DeepLearning on FPGAs

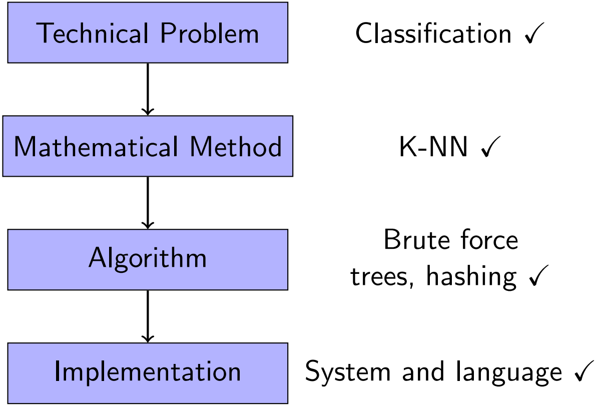
Introduction to Artificial Neural Networks

Sebastian Buschjäger

Technische Universität Dortmund - Fakultät Informatik - Lehrstuhl 8

November 2, 2016

Recap: Computer Science Approach



Recap: Data Mining (1)

Important concepts:

- **Classification** is one data mining task
- **Training data** is used to define and solve the task
- **A Method** is a general approach / idea to solve a task
- **A algorithm** is a way to realise a method
- **A model** forms the extracted knowledge from data
- **Accuracy** measures the model quality given the data

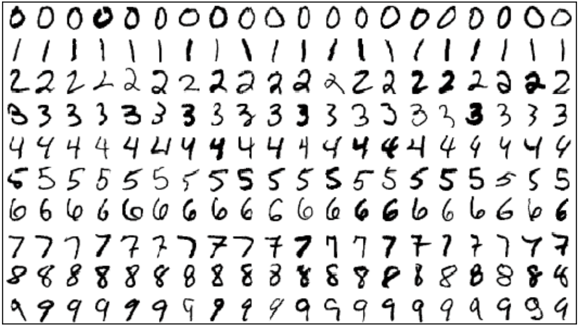
Recap: Data Mining (1)

Important concepts:

- **Classification** is one data mining task
- **Training data** is used to define and solve the task
- **A Method** is a general approach / idea to solve a task
- **A algorithm** is a way to realise a method
- **A model** forms the extracted knowledge from data
- **Accuracy** measures the model quality given the data

K-NN: Look at the k nearest neighbours of $\vec{x}^*$ and use most common label as prediction
Homework: How good was your prediction?

The MNIST dataset

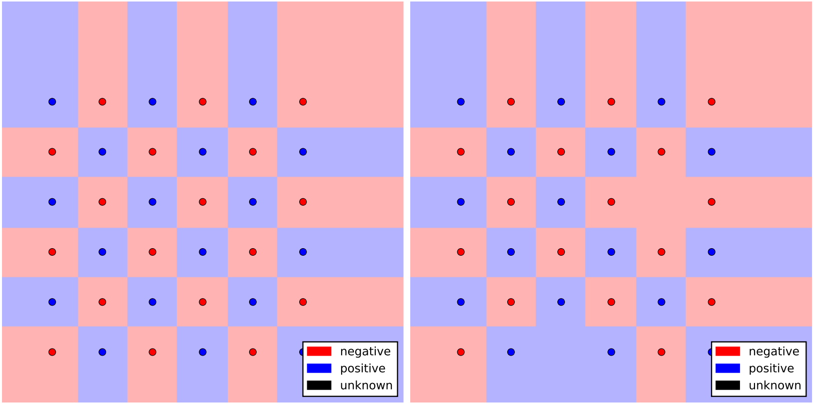


Common error rates[1] without pre-procssing:
K-NN: 2.83 % - SVM: 1.4 % - CNN: ~ 0.4 %

Big Note: Dataset already centered and scaled

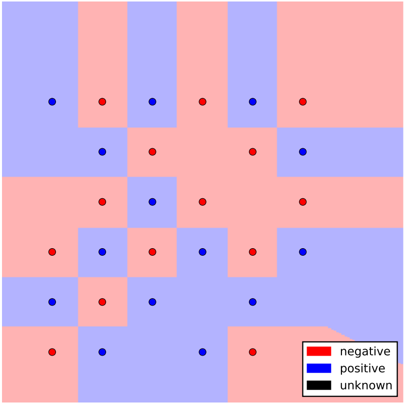
[1]See: http://yann.lecun.com/exdb/mnist/

K-NN: Example (1)

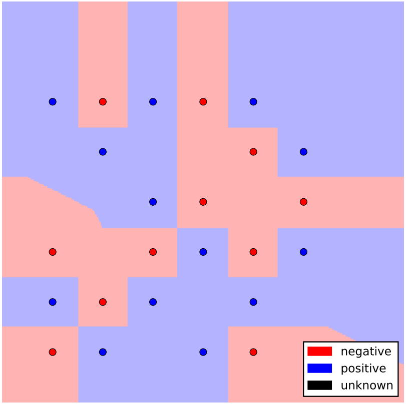


$k = 1$, all points available $k = 1$, 2 points missing

K-NN: Example (2)



$k = 1$, 8 points missing



$k = 1$, 12 points missing

Feature Engineering and Feature Dimensions

Note: K-NN fails to recognize patterns in incomplete data

Feature Engineering and Feature Dimensions

Note: K-NN fails to recognize patterns in incomplete data
Fact 1: State space grows exponentially with increasing dimension. Example $\mathcal{X} = \{1, 2, \ldots, 10\}$

- **For:** $\mathcal{X}^1$, there are 10 different observations
- **For:** $\mathcal{X}^2$, there are $10^2 = 100$ different observations
- **For:** $\mathcal{X}^3$, there are $10^3 = 1000$ different observations ...

