

# Deep learning

## Feed-forward neural networks and back-propagation

Ole Winther

Bioinformatics Centre, Department of Biology  
University of Copenhagen (UCph)

Dept for Applied Mathematics and Computer Science  
Technical University of Denmark (DTU)



November 4, 2025

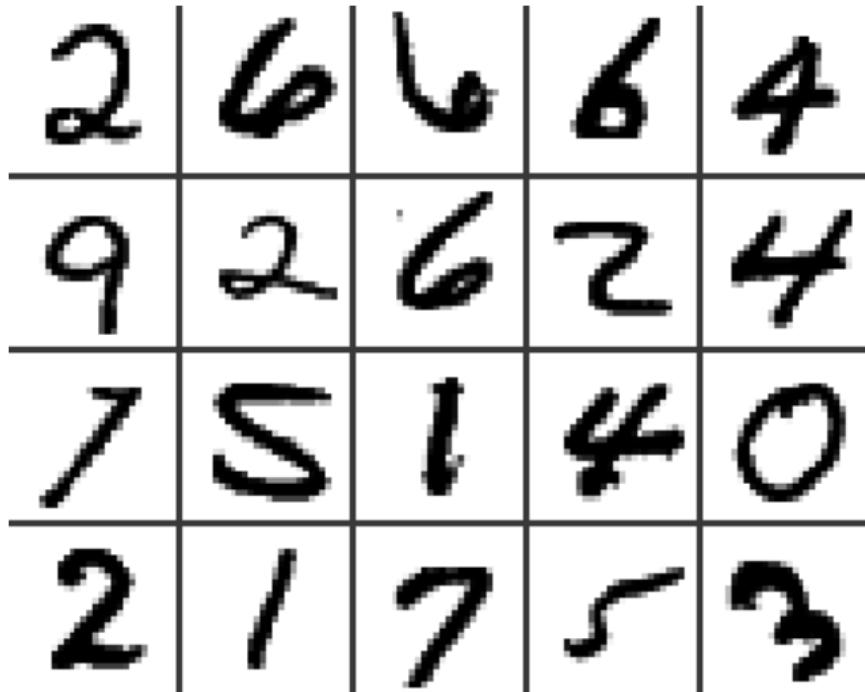
# Objectives of lecture

- Feed-forward neural network (FFNN)
- Training with **error back-propagation**
- We only need to understand the principles
- **Autograd** - automated differentiation handles the derivation for us!



# Part 1: Short recap

## Example: MNIST handwritten digits

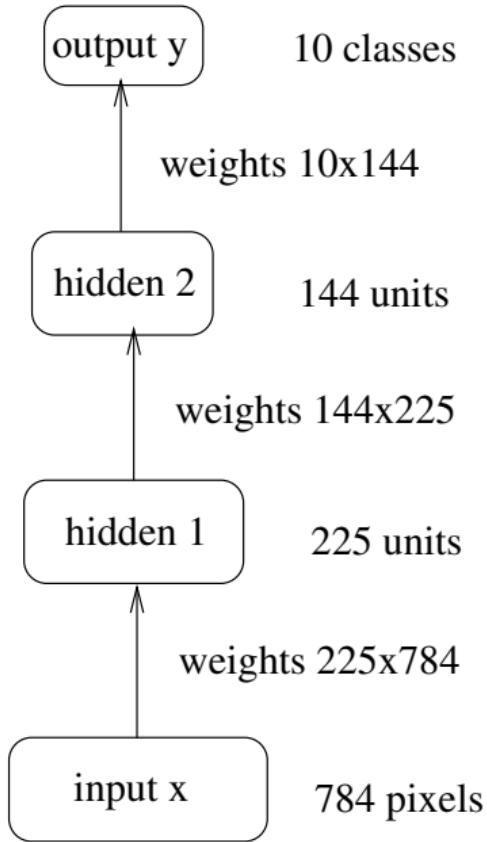


Train a network to classify  $28 \times 28$  images.

Data: 60000 input images  $\mathbf{x}_n$  and labels  $y_n$ ,  $n = 1, \dots, 60k$ .

Example model gives around 1.2% test error.

# Example Network



$$\mathbf{y} = \text{softmax}(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)})$$

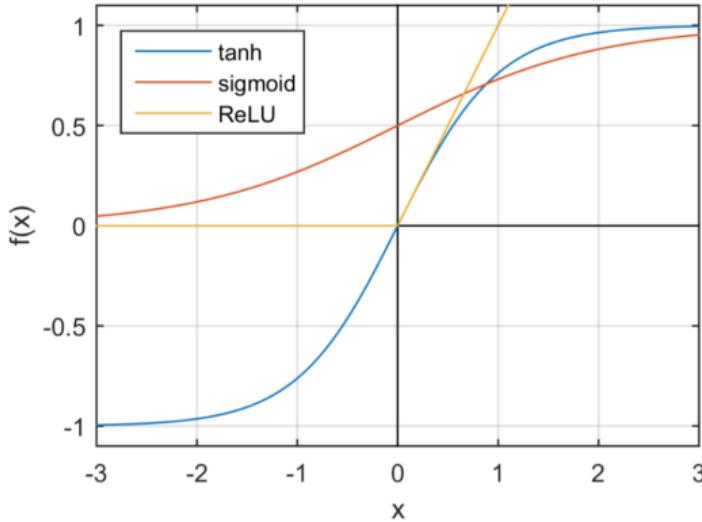
$$\mathbf{h}^{(2)} = \text{relu}(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

$$\mathbf{h}^{(1)} = \text{relu}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

