Universität Stuttgart
KI – Institute for Artificial Intelligence
Analytic Computing

Machine Learning
11 Neural Networks Part 2

Prof. Dr. Steffen Staab
Nadeen Fatallah
Daniel Frank
Akram Sadat Hosseini
Jiaxin Pan
Osama Mohamed
Arvindh Arunbabu
Tim Schneider
Yi Wang

https://www.ki.uni-stuttgart.de/

Learning Objectives

- Improved solving
  - Stochastic gradient descent
  - Newton's method
    - Hessian matrix
  - Momentum and Nestorov Accelerated Gradient
- Regularization of Neural Networks
  - big topic, many methods
  - several methods transfer to other machine learning methods!
- Convolutional Neural Networks
- Recurrent Neural Networks

1 Optimizing Gradient Descent

Initialization

- The Initialization of weights is important! Heuristics:

- Choose random weights that don't grow or vanish the gradient:
  - E.g., initialize weight vectors in $W_{l,i\cdot}$ with standard deviation 1, i.e., each entry with sdv $\frac{1}{\sqrt{h_{l\text{-}1}}}$
  - Roughly: If each element of z_l has standard deviation ϵ, the same should be true for z_{l+1}.

h_l is the width of layer l

- Choose each weight vector $W_{l,i\cdot}$ to point in a uniform random direction $\rightarrow$ same as above

- Choose biases $b_{l,i}$ randomly so that the ReLU hinges cover the input well (think of distributing hinge features for continuous piece-wise linear regression)

Optimization

Approximation of Newton‘s method

- For small data size:
  We can compute the loss and its gradient $\sum_{i=1}^{n} \nabla_\beta \ell(f_\beta(x_i), y_i)$.
  - Use classical gradient-based optimization methods
  - default: L-BFGS,

Newton‘s method in optimization (in one dimension)

Minimize $f(x)$: Iterate $\mathrm{t} = 0,1,..$ for x_t to approach minimum

Taylor expansion:

$$f(x_t + \varepsilon) \approx f(x_t) + f'(x_t)\varepsilon + \frac{1}{2}f''(x_t)\varepsilon^2$$

Derive and set 0

$$0 = \frac{d}{d\varepsilon}\left(f(x_t) + f'(x_t)\varepsilon + \frac{1}{2}f''(x_t)\varepsilon^2\right) = f'(x_t) + f''(x_t)\varepsilon$$

$$\varepsilon = -\frac{f'(x_t)}{f''(x_t)}$$

Thus: $$x_{t+1} = x_t + \varepsilon = x_t - \frac{f'(x_t)}{f''(x_t)}$$

Newton‘s method in optimization applies Newton‘s method for finding y=0 to the first derivation

Newton‘s method in several dimensions

Minimize $f(x)$: Iterate $t = 0,1,..$ for x_t to approach minimum, with $x, x_t \in \mathbb{R}^d$

Taylor expansion and setting gradients to zero:

Considers curvature to reach target more quickly

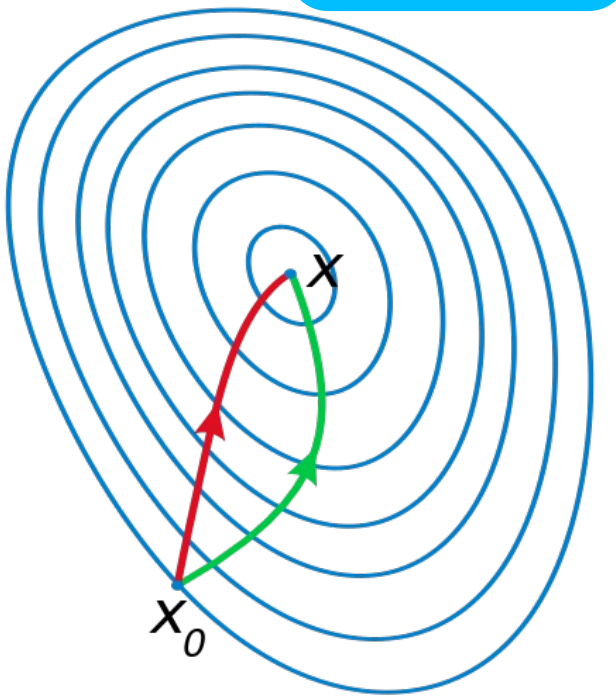
Gradient: $g_f(x) = \nabla f(x) = \left(\dots, \frac{\partial f(x_1,\dots,x_i,\dots)}{\partial x_i}, \dots\right)^T$

Hessian: $H_f(x) = \nabla^2 f(x) = \begin{pmatrix} \ddots & \vdots & \\ \dots & \frac{\partial^2 f(x_1,\dots,x_i,\dots)}{\partial x_j \partial x_i} & \dots \\ & \vdots & \ddots \end{pmatrix}$

Thus: $x_{t+1} = x_t - \eta\left[H_f(x_t)\right]^{-1} g_f(x_t)$

$g_f(x_t) \in \mathbb{R}^d, \qquad H_f(x_t) \in \mathbb{R}^{d\times d}$

$\eta < 1$ dampens the search



https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization#/media/File:Newton_optimization_vs_grad_descent.svg

Illustrating the Hessian

- Hessian is symmetric
- Largest and smallest Eigenvector indicate largest and smallest curvature

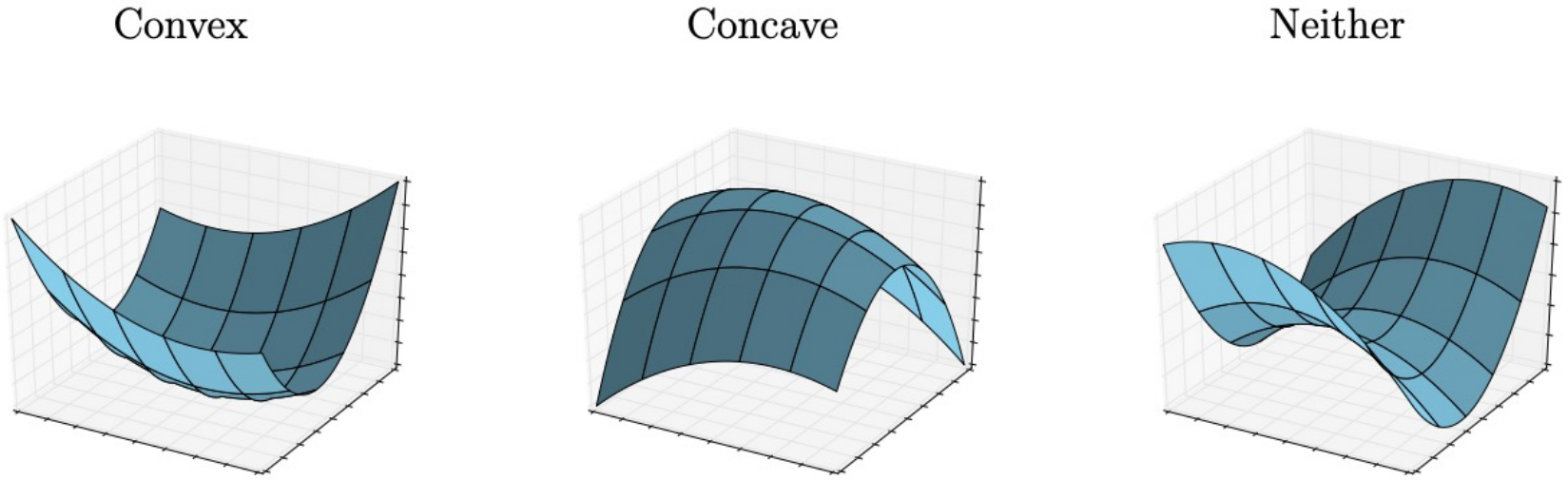


Figure 8: Quadratic forms for which the Hessian is positive definite (left), negative definite (center) and neither positive nor negative definite (right).

https://cims.nyu.edu/~cfgranda/pages/OBDA_fall17/notes/convex_optimization.pdf

Computing the Hessian matrix

- Inverting the Hessian (inverting any matrix) is expensive and should be avoided
  - Do not compute: $h(x_t) = \left[H_f(x_t)\right]^{-1} g_f(x_t)$
  - Rather solve linear equation: $H_f(x_t)\, h(x_t) = \; g_f(x_t)$

- mini batches basically produce the hessian

Convergence

- For strongly convex functions with Lipschitz Hessian matrix,

$$\|\nabla^2 f(x) - \nabla^2 f(y)\|_{op} \leq L \parallel x - y \parallel_2$$

  we have (quadratic) convergence

  - Convexity generally not given for neural networks,
    but for other machine learning methods (e.g. commonly for SVM)

Convergence typically not guaranteed for neural networks, but practically the high dimensionality often allows for avoiding – many – local minima

Optimization

- For small data size:
  We can compute the loss and its gradient $\sum_{i=1}^{n} \nabla_\beta \ell(f_\beta(x_i), y_i)$.
  - Use classical gradient-based optimization methods
  - default: L-BFGS, oldish but efficient: Rprop
  - Called **batch learning** (in contrast to online learning)

- For large data size: The $\sum_{i=1}^{N}$ is highly inefficient!
  - Adapt weights based on much smaller data subsets, **mini batches**

Stochastic Gradient Descent

- Compute the loss and gradient for a mini batch $\hat{D} \subset D$ of fixed size k.

$$L(\beta, \hat{D}) = \sum_{i \in \hat{D}} \ell(f_\beta(x_i), y_i)$$

$$\nabla_\beta L(\beta, \hat{D}) = \sum_{i \in \hat{D}} \nabla_\beta \ell(f_\beta(x_i), y_i)$$

- Naive Stochastic Gradient Descent, iterate

$$\beta \leftarrow \beta - \eta \nabla_\beta L(\beta, \hat{D})$$

  - Choice of learning rate η is crucial for convergence!
  - Exponential cooling: $\eta = \eta_0^t$

Stochastic Gradient Descent with Momentum (alternative to Newton methods)

SGD with Momentum can be linked to modeling Gradient Descent as control problem!

- SGD with momentum:

$$\Delta\beta \leftarrow \alpha\Delta\beta - \eta\nabla_\beta L(\beta, \hat{D})$$
$$\beta \leftarrow \beta + \Delta\beta$$

- Nesterov Accelerated Gradient ("Nesterov Momentum"):

$$\Delta\beta \leftarrow \alpha\Delta\beta - \eta\nabla_\beta L(\beta + \Delta\beta, \hat{D})$$
$$\beta \leftarrow \beta + \Delta\beta$$

Yurii Nesterov (1983): *A method for solving the convex programming problm with convergence rate* $O(1/k^2)$

Momentum vs Nestorov Accelerated Gradient

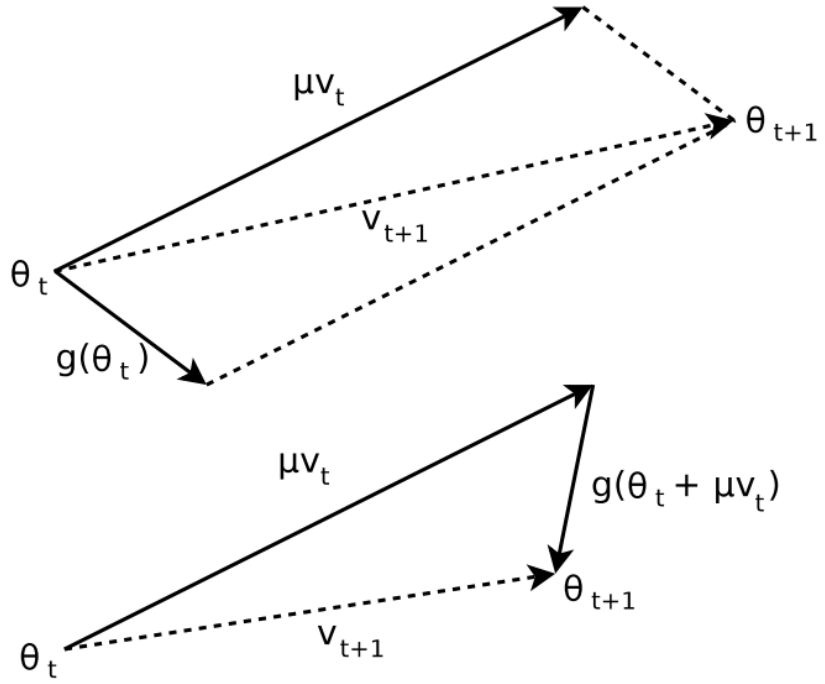
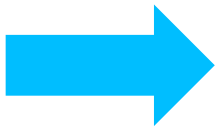


Figure 1. **(Top)** Classical Momentum **(Bottom)** Nesterov Accelerated Gradient

Ilya Sutskever, James Martens, George E. Dahl, Geoffrey E. Hinton:
On the importance of initialization and momentum in deep learning. Int. Conf. on Machine Learning - ICML 2013: 1139-1147

Trade-offs

- Higher-order methods promise faster convergence



- Many optimizations are noisy, because of
  - data subsampling (mini-batches)
  - regularization (e.g. drop-out, see later slides)

- First-order methods (with momentum) OR Stochastic Newton methods preferred

Diederik P. Kingma, Jimmy Ba: Adam: A Method for Stochastic Optimization. Int. Conf. on Learning Representations, 2015.
https://arxiv.org/abs/1412.6980

2 ADAM – Adaptive Moment Estimation

I mention this in passing.
This unit on ADAM is not exercise or exam material

How to set step size/learning rate η?

- Too small
  - very slow convergence
- Too large
  - may „jump over“ optimal value
  - instable

I mention this in passing.
This unit on ADAM is not exercise or exam material

Adam

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 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd 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)

arXiv:1412.6980

(all operations interpreted element-wise)

Adam & Nadam

- Adam interpretations (everything element-wise!):
  - $m_t \approx \langle g \rangle$ the mean gradient in the recent iterations
  - $v_t \approx \langle g^2 \rangle$ the mean gradient-square in the recent iterations
  - $\hat{m}_t, \hat{v}_t$ are bias corrected (check: in first iteration, $t = 1$, we have $\hat{m}_t = g_t$, unbiased, as desired)
  - $\Delta\theta \approx -\alpha \frac{\langle g \rangle}{\langle g^2 \rangle}$ *would* be a Newton step if $\langle g^2 \rangle$ *were* the Hessian...

I mention this in passing.
This unit on ADAM is not exercise or exam material

Extensions

- Including Nestorovs accelerated gradient into ADAM („NADAM“)
  - Timothy Dozat, Incorporating Nesterov Momentum into Adam, ICLR Workshop track, 2016
- Changing the „remembering“ rate
  - Sashank J. Reddi, Satyen Kale, Sanjiv Kumar: On the Convergence of Adam and Beyond. ICLR 2018
- Plenty of others
  - ...

I mention this in passing.
This unit on ADAM is not exercise or exam material

From
Yousseff Touti: Accelerated Schemes in Optimisation, Master thesis, Univ Potsdam

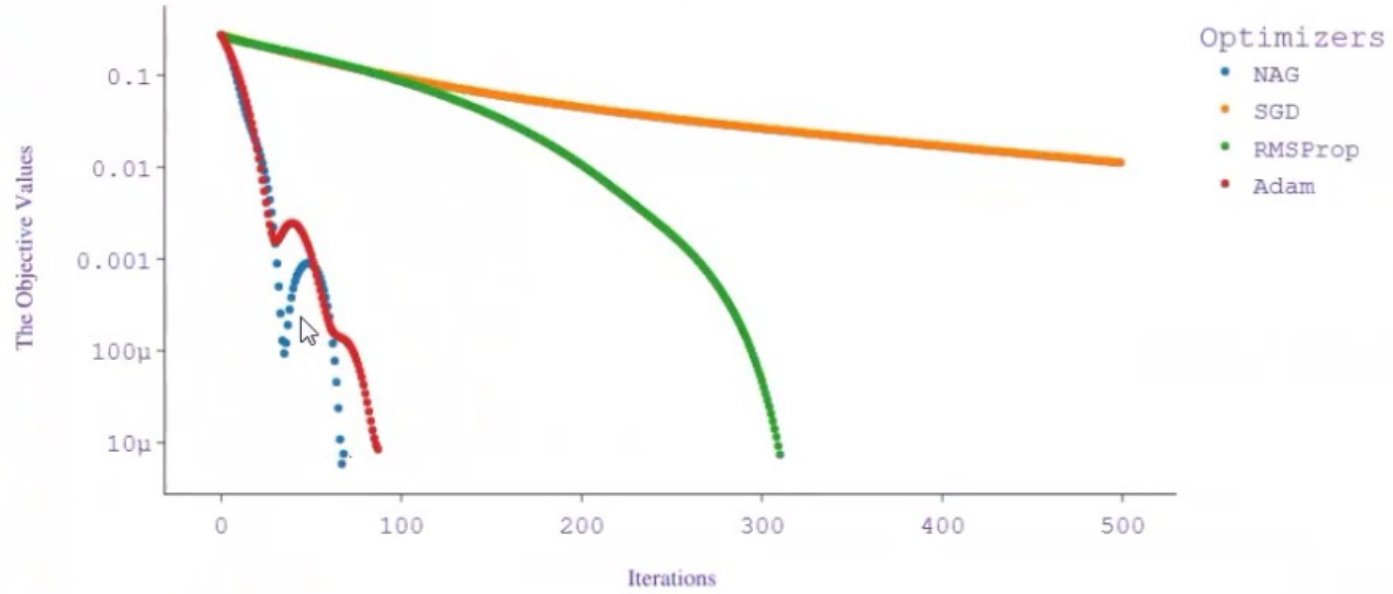
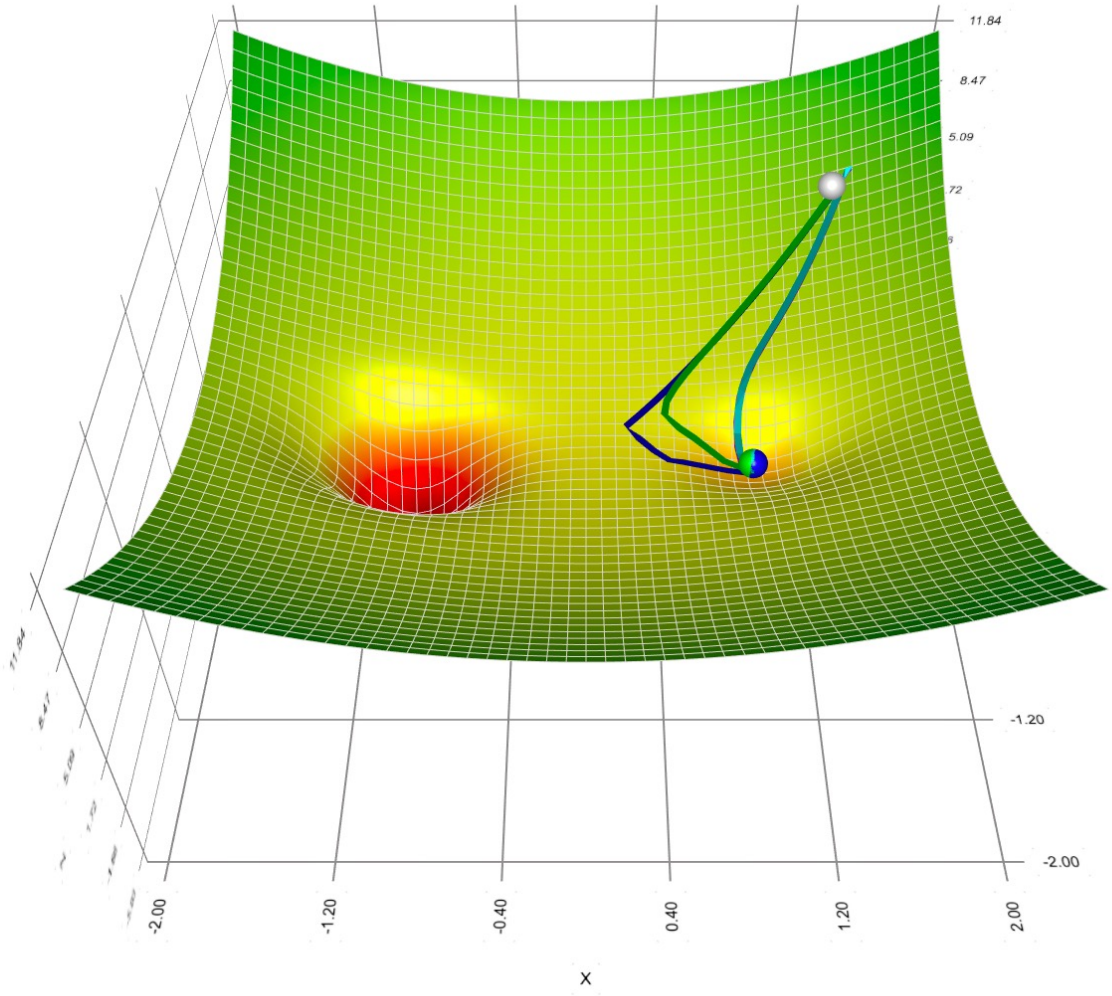
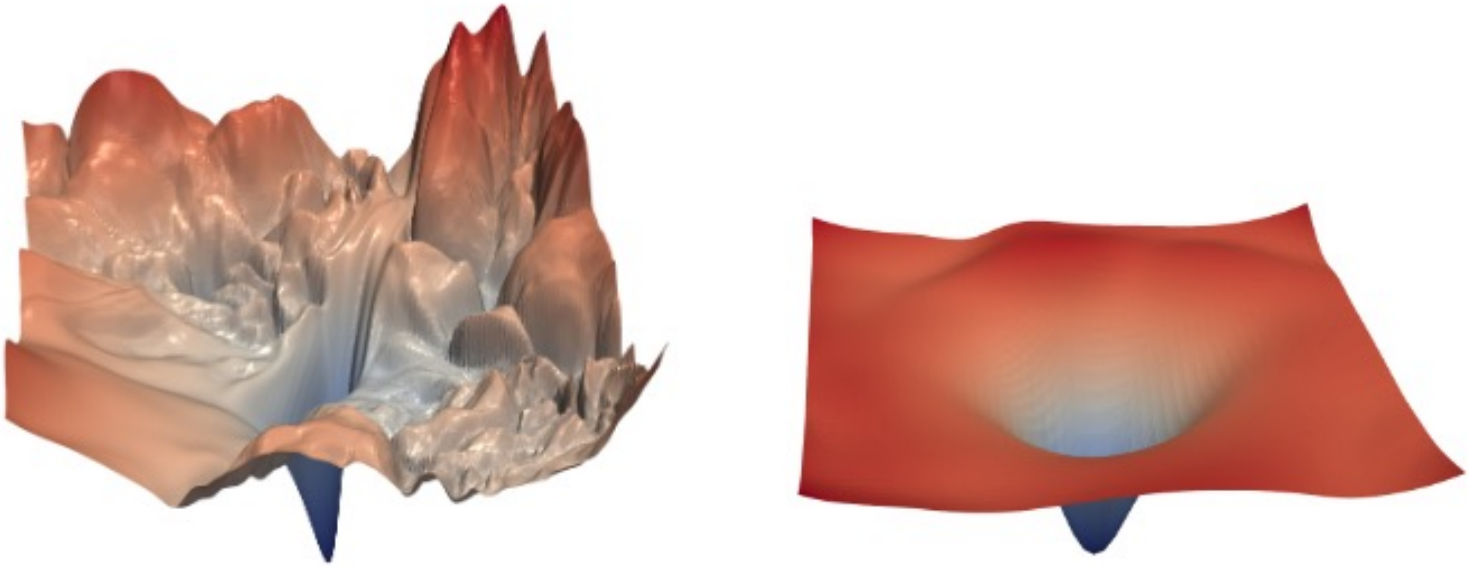