Feature Engineering and Feature Dimensions

Note: K-NN fails to recognize patterns in incomplete data
Fact 1: State space grows exponentially with increasing dimension. Example $\mathcal{X} = \{1, 2, \ldots, 10\}$

- **For:** $\mathcal{X}^1$, there are 10 different observations
- **For:** $\mathcal{X}^2$, there are $10^2 = 100$ different observations
- **For:** $\mathcal{X}^3$, there are $10^3 = 1000$ different observations ...

Fact 2: Training data is generated by a noisy real-world process

- We usually have no influence on the type of training data
- We usually cannot interfere with the real-world process

Feature Engineering and Feature Dimensions

Note: K-NN fails to recognize patterns in incomplete data
Fact 1: State space grows exponentially with increasing dimension. Example $\mathcal{X} = \{1, 2, \ldots, 10\}$

- **For:** $\mathcal{X}^1$, there are 10 different observations
- **For:** $\mathcal{X}^2$, there are $10^2 = 100$ different observations
- **For:** $\mathcal{X}^3$, there are $10^3 = 1000$ different observations $\ldots$

Fact 2: Training data is generated by a noisy real-world process

- We usually have no influence on the type of training data
- We usually cannot interfere with the real-world process

Thus: Training data should be considered incomplete and noisy

Feature Engineering and Feature Dimensions

Fact: There is no free lunch (**Wolpert, 1996**)

- Every method has is advantages and disadvantages
- Most methods are able to perfectly learn a given toy data set
- Problem occurs with noise, outlier and generalisation

Feature Engineering and Feature Dimensions

Fact: There is no free lunch (**Wolpert, 1996**)

- Every method has is advantages and disadvantages
- Most methods are able to perfectly learn a given toy data set
- Problem occurs with noise, outlier and generalisation

Conclusion: All methods are equally good or bad
But: Some methods prefer certain representations

Feature Engineering and Feature Dimensions

Fact: There is no free lunch (**Wolpert, 1996**)

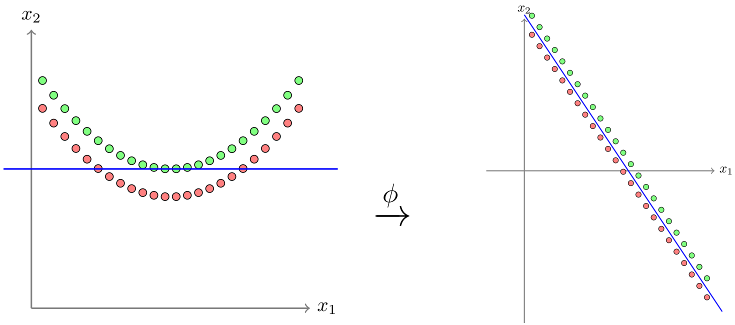
- Every method has is advantages and disadvantages
- Most methods are able to perfectly learn a given toy data set
- Problem occurs with noise, outlier and generalisation

Conclusion: All methods are equally good or bad
But: Some methods prefer certain representations

Feature Engineering: Finding the right representation for data

- Reduce dimension? Increase dimension?
- Add additional information? Regularities?
- Transform data completely?

Feature Engineering: Example



Raw data without transformation.
Linear model is a bad choice.
Parabolic model would be better.

Data transformed with
$\phi(x_1, x_2) = (x_1, x_2 - 0.3 \cdot x_1^2)$.
Now linear model fits the problem.

Feature Engineering: Conclusion

Conclusion: Good features are crucial for good results!
Question: How to get good features?

Feature Engineering: Conclusion

Conclusion: Good features are crucial for good results!
Question: How to get good features?

1. **By hand:** Domain experts and data miner examine the data and try different features based on common knowledge.
2. **Semi supervised:** Data miner examines the data and tries different similarity functions and classes of methods
3. **Unsupervised:** Data miner only encodes some assumptions about regularities into the method.

Feature Engineering: Conclusion

Conclusion: Good features are crucial for good results!
Question: How to get good features?

1. **By hand:** Domain experts and data miner examine the data and try different features based on common knowledge.
2. **Semi supervised:** Data miner examines the data and tries different similarity functions and classes of methods
3. **Unsupervised:** Data miner only encodes some assumptions about regularities into the method.

Note 1: Hand-crafted features give us insight about the process
Note 2: Semi/unsupervised features give us insight about the data
Our focus: Unsupervised feature extraction.

What is Deep Learning?

Deep Learning Basics

So... What is Deep Learning?
Well... its currently one of the big things in AI!