$$\text{relu}(z) = \max(0, z)$$

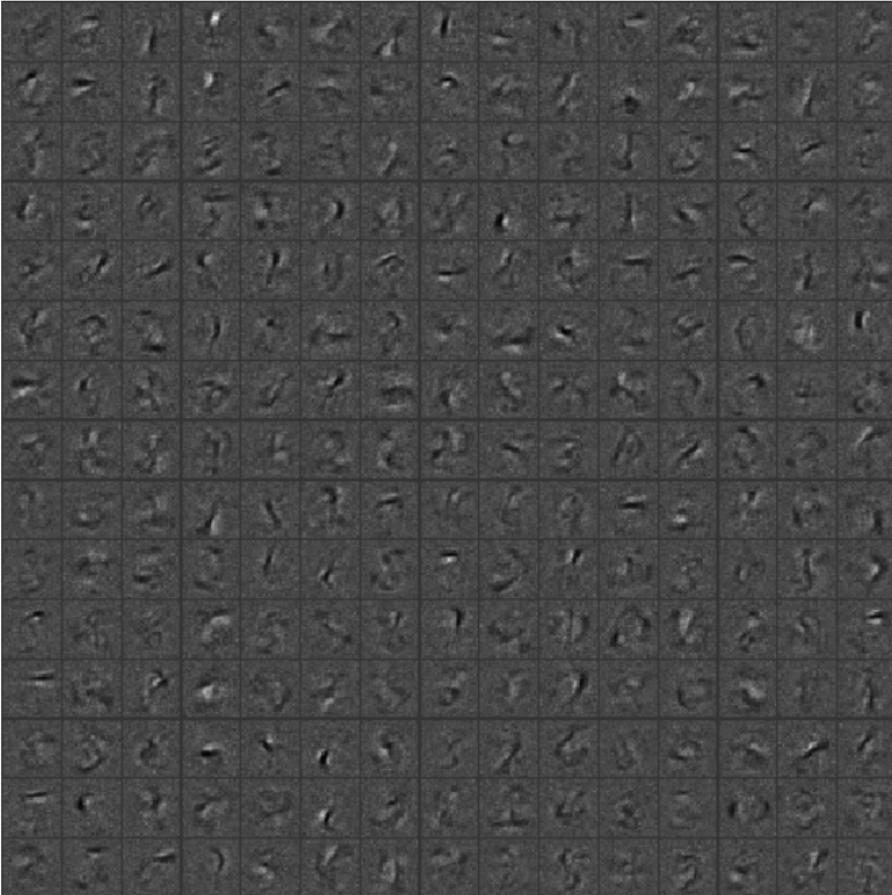
# On activation functions



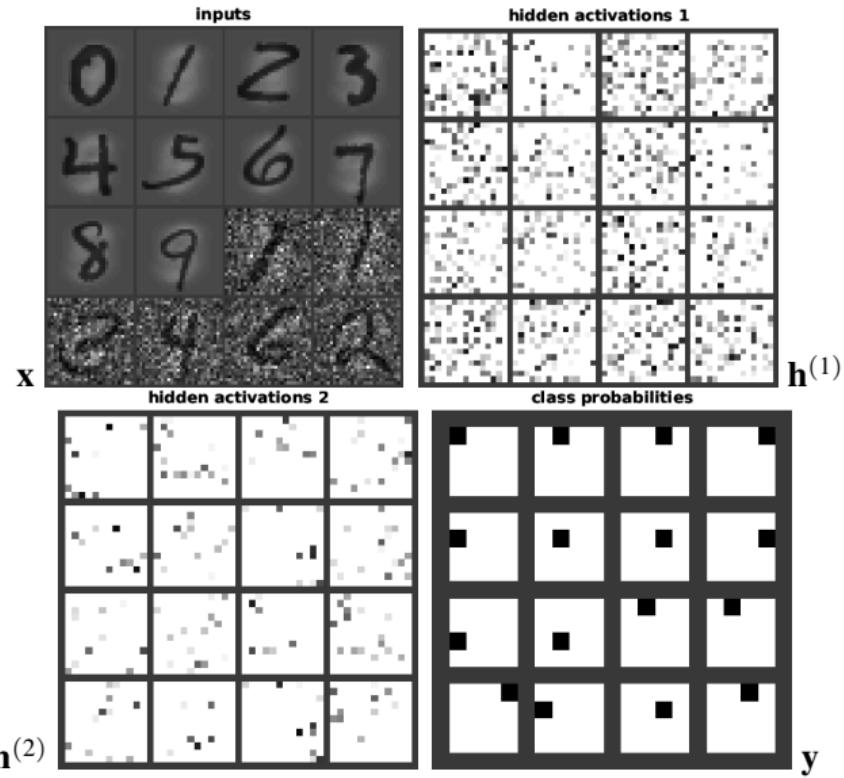
- $\text{relu}(z) = \max(0, z)$  is replacing old sigmoid and tanh.
- Note that identity function would lead into:

$$\begin{aligned}\mathbf{h}^{(2)} &= \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)} \\ &= \mathbf{W}^{(2)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)} \\ &= (\mathbf{W}^{(2)}\mathbf{W}^{(1)})\mathbf{x} + (\mathbf{W}^{(2)}\mathbf{b}^{(1)} + \mathbf{b}^{(2)}) \\ &= \mathbf{W}'\mathbf{x} + \mathbf{b}'\end{aligned}$$

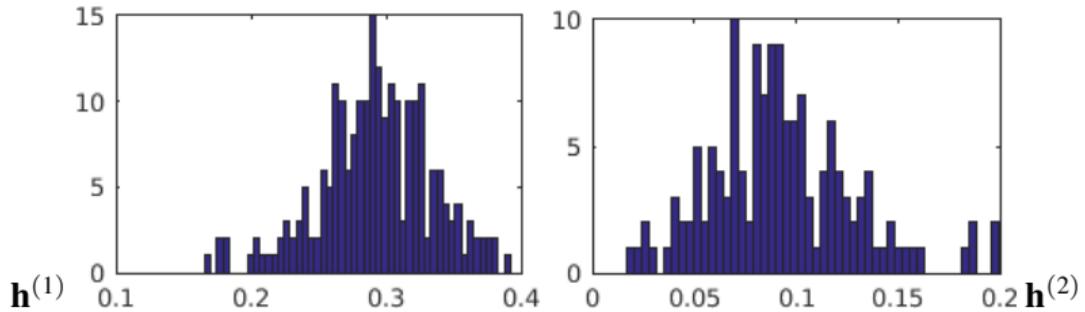
Weight matrix  $\mathbf{W}^{(1)}$  size  $225 \times 784$



Signals  $\mathbf{x} \rightarrow \mathbf{h}^{(1)} \rightarrow \mathbf{h}^{(2)} \rightarrow \mathbf{h}^{(3)}$



# On sparsity



How often  $h_i > 0$ ? Histogram over units  $i$ .  
(Sometimes units become completely dead.)

# Part 2: Neural network training

# Training criterion

Say we have a true distribution  $P(\mathbf{t} \mid \mathbf{x})$  and we would like to find a model  $Q(\mathbf{t} \mid \mathbf{x}, \boldsymbol{\theta})$  that matches  $P$ . Let us study how maximizing expected negative log-likelihood  $C = \mathbb{E}_P [-\log Q]$  works as a learning criterion.

Find parameters

$$\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^L$$

that minimize expected negative log-likelihood:

$$C = - \sum_{i=1}^n \log P(\mathbf{t}_i | \mathbf{x}_i, \boldsymbol{\theta}) .$$

Learning becomes optimization.

Let us assume that there is a  $\boldsymbol{\theta}^*$  for which  $Q(\mathbf{t} | \mathbf{x}, \boldsymbol{\theta}^*) = P(\mathbf{t} | \mathbf{x})$ . We can note that the gradient at  $\boldsymbol{\theta}^*$

$$\begin{aligned} & \frac{\partial}{\partial \boldsymbol{\theta}} \mathbb{E}_P(\mathbf{t} | \mathbf{x}) [\log Q(\mathbf{t} | \mathbf{x}, \boldsymbol{\theta}^*)] \\ &= \mathbb{E}_P(\mathbf{t} | \mathbf{x}) \left[ \frac{\partial}{\partial \boldsymbol{\theta}} \log Q(\mathbf{t} | \mathbf{x}, \boldsymbol{\theta}^*) \right] \\ &= \int P(\mathbf{t} | \mathbf{x}) \frac{\frac{\partial}{\partial \boldsymbol{\theta}} Q(\mathbf{t} | \mathbf{x}, \boldsymbol{\theta}^*)}{Q(\mathbf{t} | \mathbf{x}, \boldsymbol{\theta}^*)} d\mathbf{t} \\ &= \int \frac{\partial}{\partial \boldsymbol{\theta}} Q(\mathbf{t} | \mathbf{x}, \boldsymbol{\theta}^*) d\mathbf{t} \\ &= \frac{\partial}{\partial \boldsymbol{\theta}} \int Q(\mathbf{t} | \mathbf{x}, \boldsymbol{\theta}^*) d\mathbf{t} = \frac{\partial}{\partial \boldsymbol{\theta}} 1 = 0 \end{aligned}$$

becomes zero, that is, the learning converges when  $Q = P$ . Therefore the expected log-likelihood is a reasonable training criterion.

