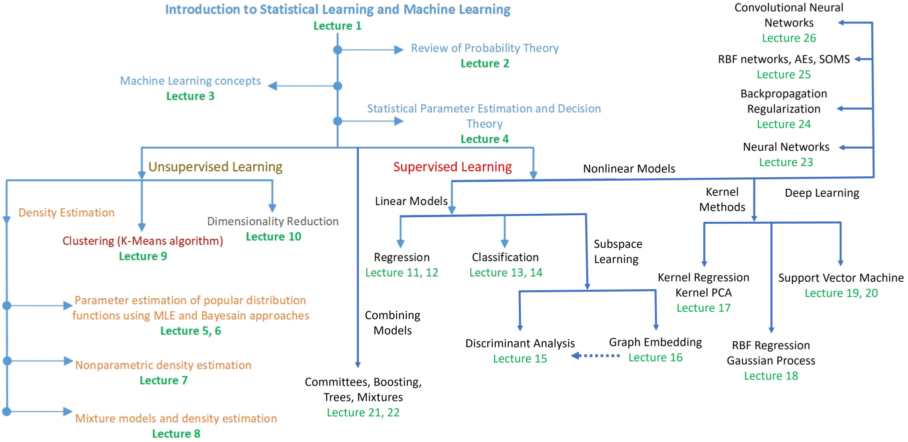
Statistical Learning and Machine Learning

Lecture 23 - Neural Networks 1

October 13, 2021

Course overview and where do we stand



Linear and kernel methods

Linear Basis Function models:

- Useful analytical properties (convex optimization)
- Usually fast to compute
- Manual design of the basis functions

Kernel methods:

- Useful analytical properties (convex optimization)
- Efficient for large D and small N
- Inefficient (even intractable) for large N
- A-priori design of the kernel functions

Why Neural Networks?

Neural networks:

- Adaptive (parametric) basis functions
- Hierarchical data transformations
- Usually more compact (and efficient) than kernel methods of similar performance
- Non-convex optimization
- Neural networks comprising of many hidden layers (*Deep Learning*) have proven to be effective in many machine learning problems

The artificial neuron

The basic building block of a neural network is called *neuron*:

- It receives as input a vector, e.g. $\mathrm{x} \in \mathbb{R}^D$ and applies the following transformation:

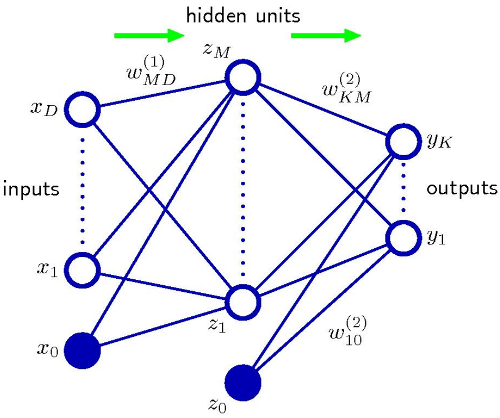
$$\alpha = \sum_{i=1}^{D} w_i x_i + w_0 = \mathrm{w}^T \mathrm{x} + w_0 \tag{1}$$

$$z = h(\alpha) \tag{2}$$

  where:

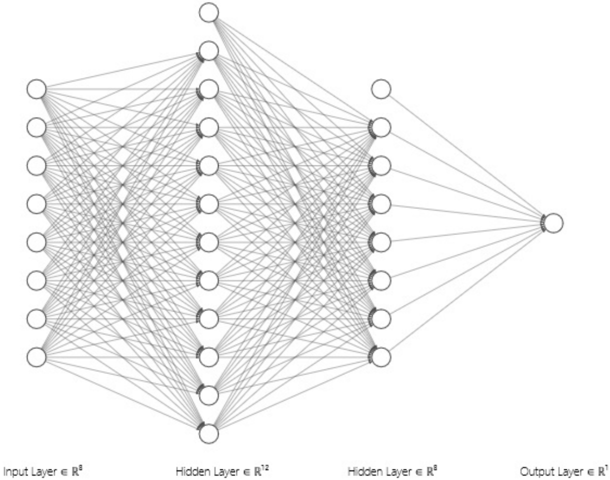
  - $\{\mathrm{w}, w_0\}$ are the parameters of the neuron
  - w is called *weight* and w_0 is called *bias*
  - α is known as the *activation*
  - $h(\cdot)$ is a nonlinear *activation function*
  - $h(\cdot)$ can take many forms, depending on the position of the neuron in the neural network and the problem at hand