Figure: the progress of the objective values with all Optimizers in 500 iterations

https://github.com/lilipads/gradient_descent_viz



Visualizing the Loss Landscape of Neural Nets
(Hao Li et al. 2018)



(a) without skip connections

(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

https://papers.nips.cc/paper_files/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf

News about ADAM from ICLR 2025

- Muon optimizer [2024]
- Adam-mini [ICLR 2025]
  - share momentum across weights, require less space, have larger batches, converge faster
- ADAM
  - worst-case divergence [Reddi et al 2018]
  - Adagrad with convergence proof [Defossez 2022]

3 Regularization in DNNs: Lasso, Ridge and Constraints

Regularization

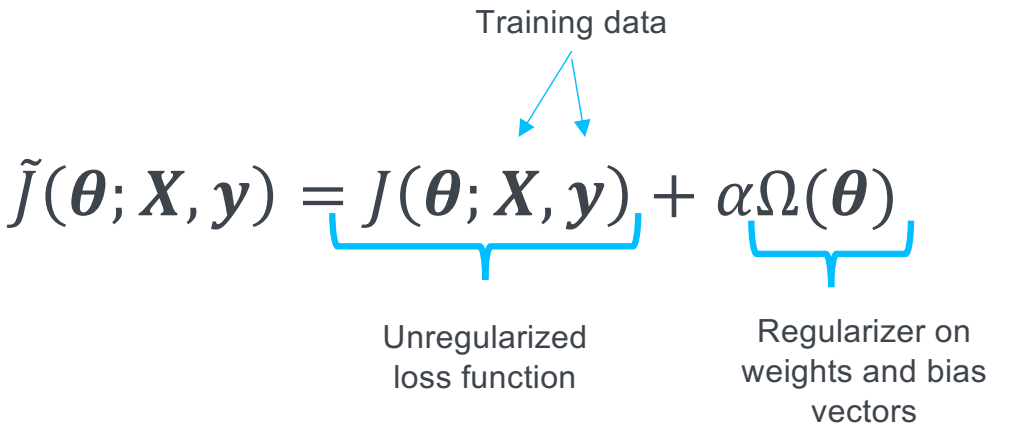
- Goodfellow, Bengio, Courville: „*any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error*.”

- Wikipedia: ***regularization*** *is the process of adding information in order to solve an ill-posed problem or to prevent overfitting*.

In extreme cases, we talk about billions of parameters in DNNs.
Hence, overfitting is a huge issue and the need for regularization is paramount

Parameter Norm Penalty

Regularized loss function:



Convention:
$\boldsymbol{w}$ denotes only the weights used in regularization
$\boldsymbol{\theta}$ denotes all the parameters

L^2 Parameter Regularization

Bias vectors are usually not regularized

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) + \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}$$

Regularized Gradient

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\boldsymbol{w}$$

Regularized Gradient Step shrinks weight vectors

$$\mathrm{w} \leftarrow w - \varepsilon\left(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})\right)$$

L_2 and L_1 regularization by weight decay

cf 4-Regression

- Weight matrices are regularized,
  biases remain unregularized (to avoid underfitting)
- Add penalty $\lambda W_{l,ij}^2$ (Ridge) or $\lambda |W_{l,ij}|$ (Lasso) for every weight
- In practice, compute the unregularized gradient as usual, then add $\lambda W_{l,ij}$ for L_2 or $\lambda \operatorname{sign}(W_{l,ij})$ for L_1 to the gradient

L_1 makes the weights become sparser than L_2

Column-wise regularization

Ridge regression minimizes/constrains using the **Froebenius** norm

$$W_l = \begin{pmatrix} w_{l,11} & \cdots & w_{l,1m} \\ \vdots & \ddots & \vdots \\ w_{l,n1} & \cdots & w_{l,nm} \end{pmatrix}$$

$$\|W_l\|_2 = \sum_i \sum_j \left|w_{l,ij}\right|^2$$

We can also constrain each of the columns such that no unit is much preferred

- using constraints and Lagrange multipliers (cf lecture on SVMs) this can be added to the cost function
- no details are given here

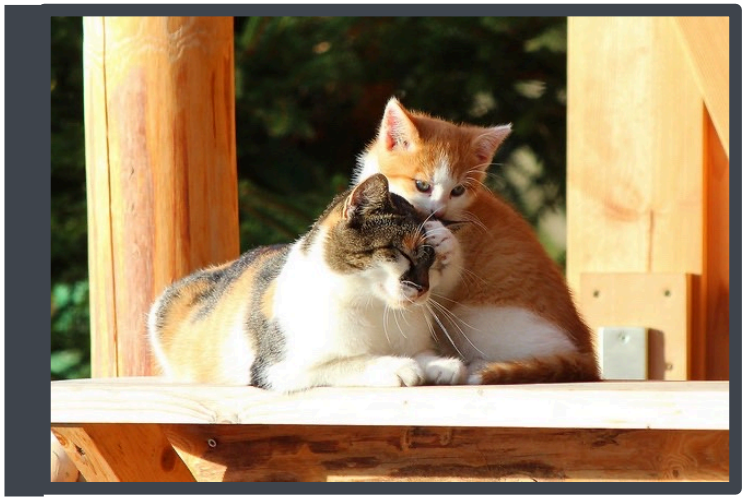
4 Regularization in DNNs: Data Augmentation

Quote of the day

On how to win Kaggle competitions:
https://www.youtube.com/watch?v=NCGkBseUSdM

Regularization through generated data

Picture of cats



Generated data
Picture of cats shifted by a few pixels

https://flic.kr/p/pyP8vQ

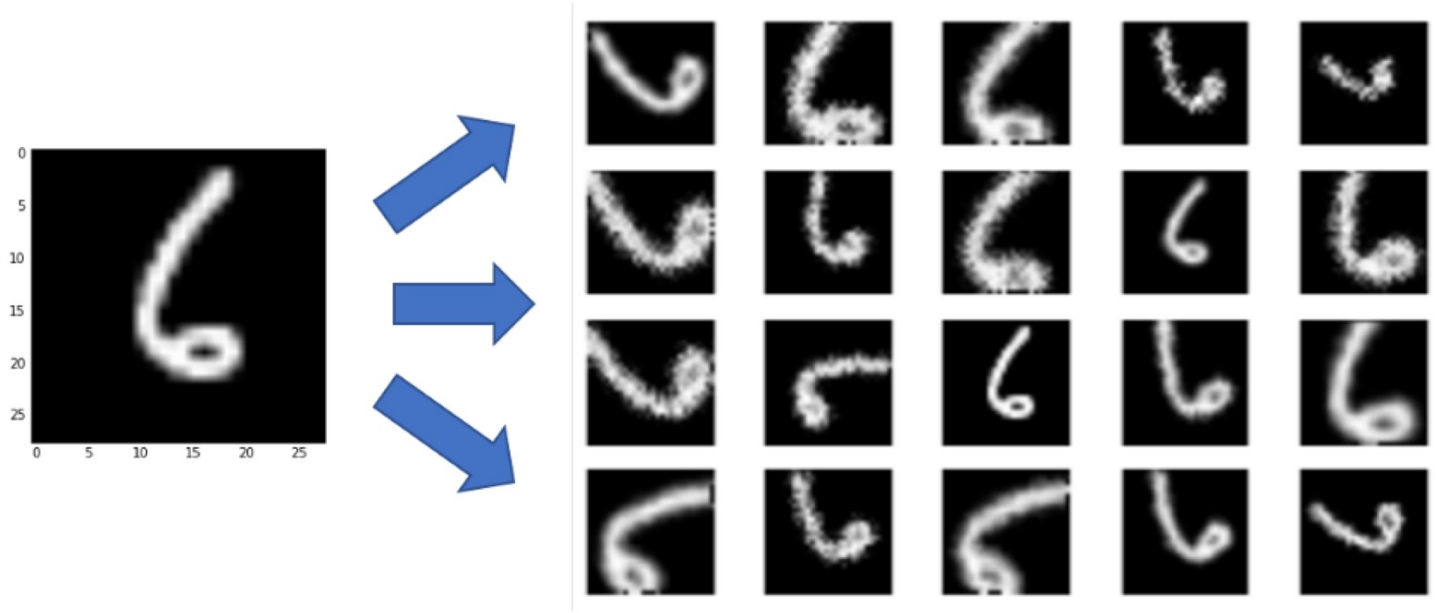
More data is better. Fake data may be useful!

Picture of cats

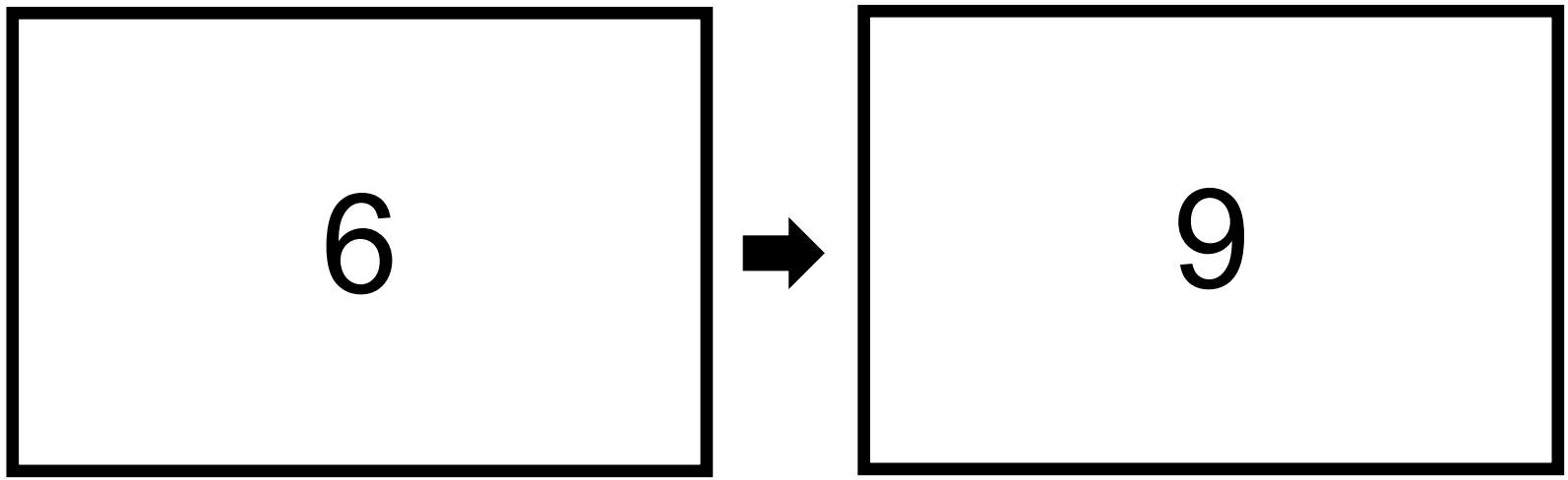
Generated data
Picture of cats
flipped
horizontally

https://flic.kr/p/pyP8vQ

Data Augmentation: Injecting noise at the input



Be careful about data augmentation



6

AUGMENTED DATA
Rotated 6

Label smoothing: Injecting noise at the output targets

- Most datasets have some number of mistakes in the y labels.
- It can be harmful to maximize $\log P(y|x)$ when y is a mistake
- Softmax: tries to learn 0 or 1 for a category
  - Problem:
    - maximal value may be hard or impossible to reach
      AND
    - the value may be wrong, because of noise
  - Solution:
    - Model the noise at the label $(\frac{\varepsilon}{k-1}, \ldots, \frac{\varepsilon}{k-1}, 1-\varepsilon, \frac{\varepsilon}{k-1}, \ldots \frac{\varepsilon}{k-1})$

very easy to implement

can be combined with other regularizations

avoid ever growing weights

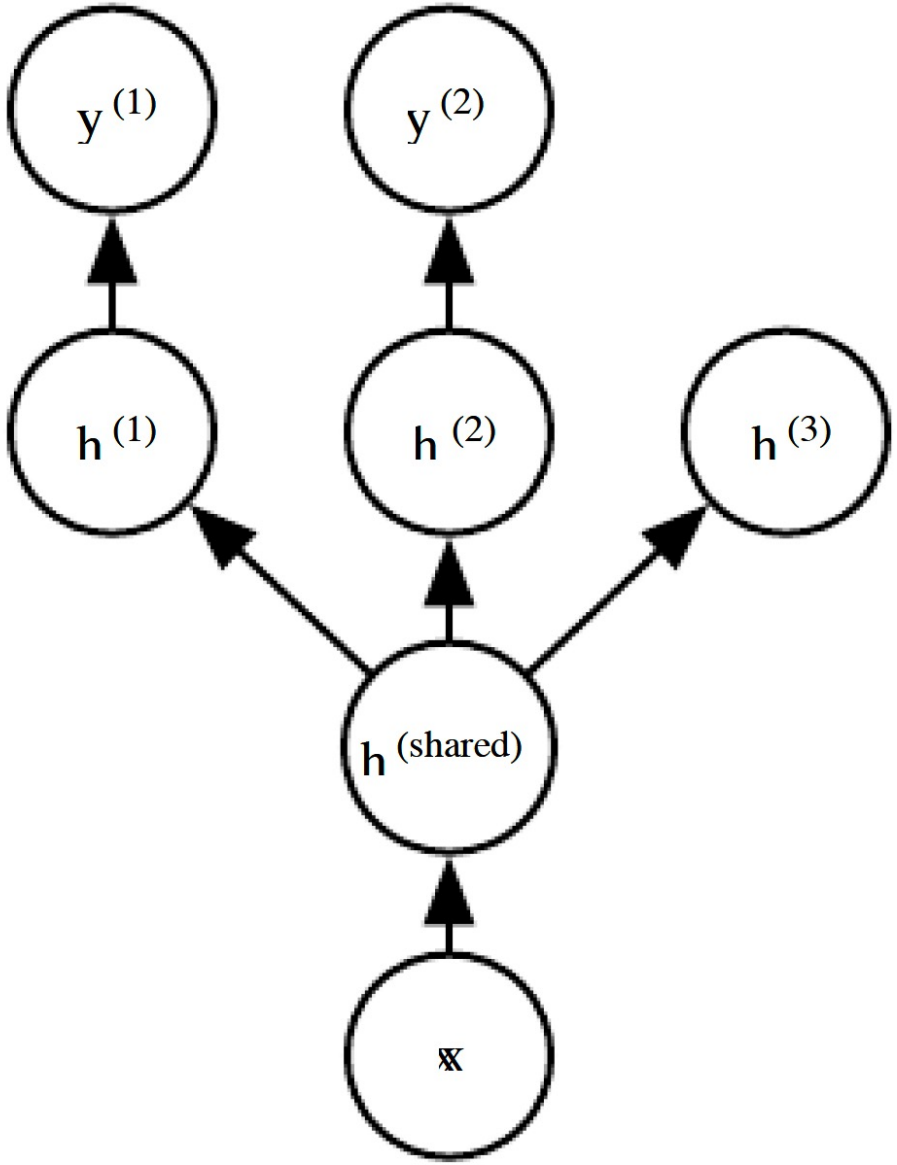
Injecting noise into the weights

- Wiggle all the weights with some noise ε_W
- Propagate with wiggled weights to compute $\hat{y}_{\varepsilon_W}$
- Compute corresponding loss $L(\hat{y}_{\varepsilon_W}, y)$ and gradient
  - Can also be put into the regularization term

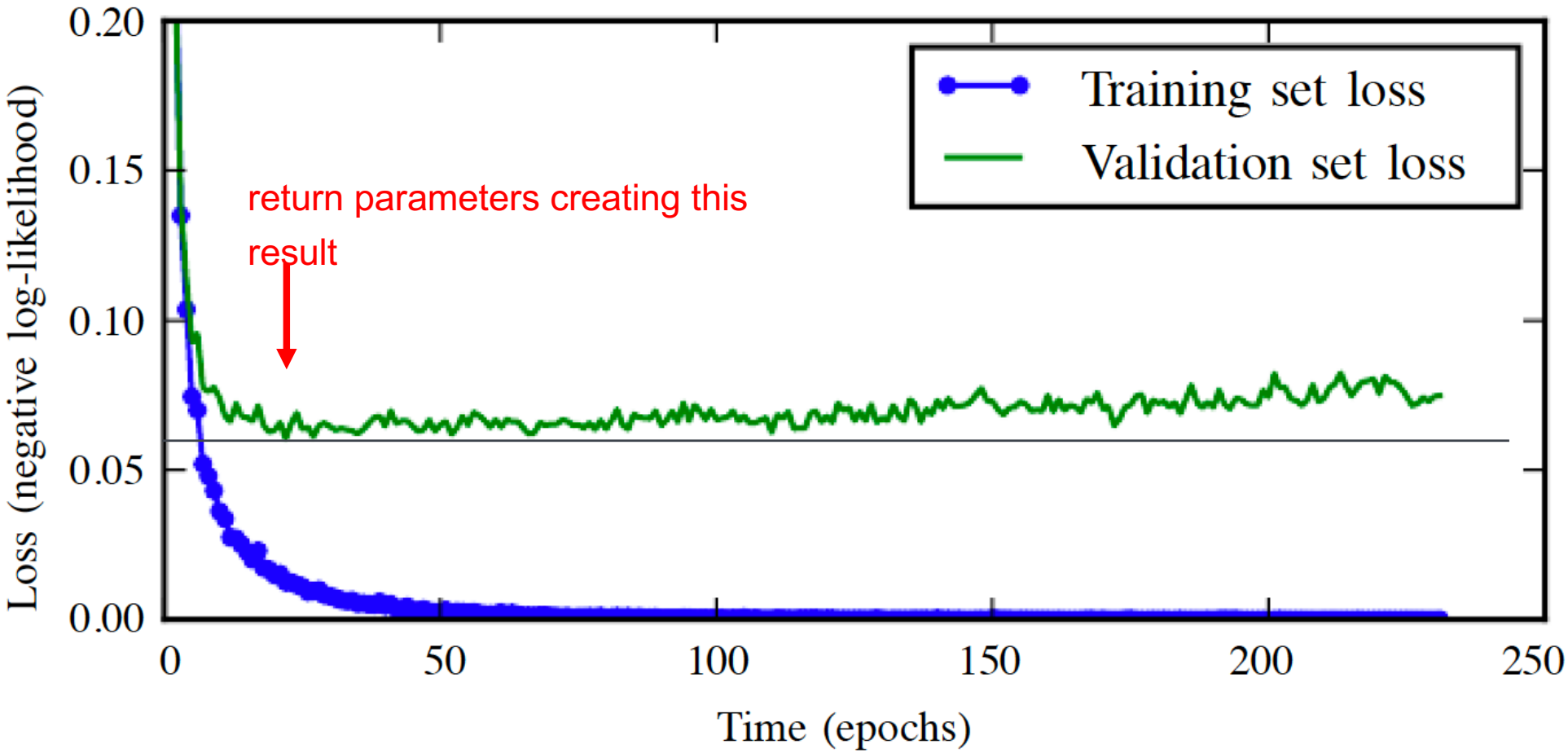
Aims to avoid at getting stuck in small local minima