# Classification - one hot encoding and cross-entropy

- MNIST, output labels:  $0, 1, \dots, 9$ .
- Convenient to use a sparse one hot encoding:

$$0 \rightarrow \mathbf{t} = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T$$

$$1 \rightarrow \mathbf{t} = (0, 1, 0, 0, 0, 0, 0, 0, 0, 0)^T$$

$$2 \rightarrow \mathbf{t} = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0)^T$$

...

$$9 \rightarrow \mathbf{t} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1)^T$$

- Output

$$P(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta}) = \mathbf{y} = \text{softmax}(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)})$$

interpreted as class(-conditional) probability.

- Cross-entropy cost - sum over **data** and **label**

$$C = - \sum_{n=1}^N \sum_{k=1}^K t_{nk} \log y_{nk}$$

# Gradient descent

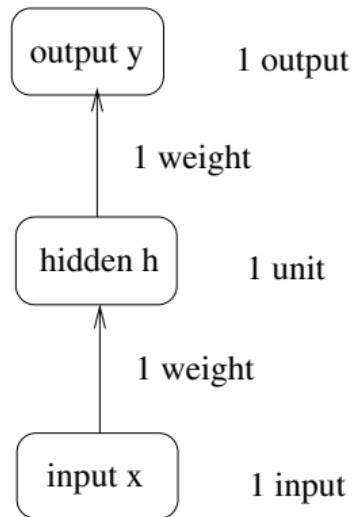
- Simple algorithm for minimizing the training criterion  $C$ .

- Gradient  $\mathbf{g} = \nabla_{\theta} C(\theta) = \begin{pmatrix} \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_n} \end{pmatrix}$

- Iterate  $\theta_{k+1} = \theta_k - \eta_k \mathbf{g}_k$
- Notation: iteration  $k$ , stepsize (or learning rate)  $\eta_k$

# Backpropagation (Linnainmaa, 1970)

Computing gradients in a network.



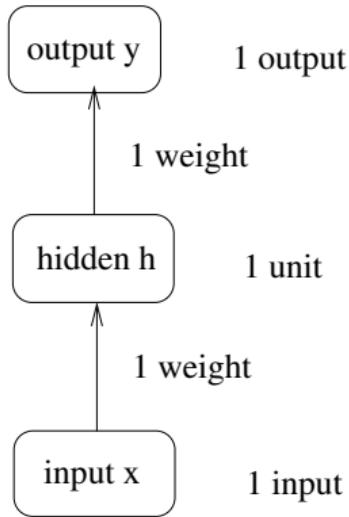
- First with scalars. Use chain rule:

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial w_2}$$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial w_1}$$

- Chain rule:  $\frac{\partial h^{(2)}}{\partial x} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial x}$

# Tiny example I



- Chain rule:

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial w_2}$$
$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial w_1}$$

- Example:  $C = (t - h^{(2)})^2$

$$h^{(2)} = w_2 h^{(1)}$$

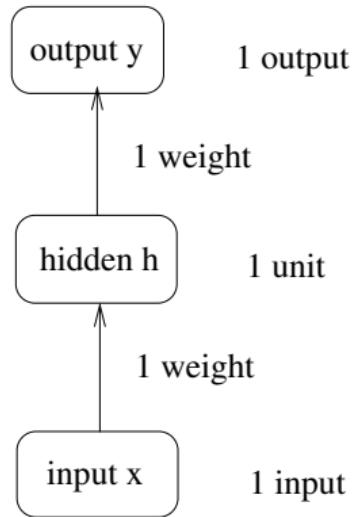
$$h^{(1)} = \text{relu}(w_1 x) = \max(0, w_1 x)$$

- Derivatives:

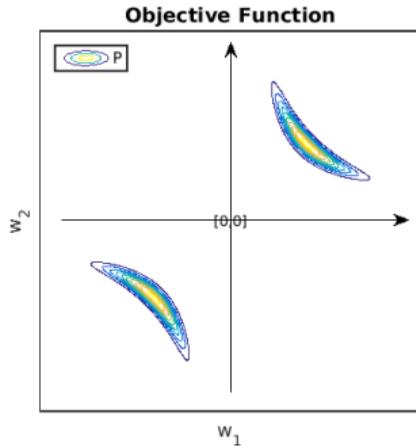
$$\frac{\partial C}{\partial w_2} = -(t - h^{(2)}) h^{(1)}$$

$$\frac{\partial C}{\partial w_1} = -(t - h^{(2)}) w_2 \mathbf{1}(w_1 h^{(1)} > 0) x$$

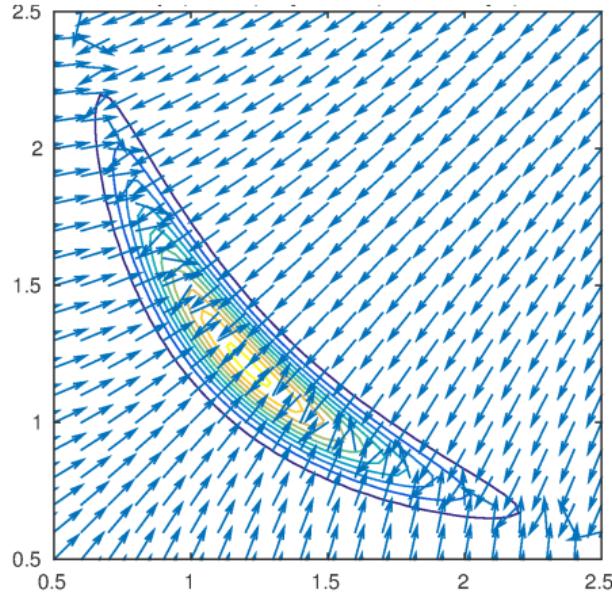
## Tiny Example II



- $t \sim \mathcal{N}(w_2 h, 1)$
- $h = w_1 x$
- “Data set”:  $\{x = 1, t = 1.5\}$
- Some weight decay.
- $C = (w_1 w_2 - 1.5)^2 + 0.04(w_1^2 + w_2^2)$

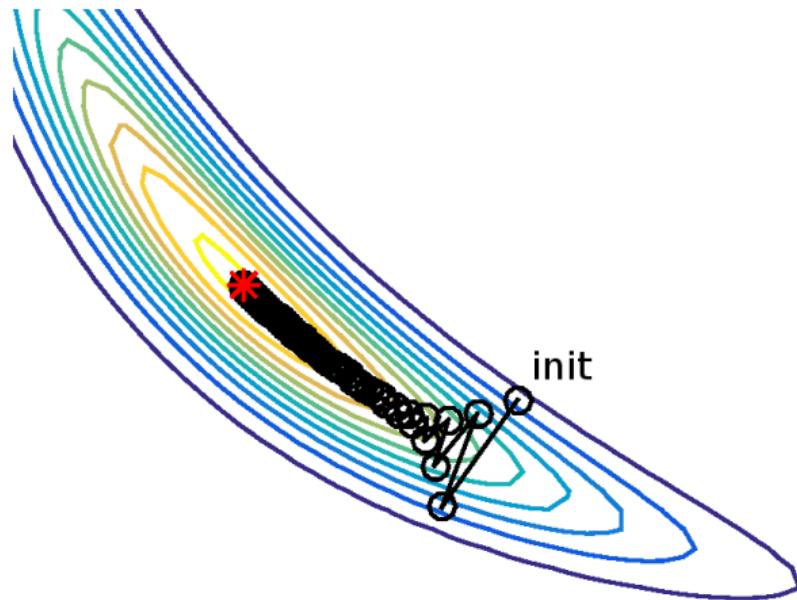


$$\text{Gradient } \mathbf{g} = \nabla_{\theta} C(\theta) = \begin{pmatrix} \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_n} \end{pmatrix}$$



