HST 951 Biomedical Decision Support

Artificial Neural Networks

Stephan Dreiseitl

Upper Austria University of Applied Sciences
Dept. of Software Engineering
Overview

• Motivation
• Linear classifiers/perceptrons
• Multi-layer networks
• Overfitting avoidance
• Bayesian perspective
Motivation

Human brain good at pattern recognition:

- can handle noisy and incomplete data
- can interpolate and extrapolate data
Motivation

Human brain good at pattern recognition:
- can handle noisy and incomplete data
- can interpolate and extrapolate data

Take inspiration from brain hardware:
- parallel processing
- distributed representation
Motivation

Human brain good at pattern recognition:
- can handle noisy and incomplete data
- can interpolate and extrapolate data

Take inspiration from brain hardware:
- parallel processing
- distributed representation

Expect to achieve similar feats:
- fault tolerance
- good generalization capability
Artificial neurons

Neurons in the brain
Artificial neurons

Neurons in the brain

Mathematical abstraction
Activation functions

The function that transforms the weighted sum of inputs \( \sum_{i=1}^{n} w_{i}x_{i} \) to an activation can take the following forms:

- **threshold function**
- **linear function**
- **logistic function**
Hyperplanes

A vector \( w = (w_1, \ldots, w_n) \) separates \( \mathbb{R}^n \) into three parts:

\[
\begin{align*}
  w_1 x_1 + \cdots + w_n x_n &> 0 \\
  w_1 x_1 + \cdots + w_n x_n &= 0 \\
  w_1 x_1 + \cdots + w_n x_n &< 0
\end{align*}
\]

Use dot product notation: \( w \cdot x = w_1 x_1 + \cdots + w_n x_n \)
A vector $w = (w_1, \ldots, w_n)$ separates $\mathbb{R}^n$ into three parts:

\begin{align*}
w_1 x_1 + \cdots + w_n x_n &> 0 \\
w_1 x_1 + \cdots + w_n x_n &= 0 \\
w_1 x_1 + \cdots + w_n x_n &< 0
\end{align*}

Use dot product notation: $w \cdot x = w_1 x_1 + \cdots + w_n x_n$

The value $w \cdot x$ is proportional to distance of $x$ to hyperplane $H = \{x \mid w \cdot x + w_0 = 0\}$

Notational convention: let $w := (w, w_0)$ and $x := (x, 1)$
Objective: Given $x_1, \ldots, x_m \in \mathbb{R}^n$ and class labels $t_1, \ldots, t_m \in \{-1, +1\}$, find hyperplane s.t. all $x_i$ with $t_i = -1$ lie on one side, and all $x_i$ with $t_i = +1$ on other side:

$$w \cdot x_i = \begin{cases} > 0 & \text{if } t_i = +1 \\ < 0 & \text{if } t_i = -1 \end{cases}$$
Linear separability

Objective: Given \( x_1, \ldots, x_m \in \mathbb{R}^n \) and class labels \( t_1, \ldots, t_m \in \{-1, +1\} \), find hyperplane s.t. all \( x_i \) with \( t_i = -1 \) lie on one side, and all \( x_i \) with \( t_i = +1 \) on other side:

\[
w \cdot x_i = \begin{cases} > 0 & \text{if } t_i = +1 \\ < 0 & \text{if } t_i = -1 \end{cases}
\]

Possible error measure is

\[
E(w) = - \sum_{i \text{ miscl.}} t_i (w \cdot x_i)
\]
Illustration of linear separability

Let class labels of $\circ$ be $+1$, and labels of $\times$ be $-1$

$\{x \mid w \cdot x > 0\}$

$\{x \mid w \cdot x < 0\}$
Illustration of linear separability

Let class labels of \( \circ \) be \( +1 \), and labels of \( x \) be \( -1 \)

\[
\{ x \mid w \cdot x > 0 \}
\]

\[
\{ x \mid w \cdot x < 0 \}
\]

