Machine Learning

Lecture *Machine Learning* vom 23/24.3.2022

Lars Gabriel, Felix Becker, Mario Stanke
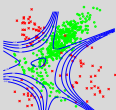Institut für Mathematik und Informatik
Universität Greifswald

Machine Learning

Lars Gabriel, Felix Becker, Mario Stanke

Artificial Neural Networks

Definition

Training (BackProp)

Activation Functions

Artificial Neural Network

$$\theta^{(0)} := \left(\theta_{ij}^{(0)}\right)_{\substack{1 \le i \le 16 \\ 1 \le j \le 7}}$$

$\theta_{ij}^{(0)}$ is the weight of the edge from x_i to $z_j^{(1)}$

$\theta^{(1)}$

$\theta^{(2)}$

Input Layer   Hidden Layer 1   Hidden Layer 2   Output Layer
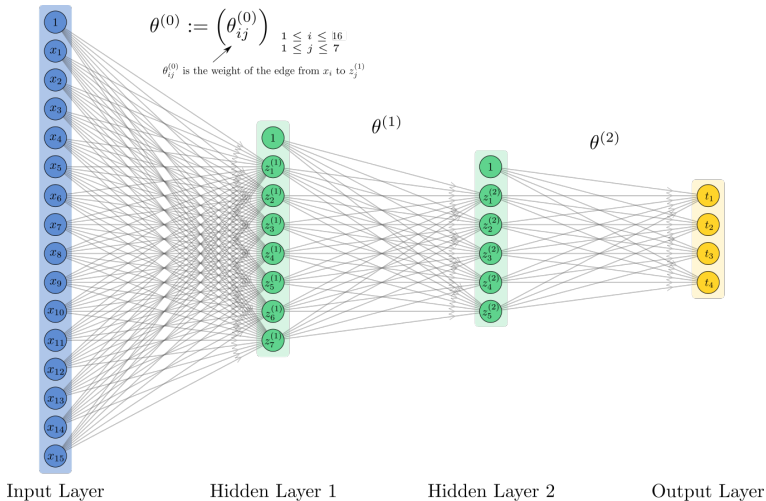
Definition 1 (Artificial Neural Network (NN, künstliches neuronales Netz))

A feed-forward artificial neural network with $L \geq 1$ layers of sizes $s_1, \ldots, s_L$ with $n =: s_0$ input variables $\mathbf{x} = (x_1, \ldots, x_n)^T$ and $K := s_L$ output variables $\mathbf{t} = (t_1, \ldots, t_K)^T$ is a function

$$\mathbf{t} = h_{\boldsymbol{\theta}}(\mathbf{x})$$
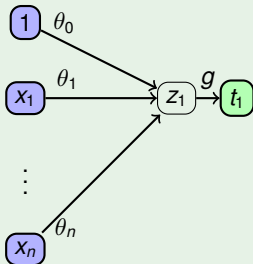
with parameters

$$\boldsymbol{\theta} = (\Theta^{(0)}, \ldots, \Theta^{(L-1)}) \qquad , \text{where} \quad \Theta^{(\ell)} \in \mathbb{R}^{s_{\ell+1} \times (s_\ell + 1)},$$

defined by the following recursions

$$
\begin{aligned}
\mathbf{t} &= \mathbf{g}(\mathbf{z}^{(L)}) \in \mathbb{R}^K \\
\mathbf{z}^{(\ell)} &= \Theta^{(\ell-1)}\mathbf{a}^{(\ell-1)} \in \mathbb{R}^{s_\ell} \quad (1 \leq \ell \leq L) \\
\mathbf{a}^{(\ell)} &= \begin{pmatrix} a_0^{(\ell)} \\ a_1^{(\ell)} \\ \vdots \\ a_{s_\ell}^{(\ell)} \end{pmatrix} = \begin{pmatrix} 1 \\ \sigma(z_1^{(\ell)}) \\ \vdots \\ \sigma(z_{s_\ell}^{(\ell)}) \end{pmatrix} \quad (1 \leq \ell < L) \\
\mathbf{a}^{(0)} &= \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}.
\end{aligned}
\tag{1}
$$

Here, σ is called an activation function and we will call $\mathbf{g} : \mathbb{R}^K \to \mathbb{R}^K$ the output activation function.

