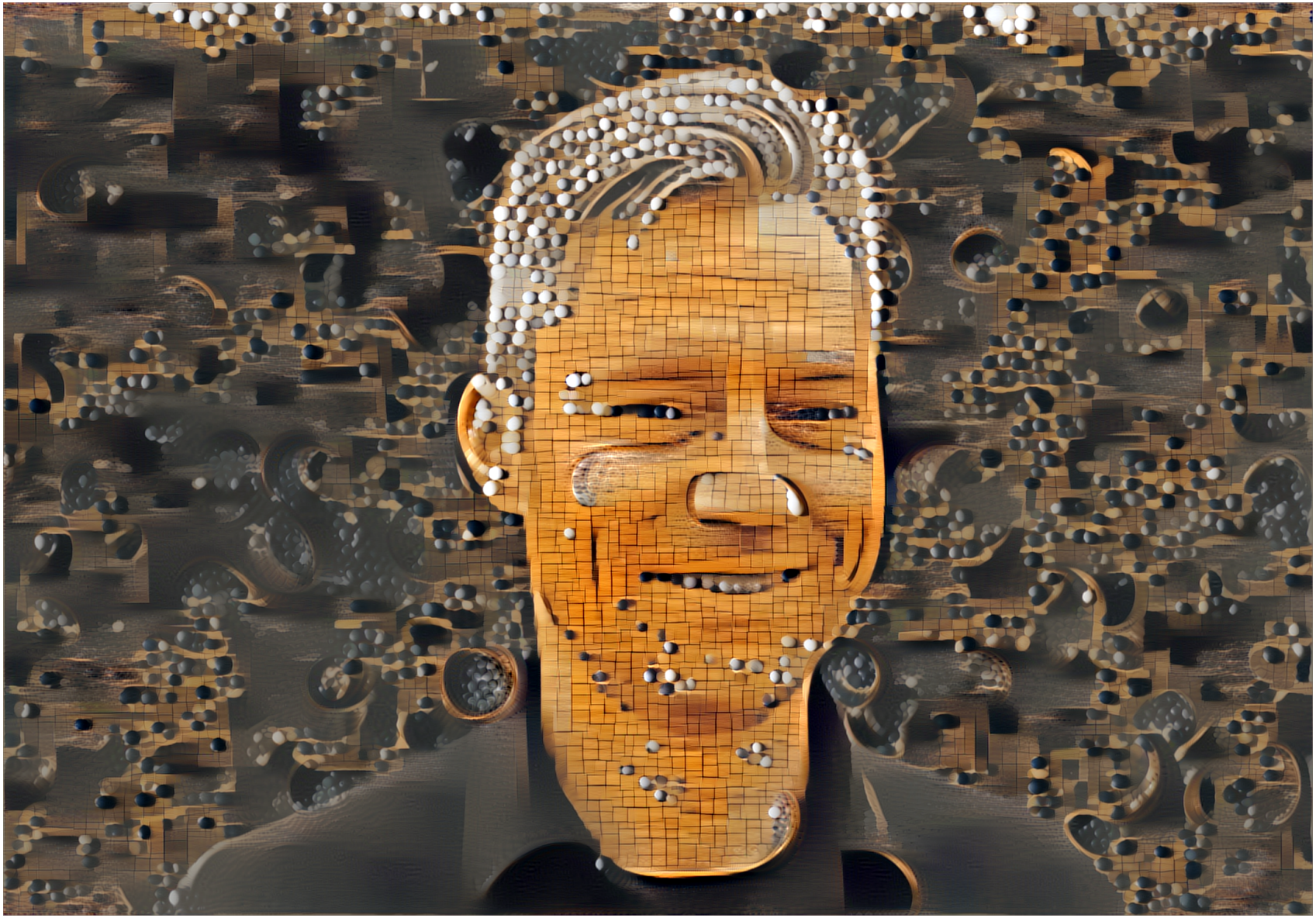
DTU Course 02456 Deep learning
1 Introduction

Ole Winther

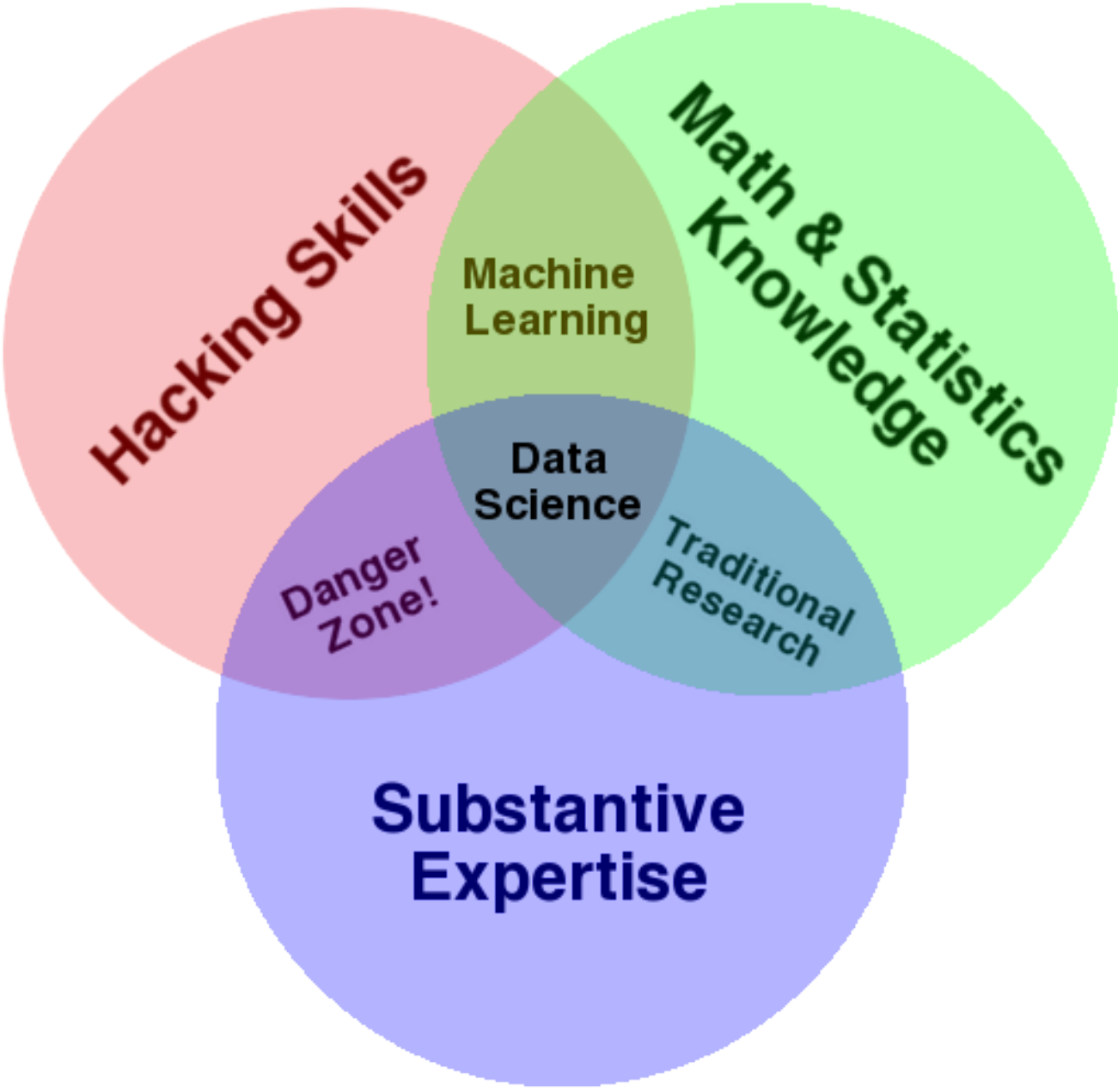
Dept for Applied Mathematics and Computer Science
Technical University of Denmark (DTU)

August 18, 2016

Ole Winther - a bit about myself



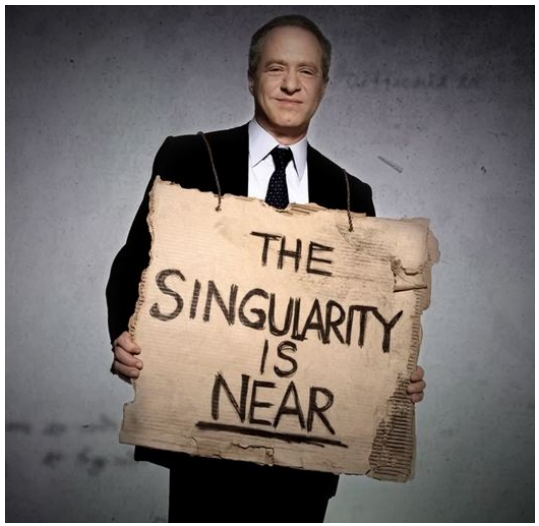
Data science - adding domain knowledge



Objectives of lectures week 1

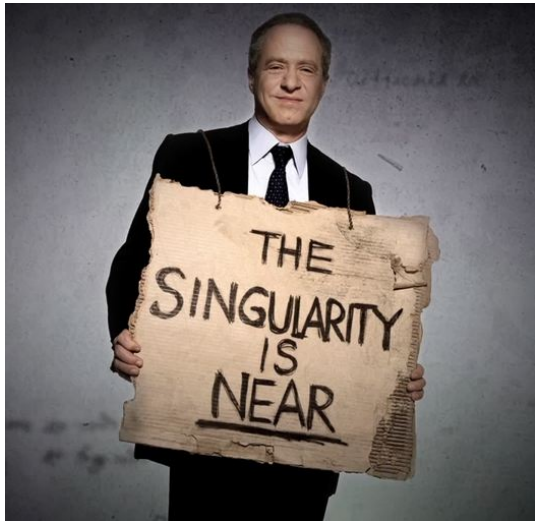
- P1: What is deep learning and statistical artificial intelligence?
- P2: The feed-forward neural network (FFNN)
- P3: Training with backpropagation
- P4: Optimisation.

Are we heading towards the singularity?



kurzweilai.net

Are we heading towards the singularity?

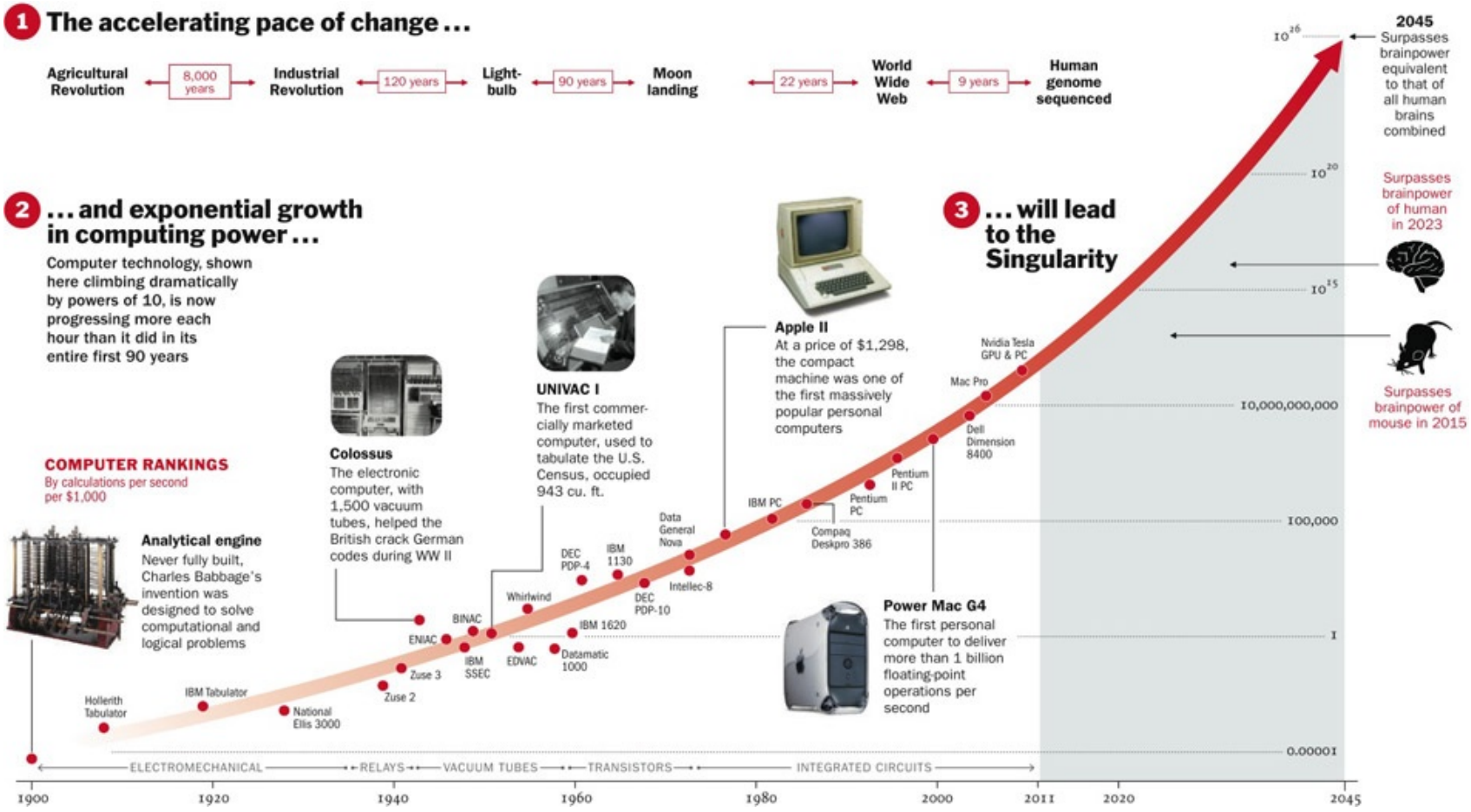


kurzweilai.net

- Elon Musk at MIT AeroAstro Symp:
- If I were to guess at what our biggest existential threat is, it's probably that...
- With artificial intelligence, we are summoning the demon..
- Inofficial quotes (email to friend):
- The risk of something seriously dangerous happening is in the five year timeframe. 10 years at most,
- Unless you have direct exposure to groups like Deepmind, you have no idea how fast — it is growing at a pace close to exponential.

Reinforcement learning

Growth in computer power



Major areas in AI

- Speech recognition
- Image classification
- Machine translation
- Question-answering
- Self-driving vehicles
- Dialogue systems
- General unsupervised learning

Major areas in AI

- Speech recognition
- Image classification
- Machine translation
- Question-answering
- Self-driving vehicles
- Dialogue systems
- General unsupervised learning

Part 1:

The deep learning revolution

Achilles' heel of traditional AI: Perception in natural environment

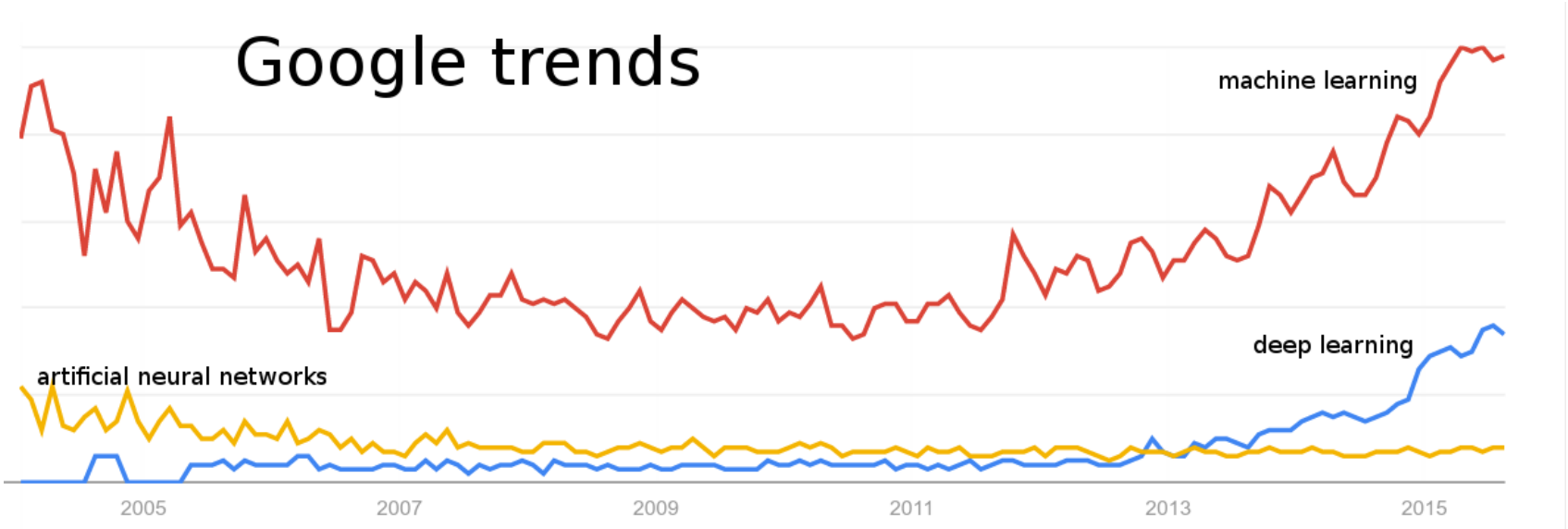
xkcd.com/1425

Many thanks to Tapani Raiko for sharing slides!