Multi-task learning

- For one data point x
- several outputs $y^{(i)}$ representing different tasks

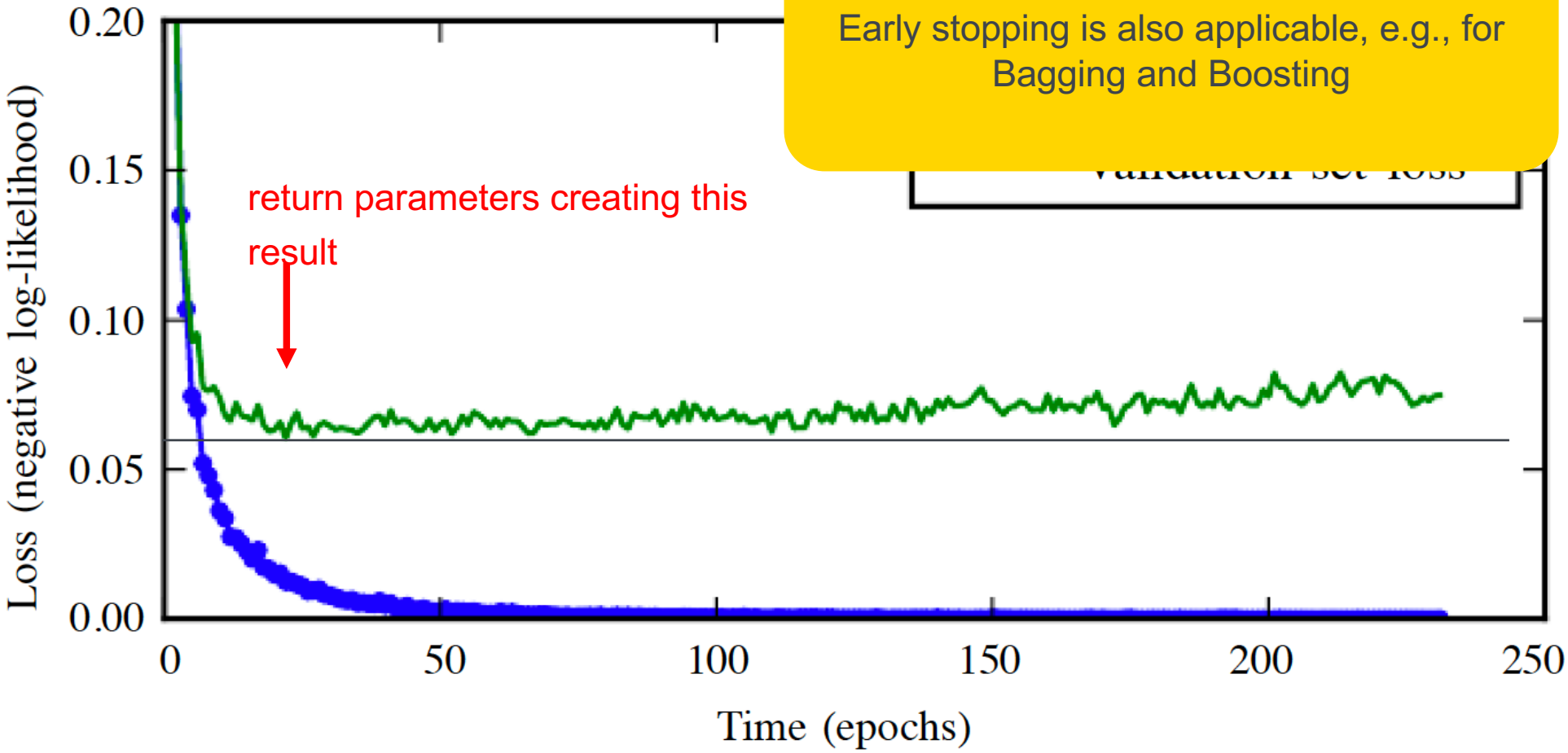


Early stopping



Effect of early stopping is related to weight decay
– at an optimal point of the parameter space (cf Goodfellow et al, Chapter 7.8)

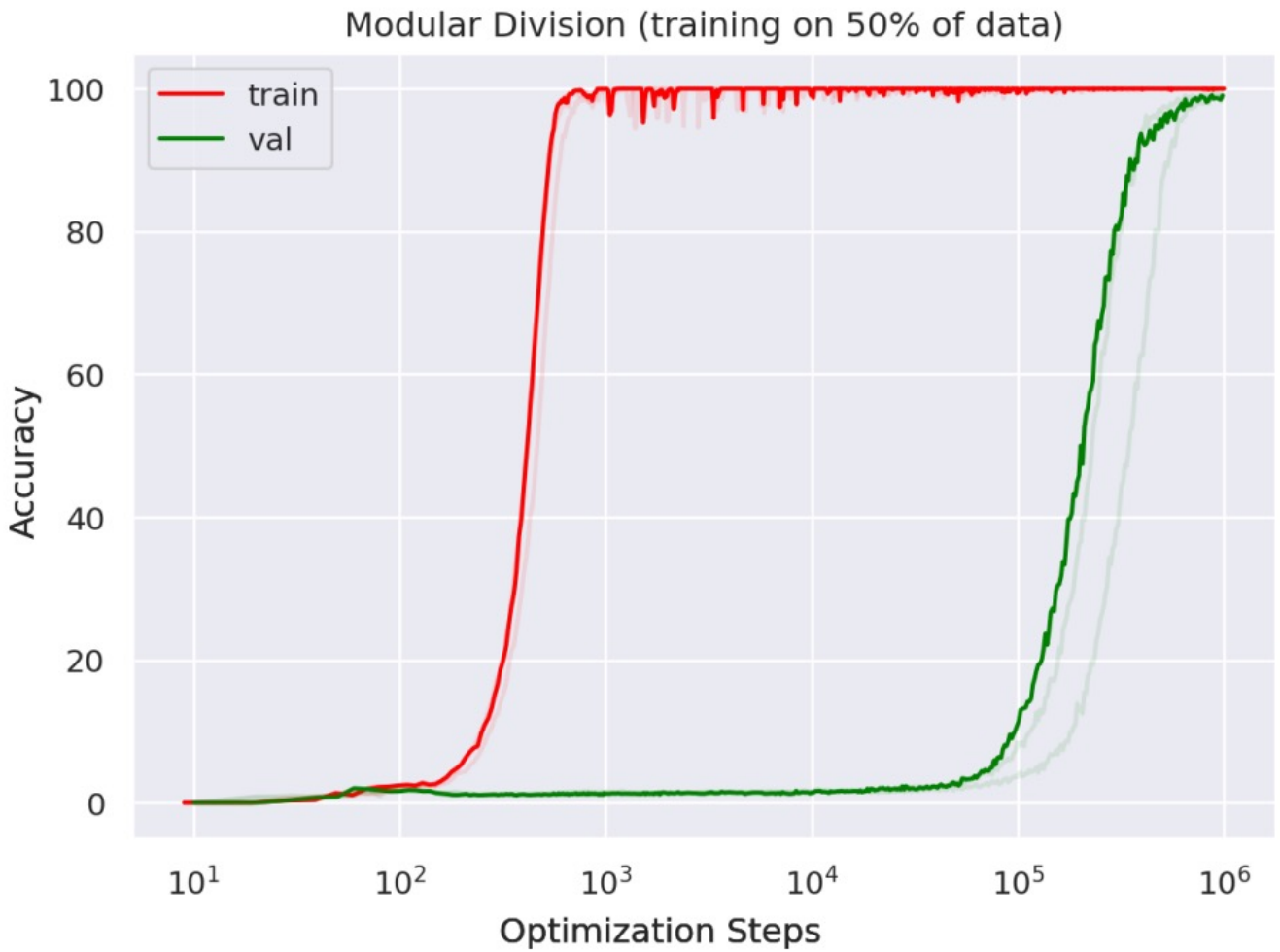
Early stopping



Early stopping is also applicable, e.g., for Bagging and Boosting

Effect of early stopping is related to weight decay
– at an optimal point of the parameter space (cf Goodfellow et al, Chapter 7.8)

Grokking?



2022 published on arxiv

Not yet peer reviewed: https://arxiv.org/pdf/2201.02177.pdf

Norm penalty regularization of representations

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h})$$

hidden units, instead of weights

$$\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad \boldsymbol{A} \in \mathbb{R}^{m \times n}$ $\quad \boldsymbol{x} \in \mathbb{R}^n$

sparse weight matrix

many ways to make representation sparse

$$\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad \boldsymbol{B} \in \mathbb{R}^{m \times n}$ $\quad \boldsymbol{h} \in \mathbb{R}^n$

sparse representation

Adversarial Examples

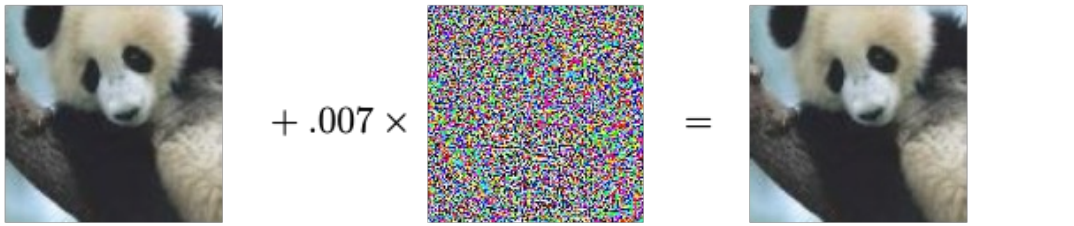


Figure 7.8

Training on adversarial examples is mostly intended to improve security, but can sometimes provide generic regularization.

5 Regularization in DNNs: Dropout

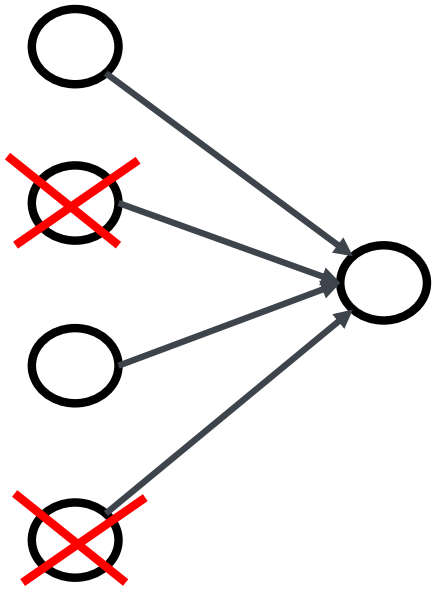
Dropout – The Intuition

Highly popular

What Dropout does

per learning cycle

- stochastically eliminate units with probability p_l^{drop} in layer l

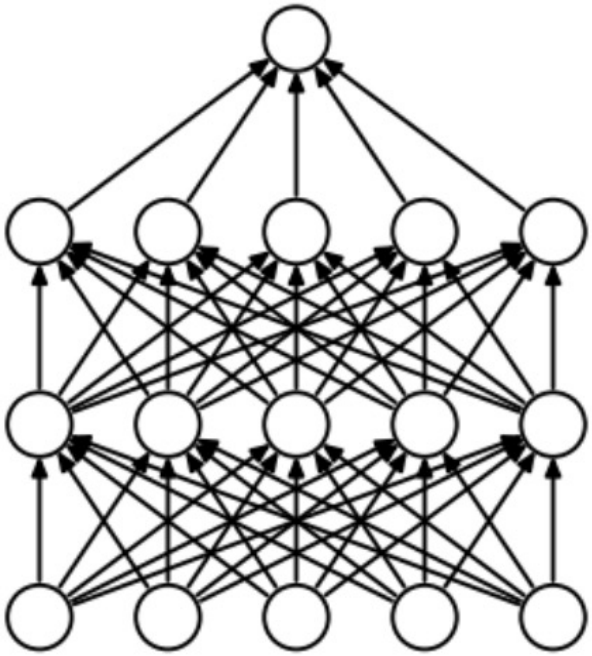


Intuition

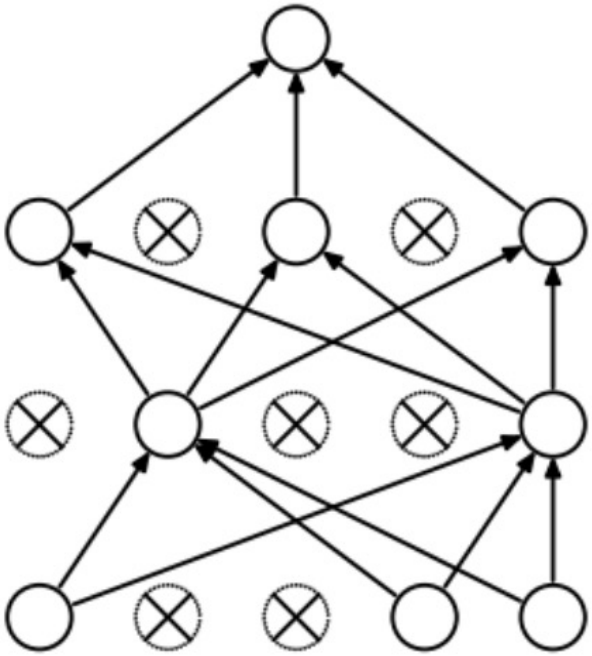
- Can‘t rely on any single feature
  - spread out the weights
- Limit reliance of one unit on a specific predecessor
  - avoid overfitting – „one path per training example“
- Dropping a predecessor shrinks its influence, its weights
  - regularization effect

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

Dropout



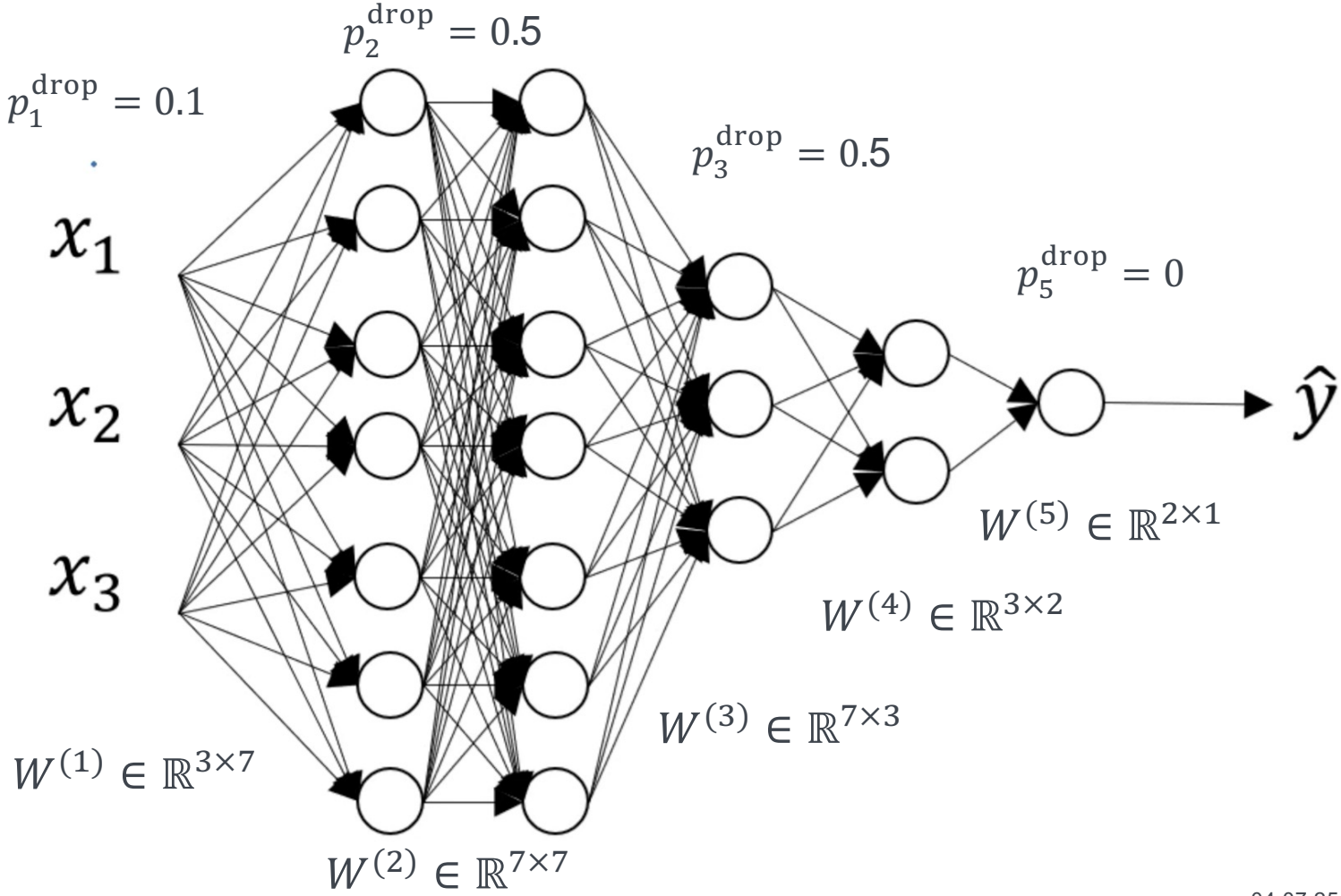
(a) Standard Neural Net



(b) After applying dropout.

Image: https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab

Dropout (example numbers)

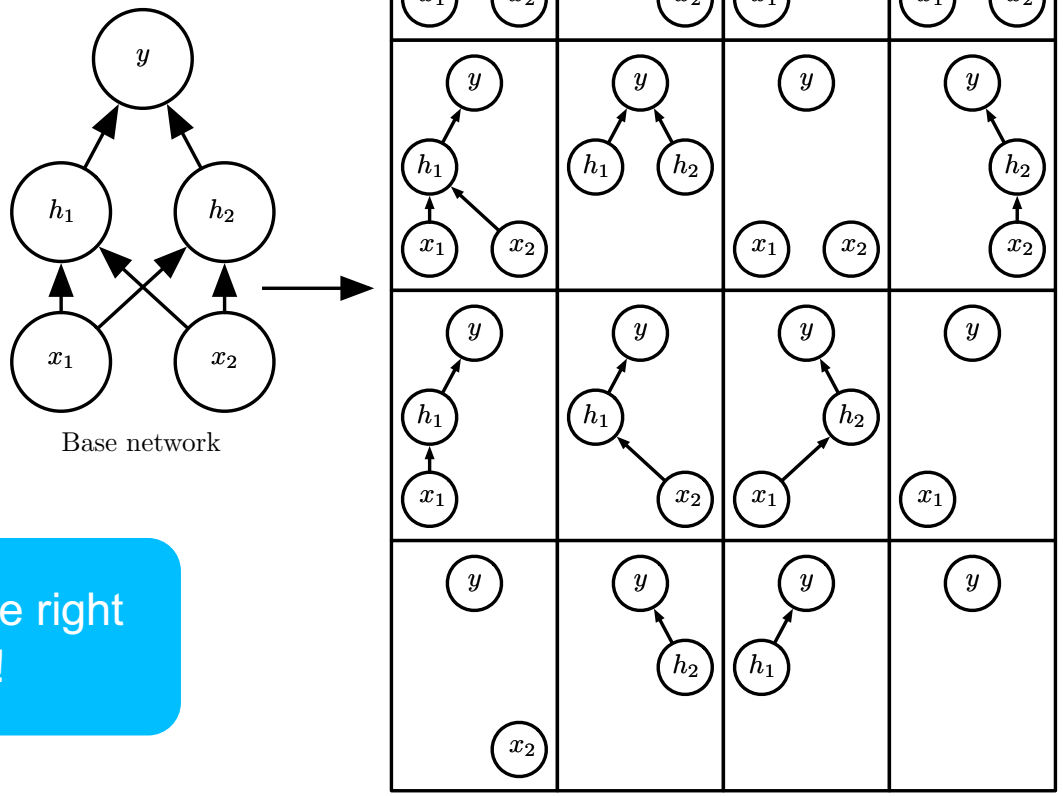


Dropout – The theory: Bagged Ensembles Training

- Actually, there is not “one theory“ on dropout, but there are various, non-contradicting views on what dropout is
  - approximation of L2 regularization in linear regression
  - Bayes‘ian approximation (variational inference)
  - **Model averaging** – which we present here (Goodfellow et al)

Baldi, P., & Sadowski, P. J. (2013). Understanding dropout. In *Advances in neural information processing systems* (pp. 2814-2822).

