

Deep Learning Lab Course 2017

(Deep Learning Practical)

Labs:

(Computer Vision) Thomas Brox,
(Robotics) Wolfram Burgard,
(Machine Learning) Frank Hutter,
(Neurorobotics) Joschka Boedecker

University of Freiburg



October 30, 2017

Technical Issues

- ▶ **Location:** Monday, 14:00 - 16:00, building 082, room 00 028
- ▶ **Remark:** We will be there for questions every week from 14:00 - 16:00.
 - ▶ *We expect you to work on your own.* Your attendance is required during lectures/presentations
 - ▶ *We expect you have basic knowledge in ML (e.g. heard the Machine Learning lecture).*
- ▶ **Contact information (tutors):**
 - Aaron Klein** kleinaa@cs.uni-freiburg.de
 - Artemij Amiranashvili** amiranashvili@cs.uni-freiburg.de
 - Mohammadreza Zolfaghari** zolfagha@cs.uni-freiburg.de
 - Maria Hügle** hueglem@cs.uni-freiburg.de
 - Jingwhei Zhang** zhang@cs.uni-freiburg.de
 - Andreas Eitel** eitel@cs.uni-freiburg.de
- ▶ **Homepage:**
http://ais.informatik.uni-freiburg.de/teaching/ws17/deep_learning_course

Schedule and outline

► Phase 1

- ▶ **Today:** Introduction to Deep Learning (lecture).
- ▶ **Assignment 1) 16.10 - 06.11** Implementation of a feed-forward Neural Network from scratch (each person individually, 1-2 page report).
- ▶ **23.10:** meeting solving open questions
- ▶ **30.10:** meeting solving open questions
- ▶ **06.11:** Intro to CNNs and Tensorflow (mandatory lecture), hand in Assignment 1
- ▶ **Assignment 2) 06.11 - 20.11** Train CNN in tensorflow (each person individually, 1-2 page report)
- ▶ **13.11** meeting solving open questions

Schedule and outline

► Phase 1

- ▶ **Today:** Introduction to Deep Learning (lecture).
- ▶ **Assignment 1) 16.10 - 06.11** Implementation of a feed-forward Neural Network from scratch (each person individually, 1-2 page report).
- ▶ **23.10:** meeting solving open questions
- ▶ **30.10:** meeting solving open questions
- ▶ **06.11:** Intro to CNNs and Tensorflow (mandatory lecture), hand in Assignment 1
- ▶ **Assignment 2) 06.11 - 20.11** Train CNN in tensorflow (each person individually, 1-2 page report)
- ▶ **13.11** meeting solving open questions

► Phase 2 (split into three tracks)

- ▶ **20.11** First meeting in tracks, hand in Assignment 2
- ▶ **Assignment 3) 20.11 - 18.12** Group work track specific task, 1-2 page report
- ▶ **Assignment 4) 18.12 - 22.01** Group work advanced topic, 1-2 page report

Schedule and outline

► Phase 1

- ▶ **Today:** Introduction to Deep Learning (lecture).
- ▶ **Assignment 1) 16.10 - 06.11** Implementation of a feed-forward Neural Network from scratch (each person individually, 1-2 page report).
- ▶ **23.10:** meeting solving open questions
- ▶ **30.10:** meeting solving open questions
- ▶ **06.11:** Intro to CNNs and Tensorflow (mandatory lecture), hand in Assignment 1
- ▶ **Assignment 2) 06.11 - 20.11** Train CNN in tensorflow (each person individually, 1-2 page report)
- ▶ **13.11** meeting solving open questions

► Phase 2 (split into three tracks)

- ▶ **20.11** First meeting in tracks, hand in Assignment 2
- ▶ **Assignment 3) 20.11 - 18.12** Group work track specific task, 1-2 page report
- ▶ **Assignment 4) 18.12 - 22.01** Group work advanced topic, 1-2 page report

► Phase 3:

- ▶ **Assignment 5) 22.01 - 10.02** Solve a challenging problem of your choice using the tools from Phase 2 (group work + 5 min presentation)

Tracks

- ▶ **Track 1 Neurorobotics/Robotics**

- ▶ Robot navigation
- ▶ Deep reinforcement learning

Tracks

- ▶ **Track 1 Neurorobotics/Robotics**
 - ▶ Robot navigation
 - ▶ Deep reinforcement learning
- ▶ **Track 2 Machine Learning**
 - ▶ Architecture search and hyperparameter optimization
 - ▶ Visual recognition

Tracks

- ▶ **Track 1 Neurorobotics/Robotics**
 - ▶ Robot navigation
 - ▶ Deep reinforcement learning
- ▶ **Track 2 Machine Learning**
 - ▶ Architecture search and hyperparameter optimization
 - ▶ Visual recognition
- ▶ **Track 3 Computer Vision**
 - ▶ Image segmentation
 - ▶ Autoencoders
 - ▶ Generative adversarial networks