Linear regression = NN with 1 layer and 1 output variable

$L = 1$ layers

$$\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n) \quad \text{parameters}$$
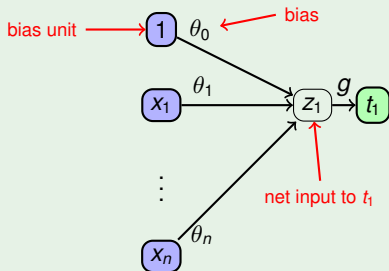
t_1 one output unit
$x_1, \ldots, x_n$ input units
output activation function $t_1 = g(z_1) = z_1$

$$t_1 = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

Linear regression = NN with 1 layer and 1 output variable

$L = 1$ layers

$\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)$ parameters
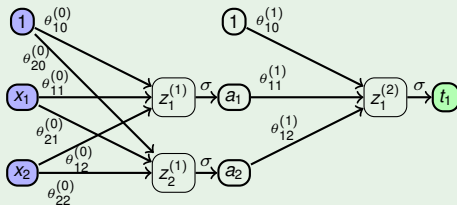
t_1 one output unit
$x_1, \ldots, x_n$ input units
output activation function $t_1 = g(z_1) = z_1$

$$t_1 = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

Artificial Neural Networks

NN with 1 hidden layer
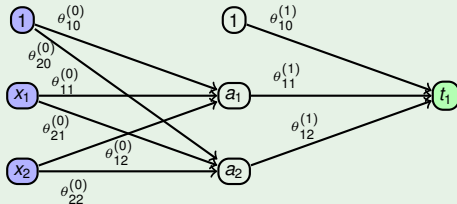

$L = 2$ layers, 1 hidden layer, $s_1 = 2$

$$\boldsymbol{\theta} = (\Theta^{(0)}, \Theta^{(1)}) \quad \text{parameters}$$

x_1, x_2 input units, t_1 single output unit with output activation function σ

$$t_1 = \sigma\left(\theta_{10}^{(1)} + \theta_{11}^{(1)}a_1 + \theta_{12}^{(1)}a_2\right) = \sigma\left(\theta_{10}^{(1)} + \theta_{11}^{(1)}\sigma(\theta_{10}^{(0)} + \theta_{11}^{(0)}x_1 + \theta_{12}^{(0)}x_2) + \theta_{12}^{(1)}\sigma(\theta_{20}^{(0)} + \theta_{21}^{(0)}x_1 + \theta_{22}^{(0)}x_2)\right)$$

Artificial Neural Networks

NN with 1 hidden layer
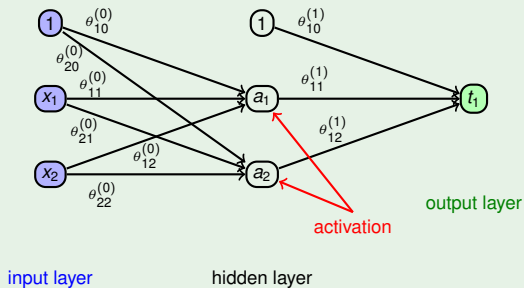

$L = 2$ layers, 1 hidden layer, $s_1 = 2$

$$\boldsymbol{\theta} = (\Theta^{(0)}, \Theta^{(1)}) \quad \text{parameters}$$

x_1, x_2 input units, t_1 single output unit with output activation function σ

$$t_1 = \sigma\left(\theta_{10}^{(1)} + \theta_{11}^{(1)}a_1 + \theta_{12}^{(1)}a_2\right) = \sigma\left(\theta_{10}^{(1)} + \theta_{11}^{(1)}\sigma(\theta_{10}^{(0)} + \theta_{11}^{(0)}x_1 + \theta_{12}^{(0)}x_2) + \theta_{12}^{(1)}\sigma(\theta_{20}^{(0)} + \theta_{21}^{(0)}x_1 + \theta_{22}^{(0)}x_2)\right)$$

Artificial Neural Networks

NN with 1 hidden layer

input layer   hidden layer

output layer

activation

$L = 2$ layers, 1 hidden layer, $s_1 = 2$

$$\boldsymbol{\theta} = (\Theta^{(0)}, \Theta^{(1)}) \quad \text{parameters}$$

x_1, x_2 input units, t_1 single output unit with output activation function σ

$$t_1 = \sigma\left(\theta_{10}^{(1)} + \theta_{11}^{(1)} a_1 + \theta_{12}^{(1)} a_2\right) = \sigma\left(\theta_{10}^{(1)} + \theta_{11}^{(1)} \sigma(\theta_{10}^{(0)} + \theta_{11}^{(0)} x_1 + \theta_{12}^{(0)} x_2) + \theta_{12}^{(1)} \sigma(\theta_{20}^{(0)} + \theta_{21}^{(0)} x_1 + \theta_{22}^{(0)} x_2)\right)$$

Suppose a *single* training observation

$$\mathbf{y} = (y_1, \ldots, y_K)^T$$

for training input

$$\mathbf{x} = (x_1, \ldots, x_n)^T$$

is given.