Dropout as Ensemble Training

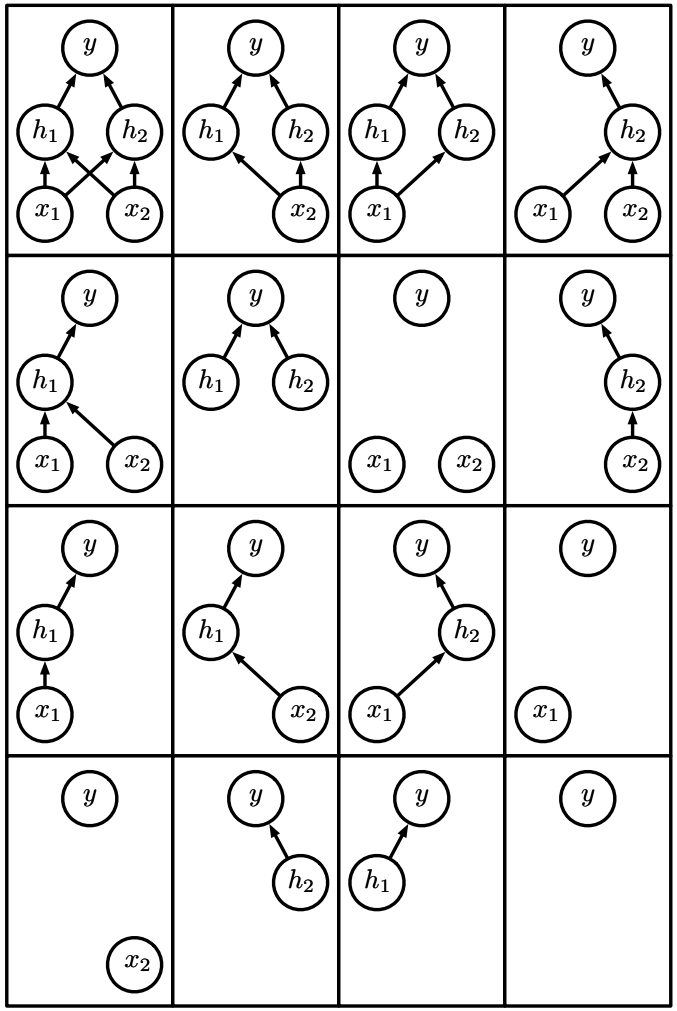


Base network

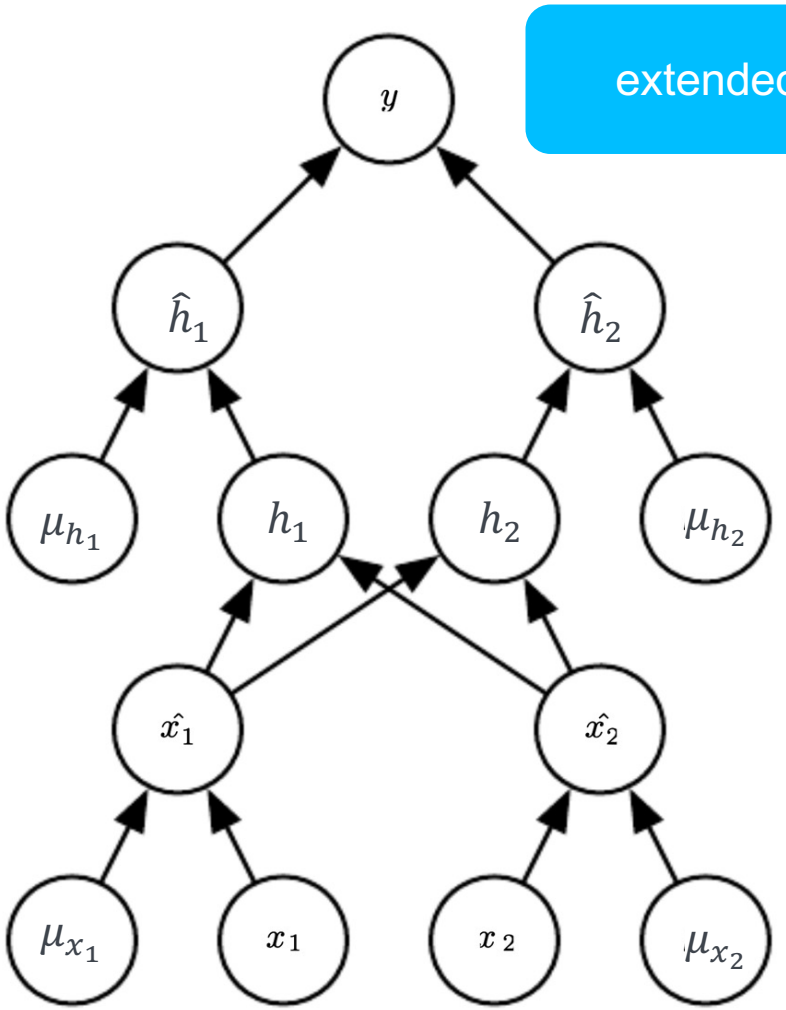
Ensemble of subnetworks

Not all networks on the right would be viable!

Modeling the influence of mask vector μ



Ensemble of subnetworks



extended model

Dropout Training

- Given mask vector μ
  specifying which units to include
- Then dropout training is:

$$\operatorname*{argmin}_{\theta} \mathbb{E}_{\mu}(J(\theta, \mu))$$

- Forward Propagation:
  - Exponentially (in the number of units) many μ
- Practically:
  - sample from possible values of μ
  - or: use approximation (next slide)

Weight Scaling Inference Rule

- Idea: approximate $P(y|x)$
  - by evaluation of the extended model (cf. 2 slides earlier)
  - by using geometric mean over all vector masks (instead of arithmetic mean)

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

Consider a single logistic unit

- Output $O = \sigma(S) = \frac{1}{1+ce^{-\lambda S}}$, where
  - $S = \sum_{j=1}^{d} w_j I_j$
  - I_j is the j-th input (which is j-th output from previous layer)
- Arithmetic Mean $E = \sum P_i O_i$
  - P_i is the probability of a specific vector mask $\mu^{(i)}$ and
  - O_i is the output produced from using this specific vector mask $\mu^{(i)}$
- Weighted Geometric Mean $G = \prod_i O_i^{P_i}$
- Weighted Geometric Mean of the Complements: $G' = \prod_i (1 - O_i)^{P_i}$
- Normalized Weighted Geometric Mean: $NWGM = \frac{G}{G+G'}$

Baldi, P., & Sadowski, P. J. (2013). Understanding dropout. In *Advances in neural information processing systems* (pp. 2814-2822).

Consider a single logistic unit

- Key averaging theorem (see paper):

$$NWGM(O_1 \dots O_m) = \sigma(\mathbb{E}(S))$$

- Therefore Forward propagation becomes simple

$$NWGM = \sigma(\sum_{i=1}^{d} w_j p_j I_j)$$

where p_j indicates the probability that unit/input I_j is kept ($p_j = 1 - p_j^{drop}$)

Baldi, P., & Sadowski, P. J. (2013). Understanding dropout. In *Advances in neural information processing systems* (pp. 2814-2822).

Generalizing Single Unit to Deep Neural Network

- Layers: h, l
- Units: i
- Output of unit i in layer h: O_i^h

more details in the paper

- Leads to approximation for forward propagation

$$\mathbb{E}(O_i^h) \approx \sigma_i^h \left[\sum_{l<h} \sum_j w_{i,j}^{hl} p_j^l \, \mathbb{E}(O_j^l) \right]$$

paper formalizes $l < h$, in our context $l = h - 1$

I found the original paper more understandable than the deep learning book in this matter:

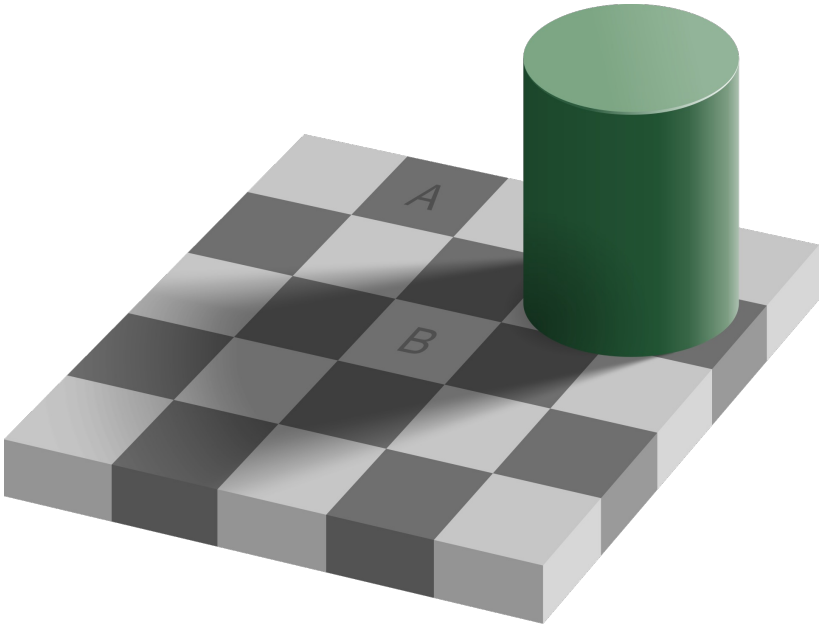
Baldi, P., & Sadowski, P. J. (2013). Understanding dropout.
In *Advances in neural information processing systems* (pp. 2814-2822).

Dropout

- most favored regularization method for many DNN tasks
  - still: not always useful!
- Dropout is also useful for machine learning methods:
  - muting inputs: logistic regression
  - muting trees:
    - e.g. DART (gradient boosting)
      http://proceedings.mlr.press/v38/korlakaivinayak15.pdf

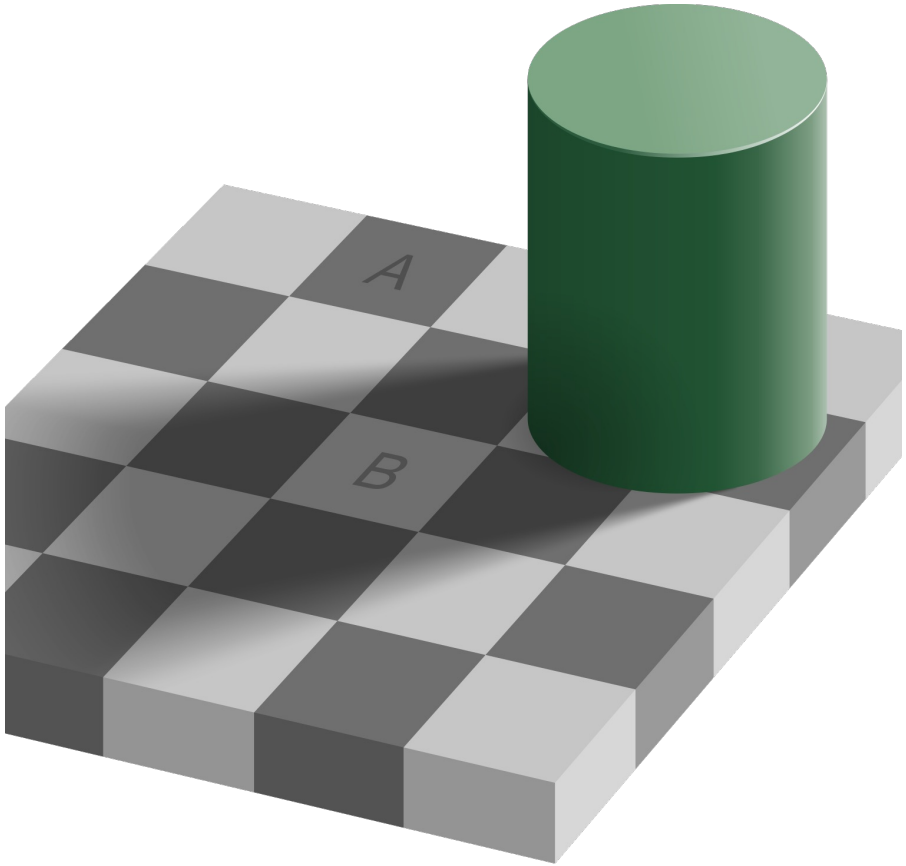
6 Convolutional Neural Networks

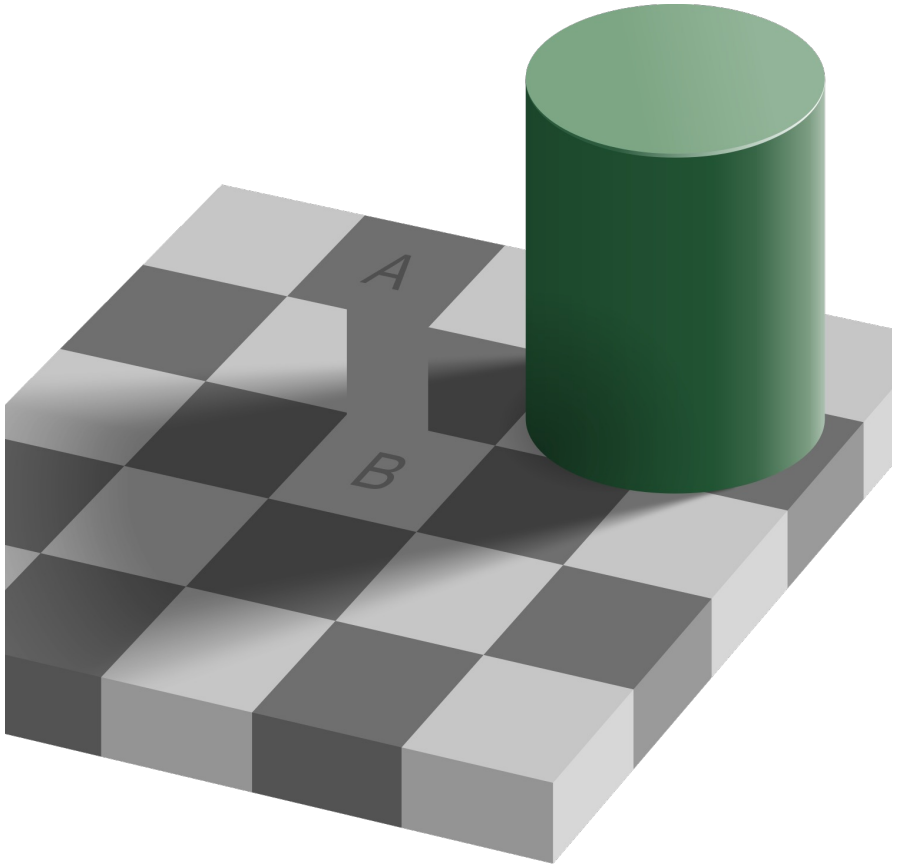
Checker shadow illusion



Checker shadow illusion

LOCALITY, EXPERIENCE, ETC.





So far: Fully connected networks

- Many parameters
- Fully connected feed-forward networks
  not suitable for every task
  - for images:
    - why should an object (e.g. Panda bear) be treated differently
      if it appears on the left side of a picture rather than on the right?

Idea for convolutional neural networks:
- **local vs global processing of image features**
- **learning of feature weights**
- **parameter sharing (feature weights)**

Convolution by Linear Filter

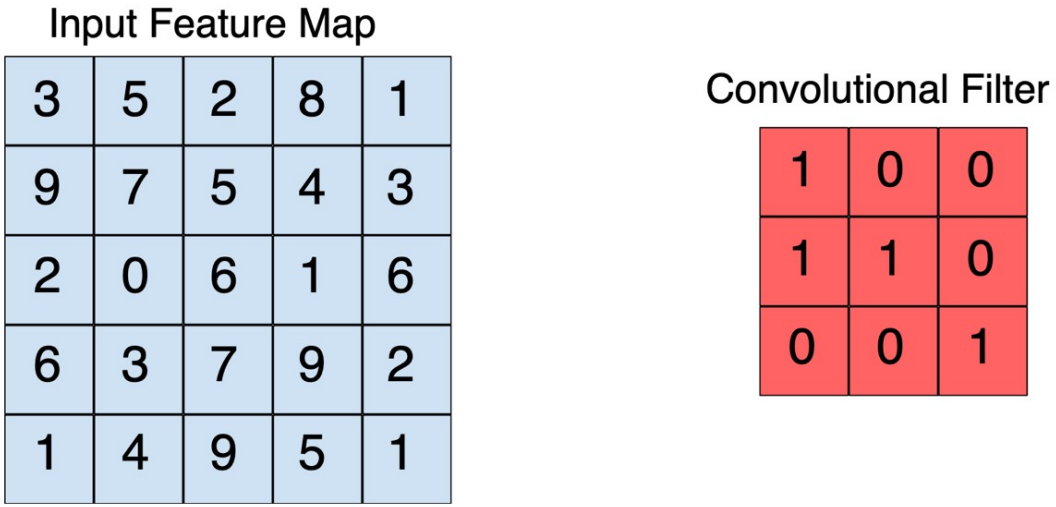


Figure 4a. ***Left****: A 5x5 input feature map (depth 1).* ***Right****: a 3x3 convolution (depth 1).*

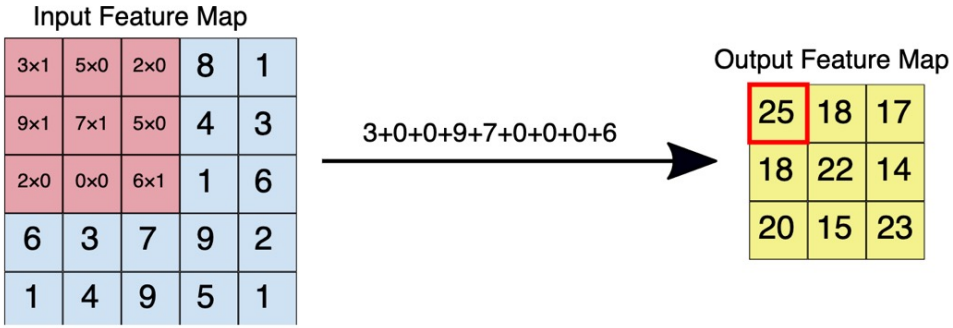
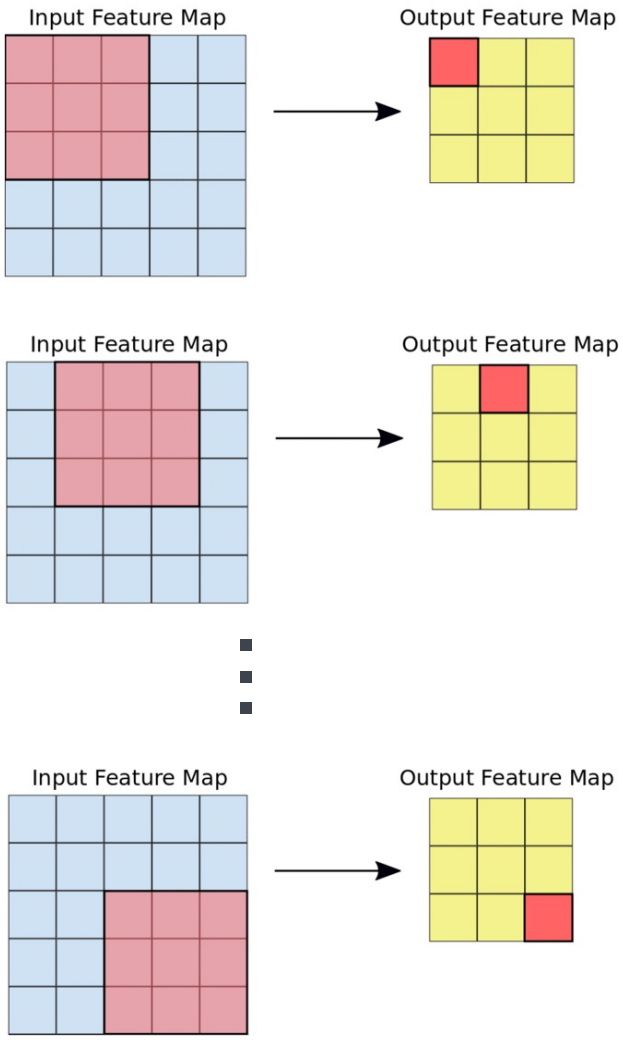


Figure 4b. ***Left****: The 3x3 convolution is performed on the 5x5 input feature map.* ***Right****: the resulting convolved feature. Click on a value in the output feature map to see how it was calculated.*

https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

Moving the Linear Filter (here with stride=1)



See the full animation on

https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

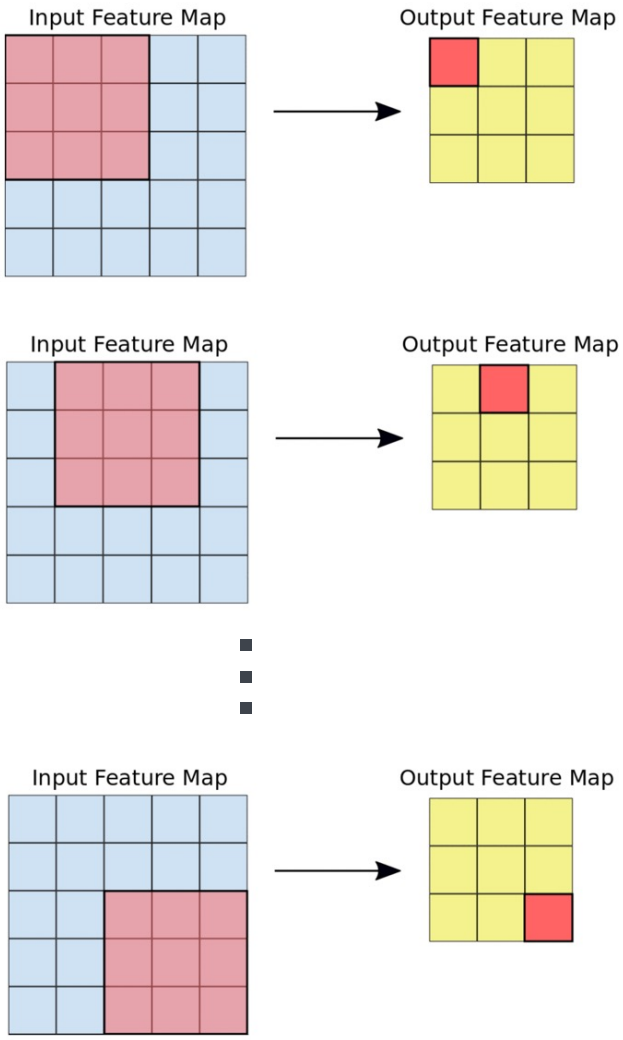
Example

kernel
matrix of
size 5x5

0	0	0	0	0
0	0	0	0	0
0	-1	1	0	0
0	0	0	0	0
0	0	0	0	0

Images copied from
http://theorycenter.cs.uchicago.edu/REU/2014/presentations/sauder-presentation.pdf
though I am pretty sure, I have seen it elsewhere ;)

Moving the Linear Filter (here with stride=1)