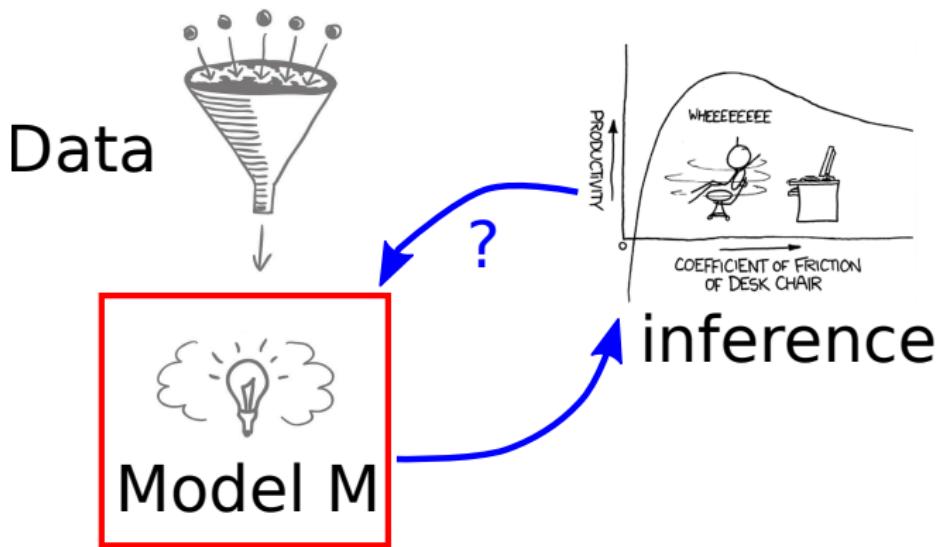
Groups and Evaluation

- ▶ In total we have three practical phases
- ▶ **Reports:**
 - ▶ Short 1-2 page report explaining your results, typically accompanied by 1-2 figures (e.g. a learning curve)
 - ▶ hand in the code you used to generate these results as well, send us your github/bitbucket repo
- ▶ **Presentations:**
 - ▶ Phase 3: 5 min group work presentation in last week
 - ▶ the reports and presentation after each exercise will be evaluated w.r.t. scientific quality and quality of presentation
- ▶ **Requirements for passing:**
 - ▶ attending the class in the lecture and presentation weeks
 - ▶ active participation in the small groups, i.e., programming, testing, writing report

Today...

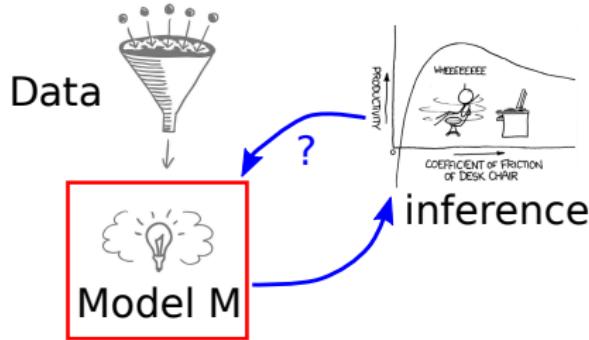
- ▶ **Groups:** You will work on your own for the first exercise
- ▶ **Assignment:** You will have three weeks to build a MLP and apply it to the MNIST dataset (more on this at the end)
- ▶ **Lecture:** Short recap on how MLPs (feed-forward neural networks) work and how to train them

(Deep) Machine Learning Overview



- 1 Learn Model **M** from the data
- 2 Let the model **M** infer unknown quantities from data

(Deep) Machine Learning Overview



| Data | Sensory Information | Query |
|--------------------|---------------------------|------------------------------------|
| Labeled Images | An image | Is a cat in the image? |
| Transcribed Speech | A speech segment | What is this person saying? |
| Paraphrases | A pair of sentences | Is this sentence a paraphrase? |
| Movie Ratings | Ratings of Y and by X | Will a user X like a movie Y ? |
| Parallel Corpora | A Finnish sentence | What is “moi” in English? |

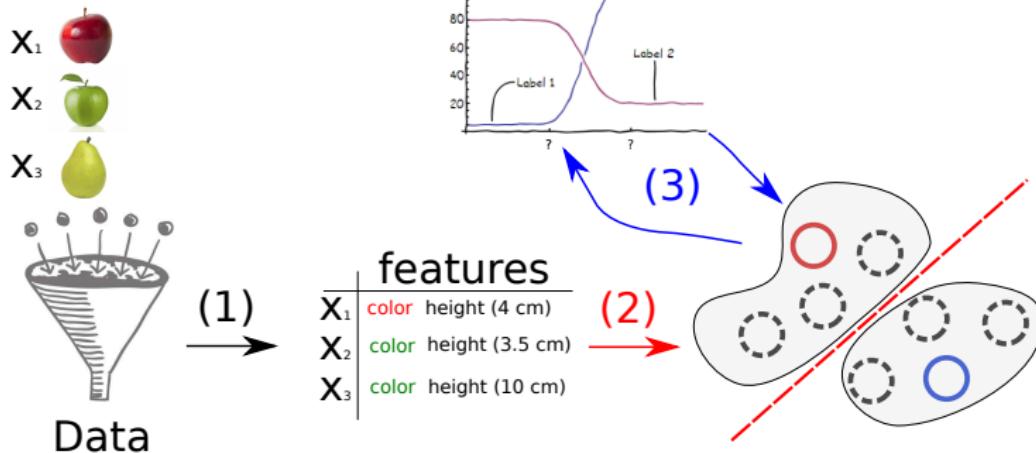
(Examples by Kyunghyun Cho)

Machine Learning Overview

What is the difference between deep learning and a standard machine learning pipeline ?