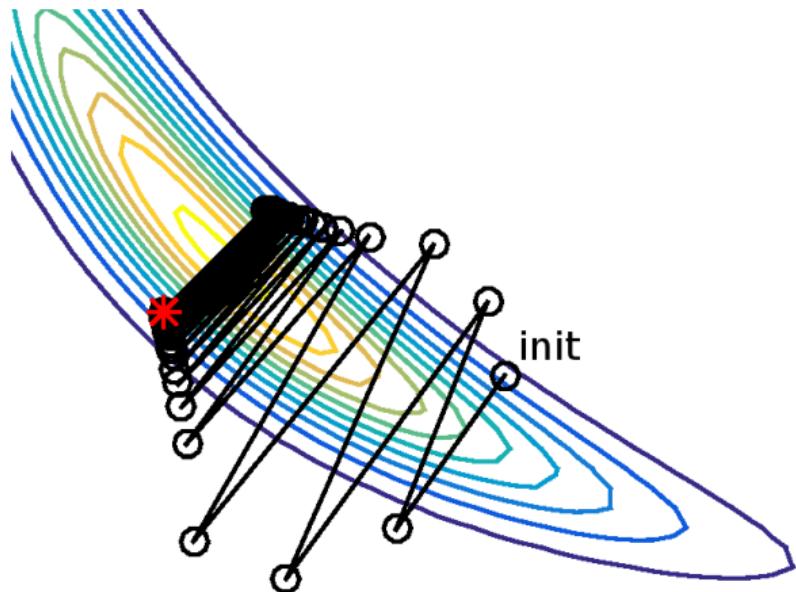
## Gradient descent, $\eta_k = 0.25$ ( $\rightarrow$ too slow)

$\theta_{k+1} = \theta_k - \eta_k \mathbf{g}_k$ , iteration  $k$ , stepsize (or learning rate)  $\eta_k$



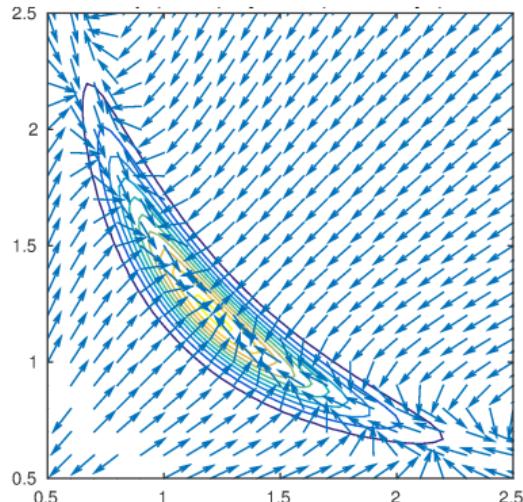
## Gradient descent, $\eta_k = 0.35$ ( $\rightarrow$ oscillates)

$$\theta_{k+1} = \theta_k - \eta_k \mathbf{g}_k$$



## Newton's method, too complex

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}_k^{-1} \mathbf{g}_k, \quad \mathbf{H} = \begin{pmatrix} \frac{\partial^2 C}{\partial \theta_1 \partial \theta_1} & \cdots & \frac{\partial^2 C}{\partial \theta_1 \partial \theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 C}{\partial \theta_n \partial \theta_1} & \cdots & \frac{\partial^2 C}{\partial \theta_n \partial \theta_n} \end{pmatrix}$$

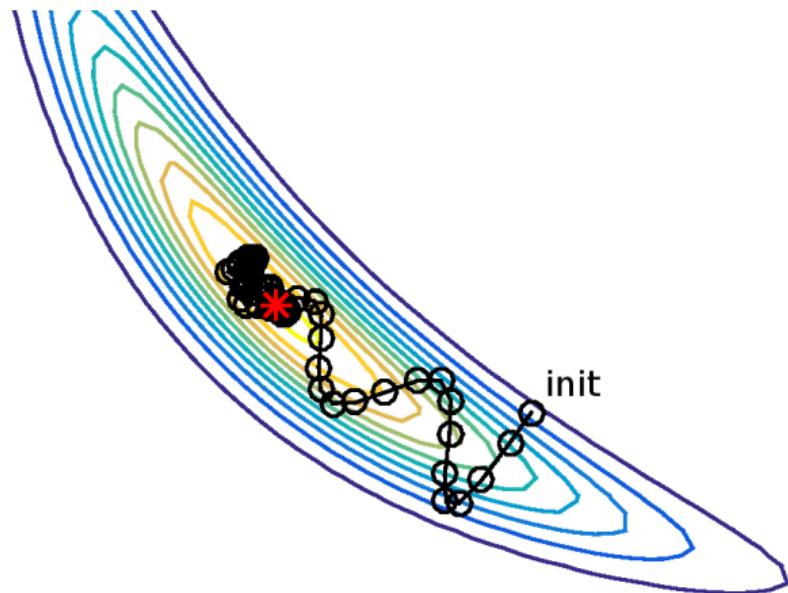


- Less oscillations.
- Points to the wrong direction in places (solvable).
- Computational complexity:  $\# \text{params}^3$  (prohibitive).
- There are approximations, but not very popular.

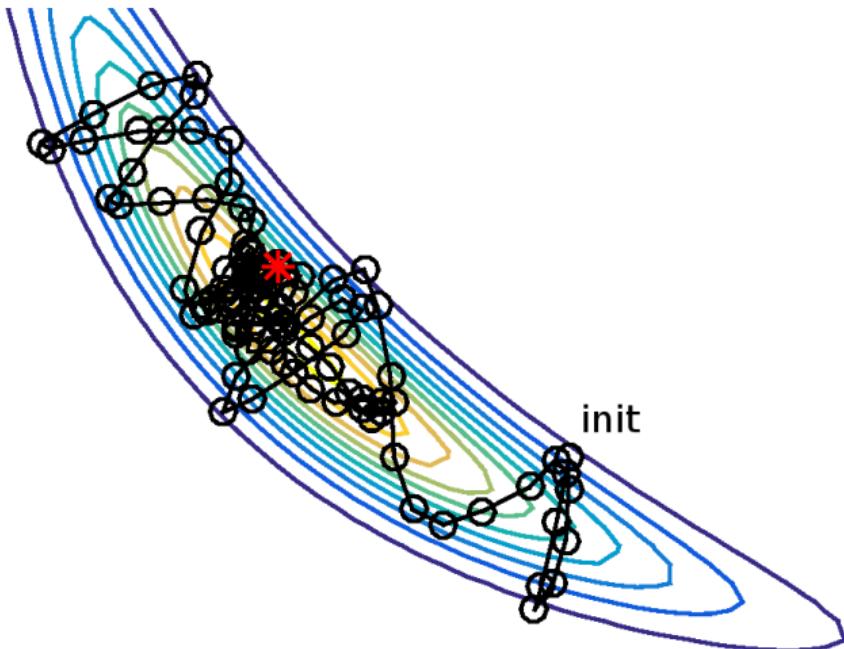
## Momentum method (Polyak, 1964)

