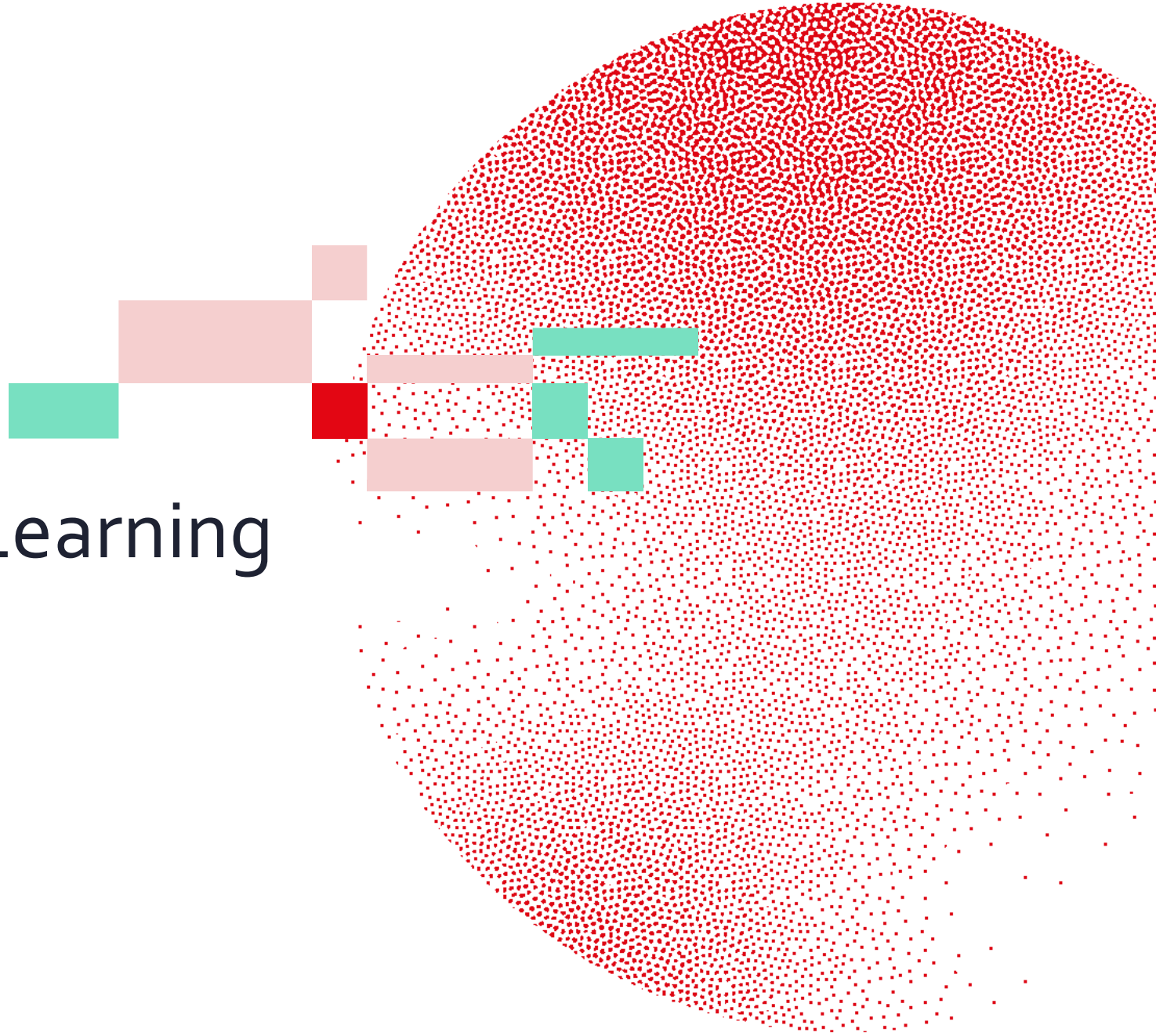
Swiss Institute of Bioinformatics



Introduction to Deep Learning

Markus Müller, Van Du Tran, Wandrille Duchemin

SIB Swiss Institute of Bioinformatics

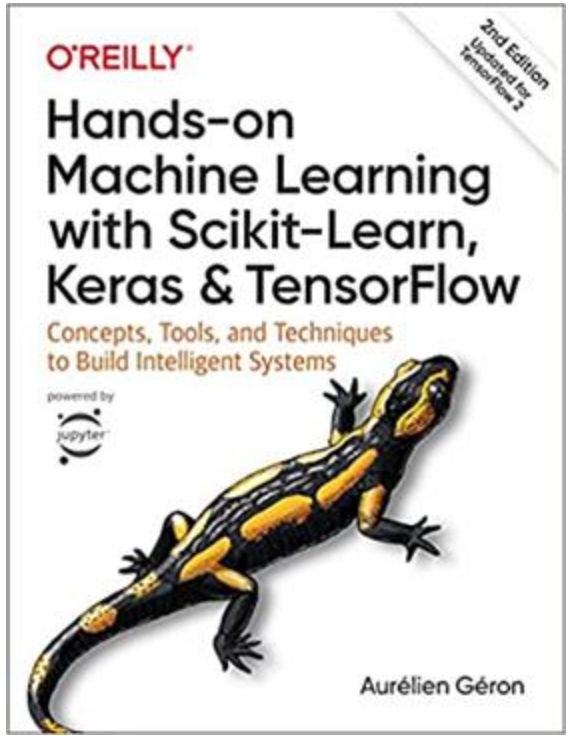
November 2024

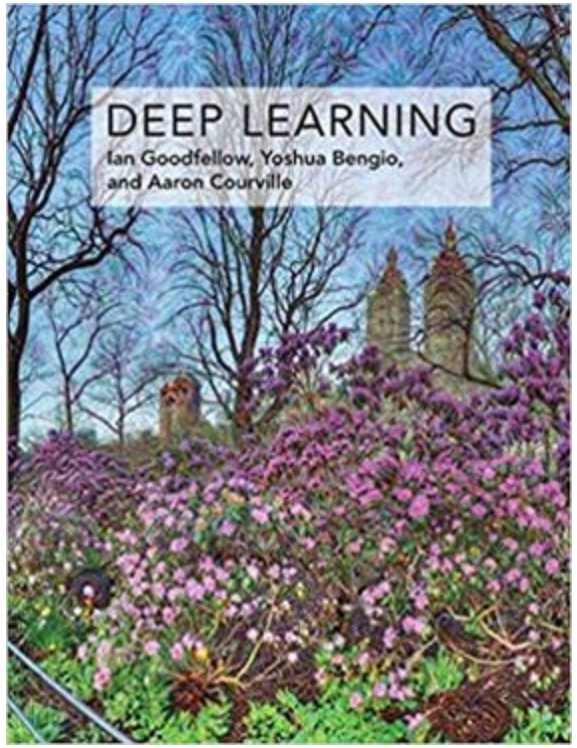
elixir SWITZERLAND

Outline

- History of deep learning
- Principles of deep learning
- How do deep neural networks learn?
- How do deep neural networks generalize?
- Biological and medical applications

Learning Deep Learning (DL)

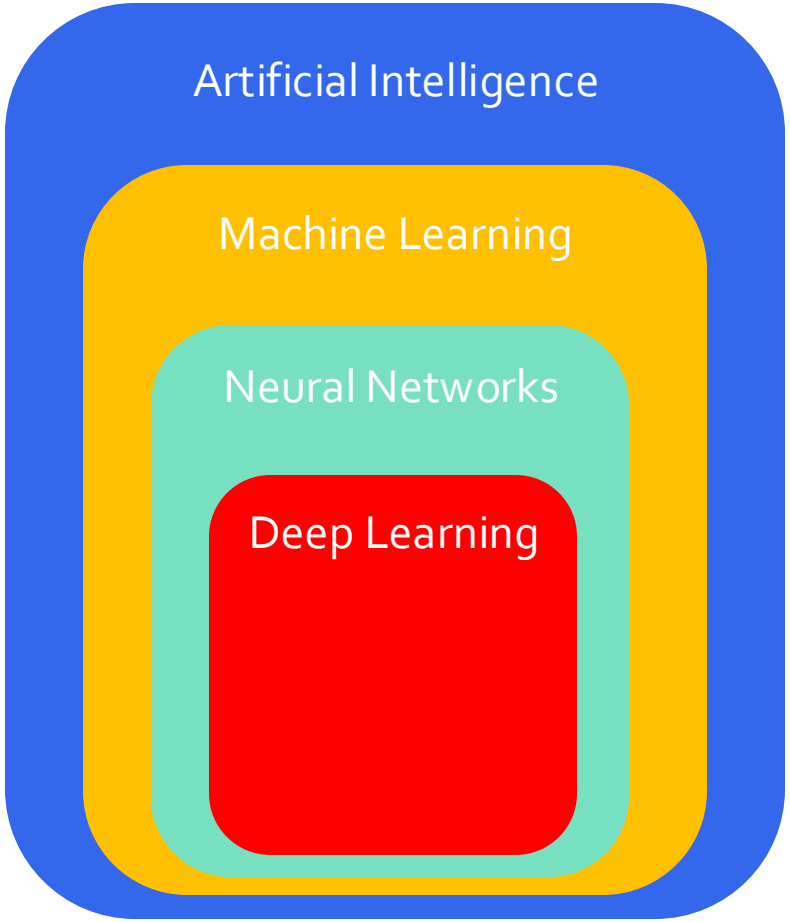




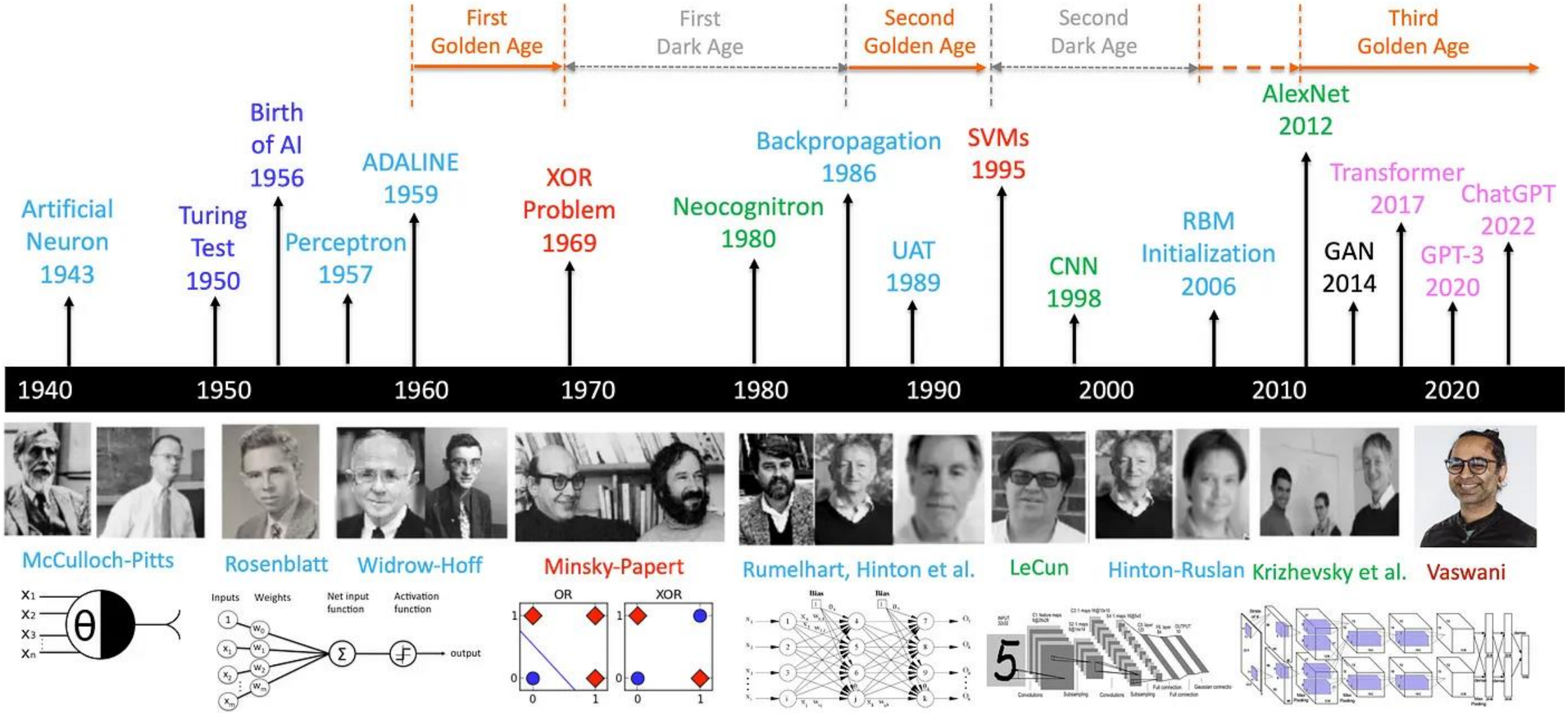
- Coursera: Andrew Ng lecture
- MIT 6.S191 Deep learning intro
- https://www.kaggle.com/learn/intro-to-deep-learning
- https://theaisummer.com/
- The carpentries incubator deep learning intro
- Tensorflow or PyTorch docs
- https://playground.tensorflow.org/

Outline

- History of deep learning
- Principles of deep learning
- How do deep neural networks learn?
- How do deep neural networks generalize?
- Biological and medical applications

