Summer school on semisupervised learning

Variational learning part 1

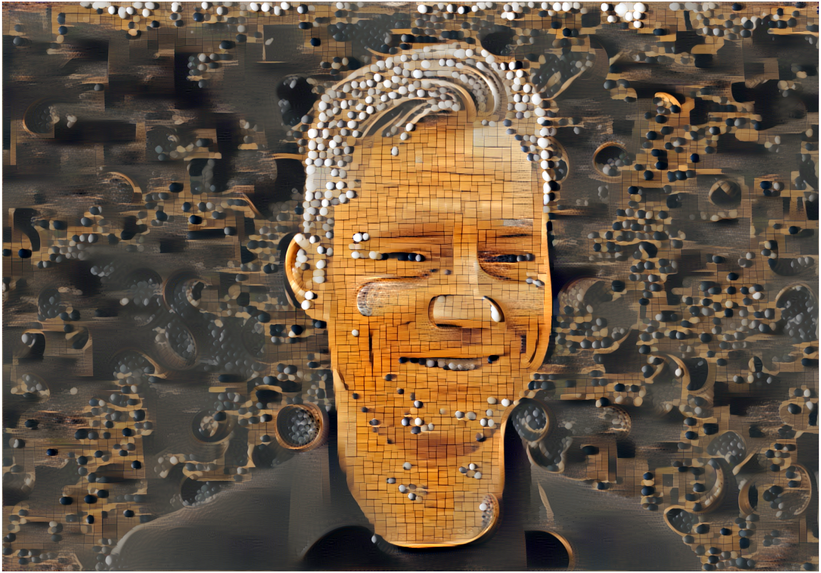
Deep learning

Ole Winther

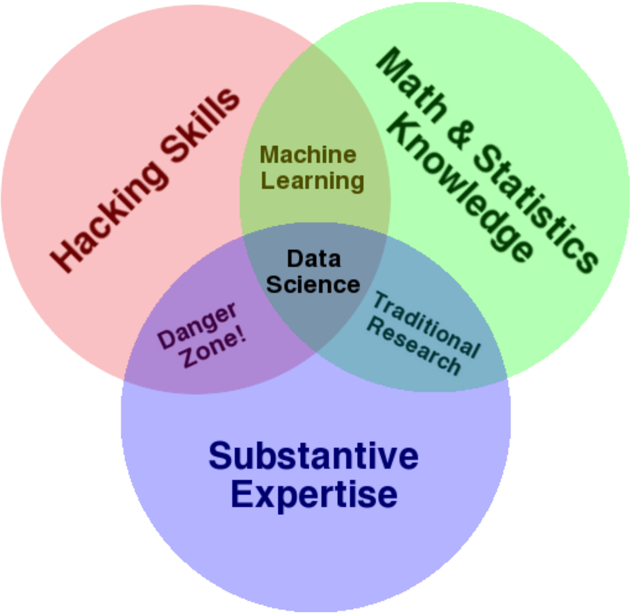
Dept for Applied Mathematics and Computer Science
Technical University of Denmark (DTU)

August 10, 2016

Ole Winther - a bit about myself



Data science - adding domain knowledge



Objectives of talk

- Neural network is used as building blocks in our semi-supervised models.
- A bit about deep learning and statistical artificial intelligence?
- How does it work?
- The feed-forward neural network (FFNN)

What is in it for you - if you pay attention to the end

- Learn about a probabilistic approach to
  - density modelling:

    $$p(x)$$

  - classification:

    $$p(y|x)$$

- with deep neural networks.

What is in it for you - if you pay attention to the end

- Learn about a probabilistic approach to
  - density modelling:
    $$p(x)$$
  - classification:
    $$p(y|x)$$
- with deep neural networks.
- Latent variable z models:
  $$p(x|z)p(z)$$

What is in it for you - if you pay attention to the end

- Learn about a probabilistic approach to
  - density modelling:

    $$p(x)$$

  - classification:

    $$p(y|x)$$

- with deep neural networks.
- Latent variable z models:

  $$p(x|z)p(z)$$

- Variational learning

What is in it for you - if you pay attention to the end

- Learn about a probabilistic approach to
  - density modelling:

$$p(x)$$

  - classification:

$$p(y|x)$$

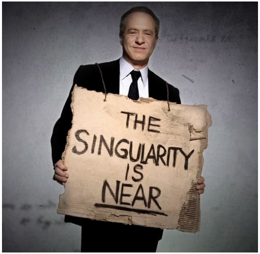
- with deep neural networks.
- Latent variable z models:

$$p(x|z)p(z)$$

- Variational learning
- State-of-the-art performance semi-supervised learning image benchmarks...