A two-layer feed-forward neural network

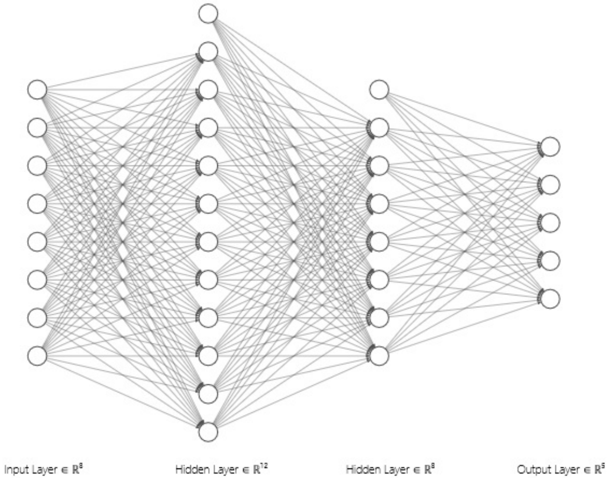


$$y_k(\mathrm{x}, \mathrm{w}) = \sigma\left(\sum_{j=1}^{M} W^{(2)}_{kj}\, h\left(\sum_{i=1}^{D} W^{(1)}_{ji} x_i + w^{(1)}_{j0}\right) + w^{(2)}_{k0}\right) \qquad (3)$$

Multi-layer feed-forward neural network



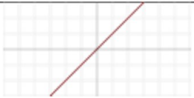
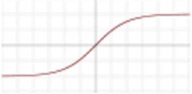
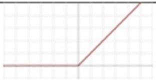
Multi-layer feed-forward neural network



Activation functions

Name	Equation	Derivative
Identity	$f(x) = x$	$f'(x) = 1$
Logistic	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
HyperbTan	$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan	$f(x) = \tanh^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
ReLU	$f(x) = \begin{cases} 0, \text{ if } x \leq 0 \\ x, \text{ if } x > 0 \end{cases}$	$f(x) = \begin{cases} 0, \text{ if } x \leq 0 \\ 1, \text{ if } x > 0 \end{cases}$
Softmax	$f_i(\mathbf{x}) = \frac{e^{x_k}}{\sum_{l=1}^{K} e^{x_l}}, \quad k = 1, \ldots, K$	$\frac{\vartheta f_i(\mathbf{x})}{\vartheta x_j} = f_i(\mathbf{x})(\delta_{ij} - f_j(\mathbf{x}))$

Activation functions

Name	Plot
Identity	
Logistic	
HypTan	
ArcTan	
ReLU	

A two-layer feed-forward neural network

If we 'absorb' the bias parameters in the corresponding weight vectors using additional dimensions (having a value equal to 1) on the input and hidden-layer output vectors:

$$y_k(\mathrm{x}, \mathrm{w}) = \sigma\left(\sum_{j=0}^{M} W_{kj}^{(2)}\, h\left(\sum_{i=0}^{D} W_{ji}^{(1)} x_i\right)\right) = \sigma\Big(\mathrm{W}^{(2)T}\, \underbrace{h(\mathrm{W}^{(1)T}\tilde{\mathrm{x}})}_{\tilde{\mathrm{z}}}\Big) \quad (4)$$

where $\tilde{\mathrm{z}} \in \mathbb{R}^{M+1}$ and:

$$\begin{aligned} \mathrm{W}^{(1)} &= [\mathrm{w}_1^{(1)}, \ldots, \mathrm{w}_M^{(1)}] \in \mathbb{R}^{(D+1)\times M} \\ \mathrm{W}^{(2)} &= [\mathrm{w}_1^{(2)}, \ldots, \mathrm{w}_K^{(2)}] \in \mathbb{R}^{(M+1)\times K} \end{aligned}$$

Note that the activation functions are applied *element-wise* (on each dimension

Multilayer Perceptron

The above neural network is called *Multilayer Perceptron* (or MLP):

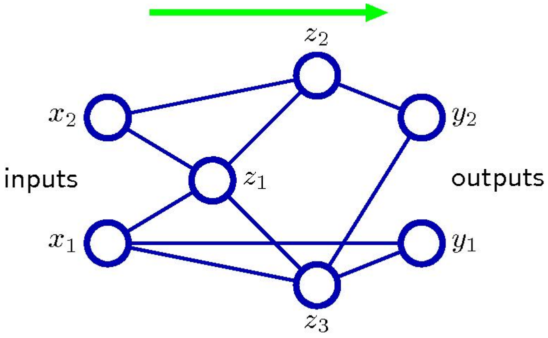
- an important property is that the activation functions of all neurons are differentiable w.r.t. their parameters

The use of nonlinear activation functions is crucial:

- If we use linear activation functions in the above two-layer neural network:
$$y_k(\mathrm{x}, \mathrm{w}) = \mathrm{W}^{(2)^T} \mathrm{W}^{(1)^T} \tilde{\mathrm{x}} = \mathcal{W}^T \tilde{\mathrm{x}} \tag{5}$$
where $\mathcal{W} = \mathrm{W}^{(2)^T} \mathrm{W}^{(1)^T} \in \mathbb{R}^{(D+1)\times K}$.
- Thus, any neural network with more than one layers and linear activation functions correspond to an one-layer neural network.