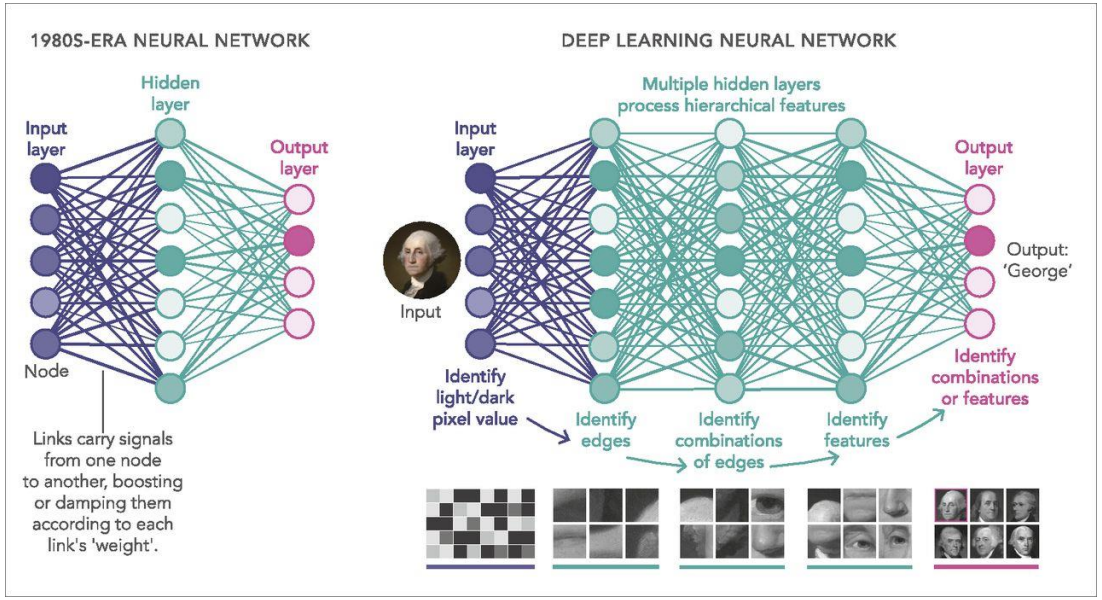

Artificial Intelligence
Machine Learning
Neural Networks
Deep Learning

A Brief History of AI with Deep Learning

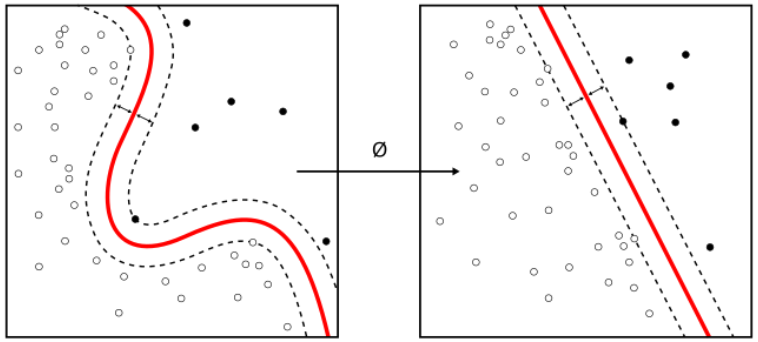


https://medium.com/@lmpo/a-brief-history-of-ai-with-deep-learning-26f7948bc87b

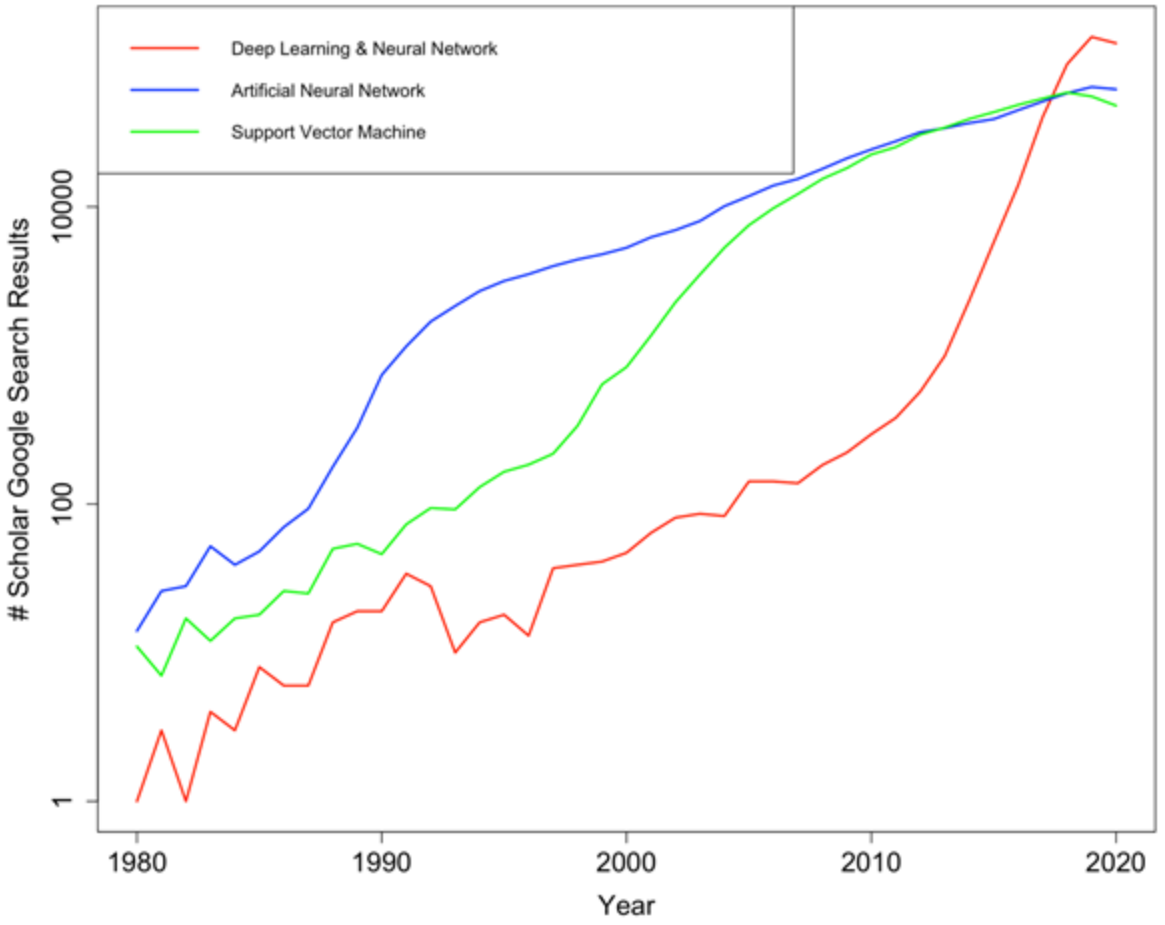
Deep Learning citations



Waldrop, PNAS 2019



https://commons.wikimedia.org/wiki/File:Kernel_Machine.svg

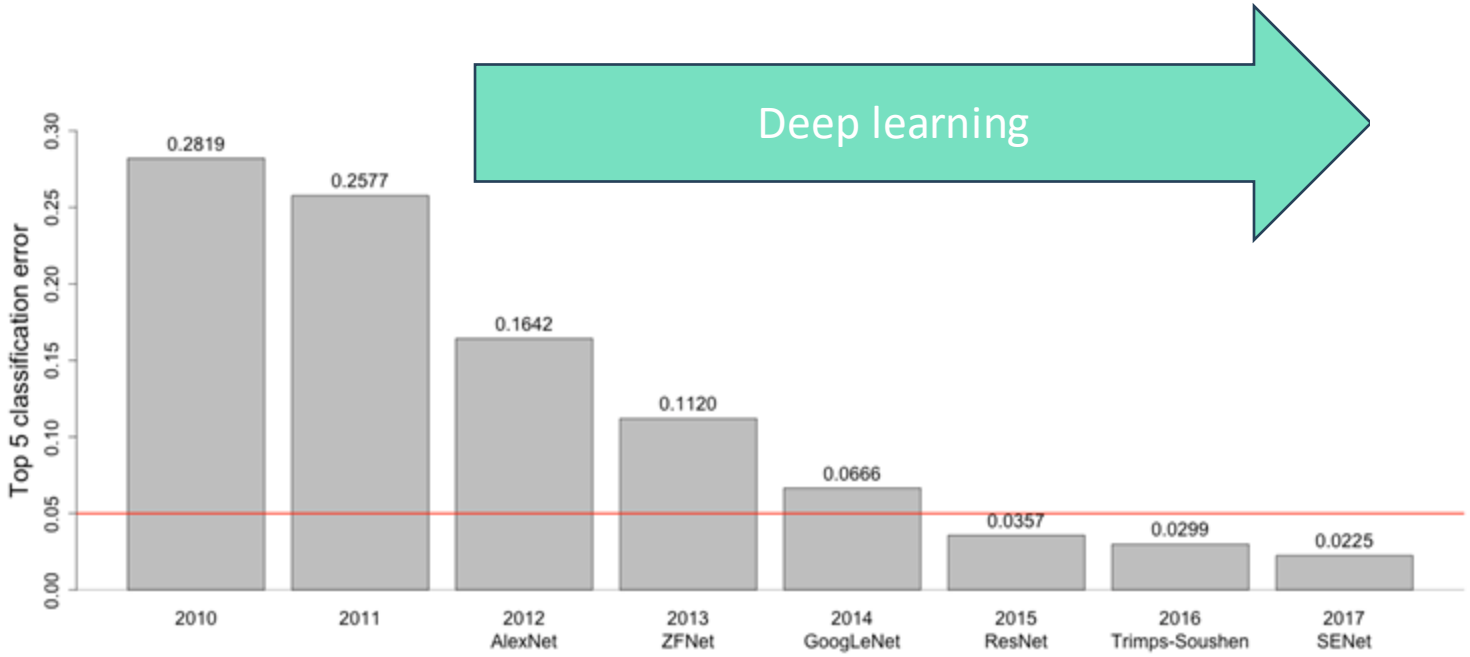


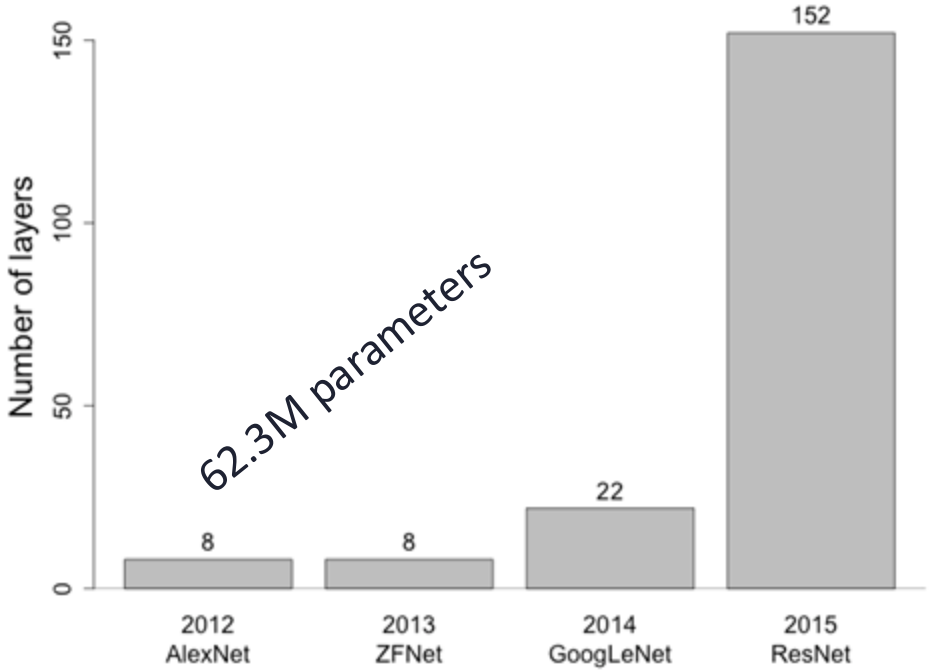
Early successes of DL

- Neural Computation & Adaptive Perception Program lead by G. Hinton, starts at CIFAR (2004)
- DNNs outperform other approaches for
  - phone classification: DBN (Mohamed et al. NIPS, 2009) on TIMIT dataset @ CIFAR
  - image classification: CNN (AlexNet, Krizhevsky et al. NIPS, 2012) using GPUs, ReLU, dropout and data augmentation. Winner in ImageNet ILSVRC classification challenge (2012) @ CIFAR
  - language translation: RNN-LSTM (Sutskever et al. NIPS, 2014) @ Google
- Training of DNNs on GPUs (since 2005), especially on NVIDIA GPUs with CUDA (after 2009) provided 10x speed-up (Raina et al, ICML, 2009)
- Tensorflow (2015), PyTorch (2016), MxNet (2015), CTNK (2016)
- Transformer architecture (2017)
- ACM A. M. Turing Award for Geoffrey Hinton, Joshua Bengio and Yann LeCun (2018)
- Nobel Prize in Physics for John Hopfield and Geoffrey Hinton (2024)
- Nobel Prize in Chemistry for Demis Hassabis, John Jumper and David Baker (2024)

Going deep (ILSRVC challenge)

1’200’000 images
1000 categories

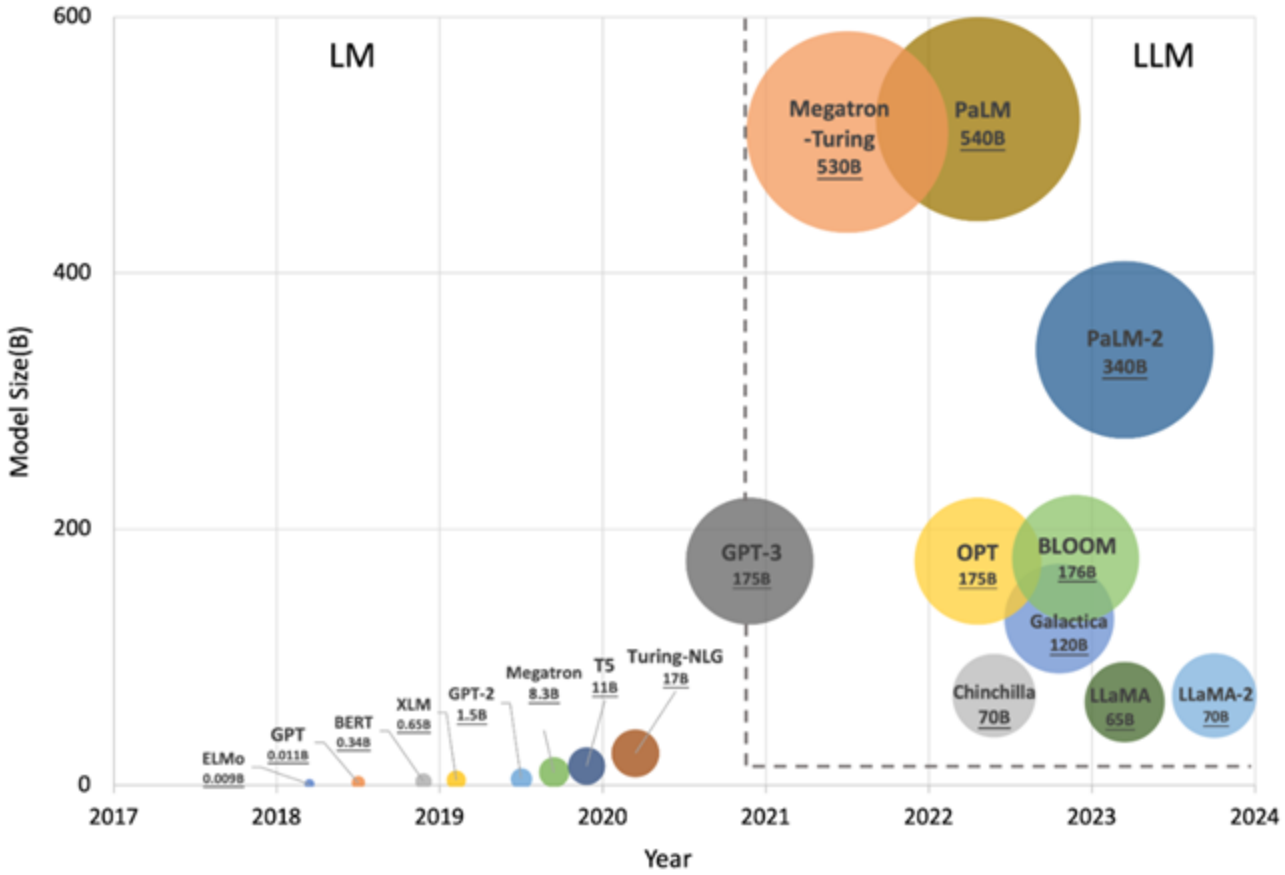




[Russakovsky et al., Int J Comput Vis 2015]

[https://www.image-net.org/]

Going deeper (Large Language Models - LLMs)



[He et al. arXiv, 2023]

- GPT-4 (OpenAI)
  - 10^{12} parameters
  - $100m for development
  - Running all Google searches with GPT-4: $6bn a year
- More parameters not always lead to better performance
- Parameters can be compressed without performance loss
- Algorithms and chips can be improved
- Adapting the models to more specialized tasks needs less parameters and training time