$$\mathbf{m}_{k+1} = \alpha \mathbf{m}_k - \eta_k \mathbf{g}_k$$

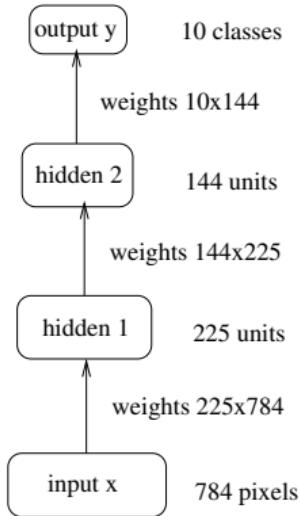
$$\theta_{k+1} = \theta_k + \mathbf{m}_{k+1}$$



## Momentum method with noisy gradient



# Backpropagation



- Multi-dimensional:

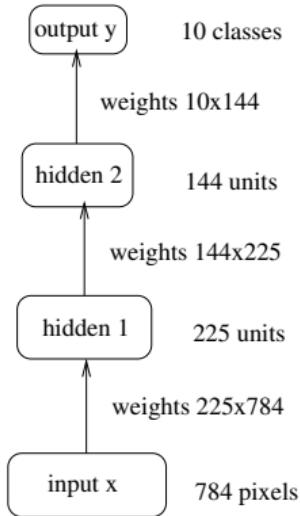
$$\frac{\partial C}{\partial W_{ij}^{(3)}} = \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial W_{ij}^{(3)}}$$

$$\frac{\partial C}{\partial W_{jk}^{(2)}} = \sum_i \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial h_j^{(2)}} \frac{\partial h_j^{(2)}}{\partial W_{jk}^{(2)}}$$

$$\frac{\partial C}{\partial W_{kl}^{(1)}} = \sum_j \sum_i \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial h_j^{(2)}} \frac{\partial h_j^{(2)}}{\partial h_k^{(1)}} \frac{\partial h_k^{(1)}}{\partial W_{kl}^{(1)}}$$

- *How many paths - for two hidden layers*
- *as a function of depth?*
- **Modern software (autograd) handles this derivation!**

# Backpropagation



- Multi-dimensional:

$$\frac{\partial C}{\partial W_{ij}^{(3)}} = \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial W_{ij}^{(3)}}$$

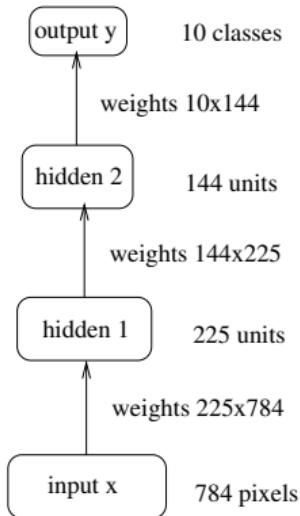
$$\frac{\partial C}{\partial W_{jk}^{(2)}} = \sum_i \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial h_j^{(2)}} \frac{\partial h_j^{(2)}}{\partial W_{jk}^{(2)}}$$

$$\frac{\partial C}{\partial W_{kl}^{(1)}} = \sum_j \sum_i \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial h_j^{(2)}} \frac{\partial h_j^{(2)}}{\partial h_k^{(1)}} \frac{\partial h_k^{(1)}}{\partial W_{kl}^{(1)}}$$

- *How many paths - for two hidden layers*
- *as a function of depth?*
- *Modern software (autograd) handles this derivation!*

# Backpropagation - dynamic programming

- Store intermediate results



$$\frac{\partial C}{\partial h_j^{(2)}} = \sum_i \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial h_j^{(2)}}$$
$$\frac{\partial C}{\partial h_k^{(1)}} = \sum_j \frac{\partial C}{\partial h_j^{(2)}} \frac{\partial h_j^{(2)}}{\partial h_k^{(1)}}$$

- In general

$$\frac{\partial C}{\partial h_j^{(l)}} = \sum_i \frac{\partial C}{\partial h_i^{(l+1)}} \frac{\partial h_i^{(l+1)}}{\partial h_j^{(l)}}$$

- and gradient:

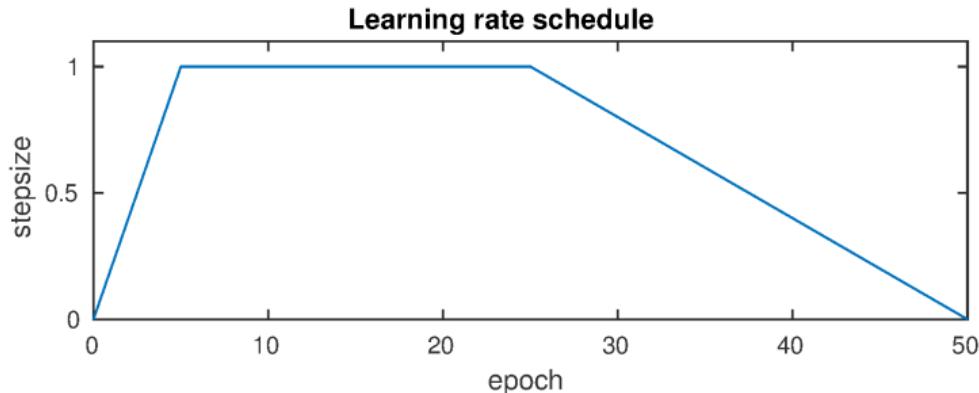
$$\frac{\partial C}{\partial W_{ij}^{(l)}} = \frac{\partial C}{\partial h_i^{(l)}} \frac{\partial h_i^{(l)}}{\partial W_{ij}^{(l)}}$$

## Mini-batch training

- No need to have an accurate estimate of  $\mathbf{g}$ .
- Use only a small batch of training data at once.
- Leads into many updates per epoch (=seeing data once).
- E.g. 600 updates with 100 samples per epoch in MNIST.