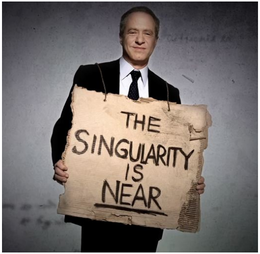
	MNIST 100 labels	SVHN 1000 labels	NORB 1000 labels
M1+TSVM (Kingma et al., 2014)	11.82% (±0.25)	55.33% (±0.11)	18.79% (±0.05)
M1+M2 (Kingma et al., 2014)	3.33% (±0.14)	36.02% (±0.10)	
VAT (Miyato, 2015)	2.12 %	24.63 %	9.88 %
Ladder Network (Rasmus et al., 2015)	1.06% (±0.37)		
Auxiliary Deep Generative Model	**0.96% (±0.02)**	22.86 %	10.06% (±0.05)
Skip Deep Generative Model	1.32% (±0.07)	**16.61% (±0.24)**	**9.40% (±0.04)**

Are we heading towards the singularity?



kurzweilai.net

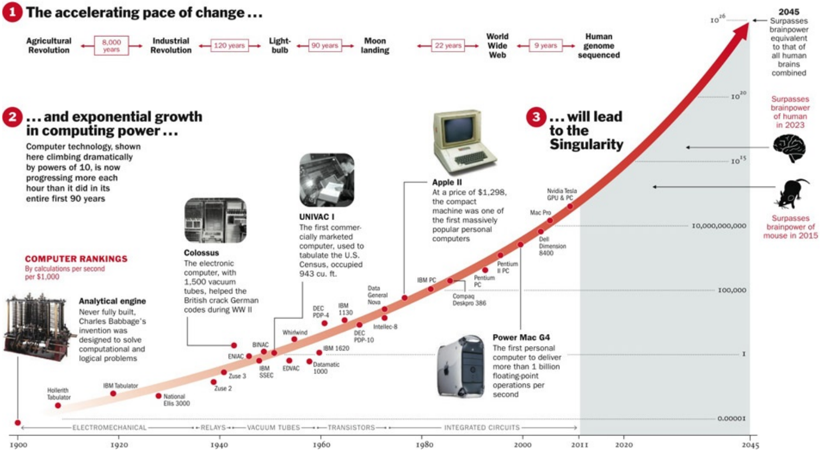
Are we heading towards the singularity?



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.

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:

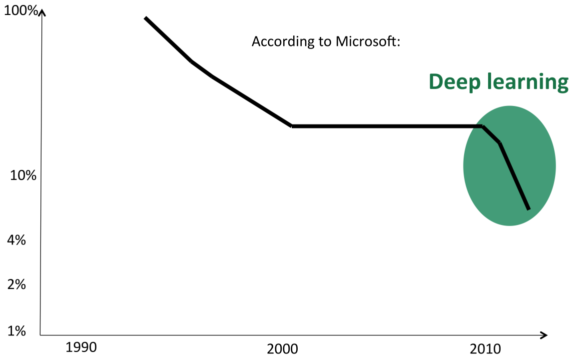
The deep learning revolution

Achilles' heel of traditional AI: Perception in natural environment

xkcd.com/1425

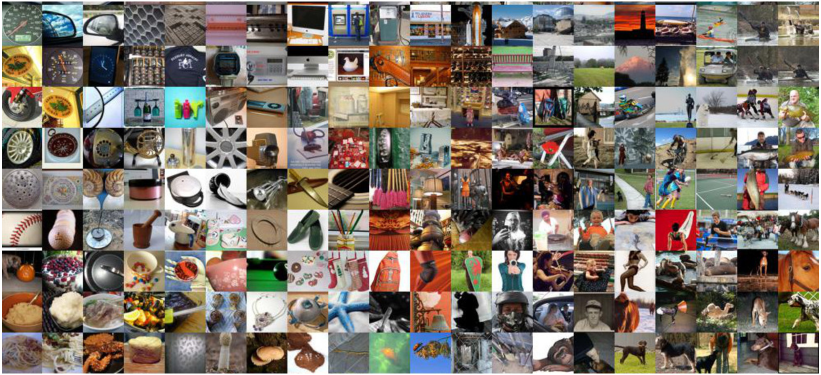
Many thanks to Tapani Raiko for sharing slides!

Speech recognition breakthrough



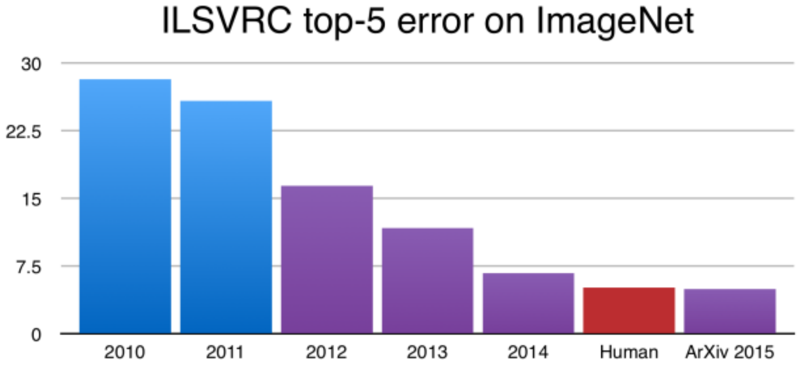
Plot from Yoshua Bengio

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.