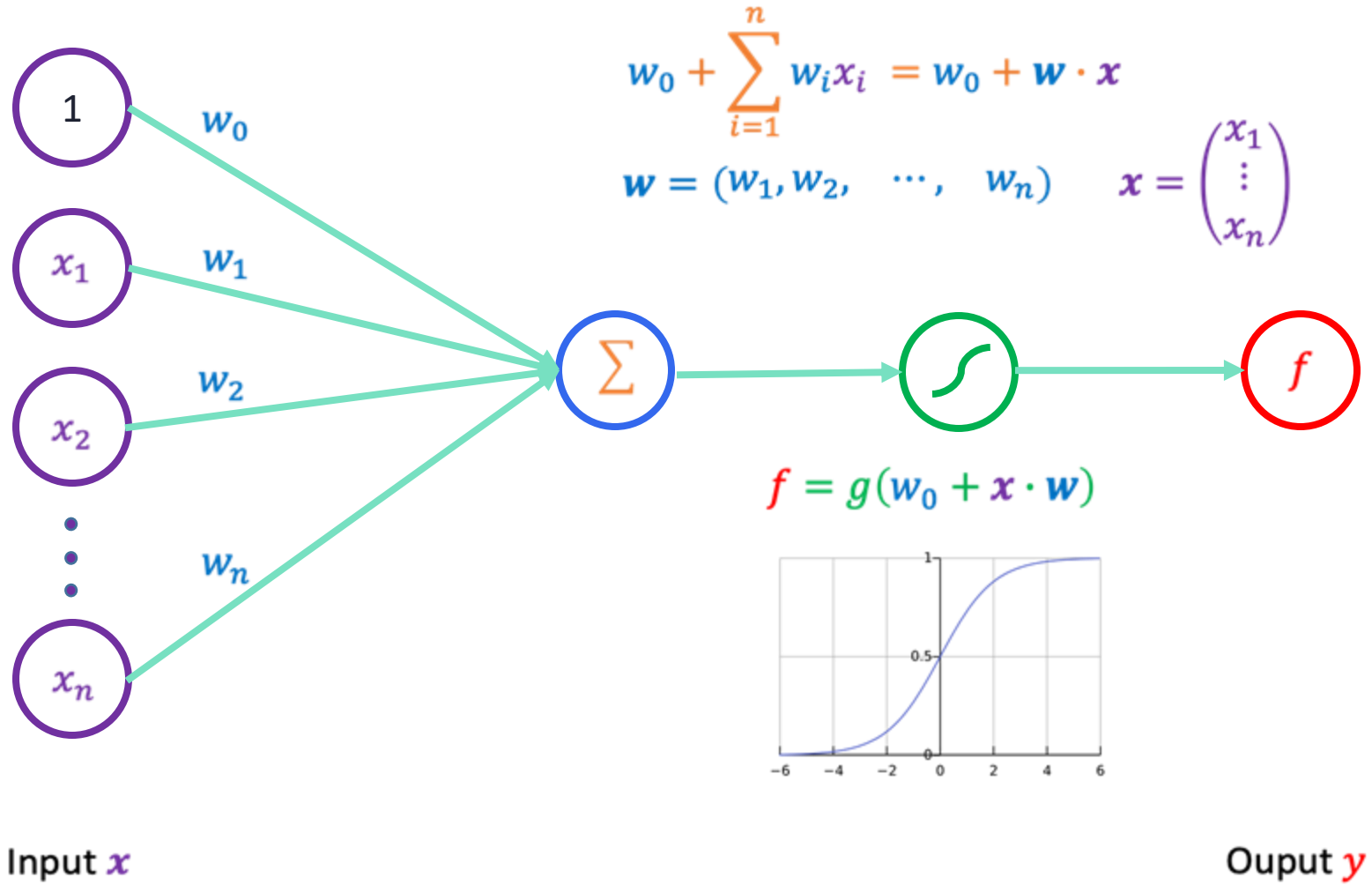
Deep learning success

- The success of DL in image, language, and sound processing is driven by the availability of large volumes of training data, an increase in computational power (e.g. GPUs).
- Improvements in weight initialization, activation functions, gradient descent, batching of training data, dropout of neurons made learning robust and feasible for deep networks without the need for pretraining.
- Improvement in the architecture of DNNs (convolution, skip connections, recurrency, attention).
- Availability of easy-to-use DL frameworks such as Tensorflow or PyTorch.

Outline

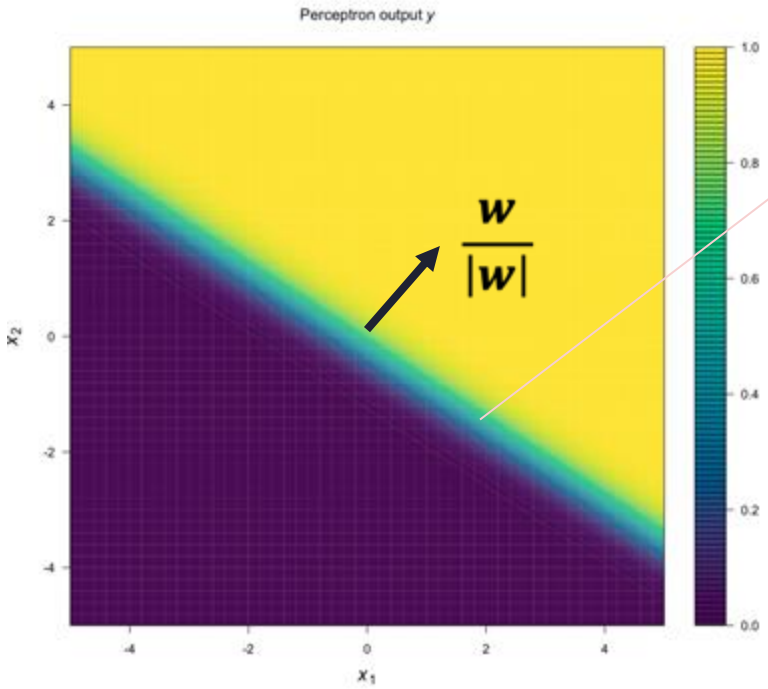
- History of deep learning
- Principles of deep learning
- How do deep neural networks learn?
- How do deep neural networks generalize?
- Biological and medical applications

Single-layer perceptron

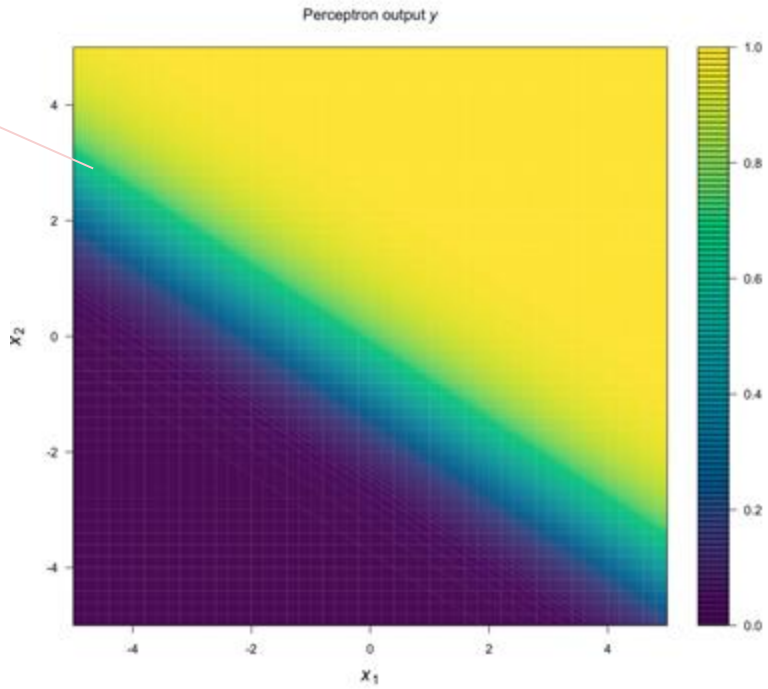


Single-layer perceptron ~ logistic regression

$w_0 + \boldsymbol{x} \cdot \boldsymbol{w} = 0$

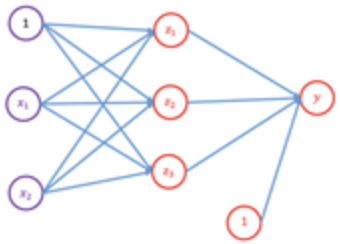


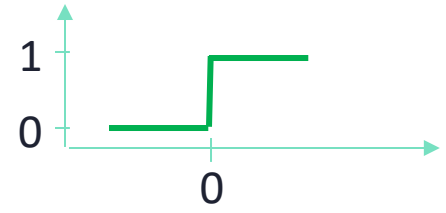
$|\boldsymbol{w}|$ large



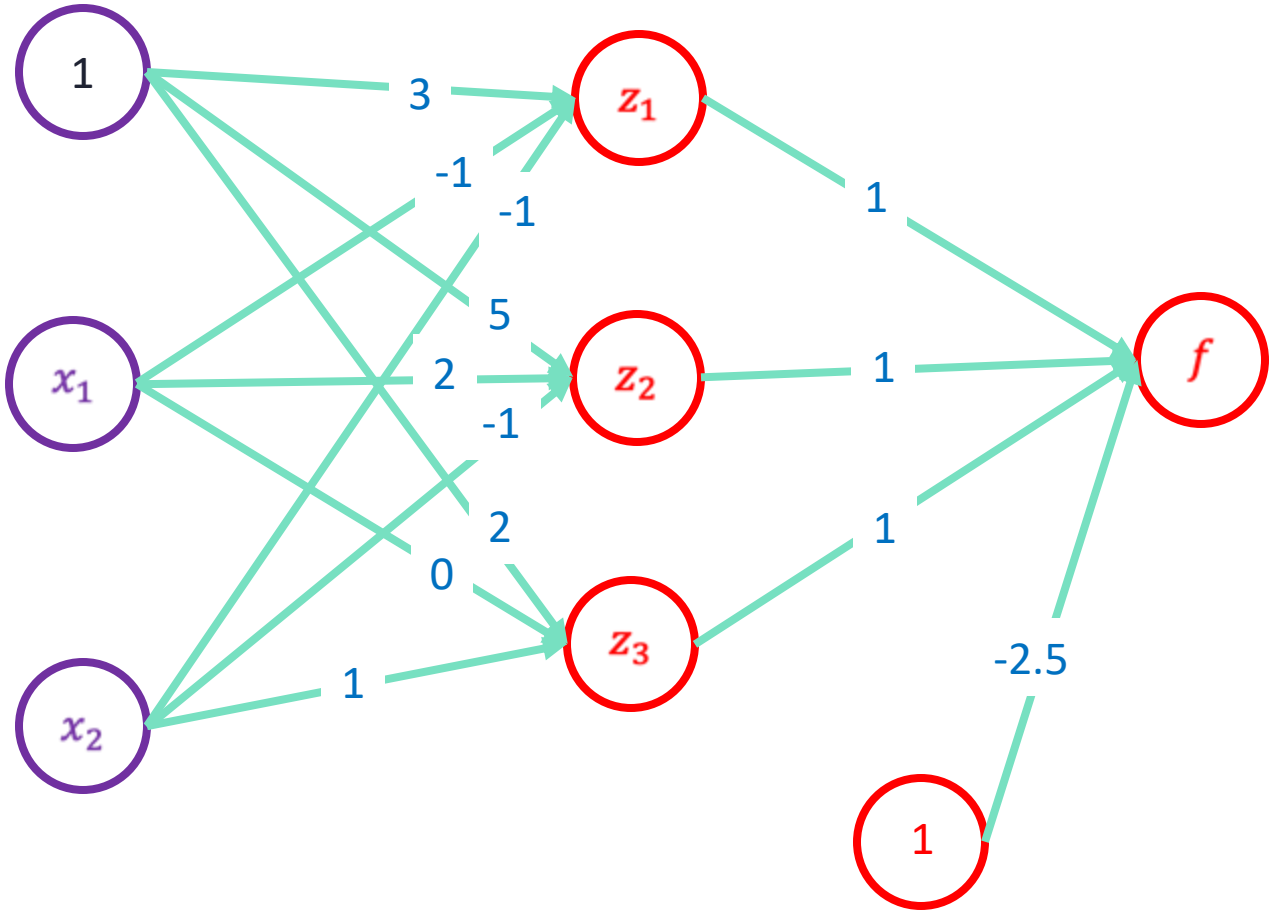
$|\boldsymbol{w}|$ small

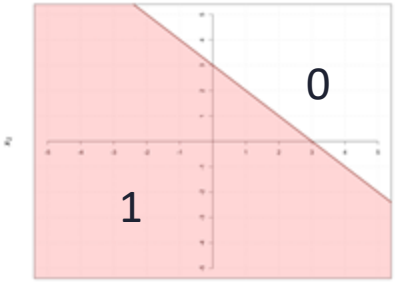
Multi-layer perceptron (MLP)