Error \( E(w) = - \sum_{i \text{ miscl.}} t_i(w \cdot x_i) \geq 0 \) proportional to distances of misclassified cases to decision boundary
Error minimization by gradient descent

Need algorithm to minimize \( E(w) = - \sum_{i \text{ miscl.}} t_i (w \cdot x_i) \)

*Gradient* \( \nabla_x f \): vector of partial derivatives \( (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}) \)

Negative gradient is direction of steepest descent

Length of gradient determined by steepness of surface
Error minimization by gradient descent

Need algorithm to minimize \( E(w) = - \sum_{i \text{ miscl.}} t_i(w \cdot x_i) \)

*Gradient* \( \nabla_x f \): vector of partial derivatives \( \left( \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right) \)

Negative gradient is direction of steepest descent

Length of gradient determined by steepness of surface

*Gradient descent*: always go downhill
Learning linear discriminators

Gradient descent finds minimum of $E$ by iterating

$$w_{k+1} = w_k - \eta \text{grad}_w E(w_k)$$

*Learning rate* $\eta$: How far to go each iteration step
Learning linear discriminators

Gradient descent finds minimum of $E$ by iterating

$$w_{k+1} = w_k - \eta \text{grad}_w E(w_k)$$

*Learning rate* $\eta$: How far to go each iteration step

For $E(w) = -\sum_i \text{miscl. } t_i (w \cdot x_i)$, gradient is

$$\text{grad}_w E = -\sum_{i \text{ miscl.}} t_i x_i$$
Learning linear discriminators

Gradient descent finds minimum of $E$ by iterating

$$w_{k+1} = w_k - \eta \text{grad}_w E(w_k)$$

*Learning rate* $\eta$: How far to go each iteration step

For $E(w) = -\sum_{i \text{ miscl.}} t_i (w \cdot x_i)$, gradient is

$$\text{grad}_w E = - \sum_{i \text{ miscl.}} t_i x_i$$

“Batch” version: $w_{k+1} = w_k + \eta \sum_{i \text{ miscl.}} t_i x_i$

“Online” version: $w_{k+1} = w_k + \eta t_i x_i$
Perceptron learning

*Perceptron* is model of single neuron

For threshold, linear, and logistic activation function $f$, perceptron $f(w \cdot x)$ implements linear discriminator

Perceptron training with $w_{k+1} = w_k + \eta \sum_{i \text{ miscl.}} t_i x_i$

guaranteed to converge for linearly separable data sets
Perceptron learning

*Perceptron* is model of single neuron

For threshold, linear, and logistic activation function $f$, perceptron $f(w \cdot x)$ implements linear discriminator

Perceptron training with $w_{k+1} = w_k + \eta \sum_{i \text{ miscl.}} t_i x_i$

guaranteed to converge for linearly separable data sets

perceptron output for 2-dim. inputs and logistic activation
Perceptron learning example

Updating $w_k$ until convergence (20 steps)

initial: $w_1$

final: $w_{21}$
Why consider multilayer perceptrons?
Why consider multilayer perceptrons?
Multilayer perceptrons

Can represent arbitrary decision regions
Can be trained similar to perceptrons
Adjustable parameters: weights $w$ in the network
Some mathematical background

Recall that goal is to find $t$ for new $x$, but problem is not deterministic.

Assume data set $D = \{(x_i, t_i)\}$ of i.i.d. samples drawn from joint distribution with density $p(x, t)$.
Some mathematical background

Recall that goal is to find $t$ for new $x$, but problem is not deterministic

Assume data set $D = \{(x_i, t_i)\}$ of i.i.d. samples drawn from joint distribution with density $p(x, t)$

Idea: Given $D$, find maximum likelihood parameters $w$ to model dependence of $t$ on $x$, i.e. $p(t|x)$