Suppose a *single* training observation

$$\mathbf{y} = (y_1, \ldots, y_K)^T$$

for training input

$$\mathbf{x} = (x_1, \ldots, x_n)^T$$

is given.
Consider some error function

$$D(\boldsymbol{\theta})$$

that typically depends on training data but that depends on the parameters $\boldsymbol{\theta}$ only through the network output $\mathbf{t}$, e.g. a squared error function.

Training of Artificial Neural Networks

Suppose a *single* training observation

$$\mathbf{y} = (y_1, \ldots, y_K)^T$$

for training input

$$\mathbf{x} = (x_1, \ldots, x_n)^T$$

is given.

Consider some error function

$$D(\boldsymbol{\theta})$$

that typically depends on training data but that depends on the parameters $\boldsymbol{\theta}$ only through the network output $\mathbf{t}$, e.g. a squared error function. To minimize the $D(\boldsymbol{\theta})$ we will require to compute all partial derivatives

$$\frac{\partial D}{\partial \theta_{ji}^{(\ell-1)}}$$

for ℓ, j and i ("gradient" in TensorFlow).

Training of Artificial Neural Networks

Suppose a *single* training observation

$$\mathbf{y} = (y_1, \ldots, y_K)^T$$

for training input

$$\mathbf{x} = (x_1, \ldots, x_n)^T$$

is given.

Consider some error function

$$D(\boldsymbol{\theta})$$

that typically depends on training data but that depends on the parameters $\boldsymbol{\theta}$ only through the network output $\mathbf{t}$, e.g. a squared error function. To minimize the $D(\boldsymbol{\theta})$ we will require to compute all partial derivatives

$$\frac{\partial D}{\partial \theta_{ji}^{(\ell-1)}}$$

for ℓ, j and i ("gradient" in TensorFlow).

$D(\boldsymbol{\theta})$ may be non-convex and have local, non-global minima.

NN Training: partial derivatives backpropagate through network
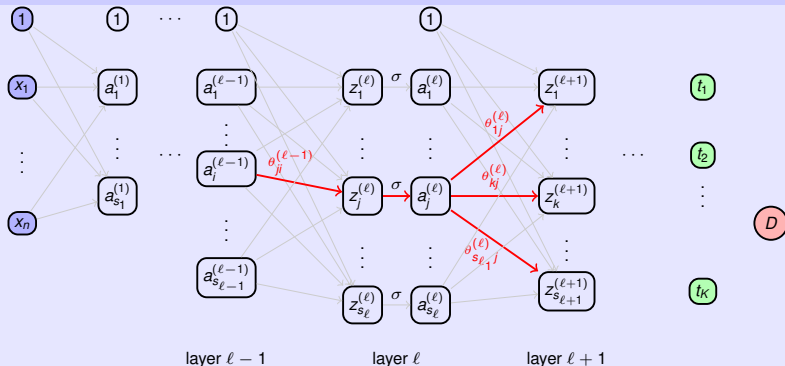
From

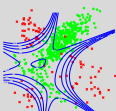$$z_j^{(\ell)} = \sum_r \theta_{jr}^{(\ell-1)} a_r^{(\ell-1)}$$

and using the multivariate chain rule that

we obain using the univariate chain rule that

$$\frac{\partial D}{\partial \theta_{ji}^{(\ell-1)}} = \frac{\partial D}{\partial z_j^{(\ell)}} a_i^{(\ell-1)}$$

$$\frac{\partial D}{\partial z_j^{(\ell)}} = \left(\frac{\partial D}{\partial z_1^{(\ell+1)}}, \cdots, \frac{\partial D}{\partial z_{s_{\ell+1}}^{(\ell+1)}} \right) \begin{pmatrix} \frac{\partial z_1^{(\ell+1)}}{\partial z_j^{(\ell)}} \\ \vdots \\ \frac{\partial z_{s_{\ell+1}}^{(\ell+1)}}{\partial z_j^{(\ell)}} \end{pmatrix}.$$
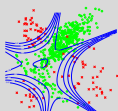
All derivates can be computed efficiently in one right-to-left pass ("backpropagation algorithm").
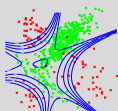TensorFlow does backprop automatically, for general models.