- **Since 2010:** DeepMind learns and plays old Atari games
- **Since 2012:** Google is able to find cats in youtube videos
- **December 2014:** Near real-time translation in Skype
- **October 2015:** AlphaGo beats the European Go champion
- **October 2015:** Tesla deploys Autopilot in their cars
- **March 2016:** AlphaGo beats the Go Worldchampion
- **June 2016:** Facebook introduces DeepText
- ...

Deep Learning: Example

Deep Learning Basics

Deep Learning: is a branch of Machine Learning dealing with

- (Deep) Artificial Neural Networks (ANN)
- High Level Feature Processing
- Fast Implementations

Deep Learning Basics

Deep Learning: is a branch of Machine Learning dealing with

- (Deep) Artificial Neural Networks (ANN)
- High Level Feature Processing
- Fast Implementations

ANNs are well known! So what's new about it?

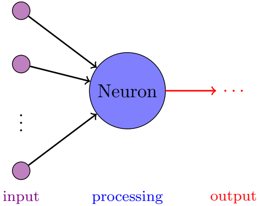
- We have more data and more computation power
- We have a better understanding of optimization
- We use a more engineering-style approach

Our focus now: Artificial Neural Networks

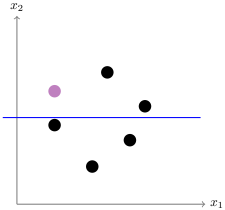
Artificial Neural Networks: Single Neuron

Simple case: Let $\vec{x} \in \mathbb{B}^d$

Biology's view:



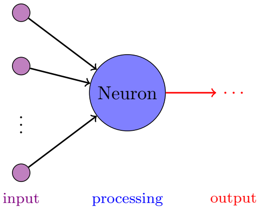
Geometrical view:



Artificial Neural Networks: Single Neuron

Simple case: Let $\vec{x} \in \mathbb{B}^d$

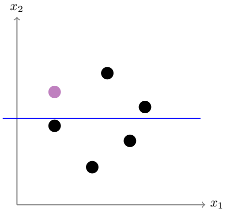
Biology's view:



"Fire" if input signals reach threshold:

$$f(\vec{x}) = \begin{cases} +1 & \text{if } \sum_{i=1}^{d} x_i \geq b \\ 0 & \text{else} \end{cases}$$

Geometrical view:



Predict class depending on side of line (count):

$$f(\vec{x}) = \begin{cases} +1 & \text{if } \sum_{i=1}^{d} x_i \geq b \\ 0 & \text{else} \end{cases}$$

Artificial Neural Networks: Single Neuron

Note: We basically count the number of positive inputs

1943: McCulloch-Pitts Neuron:

- Simple linear model with binary input and output
- Can model boolean OR with $b = 1$
- Can model boolean AND with $b = d$
- Simple extension also allows boolean NOT

Artificial Neural Networks: Single Neuron

Note: We basically count the number of positive inputs

1943: McCulloch-Pitts Neuron:

- Simple linear model with binary input and output
- Can model boolean OR with $b = 1$
- Can model boolean AND with $b = d$
- Simple extension also allows boolean NOT

Thus: A network of McCulloch-Pitts neurons can simulate every boolean function (functional complete)

Artificial Neural Networks: Single Neuron

Note: We basically count the number of positive inputs

1943: McCulloch-Pitts Neuron:

- Simple linear model with binary input and output
- Can model boolean OR with $b = 1$
- Can model boolean AND with $b = d$
- Simple extension also allows boolean NOT

Thus: A network of McCulloch-Pitts neurons can simulate every boolean function (functional complete)

Remark: That does not help with classification, thus

- **Rosenblatt 1958:** Use weights $w_i \in \mathbb{R}$ for every input $x_i \in \mathbb{B}$
- **Minksy-Papert 1959:** Allow real valued inputs $x_i \in \mathbb{R}$

Artificial Neural Networks: Perceptron

A perceptron is a linear classifier $f \colon \mathbb{R}^d \to \{0, 1\}$ with

$$\widehat{f}(\vec{x}) = \begin{cases} +1 & \text{if } \sum_{i=1}^{d} w_i \cdot x_i \geq b \\ 0 & \text{else} \end{cases}$$

Artificial Neural Networks: Perceptron

A perceptron is a linear classifier $f\colon \mathbb{R}^d \to \{0,1\}$ with

$$\widehat{f}(\vec{x}) = \begin{cases} +1 & \text{if } \sum_{i=1}^{d} w_i \cdot x_i \geq b \\ 0 & \text{else} \end{cases}$$

Linear function in $d = 2$: $y = mx + \tilde{b}$
Perceptron: $w_1 \cdot x_1 + w_2 \cdot x_2 \geq b \Leftrightarrow x_2 = \frac{b}{w_2} - \frac{w_1}{w_2} x_1$
Obviously: A perceptron is a hyperplane in d dimensions

Artificial Neural Networks: Perceptron

A perceptron is a linear classifier $f \colon \mathbb{R}^d \to \{0, 1\}$ with

$$\widehat{f}(\vec{x}) = \begin{cases} +1 & \text{if } \sum_{i=1}^{d} w_i \cdot x_i \geq b \\ 0 & \text{else} \end{cases}$$

