_j$ , and  $a_j = \sum_i w_{ji} x_i$ , then

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} x_i = \delta_j x_i$$

- ▶ Since  $E_n$  depends on  $a_j$  only via  $a_k$ , then

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j}$$

- ▶ Since  $a_k = \sum_j w_{kj} z_j$  and  $z_j = h(a_j)$ , then

$$\frac{\partial a_k}{\partial a_j} = h'(a_j) w_{kj}$$

- ▶ Putting all together, we have that

$$\delta_j = h'(a_j) \sum_k \delta_k w_{kj}$$

- ▶ Since  $y_k = a_k$  for regression and  $a_k = \sum_j w_{kj} z_j$ , then

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}} = \delta_k z_j \text{ and } \delta_k = \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

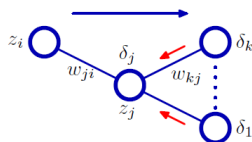
- ▶ Backpropagation algorithm:

1. Forward propagate to compute activations, and hidden and output units.
2. Compute  $\delta_k$  for the output units.
3. Backpropagate the  $\delta$ 's, i.e. evaluate  $\delta_j$  for the hidden units recursively.
4. Compute the required derivatives.

# Backpropagation Algorithm

- ▶ Backpropagation algorithm:
  1. Forward propagate to compute activations, and hidden and output units.
  2. Compute  $\delta_k$  for the output units.
  3. Backpropagate the  $\delta$ 's, i.e. evaluate  $\delta_j$  for the hidden units recursively.
  4. Compute the required derivatives.

**Figure 5.7** Illustration of the calculation of  $\delta_j$  for hidden unit  $j$  by backpropagation of the  $\delta$ 's from those units  $k$  to which unit  $j$  sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.



- ▶ For classification, we minimize the negative log likelihood function, a.k.a. cross-entropy error function:

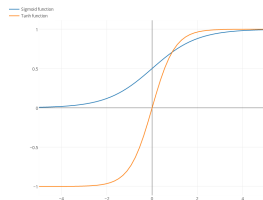
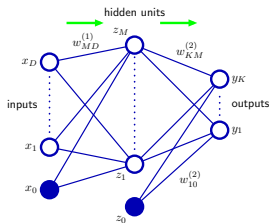
$$E_n(\mathbf{w}) = - \sum_k [t_{nk} \ln y_k(\mathbf{x}_n) + (1 - t_{nk}) \ln(1 - y_k(\mathbf{x}_n))]$$

with  $t_{nk} \in \{0, 1\}$  and  $y_k(\mathbf{x}_n) = \sigma(a_k)$ . Then, again

$$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \text{ and } \delta_k = \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

- ▶ This is an example of embarrassingly parallel algorithm.

# Backpropagation Algorithm



- ▶ Example:  $y_k = a_k$ , and  $z_j = h(a_j) = \tanh(a_j)$  where  $\tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$ .
- ▶ Note that  $h'(a) = 1 - h(a)^2$ .
- ▶ Backpropagation:

1. Forward propagation, i.e. compute

$$a_j = \sum_i w_{ji} x_i \text{ and } z_j = h(a_j) \text{ and } y_k = \sum_j w_{kj} z_j$$

2. Compute

$$\delta_k = y_k - t_k$$

3. Backpropagate, i.e. compute

$$\delta_j = (1 - z_j^2) \sum_k w_{kj} \delta_k$$

4. Compute

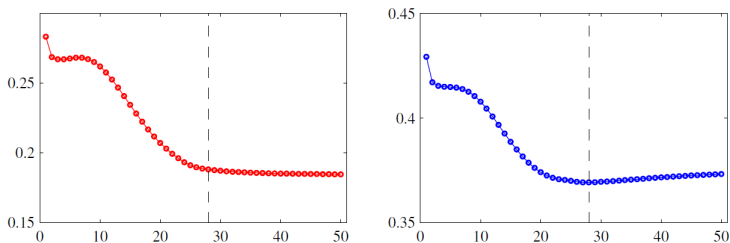
$$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \text{ and } \frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

# Backpropagation Algorithm

- ▶ The weight space is non-convex and has many symmetries, plateaus and local minima. So, the initialization of the weights in the backpropagation algorithm is crucial.
- ▶ Hints based on experimental rather than theoretical analysis:
  - ▶ Initialize the weights to different values, otherwise they would be updated in the same way because the algorithm is deterministic, and so creating redundant hidden units.
  - ▶ Initialize the weights at random, but
    - ▶ too small magnitude values may cause losing signal in the forward or backward passes, and
    - ▶ too big magnitude values may cause the activation function to saturate and lose gradient.
  - ▶ Initialize the weights to almost-zero values so that the initial model is almost-linear, i.e. the sigmoid function is almost-linear around the zero. Let the algorithm to introduce non-linearities where needed.

# Regularization

- Regularization when learning the parameters: Early stopping the backpropagation algorithm according to the error on some validation data.



**Figure 5.12** An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

## Summary

- ▶ NNs: Nonlinear mapping from input to output.
- ▶ Extremely expressive.
- ▶ Training: Backpropagation algorithm, and regularization.