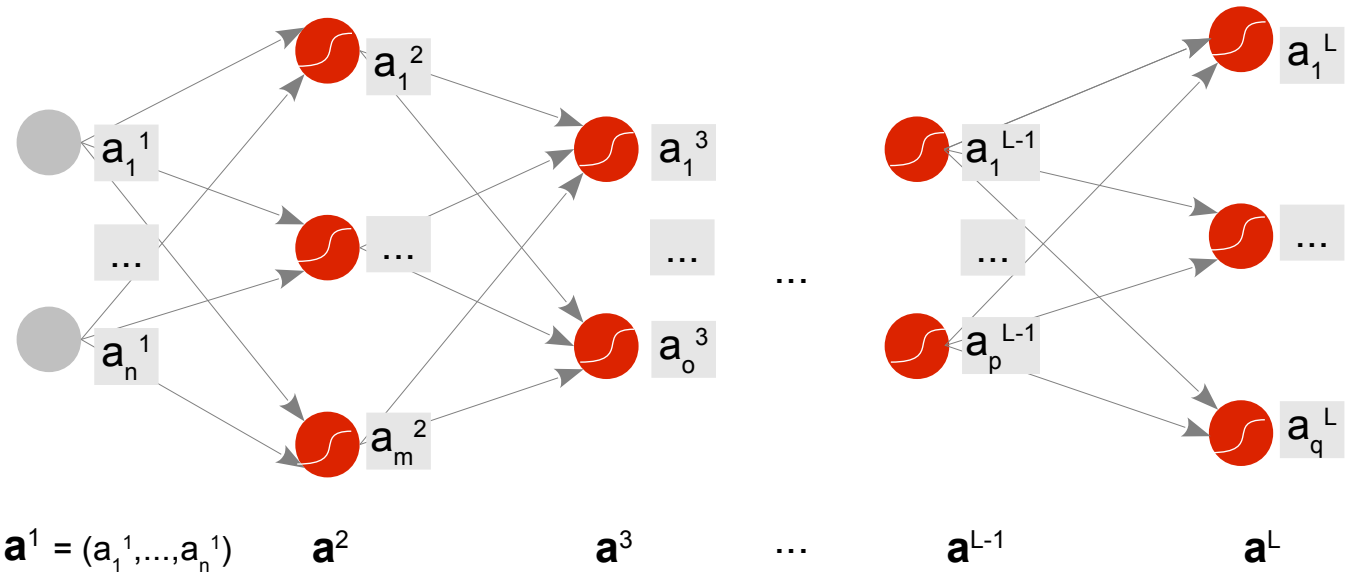
Machine Learning

– winter term 2016/17 –

Chapter 08: Neural Networks II

Prof. Adrian Ulges
Masters “Computer Science”
DCSM Department
University of Applied Sciences RheinMain

The Multi-layer Perceptron (MLP)

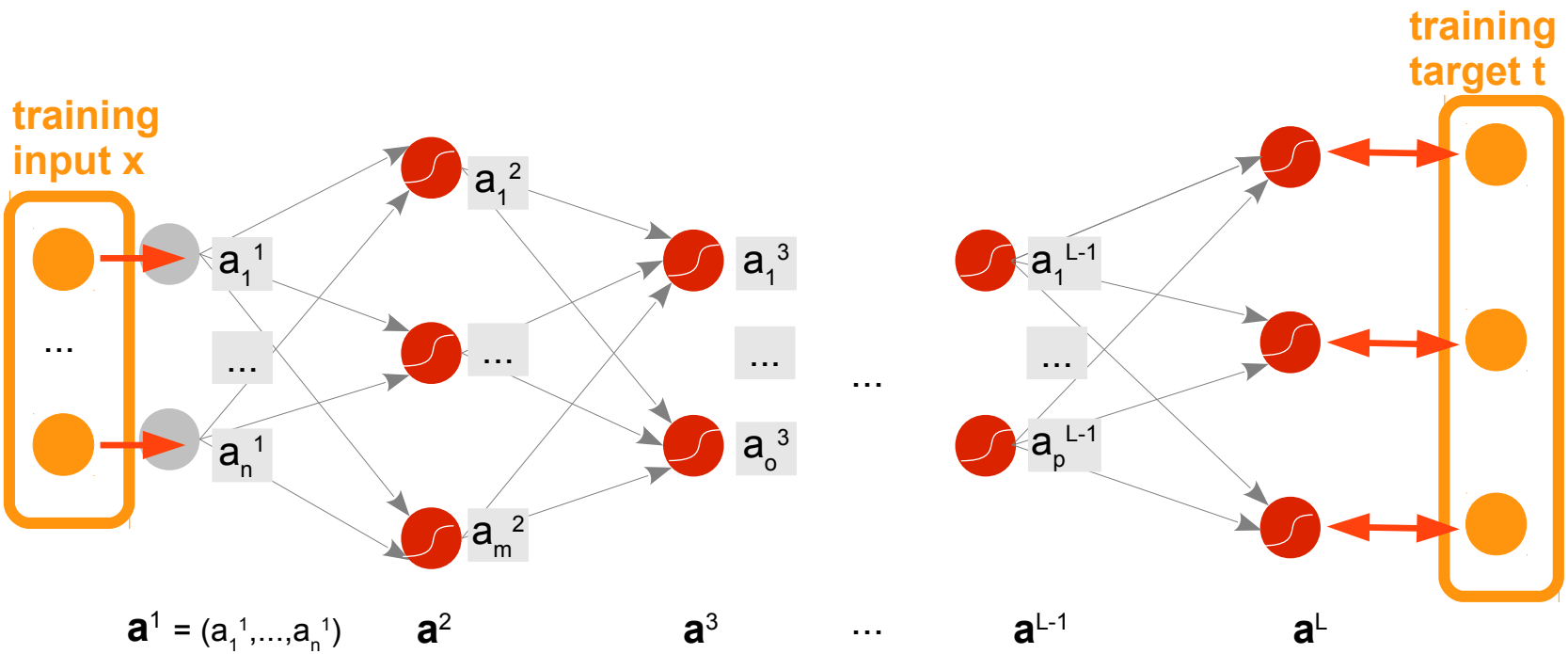


In the following, we focus on the most common type of neural network, the **multi-layer perceptron** (MLP).

- The network is **feed-forward**
- There are **L layers** in total
- All layers are **fully connected**
- We have already seen: Such a network can learn **any continuous function** *(in theory)*!

MLP: Encoding Learning Problems

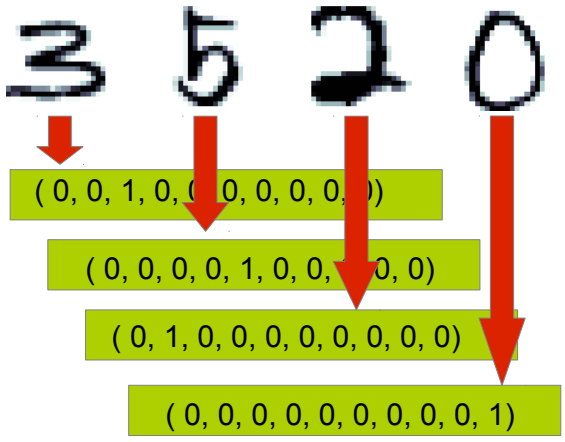
- **Learning problems** are represented as pairs of inputs and desired outputs
- $\mathbf{x} = (x_1, ..., x_d)$ input (feature vector)
- $\mathbf{t} = (t_1, ..., t_p)$ desired outputs ("targets")
- **Goal**: The network should approximate the target!