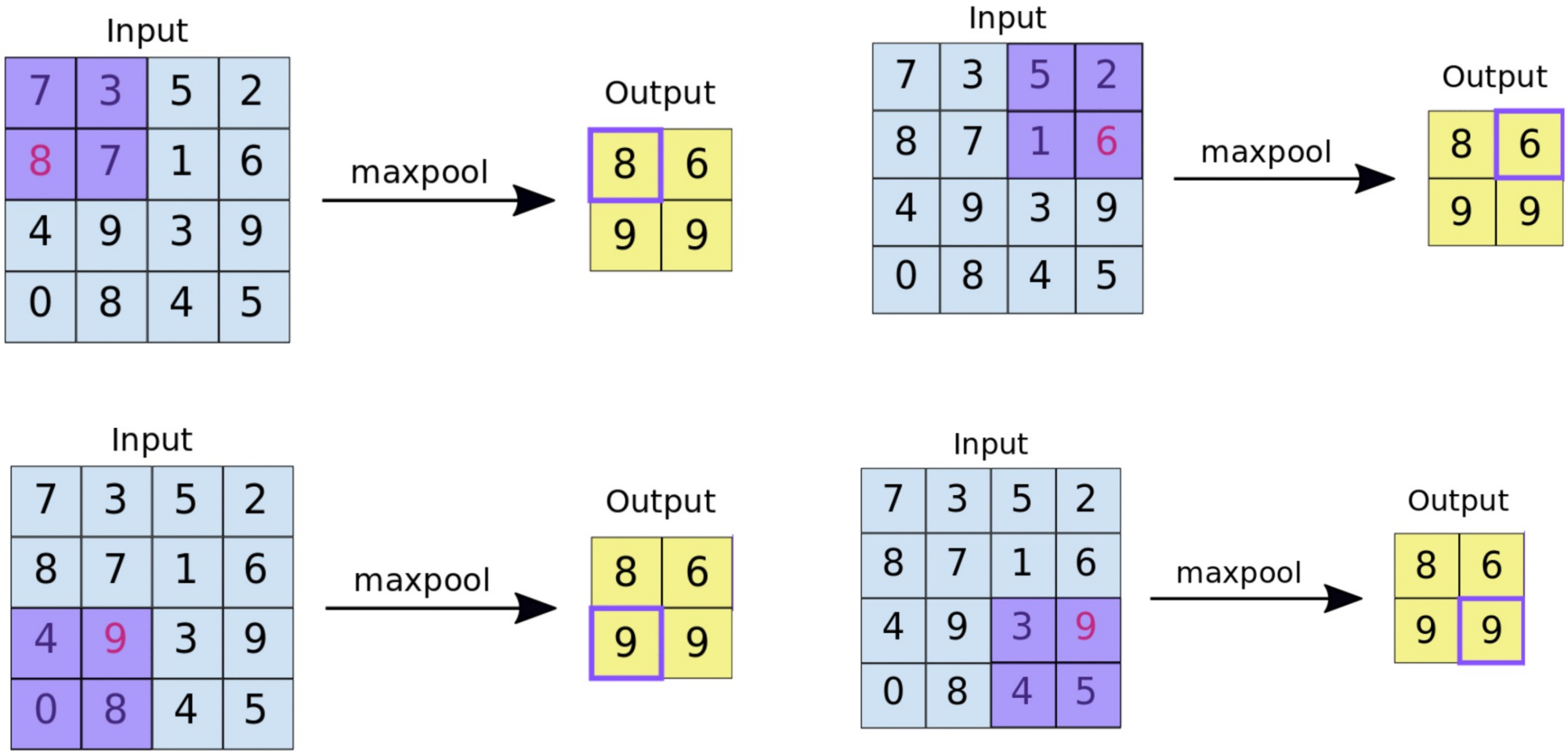
See the full animation on

https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

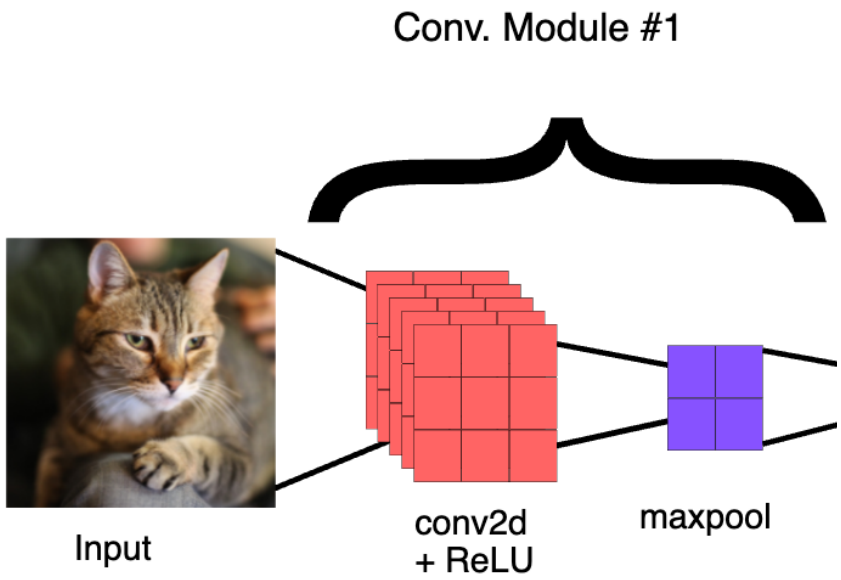
ToDo:

- Learn the filter weights
- Apply the filter with same weights uniformly over the whole input
  - *parameter sharing*

Reduce Dimensionality: Subsampling by Pooling

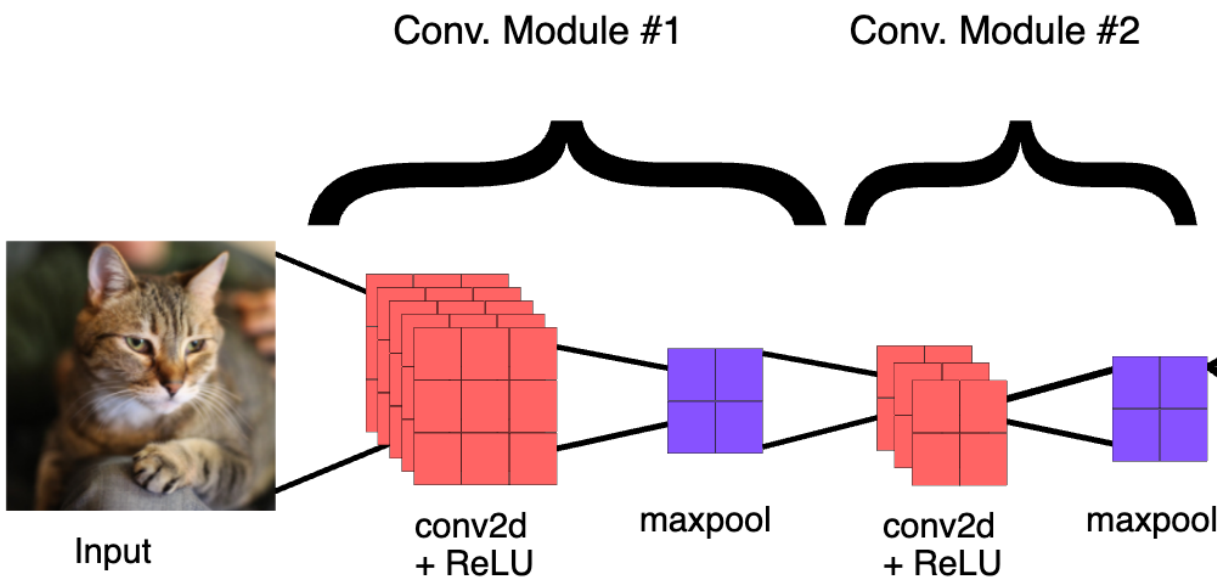


Simple end-to-end convolutional neural network



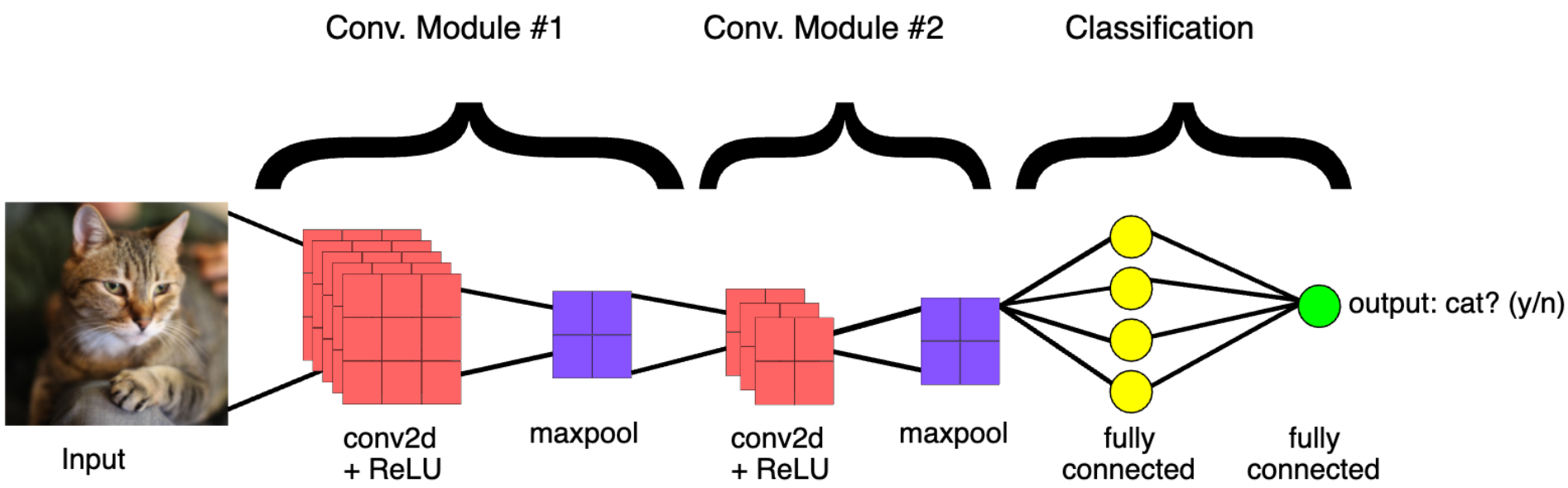
https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

Simple end-to-end convolutional neural network



https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

Simple end-to-end convolutional neural network



https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

The paper that started the deep learning hype: AlexNet

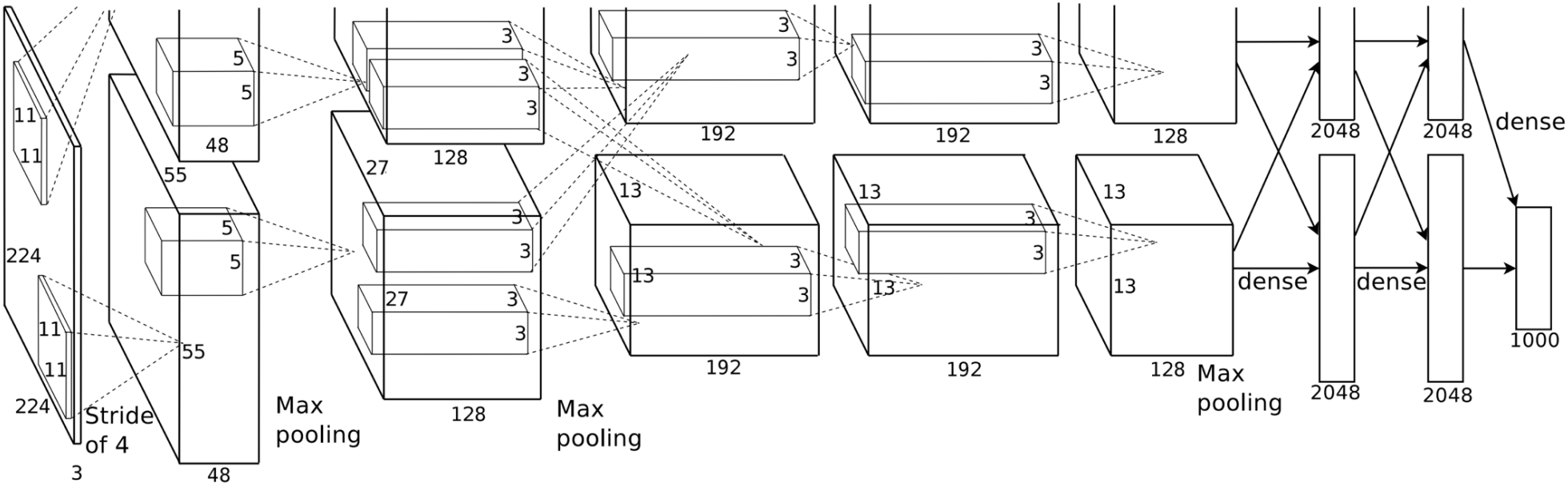
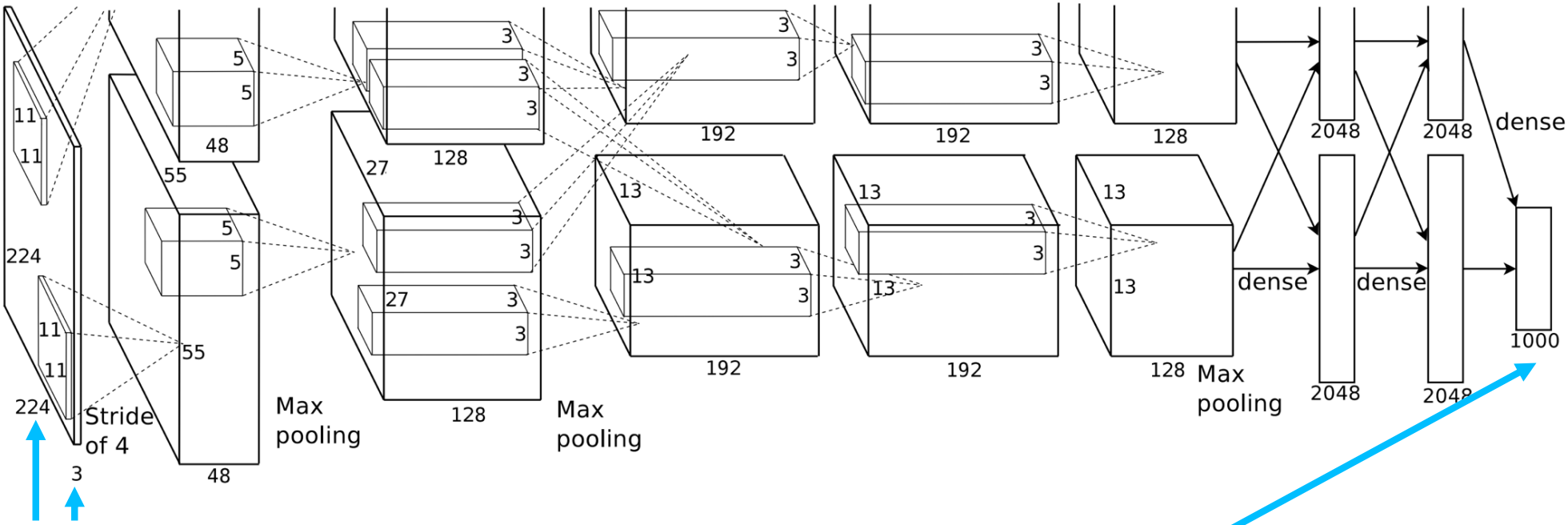


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

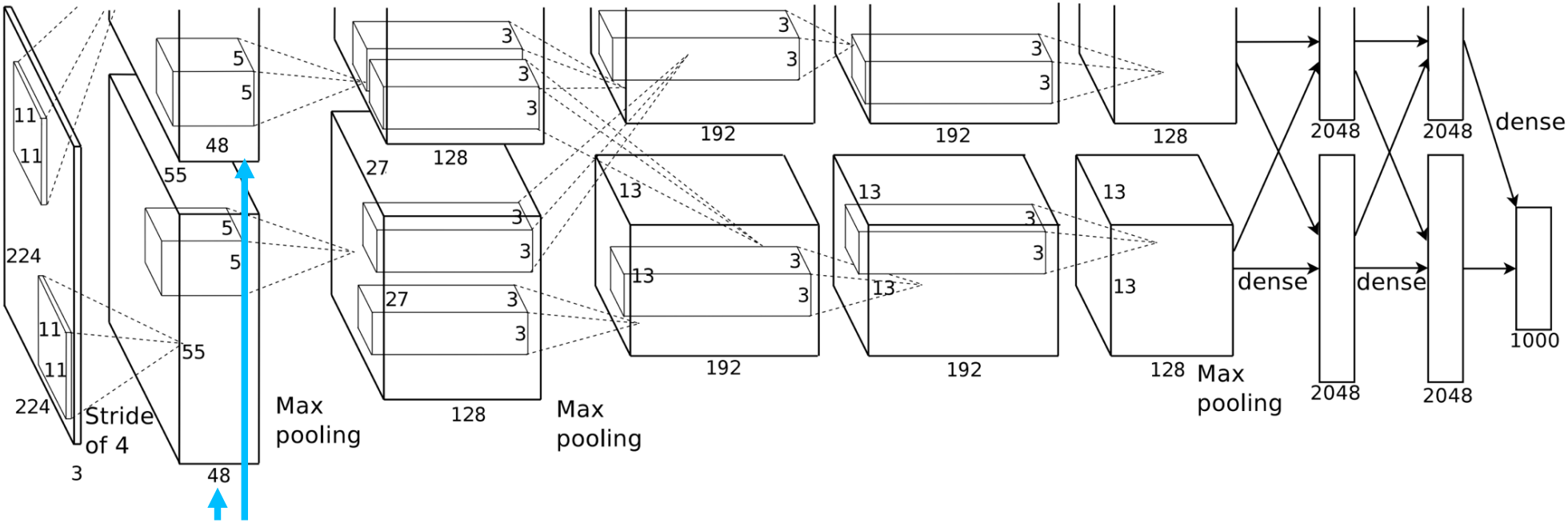
The paper that started the deep learning hype: AlexNet



input image has dimension 224×224×3
output layer: 1000 classes from ILSVRC dataset

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

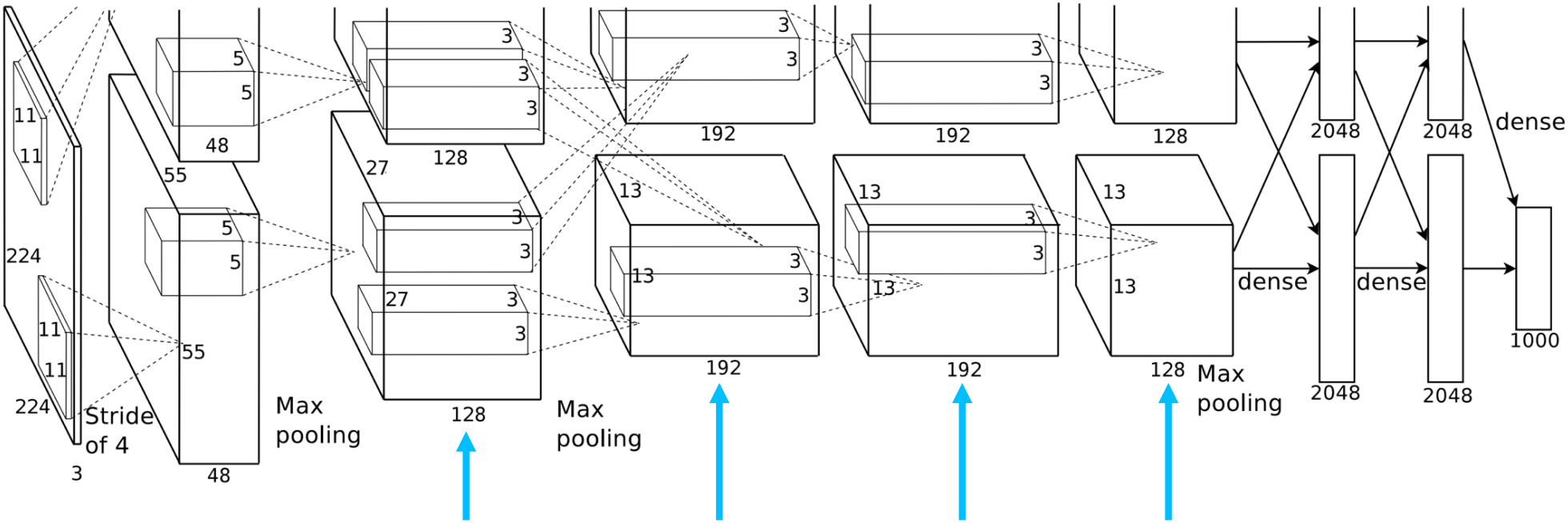
The paper that started the deep learning hype: AlexNet



96 different kernels in the first convolutional layer
(results are worked on in parallel in two GPUs)