Skip connections

The connections in the network do not need to be restricted between neurons of successive layers (however they need to be feed-forward):

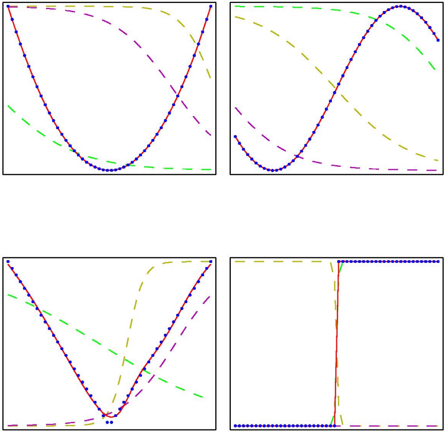


Universal approximators

The approximation properties of feed-forward networks have been widely studied:

- Neural networks are said to be *universal approximators*.
- A two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units
- This holds for a wide range of hidden unit activation functions (excluding polynomials)

Universal approximators



(left to right) $f(x) = x^2$, $f(x) = sin(x)$, $f(x) = |x|$ and $f(x) = H(x)$
Red: approximation of $f(x)$ using a two-layer network with 3 hidden neurons (*tanh* act. function - color lines) and linear output

Weight-space symmetries

Let us consider the above two-layer network (having *tanh* activation functions):

- If we use $\tilde{\mathrm{W}}^{(1)} = -\mathrm{W}^{(1)}$, then the output of the hidden layer is $\tilde{\zeta} = -\tilde{\mathrm{z}}$ (because $tanh(-z_i) = -tanh(z_i)$)
- Using $\tilde{\mathrm{W}}^{(2)} = -\mathrm{W}^{(2)}$ leads to the same output as before.

There exist multiple combinations of different weight values which can give the same result!

Neural Network Training: Regression

Given a set of data points x_n, $n = 1, \ldots, N$, the corresponding target values t_n and including all the weights of the neural network in a parameter w:

- we assume that t follows a Gaussian distribution:

$$p(t|\mathrm{x}, \mathrm{w}) = \mathcal{N}(t|y(\mathrm{x}, \mathrm{w}), \beta^{-1}) \tag{6}$$

  where β is the precision (inverse variance) of the distribution.
- The likelihood function is (we use linear output neurons):

$$p(\mathrm{t}|\mathrm{X}, \mathrm{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathrm{x}_n, \mathrm{w}, \beta) \tag{7}$$

- The error function is the negative log-likelihood (discarding the terms not depending on w and scaling factors):

$$E(\mathrm{w}) = \frac{1}{2} \sum_{n=1}^{N} \Big(y(\mathrm{x}_n, \mathrm{w}) - t_n \Big)^2 \tag{8}$$

Neural Network Training: Regression

Minimizing $E(\mathrm{w})$ will lead to w_{ML}:

- Due to the use of nonlinear activation functions for the hidden neurons, w_{ML} will be a local minimum of $E(\mathrm{w})$ (non-convex optimization)
- Using w_{ML} we can optimize for β by minimizing the negative log-likelihood function:

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n=1}^{N} \Big(y(\mathrm{x}_n, \mathrm{w}_{ML}) - t_n \Big)^2 \tag{9}$$

The optimizations w.r.t. w and β are performed by using an iterative optimization process.

Neural Network Training: Regression

When the targets are vectors t_n, $n = 1, \ldots, N$:

- We obtain w_{ML} by minimizing:

$$E(\mathrm{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathrm{y}(\mathrm{x}_n, \mathrm{w}) - \mathrm{t}_n\|^2 \tag{10}$$

- We use w_{ML} to optimize for β_{ML}:

$$\frac{1}{\beta_{ML}} = \frac{1}{NK} \|\mathrm{y}(\mathrm{x}_n, \mathrm{w}_{ML}) - \mathrm{t}_n\|^2 \tag{11}$$

Neural Network Training: Binary classification

Consider that the targets are $t_n \in \{0, 1\}$, where:

- $t_n = 1$ means that $\mathrm{x}_n \in \mathcal{C}_1$
- $t_n = 0$ means that $\mathrm{x}_n \in \mathcal{C}_2$

The neural network will have one output neuron with logistic sigmoid activation function:

$$y = \sigma(\alpha) = \frac{1}{1 + exp(-\alpha)} \tag{12}$$

ensuring that $0 \leq y(\mathrm{x}, \mathrm{w}) \leq 1$.