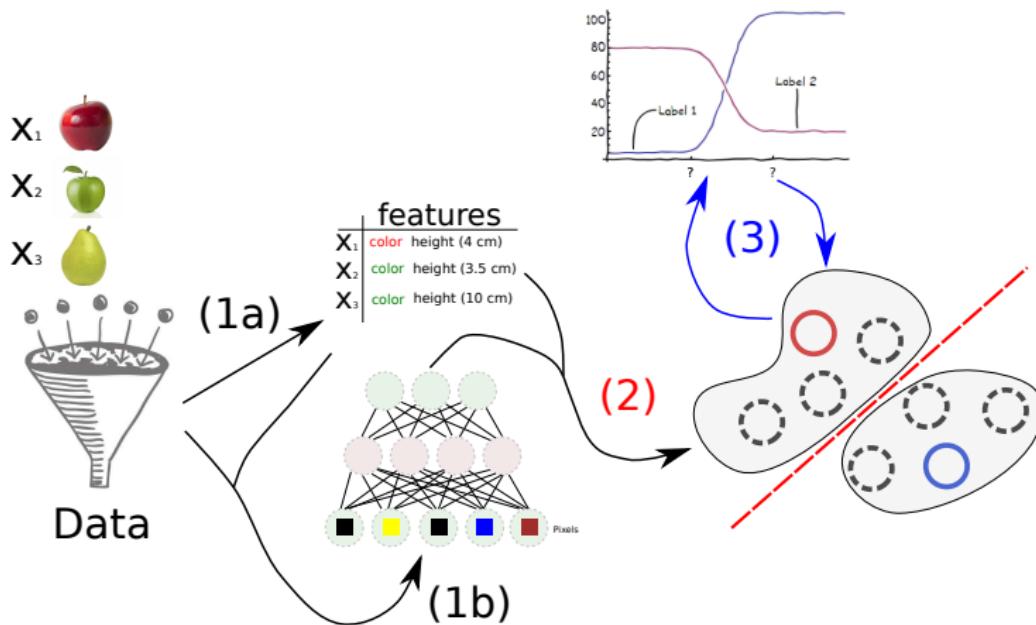
Standard Machine Learning Pipeline

- (1) Engineer good features (**not learned**)
- (2) **Learn Model**
- (3) **Inference** e.g. classes of unseen data



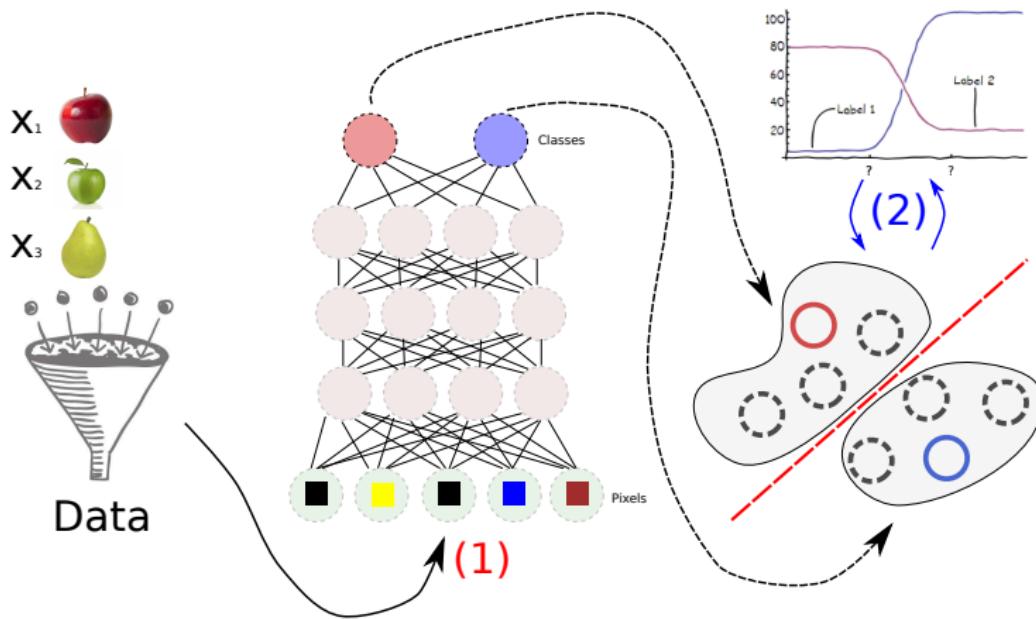
Unsupervised Feature Learning Pipeline

- (1a) Maybe engineer good features (**not learned**)
- (1b) Learn (deep) representation unsupervisedly
- (2) Learn Model
- (3) Inference e.g. classes of unseen data



Supervised Deep Learning Pipeline

- (1) Jointly **Learn** everything with a deep architecture
- (2) **Inference** e.g. classes of unseen data

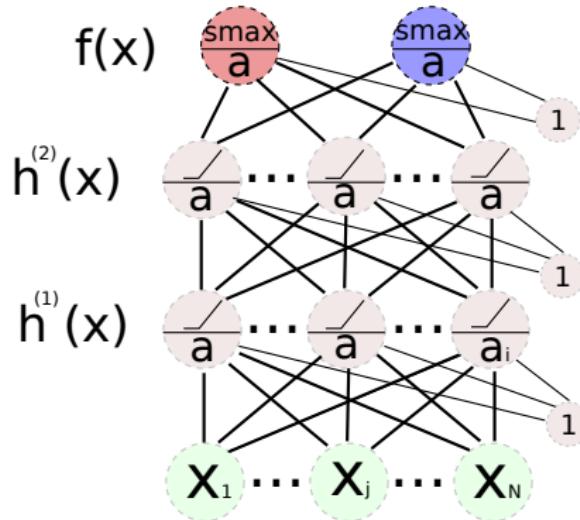


Training supervised feed-forward neural networks

- ▶ Let's formalize!
- ▶ **We are given:**
 - ▶ Dataset $D = \{(\mathbf{x}^1, \mathbf{y}^1), \dots, (\mathbf{x}^N, \mathbf{y}^N)\}$
 - ▶ A neural network with parameters θ which implements a function $f_\theta(\mathbf{x})$
- ▶ **We want to learn:**
 - ▶ The parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$

Training supervised feed-forward neural networks

- ▶ A neural network with parameters θ which implements a function $f_\theta(\mathbf{x})$
- θ is given by the network weights w and bias terms b