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

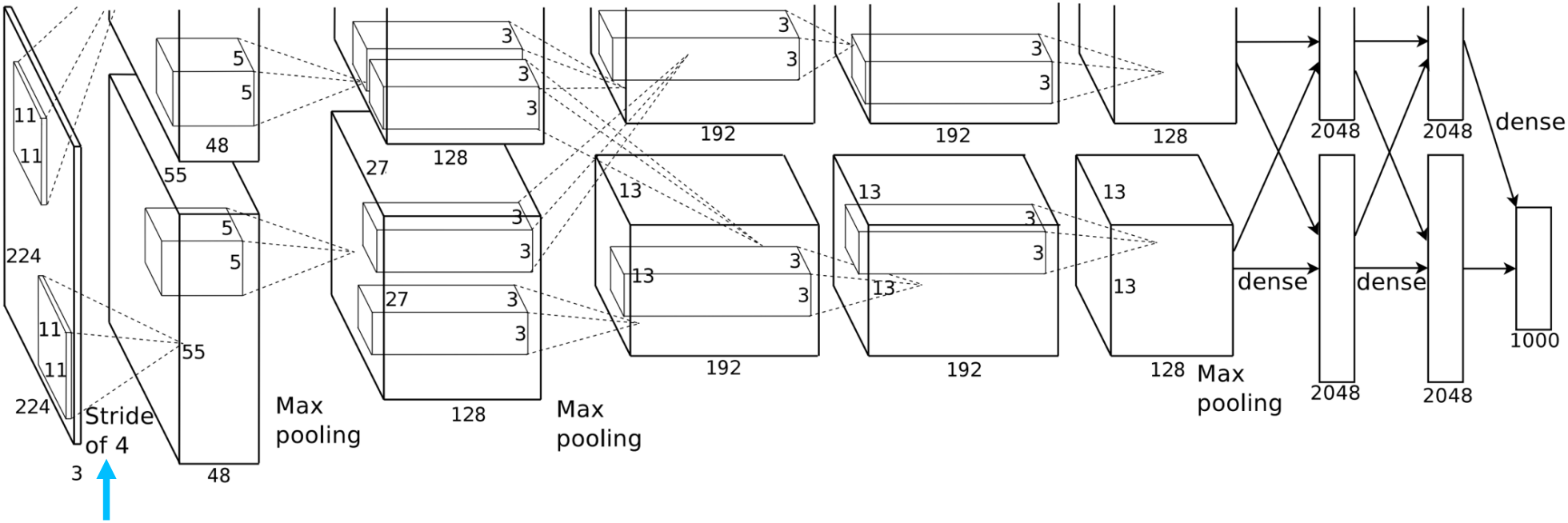
The paper that started the deep learning hype: AlexNet



second, third, fourth, fifth convolutional layer
with 256, 384, 384, 256 kernels, respectively
(results are worked on in parallel in two GPUs)

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

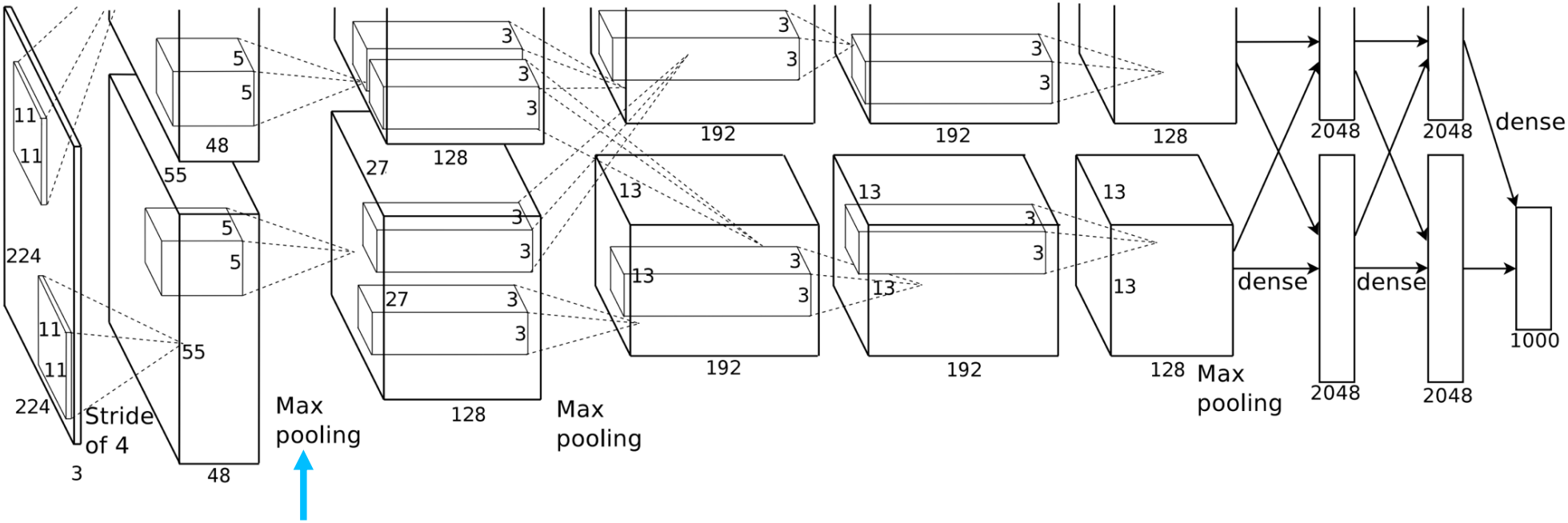
The paper that started the deep learning hype: AlexNet



stride: how much to move a kernel.
If stride is less than kernel size, kernel applications overlap
(as they do here)

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

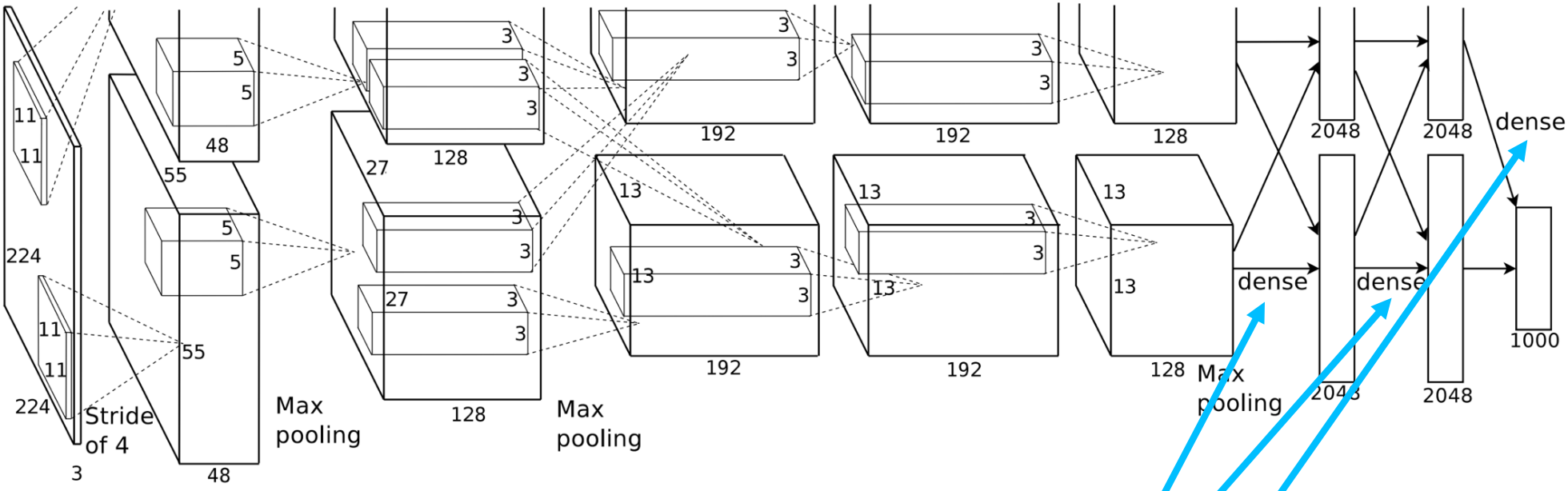
The paper that started the deep learning hype: AlexNet



max pooling: picks maximum of all the entries in the range

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

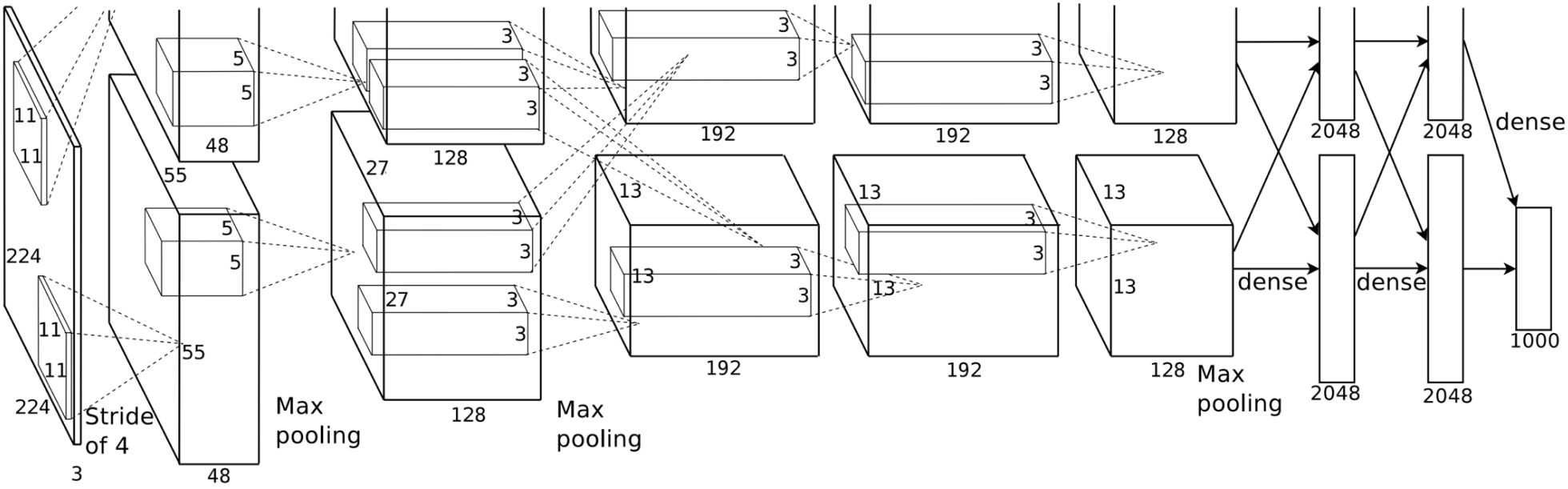
The paper that started the deep learning hype: AlexNet



dense: refers to fully connected layers

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

The paper that started the deep learning hype: AlexNet



regularization: by data augmentation and dropout

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

Most recent Imagenet competitions

Until 2017:

- http://image-net.org/

Since then:

- https://www.kaggle.com/c/imagenet-object-localization-challenge

Overall:

- Convolutional neural networks are better than humans at some metrics wrt image classification, but make other and unexpected errors

7 Recurrent Neural Networks

“Vanilla” Neural Network

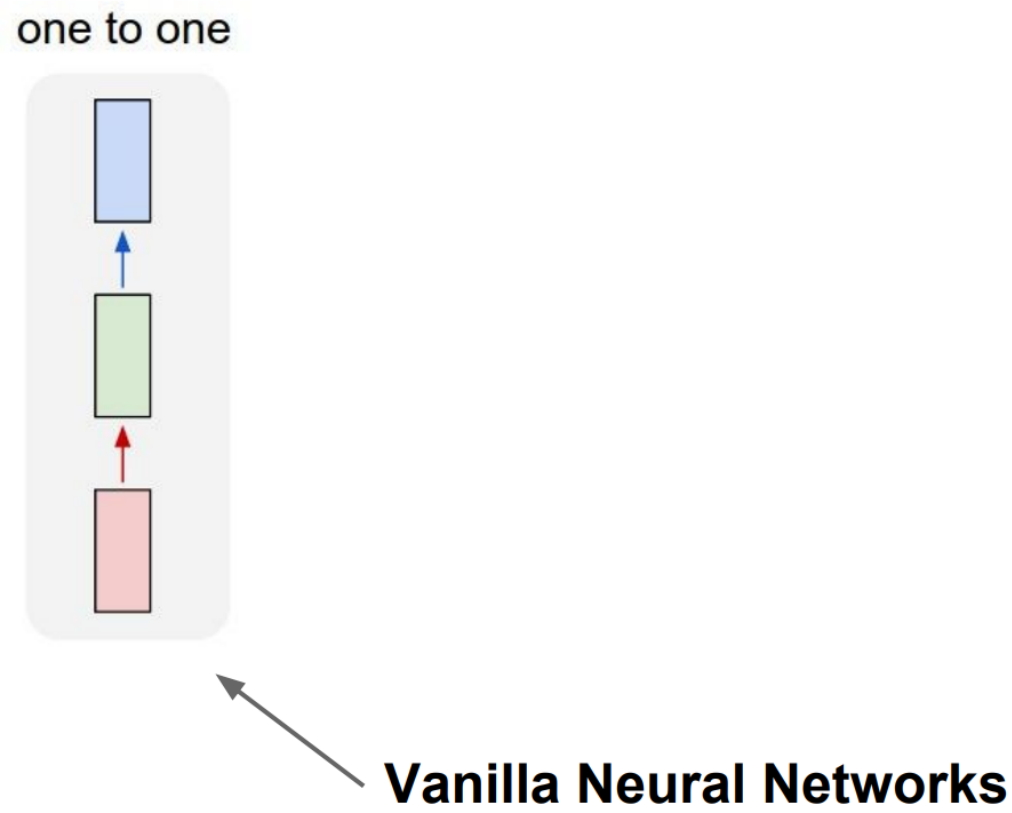
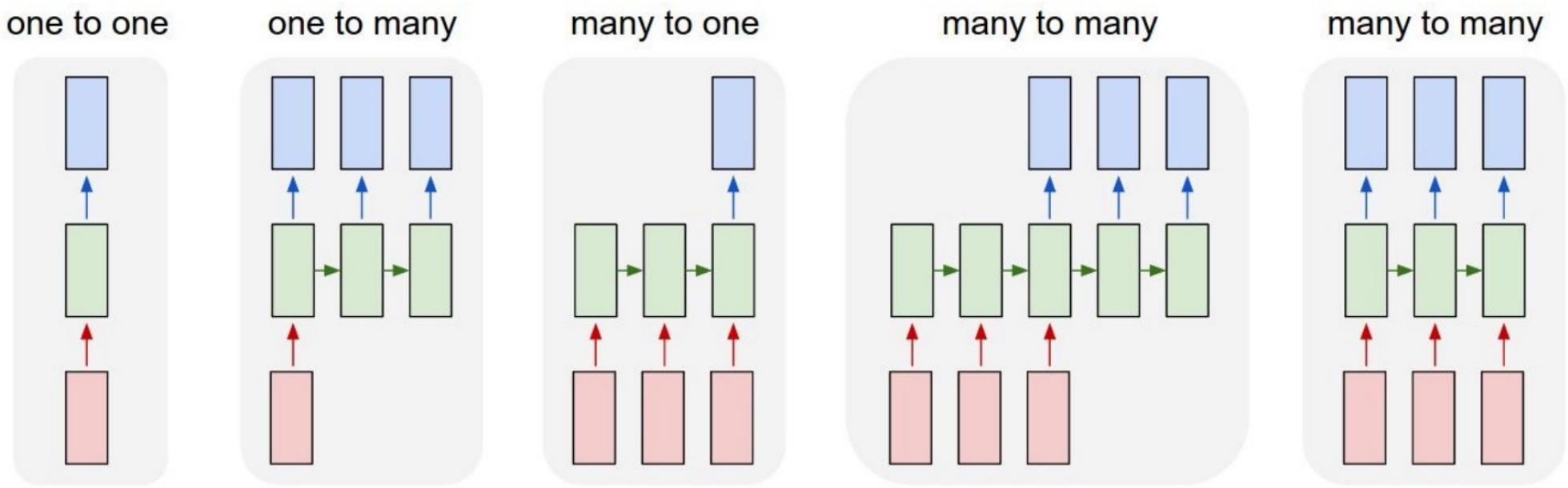


Image by (Fei-Fei Li & Justin Johnson & Serena Yeung, 2017)
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Recurrent Neural Networks: Process Sequences



e.g. **Image Captioning**
image -> sequence of words

Image by (Fei-Fei Li & Justin Johnson & Serena Yeung, 2017)
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Recurrent Neural Networks: Process Sequences

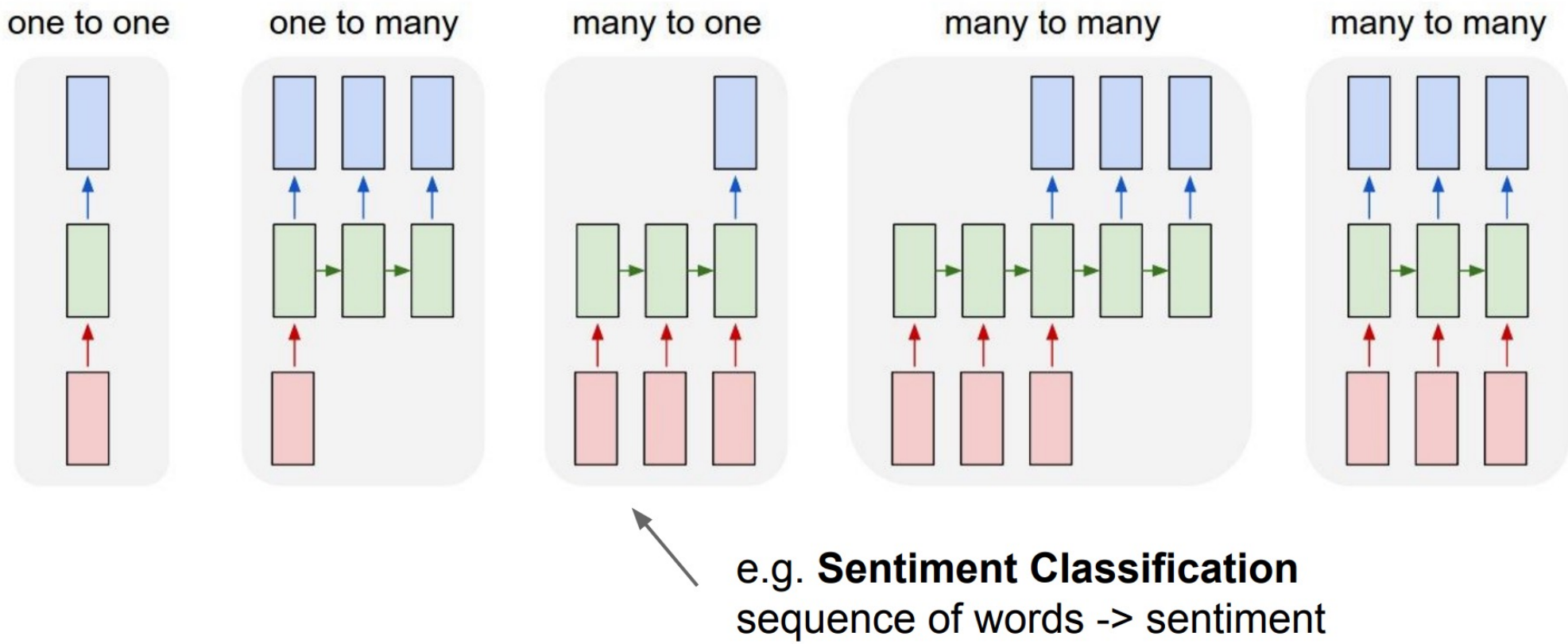
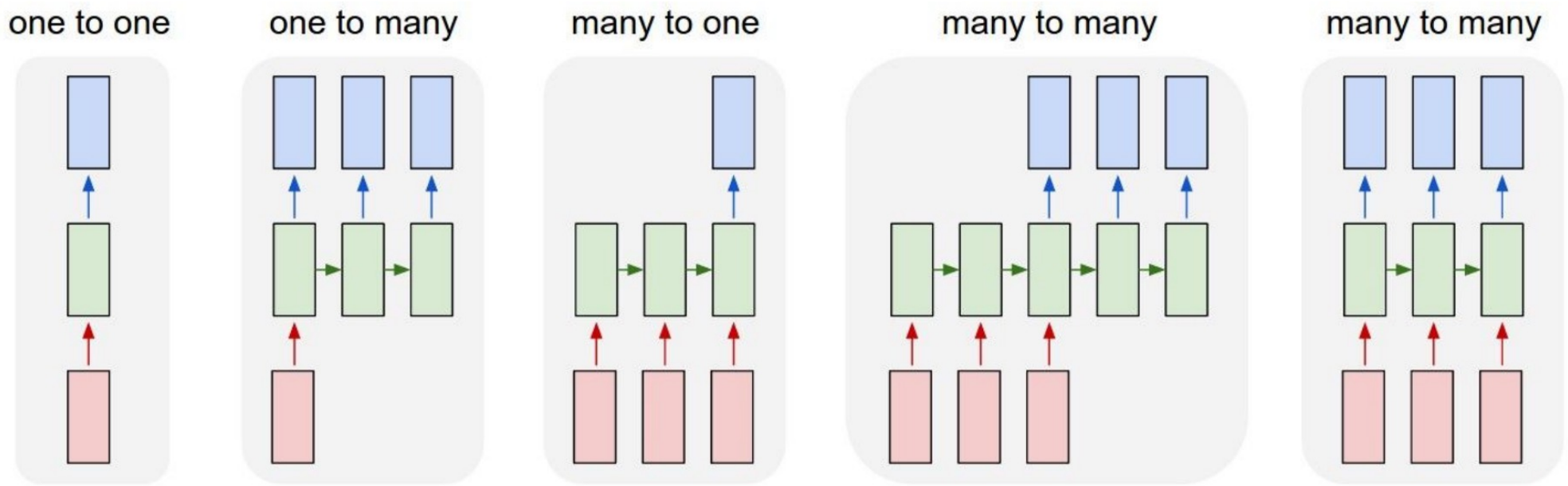


Image by (Fei-Fei Li & Justin Johnson & Serena Yeung, 2017)
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