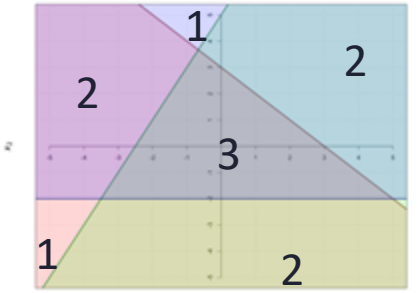




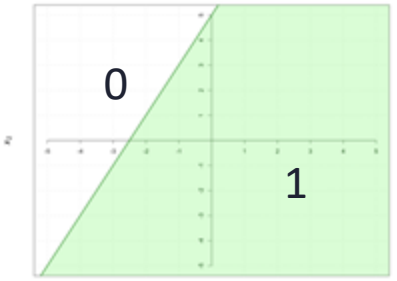
Binary step

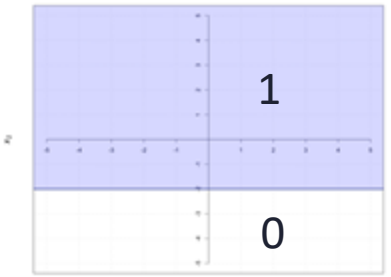


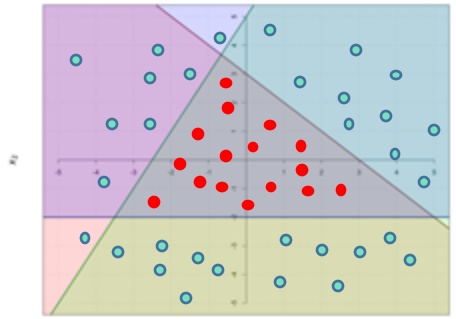




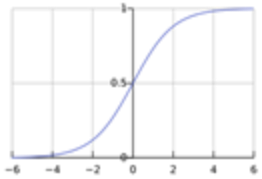
-2.5



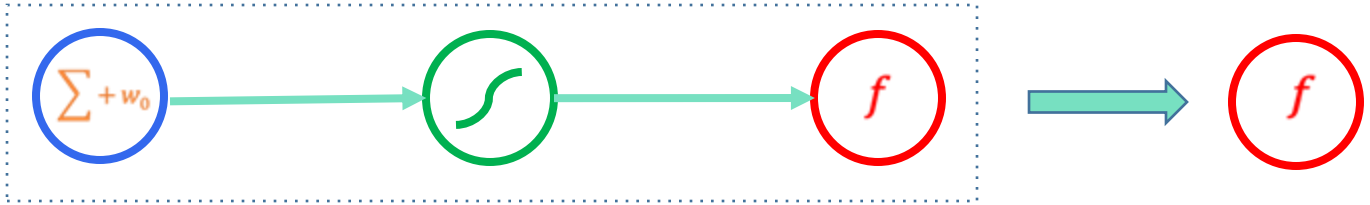


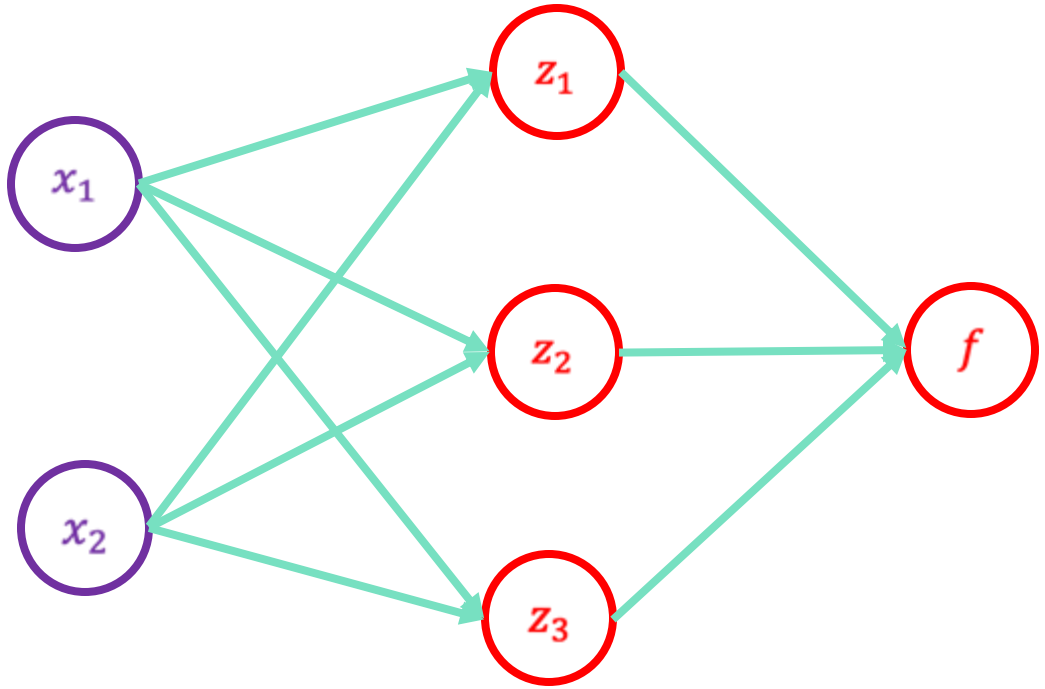


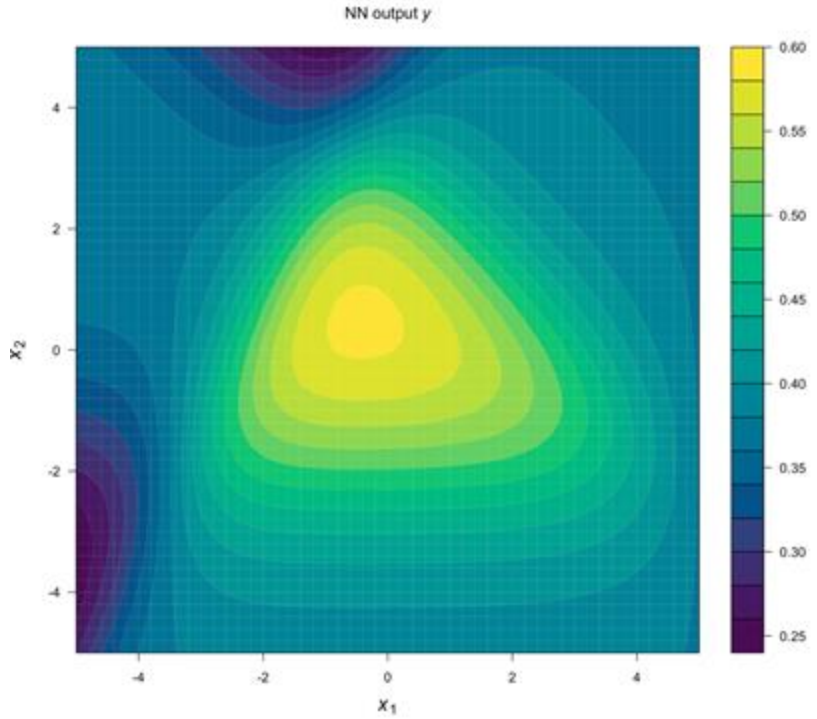
Multi-layer perceptron (MLP)



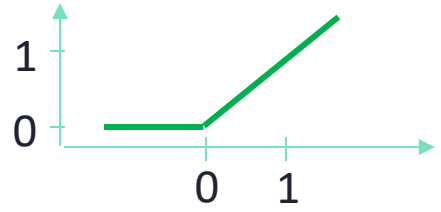
sigmoid tanh



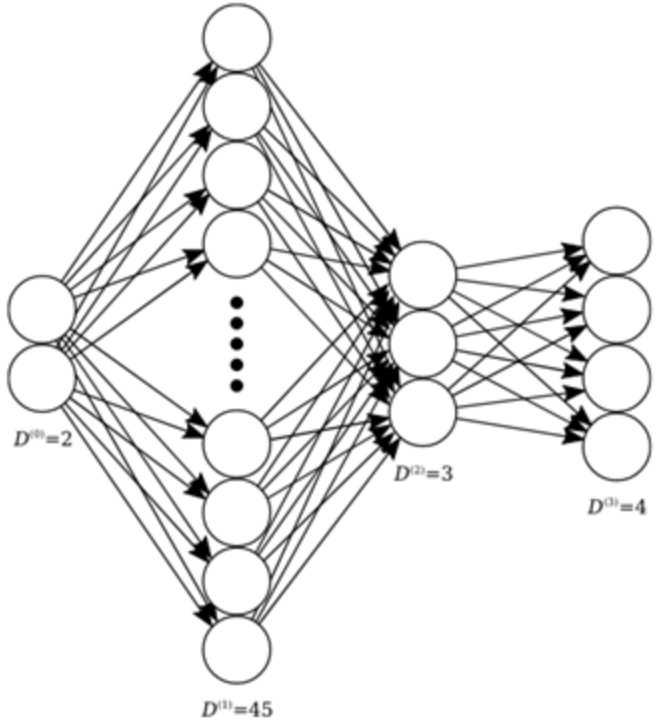


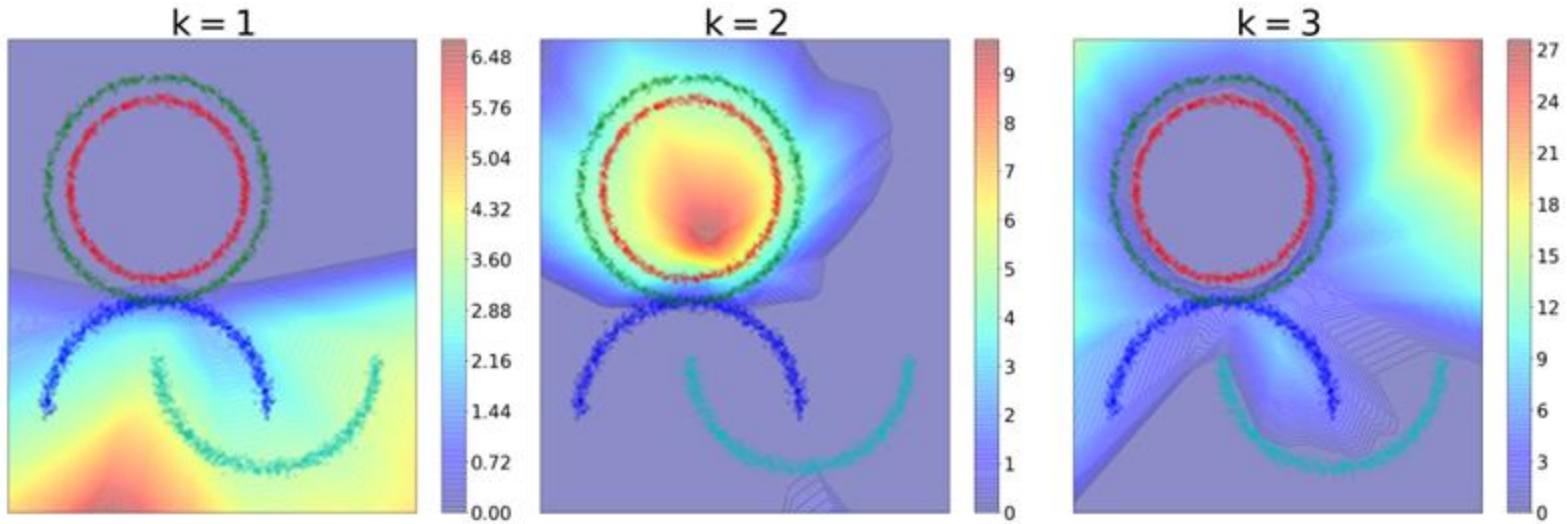


Multi-layer perceptron (MLP)



ReLU

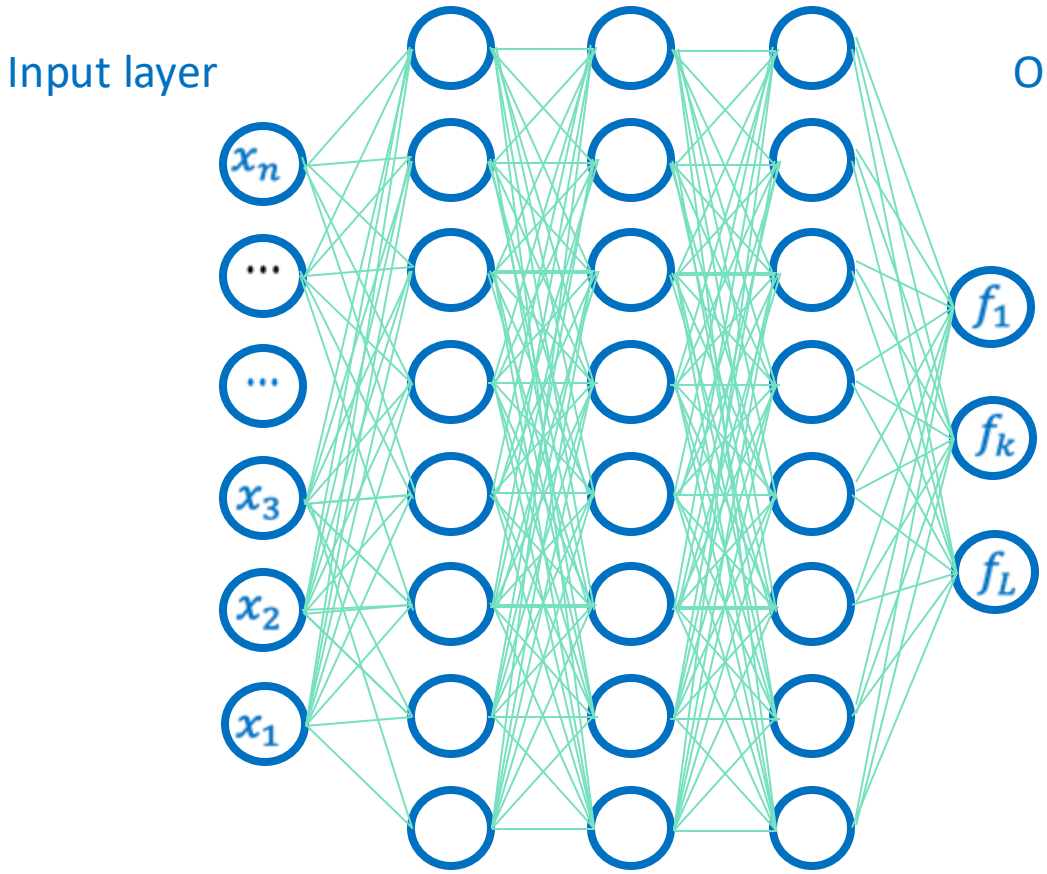




(a) ReLU

[Balestriero & Baraniuk, Proc IEEE, 2021]

MLP or Deep Neural Network (DNN)



$$\boldsymbol{W} = (\boldsymbol{W}^1, \boldsymbol{W}^2, \cdots, \boldsymbol{W}^{N+1})$$

$$\boldsymbol{w}_0 = (\boldsymbol{w}_0^1, \boldsymbol{w}_0^2, \cdots, \boldsymbol{w}_0^{N+1})$$

Feature maps

$$\boldsymbol{z}_1 = g(\boldsymbol{w}_0^1 + \boldsymbol{W}^1 \cdot \boldsymbol{x})$$

$$\boldsymbol{z}_i = g(\boldsymbol{w}_0^i + \boldsymbol{W}^i \cdot \boldsymbol{z}_{i-1})$$

$$f_k = \sigma(w_0^{N+1} + \boldsymbol{W}^{N+1} \cdot \boldsymbol{z}_N)$$

$$f_k = e^{f_k} \Big/ \sum_{i=1}^{L} e^{f_k}$$

Softmax for classification

DNN

- It can be shown mathematically that single-layer NNs are able to approximate any functions in N dimensions if there are enough neurons and one is able to find the correct weights (e.g. Cybenko, Math. Control Signals Syst., 1989, Hornik et al. Neural Networks, 1991)
- However multi-layer NNs can perform this task much more efficiently (much less weights and training time) for hierarchical sparse functions (Mhaskar et al. 2017)
- In a trained multi-layer NN, each layer is able to extract essential information from the previous layer and pass it on to the next layer. This provides a hierarchical decomposition of a big task into a sequel of smaller subtasks.

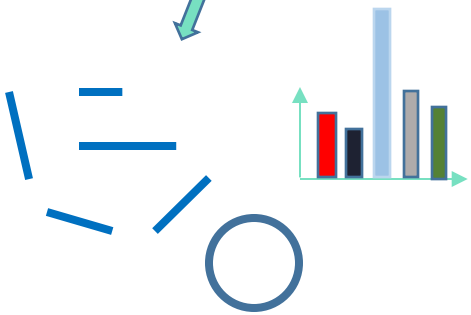
DNN

- DNNs are universal parametric function approximators that can fit any smooth non-linear function f of input vectors x to output vectors $y = f(x)$.
- The weights to achieve this fit can be efficiently learned from the data.
- The 'magic' of DNNs is that they are able to approximate the training data and still generalize well on the test data.

‘Classical’ ML

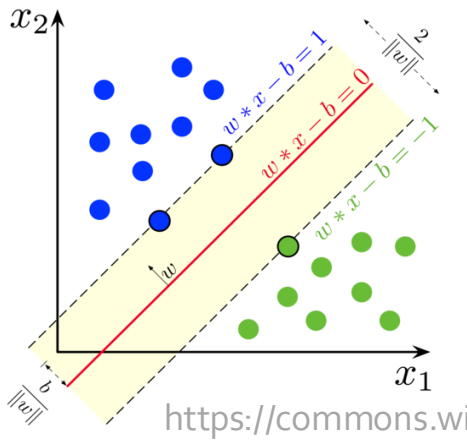
Deep Learning: end-to-end

Engineered features



$$\boldsymbol{x}_i = (x_{i1}, x_{i2}, \quad \cdots \quad x_{in}), i = 1 \ldots N$$