Artificial Neural Networks for Regression

Consider first a single training instance $\mathbf{x} \in R^n$ with a single output $\mathbf{y} \in R^K$.
(We simply average $D(\boldsymbol{\theta})$ over multiple training instances.)

Machine Learning

**Lars Gabriel, Felix
Becker, Mario Stanke**

Artificial Neural
Networks
Definition
Training (BackProp)
Activation Functions

Artificial Neural Networks for **Regression**

Consider first a single training instance $\mathbf{x} \in R^n$ with a single output $\mathbf{y} \in R^K$.
(We simply average $D(\boldsymbol{\theta})$ over multiple training instances.)

- Choose the identity function as output activation function $\mathbf{g}$, i.e. $\mathbf{t} = \mathbf{z}^{(L)}$ (unbounded).

Artificial Neural Networks for Regression

Consider first a single training instance $\mathbf{x} \in R^n$ with a single output $\mathbf{y} \in R^K$.
(We simply average $D(\boldsymbol{\theta})$ over multiple training instances.)

- Choose the identity function as output activation function $\mathbf{g}$, i.e. $\mathbf{t} = \mathbf{z}^{(L)}$ (unbounded).
- Choose the squared error function

$$D(\boldsymbol{\theta}) := \|\mathbf{t} - \mathbf{y}\|_2^2 = \sum_{k=1}^{K}(t_k - y_k)^2$$

Artificial Neural Networks for Classification

Again, consider first a single training instance. Here, let output $\mathbf{y} \in R^K$ be one-hot encoded such that

$$\mathbf{y} = \mathbf{e}_c = c\text{-th unit vector}$$

if $c \in \{1, 2, \ldots, K\}$ is the true class of the learning instance.

Again, consider first a single training instance. Here, let output $\mathbf{y} \in R^K$ be one-hot encoded such that

$$\mathbf{y} = \mathbf{e}_c = c\text{-th unit vector}$$

if $c \in \{1, 2, \ldots, K\}$ is the true class of the learning instance.

- Choose the softmax function as output activation function $\mathbf{g}$.

$$\mathbf{t} = \mathbf{g}(z_1, \ldots, z_K) = \frac{1}{\sum_{k=1}^{K} e^{z_k}} \begin{pmatrix} e^{z_1} \\ \vdots \\ e^{z_K} \end{pmatrix}$$

Artificial Neural Networks for Classification

Again, consider first a single training instance. Here, let output $\mathbf{y} \in R^K$ be one-hot encoded such that

$$\mathbf{y} = \mathbf{e}_c = c\text{-th unit vector}$$

if $c \in \{1, 2, \ldots, K\}$ is the true class of the learning instance.

- Choose the softmax function as output activation function $\mathbf{g}$.

$$\mathbf{t} = \mathbf{g}(z_1, \ldots, z_K) = \frac{1}{\sum_{k=1}^{K} e^{z_k}} \begin{pmatrix} e^{z_1} \\ \vdots \\ e^{z_K} \end{pmatrix}$$

  Properties:
  - $0 < t_k < 1$, $t_1 + \cdots + t_K = 1$
  - if $z_j \gg z_i$ for $i \neq j$, then $t_j \approx 1$ and $t_i \approx 0$ for $i \neq j$
  - softmax is a generalization of the sigmoid function to more than 2 classes

- If $K = 2$ (binary classification), $t_1 = 1 - t_2$ is often not stored. The same neural network function can then be achieved by setting the output size to 1 and using the logistic sigmoid function as activation.