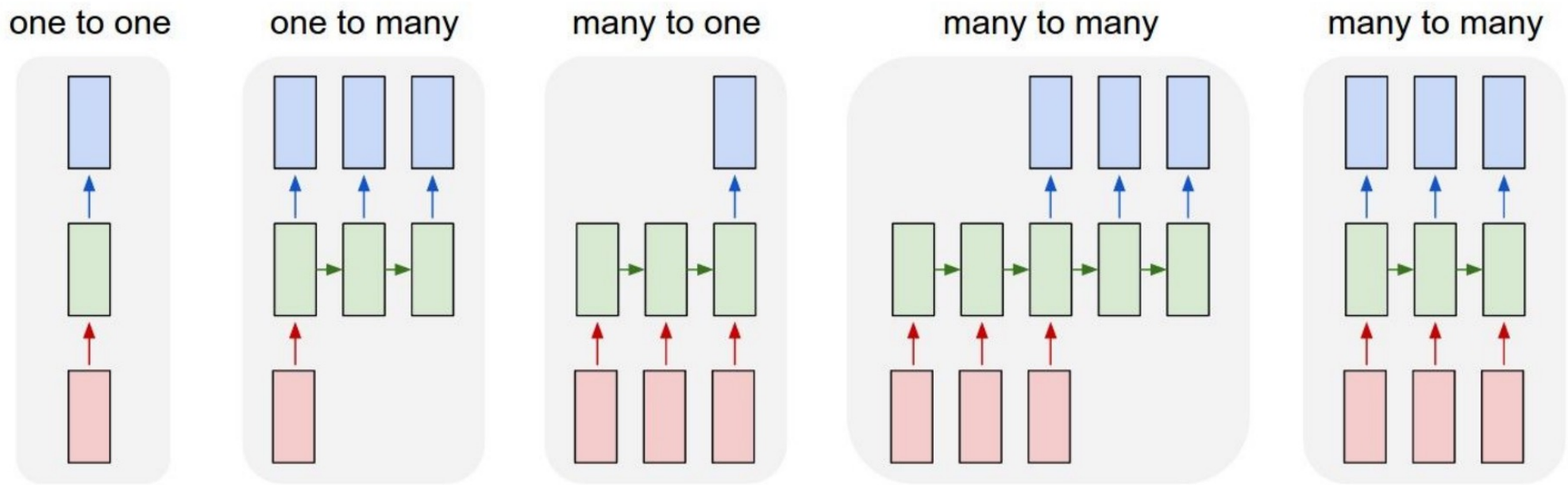
Recurrent Neural Networks: Process Sequences



e.g. **Machine Translation**
seq of words -> seq of words

Image by (Fei-Fei Li & Justin Johnson & Serena Yeung, 2017)
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Recurrent Neural Networks: Process Sequences



e.g. **Video classification on frame level**

Image by (Fei-Fei Li & Justin Johnson & Serena Yeung, 2017)
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step

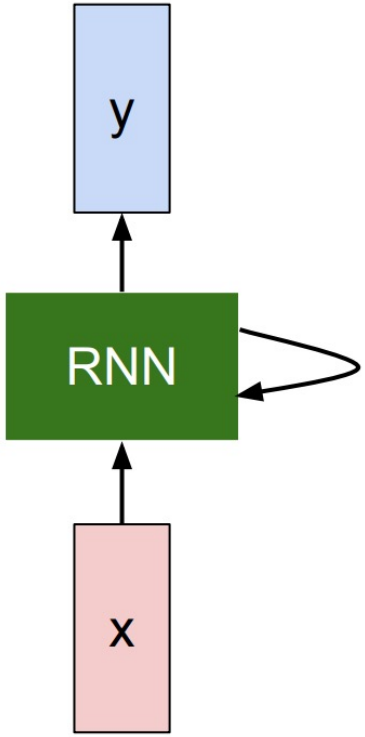


Image by (Fei-Fei Li & Justin Johnson & Serena Yeung, 2017)
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Recurrent Neural Networks

- Feedback output from time step *t*
  to input at next time step *t+1*
- Daniel Frank uses RNNs (LSTMs) for multi-step prediction of vehicle motion
  - ships
  - drones

Recent trend: replace recurrent neural networks by transformer architectures

Even more recent trend: new RNNs
xLSTM by Hochreiter; Mamba: state space model

8 When to Use DNNs

Curse of Dimensionality

Given a new data point x for which a result y must be predicted:

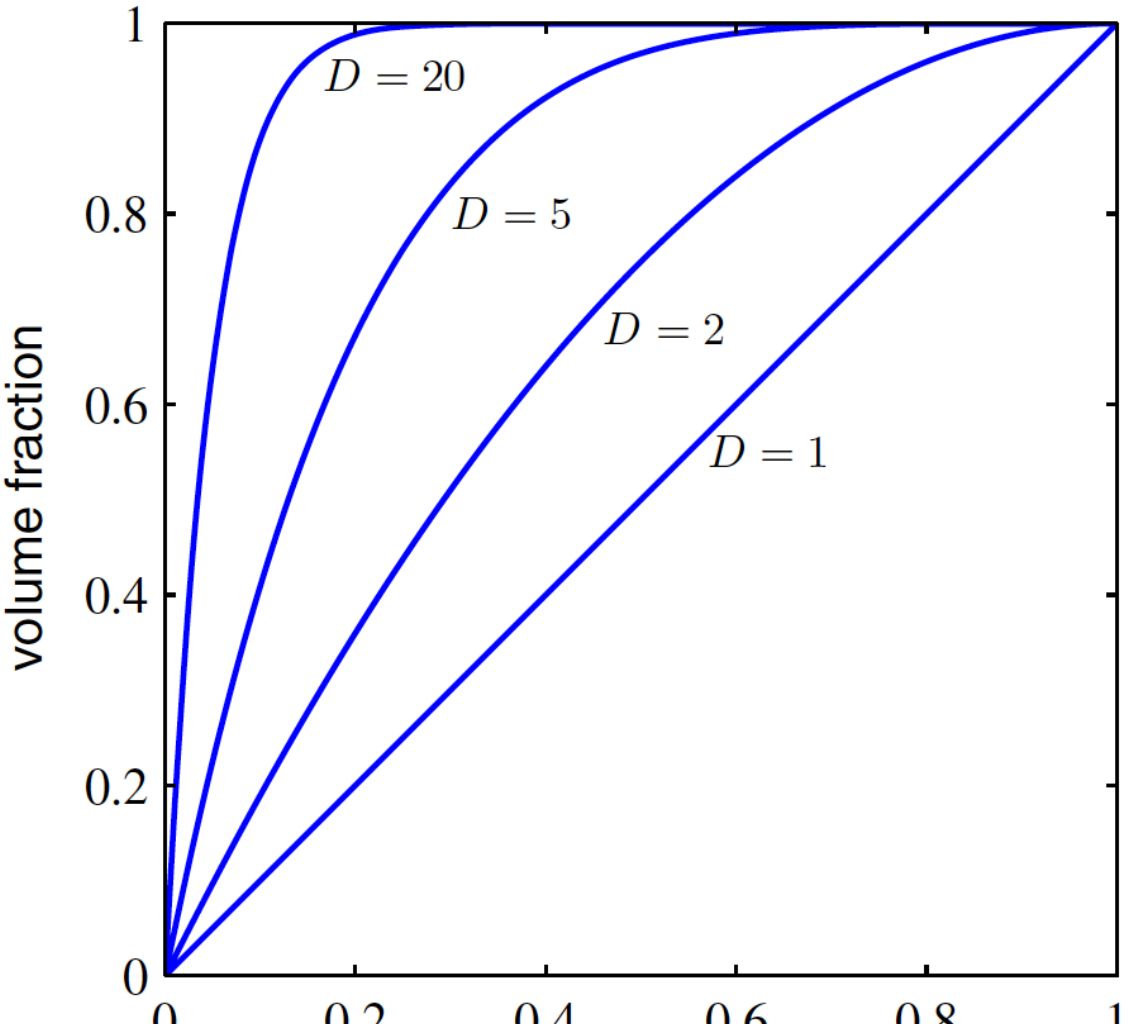
- Low dimensionality:
  x has few neighbors which are used to predict y
- High dimensionality:
  x has (too) many neighbors in roughly the same distance, makes it hard to predict y
- True for kNN, Decision Trees, SVMs, Bayes, etc.

Curse of dimensionality

Plot of the fraction of the volume of a sphere lying in the range $r = 1-\epsilon$ to $r = 1$ for various values of the dimensionality D.

(Bishop et al),
Fig. 1.22

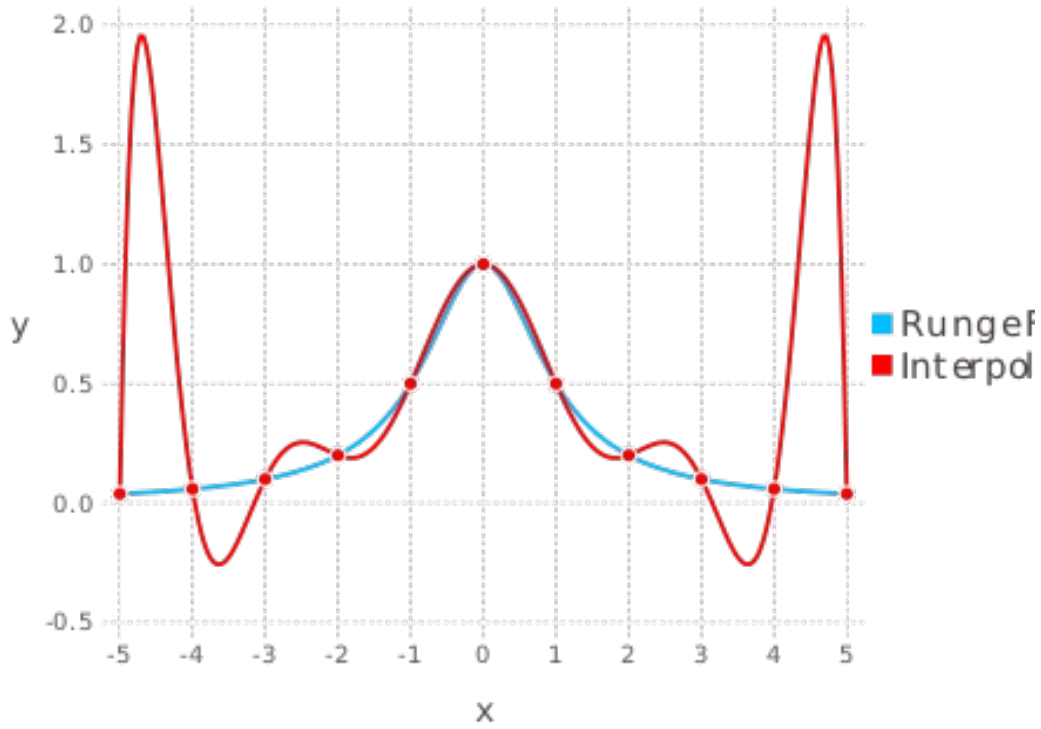
This makes it difficult for all „distance-based“ algorithms (kNN, linear regression, Naive Bayes...)



Why DeepNNs? Local constancy

What do kNN, SVM, Bayes, Decision Trees do?

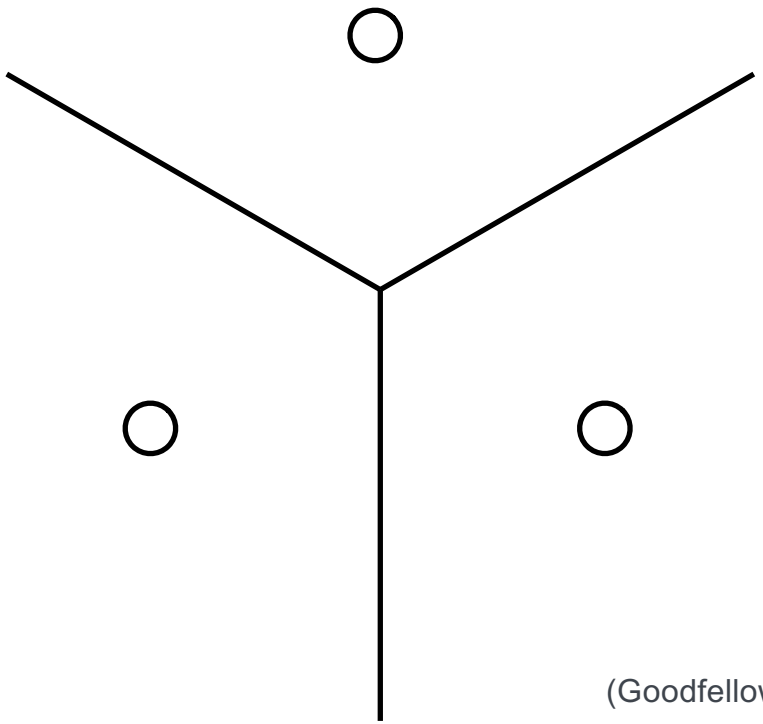
- Local constancy prior/smoothness prior:
  - learn a function f,
    such that
    $f(x) \approx f(x+\epsilon)$
  - Learn rather blue
    than red curve



https://de.wikipedia.org/wiki/Polynominterpolation#/media/File:Myplot_p2.svg

Why DeepNNs? Local constancy

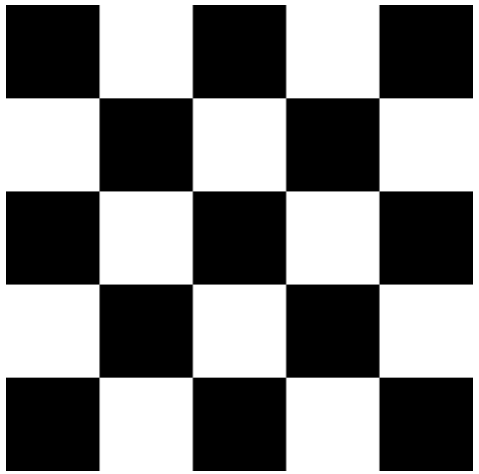
What do kNN, SVM, Bayes, Decision Trees do?



(Goodfellow 2016)

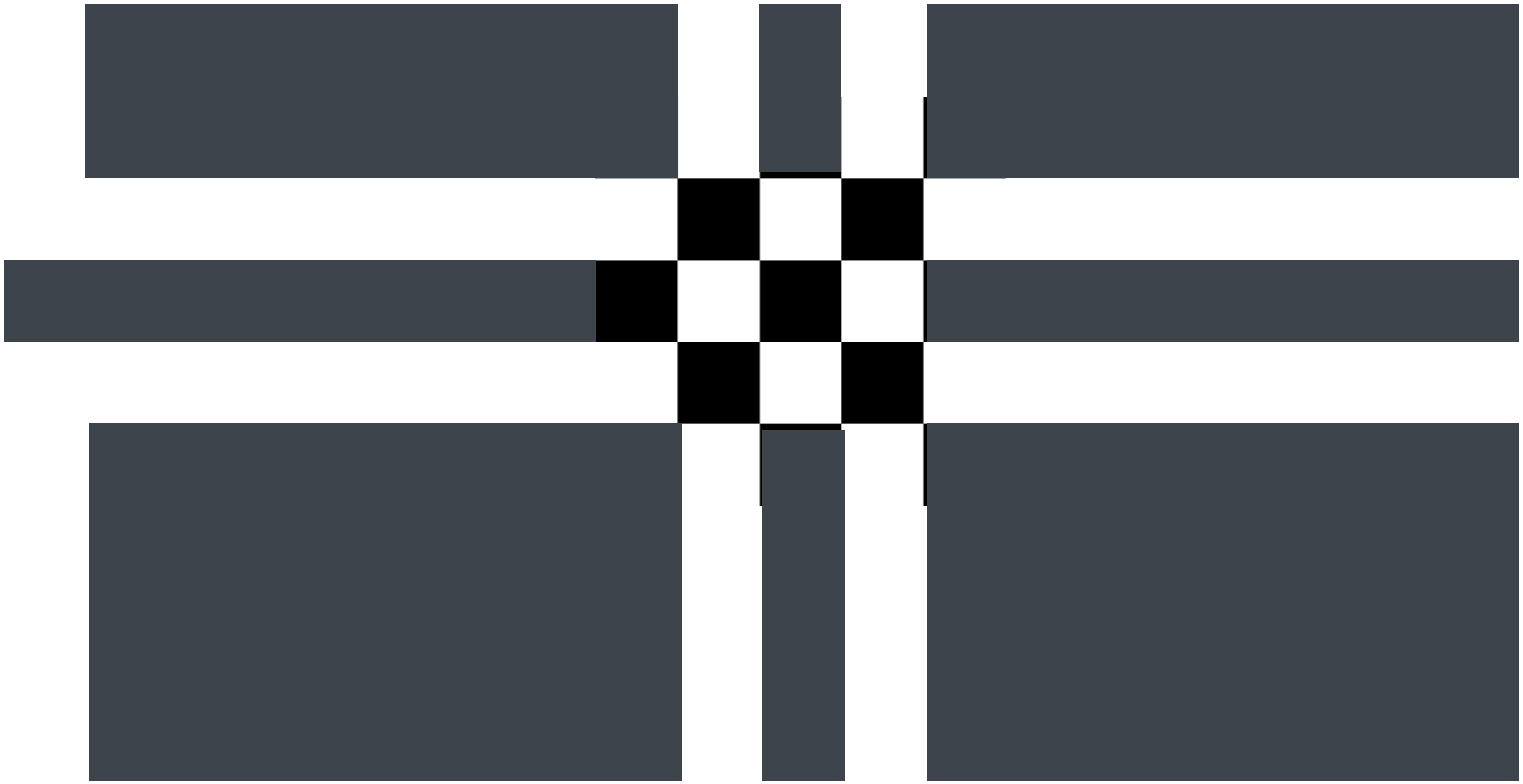
Local constancy

Training data



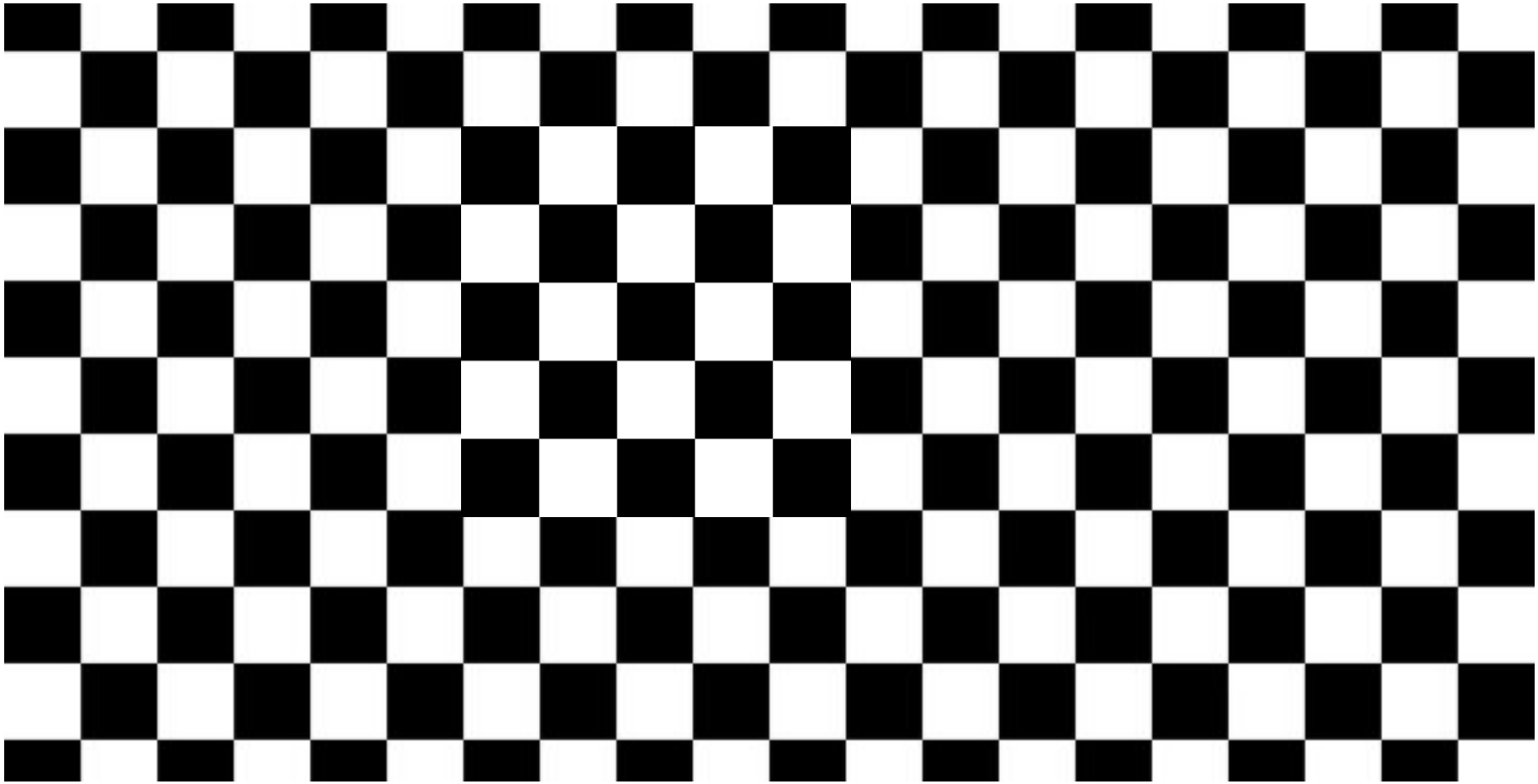
Local constancy - counterexample

Generalization by kNN or decision trees due to local constancy



Local constancy - counterexample

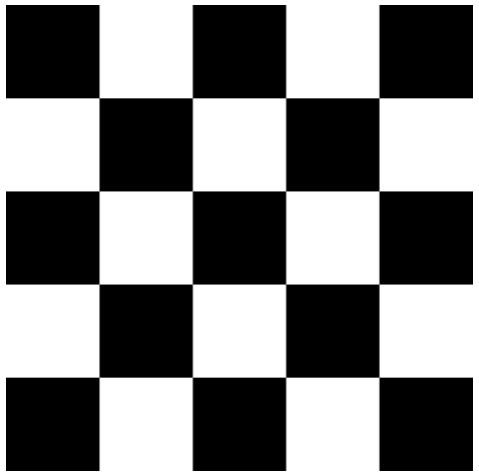
Intended generalization



Learning periodicity (and more)

kNN, SVM, etc.