For best $w$, maximize likelihood function

$$L(w) = \prod_{i=1}^{m} p(x_i, t_i | w) = \prod_{i=1}^{m} p(t_i | x_i, w) p(x_i)$$
Maximizing $L(w)$ equivalent to minimizing $-\log L(w)$:

$$-\log L(w) = -\log \prod_{i=1}^{m} p(t_i|x_i, w)p(x_i)$$

$$= -\sum_{i=1}^{m} \log p(t_i|x_i, w) - \sum_{i=1}^{m} p(x_i)$$
Maximizing $L(w)$ equivalent to minimizing $-\log L(w)$:

$$-\log L(w) = -\log \prod_{i=1}^{m} p(t_i|x_i, w)p(x_i)$$

$$= -\sum_{i=1}^{m} \log p(t_i|x_i, w) - \sum_{i=1}^{m} p(x_i)$$

Drop second sum: does not depend on $w$

Two cases for functional form of $p(t_i|x_i, w)$: regression and classification
Likelihood for regression

For regression, targets $t$ are real values

Assume that network outputs $y(x_i, w)$ approximate targets $t_i$ up to additive Gaussian noise:

$$p(t_i|x_i, w) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left( -\frac{(y(x_i, w) - t_i)^2}{2\sigma^2} \right)$$
Likelihood for regression

For regression, targets $t$ are real values

Assume that network outputs $y(x_i, w)$ approximate targets $t_i$ up to additive Gaussian noise:

$$p(t_i|x_i, w) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y(x_i, w) - t_i)^2}{2\sigma^2}\right)$$

With this model,

$$-\sum_{i=1}^{m} \log p(t_i|x_i, w) = \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y(x_i, w) - t_i)^2 + \frac{m}{2} \log(2\pi\sigma^2)$$
Thus, maximum likelihood estimate of network weights obtained by minimizing error function

\[ E(w) = \frac{1}{2} \sum_{i=1}^{m} (y(x_i, w) - t_i)^2 \]  

(sum-of-squares error)

Other error functions can be derived if noise distribution is different from Gaussian.
Likelihood for classification

For classification, targets $t$ are class labels.

Assume dichotomous case with $t_i \in \{0, 1\}$, and that network outputs $y(x_i, w)$ are $p(1|x_i)$.

Then $p(0|x_i) = 1 - y(x_i, w)$, and together

$$p(t_i|x, w) = y(x_i, w)^{t_i} (1 - y(x_i, w))^{1-t_i}$$
Likelihood for classification

For classification, targets $t$ are class labels.

Assume dichotomous case with $t_i \in \{0, 1\}$, and that network outputs $y(x_i, w)$ are $p(1|x_i)$.

Then $p(0|x_i) = 1 - y(x_i, w)$, and together

$$p(t_i|x, w) = y(x_i, w)^{t_i} (1 - y(x_i, w))^{1-t_i}$$

Maximum likelihood approach requires minimization of

$$-\sum_{i=1}^{m} \log p(t_i|x_i, w) = -\sum_{i=1}^{m} \log \left( y(x_i, w)^{t_i} (1-y(x_i, w))^{1-t_i} \right)$$
Likelihood for classification (cont.)

Error function for maximum likelihood estimation of classification parameters is therefore

\[ E(w) = - \sum_{i=1}^{n} t_i \log y(x_i, w) + (1 - t_i) \log (1 - y(x_i, w)) \]

(cross-entropy error)

Variations possible more multiple classes
Backpropagation algorithm

- minimize error function by gradient descent:
  \[ w_{k+1} = w_k - \eta \text{grad}_w E \]
- Iterative gradient calculation by propagating error signals
- Mostly of historical interest: possible to implement in parallel structure
Algorithm convergence depends on learning rate $\eta$
Algorithm convergence depends on learning rate $\eta$

Better: use more advanced minimization algorithms that take second-order information into account
Comparison of network structure

Classification

- cross-entropy error
- sigmoidal neuron
- sigmoidal neurons
- linear neurons

Regression

- sum-of-squares error
- linear neuron
- sigmoidal neurons
- linear neurons
mean of $p(t|x)$
ANN output for classification

\[ P(t = 1|x) \]
Judging network performance

Given these data points, which line is a better representation?