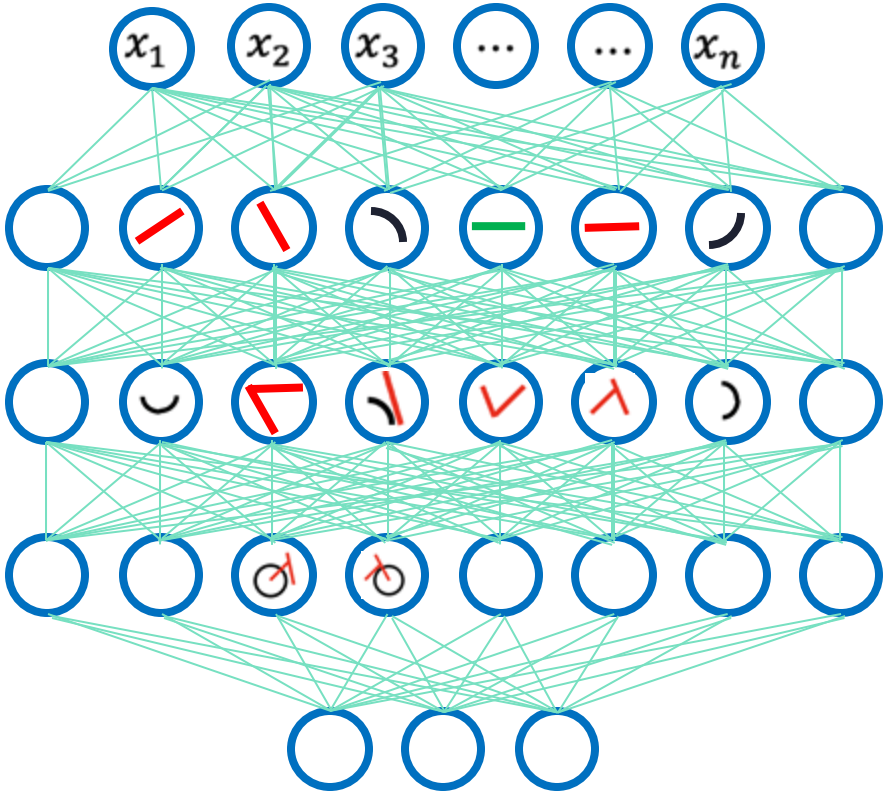
SVM Classifier



Learned features 1

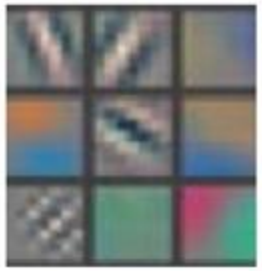
Learned features 2

Learned features 3

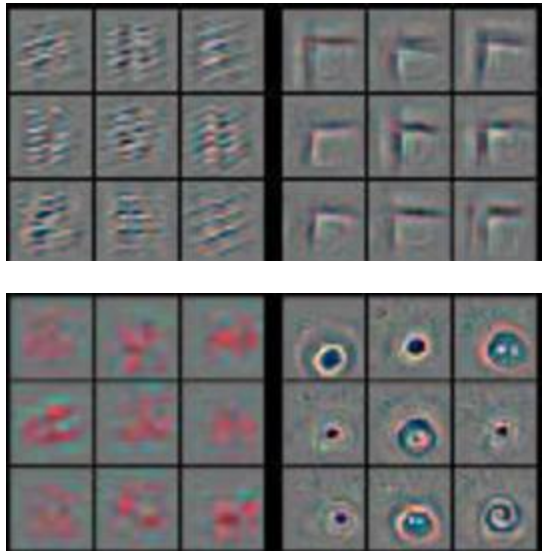


https://commons.wikimedia.org/wiki/File:SVM_margin.png

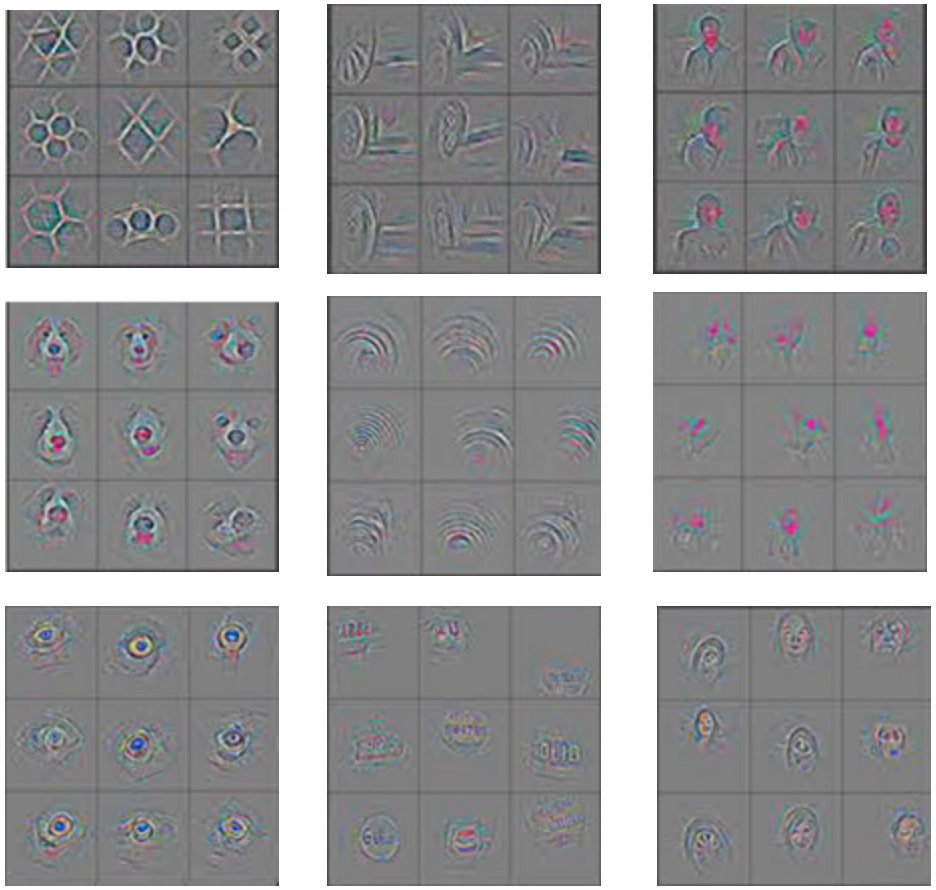
Representations in Convolutional NN



Layer 1



Layer 2



Layer 3

Layer 4

Layer 5

[Zeiler & Fergus, ECCV, 2014]

Top 9 activations by test images of randomly selected feature maps in 8-layer AlexNet projected back to image space by DeconvNet together with validation set image patches that cause the activation.

Outline

- History of deep learning
- Principles of deep learning
- How do deep neural networks learn?
- How do deep neural networks generalize?
- Biological and medical applications

Number of hidden layers and neurons in MLP

- Start with one layer and add layer until overfitting becomes obvious or no more performance gains.
- For MLPs the number of neurons in the hidden layers is usually the same for all layers. Sometimes the first layer is a bit bigger. Keep the number of neurons in the first layer larger than the input size.
- Instead of trying to find the optimal numbers of layers and neurons it is better to choose a network that is a bit too large, and then use regularization during training to shrink it down to the right size (*stretch pants* approach)
- Trainability of DNN depends on network design

Training data size and model complexity

Training data size	Model complexity: High	Model complexity: Low
Large	ideal scenario	underfitting
Small	overfitting risk	low utility

Scaling complexity with data: scale network depth and parameter count to match the data availability

Supervised training of DNN

Loss function J evaluates how close the output of the DNN is to the labels.

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

All weights and offsets from all layers

$$\boldsymbol{W} = \{(\boldsymbol{W}^1, \boldsymbol{W}^2, \cdots, \boldsymbol{W}^{N+1}), (\boldsymbol{w}_0^1, \boldsymbol{w}_0^2, \cdots, \boldsymbol{w}_0^{N+1})\}$$

We try to find the weights $\boldsymbol{W}^*$ that minimize the discrepancy between values predicted by the DNN and true values from a training dataset (loss function J). We use these weights then to predict values for new unlabeled data.

Loss or cost functions

Empirical loss:

$$J(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{W}), y_i)$$

Square loss:
regression

$$\mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{W}), y_i) = (f(\boldsymbol{x}_i; \boldsymbol{W}) - y_i)^2$$

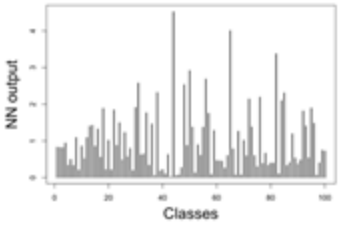
Cross entropy loss:
Classification K classes

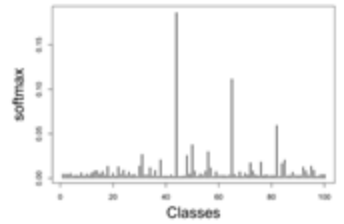
$$\mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{W}), y_i) = -\sum_{k=1}^{K} y_{ik} \log\left(f_k(\boldsymbol{x}_i; \boldsymbol{W})\right)$$

$$y_{ik} = \begin{cases} 1 \; if \;\; y_i = k \\ 0 \; if \;\; y_i \neq k \end{cases}; \;\; \sum_{k=1}^{K} f_k(\boldsymbol{x}_i; \boldsymbol{W}) = 1$$

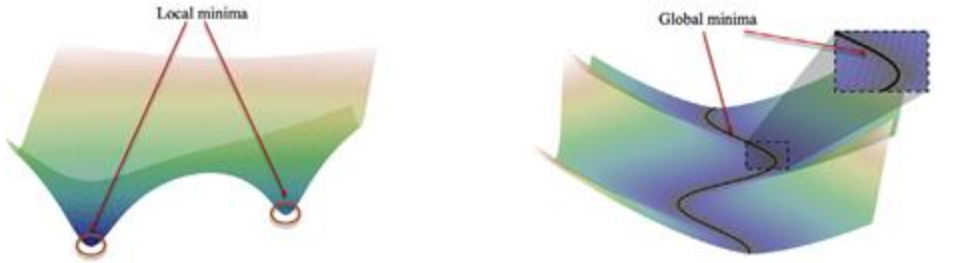
Softmax:

$$f_k = e^{f_k} \Big/ \sum_{i=1}^{L} e^{f_k}$$





Loss landscape



(a) Loss landscape of under-parameterized models (b) Loss landscape of over-parameterized models

Fig. 1. Panel (a): Loss landscape is locally convex at local minima. Panel (b): Loss landscape incompatible with local convexity as the set of global minima is not locally linear.

- Large number of parameters must not pose a problem
- In large DNN the vast majority of points with vanishing gradients are saddle points (Pascanu et al, arXiv, 2014)
- The smoother and convex the loss surface, the better the performance on test data (Li et al, NIPS, 2018)
- In DNNs with n parameters and d data points ($n > d$) the global minimum is usually a $n - d$ dimensional non-convex submanifold of $\mathbb{R}^n$ (Cooper, arXiv, 2018).
- SGD converges to global minima solutions with large margins (Zhang et al. Commun. ACM, 2021, Liu et al. Appl. Comp. Harmon. Anal., 2022)

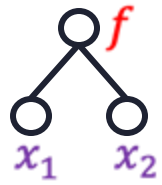
Gradient descent (GD)

1. Initialize weights randomly $\sim\mathcal{N}\left(0,\sigma_i^2\right)$, $epoche = 1$
2. While $epoche < N_e$ and $stop \neq True$
   i. Compute $J(\boldsymbol{W})$, $\frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$ by backpropagation
   ii. Gradient descent $\boldsymbol{W} \rightarrow \boldsymbol{W} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$
   iii. $epoche{+}{+}$
3. Return $\boldsymbol{W}$

AutoDiff/Backpropagation

$$\frac{\partial J(u(x))}{\partial x} = \frac{\partial J(u)}{\partial u} \cdot \frac{\partial u(x)}{\partial x}$$

Chain rule



$J = (f - y)^2$

$\frac{\partial J}{\partial f} = 2(f - y)$

$f = g(z)$

$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial f}\frac{\partial f}{\partial z} = 2(f - y) \cdot g'(z)$

$z = z_1 + z_2 + w_0$

$\frac{\partial J}{\partial w_0} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial w_0} = 2(f - y) \cdot g'(z)$

$z_1 = w_1 \times x_1$

$\frac{\partial J}{\partial z_1} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial z_1} = 2(f - y) \cdot g'(z)$

$z_2 = w_2 \times x_2$

$\frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial z_2} = 2(f - y) \cdot g'(z)$

w_1

x_1

$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z_1}\frac{\partial z_1}{\partial w_1} = 2(f - y) \cdot g'(z) \cdot x_1$

w_2

x_2

$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial w_2} = 2(f - y) \cdot g'(z) \cdot x_2$

DNN can be difficult to train with GD

- Vanishing gradients problem:
  - Some gradients disappear during backpropagation and learning stops (Hochreiter et al., 2001, Glorot & Bengio, PMLR, 2010)

- Exploding gradient problem:
  - Some gradients grow large during training and learning becomes unstable. (Hochreiter et al., 2001, Pascanu et al arXiv, 2012).

How can we better train DNNs?

Better weight initialization, better gradient descent, regularization, more efficient code, and more GPU's

Momentum gradient descent

Gradient descent:

1. $\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$

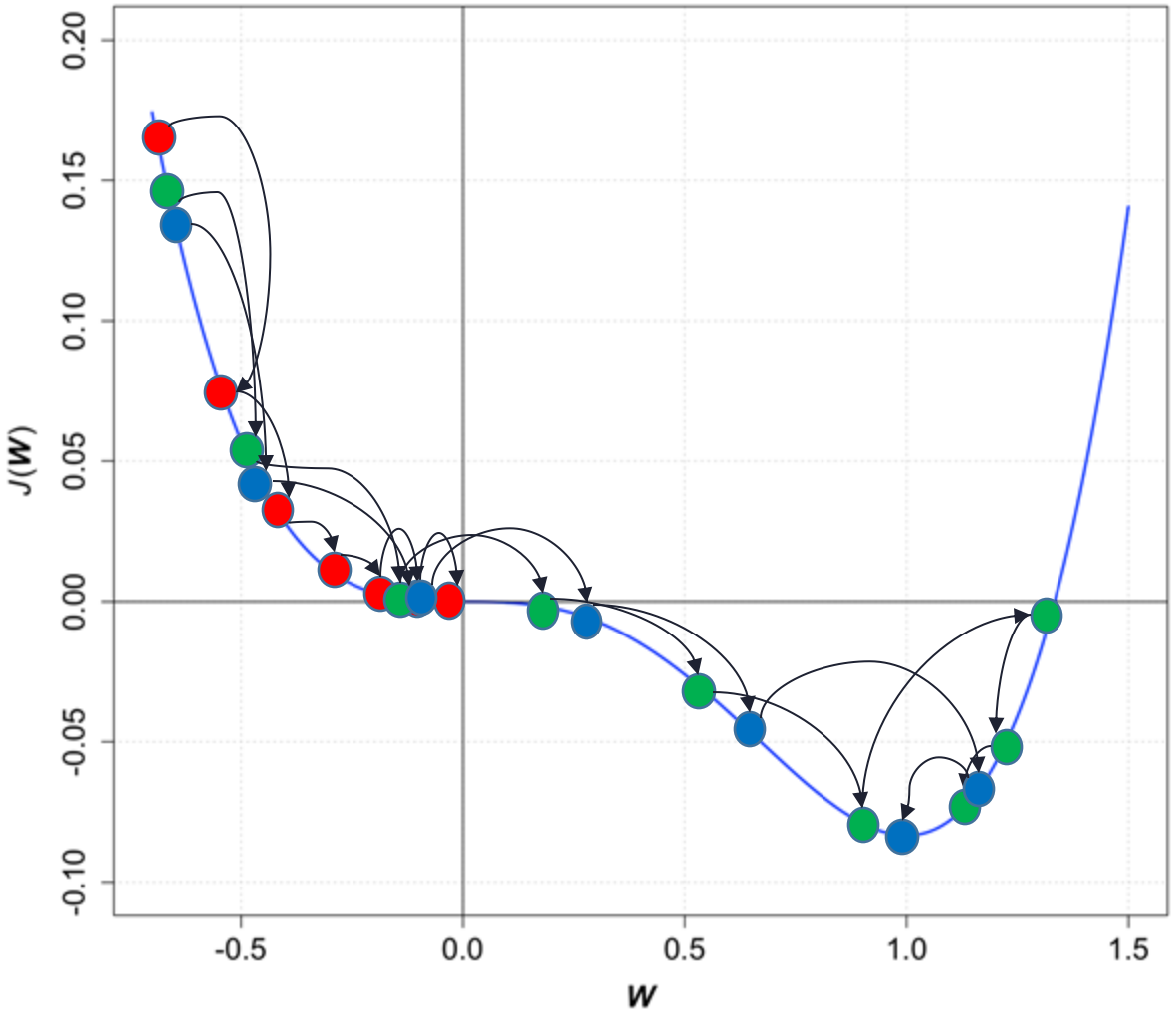
Momentum gradient descent:

1. $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$
2. $\boldsymbol{W} \leftarrow \boldsymbol{W} + \boldsymbol{m}$

Nesterov gradient descent:

1. $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W} + \beta \boldsymbol{m})$
2. $\boldsymbol{W} \leftarrow \boldsymbol{W} + \boldsymbol{m}$

Further variants: AdaGrad, RMSProb, Adam, Nadam, …



Stochastic gradient descent (SGD)

$J(\boldsymbol{W})$ can be slow to calculate for large datasets. Mini-batches can be run in parallel on GPU's, which accelarates learning considerably. They also add randomness to the gradient descent, which allows to it wiggle out of bad solutions. Small batches (32, 64, 128) often lead to better models.