Cross-entropy error function for multiclass problem

Consider the true class of the training instance to be a random variable $C \in \{1, \ldots, K\}$ with observation c and the output vector $\mathbf{t}$ of the net to be the parameters of a multinomial distribution for C. Then the likelihood is

$$L(\boldsymbol{\theta}) =$$

Cross-entropy error function for multiclass problem

Consider the true class of the training instance to be a random variable $C \in \{1, \ldots, K\}$ with observation c and the output vector $\mathbf{t}$ of the net to be the parameters of a multinomial distribution for C. Then the likelihood is

$$L(\boldsymbol{\theta}) = t_c$$

Cross-entropy error function for multiclass problem

Consider the true class of the training instance to be a random variable $C \in \{1, \ldots, K\}$ with observation c and the output vector $\mathbf{t}$ of the net to be the parameters of a multinomial distribution for C. Then the likelihood is

$$L(\boldsymbol{\theta}) = t_c$$

and we seek to minimize the negative log-likelihood

$$-\ln L(\boldsymbol{\theta}) = -\ln t_c = -\sum_{k=1}^{K} y_k \ln t_k$$

Cross-entropy error function for multiclass problem

Consider the true class of the training instance to be a random variable $C \in \{1, \ldots, K\}$ with observation c and the output vector $\mathbf{t}$ of the net to be the parameters of a multinomial distribution for C. Then the likelihood is

$$L(\boldsymbol{\theta}) = t_c$$

and we seek to minimize the negative log-likelihood

$$-\ln L(\boldsymbol{\theta}) = -\ln t_c = -\sum_{k=1}^{K} y_k \ln t_k$$

Define the cross-entropy error function

$$D(\boldsymbol{\theta}) = -\sum_{k=1}^{K} y_k \ln t_k.$$
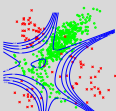
Artificial Neural Networks

Training set

Let

$$\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$$

be training inputs and

$$\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(m)}$$

be the corresponding training outputs / labels. Let

$$\mathbf{t}^{(i)} = h_\Theta(\mathbf{x}^{(i)}) \qquad (i = 1..m)$$

be the outputs of the NN.

Overfitting

If the model has many parameters, the training may result in a model that fits the training data 'too well', therefore does not generalize well and performs poorly on independent test data. Remedies:

1. Make model less complex, e.g. reduce number of parameters or change model class (e.g. lin. regresion over NN).

2. Regularize the model: Penalize certain parameter values independent of the data.

Overfitting

If the model has many parameters, the training may result in a model that fits the training data 'too well', therefore does not generalize well and performs poorly on independent test data. Remedies:

1. Make model less complex, e.g. reduce number of parameters or change model class (e.g. lin. regresion over NN).
2. Regularize the model: Penalize certain parameter values independent of the data.

Error function with regularization for neural networks

$$E(\boldsymbol{\theta}; X) = D(\boldsymbol{\theta}; X) + \lambda R(\boldsymbol{\theta})$$

Overfitting

If the model has many parameters, the training may result in a model that fits the training data 'too well', therefore does not generalize well and performs poorly on independent test data. Remedies:

1. Make model less complex, e.g. reduce number of parameters or change model class (e.g. lin. regresion over NN).
2. Regularize the model: Penalize certain parameter values independent of the data.

Error function with regularization for neural networks

$$E(\boldsymbol{\theta}; X) = D(\boldsymbol{\theta}; X) + \lambda R(\boldsymbol{\theta})$$

An common choice for the regularization term is the (scaled) L2 norm:

$$R(\boldsymbol{\theta}) = \frac{1}{m} \sum_{\ell=0}^{L-1} \sum_{i=1}^{s_\ell} \sum_{j=1}^{s_{\ell+1}} \left(\theta_{ji}^{(\ell)} \right)^2.$$

Note that the bias terms $\theta_{j0}^{(\ell)}$ are left out (not penalized).

Artificial Neural Networks

Error function with regularization for regression

$$E(\theta) = \frac{1}{m} \left\{ \sum_{i=1}^{m} \sum_{k=1}^{K} \left(t_k^{(i)} - y_k^{(i)} \right)^2 + \lambda \sum_{\ell=0}^{L-1} \sum_{i=1}^{s_\ell} \sum_{j=1}^{s_{\ell+1}} \left(\theta_{ji}^{(\ell)} \right)^2 \right\} \qquad (2)$$