Neural network forward-pass

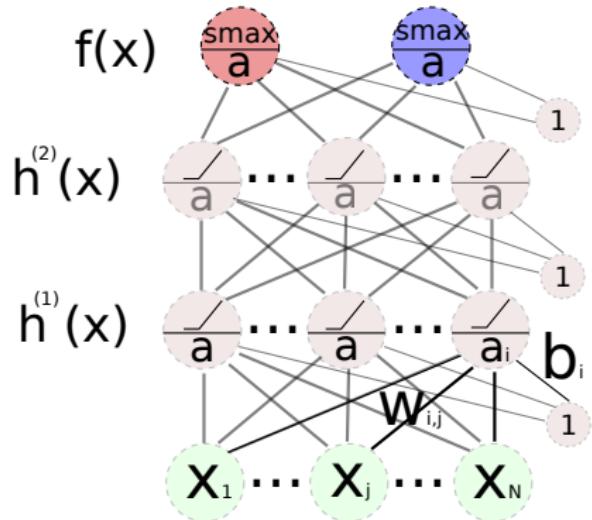
- ▶ Computing $f_\theta(\mathbf{x})$ for a neural network is a forward-pass

- ▶ unit i **activation**:

$$a_i = \sum_{j=1}^N w_{i,j} x_j + b_i$$

- ▶ unit i **output**:

$$h_i^{(1)}(\mathbf{x}) = t(a_i) \text{ where } t(\cdot) \text{ is an activation or transfer function}$$

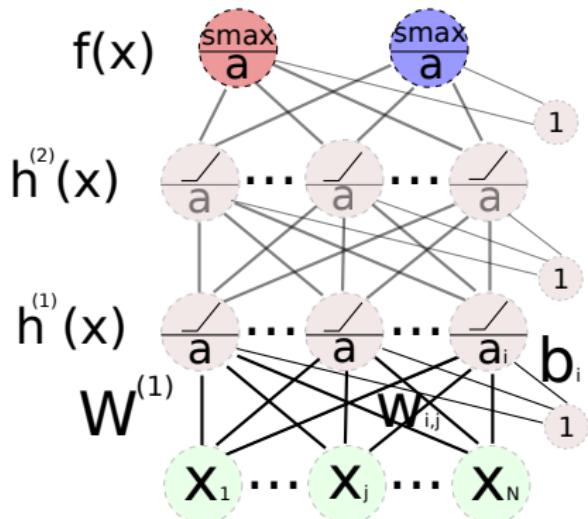


Neural network forward-pass

- ▶ Computing $f_\theta(\mathbf{x})$ for a neural network is a forward-pass

alternatively (and much faster) use vector notation:

- ▶ **layer activation:**
 $\mathbf{a}^{(1)} = \mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}$
- ▶ **layer output:**
 $h^{(1)}(\mathbf{x}) = t(\mathbf{a}^{(1)})$
 where $t(\cdot)$ is applied element wise

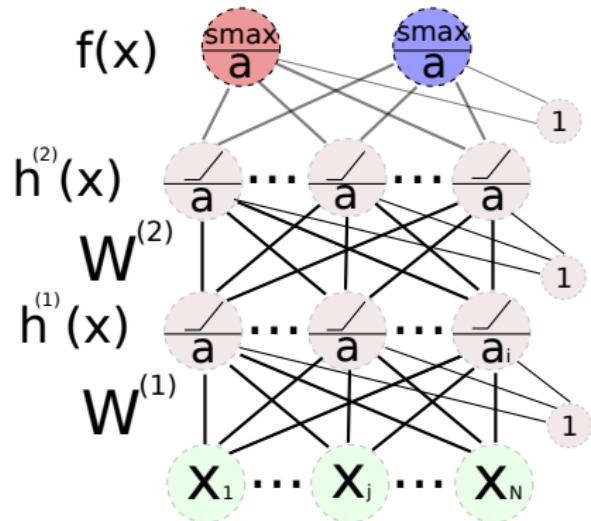


Neural network forward-pass

- ▶ Computing $f_\theta(x)$ for a neural network is a forward-pass

Second layer

- ▶ **layer 2 activation:**
 $a^{(2)} = W^{(2)}h^{(1)}(x) + b^{(2)}$
- ▶ **layer 2 output:**
 $h^{(1)}(x) = t(a^{(2)})$
 where $t(\cdot)$ is applied element wise



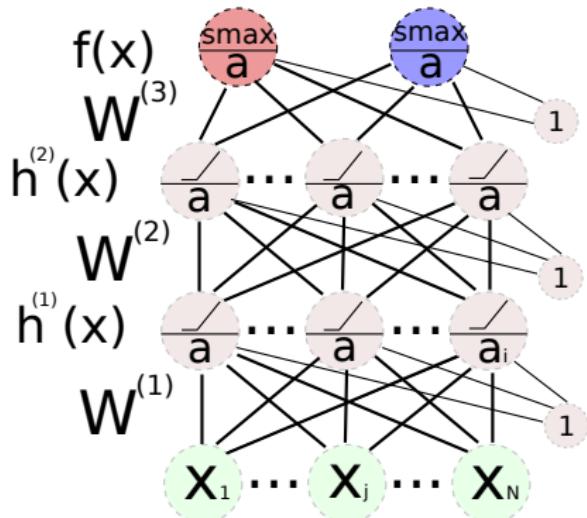
Neural network forward-pass

- ▶ Computing $f_\theta(\mathbf{x})$ for a neural network is a forward-pass

Output layer

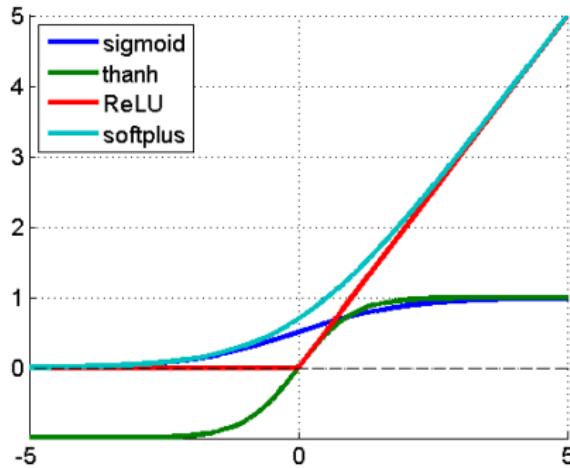
- ▶ output layer **activation**:
 $\mathbf{a}^{(3)} = \mathbf{W}^{(3)} h^{(2)}(\mathbf{x}) + \mathbf{b}^{(3)}$
- ▶ network **output**:
 $f(\mathbf{x}) = o(\mathbf{a}^{(3)})$
 where $o(\cdot)$ is the output nonlinearity
- ▶ for **classification** use softmax:

$$o_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{|z|} e^{z_j}}$$



Training supervised feed-forward neural networks

- ▶ Neural network activation functions
- ▶ Typical nonlinearities for hidden layers are: $\tanh(a_i)$,
 $\text{sigmoid } \sigma(a_i) = \frac{1}{1 + e^{-a_i}}$, $\text{ReLU } \text{relu}(a_i) = \max(a_i, 0)$
- ▶ \tanh and sigmoid are both **squashing** non-linearities
- ▶ ReLU just **thresholds** at 0
- Why not linear ?



Training supervised feed-forward neural networks

- ▶ Train parameters θ such that $\forall i \in [1, N] : f_{\theta}(\mathbf{x}^i) = \mathbf{y}^i$

Training supervised feed-forward neural networks

- ▶ Train parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$
- ▶ We can do this via minimizing the empirical risk on our dataset D

$$\min_{\theta} L(f_\theta, D) = \min_{\theta} \frac{1}{N} \sum_{i=1}^N l(f_\theta(\mathbf{x}^i), \mathbf{y}^i), \quad (1)$$

where $l(\cdot, \cdot)$ is a per example loss

Training supervised feed-forward neural networks

- ▶ Train parameters θ such that $\forall i \in [1, N] : f_\theta(\mathbf{x}^i) = \mathbf{y}^i$
- ▶ We can do this via minimizing the empirical risk on our dataset D

$$\min_{\theta} L(f_\theta, D) = \min_{\theta} \frac{1}{N} \sum_{i=1}^N l(f_\theta(\mathbf{x}^i), \mathbf{y}^i), \quad (1)$$

where $l(\cdot, \cdot)$ is a per example loss

- ▶ For regression often use the squared loss:

$$l(f_\theta(\mathbf{x}), \mathbf{y}) = \frac{1}{2} \sum_{j=1}^M (f_{j,\theta}(\mathbf{x}) - y_j)^2$$

- ▶ For M-class classification use the negative log likelihood:

$$l(f_\theta(\mathbf{x}), \mathbf{y}) = \sum_j^M -\log(f_{j,\theta}(\mathbf{x})) \cdot y_j$$

Gradient descent

- ▶ The simplest approach to minimizing $\min_{\theta} L(f_{\theta}, D)$ is gradient descent

Gradient descent:

$\theta^0 \leftarrow$ init randomly

do

$$\blacktriangleright \theta^{t+1} = \theta^t - \gamma \frac{\partial L(f_{\theta}, D)}{\partial \theta}$$

while

$$(L(f_{\theta^{t+1}}, V) - L(f_{\theta^t}, V))^2 > \epsilon$$

Gradient descent

- The simplest approach to minimizing $\min_{\theta} L(f_{\theta}, D)$ is gradient descent

Gradient descent:

$\theta^0 \leftarrow \text{init randomly}$

do

- $$\theta^{t+1} = \theta^t - \gamma \frac{\partial L(f_{\theta}, D)}{\partial \theta}$$

while

$$(L(f_{\theta^{t+1}}, V) - L(f_{\theta^t}, V))^2 > \epsilon$$

- Where V is a validation dataset (why not use D ?)

- Remember in our case: $L(f_{\theta}, D) = \frac{1}{N} \sum_{i=1}^N l(f_{\theta}(\mathbf{x}^i), \mathbf{y}^i)$

- We will get to computing the derivatives shortly

Gradient descent

- ▶ Gradient descent example: $D = \{(x^1, y^1), \dots, (x^{100}, y^{100})\}$ with

$$x \sim \mathcal{U}[0, 1]$$

$$y = 3 \cdot x + \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, 0.1)$$

Gradient descent

- ▶ Gradient descent example: $D = \{(x^1, y^1), \dots, (x^{100}, y^{100})\}$ with

$$x \sim \mathcal{U}[0, 1]$$

$$y = 3 \cdot x + \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, 0.1)$$

- ▶ Learn parameters θ of function $f_\theta(x) = \theta x$ using loss

$$l(f_\theta(x), y) = \frac{1}{2} \|f_\theta(\mathbf{x}) - \mathbf{y}\|_2^2 = \frac{1}{2} (f_\theta(x) - y)^2$$

$$\frac{\partial L(f_\theta, D)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \frac{\partial l(f_\theta, D)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N (\theta x - y)x$$

Gradient descent

- ▶ Gradient descent example: $D = \{(x^1, y^1), \dots, (x^{100}, y^{100})\}$ with

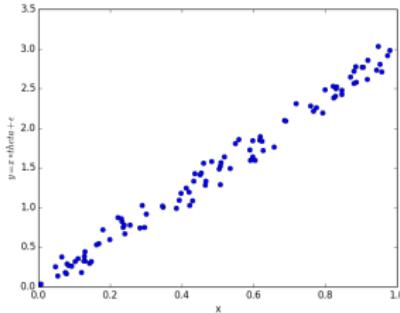
$$x \sim \mathcal{U}[0, 1]$$

$$y = 3 \cdot x + \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, 0.1)$$

- ▶ Learn parameters θ of function $f_\theta(x) = \theta x$ using loss

$$l(f_\theta(x), y) = \frac{1}{2} \|f_\theta(\mathbf{x}) - \mathbf{y}\|_2^2 = \frac{1}{2} (f_\theta(x) - y)^2$$

$$\frac{\partial L(f_\theta, D)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \frac{\partial l(f_\theta, D)}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N (\theta x - y)x$$