Deep learning is hot in academia

- ”Deep learning . . . dramatically improved the state-of-the-art in speech recognition, visual object recognition. . . ” (LeCun et al., Nature, 2015)
- ”. . . bridges the divide between high-dimensional sensory inputs and actions, resulting in the first artificial agent. . . ” (Mnih et al., Nature, 2015)
- ”Knowing the sequence specificities of DNA- and RNA-binding proteins is essential . . . deep learning outperforms other state-of-the-art methods” (Alipanahi et al., Nature Biotechnology, 2015)
- ”This is the first time that a computer program has defeated a human professional payer in the full-sized game of Go” (Silver et al, Nature 2016)

Deep learning is hot in industry

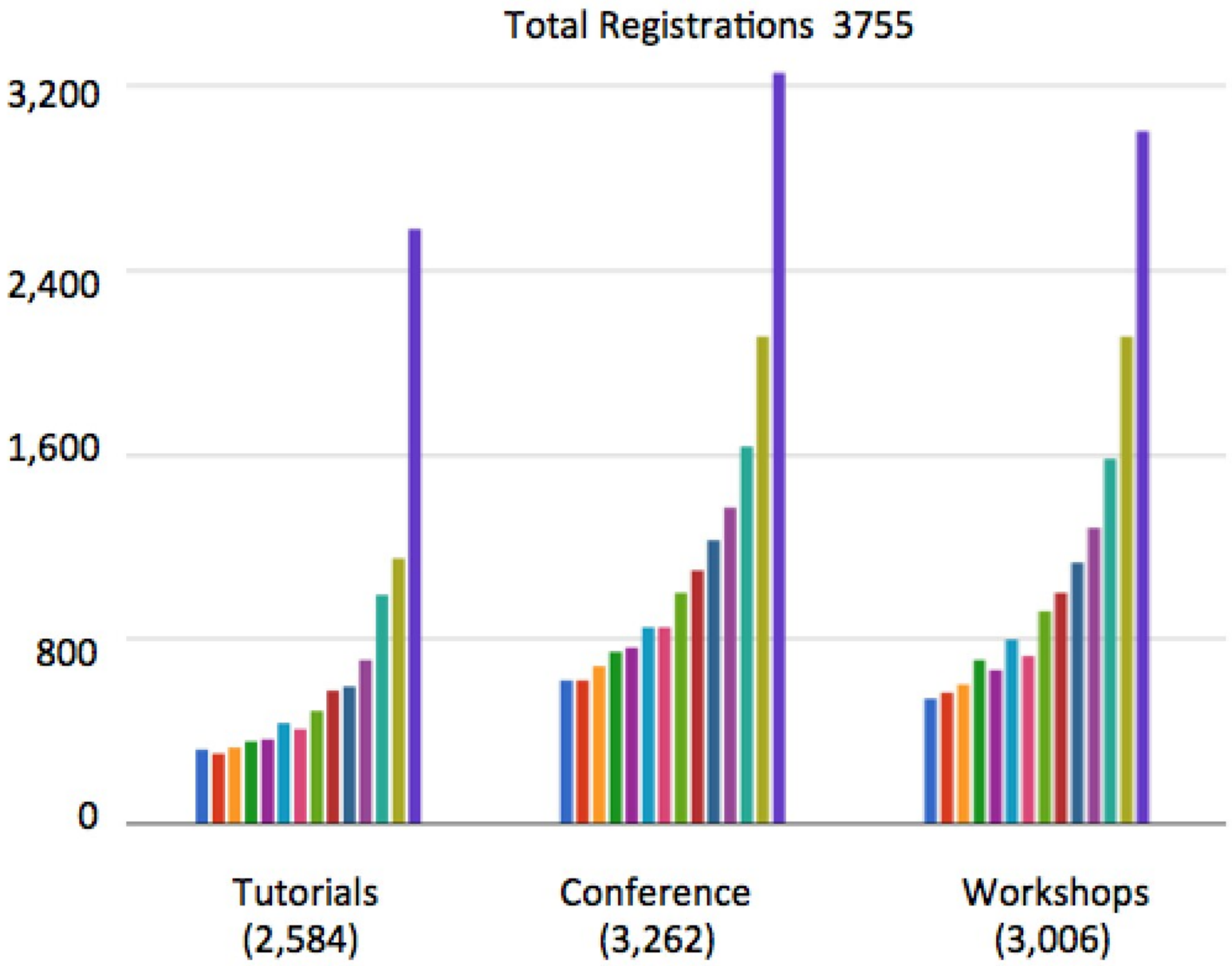


Google acquired startup DeepMind for $500M in 2014.
Also racing: Facebook, Baidu, IBM, Amazon, Samsung, Nvidia, Apple, Nokia, . . .

On the cover of The Economist

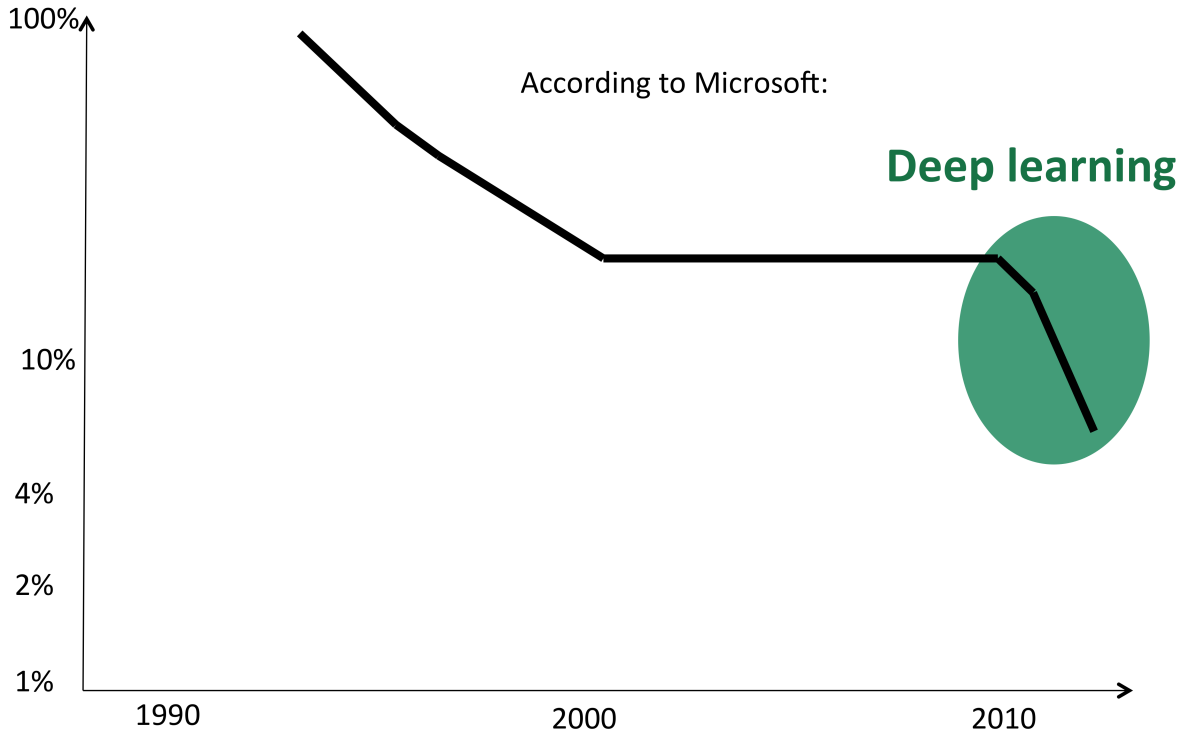
Economist 25th June 2016

NIPS Growth


Total Registrations 3755
3,200
2,400
1,600
800
0
Tutorials
(2,584)
Conference
(3,262)
Workshops
(3,006)

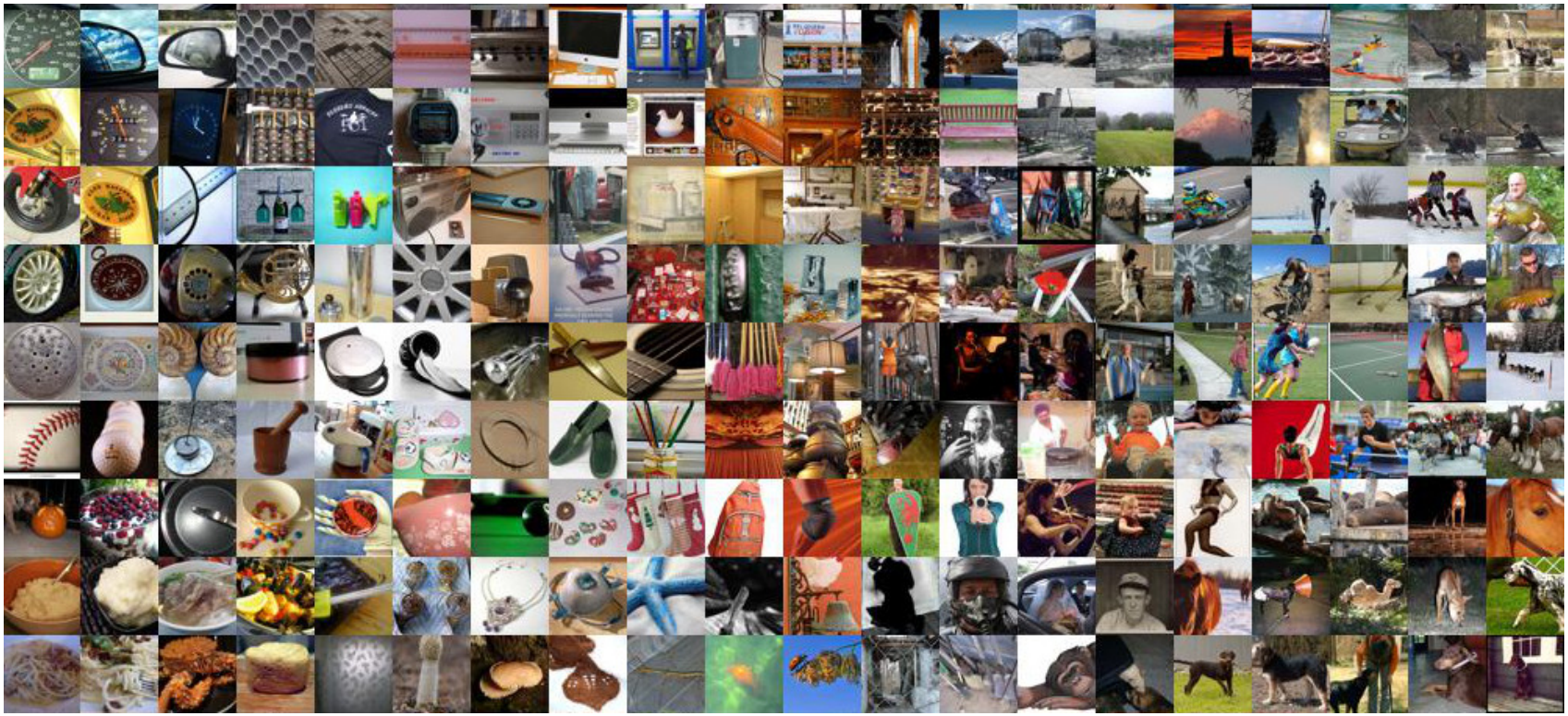
Deep learning is changing the world

Speech recognition breakthrough



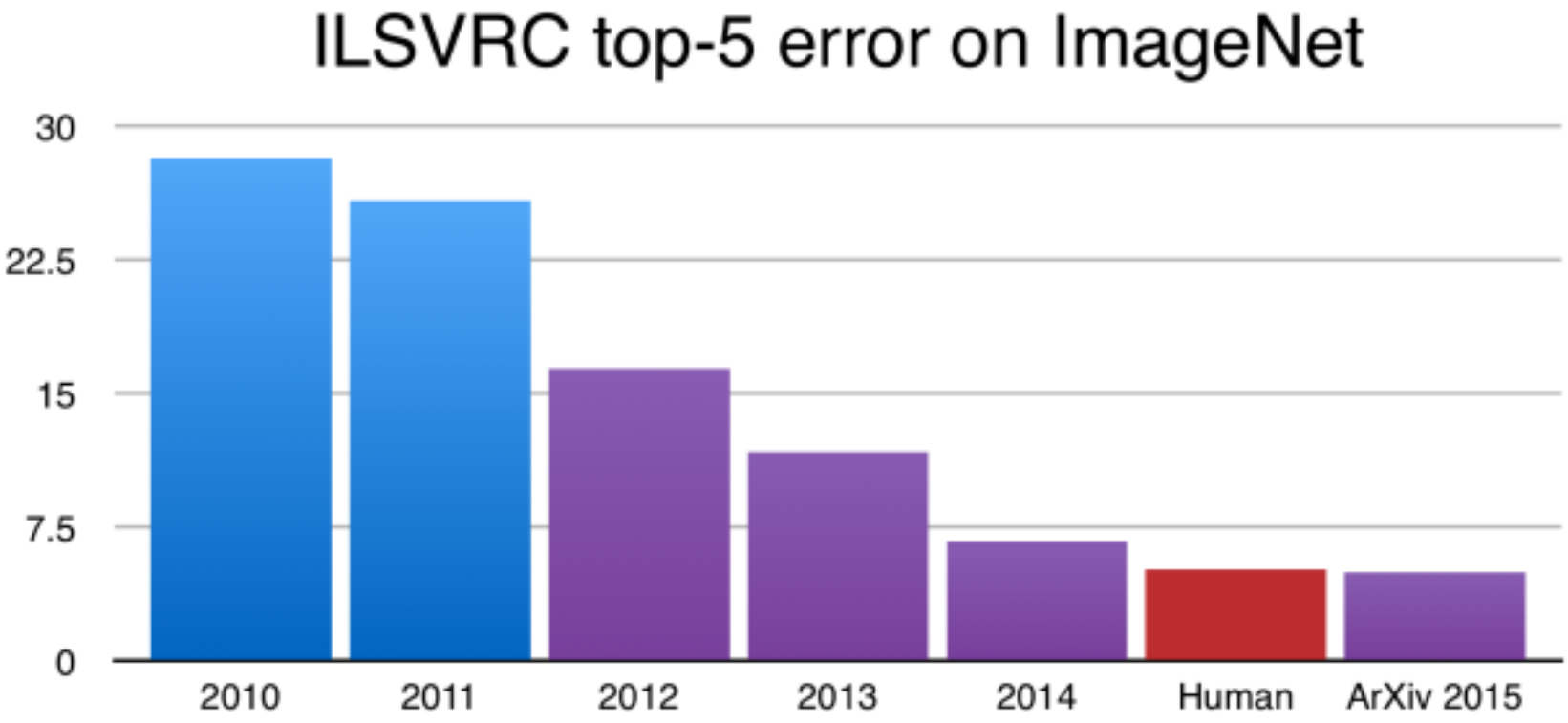
Plot from Yoshua Bengio

Imagenet classification challenge



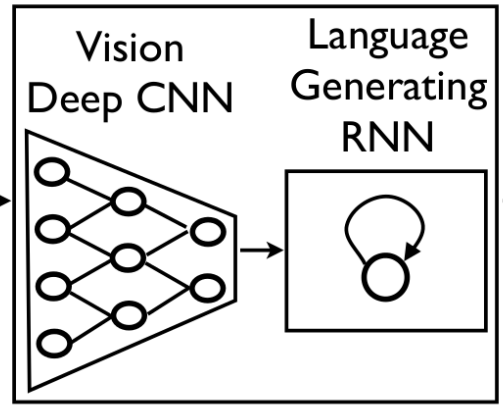
Annual competition in computer vision.

Imagenet classification challenge



Krizhevsky et al. (2012) won with huge margin
(16.4% error compared to 26.2%) by deep learning.
Soon everyone started using deep learning and GPUs.

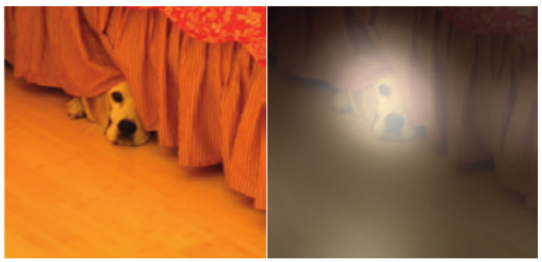
Caption generation (Xu et al., 2015)



A group of people shopping at an outdoor market.

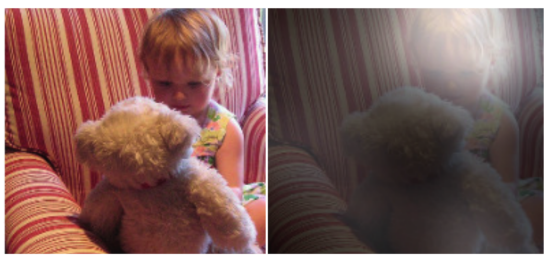
There are many vegetables at the fruit stand.

A woman is throwing a **frisbee** in a park.



A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

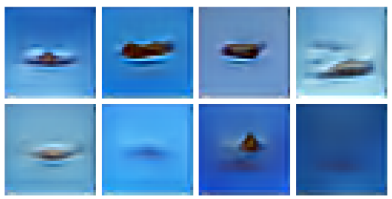


A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

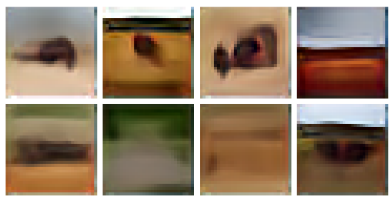
A giraffe standing in a forest with **trees** in the background.

Text to images (Mansimov et al., 2015)

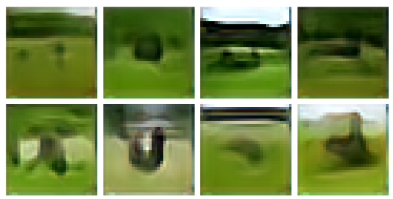


A very large commercial plane flying in blue skies.

A very large commercial plane flying in rainy skies.