Linear function in $d = 2$: $y = mx + \tilde{b}$
Perceptron: $w_1 \cdot x_1 + w_2 \cdot x_2 \geq b \Leftrightarrow x_2 = \frac{b}{w_2} - \frac{w_1}{w_2} x_1$
Obviously: A perceptron is a hyperplane in d dimensions

Note: $\vec{w} = (w_1, \ldots, w_d, b)^T$ are the parameters of a perceptron
Notation: Given $\vec{x}$ we add a 1 to the end of it $\vec{x} = (x_1, \ldots, x_d, 1)^T$

$$\textbf{Then}: \ \widehat{f}(\vec{x}) = \begin{cases} +1 & \text{if } \vec{x} \cdot \vec{w}^T \geq 0 \\ 0 & \text{else} \end{cases}$$

ANN: Perceptron Learning

Note: A perceptron assumes that the data is linear separable

ANN: Perceptron Learning

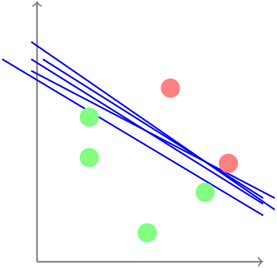
Note: A perceptron assumes that the data is linear separable
Big Note: This is an assumption and not necessarily true!

ANN: Perceptron Learning

Note: A perceptron assumes that the data is linear separable
Big Note: This is an assumption and not necessarily true!
But: In case of linear separability, there are many "good" $\vec{w}$

ANN: Perceptron Learning

Note: A perceptron assumes that the data is linear separable
Big Note: This is an assumption and not necessarily true!
But: In case of linear separability, there are many "good" $\vec{w}$



Note: We are happy with **one** separative vector $\vec{w}$

ANN: Perceptron Learning

Question: How do we get the weights $\vec{w}$?

ANN: Perceptron Learning

Question: How do we get the weights $\vec{w}$?
Observation: We look at $\vec{x} \cdot \vec{w}^T \geq 0$

- if output was 0 but should have been 1 increment weights
- if output was 1 but should have been 0 decrement weights
- if output was correct, don't change weights

ANN: Perceptron Learning

Question: How do we get the weights $\vec{w}$?

Observation: We look at $\vec{x} \cdot \vec{w}^T \geq 0$

- if output was 0 but should have been 1 increment weights
- if output was 1 but should have been 0 decrement weights
- if output was correct, don't change weights

$\vec{w} = rand(1, \ldots, d+1)$
while ERROR **do**
  for $(\vec{x}_i, y_i) \in \mathcal{D}$ **do**
    $\vec{w} = \vec{w} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}(\vec{x}_i))$
  end for
end while

ANN: Perceptron Learning

Question: How do we get the weights $\vec{w}$?
Observation: We look at $\vec{x} \cdot \vec{w}^T \geq 0$

- if output was 0 but should have been 1 increment weights
- if output was 1 but should have been 0 decrement weights
- if output was correct, don't change weights

$\vec{w} = rand(1, \ldots, d+1)$
while ERROR **do**
    for $(\vec{x}_i, y_i) \in \mathcal{D}$ **do**
        $\vec{w} = \vec{w} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}(\vec{x}_i))$
    end for
end while

Note: $\alpha \in \mathbb{R}_{>0}$ is a stepsize / learning rate

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$

Wrong classification:

- **Case 1:** $y_i - \widehat{f}_{old}(\vec{x}_i) = 1 \Rightarrow y_i = 1, \widehat{f}_{old}(\vec{x}_i) = 0$

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$

Wrong classification:

- **Case 1:** $y_i - \widehat{f}_{old}(\vec{x}_i) = 1 \Rightarrow y_i = 1, \widehat{f}_{old}(\vec{x}_i) = 0$

$$\widehat{f}_{new}(\vec{x}_i) = \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} + \alpha \cdot 1 \cdot \vec{x}_i)^T$$

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$

Wrong classification:

- **Case 1:** $y_i - \widehat{f}_{old}(\vec{x}_i) = 1 \Rightarrow y_i = 1, \widehat{f}_{old}(\vec{x}_i) = 0$

$$\begin{aligned}\widehat{f}_{new}(\vec{x}_i) &= \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} + \alpha \cdot 1 \cdot \vec{x}_i)^T \\ &= \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot \vec{x}_i \cdot \vec{x}_i^T = \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot ||\vec{x}_i||^2\end{aligned}$$

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$
Wrong classification:

- **Case 1:** $y_i - \widehat{f}_{old}(\vec{x}_i) = 1 \Rightarrow y_i = 1, \widehat{f}_{old}(\vec{x}_i) = 0$

$$\begin{aligned}\widehat{f}_{new}(\vec{x}_i) &= \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} + \alpha \cdot 1 \cdot \vec{x}_i)^T \\ &= \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot \vec{x}_i \cdot \vec{x}_i^T = \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot ||\vec{x}_i||^2\end{aligned}$$

$\rightarrow \vec{w}$ is incremented and classification is moved towards 1 ✓

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$

Wrong classification:

- **Case 1:** $y_i - \widehat{f}_{old}(\vec{x}_i) = 1 \Rightarrow y_i = 1, \widehat{f}_{old}(\vec{x}_i) = 0$

$$\begin{aligned} \widehat{f}_{new}(\vec{x}_i) &= \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} + \alpha \cdot 1 \cdot \vec{x}_i)^T \\ &= \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot \vec{x}_i \cdot \vec{x}_i^T = \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot ||\vec{x}_i||^2 \end{aligned}$$

  $\rightarrow \vec{w}$ is incremented and classification is moved towards 1 ✓

- **Case 2:** $y_i - \widehat{f}_{old}(\vec{x}_i) = -1 \Rightarrow y_i = 0, \widehat{f}_{old}(\vec{x}_i) = 1$

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$

Wrong classification:

- **Case 1:** $y_i - \widehat{f}_{old}(\vec{x}_i) = 1 \Rightarrow y_i = 1, \widehat{f}_{old}(\vec{x}_i) = 0$

$$\begin{aligned}\widehat{f}_{new}(\vec{x}_i) &= \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} + \alpha \cdot 1 \cdot \vec{x}_i)^T \\ &= \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot \vec{x}_i \cdot \vec{x}_i^T = \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot ||\vec{x}_i||^2\end{aligned}$$

$\rightarrow \vec{w}$ is incremented and classification is moved towards 1 ✓

- **Case 2:** $y_i - \widehat{f}_{old}(\vec{x}_i) = -1 \Rightarrow y_i = 0, \widehat{f}_{old}(\vec{x}_i) = 1$

$$\widehat{f}_{new}(\vec{x}_i) = \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} - \alpha \cdot 1 \cdot \vec{x}_i)^T$$

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$

Wrong classification:

- **Case 1:** $y_i - \widehat{f}_{old}(\vec{x}_i) = 1 \Rightarrow y_i = 1, \widehat{f}_{old}(\vec{x}_i) = 0$

$$\begin{aligned}\widehat{f}_{new}(\vec{x}_i) &= \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} + \alpha \cdot 1 \cdot \vec{x}_i)^T \\ &= \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot \vec{x}_i \cdot \vec{x}_i^T = \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot ||\vec{x}_i||^2\end{aligned}$$

  $\rightarrow \vec{w}$ is incremented and classification is moved towards 1 ✓

- **Case 2:** $y_i - \widehat{f}_{old}(\vec{x}_i) = -1 \Rightarrow y_i = 0, \widehat{f}_{old}(\vec{x}_i) = 1$

$$\begin{aligned}\widehat{f}_{new}(\vec{x}_i) &= \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} - \alpha \cdot 1 \cdot \vec{x}_i)^T \\ &= \vec{x}_i \cdot \vec{w}_{old}^T - \alpha \cdot \vec{x}_i \cdot \vec{x}_i^T = \vec{x}_i \cdot \vec{w}_{old}^T - \alpha \cdot ||\vec{x}_i||^2\end{aligned}$$

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$

Wrong classification:

- **Case 1:** $y_i - \widehat{f}_{old}(\vec{x}_i) = 1 \Rightarrow y_i = 1, \widehat{f}_{old}(\vec{x}_i) = 0$

$$\begin{aligned} \widehat{f}_{new}(\vec{x}_i) &= \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} + \alpha \cdot 1 \cdot \vec{x}_i)^T \\ &= \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot \vec{x}_i \cdot \vec{x}_i^T = \vec{x}_i \cdot \vec{w}_{old}^T + \alpha \cdot ||\vec{x}_i||^2 \end{aligned}$$

  $\rightarrow \vec{w}$ is incremented and classification is moved towards 1 ✓

- **Case 2:** $y_i - \widehat{f}_{old}(\vec{x}_i) = -1 \Rightarrow y_i = 0, \widehat{f}_{old}(\vec{x}_i) = 1$

$$\begin{aligned} \widehat{f}_{new}(\vec{x}_i) &= \vec{x}_i \cdot (\vec{w}_{new})^T = \vec{x}_i \cdot (\vec{w}_{old} - \alpha \cdot 1 \cdot \vec{x}_i)^T \\ &= \vec{x}_i \cdot \vec{w}_{old}^T - \alpha \cdot \vec{x}_i \cdot \vec{x}_i^T = \vec{x}_i \cdot \vec{w}_{old}^T - \alpha \cdot ||\vec{x}_i||^2 \end{aligned}$$

  $\rightarrow \vec{w}$ is decremented and classification is moved towards 0 ✓

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$
Correct classification: $y_i - \widehat{f}(\vec{x}_i) = 0$

- $\vec{w}_{new} = \vec{w}_{old}$, thus $\vec{w}$ is unchanged ✓

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$
Correct classification: $y_i - \widehat{f}(\vec{x}_i) = 0$

- $\vec{w}_{new} = \vec{w}_{old}$, thus $\vec{w}$ is unchanged ✓

Rosenblatt 1958 showed:

- Algorithms converges if $\mathcal{D}$ is linear separable
- Algorithm may have exponential runtime

ANN: Perceptron Learning

Update rule: $\vec{w}_{new} = \vec{w}_{old} + \alpha \cdot \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$
Correct classification: $y_i - \widehat{f}(\vec{x}_i) = 0$