MLP Learning Problems: Examples image from [5]

Classification: Digit Recognition

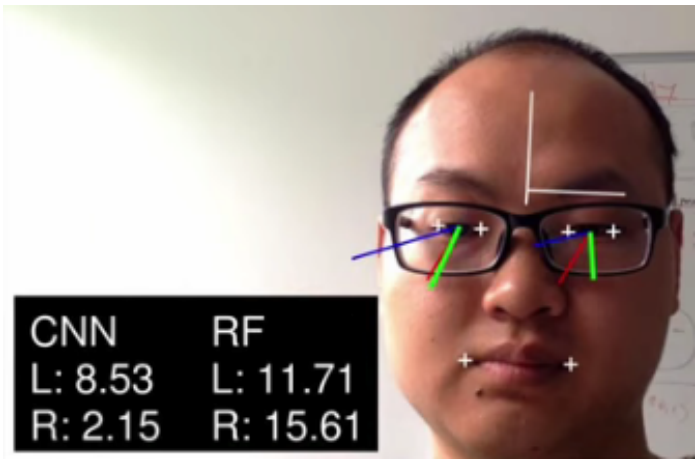
- $\mathbf{x} = (x_1, ..., x_{748})$
- target = **one-hot encoding** of class
- Example: picture shows a 4 $\rightarrow$ $\mathbf{t} = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$



target vectors

Regression: Gaze Tracking

- $\mathbf{x} = (x_1, ..., x_{748})$: an eye image
- $\mathbf{t} = (t_1, t_2)$: the eye's viewing direction (two polar coordinates)



Outline

MLP: Learning

Learning in neural networks works just like for single neurons. We initialize all weights with **random values**. In each iteration, we...

- choose a training sample ($\mathbf{x}$,$\mathbf{t}$) and feed it to the network
- compare the result $\mathbf{a}^L$ with the target $\mathbf{t}$
- **correct** the weights of the network such that the output moves towards the targets

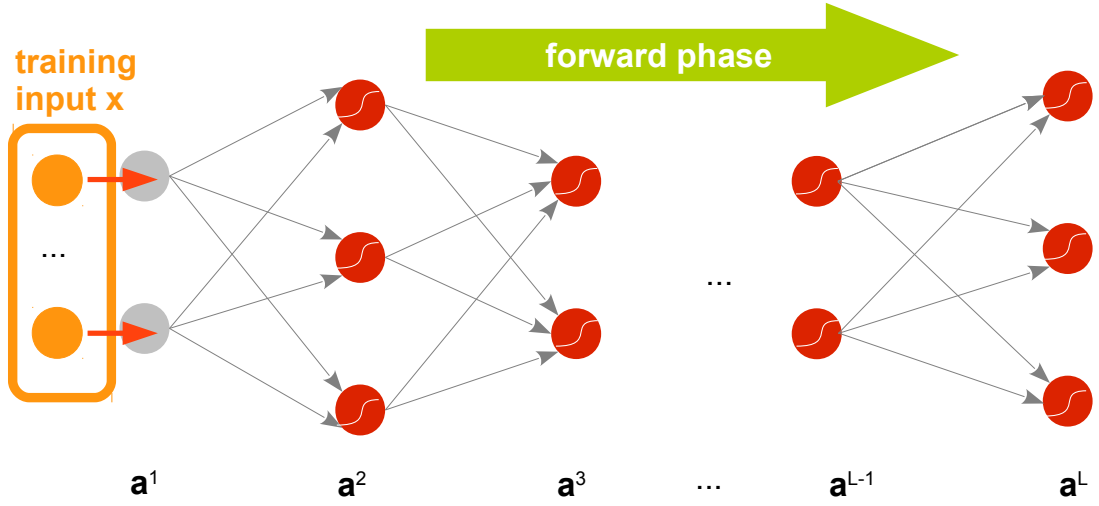
Remarks

- The difference to training **single neurons**: We need to adapt weights across **multiple layers** of the network!

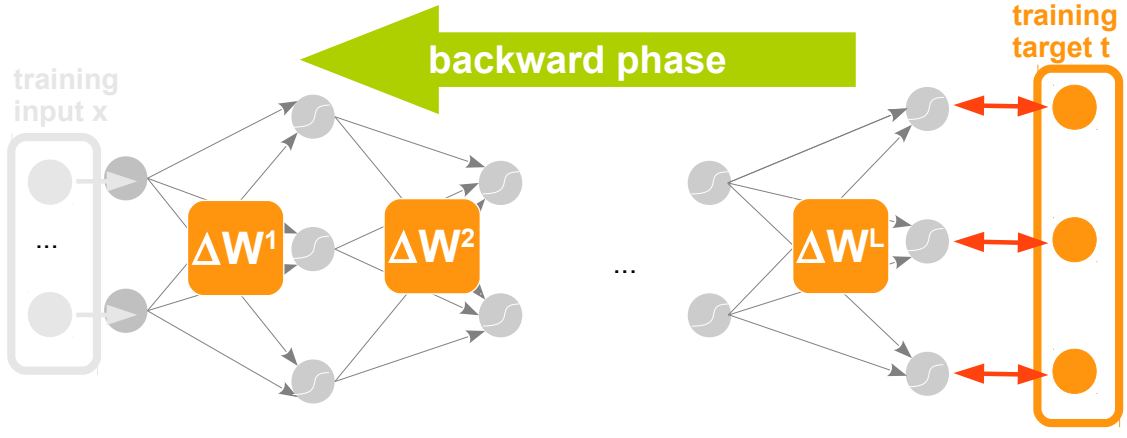
MLP: Learning

The Phases of Learning

Each learning iteration consists of two phases



1. **Forward phase**:
   Feed **x** to the net.
   The signal propagates forward through the layers. The activations of all neurons, $\mathbf{a}^1, ..., \mathbf{a}^L$, are computed.
2. **Backward phase**: We compare $\mathbf{a}^L$ with **t** and update the weights (and biases) in all layers $l = L, (L-1), ..., 2, 1$:
   - $\mathbf{W}^l := \mathbf{W}^l + \Delta\mathbf{W}^l$ (weights)
   - $\mathbf{b}^l := \mathbf{b}^l + \Delta\mathbf{b}^l$ (biases)



Backpropagation

- How do we determine $\Delta \mathbf{W}^l$ and $\Delta \mathbf{b}^l$?
- Like for the Delta rule, we formulate a **minimization problem**
- Goal: Minimize the **error between targets and output**

$$E(\mathbf{W}^1, \mathbf{W}^2, ..., \mathbf{W}^L, \mathbf{b}^1, \mathbf{b}^2, ..., \mathbf{b}^L) = \sum_{k=1}^{p} (a_k^L - t_k)^2$$