Gradient descent

- ▶ Gradient descent example $\gamma = 2$.

gradient descent

Stochastic Gradient descent (SGD)

- ▶ There are two problems with gradient descent:
 1. We have to find a good γ
 2. Computing the gradient is expensive if the training dataset is large!
- ▶ Problem 2 can be attacked with online optimization
(we will have a look at this)
- ▶ Problem 1 remains but can be tackled via second order methods or other advanced optimization algorithms (rprop/rmsprop, adagrad)

Gradient descent

1. We have to find a good γ ($\gamma = 2.$, $\gamma = 5.$)

gradient descent 2

Stochastic Gradient descent (SGD)

- 2 Computing the gradient is expensive if the training dataset is large!
► What if we would only evaluate f on parts of the data ?

Stochastic Gradient Descent:

$\theta^0 \leftarrow$ init randomly

do

► $(\mathbf{x}', \mathbf{y}') \sim D$

sample example from dataset D

► $\theta^{t+1} = \theta^t - \gamma^t \frac{\partial l(f_\theta(\mathbf{x}'), \mathbf{y}')}{\partial \theta}$

while $(L(f_{\theta^{t+1}}, V) - L(f_{\theta^t}, V))^2 > \epsilon$

where $\sum_{t=1}^{\infty} \gamma^t \rightarrow \infty$ and $\sum_{t=1}^{\infty} (\gamma^t)^2 < \infty$

(γ should go to zero but not too fast)

Stochastic Gradient descent (SGD)

- 2 Computing the gradient is expensive if the training dataset is large!
- ▶ What if we would only evaluate f on parts of the data ?

Stochastic Gradient Descent:

$\theta^0 \leftarrow$ init randomly

do

▶ $(\mathbf{x}', \mathbf{y}') \sim D$

sample example from dataset D

▶ $\theta^{t+1} = \theta^t - \gamma^t \frac{\partial l(f_\theta(\mathbf{x}'), \mathbf{y}')}{\partial \theta}$

while $(L(f_{\theta^{t+1}}, V) - L(f_{\theta^t}, V))^2 > \epsilon$

where $\sum_{t=1}^{\infty} \gamma^t \rightarrow \infty$ and $\sum_{t=1}^{\infty} (\gamma^t)^2 < \infty$

(γ should go to zero but not too fast)

- SGD can speed up optimization for large datasets
- but can yield very noisy updates
- in practice mini-batches are used
(compute $l(\cdot, \cdot)$ for several samples and average)
- we still have to find a learning rate schedule γ^t

Stochastic Gradient descent (SGD)

- Same data, assuming that gradient evaluation on all data takes 4 times as much time as evaluating a single datapoint

(gradient descent ($\gamma = 2$), stochastic gradient descent ($\gamma^t = 0.01 \frac{1}{t}$))

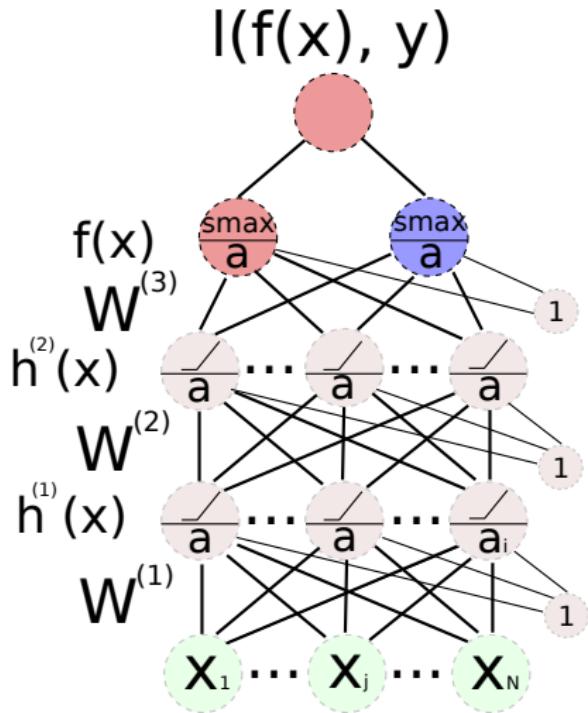
Neural Network backward pass

→ Now how do we compute the gradient for a network ?

- ▶ Use the chain rule:

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x}$$

- ▶ first compute loss on output layer
- ▶ then backpropagate to get
 $\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{W}^{(3)}}$ and $\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}^{(3)}}$



Neural Network backward pass

→ Now how do we compute the gradient for a network ?

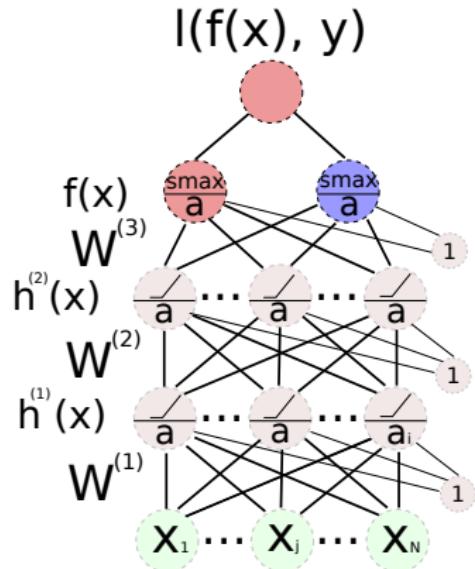
- ▶ gradient wrt. layer 3 **weights**:

$$\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{W}^{(3)}} = \frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}^{(3)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}}$$
- ▶ assuming l is NLL and softmax outputs, gradient wrt. layer 3 activation is:

$$\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}^{(3)}} = -(\mathbf{y} - f(\mathbf{x}))$$

 \mathbf{y} is one-hot encoded
- ▶ gradient of \mathbf{a} wrt. $\mathbf{W}^{(3)}$:

$$\frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}} = h^{(2)}(\mathbf{x})^T$$



→ recall

$$\mathbf{a}^{(3)} = \mathbf{W}^{(3)} h^{(2)}(\mathbf{x}) + \mathbf{b}^{(3)}$$

Neural Network backward pass

→ Now how do we compute the gradient for a network ?