- $\vec{w}_{new} = \vec{w}_{old}$, thus $\vec{w}$ is unchanged ✓

Rosenblatt 1958 showed:

- Algorithms converges if $\mathcal{D}$ is linear separable
- Algorithm may have exponential runtime

Variation: Batch processing - Update $\vec{w}$ after testing all examples

$$\vec{w}_{new} = \vec{w}_{old} + \alpha \sum_{(\vec{x}_i, y_i) \in \mathcal{D}_{wrong}} \vec{x}_i \cdot (y_i - \widehat{f}_{old}(\vec{x}_i))$$

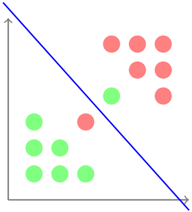
Usually: Faster convergence, but more memory needed

ANN: The XOR Problem

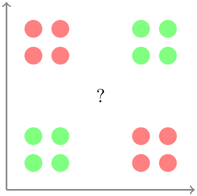
Question: What happens if data is not linear separable?

ANN: The XOR Problem

Question: What happens if data is not linear separable?



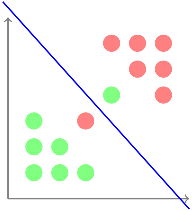
Data linear separable, but noisy



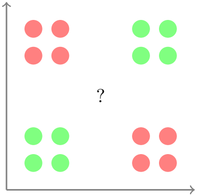
Data not linear separable

ANN: The XOR Problem

Question: What happens if data is not linear separable?



Data linear separable, but noisy



Data not linear separable

Answer: Algorithm will never converge, thus:

- Use fixed number of iterations
- Introduce some acceptable error margin

ANN: Multilayer perceptrons

Recap: (Hand crafted) Feature transformation always possible
But: What about an automatic way?

ANN: Multilayer perceptrons

Recap: (Hand crafted) Feature transformation always possible
But: What about an automatic way?

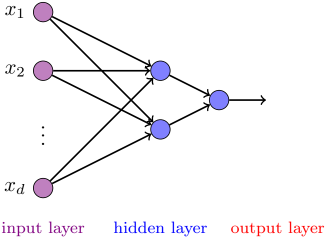
Idea: If all you have is a perceptron, use more perceptrons!

ANN: Multilayer perceptrons

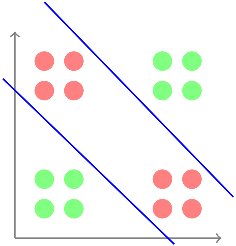
Recap: (Hand crafted) Feature transformation always possible
But: What about an automatic way?

Idea: If all you have is a perceptron, use more perceptrons!

Biology's view:



Geometric view:

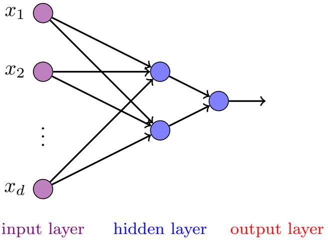


ANN: Multilayer perceptrons

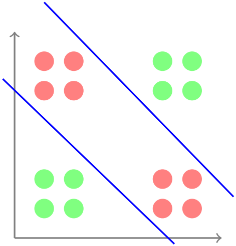
Recap: (Hand crafted) Feature transformation always possible
But: What about an automatic way?

Idea: If all you have is a perceptron, use more perceptrons!

Biology's view:



Geometric view:



Now outputs depends on layers: $\widehat{f}(\vec{x}) = f_K(\ldots f_2(f_1(\vec{x})))$

ANN: Multilayer perceptrons

Observation:

- **1 perceptron:** Separates space into two sets
- **Many perceptrons in 1 layer:** Identifies convex sets
- **Many perceptrons in 2 layer:** Identifies arbitrary sets

ANN: Multilayer perceptrons

Observation:

- **1 perceptron:** Separates space into two sets
- **Many perceptrons in 1 layer:** Identifies convex sets
- **Many perceptrons in 2 layer:** Identifies arbitrary sets

Hornik et. al 1989: MLP is a universal approximator
→ Given enough hidden units, a MLP is able to represent any "well-conditioned" function **perfectly**

ANN: Multilayer perceptrons

Observation:

- **1 perceptron:** Separates space into two sets
- **Many perceptrons in 1 layer:** Identifies convex sets
- **Many perceptrons in 2 layer:** Identifies arbitrary sets

Hornik et. al 1989: MLP is a universal approximator
→ Given enough hidden units, a MLP is able to represent any "well-conditioned" function **perfectly**

Barron 1993: Worst case needs exponential number of hidden units

ANN: Multilayer perceptrons

Observation:

- **1 perceptron:** Separates space into two sets
- **Many perceptrons in 1 layer:** Identifies convex sets
- **Many perceptrons in 2 layer:** Identifies arbitrary sets

Hornik et. al 1989: MLP is a universal approximator
$\rightarrow$ Given enough hidden units, a MLP is able to represent any "well-conditioned" function **perfectly**

Barron 1993: Worst case needs exponential number of hidden units

But: That does not necessarily mean, that we will find it!