Modifying visual features (Larsen et al., 2015)

Representation learning

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

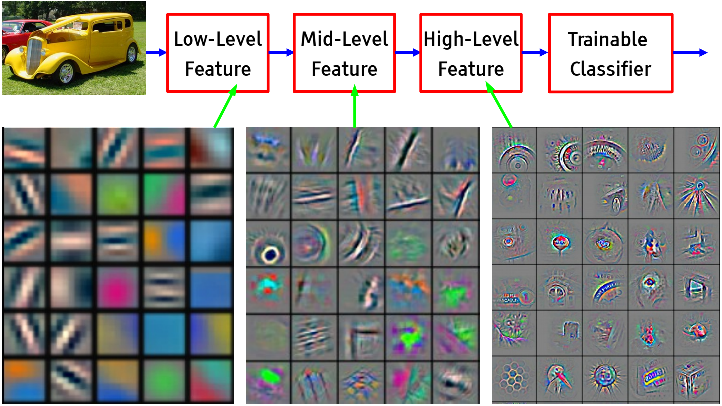
- Feature selection
- Feature extraction (e.g. PCA)
- Feature construction (e.g. SIFT)

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.

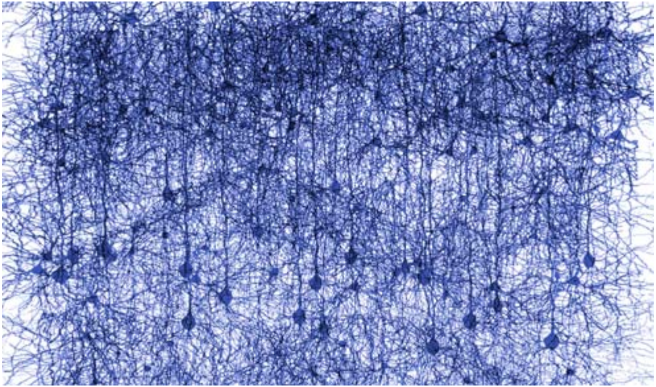
- Book: M.Nielsen, Neural networks and deep learning
- Book, draft available online:

- Portal: `deeplearning.net`
- DTU fall term master-level course: 02456 Deep learning,

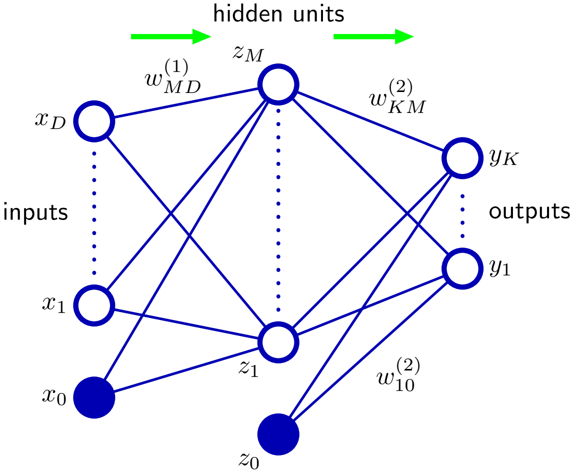
Part 2:
Neural networks

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

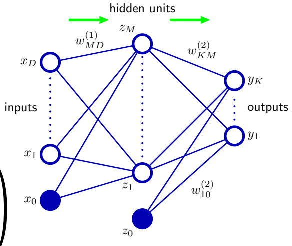
- Compute weighted sum of inputs:

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

- Output k two-layer network:

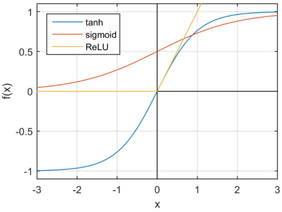
$$h_k^{(2)}(\mathbf{x}, \mathbf{w}) = f_2 \left(\sum_{j=0}^{M} w_{kj}^{(2)} f_1 \left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right)$$

- f_1 and f_2 hidden unit activation functions



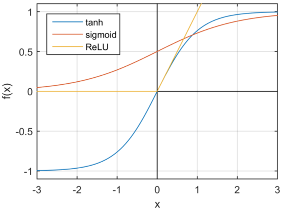
Non-linearity and training

- 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 $\mathrm{relu}(a) = \max(0, a)$

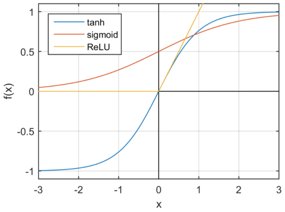


- Supervised learning
- Labeled training set

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

- 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 $\mathrm{relu}(a) = \max(0, a)$

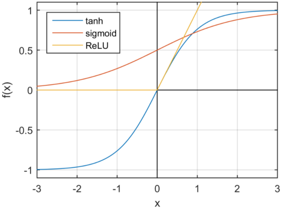
- Supervised learning
- Labeled training set