- ▶ gradient wrt. layer 3 **weights**:

$$\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{W}^{(3)}} = \frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}^{(3)}} \mathbf{W}^{(3)}$$

- ▶ assuming l is NLL and softmax outputs, gradient wrt. layer 3 activation is:

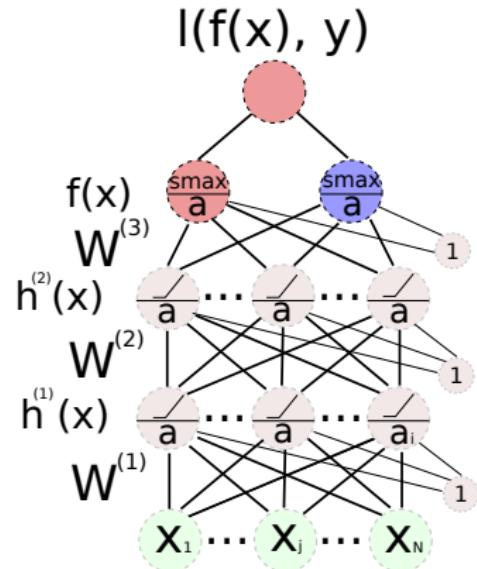
$$\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}^{(3)}} = -(\mathbf{y} - f(\mathbf{x}))$$

- ▶ gradient of \mathbf{a} wrt. $\mathbf{W}^{(3)}$:

$$\frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}} = h^{(2)}(\mathbf{x})^T$$

- ▶ combined:

$$\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{W}^{(3)}} = -(\mathbf{y} - f(\mathbf{x}))(h^{(2)}(\mathbf{x}))^T$$



→ recall

$$\mathbf{a}^{(3)} = \mathbf{W}^{(3)} h^{(2)}(\mathbf{x}) + \mathbf{b}^{(3)}$$

Neural Network backward pass

→ Now how do we compute the gradient for a network ?

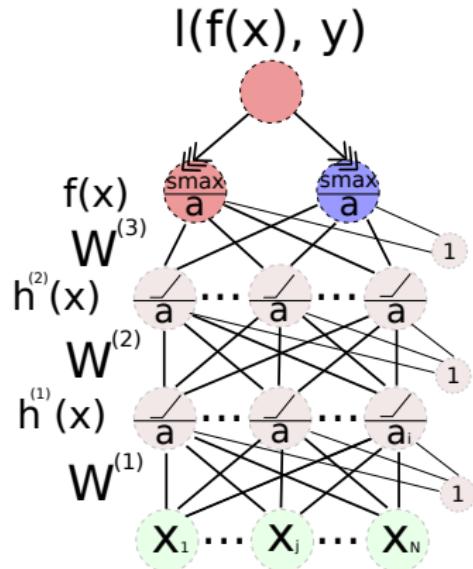
- ▶ gradient wrt. layer 3 **weights**:

$$\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{W}^{(3)}} = \frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}^{(3)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}}$$
- ▶ assuming l is NLL and softmax outputs, gradient wrt. layer 3 activation is:

$$\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}^{(3)}} = -(\mathbf{y} - f(\mathbf{x}))$$
- ▶ gradient of \mathbf{a} wrt. $\mathbf{W}^{(3)}$:

$$\frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{W}^{(3)}} = (h^{(2)}(\mathbf{x}))^T$$
- ▶ gradient wrt. **previous layer**:

$$\begin{aligned}\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial h^{(2)}(\mathbf{x})} &= \frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}^{(3)}} \frac{\partial \mathbf{a}^{(3)}}{\partial h^{(2)}(\mathbf{x})} \\ &= (\mathbf{W}^{(3)})^T \frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{a}^{(3)}}\end{aligned}$$



→ recall

$$\mathbf{a}^{(3)} = \mathbf{W}^{(3)} h^{(2)}(\mathbf{x}) + \mathbf{b}^{(3)}$$

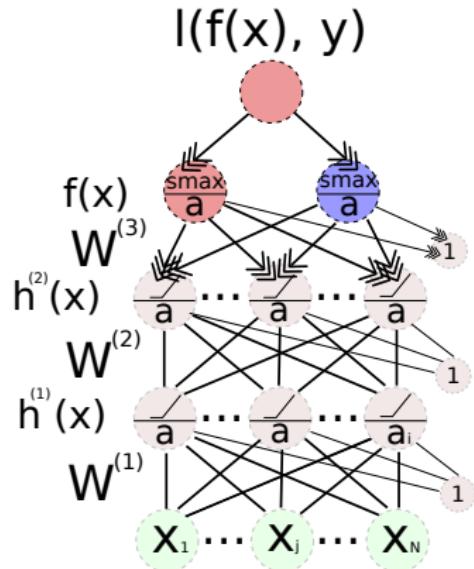
Neural Network backward pass

→ Now how do we compute the gradient for a network ?