- Usually we cannot afford exponentially large networks
- Learning of $\vec{w}$ might fail due to data or numerical reasons

MLP: Learning

Question: So how do we learn the weights of our MLP?
Unfortunately: We need some more background

MLP: Learning

Question: So how do we learn the weights of our MLP?
Unfortunately: We need some more background

So far: We formulated an **optimization** algorithm to find perceptron weights that minimize classification **error**

MLP: Learning

Question: So how do we learn the weights of our MLP?
Unfortunately: We need some more background

So far: We formulated an **optimization** algorithm to find perceptron weights that minimize classification **error**

This is a common approach in Data Mining:

- Specify model family
- Specify optimization procedure
- Specify a cost / loss function

MLP: Learning

Question: So how do we learn the weights of our MLP?
Unfortunately: We need some more background

So far: We formulated an **optimization** algorithm to find perceptron weights that minimize classification **error**

This is a common approach in Data Mining:

- Specify model family
- Specify optimization procedure
- Specify a cost / loss function

Note: Loss function $\neq$ Accuracy
$\rightarrow$ The loss function is minimized during learning
$\rightarrow$ Accuracy is used to measure the model's quality after learning

Data Mining: Loss function (1)

Question: Given a model $\widehat{f}$, some data $\mathcal{D}$, how good is $\widehat{f}$?

Fact: There are many different ways to measure the quality of $\widehat{f}$

Data Mining: Loss function (1)

Question: Given a model $\widehat{f}$, some data $\mathcal{D}$, how good is $\widehat{f}$?
Fact: There are many different ways to measure the quality of $\widehat{f}$

0-1-loss:

$$\ell(\mathcal{D}, \widehat{\theta}) = \sum_{i=1}^{N} |y_i - 1 f_{\widehat{\theta}}(\vec{x}_i)|$$

Note: We implicitly used **0-1-loss** for perceptron learning

Data Mining: Loss function (1)

Question: Given a model $\widehat{f}$, some data $\mathcal{D}$, how good is $\widehat{f}$?
Fact: There are many different ways to measure the quality of $\widehat{f}$

0-1-loss:

$$\ell(\mathcal{D}, \widehat{\theta}) = \sum_{i=1}^{N} |y_i - 1 f_{\widehat{\theta}}(\vec{x}_i)|$$

Note: We implicitly used **0-1-loss** for perceptron learning

Root-Mean Squared Error (RMSE):

$$\ell(\mathcal{D}, \widehat{\theta}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(y_i - f_{\widehat{\theta}}(\vec{x}_i)\right)^2}$$

Note: Well known, has been around for ~ 200 years

Data Mining: Loss function (2)

Let: $\mathcal{Y} = \{0, +1\}$ and $f_{\widehat{\theta}}(\vec{x}_i) \in [0, 1]$

Cross-entropy / log liklihood

$$\ell(\mathcal{D}, \widehat{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} \left(y_i \ln \left(f_{\widehat{\theta}}(\vec{x}_i) \right) + (1 - y_i) \ln \left(1 - f_{\widehat{\theta}}(\vec{x}_i) \right) \right)$$

Data Mining: Loss function (2)

Let: $\mathcal{Y} = \{0, +1\}$ and $f_{\widehat{\theta}}(\vec{x}_i) \in [0, 1]$
Cross-entropy / log liklihood

$$\ell(\mathcal{D}, \widehat{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} \left(y_i \ln \left(f_{\widehat{\theta}}(\vec{x}_i) \right) + (1 - y_i) \ln \left(1 - f_{\widehat{\theta}}(\vec{x}_i) \right) \right)$$

Observation 1: All values in logarithms are negative
Therefore: Minus sign for minimization

Data Mining: Loss function (2)

Let: $\mathcal{Y} = \{0, +1\}$ and $f_{\widehat{\theta}}(\vec{x}_i) \in [0, 1]$
Cross-entropy / log liklihood

$$\ell(\mathcal{D}, \widehat{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} \left(y_i \ln \left(f_{\widehat{\theta}}(\vec{x}_i) \right) + (1 - y_i) \ln \left(1 - f_{\widehat{\theta}}(\vec{x}_i) \right) \right)$$

Observation 1: All values in logarithms are negative
Therefore: Minus sign for minimization

Statistical interpretation: Given two distributions p and q

- how much entropy ($\approx$ chaos) is present in p
- how similar are p and q to each other?

Usually: Faster learning convergence than RMSE

Data Mining: Optimization

Question: Given loss ℓ, some data $\mathcal{D}$, how to find optimal θ?

Data Mining: Optimization

Question: Given loss ℓ, some data $\mathcal{D}$, how to find optimal θ?