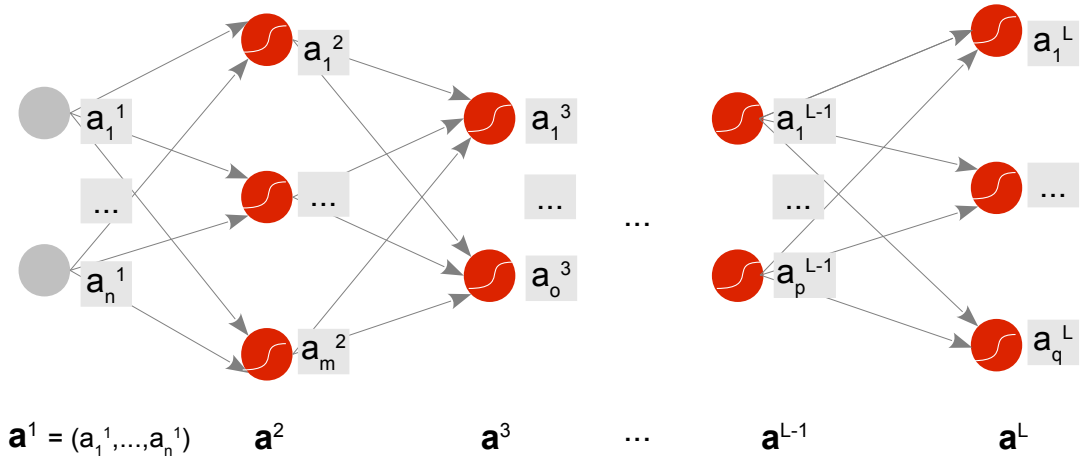
- We optimize using **gradient descent** with learning rate λ

$$\Delta w_{ij}^l := -\lambda \cdot \frac{\partial E}{\partial w_{ij}^l}$$

$$\Delta b_j^l := -\lambda \cdot \frac{\partial E}{\partial b_j^l}$$

- The resulting algorithm is called **backpropagation**.

MLP: Backpropagation



MLP: Backpropagation

MLP: Backpropagation

MLP: Backpropagation

MLP: Backpropagation

MLP: Backpropagation

Backpropagation: Algorithm

```
1  function BACKPROP(x_1, ..., x_n, t_1, ..., t_n, λ):
2      W^1, W^2, ..., W^L, b^1, b^2, ..., , b^L := initialize()
3      repeat
4          (x, t) := choose_sample()
5          feed x to the net, compute the activations a^1, ..., a^L   // forward phase
6          δ^L := (a^L − t) · f'(z^L)
7          for layer l in L, L-1, ..., 1:
8              w_ij^l := w_ij^l − λ · a_i^{l−1} · δ_j^l   for all i, j     // update weights
9              b_j^l := b_j^l − λ · δ_j^l                 for all j       // update biases
10             δ^{l−1} := (W^l · δ^l) ⊙ f'(z^{l−1})                       // update errors for propagation
11     until weights+biases stop changing
12
```

Remarks

- $\lambda > 0$ – the *learning rate* – is tricky to pick! (Why?)
- Often, we decrease λ linearly until reaching t_{max} iterations:

$$\lambda_t := (1 - \alpha_t) \cdot \lambda_{high} + \alpha_t \cdot \lambda_{low}$$

where t is the number of iterations and $\alpha_t = min(t/t_{max}, 1)$.

Backpropagation: Do-it-Yourself

- Does backpropagation always reach the global minimum of the error function E?
- Does backpropagation always converge?

Outline

Backpropagation: Batching image from [4]

1. The above algorithm updates weights based on **one** training example $(\mathbf{x}, \mathbf{t})$.

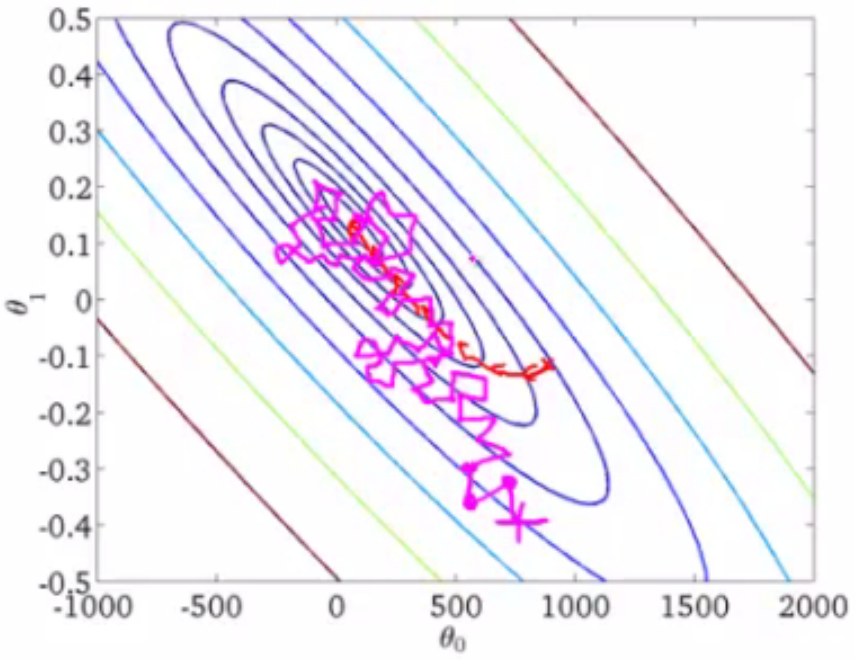
$$w_{ij}^l := w_{ij}^l - \lambda \cdot a_i^{l-1} \cdot \delta_j^l$$

2. Overall, we want to minimize the error over **all** training samples $(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_n, \mathbf{t}_n)$, however!

$$w_{ij}^l := w_{ij}^l - \lambda \cdot \frac{1}{n} \sum_{k=1}^{n} a_i^{l-1,k} \cdot \delta_j^{l,k}$$

We call these versions **(2.) gradient descent** vs. **(1.) stochastic gradient descent**. What difference does it make? Steps in gradient descent are

- smoother *(→ larger λ)*
- a lot more expensive
- parallelizable!



Backpropagation: Minibatch Version

Usually, we follow a compromise

- We split the training data into **subsets of size B**, called **mini-batches**
- We update the weights/biases by averaging over all samples of a mini-batch
- We run a few iterations of training and move on to the next minibatch

```
1  function BACKPROP_MINIBATCH(x_1, ..., x_B, t_1, ..., t_B, λ):
2          // runs a few iterations of training on a mini-batch of B samples
3      for a few iterations:
4          for each b = 1, ..., B:
5              compute activations a^{1,b}, ..., a^{L,b}   // forward phase
6              δ^{L,b} := (a^{L,b} − t_b) · f'(z^{L,b})
7          for layer l in L, L-1, ..., 1:
8              w_ij^l := w_ij^l − λ · (1/B) Σ_{b=1}^{B} a_i^{l−1,b} · δ_j^{l,b}   for all i, j   // update weights
9              b_j^l := b_j^l − λ · (1/B) Σ_{b=1}^{B} δ_j^{l,b}                   for all j     // update biases
10             δ^{l−1,b} := (W^l · δ^{l,b}) ⊙ f'(z^{l−1,b})  for b=1,..,B                       // update errors
11
```