Impossible to answer without additional information (cf. bias-variance dilemma in statistics)
Judging network performance

Given these data points, which line is a better representation?

Impossible to answer without additional information (cf. bias-variance dilemma in statistics)
Improving generalization capability

Goal of network training: minimize error on unseen cases drawn from same distribution as training set ("generalize well", "avoid overtraining")

Need test set to estimate generalization error
Improving generalization capability

Goal of network training: minimize error on unseen cases drawn from same distribution as training set (“generalize well”, “avoid overtraining”)

Need test set to estimate generalization error

Avoid overtraining by

- limiting network topology
- early stopping
- weight decay
Limit network topology

Idea: fewer weights mean less flexibility

nhidden = 1

nhidden = 10

nhidden = 3
Early stopping

Idea: stop training when information (but not noise) is modeled

Need validation set to determine when to stop training
Early stopping (cont.)

training until convergence

with early stopping
Weight decay

Idea: Control smoothness of network output by controlling size of weights in network

Add term $\alpha \| w \|^2$ to error function

random $w \sim N(0, 1)$

random $w \sim N(0, 10)$
Weight decay

\[ \alpha = 0.001 \]

\[ \alpha = 0.1 \]

\[ \alpha = 0.00001 \]
Bayesian perspective

Error function minimization corresponds to maximum likelihood (ML) estimate: find single best solution $w_{ML}$

This can lead to overtraining
Bayesian perspective

Error function minimization corresponds to maximum likelihood (ML) estimate: find single best solution $w_{ML}$

This can lead to overtraining

Bayesian approach: consider weight posterior distribution $p(w|D)$

Take mode, or average over samples of posterior
Recall that posterior distribution is likelihood $\times$ prior:

$$p(w|D) = \alpha p(D|w)p(w)$$
Recall that posterior distribution is likelihood $\times$ prior:

$$p(w|D) = \alpha \ p(D|w)p(w)$$

Two approaches to calculating $p(w|D)$:

- sampling
- Gaussian approximation
Sampling from $p(w|D)$

prior

likelihood
Sampling from $p(w|D)$ (cont.)

prior $\times$ likelihood = posterior
Gaussian approximation to $p(w|D)$

Find maximum $w_{\text{MAP}}$ of $p(w|D)$

Approximate $p(w|D)$ by Gaussian around $w_{\text{MAP}}$

Fit mean ($w_{\text{MAP}}$) and variance (by curvature of $p(w|D)$)
Gaussian approximation to $p(w|D)$ (cont.)

Find $w_{\text{MAP}}$ by maximizing $p(w|D)$

With $p(w|D) \propto p(D|w)p(w)$, get

$$- \log p(w|D) = - \log p(D|w) - \log p(w)$$

Minimizing first term finds ML solution

For $p(w)$ zero-mean Gaussian, $- \log p(w) = \alpha \|w\|^2$

Therefore, adding weight decay to ANN training amounts to finding MAP solution
Find $w_{\text{MAP}}$ by maximizing $p(w|D)$

With $p(w|D) \propto p(D|w)p(w)$, get

$$-\log p(w|D) = -\log p(D|w) - \log p(w)$$

Minimizing first term finds ML solution

For $p(w)$ zero-mean Gaussian, $-\log p(w) = \alpha \|w\|^2$

Therefore, adding weight decay to ANN training amounts to finding MAP solution
Advantage of Bayesian approach

Error bars for regression, averaged estimates for classification
ANN models inspired by functionality of the brain
ANNs are nonlinear data models
ANN training is minimization of error function
Different error functions for classification and regression
Training goal is to generalize well (avoid overtraining)
Distinguish between ML solution and Bayesian approach