A herd of elephants walking across a dry grass field.



A herd of elephants walking across a green grass field.

Modifying visual features (Larsen et al., 2015)

Representation learning

Traditional way:
Data $\rightarrow$ Feature engineering $\rightarrow$ Machine learning

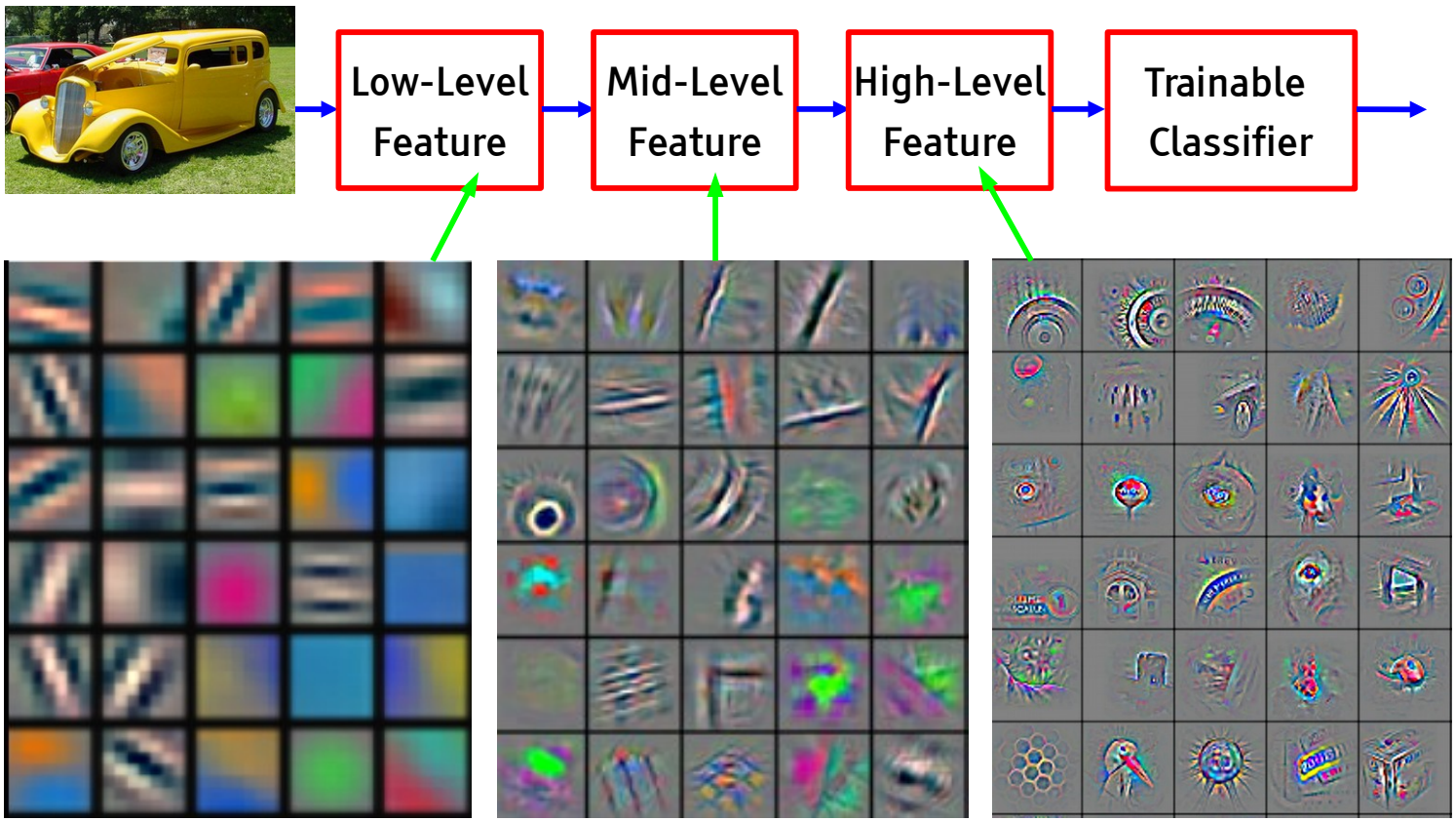
- Feature selection
- Feature extraction (e.g. PCA)
- Feature construction (e.g. SIFT)

Deep learning way:
Data $\rightarrow$ End-to-end learning

Deep Learning = Learning Hierarchical Representations

Y LeCun

- It's deep if it has more than one stage of non-linear feature transformation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

- Review article, May 2015:

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

nature

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

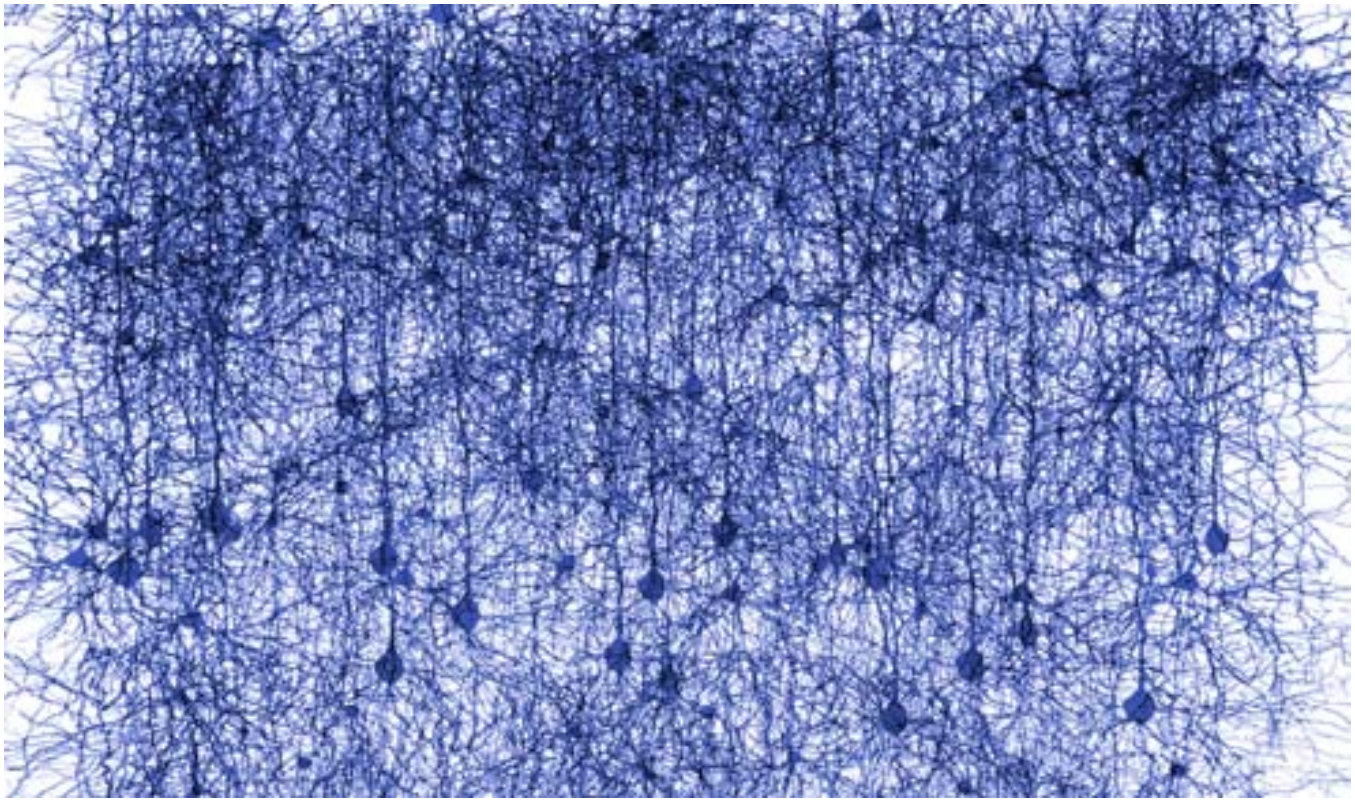
- Book: M.Nielsen, Neural networks and deep learning
- Book, draft available online:

- Portal: `deeplearning.net`

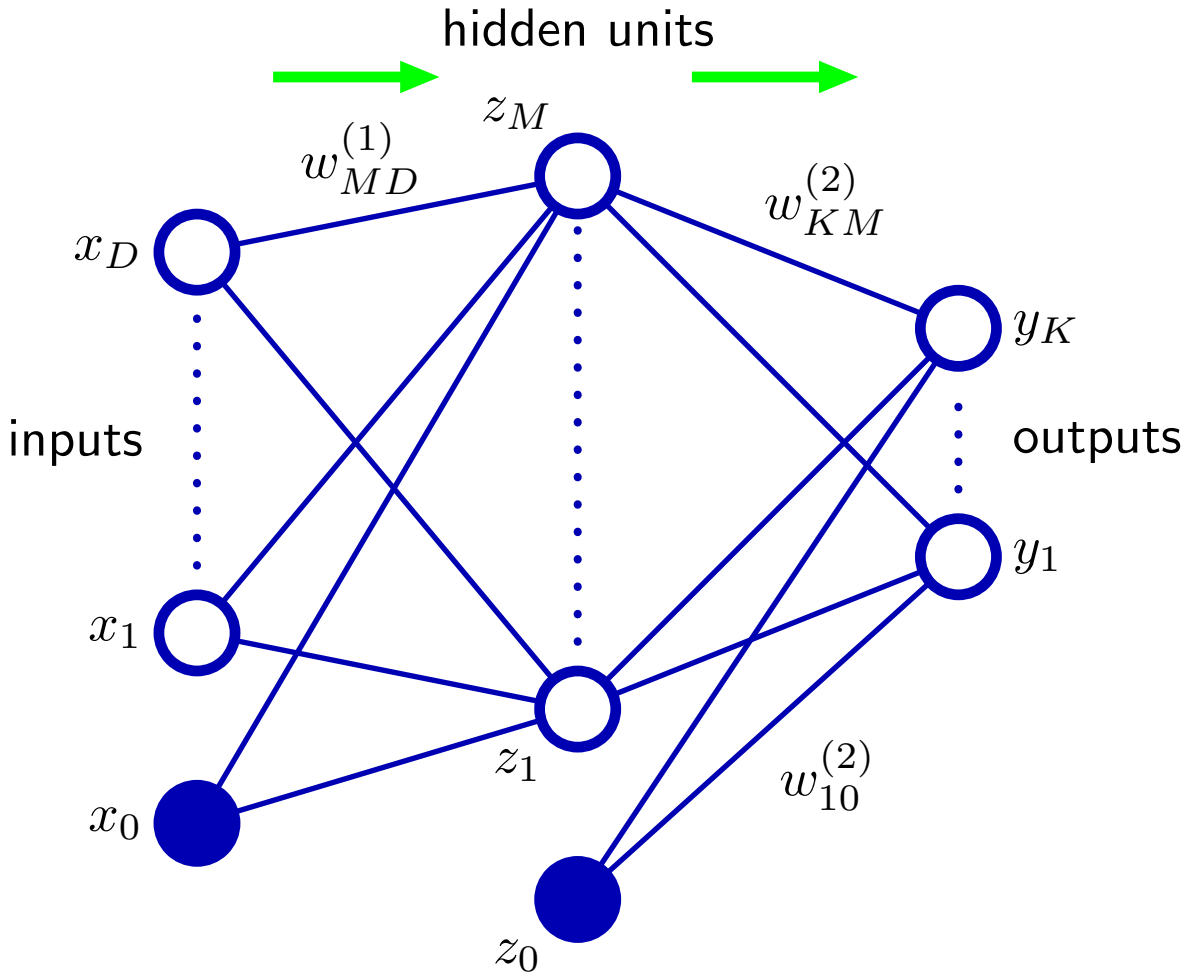
Part 2:
Neural networks

Neural networks (NNs)

- Feedforward neural networks (FFNNs)
- Convolutional neural networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Auto-encoders (AE)



Feed forward neural networks



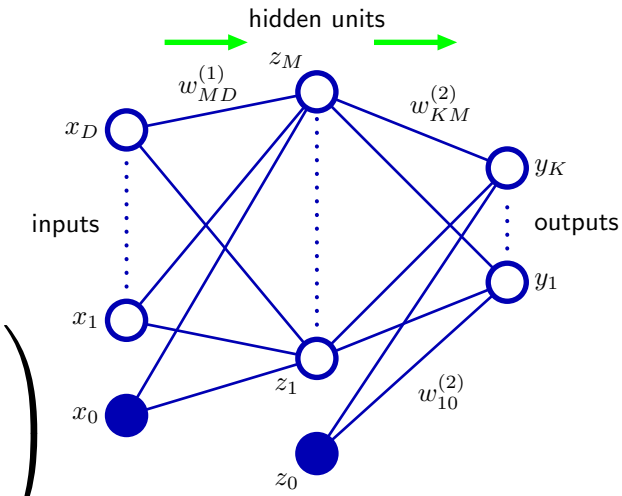
Neural network mapping

- Compute weighted sum of inputs:

$$\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

- Output k two-layer network:

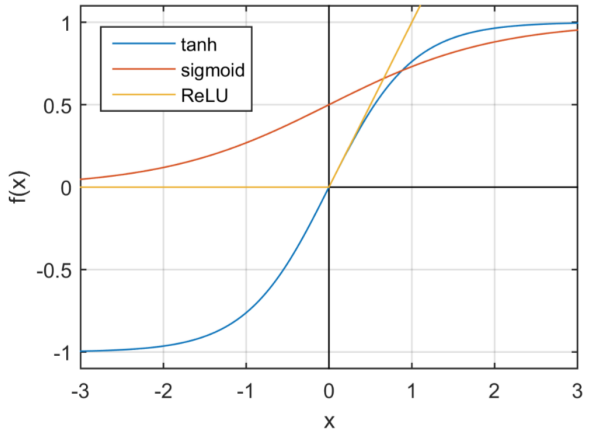
$$h_k^{(2)}(\mathbf{x}, \mathbf{w}) = f_2 \left(\sum_{j=0}^{M} w_{kj}^{(2)} f_1 \left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right)$$



- f_1 and f_2 hidden unit activation functions

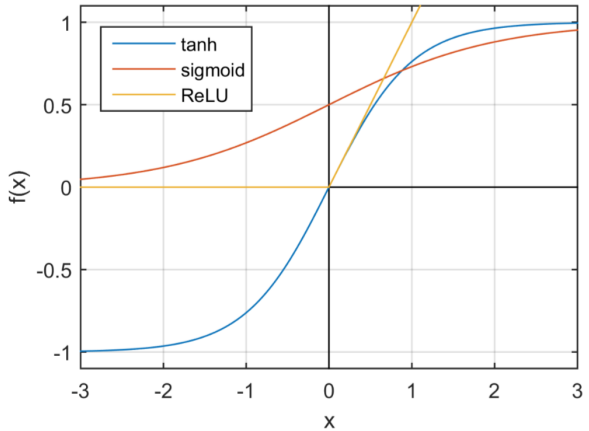
Non-linearity and training

- Linear activation functions will give a linear network.
- Logistic function $\sigma(a) = \frac{1}{1+e^{-a}}$
- Hyperbolic tangent $\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- Rectified linear $\mathrm{relu}(a) = \max(0, a)$



Non-linearity and training

- Linear activation functions will give a linear network.
- Logistic function $\sigma(a) = \frac{1}{1+e^{-a}}$
- Hyperbolic tangent $\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- Rectified linear $\mathrm{relu}(a) = \max(0, a)$



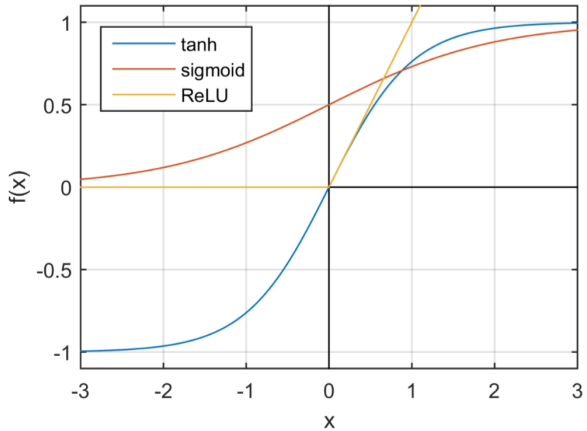
- Supervised learning
- Labeled training set

$$\mathcal{D} = \{(x_i, y_i) | i = 1, \ldots, n\} \ .$$

- Input x_i and output y_i.