- ▶ gradient wrt. layer 2 **weights**:

$$\frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial \mathbf{W}^{(2)}} = \frac{\partial l(f(\mathbf{x}), \mathbf{y})}{\partial h^{(2)}(\mathbf{x})} \frac{\partial h^{(2)}(\mathbf{x})}{\partial \mathbf{a}^{(2)}} \mathbf{W}^{(2)}$$
- ▶ same schema as before just have to consider computing derivative of activation function $\frac{\partial h^{(2)}(\mathbf{x})}{\partial \mathbf{a}^{(2)}}$, e.g. for sigmoid $\sigma(\cdot)$

$$\frac{\partial h^{(2)}(\mathbf{x})}{\partial \mathbf{a}^{(2)}} = \sigma(a_i)(1 - a_i)$$
- ▶ and backprop even further



→ recall

$$\mathbf{a}^{(3)} = \mathbf{W}^{(3)} h^{(2)}(\mathbf{x}) + \mathbf{b}^{(3)}$$

Gradient Checking

- Backward-pass is just repeated application of the **chain rule**
- However, there is a huge potential for bugs ...
- Gradient checking to the rescue (simply check code via finite-differences):

Gradient Checking:

$\theta = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots)$ init randomly

$\mathbf{x} \leftarrow$ init randomly ; $\mathbf{y} \leftarrow$ init randomly

$g_{\text{analytic}} \leftarrow$ compute gradient $\frac{\partial l(f_\theta(\mathbf{x}), \mathbf{y})}{\partial \theta}$ via backprop

for i in $\#\theta$

▶ $\hat{\theta} = \theta$

▶ $\hat{\theta}_i = \hat{\theta}_i + \epsilon$

▶ $g_{\text{numeric}} = \frac{l(f_{\hat{\theta}}(\mathbf{x}), \mathbf{y}) - l(f_\theta(\mathbf{x}), \mathbf{y})}{\epsilon}$

▶ assert($|g_{\text{numeric}} - g_{\text{analytic}}| < \epsilon$)

- ▶ can also be used to test partial implementations
(i.e. layers, activation functions)
 - simply remove loss computation and backprop **ones**

Overfitting

- ▶ If you train the parameters of a large network $\theta = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots)$ you will see overfitting!
→ $L(f_\theta(x), D) \ll L(f_\theta(x), V)$
- ▶ This can be at least partly conquered with regularization

Overfitting

- ▶ If you train the parameters of a large network $\theta = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots)$ you will see overfitting!
→ $L(f_\theta(x), D) \ll L(f_\theta(x), V)$
- ▶ This can be at least partly conquered with regularization
 - ▶ **weight decay:** change cost (and gradient)

$$L(f_\theta, D) = \frac{1}{N} \min_{\theta} \sum_{i=1}^N l(f_\theta(\mathbf{x}^i), \mathbf{y}^i) + \frac{1}{\#\theta} \sum_i \|\theta_i\|^2$$

→ enforces small weights (occams razor)

Overfitting

- ▶ If you train the parameters of a large network $\theta = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots)$ you will see overfitting!
 → $L(f_\theta(x), D) \ll L(f_\theta(x), V)$
- ▶ This can be at least partly conquered with regularization
 - ▶ **weight decay:** change cost (and gradient)

$$L(f_\theta, D) = \frac{1}{N} \min_{\theta} \sum_{i=1}^N l(f_\theta(\mathbf{x}^i), \mathbf{y}^i) + \frac{1}{\#\theta} \sum_i \|\theta_i\|^2$$

- enforces small weights (occams razor)
- ▶ **dropout:** kill $\approx 50\%$ of the activations in each hidden layer **during training** forward pass. Multiply hidden activations by $\frac{1}{2}$ **during testing**
 → prevents co-adaptation / enforces robustness to noise

Overfitting

- ▶ If you train the parameters of a large network $\theta = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots)$ you will see overfitting!
→ $L(f_\theta(x), D) \ll L(f_\theta(x), V)$
- ▶ This can be at least partly conquered with regularization
 - ▶ **weight decay:** change cost (and gradient)

$$L(f_\theta, D) = \frac{1}{N} \min_{\theta} \sum_{i=1}^N l(f_\theta(\mathbf{x}^i), \mathbf{y}^i) + \frac{1}{\#\theta} \sum_i \|\theta_i\|^2$$

- enforces small weights (occams razor)
- ▶ **dropout:** kill $\approx 50\%$ of the activations in each hidden layer **during training** forward pass. Multiply hidden activations by $\frac{1}{2}$ **during testing**
→ prevents co-adaptation / enforces robustness to noise
- ▶ Many, many more !

Assignment

- ▶ **Implementation:** Implement a simple feed-forward neural network by completing the provided *stub* this includes:
 - ▶ possibility to use 2-4 layers
 - ▶ sigmoid/tanh and ReLU for the hidden layer
 - ▶ softmax output layer
 - ▶ optimization via gradient descent (gd)
 - ▶ optimization via stochastic gradient descent (sgd)
 - ▶ gradient checking code (!!!)
 - ▶ weight initialization with random noise (!!!) (use normal distribution with changing std. deviation for now)
- ▶ Bonus points for testing some advanced ideas:
 - ▶ implement dropout, l2 regularization
 - ▶ implement a different optimization scheme (rprop, rmsprop, adagrad)
- ▶ **Code stub:** https://github.com/aisrobots/dl_lab_2017
- ▶ **Evaluation:**
 - ▶ Find good parameters (learning rate, number of iterations etc.) using a validation set (usually take the last 10k examples from the training set)
 - ▶ After optimizing parameters run on the full dataset and test once on the test-set (you should be able to reach $\approx 1.6 - 1.8\%$ error)
- ▶ **Submission:** Clone our github repo and send us the link to your github/bitbucket repo including your solution code and report. Email to eitel@cs.uni-freiburg.de, hueglem@cs.uni-freiburg.de and zhang@cs.uni-freiburg.de

Slide Information

- ▶ These slides have been created by Tobias Springenberg for the DL Lab Course WS 2016/2017.