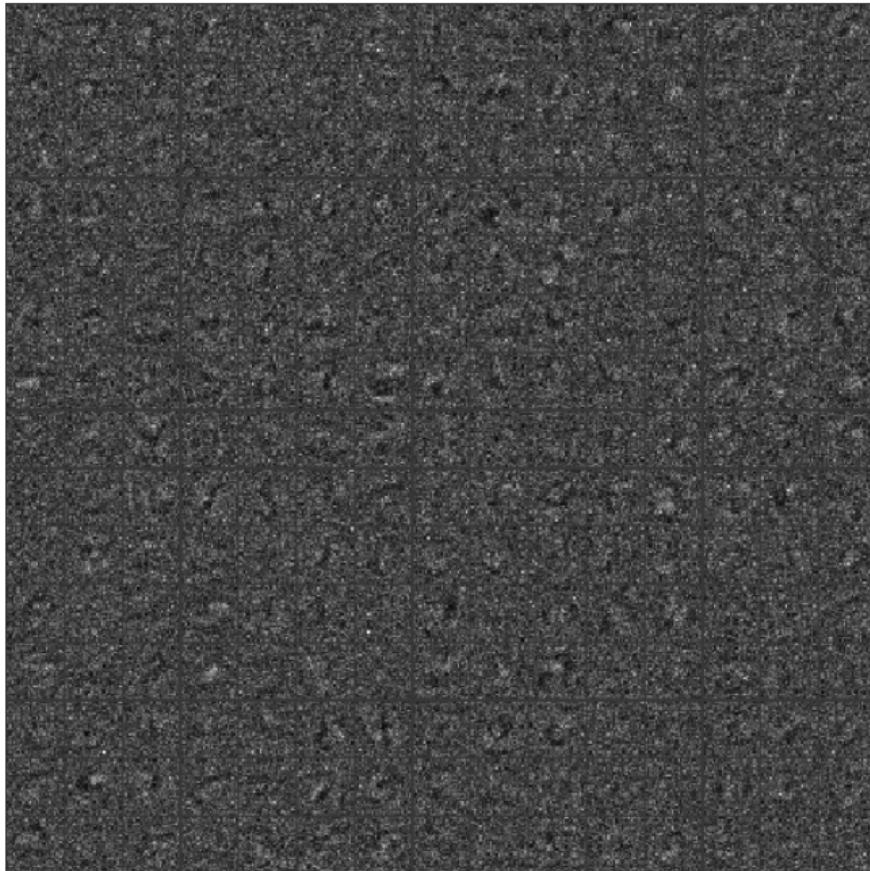
## Mini-batch training

- No need to have an accurate estimate of  $\mathbf{g}$ .
- Use only a small batch of training data at once.
- Leads into many updates per epoch (=seeing data once).
- E.g. 600 updates with 100 samples per epoch in MNIST.
- Important to anneal stepsize  $\eta_k$  towards the end, e.g.

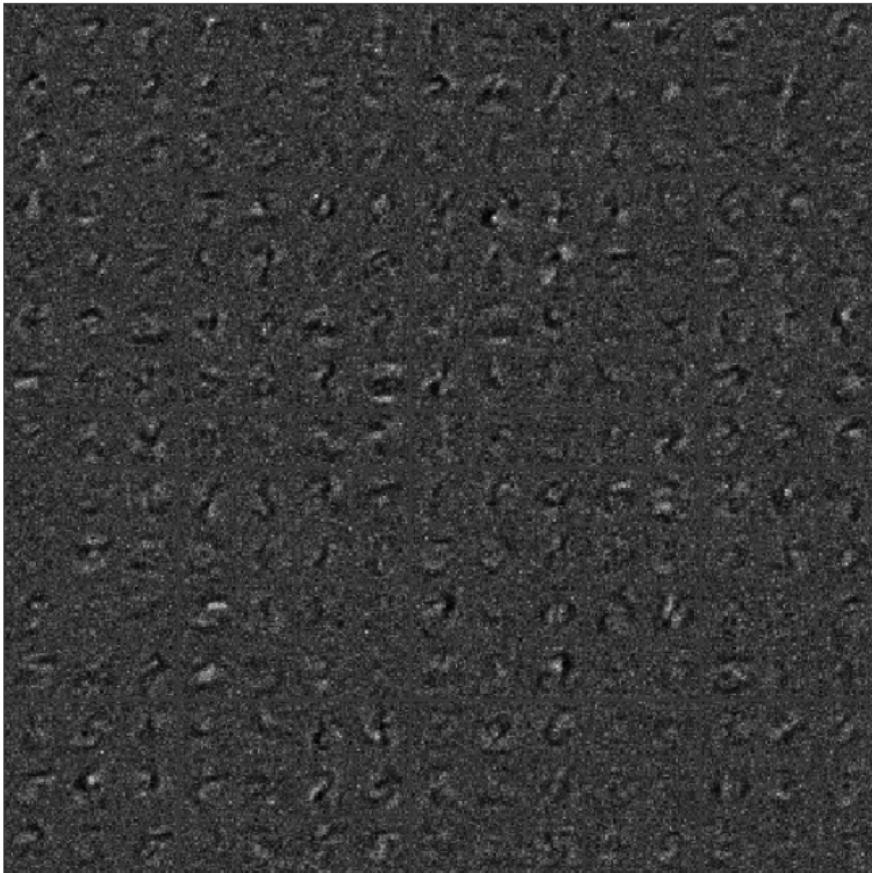


- Adaptation of  $\eta_k$  possible (Adam, Adagrad, Adadelta).