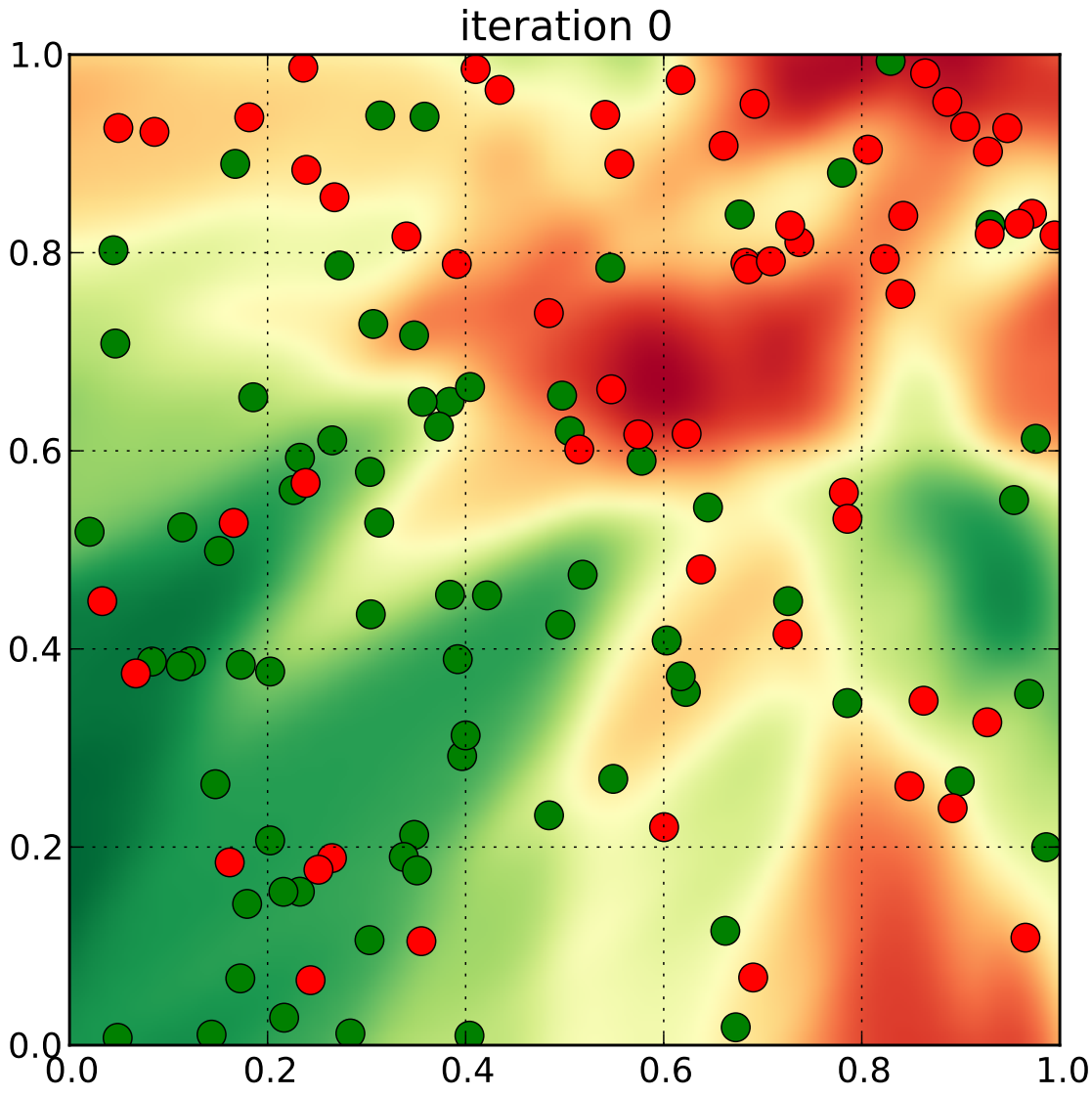
Minibatch: Best Size?

- If the mini-batches are **too small**, the gradient descent steps become very *"noisy"*. This noise is known to decrease with $1/\sqrt{B}$.
- If the mini-batches are **too large**, the single steps become very expensive.
- **Common approach**: Choose the mini-batch size to fit your (GPU) **memory** $\rightarrow$ reduces overhead for loading and storing training data, and makes maximum use of parallelization!

"To set the minibatch size, plot the validation accuracy versus time (as in, real elapsed time, not epoch!), and choose whichever mini-batch size gives you the most rapid improvement in accuracy."

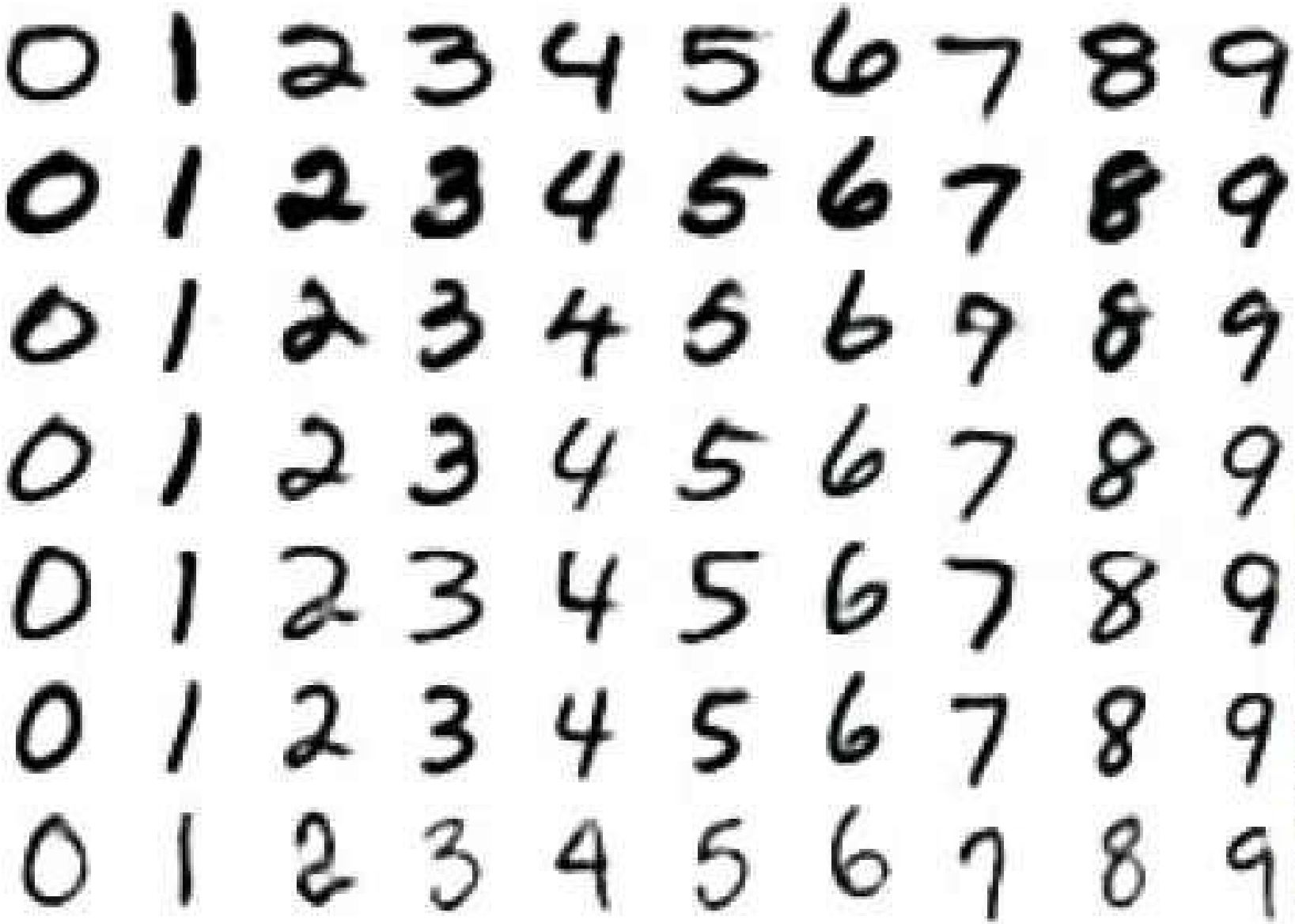
(Nielsen: "Neural Networks and Deep Learning")

MLP: Training Example



Outline

What's going wrong here? image from [1]



Neural Network Training: Parameters

Network Topology

- number of layers
- number of nodes per layer
- connectivity between layers
- Activation functions

Training

- learning rate
- number of iterations
- size of mini-batches
- initialization of weights+biases
- cost Function

Others

- input data: Preprocessing (normalization, balancing)
- dropout (later...)