1. Initialize weights randomly $\sim \mathcal{N}\left(0, \sigma_i^2\right)$, $epoche = 1$
2. While $epoch < N_e$ and $stop \neq True$

   While $\mathcal{B} = \{\boldsymbol{B}_i\} \neq \emptyset$

   i. Pick batch $\boldsymbol{B}_i$ of size $|\boldsymbol{B}|$ randomly ($|\boldsymbol{B}|$ = 32, 64, 128, ..,8192)

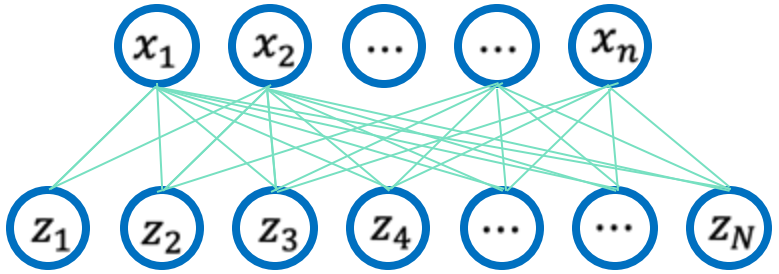
   ii. Compute $J(\boldsymbol{W}) = \frac{1}{|\boldsymbol{B}|}\sum_{j \in \boldsymbol{B}_i} \mathcal{L}\left(f\left(\boldsymbol{x}_j; \boldsymbol{W}\right), y_j\right)$

   iii. Gradient descent $\boldsymbol{W} \rightarrow \boldsymbol{W} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$

   iv. $\mathcal{B} = \mathcal{B} \backslash \boldsymbol{B}_i$

3. Return $\boldsymbol{W}$

Weight initialization

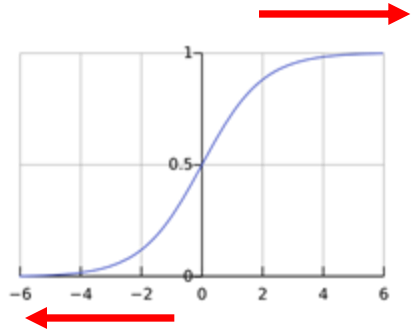


$$x_i, w_i \sim \mathcal{N}(0,1)$$

$$z_j' = \sum_{i=1}^{n} w_i \cdot x_i, \; z_j = g(z_j')$$

$$\sigma(z_j') = \sqrt{n}\,\sigma(w_i)\sigma(x_i) = \sqrt{n}$$

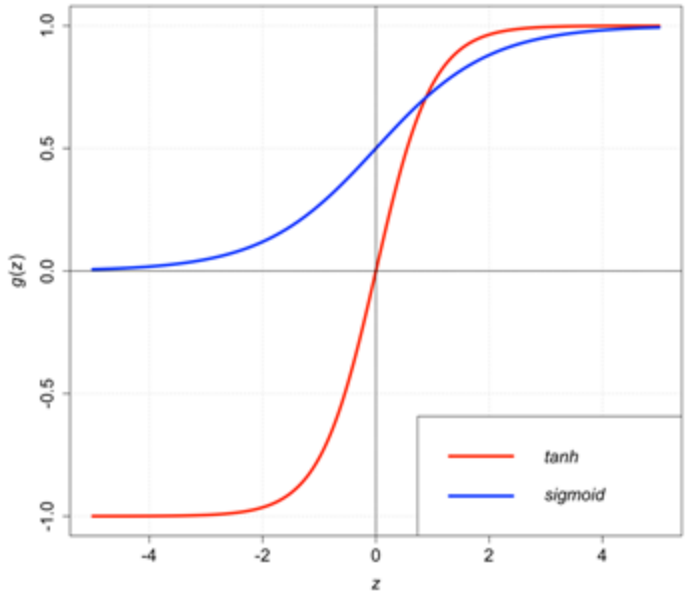
Vanishing gradients



Solution

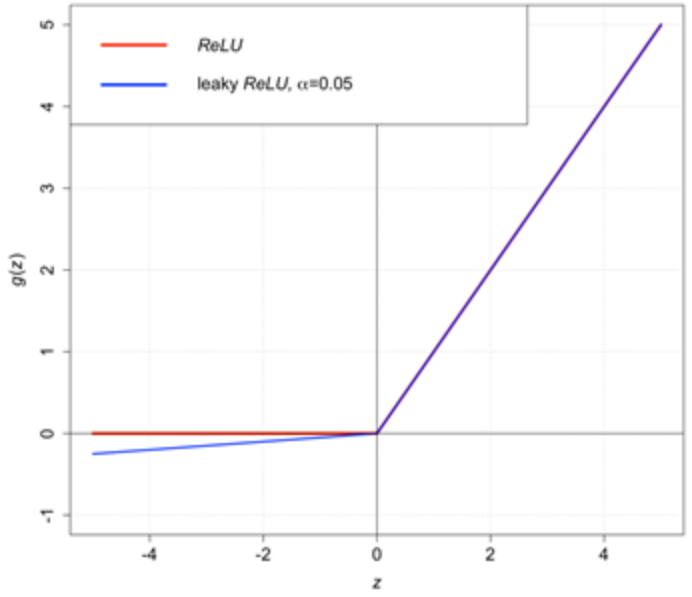
Publication	Weight initialization	Activation function
Glorot & Bengio, PMLR, 2010	$w_i \sim \mathcal{N}(0, {}^{2}/_{n+N})$	logistic, tanh
He et al., ICCV, 2015		ReLU
LeCun	$w_i \sim \mathcal{N}(0, {}^{1}/_{n})$	SELU

Activation functions



+ Smooth
+ tanh: ~ zero centered outputs
- Slow to calculate
- Sigmoid has only positive outputs
- Vanishing gradients

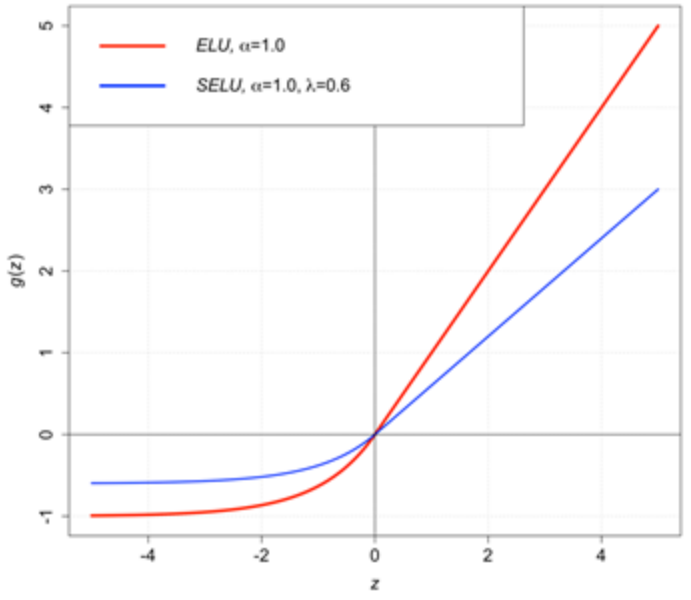
Rectified LU



+ Fast
+ Positive gradient
+ Sparse outputs
+ Piecewise linear spline
- Kink at 0 leads to jump in derivative
- Positive unbounded outputs
- Invariance to scaling

ReLU: Glorot et al, 2011
leaky ReLU: Xu et al, 2015
Spline: Poggio, CBMM, 2015
Strang SIAM News, 2018,
Belstrino & Baraniuk, ICML, 2018

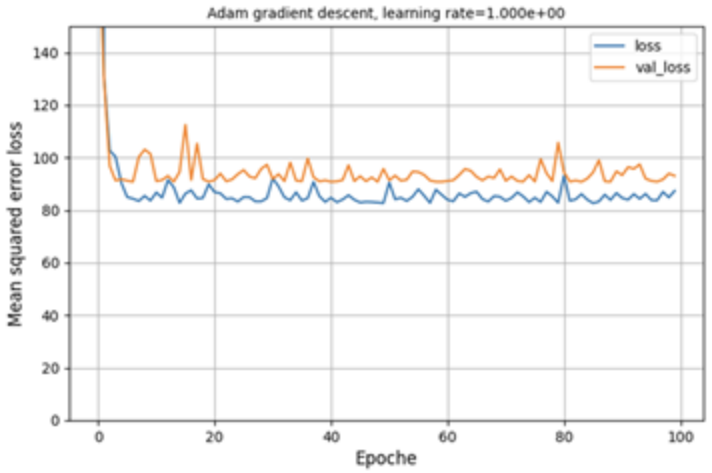
Exponential LU

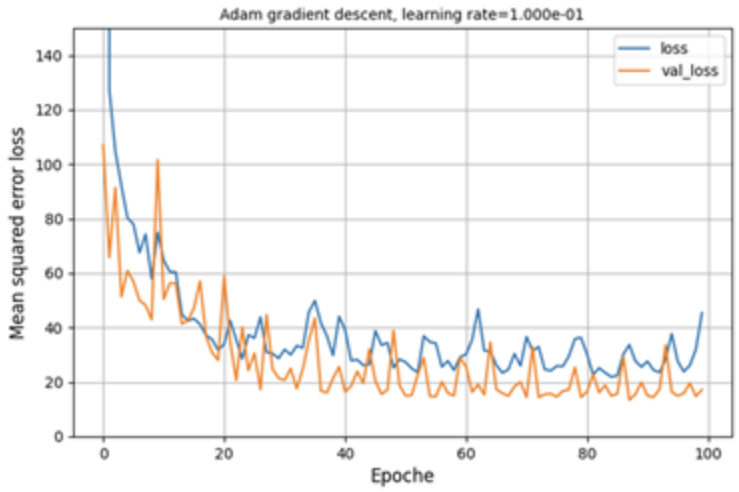


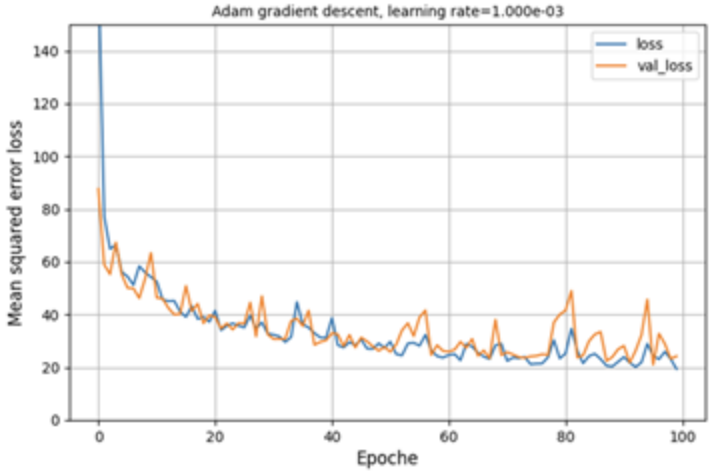
+ Smooth positive gradient
+ ~ Zero centered outputs
+ Adaptively scaled ELU (SELU) leads to normalized outputs $z_i \sim \mathcal{N}(0,1)$
- Slow to calculate

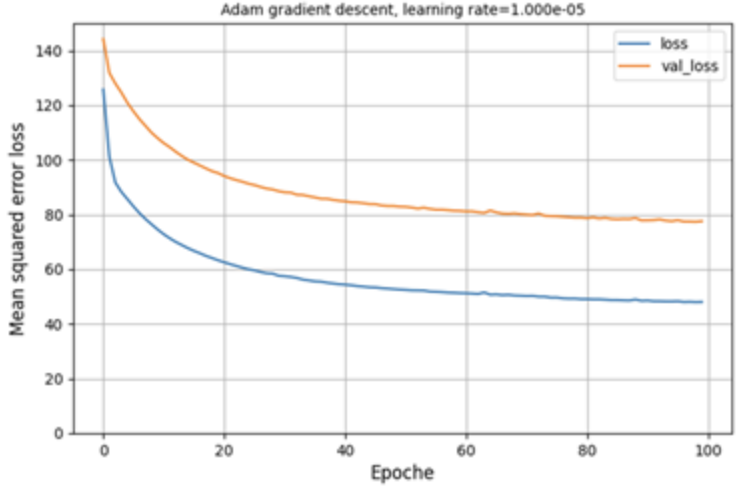
ELU: Clevert et al, 2016
SELU: Klambauer et al, 2017

Learning rate





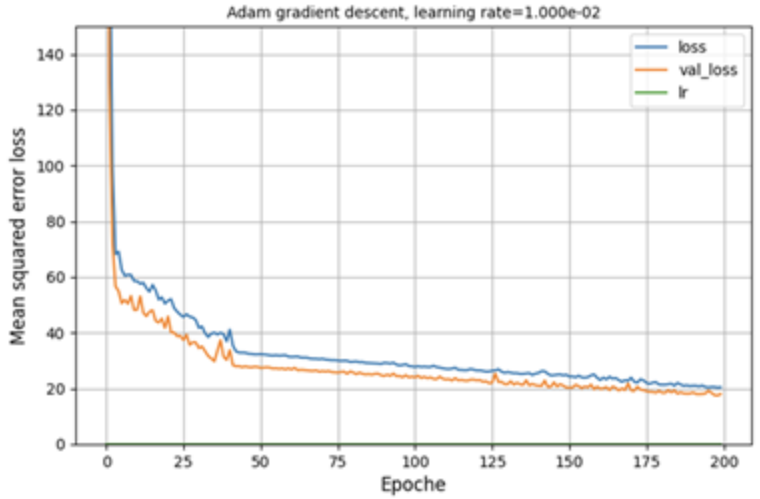




Learning rate γ:

- Way too high: converge early to non-optimal solution
- Too high: converges to good solution but noisy
- About right: fast stable convergence to good solution
- Too low: slow convergence. May not reach good solution

Learning rate schedule



- Power scheduling: $\gamma(t) = {\gamma_0}/{(1+t/s)^c}$
- Exponential scheduling: $\gamma(t) = {\gamma_0}/{10^{t/s}}$
- Piecewise scheduling: e.g $\gamma(t) = \begin{cases} 0.01 \; if \; epoch \; \leq 40 \\ 0.001 \; if \; epoch > 40 \end{cases}$
- Adaptive scheduling: reduce γ by a factor if error stops dropping
- *1cycle* scheduling: first linearly increase learning rate to maximum and then linearly drop to minimum. Adapt momentum.

- Senior et al. ICASSP, 2013, Smith arXiv, 2018

Regularization

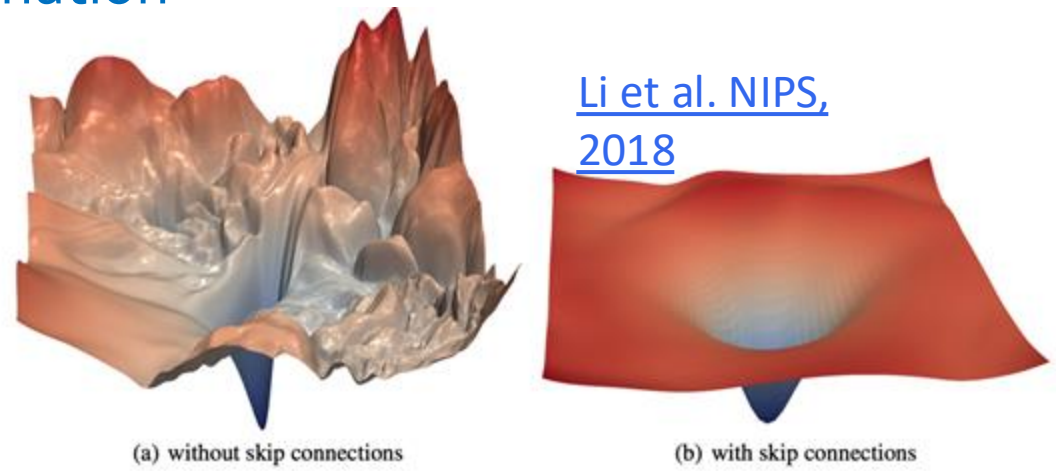
- SGD: leads to small weights (Zhang et al. Commun. ACM, 2021)
- ℓ_1 and ℓ_2 regularization or weight decay: for a layer k you can add ℓ_1 and ℓ_2 regularization, which adds $C_1 \sum_{ij} |w_{ij}^k|$ for ℓ_1 or $C_2 \sum_{ij} (w_{ij}^k)^2$ for ℓ_2 to the loss function. This forces the weights w_{ij}^k to remain small.
- Weight constraints: adjust the norm of the weights in a layer or force the weights to be in a certain range
- Gradient clipping: keep the norm or components of the gradient within a certain range

Regularization

- Skip connections: the input can be added to the output of a set of layers if the output has the same dimension and is similar to the input. This refreshes the output, kickstarts learning and can avoid vanishing gradients.