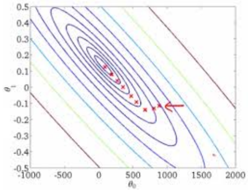
$$\mathcal{D} = \{(x_i, y_i) | i = 1, \ldots, n\} \, .$$

- 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 $\mathrm{relu}(a) = \max(0, a)$





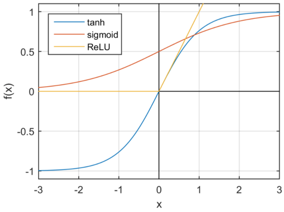
- Supervised learning
- Labeled training set

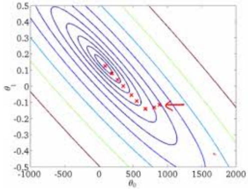
$$\mathcal{D} = \{(x_i, y_i) | i = 1, \ldots, n\} \ .$$

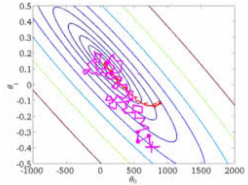
- Input x_i and output y_i.
- Minimize training error by (stochastic) gradient descent

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 $\mathrm{relu}(a) = \max(0, a)$







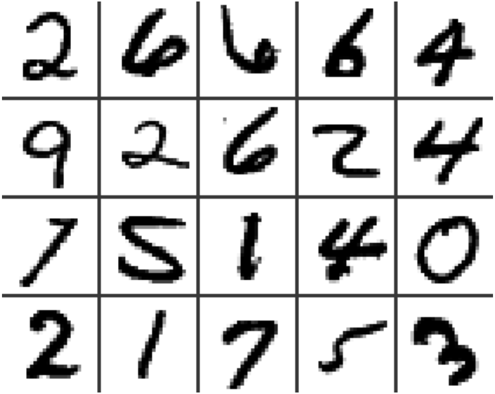
- Supervised learning
- Labeled training set

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

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

Overfitting!

Example: MNIST handwritten digits

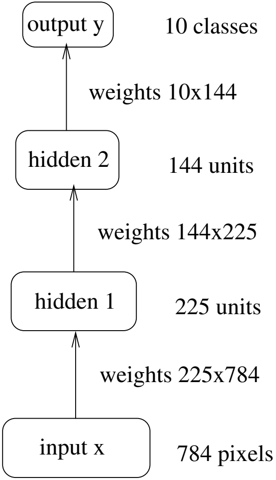


Train a network to classify 28×28 images.

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

Example model gives around 1.2% test error.

Example Network



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

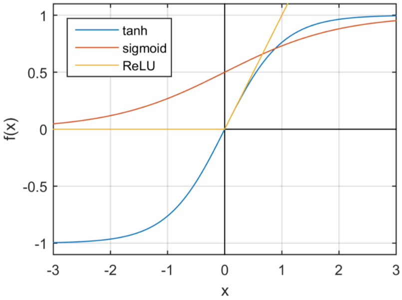
$$\mathbf{h}^{(2)} = \text{relu}(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

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

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

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

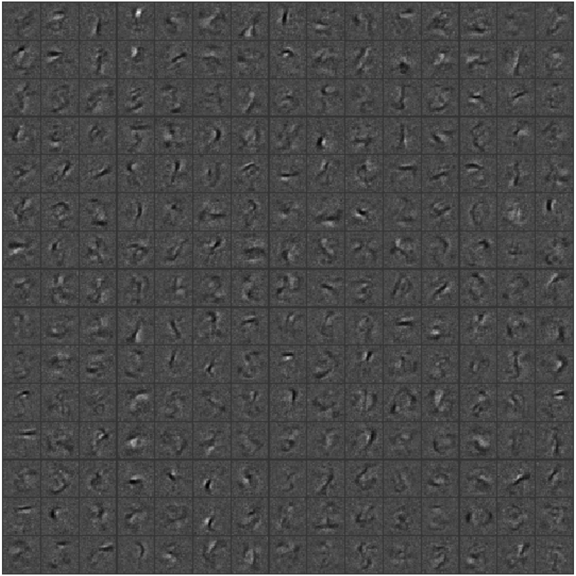
On activation functions



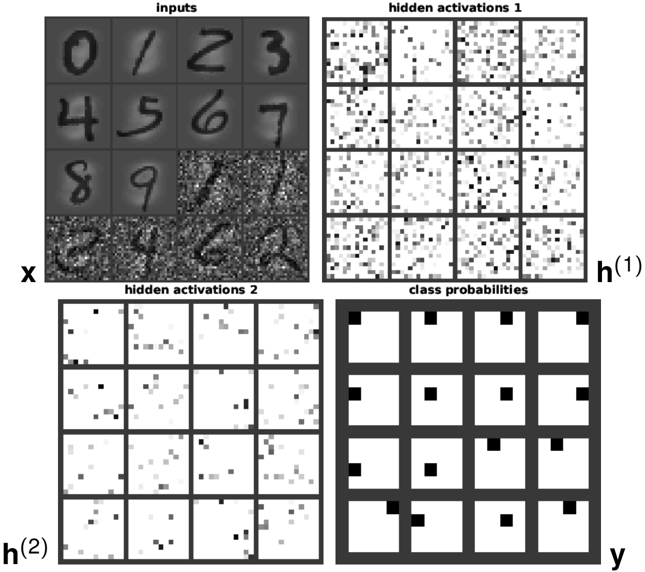
- $\text{relu}(z) = \max(0, z)$ is replacing old sigmoid and tanh.
- Note that identity function would lead into:

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

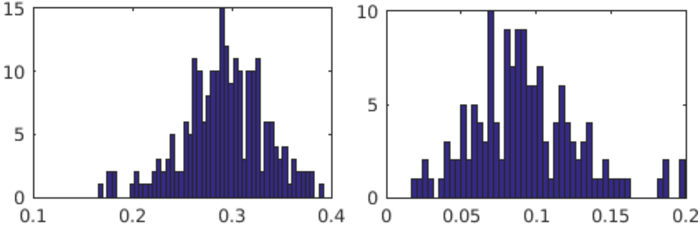
Weight matrix $\mathbf{W}^{(1)}$ size 225×784



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



On sparsity



$\mathbf{h}^{(1)}$ $\mathbf{h}^{(2)}$

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

Part 3:
Neural network training

Training criterion

Find parameters

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

that minimize expected negative log-likelihood:

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

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.

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

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:

$$
\begin{aligned}
0 &\rightarrow \mathbf{y} = (1,0,0,0,0,0,0,0,0,0)^T \\
1 &\rightarrow \mathbf{y} = (0,1,0,0,0,0,0,0,0,0)^T \\
2 &\rightarrow \mathbf{y} = (0,0,1,0,0,0,0,0,0,0)^T \\
&\ldots. \\
9 &\rightarrow \mathbf{y} = (0,0,0,0,0,0,0,0,0,1)^T
\end{aligned}
$$

- 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

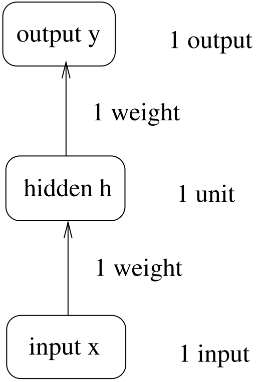
$$C = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \log h_{nk}^{(3)}$$

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.

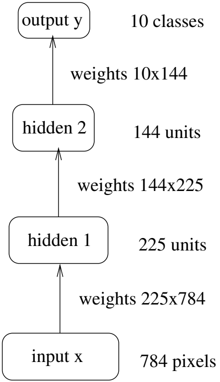


- First with scalars. Use chain rule:

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

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

Backpropagation



- Multi-dimensional:

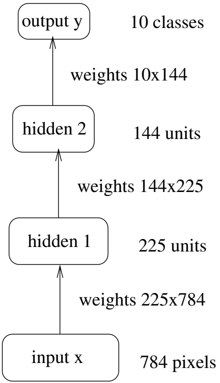
$$\frac{\partial C}{\partial W_{ij}^{(3)}} = \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial W_{ij}^{(3)}}$$

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

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

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

Backpropagation



- Multi-dimensional:

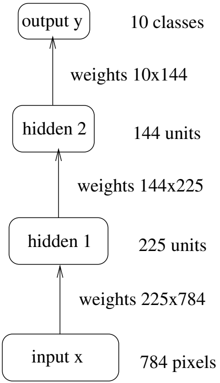
$$\frac{\partial C}{\partial W_{ij}^{(3)}} = \frac{\partial C}{\partial h_i^{(3)}} \frac{\partial h_i^{(3)}}{\partial W_{ij}^{(3)}}$$

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

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

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

Backpropagation - dynamic programming



- Store intermediate results

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

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

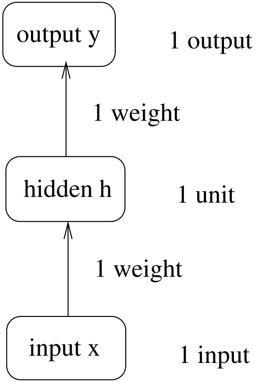
- In general

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

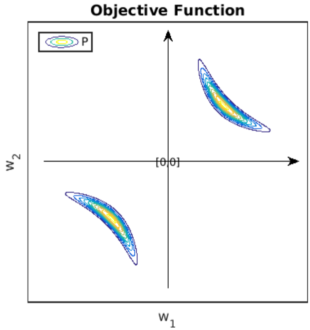
- and gradient:

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

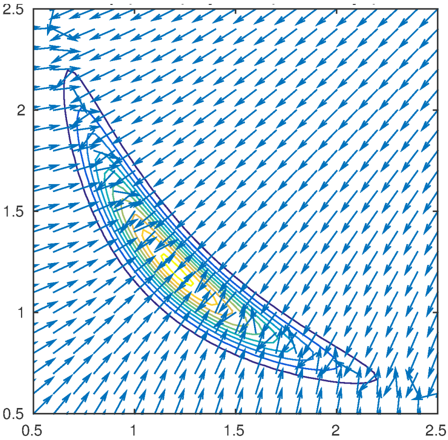
Tiny Example



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

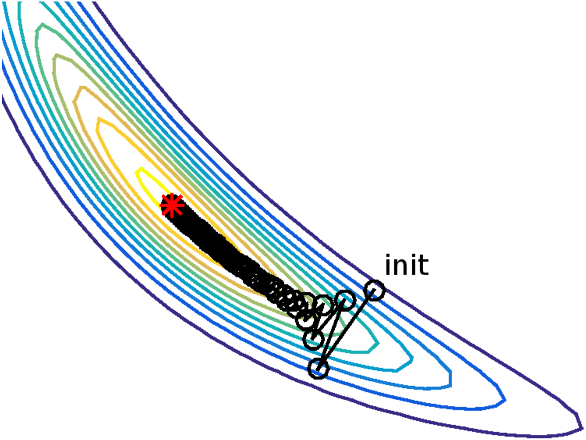


$$\text{Gradient } \mathbf{g} = \nabla_\theta C(\theta) = \begin{pmatrix} \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_n} \end{pmatrix}$$



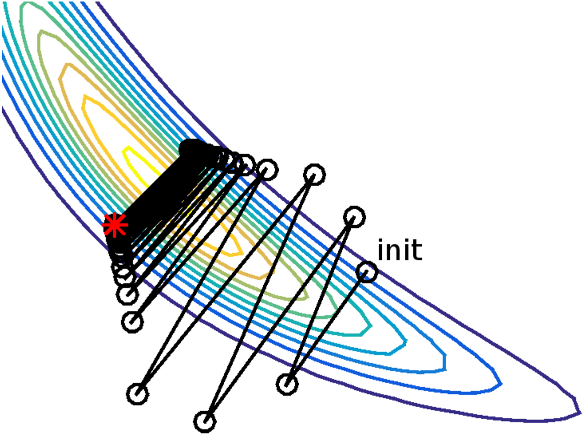
Gradient descent, $\eta_k = 0.25$ ($\rightarrow$ too slow)

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



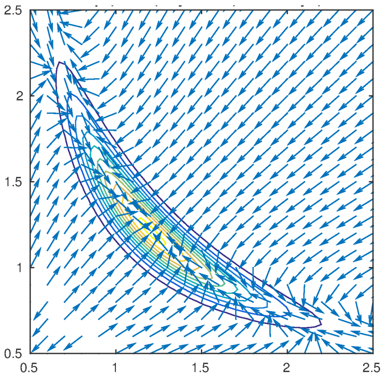
Gradient descent, $\eta_k = 0.35$ ($\rightarrow$ oscillates)

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k$$



Newton's method, too complex

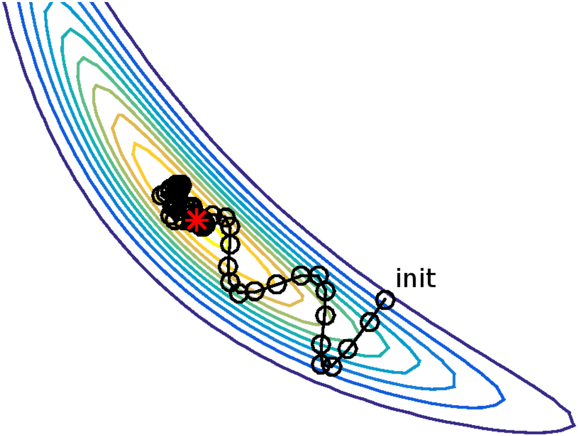
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}_k^{-1}\mathbf{g}_k, \quad \mathbf{H} = \begin{pmatrix} \frac{\partial^2 C}{\partial\theta_1\partial\theta_1} & \cdots & \frac{\partial^2 C}{\partial\theta_1\partial\theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 C}{\partial\theta_n\partial\theta_1}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



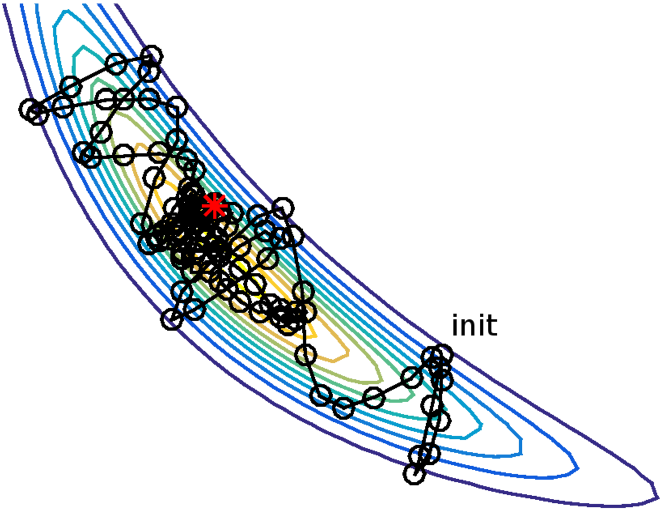
- Less oscillations.
- Points to the wrong direction in places (solvable).
- Computational complexity: #params^3 (prohibitive).
- There are approximations, but not very popular.