- Model periodic functions
- Learn parameters for periodic functions

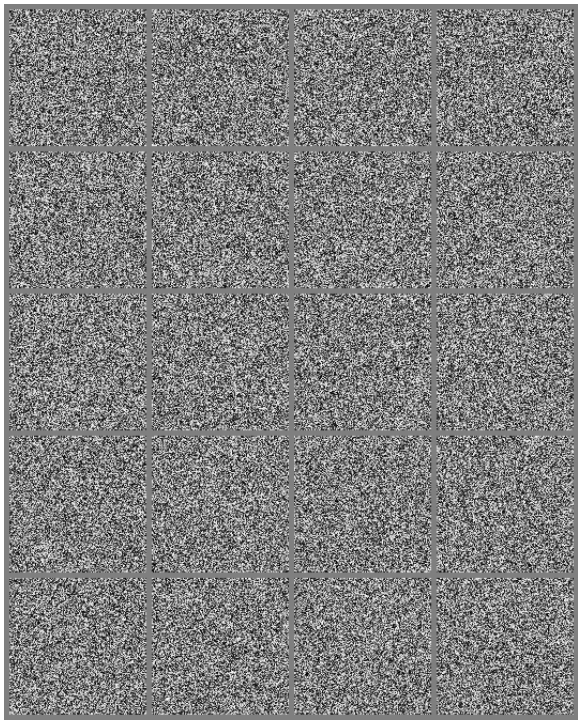


Deep learning

- Acquire periodicity as a hidden property

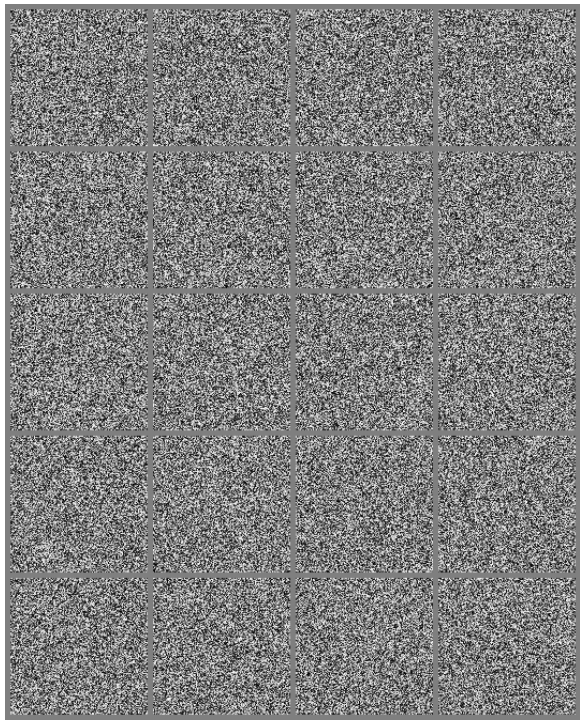
Manifold learning

- Given task:
  distinguish between male and female faces
- Which is this?



Manifold learning

- Given task:
  distinguish between male and female faces
- Which is this?
- Neither:
  - most pictures do not contain human faces at all
  - Only very few possible pictures actually do
  - > you are interested in a *manifold*



Manifold Learning

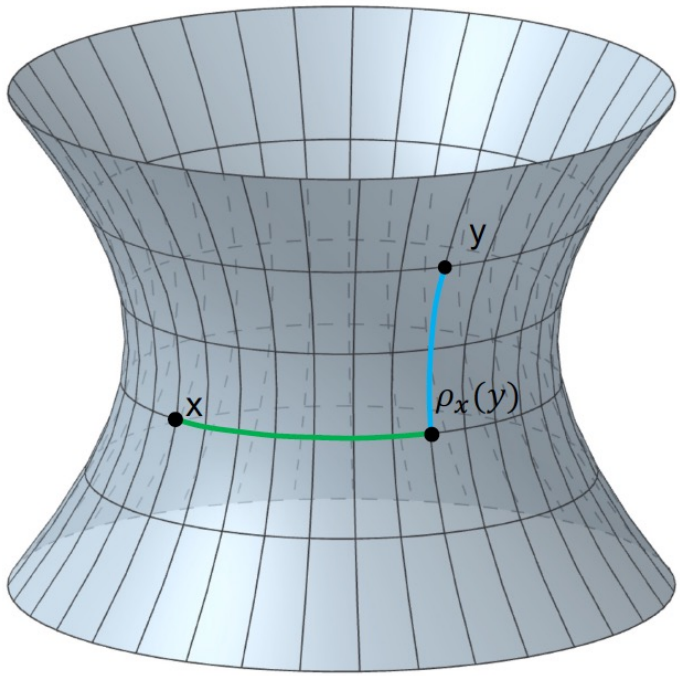
What is a manifold?

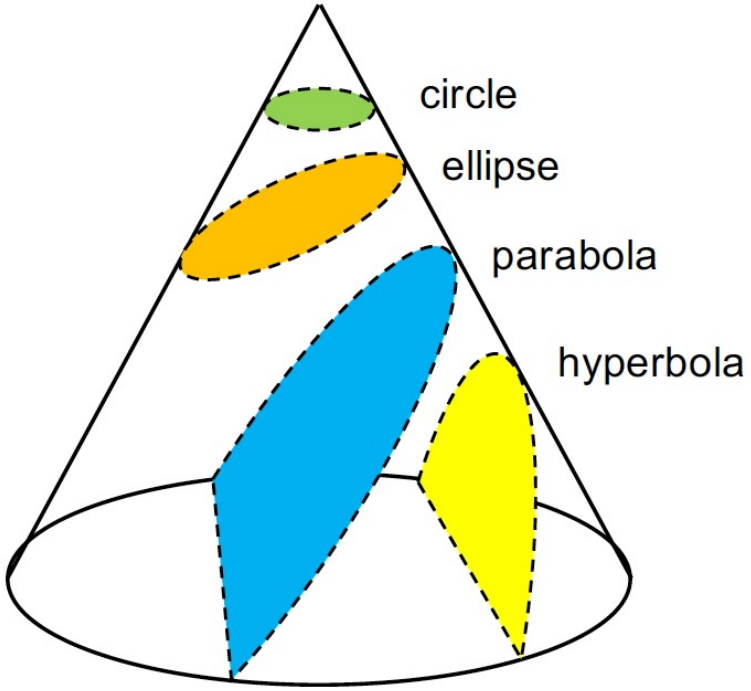
- A set of points D is called an n-dimensional manifold if it is locally homeomorphic to $\mathbb{R}^n$
- Example:
  - Circle: 1-dimensional manifold
    - But: not 1 dimensional, it exists in $\mathbb{R}^2$
  - Sphere: 2-dimensional manifold
    - But: not 2 dimensional, it exists in $\mathbb{R}^3$
  - Torus: 2-dimensional manifold
    - But: not 2 dimensional, it exists in $\mathbb{R}^3$
  - Line: 1-dimensional manifold and also globally homeomorphic to $\mathbb{R}^1$

Manifold is a data set with a topology typically embedded in a higher dimensional space

$$\mathbb{U}_{\alpha}^{p,q} = \left\{ \mathbf{x} = \left(x_1, x_2, \cdots, x_{p+q}\right)^{\mathsf{T}} \in \mathbb{R}^{p,q} : \| \mathbf{x} \|_q^2 = -\alpha^2 \right\}$$

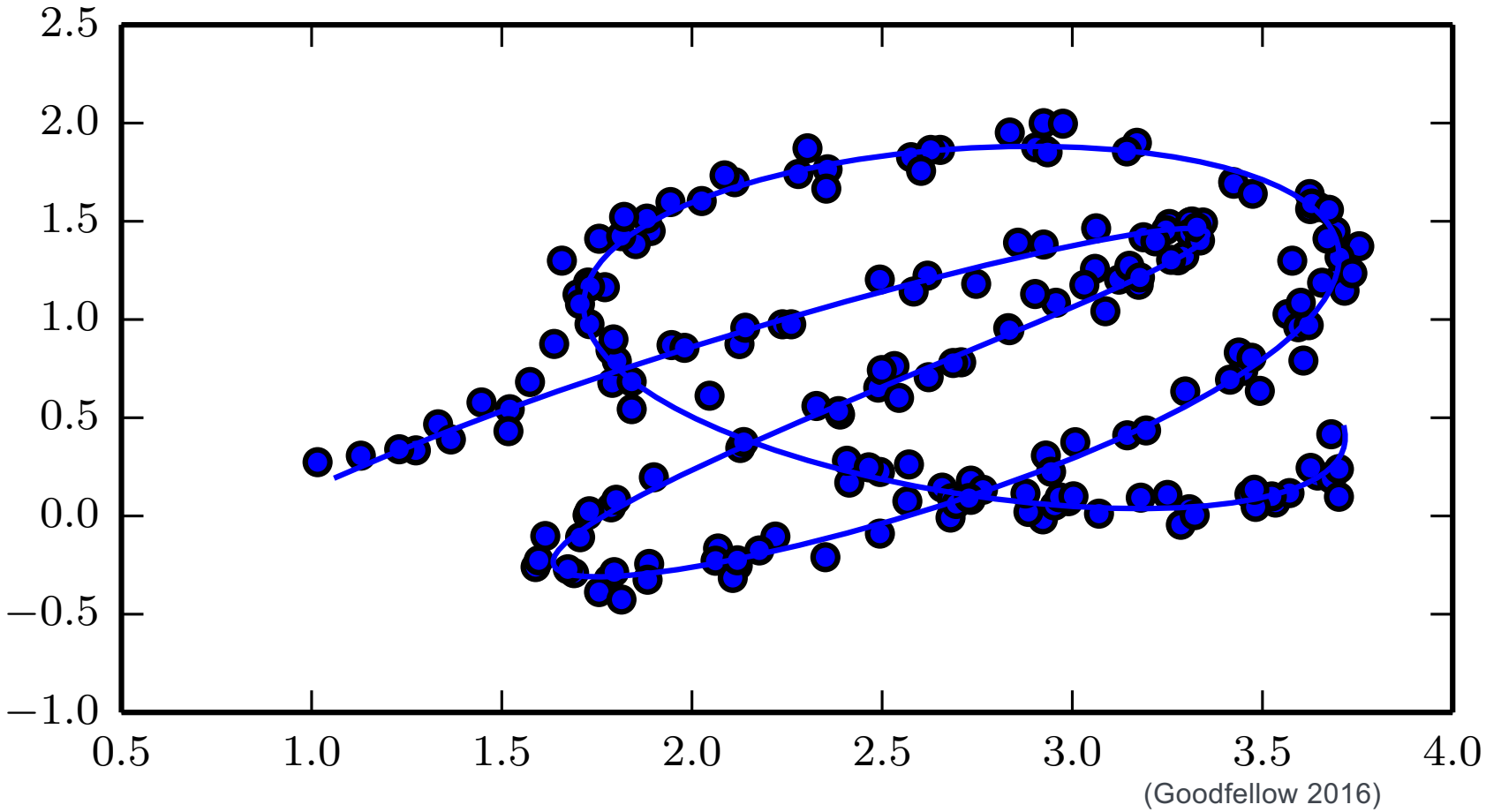
$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^{p,q}, \langle \mathbf{x}, \mathbf{y} \rangle_q = \sum_{i=1}^{p} \mathbf{x}_i \mathbf{y}_i - \sum_{j=p+1}^{p+q} \mathbf{x}_j \mathbf{y}_j$$





From: B. Xiong, S. Zhu, M. Nayyeri, C. Xu, S. Pan, C. Zhou, S. Staab.
Ultrahyperbolic Knowledge Graph Embeddings.
In: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2022.

Example for a manifold



(Goodfellow 2016)

A „learning manifold“
subset of possible pictures with faces

QMUL dataset from (Goodfellow 2016)

In machine learning the notion „manifold“ is used in a loose way!

„Remembering“ vs. „Generalization“

- neural networks are good at compression
- they “remember“ objects
  - test for this*: take a test data set, randomly assign classes; check for training accuracy – very good – in spite of randomness
  - prediction will be random – since there is no meaningful class belonging to discover
- neural networks require large data set sizes
  - Bayesian neural networks can cope with smaller sizes like 5000 examples

*Unfortunately, I could not identify and find that paper again ☹

Remembering in Huge Models (for some very specific, but exciting tasks)

- GPT-3 is a DNN language model
  - trains 175 Billion parameters
    - remember which word follows which other word AND generalize
  - would cost over **$4.6 million** to train in the cloud
  - Some entry point for discussions
    https://lambdalabs.com/blog/demystifying-gpt-3/

- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Agarwal, S. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.
- *[Submitted on 28 May 2020 (v1), last revised 5 Jun 2020 (this version, v3)]*

Remembering if seen often enough

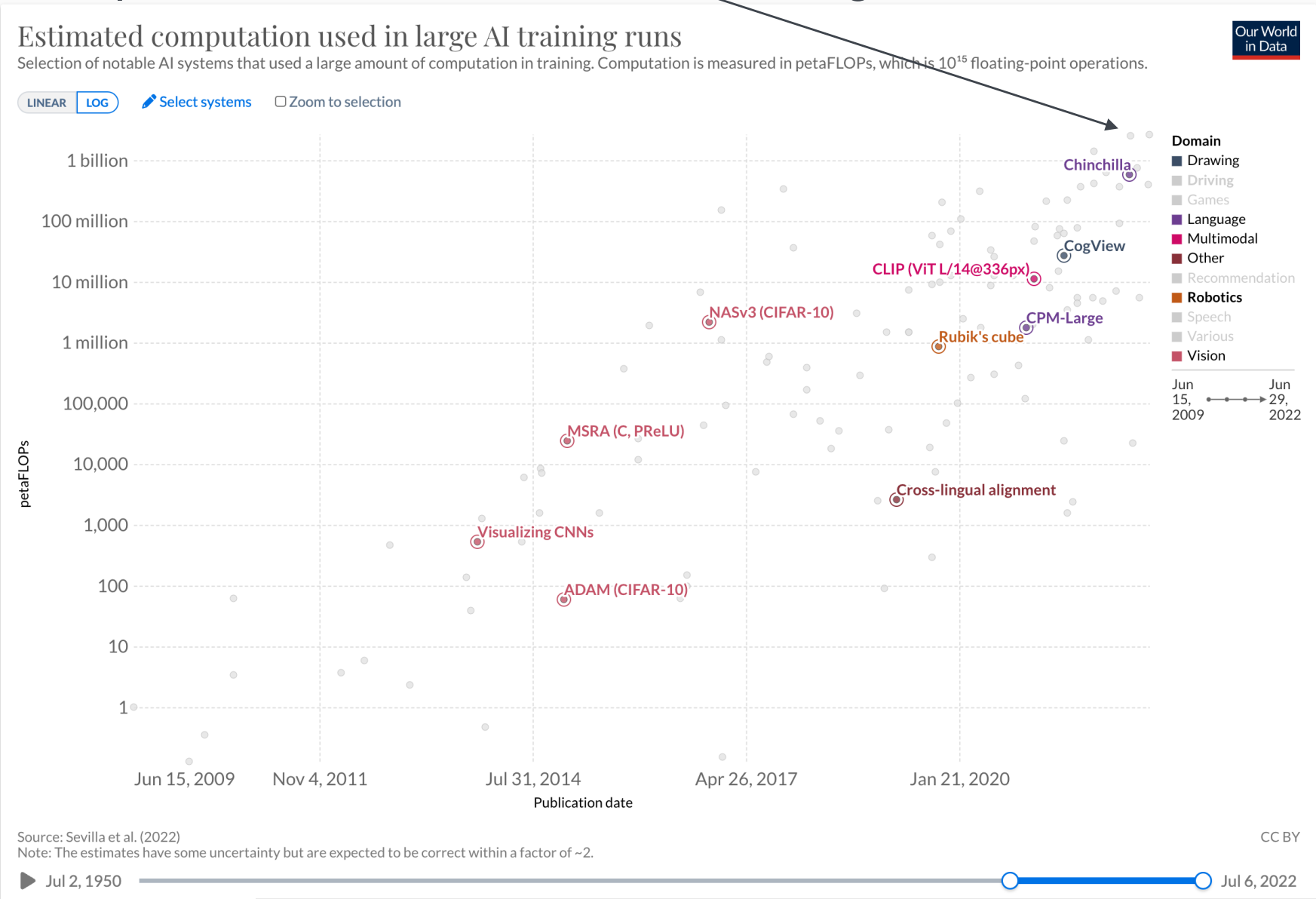
Who is Steffen Staab and in which year was he born?

Steffen Staab is a computer scientist known for his contributions to the fields of semantic web, knowledge graphs, and artificial intelligence. He was born in the year 1968.

Who is King Charles and in which year was he born?

King Charles III, born Charles Philip Arthur George, is the king of the United Kingdom and other Commonwealth realms. He was born on November 14, 1948. He ascended to the throne on September 8, 2022, following the death of his mother, Queen Elizabeth II.

Compute cost estimates for Palm range in 9-23 Mio USD



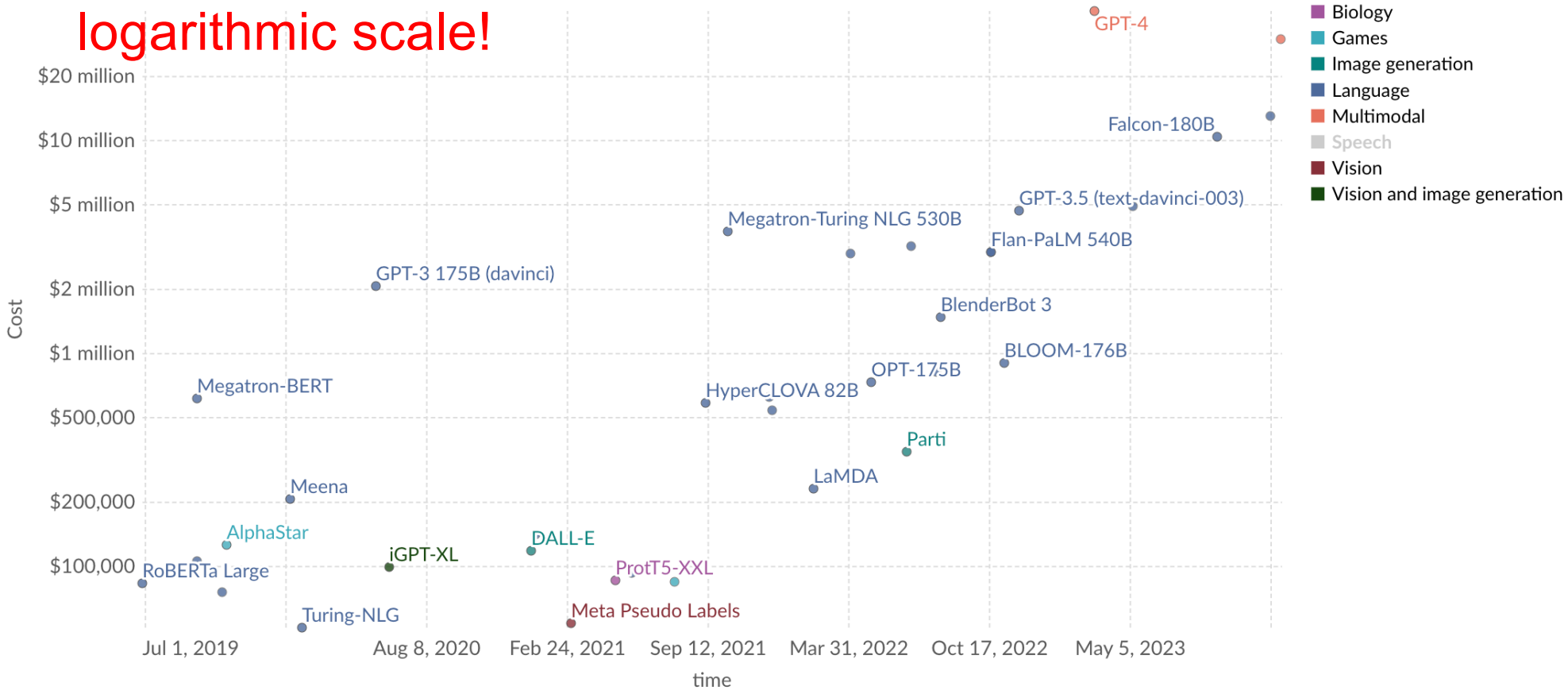
Hardware and energy cost to train notable AI systems

This data is expressed in US dollars, adjusted for inflation.

Our World in Data

Table | Chart

Settings



Play time-lapse | Dec 8, 2015 | Dec 6, 2023

References

Ian Goodfellow and Yoshua Bengio and Aaron Courville. Deep Learning MIT Press, 2016. Online: http://www.deeplearningbook.org/

(Note: I found the chapter in Hastie et al less approachable)

Universität Stuttgart
KI

Thank you!

Steffen Staab

E-Mail Steffen.staab@ki.uni-stuttgart.de
Telefon +49 (0) 711 685-88100
www. ki.uni-stuttgart.de/

Universität Stuttgart
Analytic Computing, KI
Universitätsstraße 32, 50569 Stuttgart