Non-linearity and training

- Linear activation functions will give a linear network.
- Logistic function $\sigma(a) = \frac{1}{1+e^{-a}}$
- Hyperbolic tangent $\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- Rectified linear $\mathrm{relu}(a) = \max(0, a)$



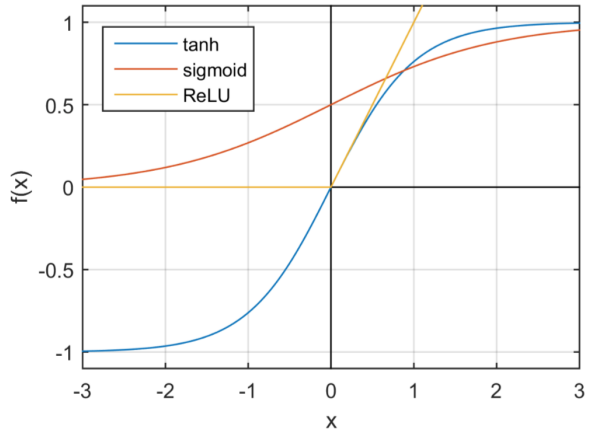
- Supervised learning
- Labeled training set

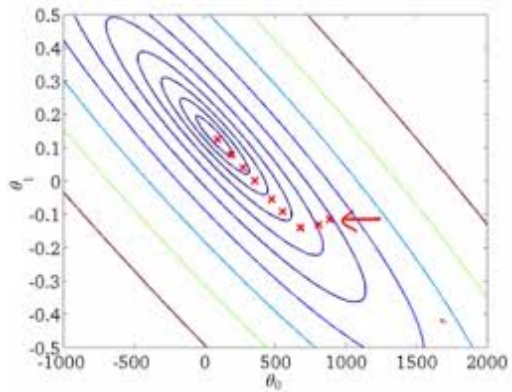
$$\mathcal{D} = \{(x_i, y_i) | i = 1, \ldots, n\} .$$

- Input x_i and output y_i.
- Minimize training error by (stochastic) gradient descent

Non-linearity and training

- Linear activation functions will give a linear network.
- Logistic function $\sigma(a) = \frac{1}{1+e^{-a}}$
- Hyperbolic tangent $\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- Rectified linear $\mathrm{relu}(a) = \max(0, a)$





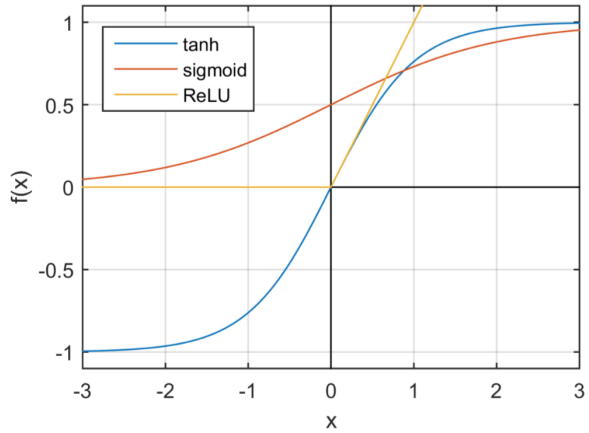
- Supervised learning
- Labeled training set

$$\mathcal{D} = \{(x_i, y_i) | i = 1, \ldots, n\} \ .$$

- Input x_i and output y_i.
- Minimize training error by (stochastic) gradient descent

Non-linearity and training

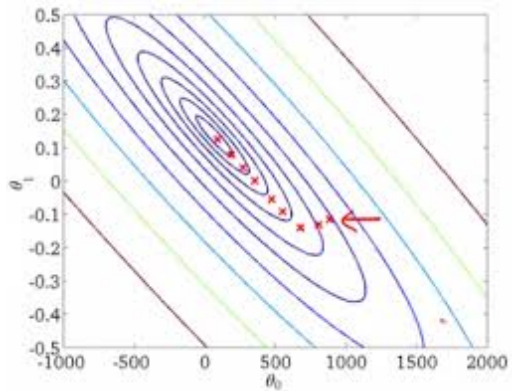
- Linear activation functions will give a linear network.
- Logistic function $\sigma(a) = \frac{1}{1+e^{-a}}$
- Hyperbolic tangent $\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- Rectified linear $\mathrm{relu}(a) = \max(0, a)$

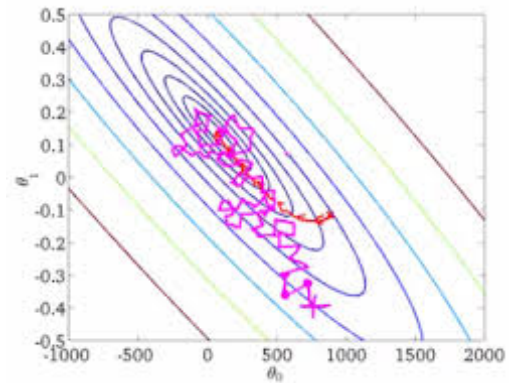


- Supervised learning
- Labeled training set

$$\mathcal{D} = \{(x_i, y_i) | i = 1, \ldots, n\} .$$

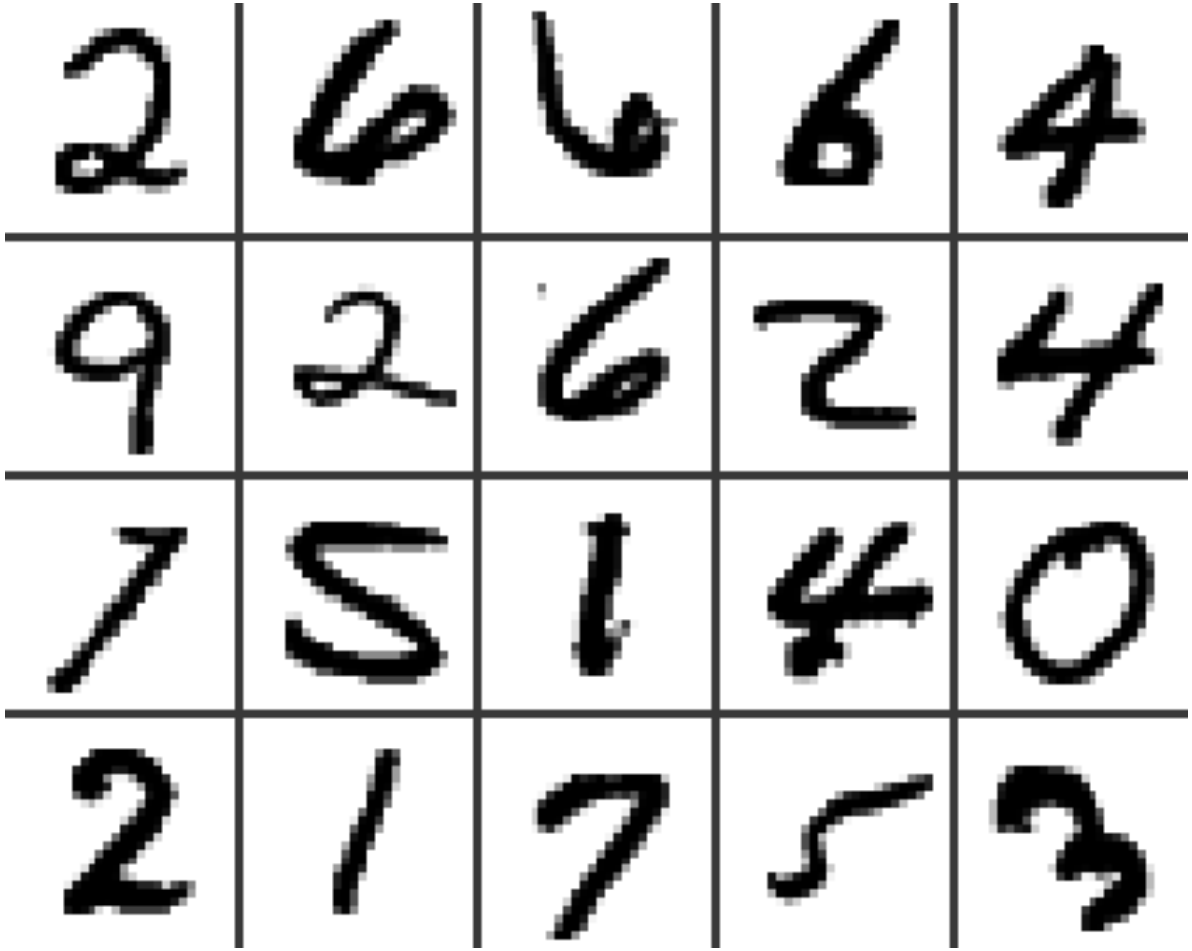
- Input x_i and output y_i.
- Minimize training error by (stochastic) gradient descent





Overfitting!

Example: MNIST handwritten digits

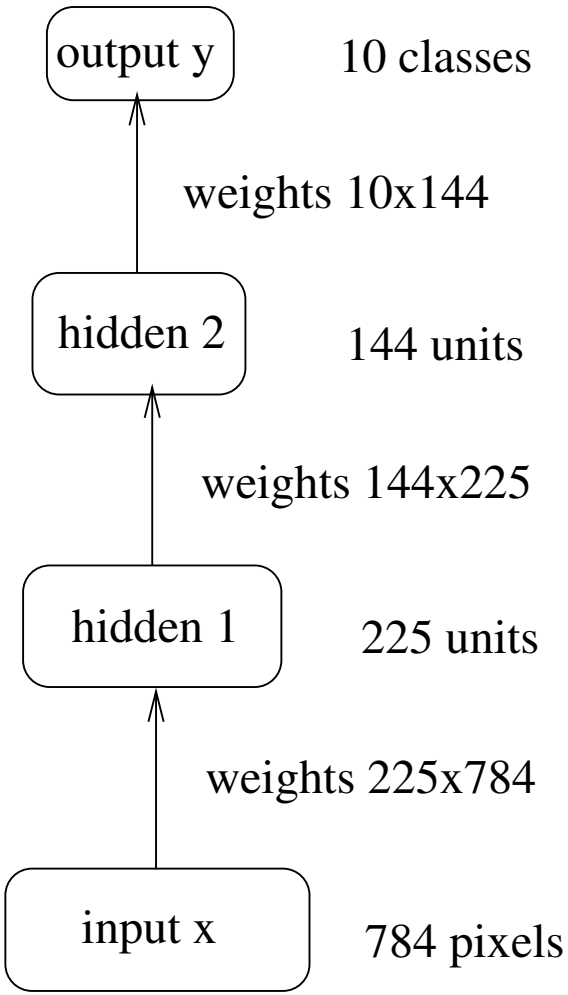


Train a network to classify 28×28 images.

Data: 60000 input images $\mathbf{x}(n)$ and labels $y(n)$.

Example model gives around 1.2% test error.

Example Network



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

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

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

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

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

Softmax

- Softmax function

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

has two nice properties:

- $\text{softmax}(\mathbf{z})_i \geq 0$
- $\sum_i \text{softmax}(\mathbf{z})_i = 1$

Softmax

- Softmax function

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

  has two nice properties:

- $\text{softmax}(\mathbf{z})_i \geq 0$
- $\sum_i \text{softmax}(\mathbf{z})_i = 1$
- MNIST, output labels: $0, 1, \ldots, 9$.
- Output of network

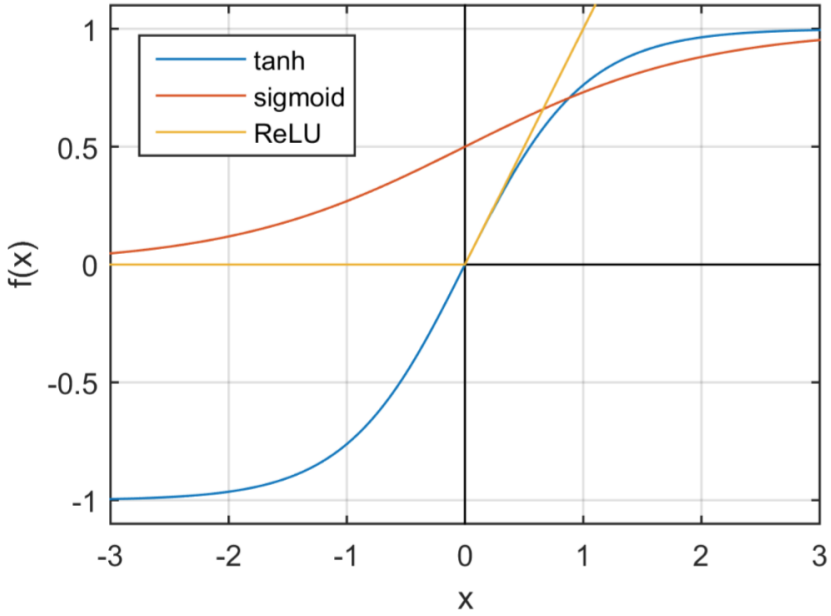
$$\mathbf{h}^{(3)} = \text{softmax}(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)})$$

  interpreted as class(-conditional) probabilities:

- For example:

$$p(5|\mathbf{x}) = \mathbf{h}_5^{(3)}$$

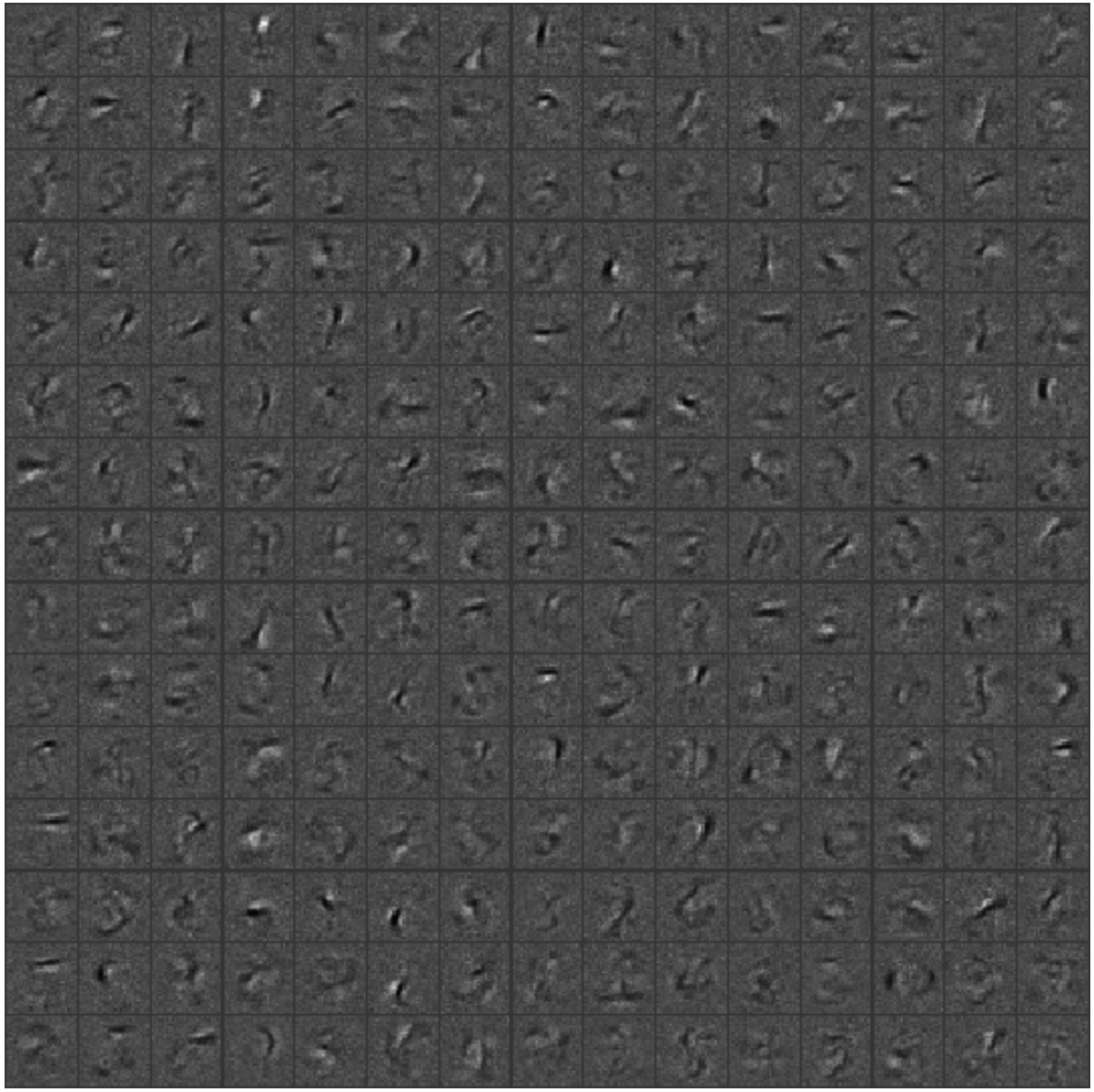
On activation functions



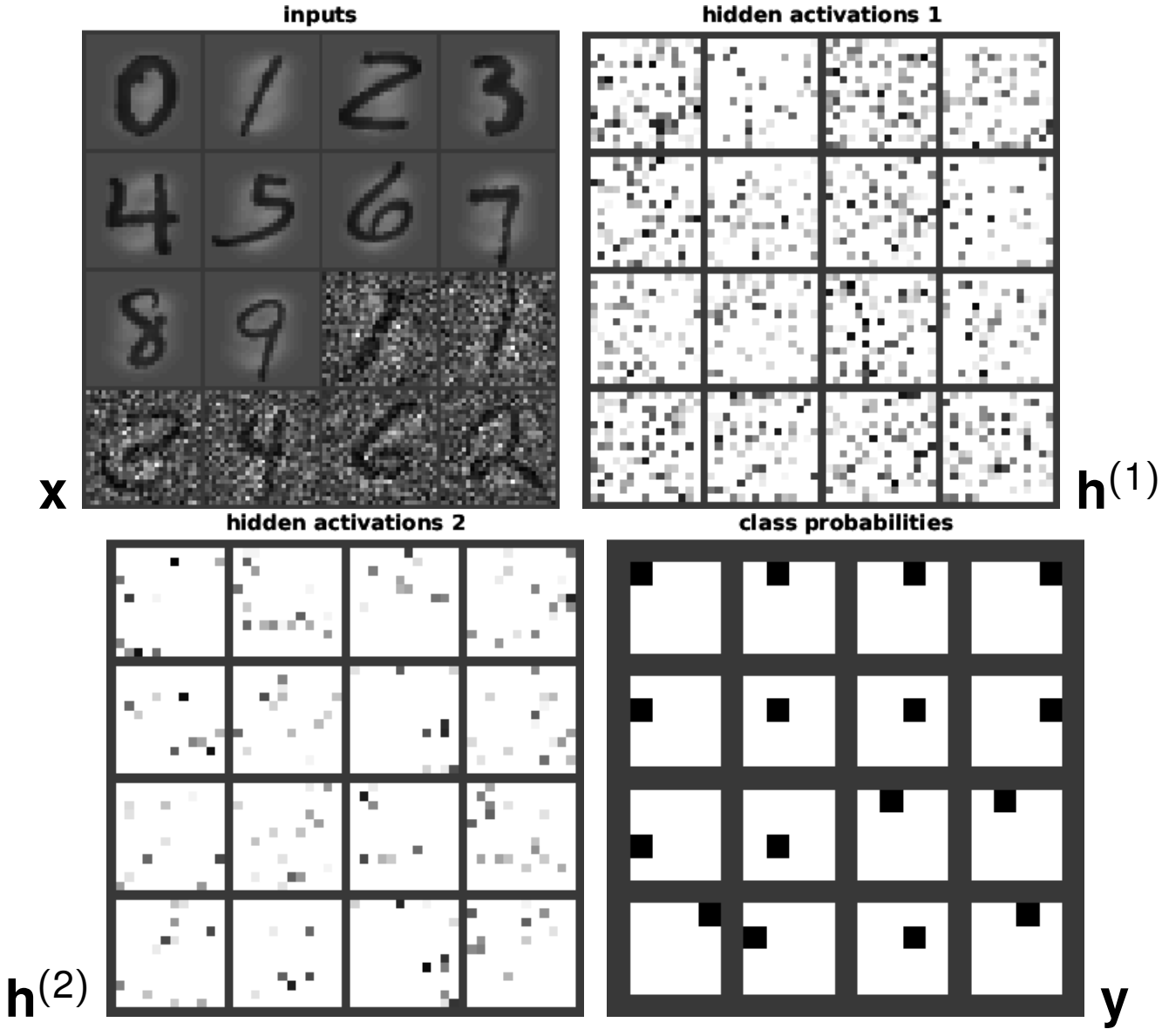
- $\text{relu}(z) = \max(0, z)$ is replacing old sigmoid and tanh.
- Note that identity function would lead into:

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

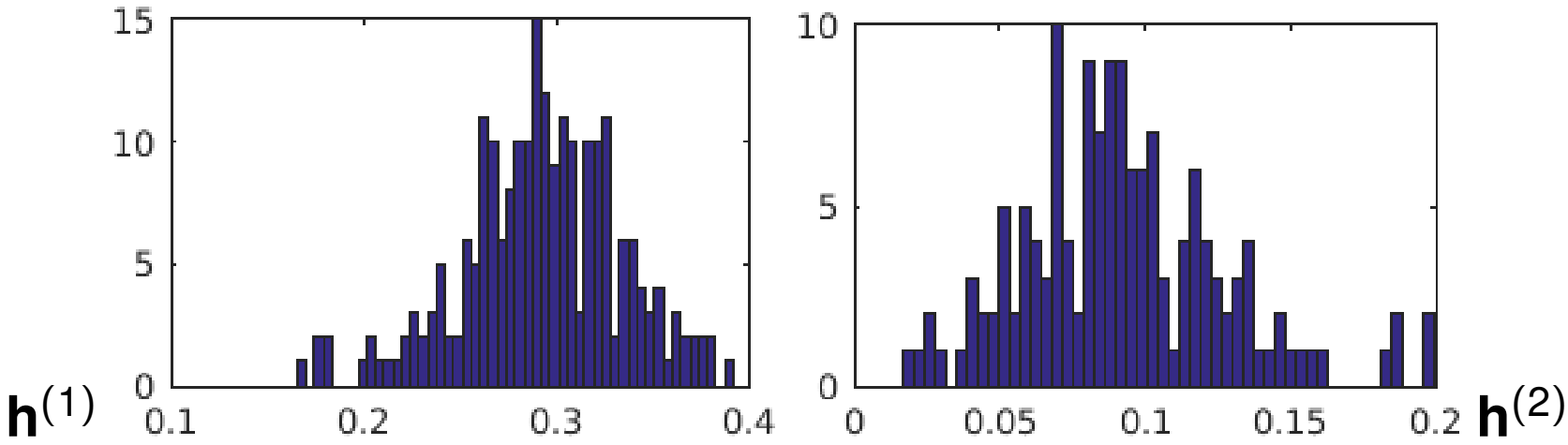
Weight matrix $\mathbf{W}^{(1)}$ size 225×784



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 3:

Neural network training

Backpropagation

Training criterion

Find parameters

$$\theta = \{\mathbf{W}^{(L)}, \mathbf{b}^{(L)}\}$$

that minimize expected negative log-likelihood:

$$C = \mathbb{E}_{\text{data}} \left[-\log P(\mathbf{y}|\mathbf{x}, \theta) \right].$$

Learning becomes optimization.

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

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_\theta C(\theta) = \operatorname*{argmin}_\theta \mathbb{E}_{P(\mathbf{y}|\mathbf{x})} \left[-\log Q(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}) \right].$$

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

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

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: $0, 1, \ldots, 9$.
- Convenient to use a sparse one hot encoding:

$$0 \to \mathbf{y} = (1,0,0,0,0,0,0,0,0,0)^T$$
$$1 \to \mathbf{y} = (0,1,0,0,0,0,0,0,0,0)^T$$
$$2 \to \mathbf{y} = (0,0,1,0,0,0,0,0,0,0)^T$$
$$\ldots.$$
$$9 \to \mathbf{y} = (0,0,0,0,0,0,0,0,0,1)^T$$

- Output

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

  interpreted as class(-conditional) probability.
- Cross-entropy cost - sum over data and label

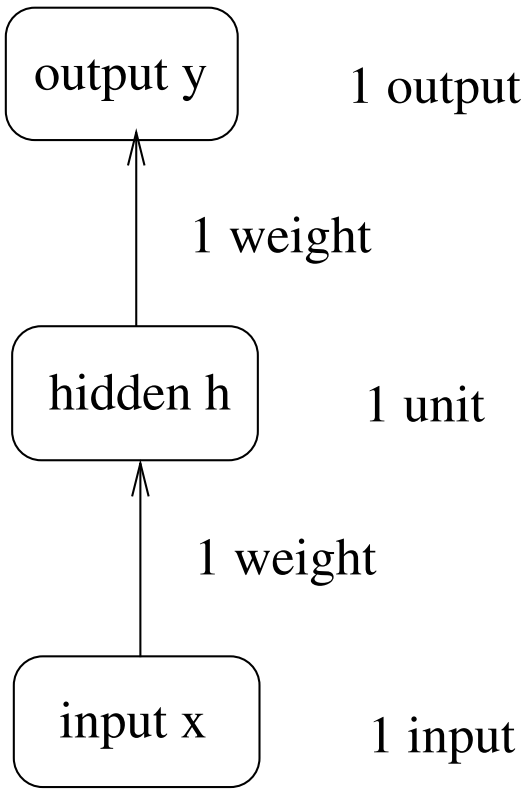
$$C = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \log h_{nk}^{(3)}$$

Gradient descent

- Simple algorithm for minimizing the training criterion C.
- Gradient $\mathbf{g} = \nabla_{\boldsymbol{\theta}} C(\theta) = \begin{pmatrix} \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_n} \end{pmatrix}$
- Iterate $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k$
- Notation: iteration k, stepsize (or learning rate) η_k

Backpropagation (Linnainmaa, 1970)

Computing gradients in a network.



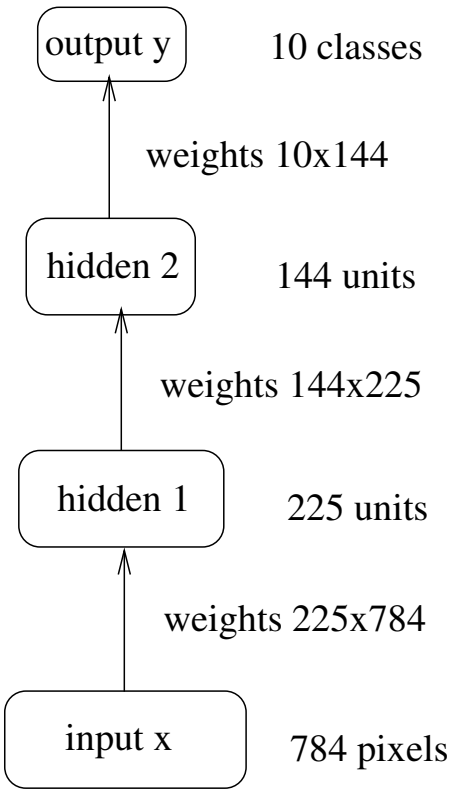
- First with scalars. Use chain rule:

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

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

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

Backpropagation



- Multi-dimensional:

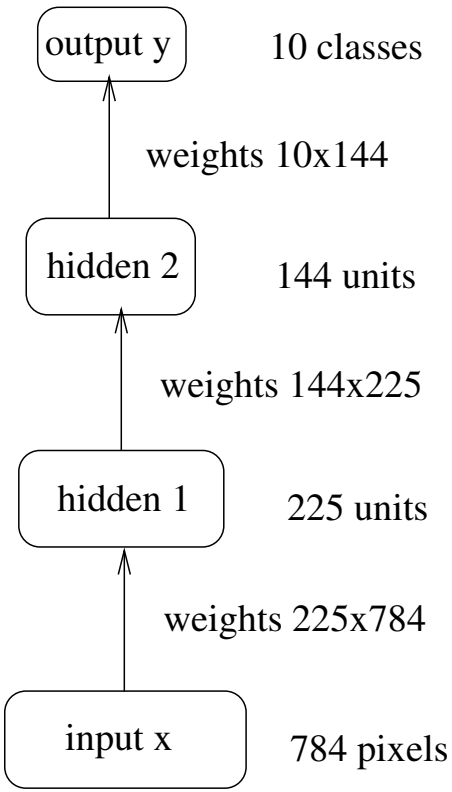
$$\frac{\partial C}{\partial W_{ij}^{(3)}} = \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial W_{ij}^{(3)}}$$

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

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

- *How many paths - for two hidden layers*
- *as a function of depth?*

Backpropagation



- Multi-dimensional:

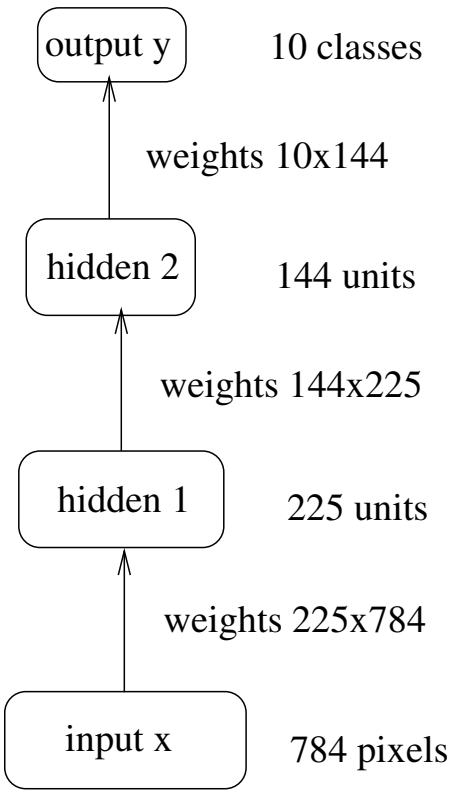
$$\frac{\partial C}{\partial W_{ij}^{(3)}} = \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial W_{ij}^{(3)}}$$

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

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

- *How many paths - for two hidden layers*
- *as a function of depth?*

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:

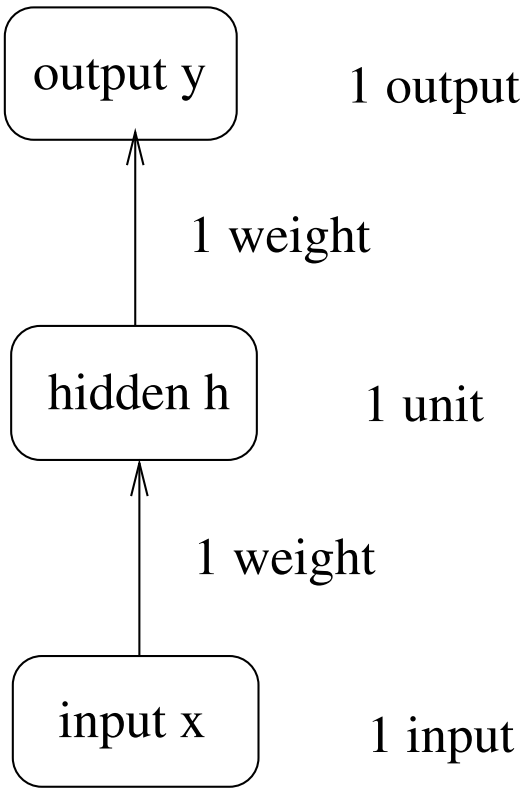
$$\frac{\partial C}{\partial W_{ij}^{(l)}} = \frac{\partial C}{\partial h_i^{(l)}} \frac{\partial h_i^{(l)}}{\partial W_{ij}^{(l)}}$$

Part 4:

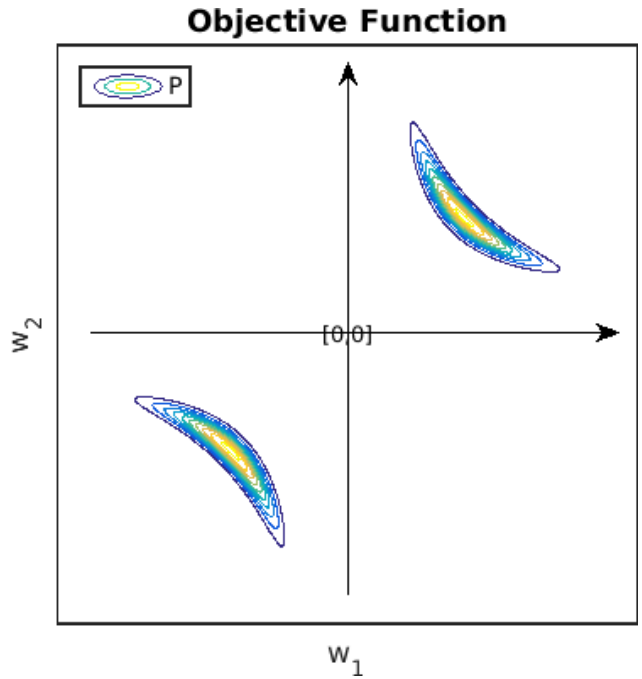
Neural network training

Optimisation

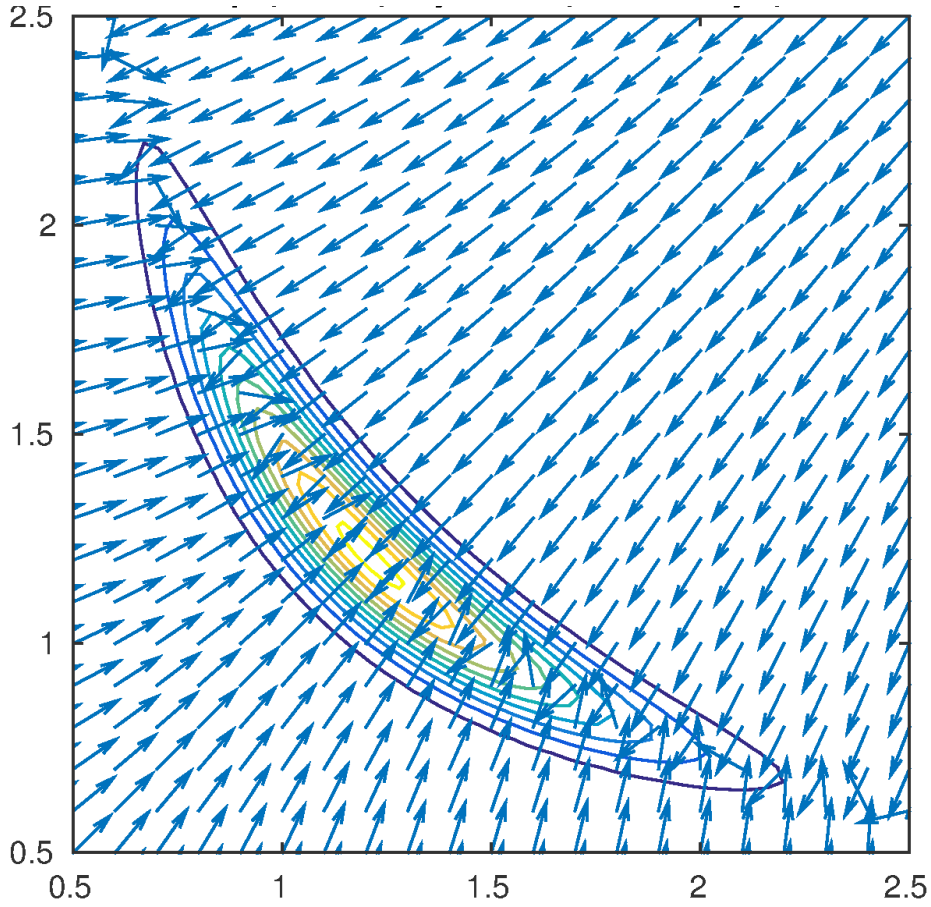
Tiny Example



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

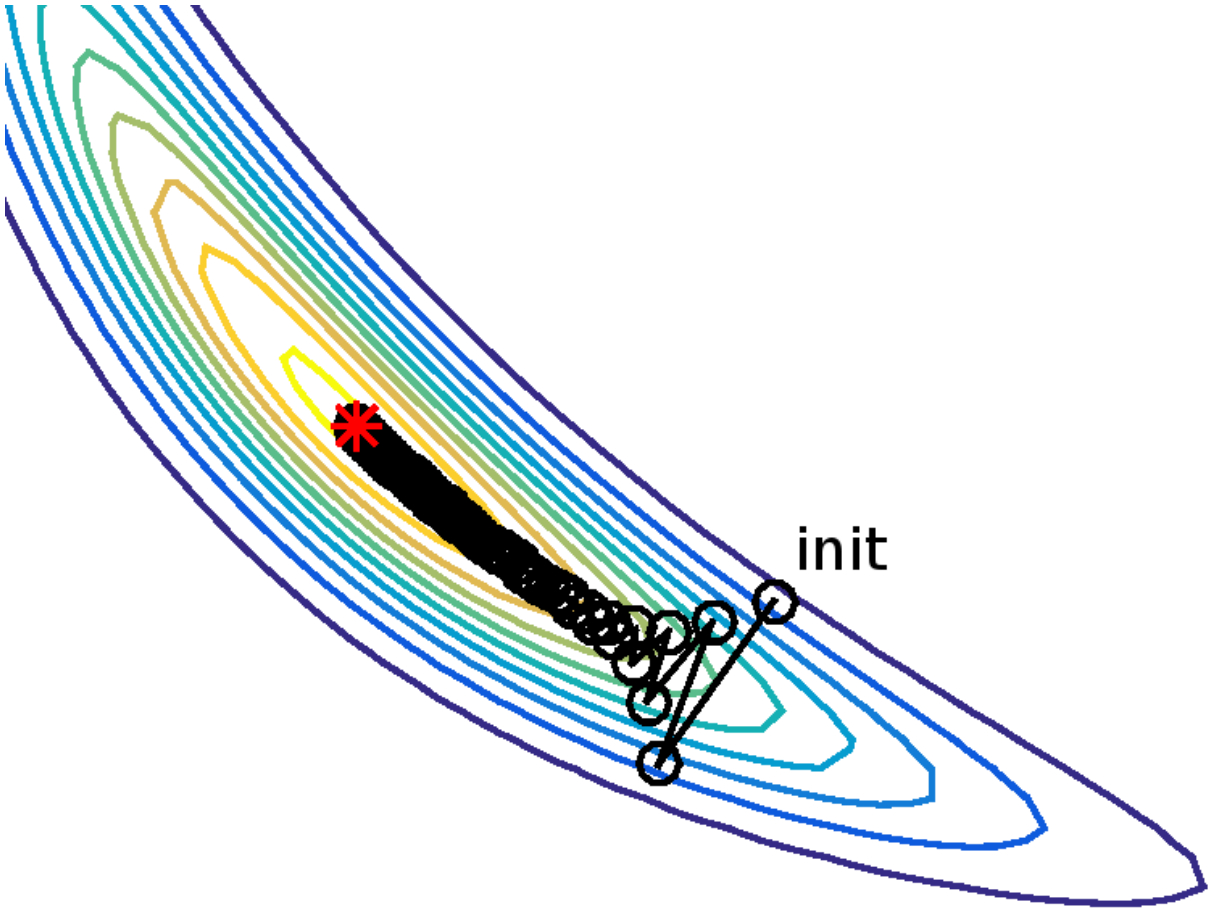


Gradient $\mathbf{g} = \nabla_\theta C(\theta) = \begin{pmatrix} \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_n} \end{pmatrix}$



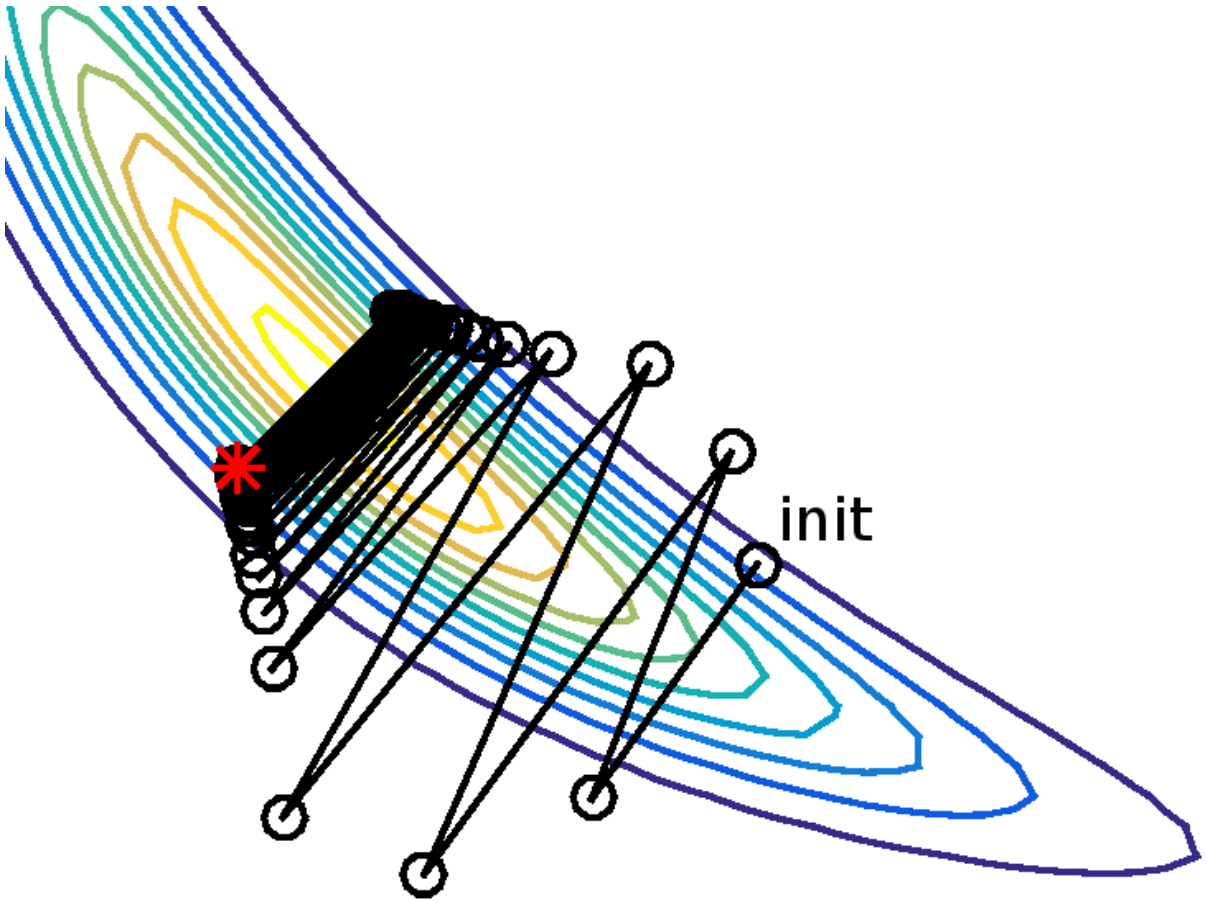
Gradient descent, $\eta_k = 0.25$ ($\rightarrow$ too slow)

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



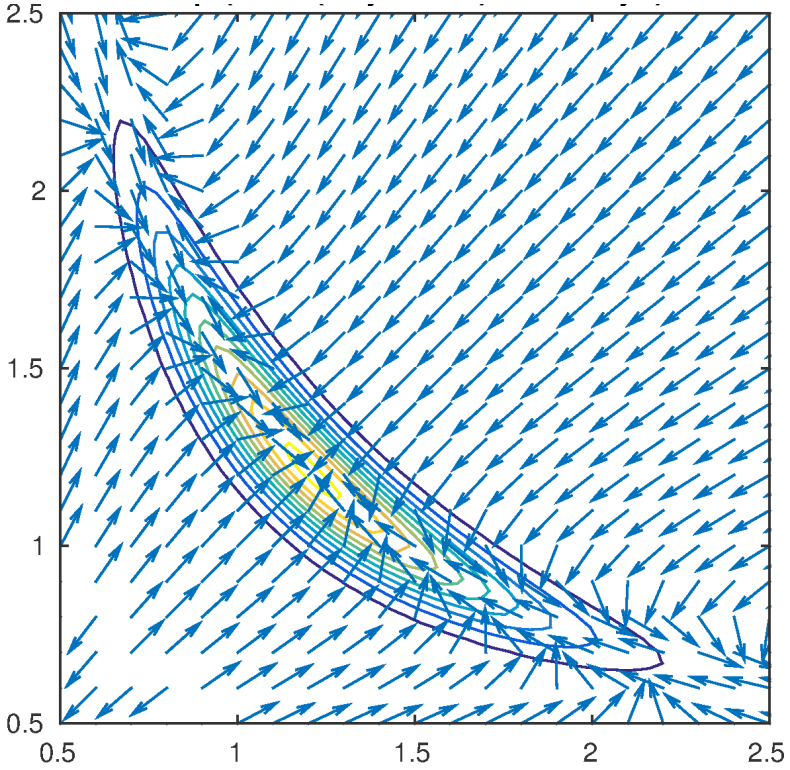
Gradient descent, $\eta_k = 0.35$ ($\rightarrow$ oscillates)

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



Newton's method, too complex

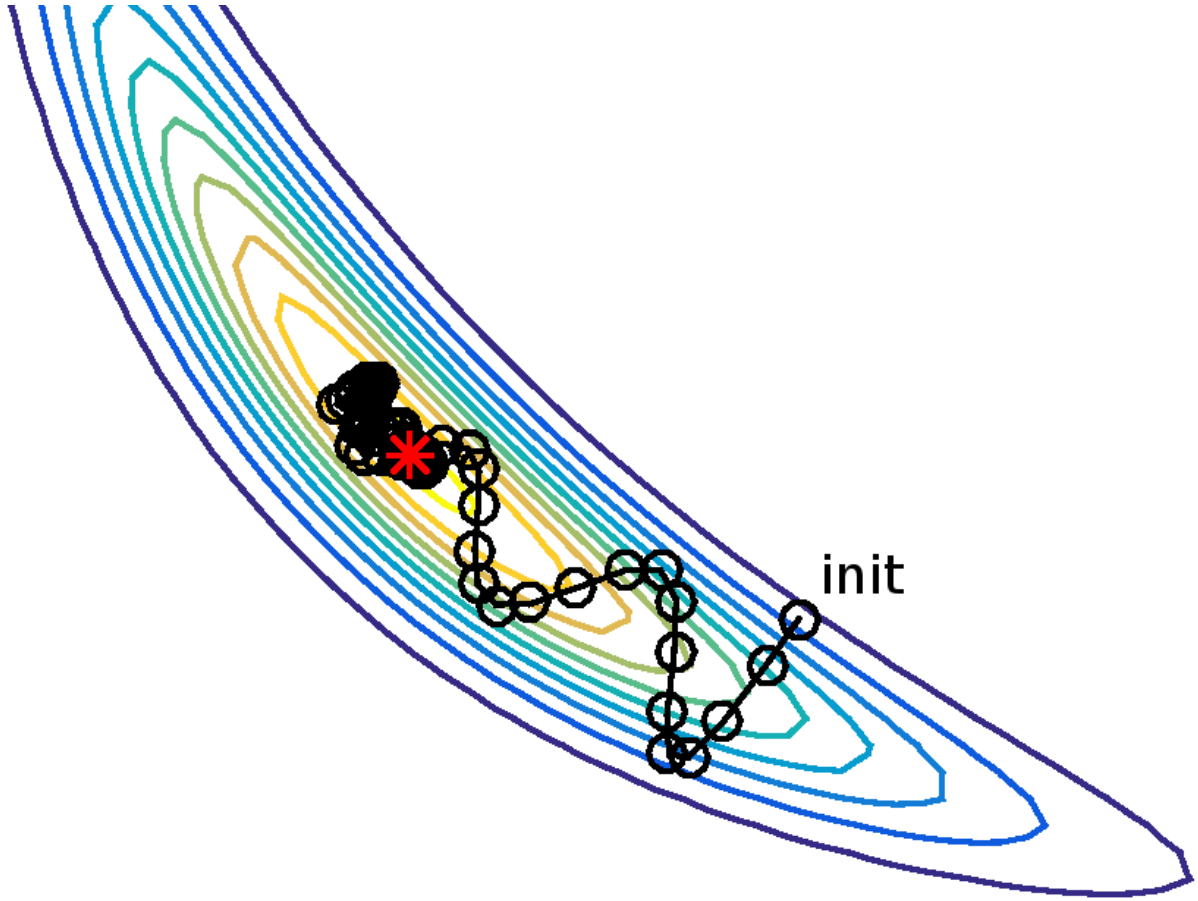
$$\theta_{k+1} = \theta_k - \mathbf{H}_k^{-1}\mathbf{g}_k, \quad \mathbf{H} = \begin{pmatrix} \frac{\partial^2 C}{\partial\theta_1\partial\theta_1} & \cdots & \frac{\partial^2 C}{\partial\theta_1\partial\theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 C}{\partial\theta_n\partial\theta_1} & \cdots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



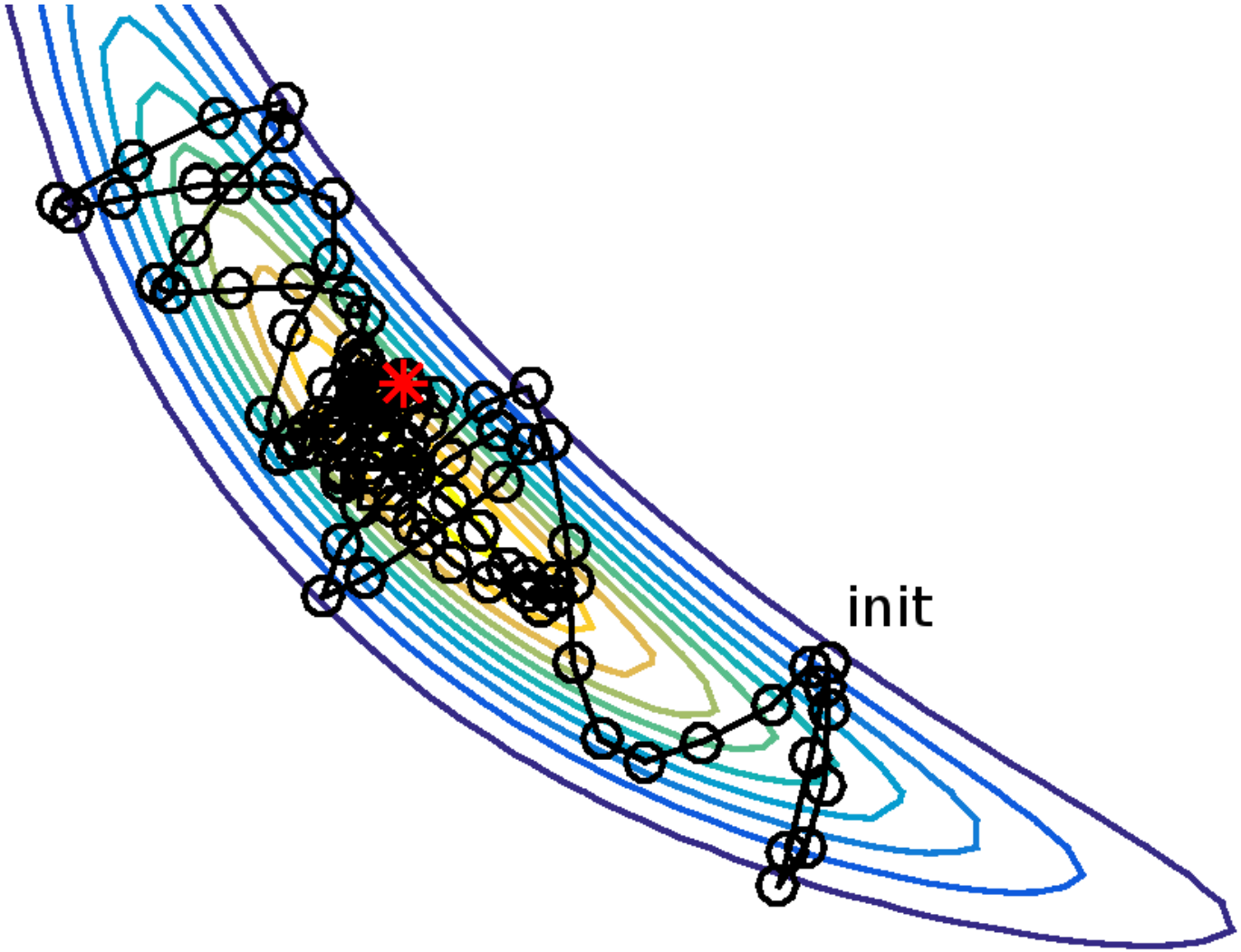
- Less oscillations.
- Points to the wrong direction in places (solvable).
- Computational complexity: #params^3 (prohibitive).
- There are approximations, but not very popular.

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

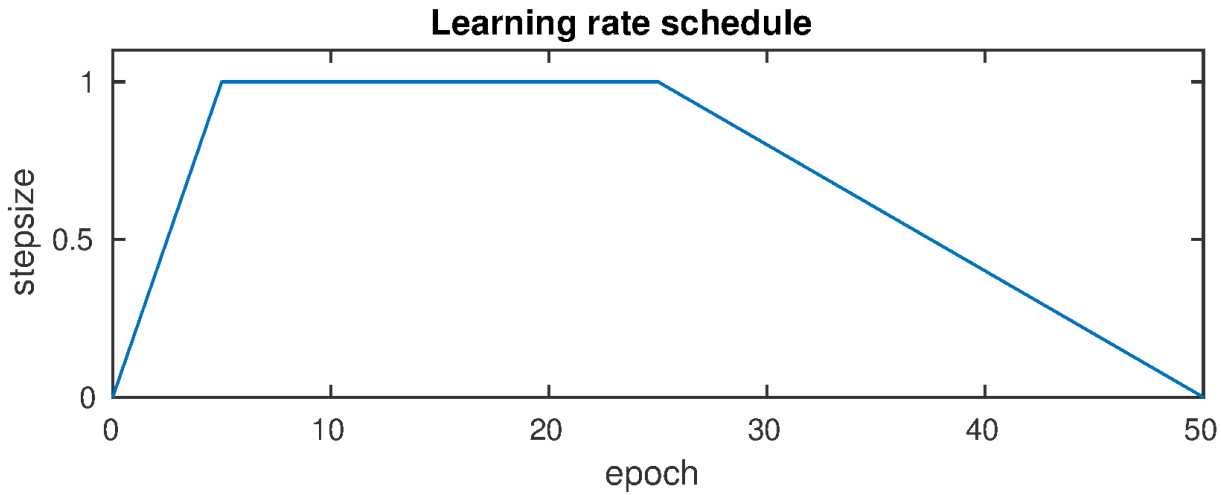


Mini-batch training

- 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.