Momentum method (Polyak, 1964)

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



Momentum method with noisy gradient

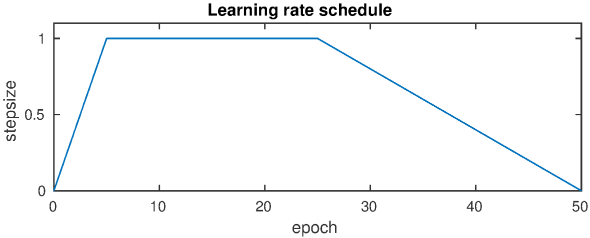


Mini-batch training

- 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.

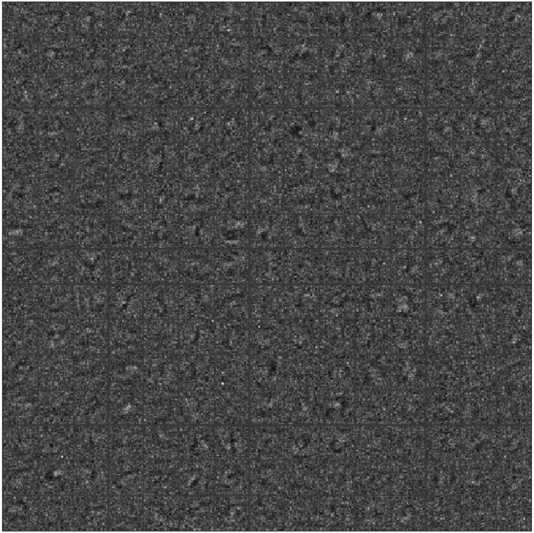
Mini-batch training

- 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.

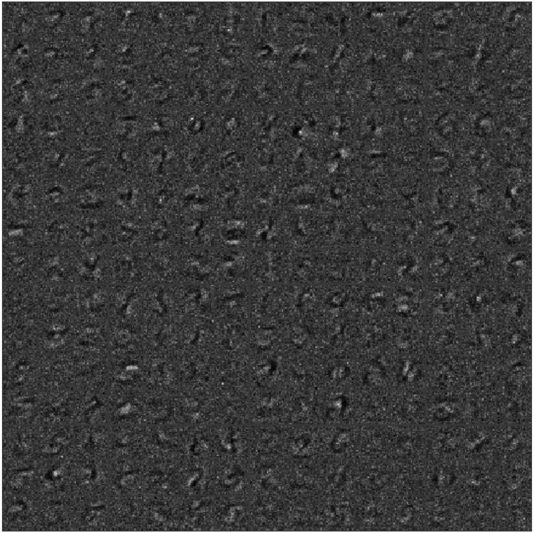


- Adaptation of η_k possible (Adam, Adagrad, Adadelta).