  Addition or Concatenation

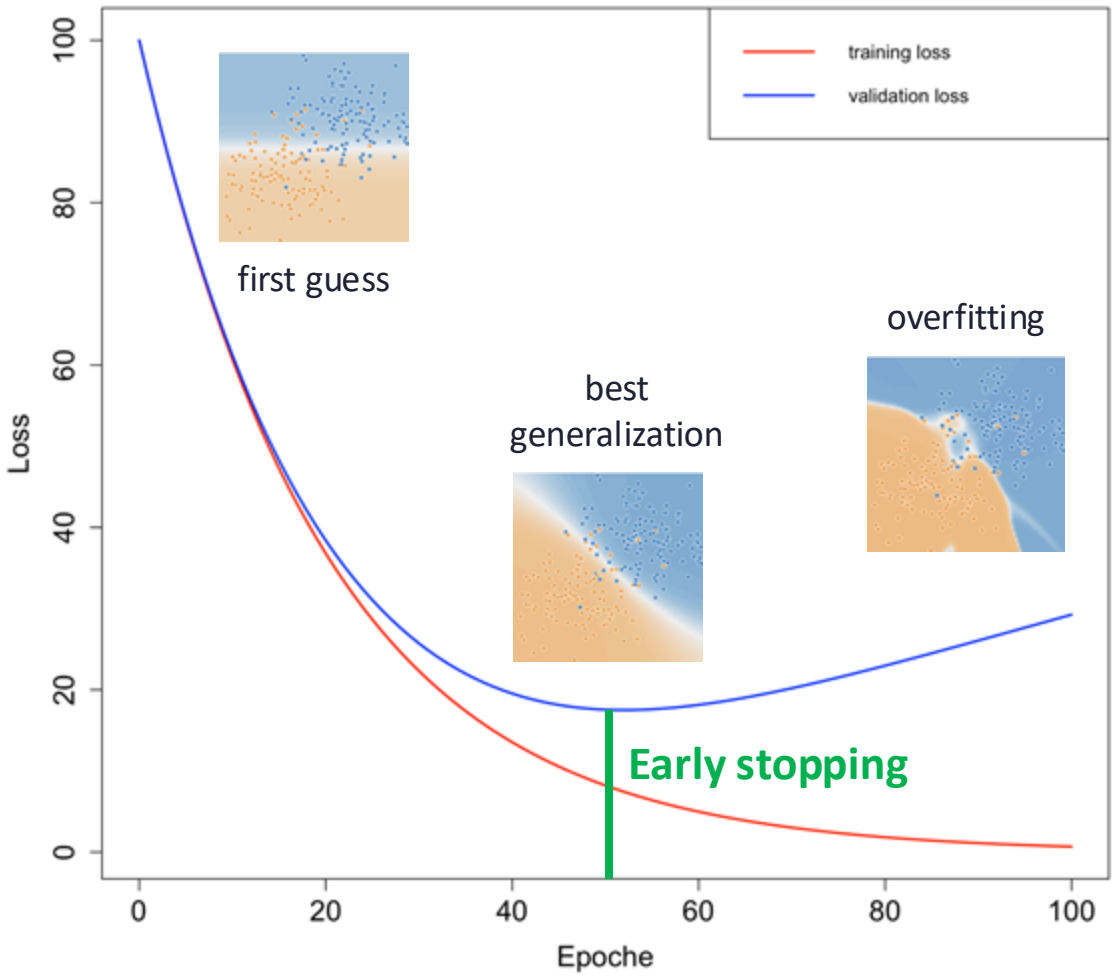
- Transfer learning or pretraining: initialize weights with good starting values from a network that was trained with lots of data for a similar task.
- Data augmentation: for images, flip rotate, shift, zoom and change lighting to create new training data.



(a) without skip connections (b) with skip connections

Li et al. NIPS, 2018

Regularization: early stopping



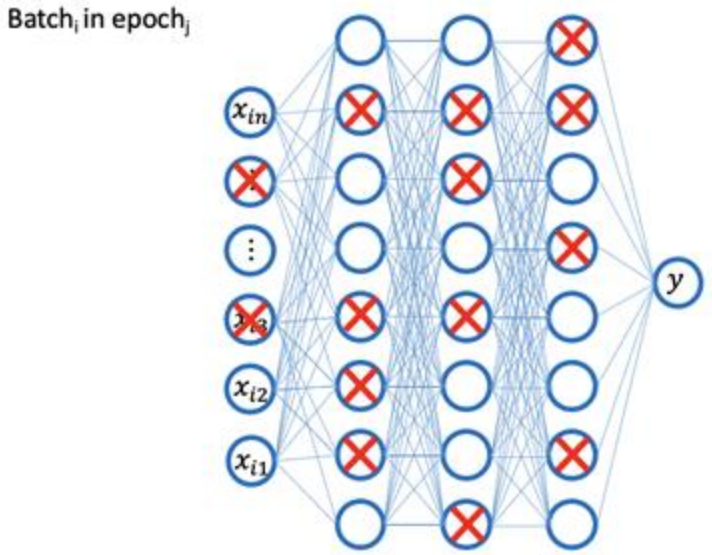
Training set loss: minimized by gradient descent
Validation set loss: used to control overfitting and for early stopping

Early stopping: if the validation loss does not decrease anymore for an number of epoches, stop the training. Keep the weights that gave the lowest validation loss.

Regularization: dropout

- At the beginning of each batch, every neuron (including input, expluding output) has a probablility $p \sim 0.1 - 0.5$ of being dropped out, i.e. its weights are set to 0.
- Each batch trains a different network and the final weights represent an average of all these different networks.
- This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors.
- This makes the network more robust and resilient and usually adds a significant performance improvement for large networks.
- If the model is overfitting, increase the dropout rate.
- Dropout can also be activated during prediction to obtain MC estimates of the predicted output.

- Hinton et al. arXiv, 2012, Srivastava et al., JMLR, 2014, Gal & Ghahramani, PMLR, 2016



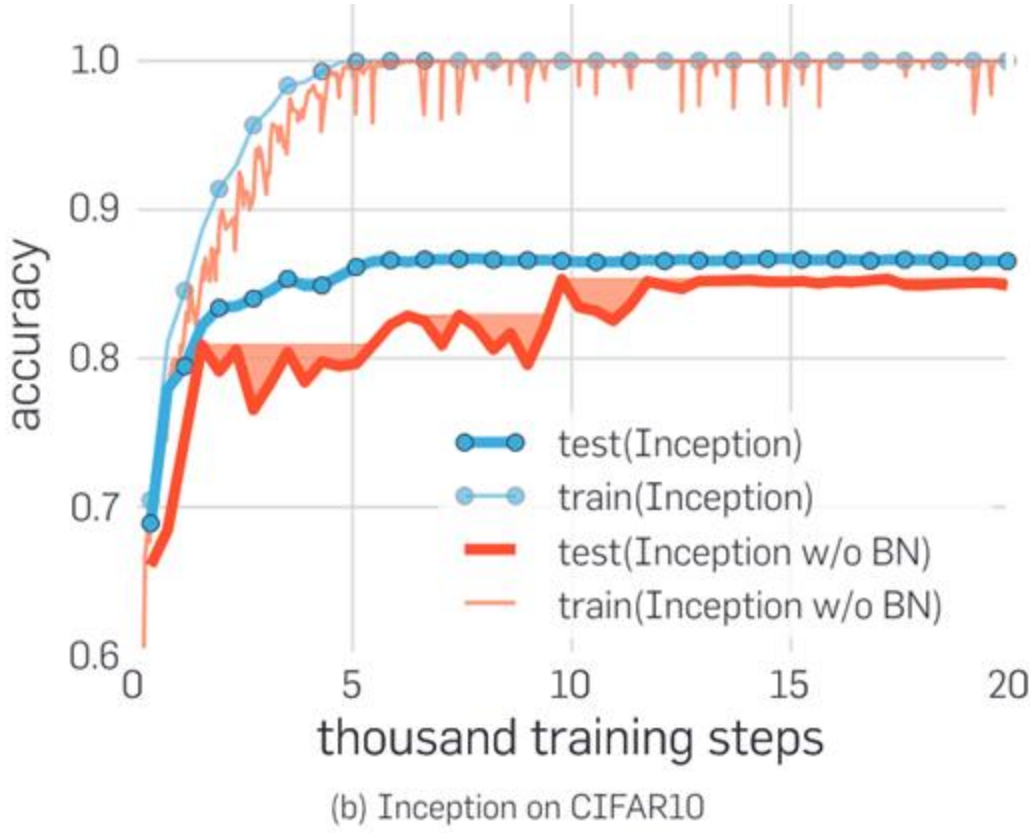
Regularization: batch normalization

- Batch normalization first standardizes the batch inputs to a layer k and then learns optimal scales $\boldsymbol{\alpha}_k$ and shifts $\boldsymbol{\beta}_k$ during training.
  - $\boldsymbol{z}_i$: i-th input of batch B to layer k, where batch normalization is requested
  - $\boldsymbol{\mu}_B = \frac{1}{|B|}\sum_{i=1}^{|B|} \boldsymbol{z}_i \; ; \sigma_B^2 = \frac{1}{|B|}\sum_{i=1}^{|B|}(\boldsymbol{z}_i - \boldsymbol{\mu}_B)^2 \; ; \bar{\boldsymbol{z}}_i = \frac{\boldsymbol{z}_i - \boldsymbol{\mu}_B}{\sqrt{\sigma_B^2 + \varepsilon}}$
  - $\bar{\bar{\boldsymbol{z}}}_i = \boldsymbol{\alpha}_k \otimes \bar{\boldsymbol{z}}_i + \boldsymbol{\beta}_k$
  - Final $\bar{\boldsymbol{\mu}}$ and $\overline{\sigma^2}$ obtained by averaging over all batches are used for prediction.

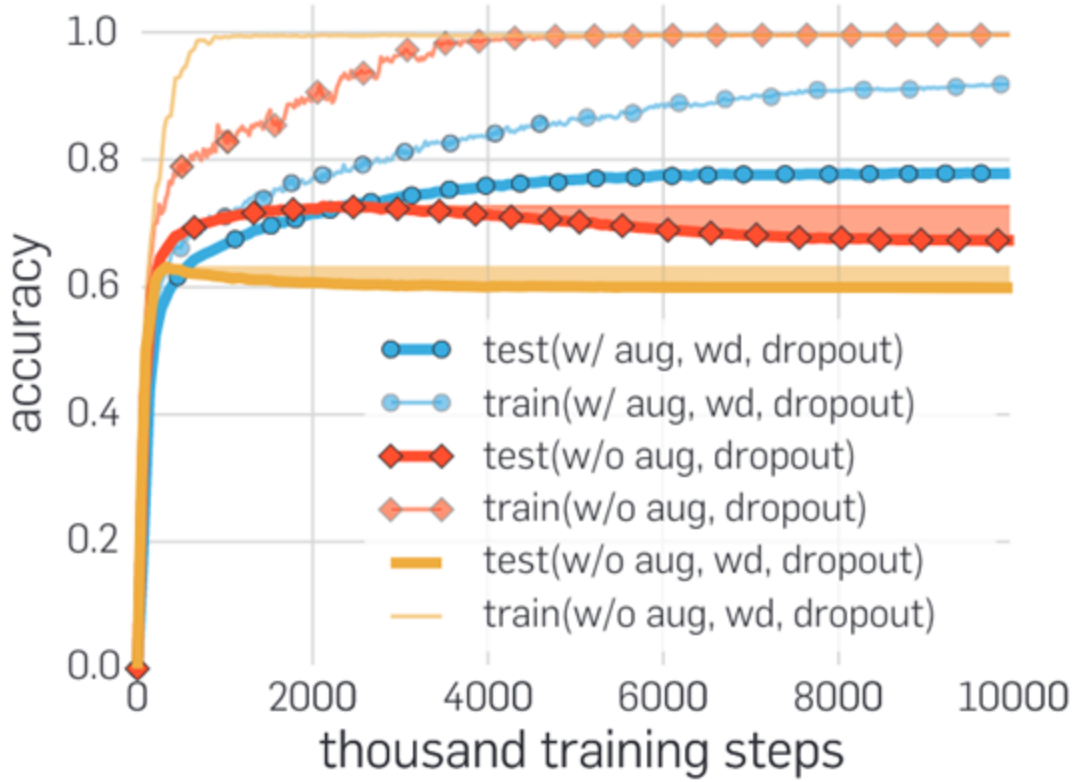
- Joffe & Szegedy, PMLR, 2015

Regularization and generalisation

Inception v3. ConvNet 13 layers, 1 ,649,402 parameters



(b) Inception on CIFAR10



(a) Inception on ImageNet

[Zhang et al. CACM, 2021]

The art of DL: chose the right architecture and hyperparameters

Table 2. Central parameters of a neural network and recommended settings

Name	Range	Default value
Learning rate	0.1, 0.01, 0.001, 0.0001	0.01
Batch size	64, 128, 256	128
Momentum rate	0.8, 0.9, 0.95	0.9
Weight initialization	Normal, Uniform, Glorot uniform	Glorot uniform
Per-parameter adaptive learning rate methods	RMSprop, Adagrad, Adadelta, Adam	Adam
Batch normalization	Yes, no	Yes
Learning rate decay	None, linear, exponential	Linear (rate 0.5)
Activation function	Sigmoid, Tanh, ReLU, Softmax	ReLU
Dropout rate	0.1, 0.25, 0.5, 0.75	0.5
L1, L2 regularization	0, 0.01, 0.001	

Angermueller et al. Mol. Sys. Biol. 2016

Good parameters can be found through trial and error. You can test different values for each parameter individually and find which values that work well. The most important parameters are the network size, the learning rate.

You can try to find good combinations of these parameters more automatically via a parameter optimization framework such as sklearn GridSearchCV or RandomSearchCV. More advanced tuners include Hyperband, Hyperopt, AutoML or Sklearn-Deep.

Outline

- History of deep learning
- Principles of deep learning
- How do deep neural networks learn?
- How do deep neural networks generalize?
- Biological and medical applications

Generalization of DNNs

- Why strongly over parametrized DNNs do not overfit and generalize well on test data is still not fully understood
- Regularization methods improve generalization performance but are not necessary to make generalization work (Zhang et al. CACM, 2021).
- (S)GD and early stopping find solutions that generalize well (large margin for separable classes) (Zhang et al. CACM, 2021, Poggio et al., arXiv, 2018).
- Do DNNs just interpolate the training data, or do they produce a coherent model reflecting the data generation process?