Lost in Hyperparameter Space

> *"When you understand something poorly - as the explorers understood geography, and as we understand neural nets today - it's more important to explore boldly than it is to be rigorously correct in every step of your thinking."*
>
> (M. Nielsen)

Tricks of the Trade

- **Prioritize!** Investigate the most unclear parameters first, set *reasonable values* wherever possible.
- **Be quick!** Iterate quickly over small parts of the data to find promising settings.
- **Hang in there!** This is a hard problem – it has taken ML research 50 years to 'get neural network training right'.

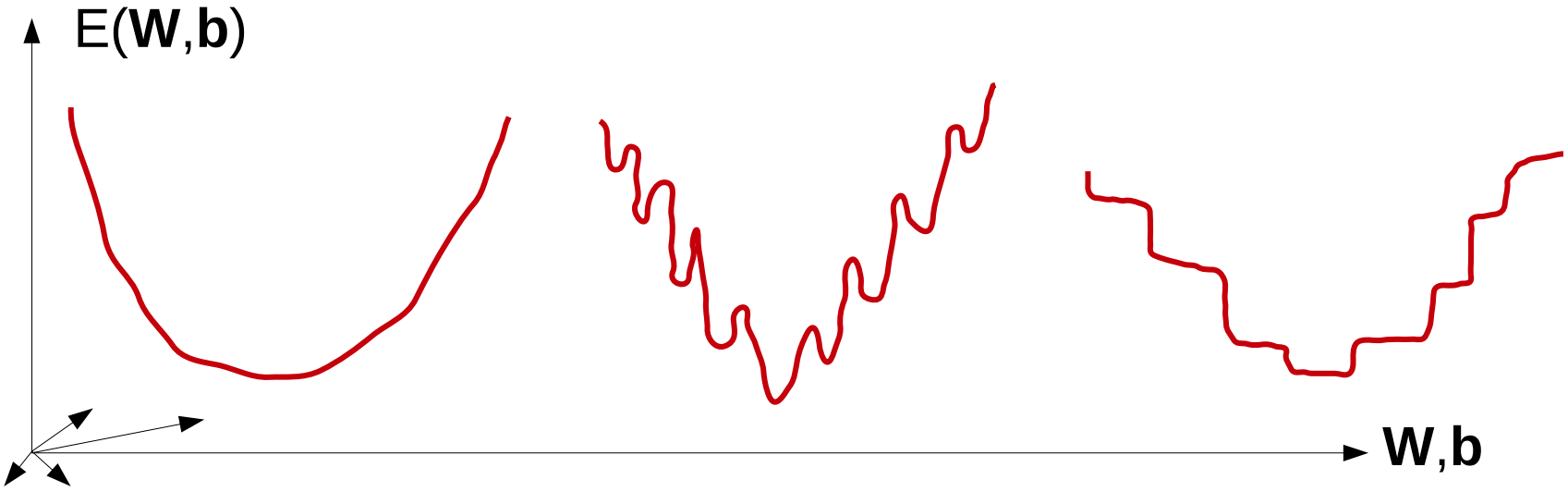
Reads

- Bengio: "Practical recommendations for gradient-based training of deep architectures" (2012).
- Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller: "Neural Networks: Tricks of the Trade" (2012).

Outline

Cost Functions in Backpropagation

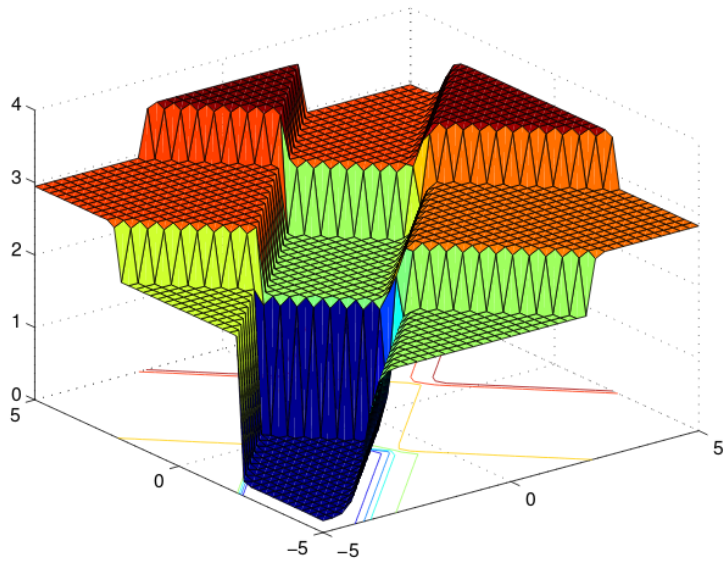
Backpropagation is just gradient descent of a high-dimensional cost function E. But what does E "look like"?

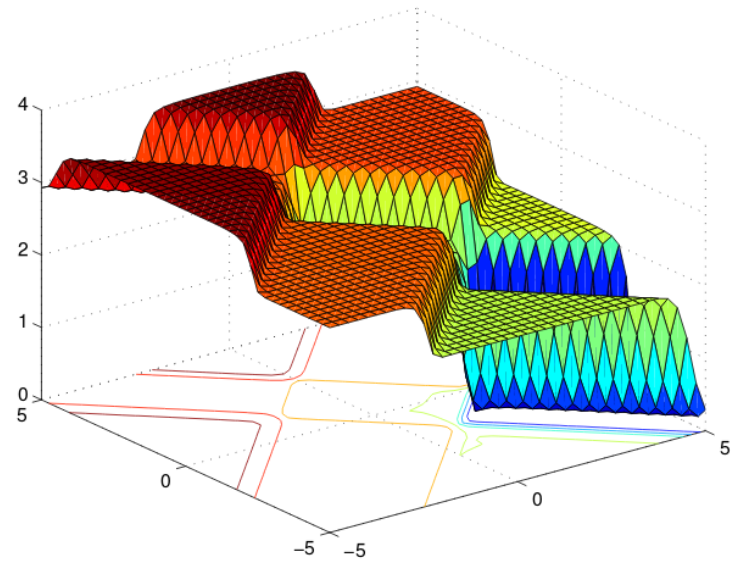


"Most practitioners believed that local minima were a common problem plaguing neural network optimization. Today, that does not appear to be the case. [...] Local minima are in fact rare compared to another kind of point with zero gradient: a **saddle point***."*

(Courville et al. [3])