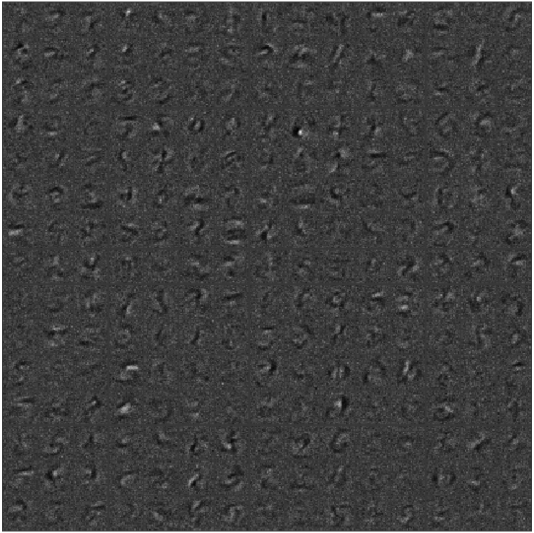
$\mathbf{W}^{(1)}$ after epoch 1



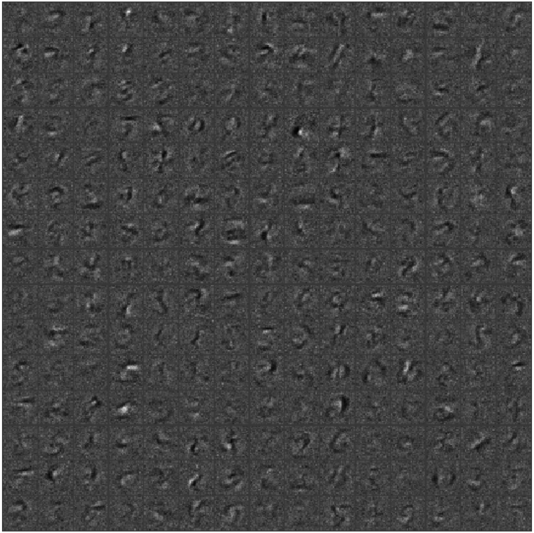
$\mathbf{W}^{(1)}$ after epoch 2



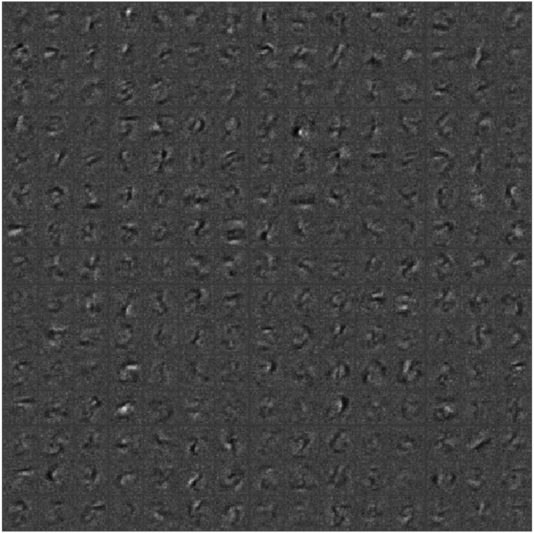
$\mathbf{W}^{(1)}$ after epoch 3



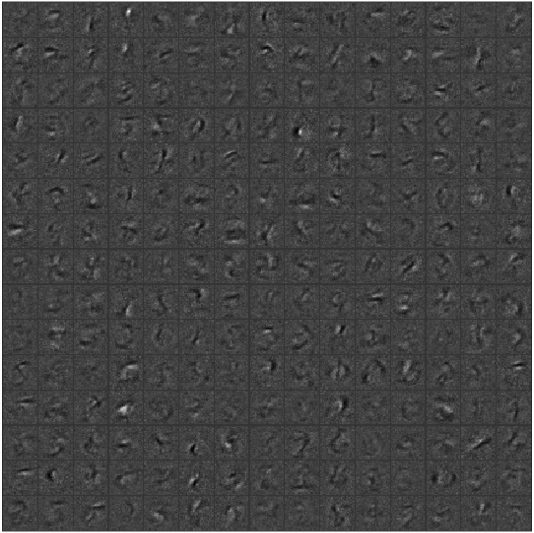
$\mathbf{W}^{(1)}$ after epoch 4



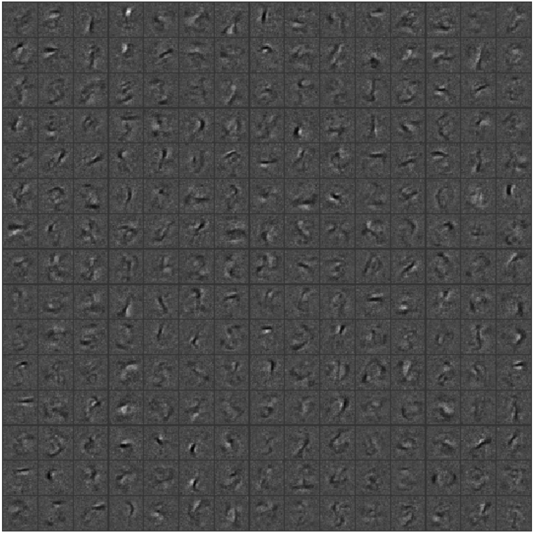
$\mathbf{W}^{(1)}$ after epoch 5



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

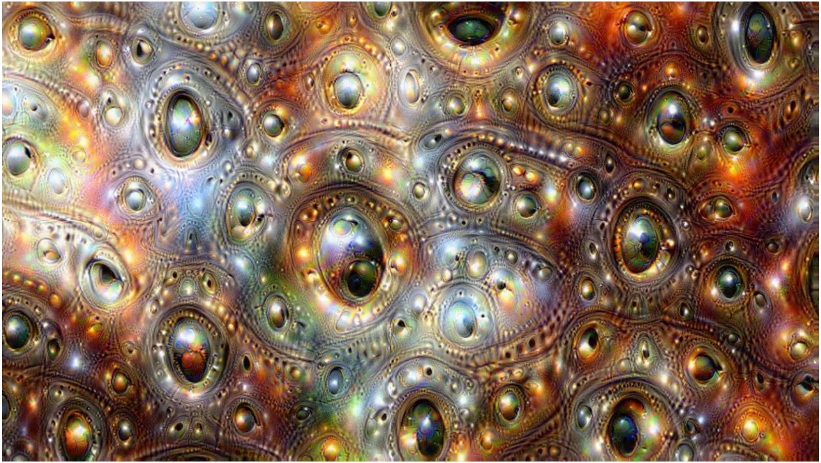


$\mathbf{W}^{(1)}$ after epoch 50 (final)



References

- 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,



Thanks!
Ole Winther