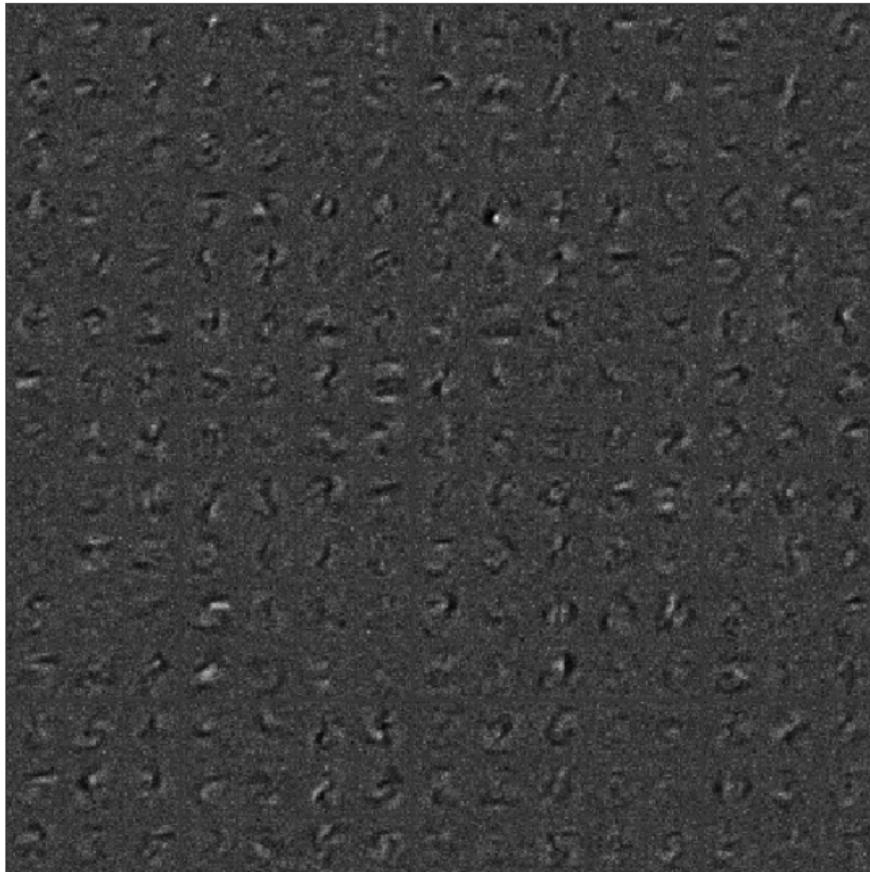
$\mathbf{W}^{(1)}$  after epoch 1



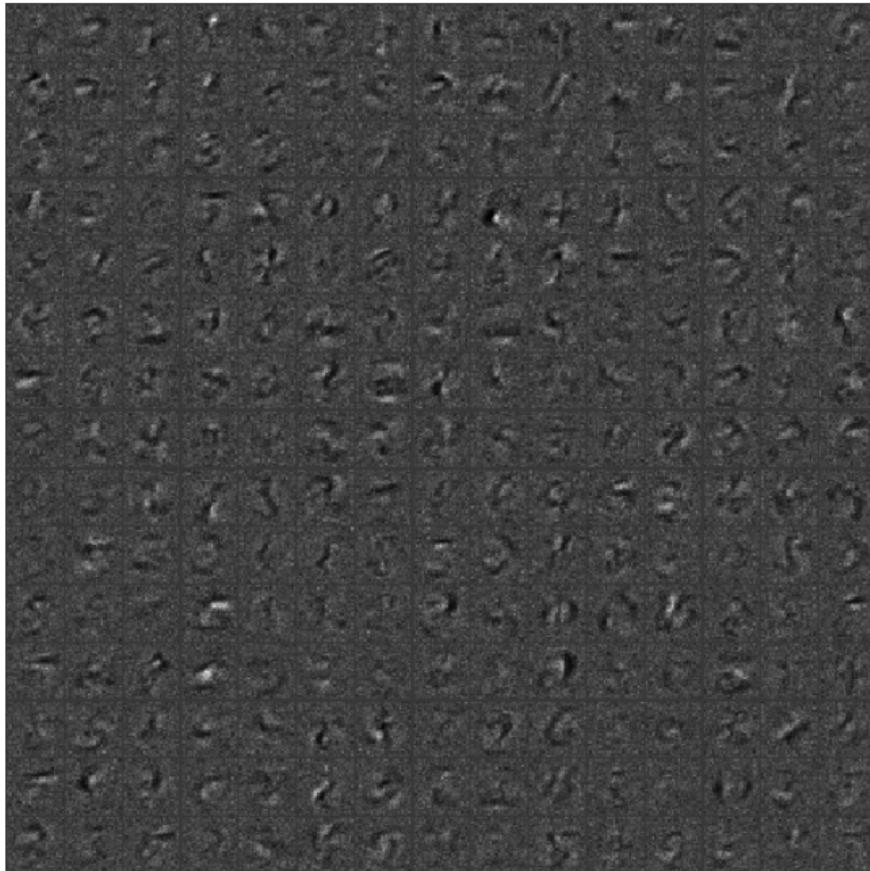
$\mathbf{W}^{(1)}$  after epoch 2



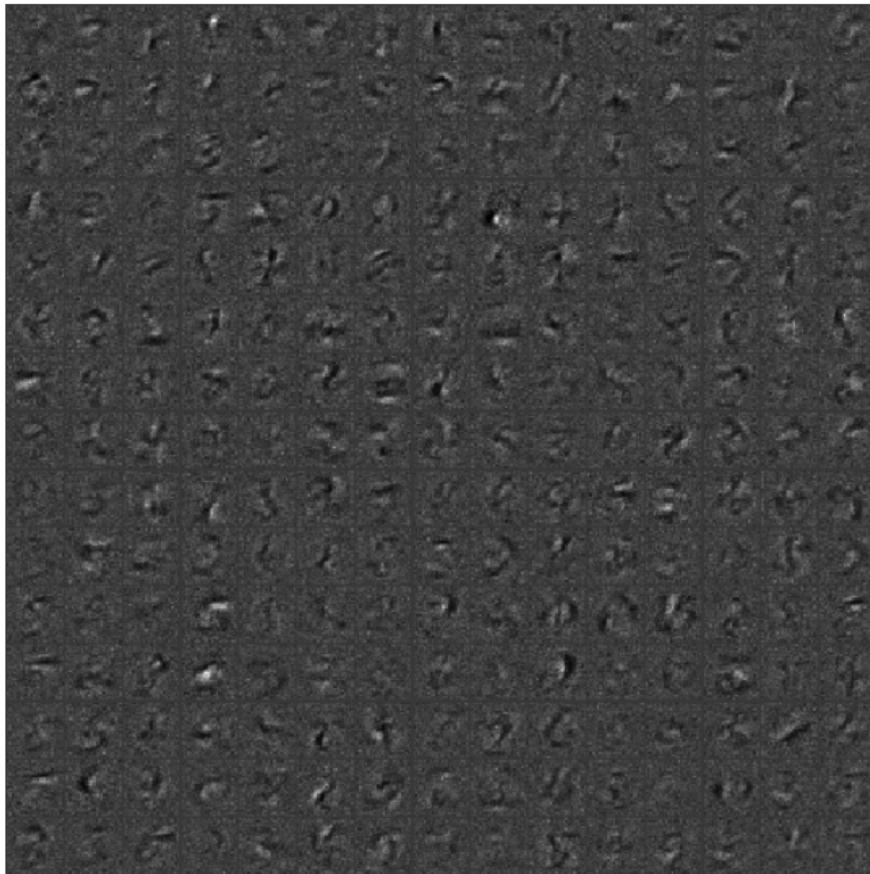
$\mathbf{W}^{(1)}$  after epoch 3



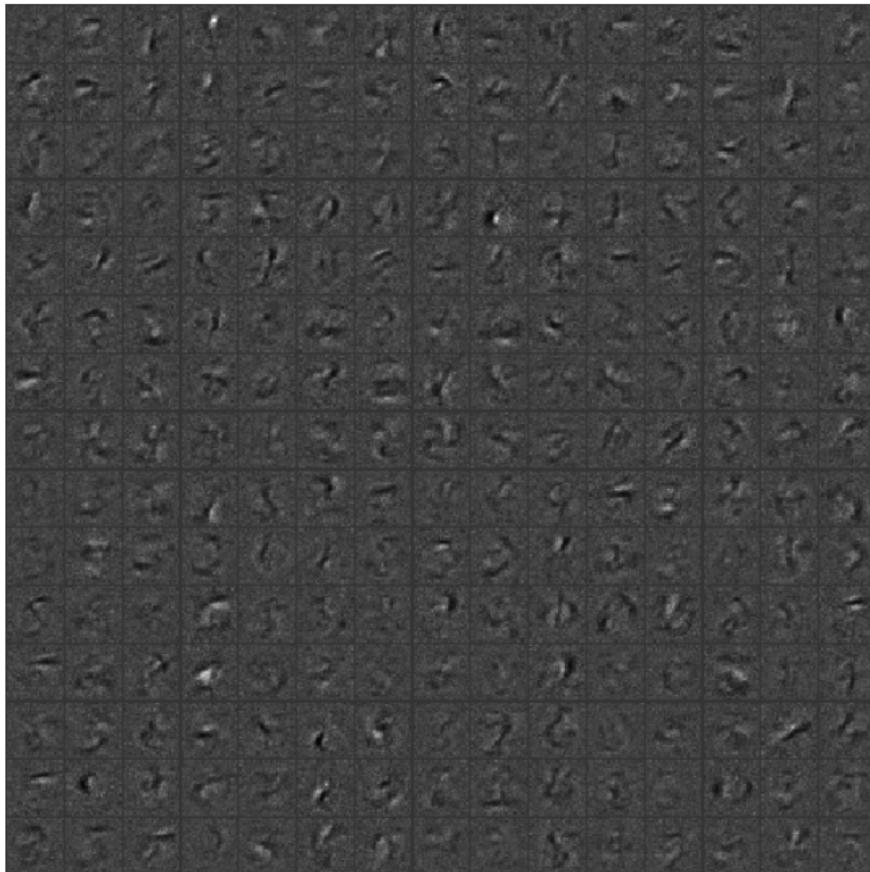
$\mathbf{W}^{(1)}$  after epoch 4



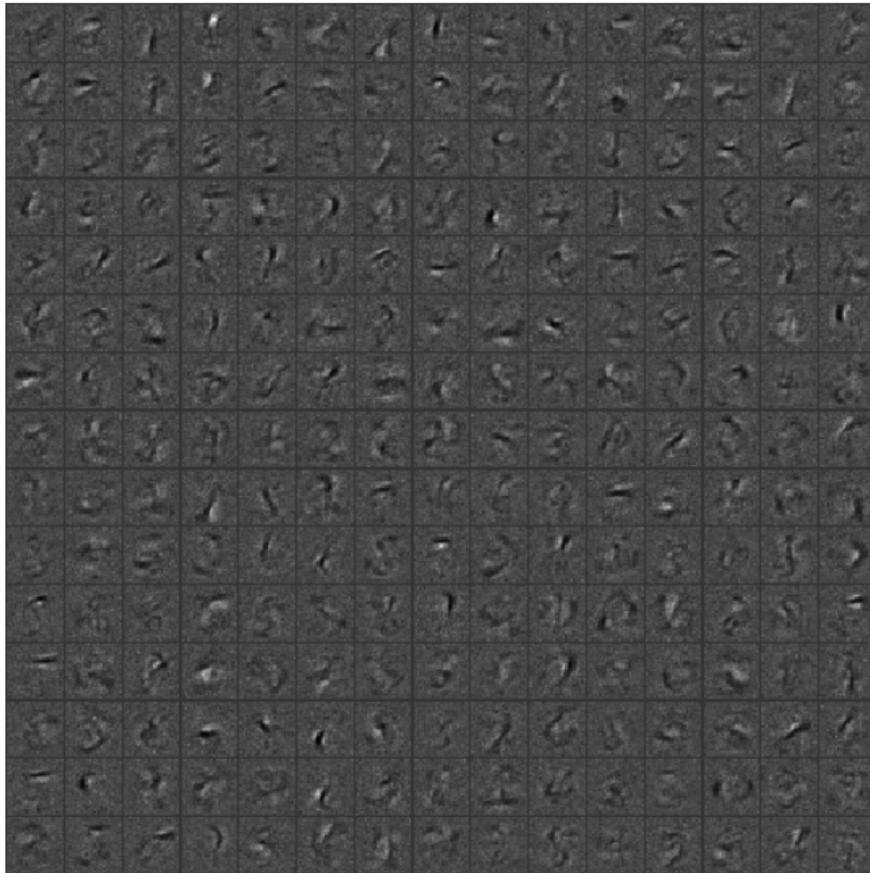
$\mathbf{W}^{(1)}$  after epoch 5



$\mathbf{W}^{(1)}$  after epoch 10

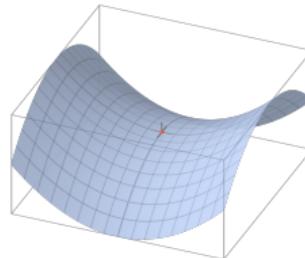


$\mathbf{W}^{(1)}$  after epoch 50 (final)



## Definitions

- Learning stops at a critical point (gradient  $\mathbf{g} = 0$ )
- If all eigenvalues of Hessian  $\mathbf{H}$  are positive it is a local minimum
- If all eigenvalues of Hessian  $\mathbf{H}$  are negative it is a local maximum



- If Hessian has both positive and negative eigenvalues it is a **saddle point**

## Theory: Local minima not an issue

(Dauphin et al., 2014, Choromanska et al., 2015)

- Local minima dominate in low-dimensional optimization, but saddle points dominate in high dimensions
- Most local minima are close to the global minimum
- Noisy gradient  $\mathbf{g}$  helps in escaping saddle points

## Backpropagation, tiny example

$$y = w_2 h$$

$$h = w_1 x$$

$$C = (y - 1.5)^2$$

$$\frac{\partial C}{\partial w_2} = 2(w_2 w_1 x - 1.5) w_1 x$$

$$\frac{\partial C}{\partial w_1} = 2(w_2 w_1 x - 1.5) w_2 x$$

Note a scaling issue: If  $w_1$  is doubled and  $w_2$  is halved,

- output  $y$  stays the same.
- system is twice as sensitive to changes in  $w_2$ .
- gradient of  $w_2$  is doubled!

# Exponential growth/decay forward

- Recall the model