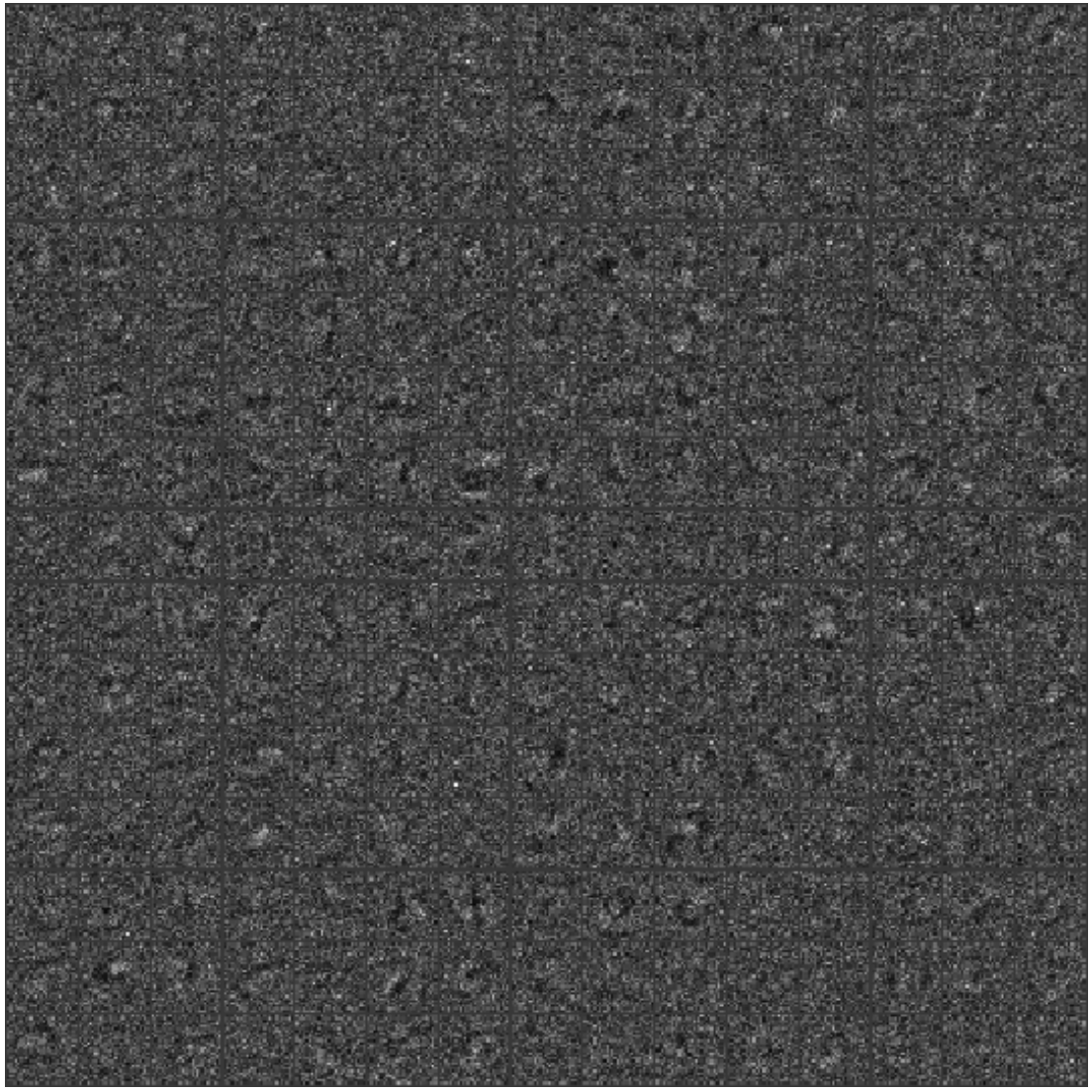
Mini-batch training

- No need to have an accurate estimate of **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 η_k towards the end, e.g.

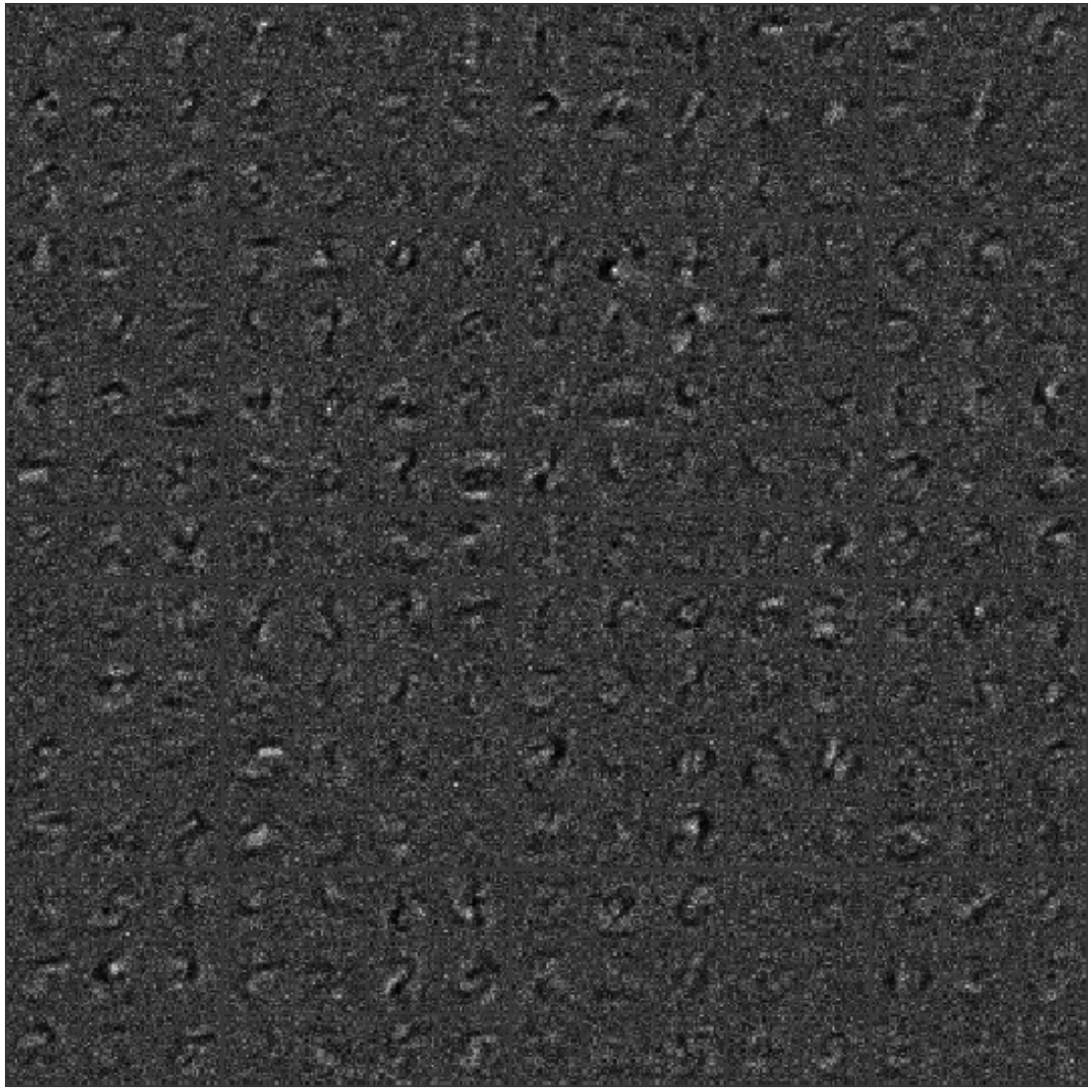


- Adaptation of η_k possible (Adam, Adagrad, Adadelta).