Neural Network Training: Binary classification

The conditional distribution of t w.r.t. x and w is then a Bernoulli distribution:

$$p(t|\mathrm{x}, \mathrm{w}) = y(\mathrm{x}, \mathrm{w})^t \left(1 - y(\mathrm{x}, \mathrm{w})\right)^{1-t} \tag{13}$$

The negative log-likelihood function becomes the *cross-entropy* error function:

$$E(\mathrm{w}) = -\sum_{n=1}^{N} \Big(t_n \ln y(\mathrm{x}_n, \mathrm{w}) + (1 - t_n) \ln(1 - y(\mathrm{x}_n, \mathrm{w})) \Big) \tag{14}$$

Neural Network Training: Multi-class classification

The target vectors $\mathrm{t}_n \in \mathbb{R}^K$ follow the 1-of-K coding scheme.

The error function is:

$$E(\mathrm{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathrm{x}_n, \mathrm{w}) \tag{15}$$

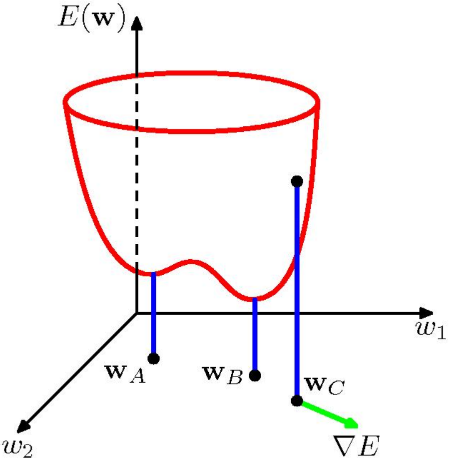
The output neuron activation function is the *softmax* function:

$$y_k(\mathrm{x}, \mathrm{w}) = \frac{exp(\alpha_k(\mathrm{x}, \mathrm{w}))}{\sum_j exp(\alpha_j(\mathrm{x}, \mathrm{w}))} \tag{16}$$

ensuring that $0 \leq y_k(\mathrm{x}, \mathrm{w}) \leq 1$ and $\sum_k y_k(\mathrm{x}, \mathrm{w}) = 1$.

Parameter optimization

Local minima correspond to $\nabla E(\mathrm{w}) = 0$.



Parameter optimization

Process:

1. Choose an initial set of parameter values $\mathrm{w}^{(0)}$
2. Update the parameters until reaching a local minimum using:

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{\tau} + \Delta \mathrm{w}^{(\tau)} \tag{17}$$

Several optimization techniques which use different choices for $\Delta \mathrm{w}^{(\tau)}$:

- Local quadratic approximation
- Gradient descent optimization
- Error Backpropagation
- Recent schemes: RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam

Parameter optimization: Local quadratic approximation

Taylor expansion of $E(\mathrm{w})$ around some point $\hat{\mathrm{w}}$:

$$E(\mathrm{w}) \cong E(\hat{\mathrm{w}}) + (\mathrm{w}-\hat{\mathrm{w}})^T \mathrm{b} + \frac{1}{2}(\mathrm{w}-\hat{\mathrm{w}})^T \mathrm{H}(\mathrm{w}-\hat{\mathrm{w}}) + \text{higher order terms} \quad (18)$$

where:

$$\mathrm{b} \equiv \nabla E|_{\mathrm{w}=\hat{\mathrm{w}}} \qquad \text{and} \qquad \mathrm{H} = \nabla\nabla E \quad (19)$$

Thus:

$$\nabla E \cong \mathrm{b} + \mathrm{H}(\mathrm{w} - \hat{\mathrm{w}}) \quad (20)$$

Parameter optimization: Gradient descent

Gradient descent updates w using:

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{\tau} - \eta \nabla E(\mathrm{w}^{(\tau)}) \tag{21}$$

To find a sufficiently good minimum, it may be necessary to run a gradient-based algorithm multiple times, each time using a different randomly chosen starting point.

When

$$E(\mathrm{w}) = \sum_{n=1}^{N} E_n(\mathrm{w}) \tag{22}$$

stochastic gradient descent (SGD) can be used:

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{\tau} - \eta \nabla E_n(\mathrm{w}^{(\tau)}) \tag{23}$$

Mini-batch gradient descent uses small chunks (e.g. 64) data points for each update.