$$\begin{aligned}\mathbf{y} &= \text{softmax}(\mathbf{W}^{(3)} \mathbf{h}^{(2)} + \mathbf{b}^{(3)}) \\ \mathbf{h}^{(2)} &= \text{relu}(\mathbf{W}^{(2)} \mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \\ \mathbf{h}^{(1)} &= \text{relu}(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)})\end{aligned}$$

- Ignoring softmax and biases, we can write

$$y_i = \sum_{j,k,l} \mathbf{1}(h_j^{(2)} > 0) \mathbf{1}(h_k^{(1)} > 0) W_{ij}^{(3)} W_{jk}^{(2)} W_{kl}^{(1)} x_l$$

- Exponential growth/decay of forward signals!

## Exponential growth/decay backward

- Given indicators  $\mathbf{1}(\cdot)$ , model is linear

$$y_i = \sum_{j,k,l} \mathbf{1}\left(h_j^{(2)} > 0\right) \mathbf{1}\left(h_k^{(1)} > 0\right) W_{ij}^{(3)} W_{jk}^{(2)} W_{kl}^{(1)} x_l$$

$$\frac{\partial y_i}{\partial x_l} = \sum_{j,k} \mathbf{1}\left(h_j^{(2)} > 0\right) \mathbf{1}\left(h_k^{(1)} > 0\right) W_{ij}^{(3)} W_{jk}^{(2)} W_{kl}^{(1)}$$

- Exponential growth/decay of gradient, too!
- $\Rightarrow$  Scale of initialization important.

# References

- S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's Thesis (in Finnish), Univ. Helsinki, 1970.
- Polyak, B.T. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.
- Kingma, D. and Ba, J. (2015). ADAM: A method for stochastic optimization. In the International Conference on Learning Representations (ICLR), San Diego. arXiv:1412.6980.
- Dauphin, Pascanu, Gulcehre, Cho, Ganguli, and Bengio. Identifying and attacking the saddle point problem in high dimensional non-convex optimization. In NIPS 2014.
- Choromanska, Henaff, Mathieu, Ben Arous, and LeCun. The Loss Surface of Multilayer Nets. In AISTATS 2015.
- Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." International conference on artificial intelligence and statistics. 2010.
- Martens, J. Deep learning via Hessian-free optimization. In ICML, 2010.



Thanks!  
Ole Winther