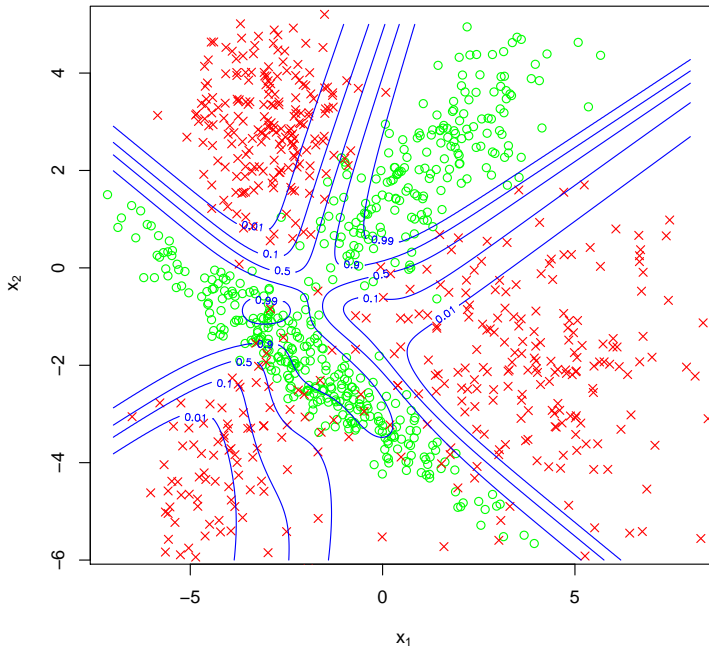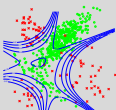# Artificial Neural Networks

Error function with regularization for regression

$$E(\boldsymbol{\theta}) = \frac{1}{m}\left\{\sum_{i=1}^{m}\sum_{k=1}^{K}\left(t_k^{(i)} - y_k^{(i)}\right)^2 + \lambda\sum_{\ell=0}^{L-1}\sum_{i=1}^{s_\ell}\sum_{j=1}^{s_{\ell+1}}\left(\theta_{ji}^{(\ell)}\right)^2\right\} \tag{2}$$

Error function with regularization for classification

$$E(\boldsymbol{\theta}) = \frac{1}{m}\left\{-\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \ln t_k^{(i)} + \lambda\sum_{\ell=0}^{L-1}\sum_{i=1}^{s_\ell}\sum_{j=1}^{s_{\ell+1}}\left(\theta_{ji}^{(\ell)}\right)^2\right\} \tag{3}$$

Machine Learning

Lars Gabriel, Felix Becker, Mario Stanke
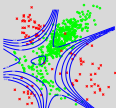
Artificial Neural Networks

Definition

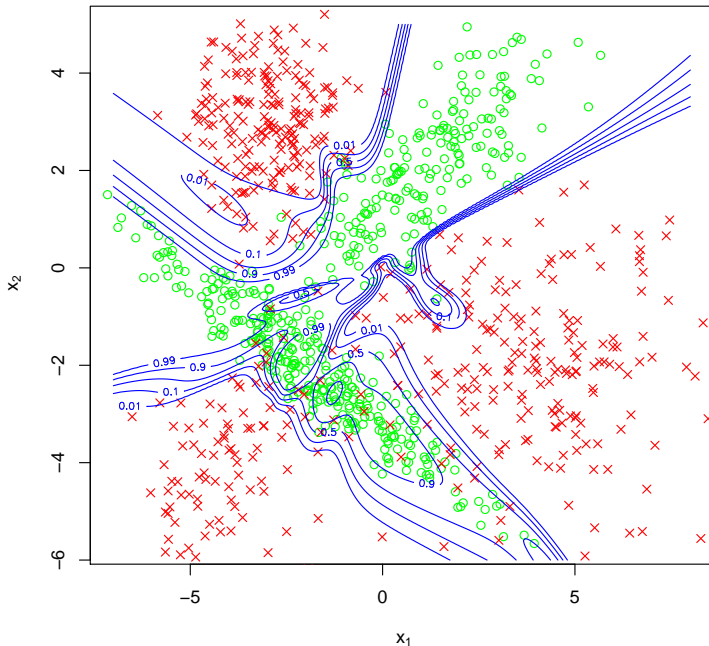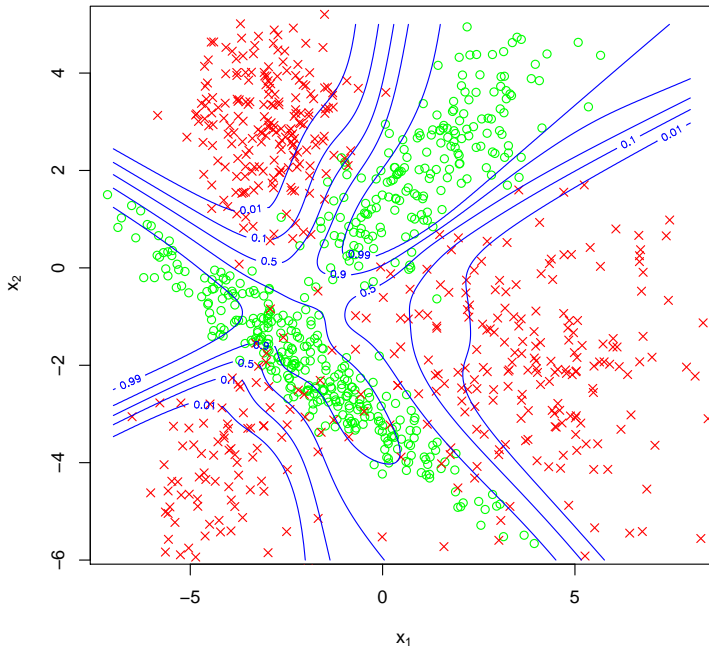Training (BackProp)