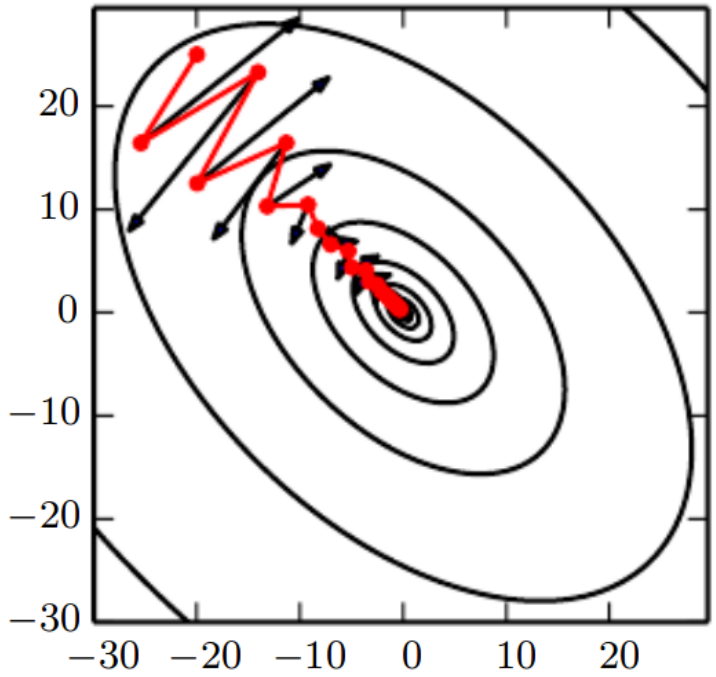
Cost Functions in Backpropagation: Example





- E is usually *flat* around saddle points.
- Learning terminates ("starves") on *plateaus!*
- Choosing a good learning rate λ becomes difficult!

Improvement: Momentum image from [3]



- Idea: **smooth** the direction of the gradient over training iterations.
- Let ∇E^t be the gradient update in iteration t. We compute

$$v^t = \alpha \cdot v^{t-1} + \nabla E^t$$

- We update the weights (and biases):

$$W := W - \lambda \cdot v^t \qquad \left(\text{instead of } W := W - \lambda \cdot \nabla E^t \right)$$

- α is the strength of momentum. Think of it in terms of $1/(1-\alpha)$ ("geometric series"): For $\alpha = 0.9$, the maximum speed is $10\times$ the gradient length.
- Momentum helps training keep a stable directions and 'roll over' plateaus.

Variations to Backpropagation: RMSProp [3] (Chapter 8)

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
  Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

RMSProp

- For each parameter, a separate multiplier $\frac{1}{\sqrt{r+\delta}}$ is learned.
- Progress is **increased** in weakly sloped directions
- no **momentum** (but could be included, though)

Variations to Backpropagation: ADAM [3] (Chapter 8)

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

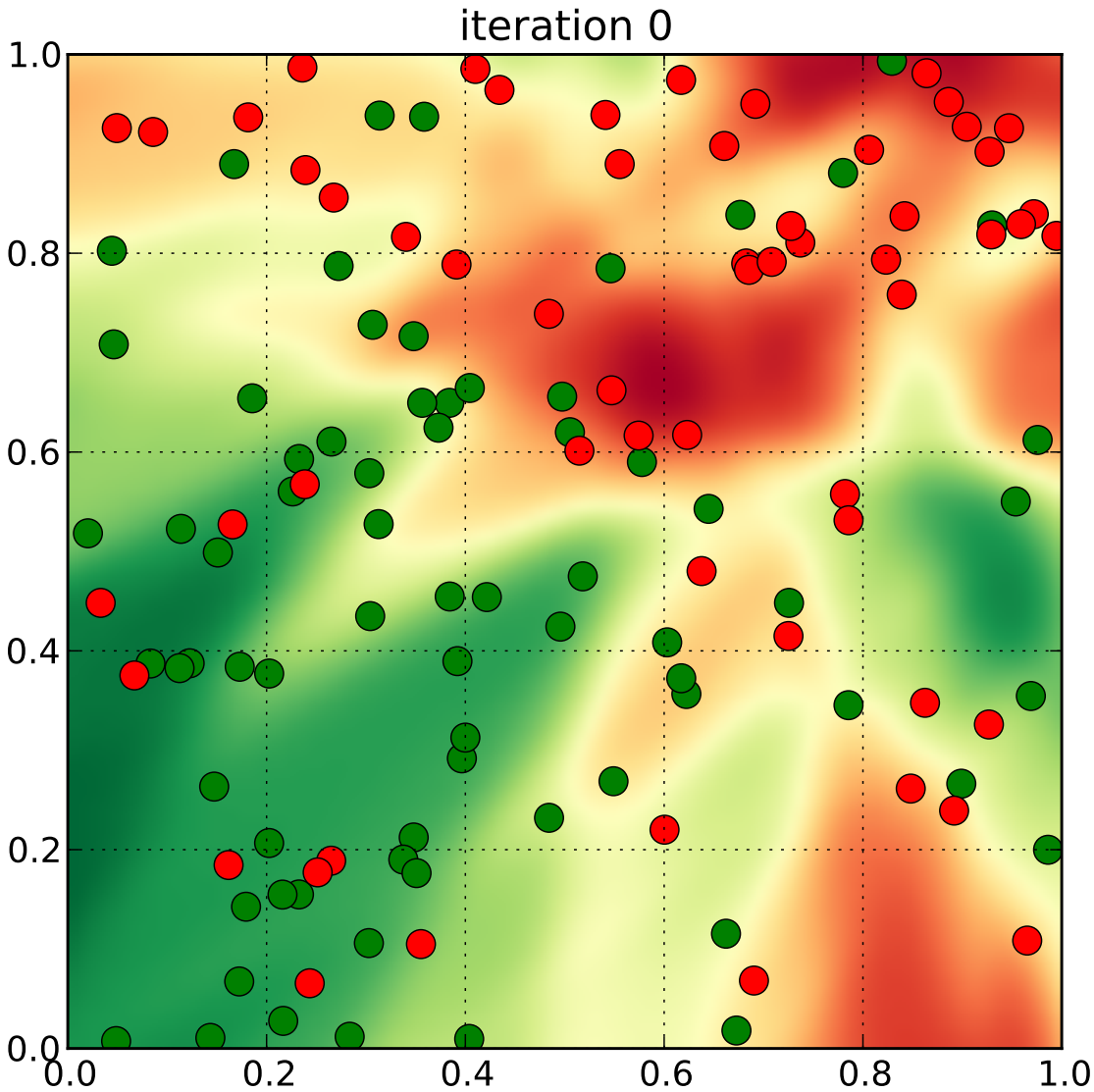
Variations to Backpropagation: ADAM [3] (Chapter 8)

Adam (="Adaptive Moments")