Mathematically:

$$\widehat{\theta} = \arg\min_{\theta} \ell(\mathcal{D}, \theta)$$

Data Mining: Optimization

Question: Given loss ℓ, some data $\mathcal{D}$, how to find optimal θ?
Mathematically:

$$\widehat{\theta} = \arg\min_{\theta} \ell(\mathcal{D}, \theta)$$

Gradient descent: Follow steepest descent of ℓ with stepsize α

Data Mining: Optimization

Question: Given loss ℓ, some data $\mathcal{D}$, how to find optimal θ?
Mathematically:

$$\widehat{\theta} = \arg\min_{\theta} \ell(\mathcal{D}, \theta)$$

Gradient descent: Follow steepest descent of ℓ with stepsize α
$\rightarrow$ use 1st derivative $\nabla_\theta \ell(\mathcal{D}, \theta) = (\frac{\partial \ell(\mathcal{D}, \widehat{\theta})}{\partial \theta_1}, \ldots, \frac{\partial \ell(\mathcal{D}, \widehat{\theta})}{\partial \theta_d})^T$
$\rightarrow$ make a step in direction of $\nabla_\theta \ell(\mathcal{D}, \theta)$ with stepsize $\alpha \in \mathbb{R}_{>0}$

Data Mining: Optimization

Question: Given loss ℓ, some data $\mathcal{D}$, how to find optimal θ?
Mathematically:

$$\widehat{\theta} = \arg\min_{\theta} \ell(\mathcal{D}, \theta)$$

Gradient descent: Follow steepest descent of ℓ with stepsize α
→ use 1st derivative $\nabla_\theta \ell(\mathcal{D}, \theta) = (\frac{\partial \ell(\mathcal{D}, \widehat{\theta})}{\partial \theta_1}, \ldots, \frac{\partial \ell(\mathcal{D}, \widehat{\theta})}{\partial \theta_d})^T$
→ make a step in direction of $\nabla_\theta \ell(\mathcal{D}, \theta)$ with stepsize $\alpha \in \mathbb{R}_{>0}$

```
1: θ̂ = rand(1, ..., d)
2: while NOT STOP do
3:     θ̂ = θ̂ − α · ∇_θ ℓ(D, θ̂)
4: end while
```

Data Mining: Optimization

Question: Given loss ℓ, some data $\mathcal{D}$, how to find optimal θ?

Mathematically:

$$\widehat{\theta} = \arg\min_{\theta} \ell(\mathcal{D}, \theta)$$

Gradient descent: Follow steepest descent of ℓ with stepsize α

$\rightarrow$ use 1st derivative $\nabla_\theta \ell(\mathcal{D}, \theta) = (\frac{\partial \ell(\mathcal{D}, \widehat{\theta})}{\partial \theta_1}, \ldots, \frac{\partial \ell(\mathcal{D}, \widehat{\theta})}{\partial \theta_d})^T$

$\rightarrow$ make a step in direction of $\nabla_\theta \ell(\mathcal{D}, \theta)$ with stepsize $\alpha \in \mathbb{R}_{>0}$

1: $\widehat{\theta} = rand(1, \ldots, d)$
2: **while** NOT STOP **do**
3: $\quad \widehat{\theta} = \widehat{\theta} - \alpha \cdot \nabla_\theta \ell(\mathcal{D}, \widehat{\theta})$
4: **end while**

e.g. 100 iterations
e.g. minimum change in θ

Data Mining: Optimization

Question: Given loss ℓ, some data $\mathcal{D}$, how to find optimal θ?
Mathematically:

$$\widehat{\theta} = \arg\min_{\theta} \ell(\mathcal{D}, \theta)$$

Gradient descent: Follow steepest descent of ℓ with stepsize α
$\rightarrow$ use 1st derivative $\nabla_\theta \ell(\mathcal{D}, \theta) = (\frac{\partial \ell(\mathcal{D}, \widehat{\theta})}{\partial \theta_1}, \ldots, \frac{\partial \ell(\mathcal{D}, \widehat{\theta})}{\partial \theta_d})^T$
$\rightarrow$ make a step in direction of $\nabla_\theta \ell(\mathcal{D}, \theta)$ with stepsize $\alpha \in \mathbb{R}_{>0}$

$\widehat{\theta} = rand(1, \ldots, d)$
while NOT STOP **do**
    $\widehat{\theta} = \widehat{\theta} - \alpha \cdot \nabla_\theta \ell(\mathcal{D}, \widehat{\theta})$
end while

e.g. 100 iterations
e.g. minimum change in θ

Note: We implicitly used $\nabla_\theta \ell(\mathcal{D}, \widehat{\theta}) = -\vec{x}_i \cdot (y_i - \widehat{f}(\vec{x}_i))$

Summary

Important concepts:

- **Feature Engineering** is key to solve Data Mining tasks
- **Deep Learning** combines learning and Feature Engineering
- **A perceptron** is a simple linear model for classification
- **A multilayer perceptron** combine multiple perceptrons
- **For parameter optimization** we define a loss function
- **For parameter optimization** we use gradient descent
- **The learning rule** performs the actual optimization

Homework

Homework until next meeting

- Implement perceptron learning
- Test your implementation on the MNIST dataset
  - MNIST has 10 classes, so you'll need 10 perceptrons
  - Train one perceptron per class: corresponding perceptron has label 1 and remaining perceptrons label 0
  - Check predictions of all perceptrons: Predict corresponding number of perceptron with positive prediction
  - If multiple percpetrons predict 1, use that one with highest prediction value

Note 1: We will later use `C`, so please use `C` or a `C`-like language
Note 2: Use the smaller split for development and the complete data set for testing $\rightarrow$ What's your accuracy?