Activation Functions

1 hidden layer of size 5, $\lambda = 0$ (no regularization)

Machine Learning

Lars Gabriel, Felix Becker, Mario Stanke

Artificial Neural Networks
Definition
Training (BackProp)
Activation Functions

2 hidden layers of size 8 each, $\lambda = 0$ (no regularization)

2 hidden layers of size 20 each, $\lambda = 0$ (no regularization)

Machine Learning

Lars Gabriel, Felix Becker, Mario Stanke

Artificial Neural Networks

Definition

Training (BackProp)

Activation Functions

2 hidden layers of size 20 each, $\lambda = 0.1$ (with regularization)
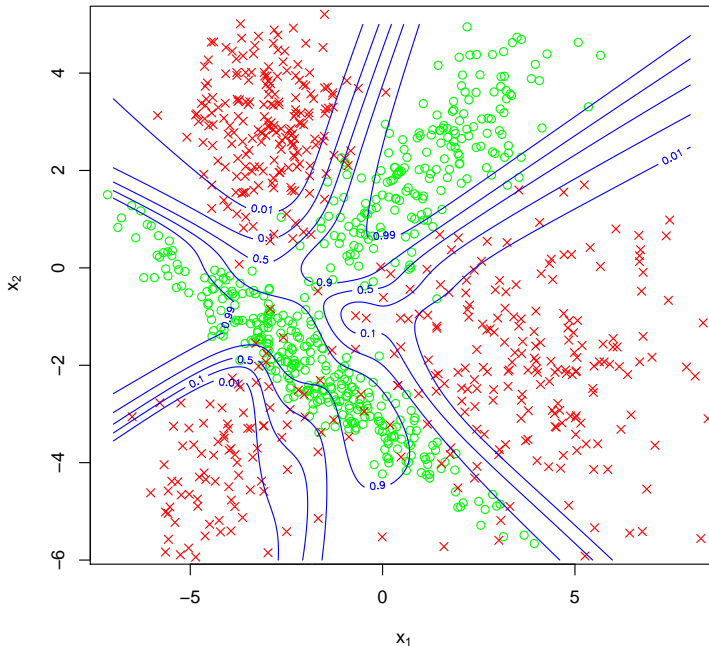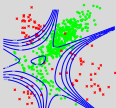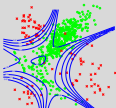
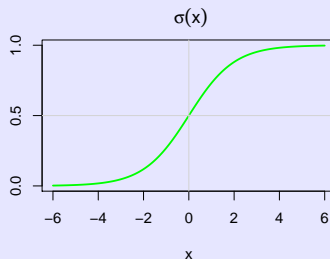# Posterior probability of class (theoretical optimum if distr. was known)
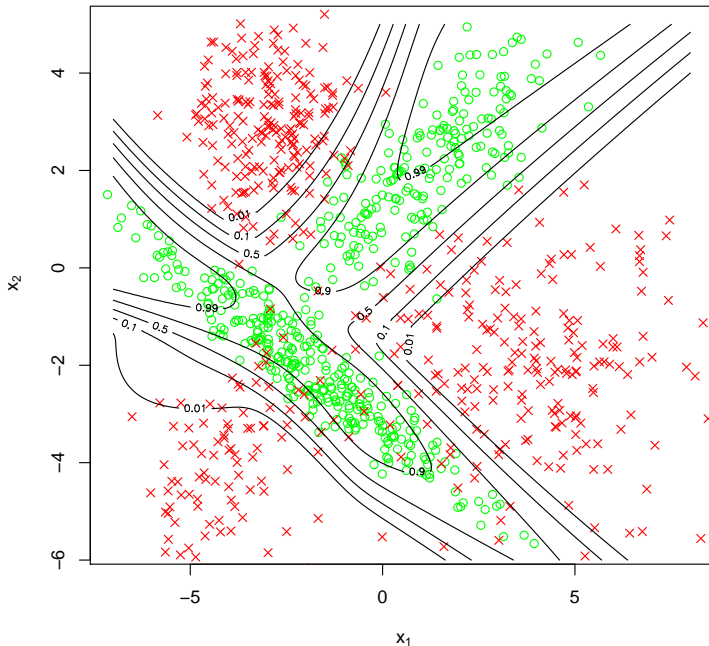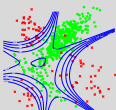
Activation Functions