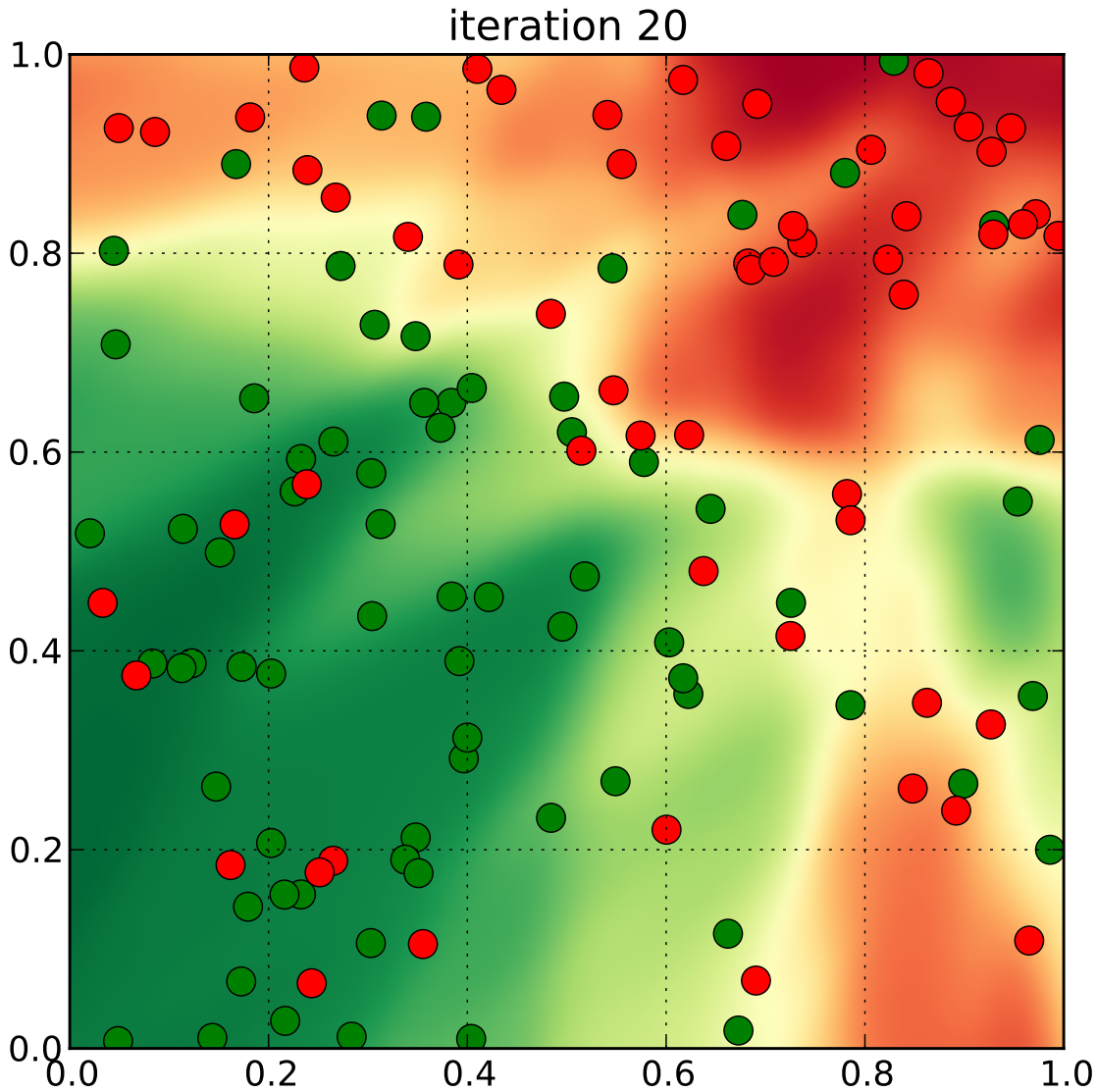
- Like RMSProp: uses dimension-specific multiplier (see $\mathbf{r}$)
- Unlike RMSProp: uses momentum (see $\mathbf{s}$)
- Unlike RMSProp: bias correction (see $\hat{\mathbf{r}}, \hat{\mathbf{s}}$), since initially $\mathbf{r} = \mathbf{s} = \mathbf{0}$

Outline

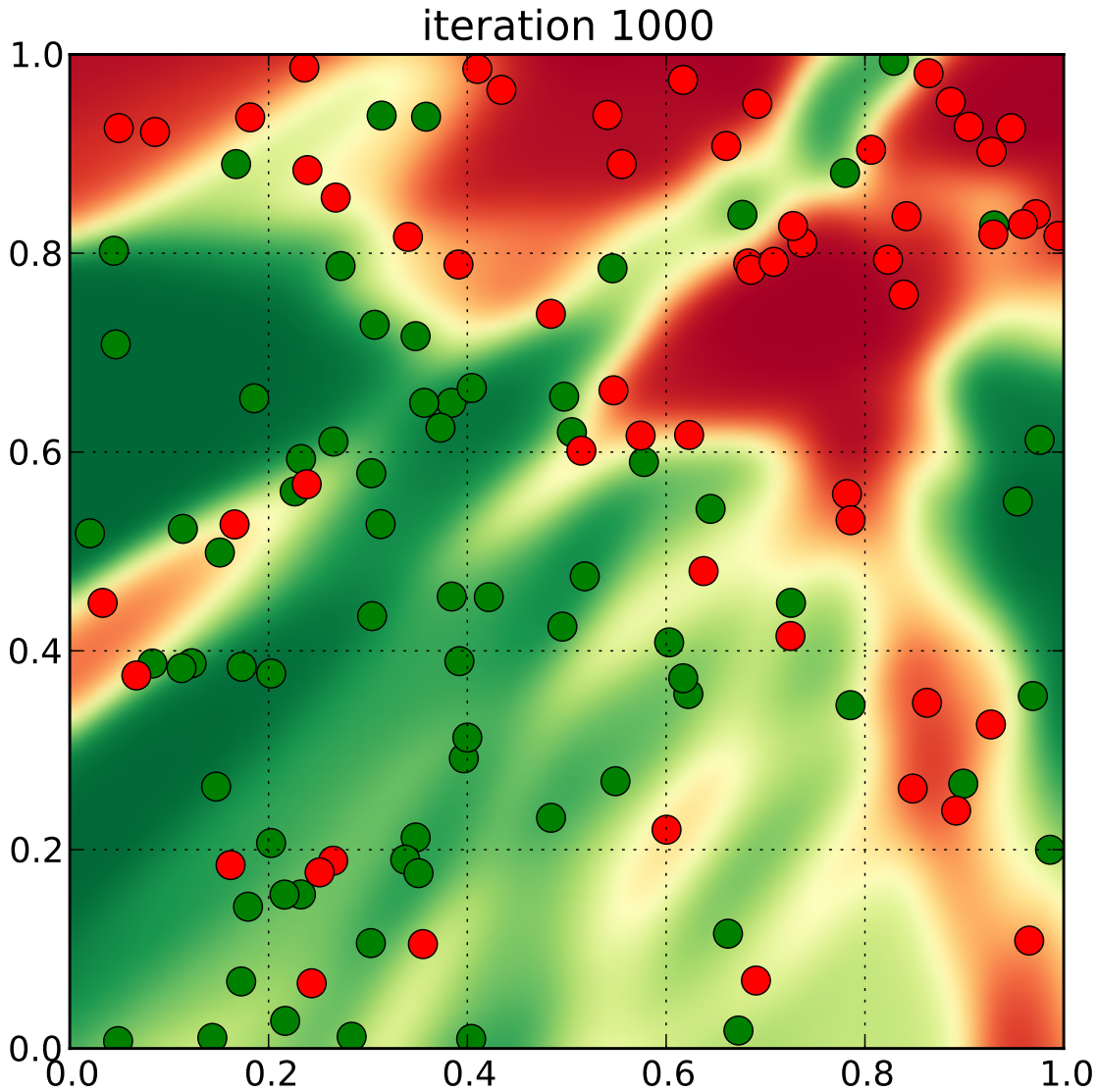
MLP: Training Example (see above)