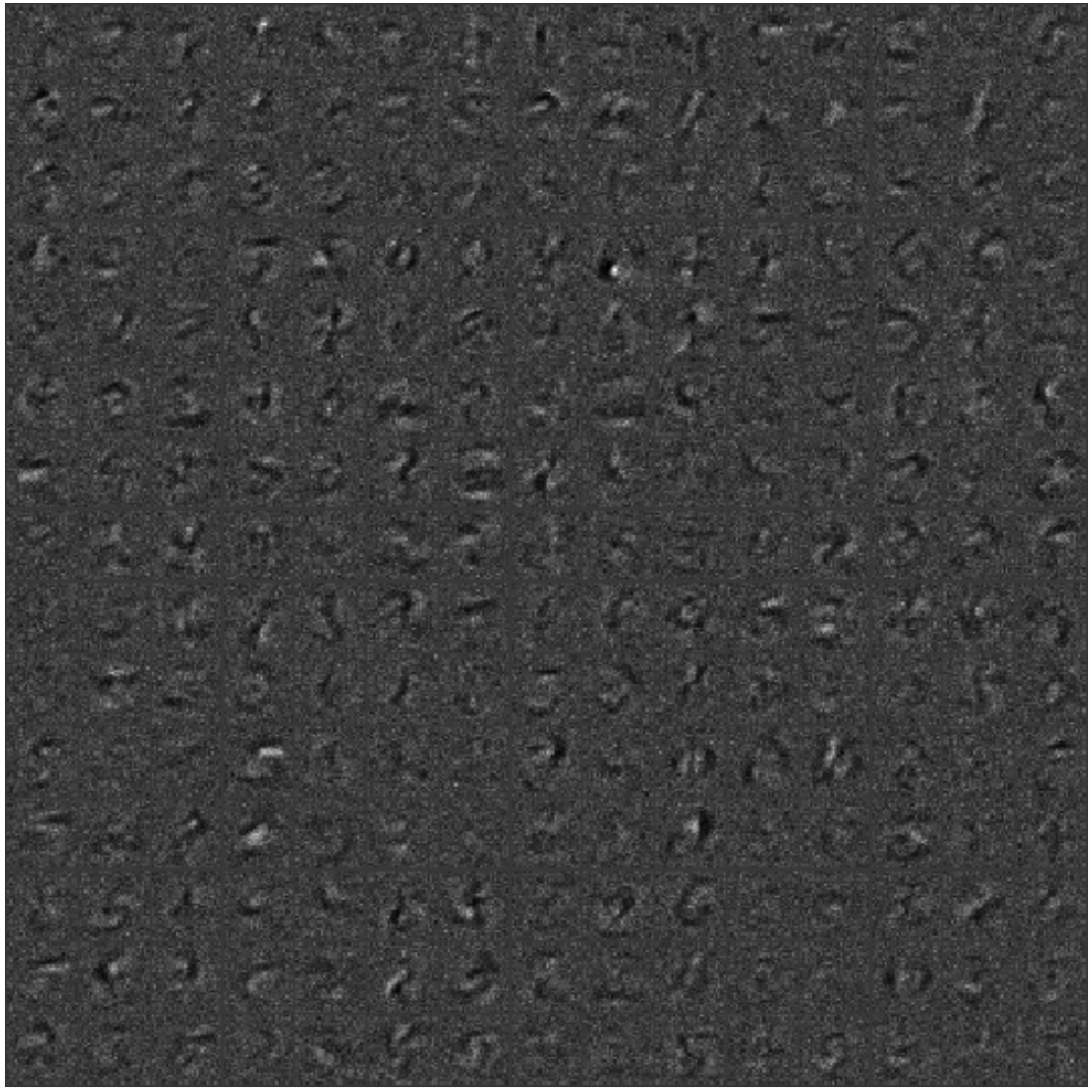
$\mathbf{W}^{(1)}$ after epoch 1



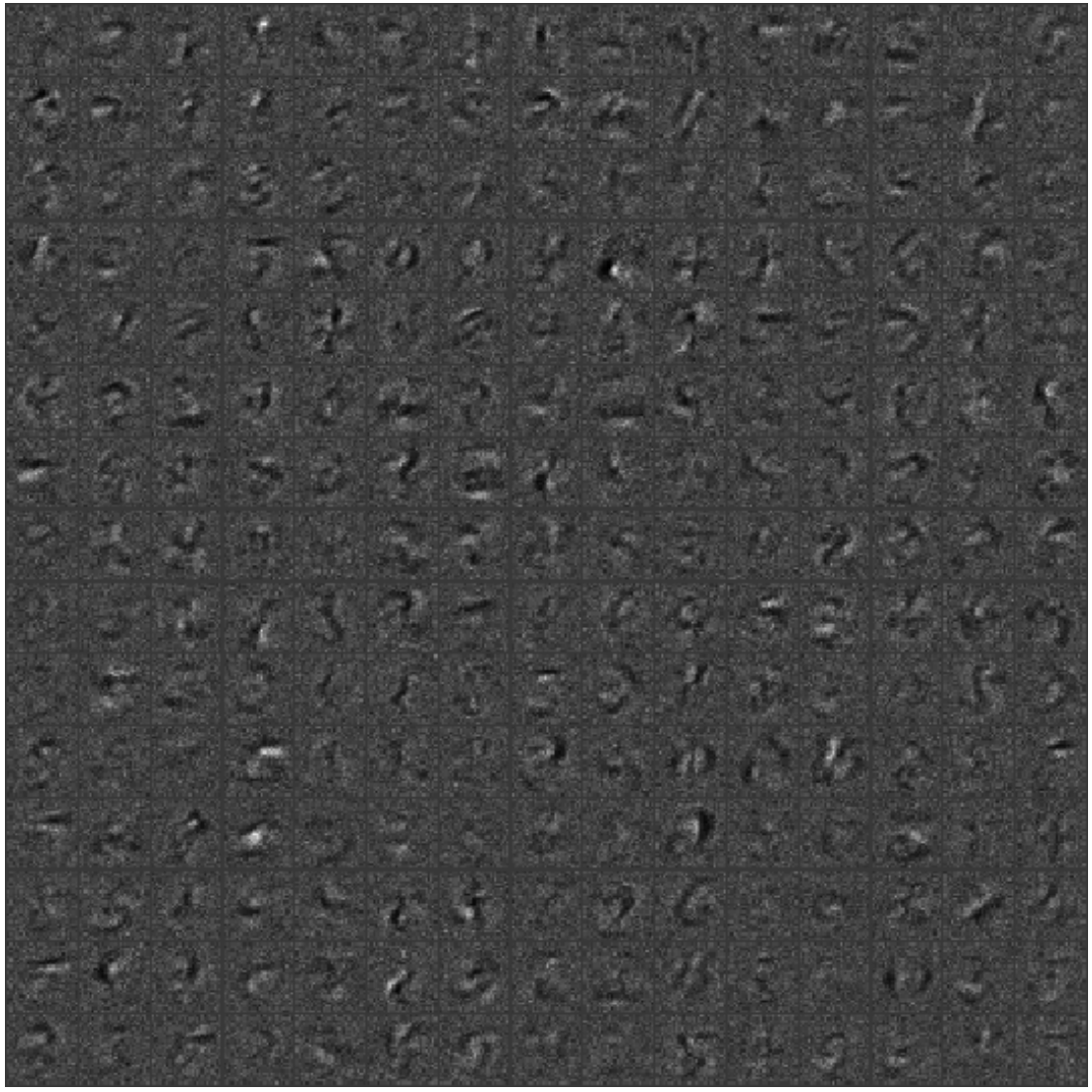
$\mathbf{W}^{(1)}$ after epoch 2



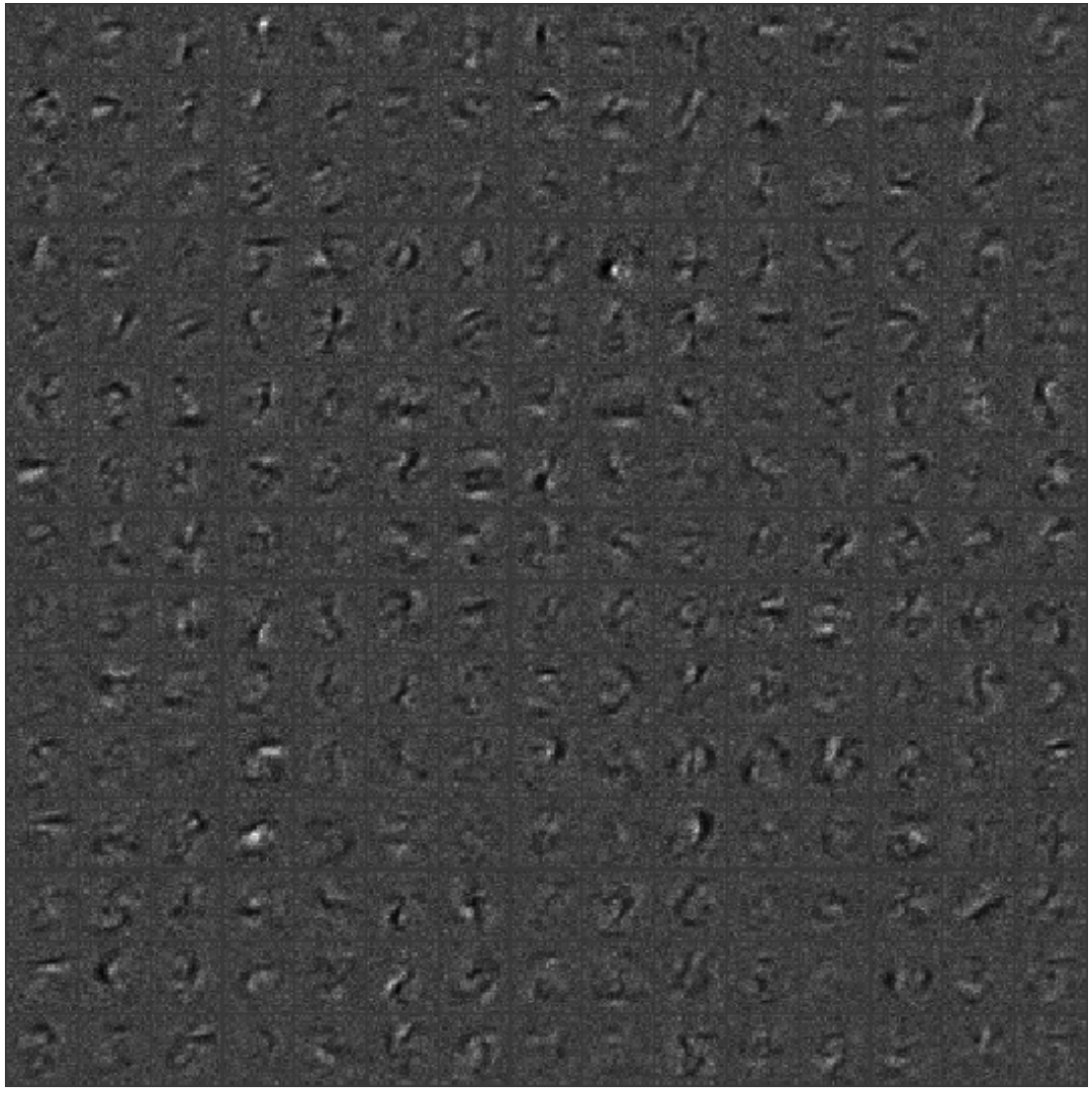
$\mathbf{W}^{(1)}$ after epoch 3



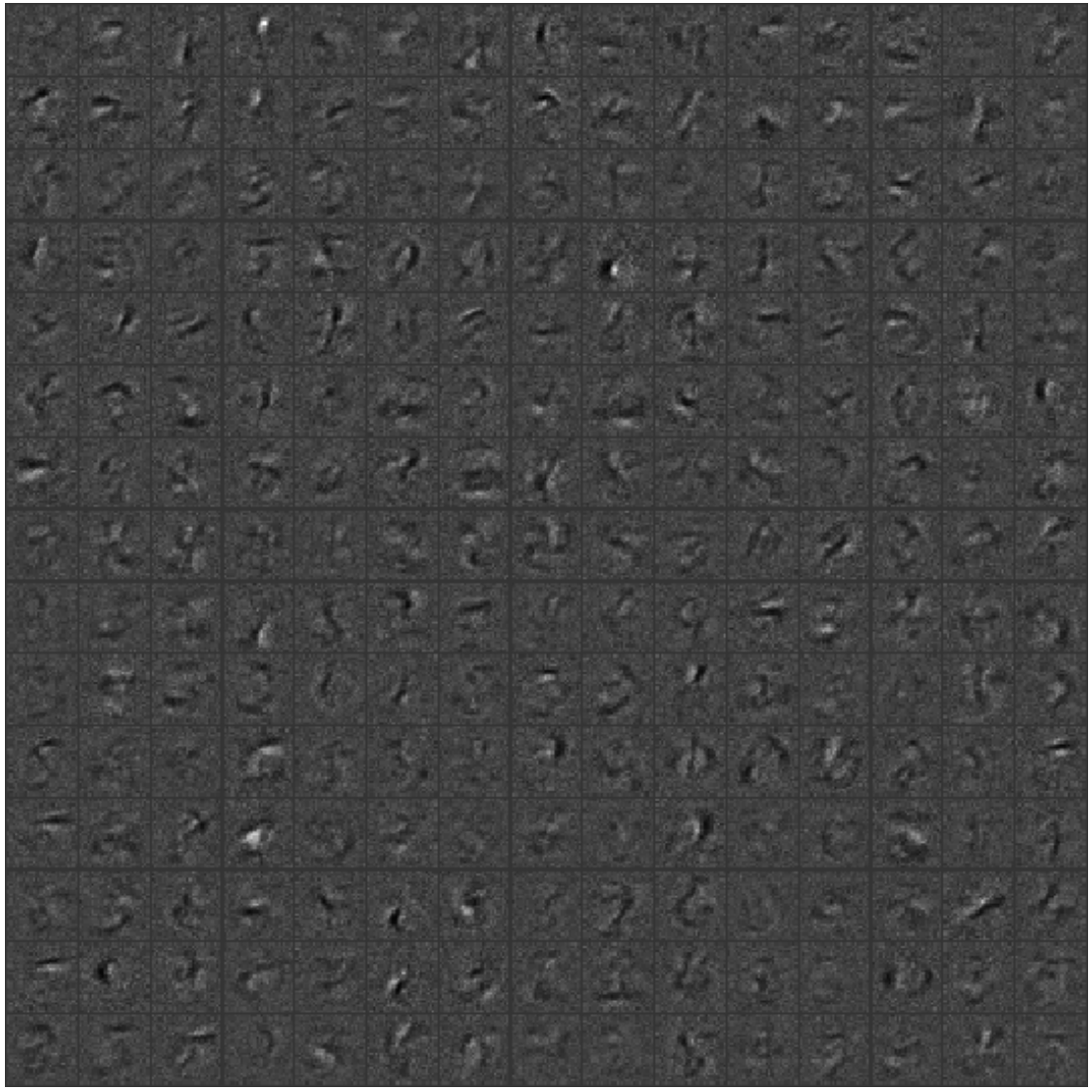
$\mathbf{W}^{(1)}$ after epoch 4



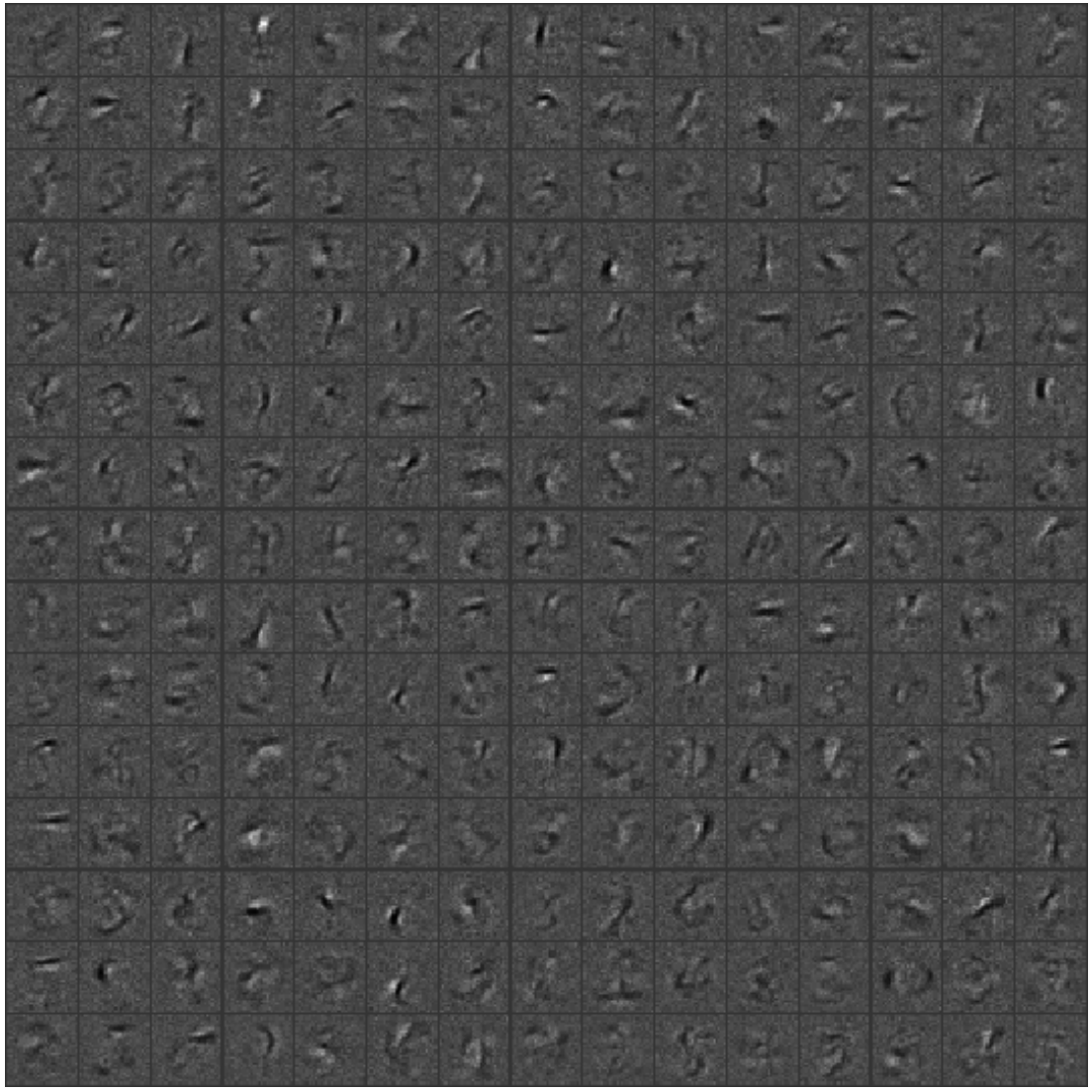
$\mathbf{W}^{(1)}$ after epoch 5



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



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



Adam algorithm by Kingma and Ba

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Adam algorithm by Kingma and Ba

- $\mathbb{E}[(\theta)]$ is the cost function we want to optimize
- $f(\theta_t)$ stochastic estimate of $\mathbb{E}[(\theta)]$, e.g. training cost on mini-batch.
- Momentum:

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta f(\theta_{t-1})$$

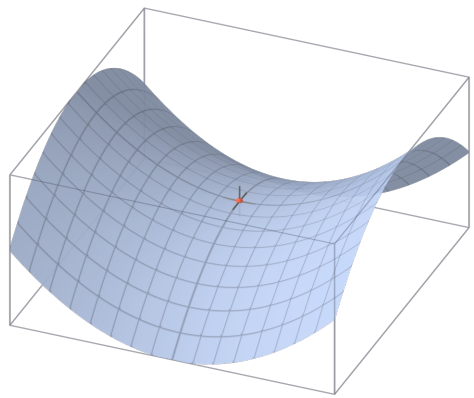
- Correct bias from $m_0 = 0$.
- Normalize stepsize with $\sqrt{\text{estimate of gradient}^2}$ so that

$$|\Delta\theta| \lesssim \alpha$$

- Never take step larger than α
- Take roughly equal steps in all directions, but $\beta_2 > \beta_1$ and $\epsilon > 0$ ensures that we will stop at minimum.
- Demo www.denizyuret.com/2015/03/alec-radfords-animations-for

Definitions

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

$$
\begin{aligned}
y &= w_2 h + \text{noise} \\
h &= w_1 x \\
C &= (y - 1.5)^2 \\
\frac{\partial C}{\partial w_2} &= 2(w_2 w_1 x - 1.5) w_1 x \\
\frac{\partial C}{\partial w_1} &= 2(w_2 w_1 x - 1.5) w_2 x
\end{aligned}
$$

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

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

- Exponential growth/decay of forward signals!

Exponential growth/decay backward

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

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

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

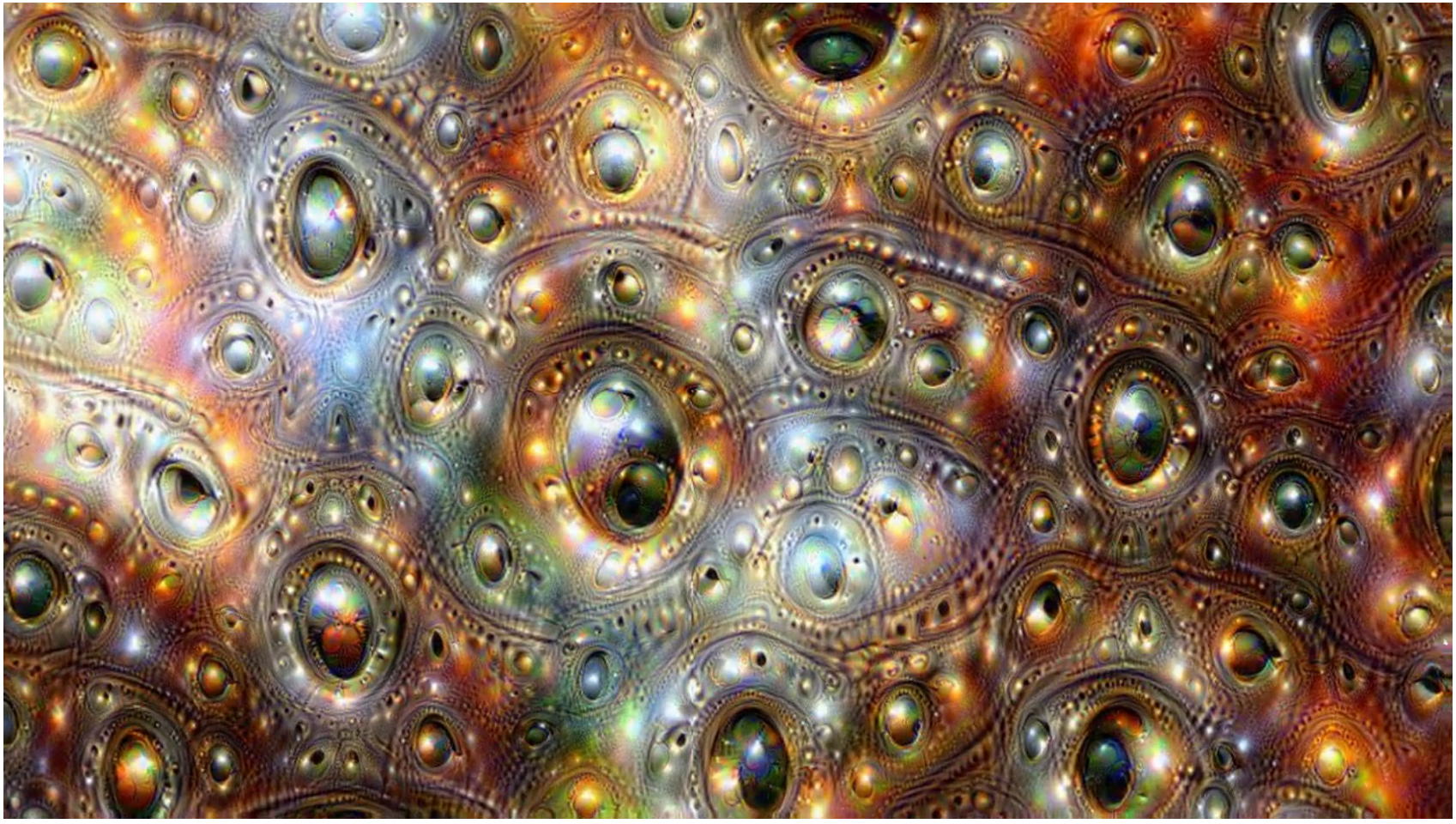
- Exponential growth/decay of gradient, too!
- $\Rightarrow$ Scale of initialization important.

References 1

- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton, Deep learning, Nature 521.7553 (2015): 436-444.
- Mnih, Volodymyr, et al., Human-level control through deep reinforcement learning, Nature 518.7540 (2015): 529-533.
- Alipanahi, Babak, et al., Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning, Nature biotechnology (2015).
- Silver, David, et al., Mastering the game of Go with deep neural networks and tree search, Nature 529.7587 (2016): 484-489.
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015), Show, attend and tell: Neural image caption generation with visual attention. arXiv preprint arXiv:1502.03044.
- Mansimov, Elman, et al., Generating Images from Captions with Attention. arXiv preprint arXiv:1511.02793 (2015).
- Larsen, Anders Boesen Lindbo, Soren Kaae Sonderby, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. arXiv preprint arXiv:1512.09300 (2015).
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. Book in preparation for MIT Press, `http://goodfeli.github.io/dlbook/`, 2016.
- Michael Nielsen, Neural Networks and Deep Learning, `http://neuralnetworksanddeeplearning.com/`
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional, NIPS, 2012. Neural Networks,

References 2

- 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.
- Saxe, Andrew M., James L. McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks." In ICLR 2014.
- Pascanu, R., "On Recurrent and Deep Neural Networks", PhD thesis, University of Montreal, 2014.
- Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." Proc. of ICML, 2015.
- T. Raiko, H. Valpola, and Y. LeCun. Deep Learning Made Easier by Linear Transformations in Perceptrons. Proc. of AISTATS, 2012.
- Y. LeCun, L. Bottou, G. Orr and K. Muller: Efficient BackProp, in Orr, G. and Muller K. (Eds), Neural Networks: Tricks of the trade, Springer, 1998.
- T. Vatanen, T. Raiko, H. Valpola, and Y. LeCun. Pushing Stochastic Gradient towards Second-Order Methods - Backpropagation Learning with Transformations in Nonlinearities. In Proc. ICONIP, 2013.



Thanks!
Ole Winther