Logistic sigmoid, `tf.nn.sigmoid`

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
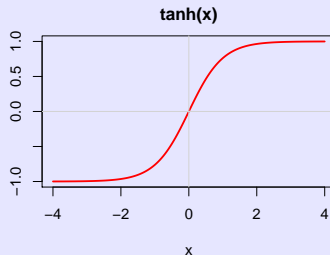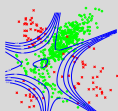
$$\sigma' = \sigma(1 - \sigma)$$

σ(x)

Tangens hyperbolicus

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$= 2\sigma(2x) - 1$$

$$\tanh'(x) = 1 - (\tanh(x))^2$$

tanh(x)

Machine Learning

Lars Gabriel, Felix Becker, Mario Stanke

Artificial Neural Networks
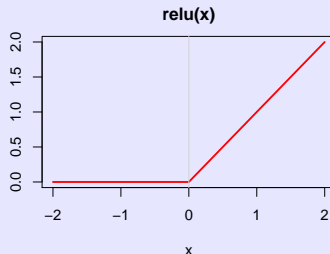Definition
Training (BackProp)
Activation Functions

Rectified linear unit (ReLU), `tf.nn.relu`

$$\text{relu}(x) = x^+ = \max\{0, x\}$$

$$\text{relu}'(x) = \begin{cases} 1 & \text{, if } x > 0 \\ 0 & \text{, if } x < 0 \end{cases}$$

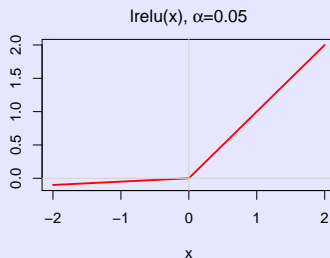$$\text{relu}'(0) := 1 \quad \text{(arbitrary)}$$

relu(x)

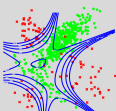## Leaky rectified linear unit (Leaky ReLU), `tf.nn.leaky_relu`

$$\text{lrelu}(x) = \max\{\alpha x, x\}$$

$$\text{lrelu}'(x) = \begin{cases} 1 & \text{, if } x > 0 \\ \alpha & \text{, if } x < 0 \end{cases}$$

$$\text{lrelu}'(0) := 1 \quad \text{(arbitrary)}$$

$$\alpha = 0.01$$

lrelu(x), α=0.05

Machine Learning

Lars Gabriel, Felix Becker, Mario Stanke

Artificial Neural Networks
Definition
Training (BackProp)
Activation Functions

Activation Functions

Vanishing gradients

- vanishing gradient problem:
    - during backprop in each layer, a derivative of the activation function is multiplied
    - for NNs with many layers (deep networks) this can result in gradients that are practically zero ("vanishing"), in particular for the sigmoid function

- ReLU
    - most frequently chosen
    - introduced in 2010

      (Nair and Hinton, "Rectified linear units improve restricted Boltzmann machines", *Proc. ICML*, 2010)

    - The derivative is either 0 or 1. Can lead to sparse gradients.