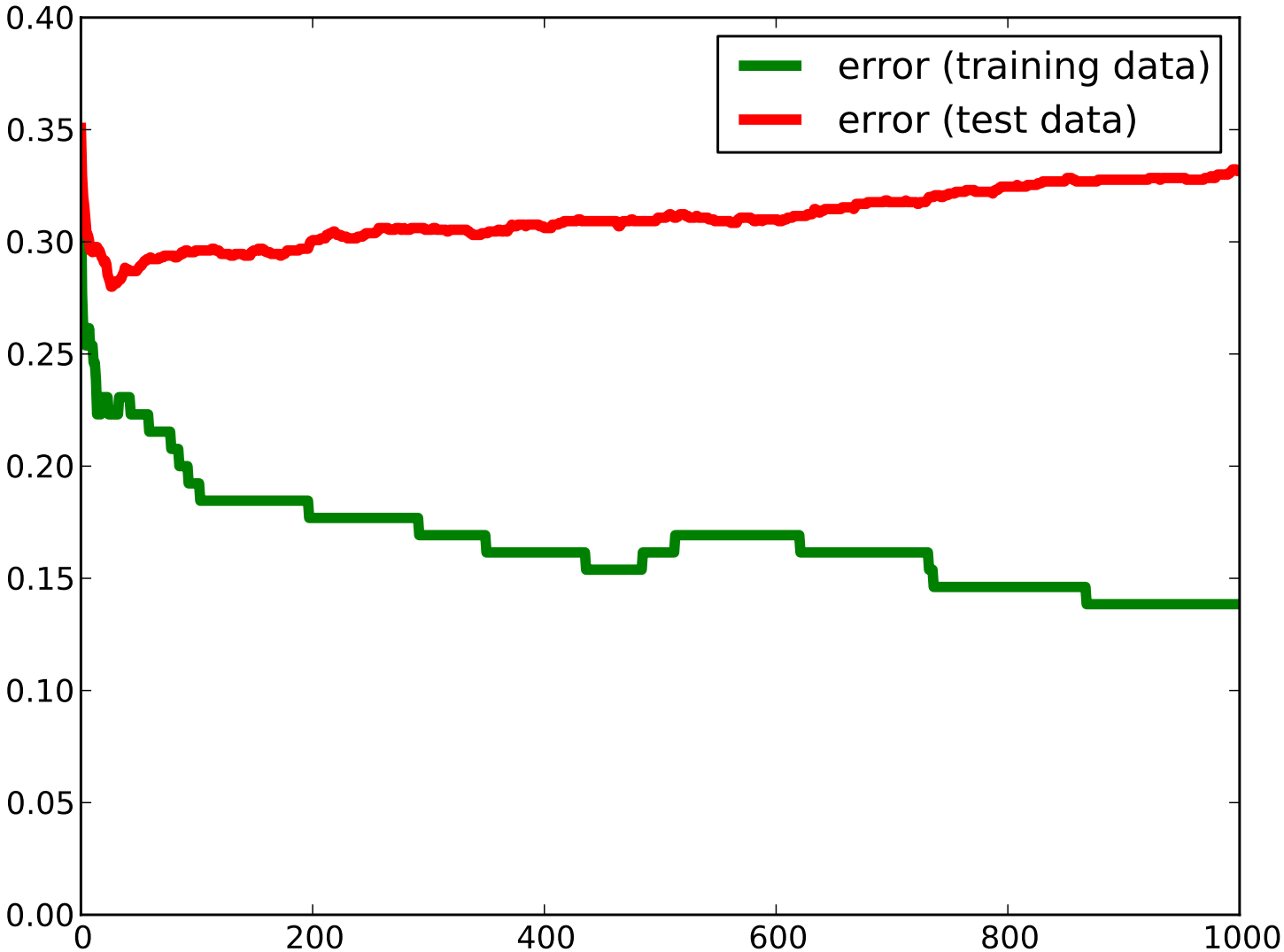
MLP: Training Example (see above)



MLP: Training Example (see above)



MLP: Training Example



Backpropagation: Early Stopping

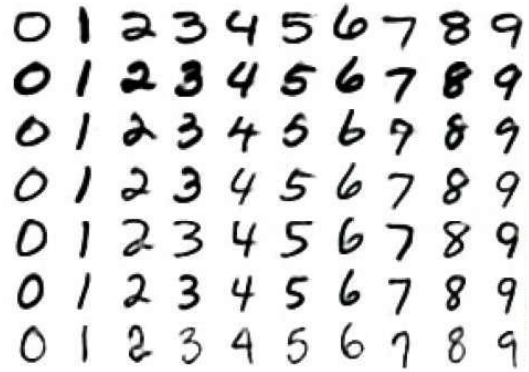
When to stop Backpropagation?

- One iteration through all training data is called an **epoch**.
- Surprisingly, optimizing for as many epochs as possible is not the best answer! *The reason:* **Overfitting**.
- During optimization, record the error rate on held-out **validation set**.
- Stop backpropagation once validation set error does not improve **significantly**.

Remarks

- The 'right' number of epochs varies depending on training data size, number of hidden units, noise, the underlying distributions, and the starting point of optimization!

Weight Initialization



Symmetry Breaking

- What went wrong in the MNIST Example above?
- All weights and biases were initialized **with zeros**!

- All neurons of a layer get exactly the **same updates**
- Effectively, the network learns only **one neuron per layer**!
- Even with small random initialization, different neurons may end up learning the same features/weights :-(
- We need **strong** randomization for **symmetry breaking**!

Weight Initialization

- So we initialize with **super-large** random numbers, right? Let's draw weights W from $[-1000, 1000]$.
- What happens with the **sigmoid's activation** f if the weights become very high?

- **Goal**: Initialize the weights such that we are close to the *'interesting' parts* of the activation function!

Weight Initialization

- Larger initial weights will yield better symmetry breaking.
- Smaller initial weights lead to more effective learning.
- Balancing these effects is **crucial**!

Sample Heuristics

1. Given a layer with $m(n)$ input(output) neurons, initialize [2]:

$$W_{ij} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right) \quad \text{for all edges (i,j) in the layer}$$

2. **Sparse initialization**: each neuron gets exactly k (randomly sampled) non-zero inputs *(avoids small weights for big layers)*
3. **Grid search**: treat initialization as a hyperparameter and search for a good value using cross-validation
4. **Inspection**: look at activations of the different layers. Where the variance of activations is low, increase the variance of initial weights.

Backpropagation: Training Data Issues

Balancing Training Data

- Sometimes, we can collect many samples from one class but only few from another
- Example (spam filtering): Let 90% of all e-mails be spam. An MLP classifying any mail as spam has an error of only 10% $\rightarrow$ **local minimum!**
- We call such training data **unbalanced**
- To avoid local minima when learning, we usually balance training data: We use (about) equally many training samples from any class.

Normalizing Training Data

- For large absolute input values, the sigmoid is very flat $\rightarrow$ **slow learning**!
- **Common Strategy**: normalize training samples to be ≈ 0

References

[1] The MNIST Database of Handwritten Digits.
http://yann.lecun.com/exdb/mnist/ (retrieved: Oct 2016).

[2] X. Glorot and Y. Bengio.
Understanding the Difficulty of Training Deep Feedforward Neural Networks.
In Proc. AISTATS-10, volume 9, pages 249–256, 2010.

[3] I. Goodfellow, Y. Bengio, and A. Courville.
Deep Learning.
Book in preparation for MIT Press (retrieved Nov 2016), 2016.

[4] A. Holehouse.
Stanford Machine Learning (Transcript of Course by Prof. Andrew Ng).
http://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html (retrieved: Nov 2016).

[5] X. e. a. Zhang.
Appearance-based Gaze Estimation in the Wild (Video from Project Webpage).
https://www.mpi-inf.mpg.de/departments/computer-vision-and-multimodal-computing/research/gaze-based-human-computer-interaction/appearance-based-gaze-estimation-in-the-wild-mpiigaze/ (retrieved: Nov 2016).