The magic of DNN: breaking the variance barrier

Belkin et al. PNAS, 2019

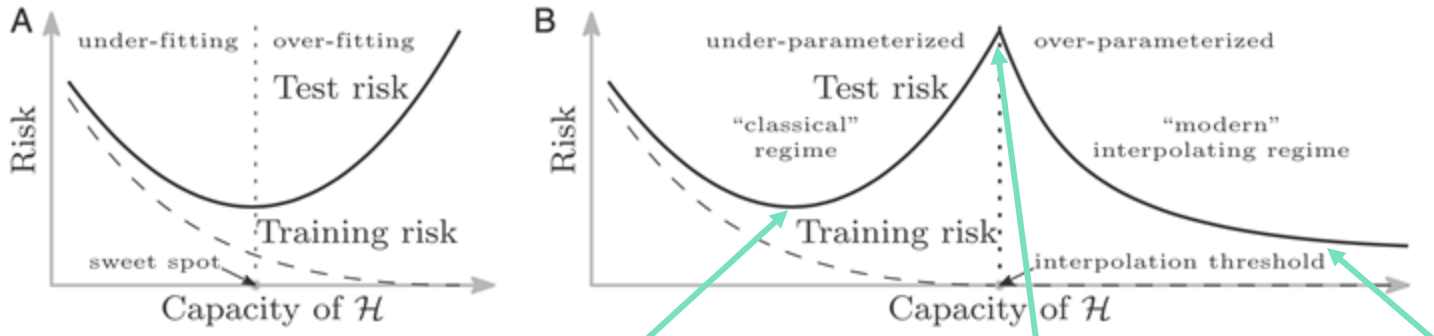
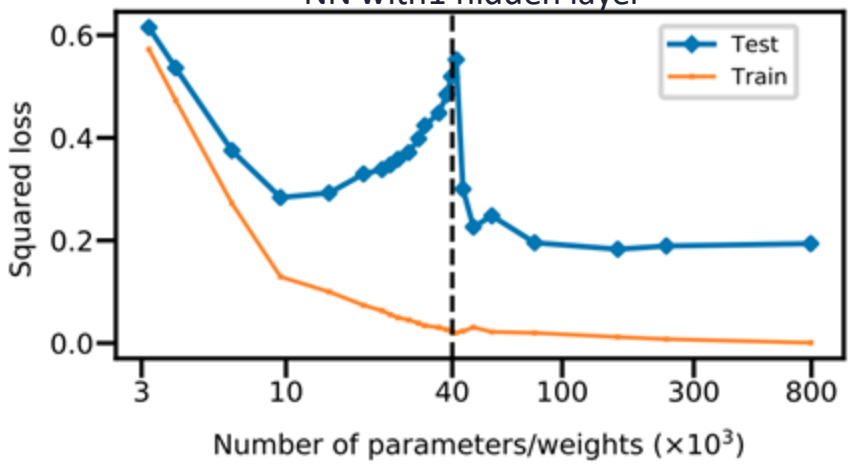
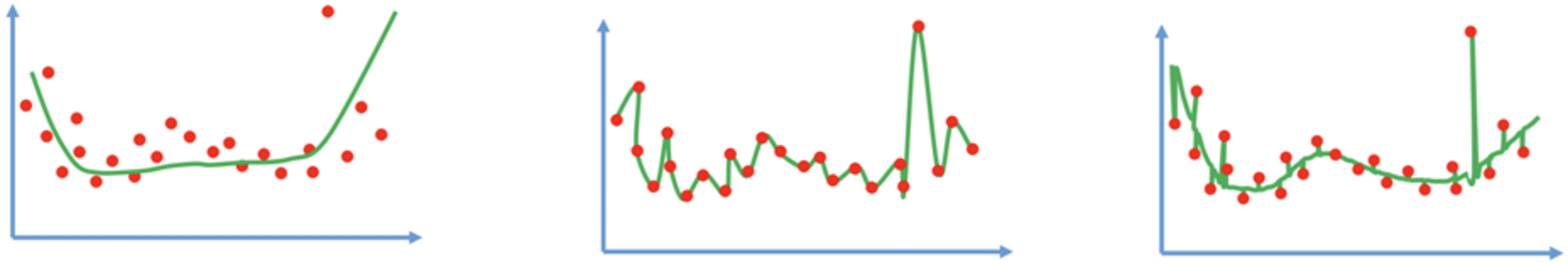


Fig. 1. Curves for training risk (dashed line) and test risk (solid line). (*A*) The classical U-shaped risk curve arising from the bias–variance trade-off. (*B*) The double-descent risk curve, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high-capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

MNIST ($n = 4 \cdot 103$, $d = 784$, K=10 classes)
NN with1 hidden layer





Generalization of CNNs for corrupted images

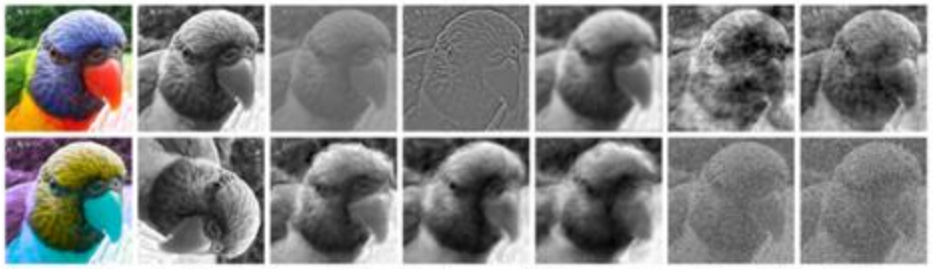
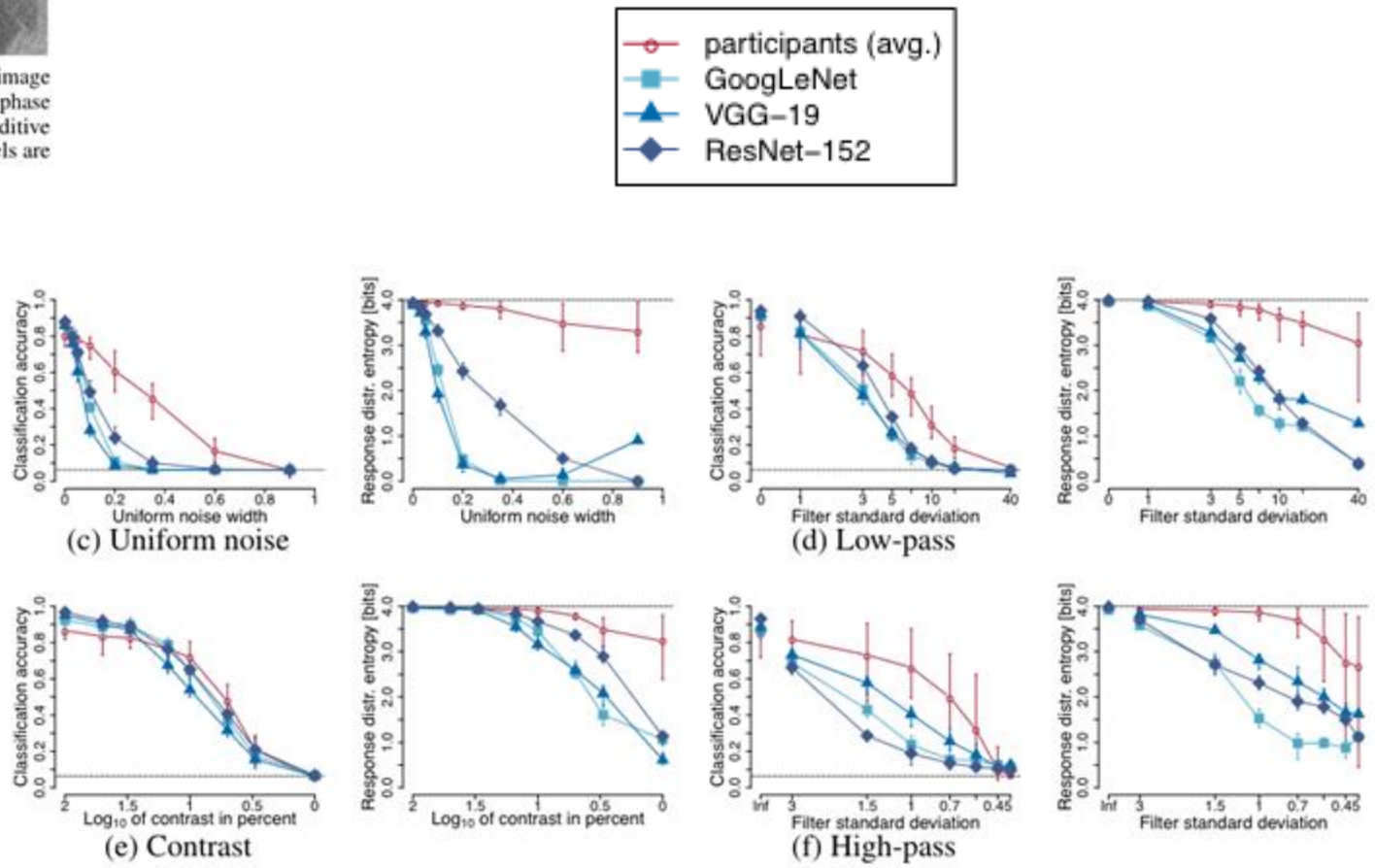


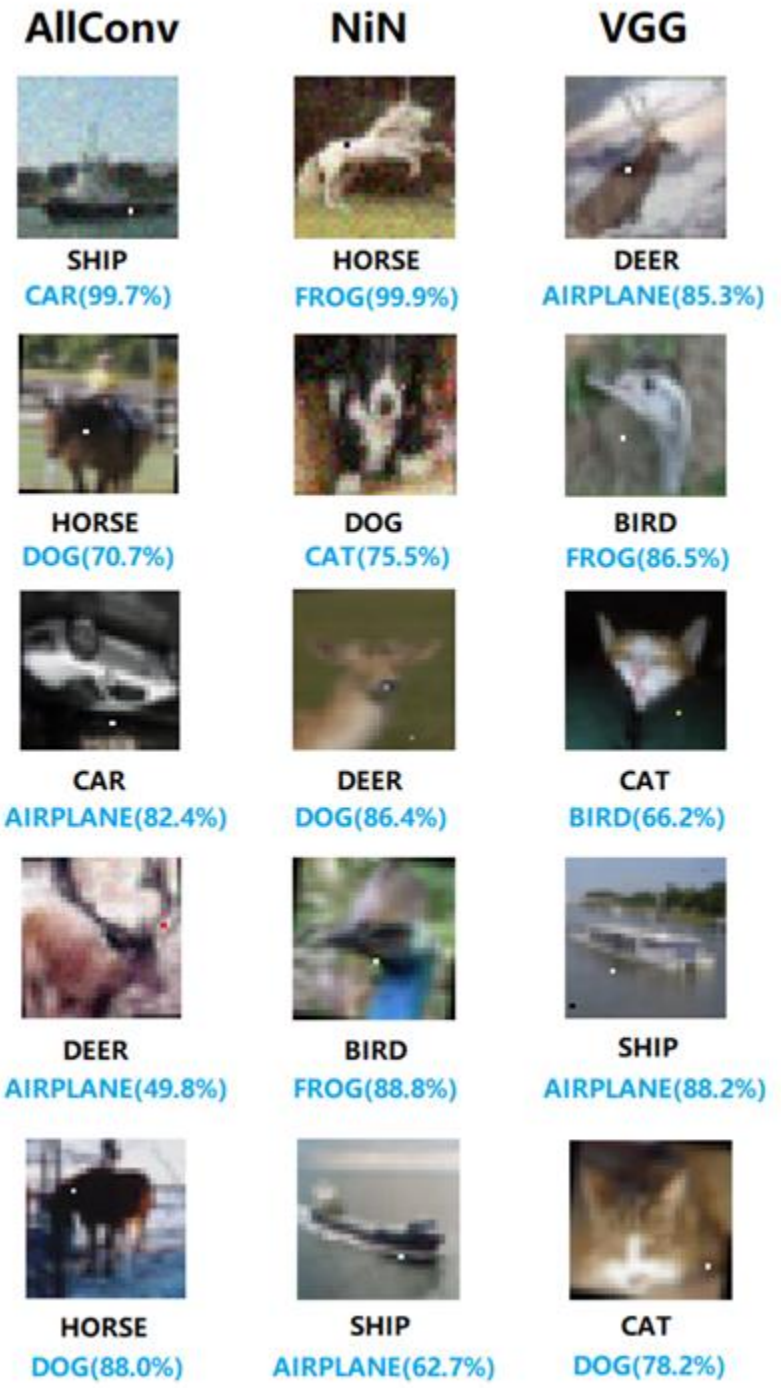
Figure 2: Example stimulus image of class `bird` across all distortion types. From left to right, image manipulations are: colour (undistorted), greyscale, low contrast, high-pass, low-pass (blurring), phase noise, power equalisation. Bottom row: opponent colour, rotation, Eidolon I, II and III, additive uniform noise, salt-and-pepper noise. Example stimulus images across all used distortion levels are available in the supplementary material.



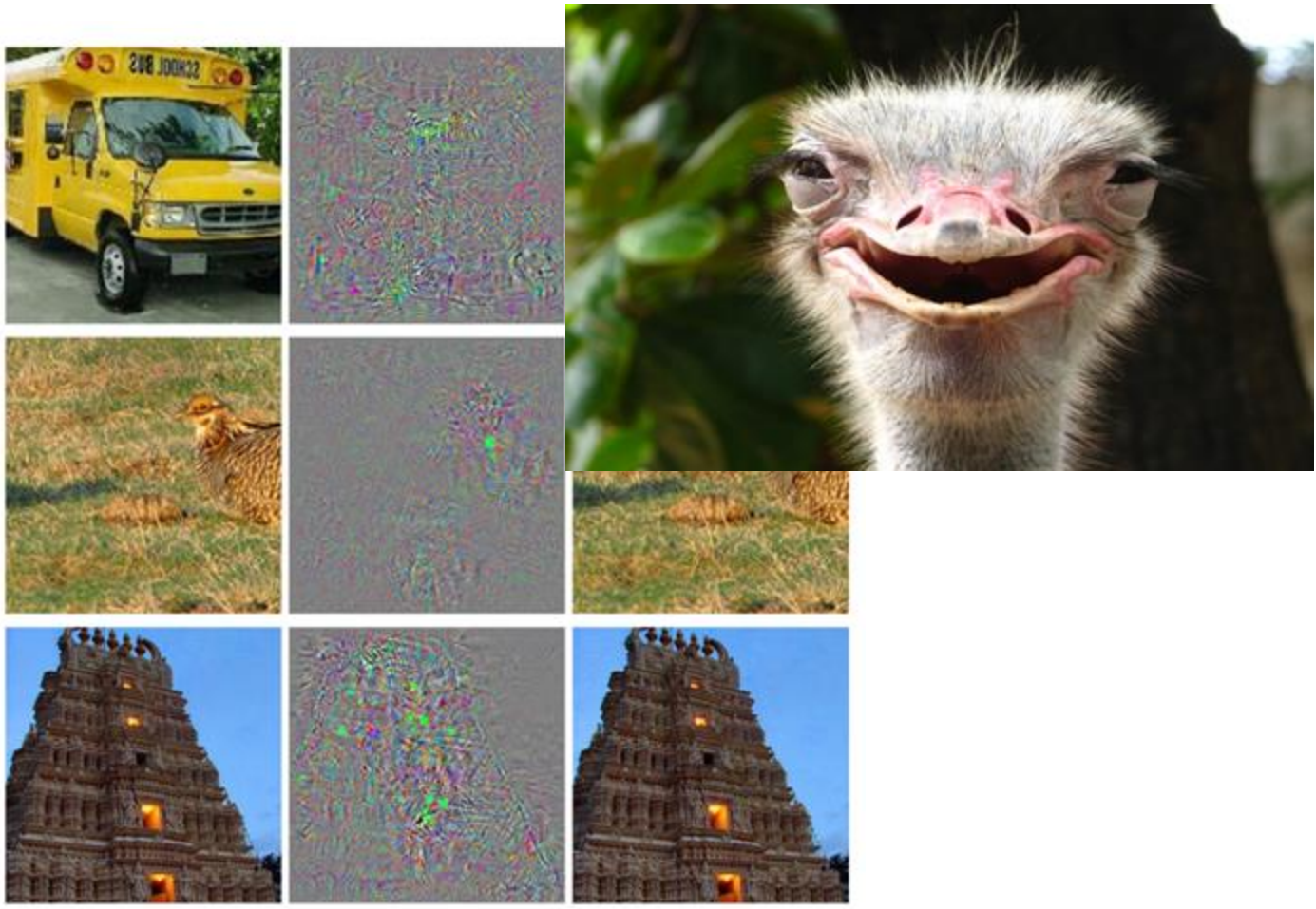
[Geirhos et al, NeurIPS 2018]

Is there a free lunch?

Robustness to adversarial attacks



Su et al, IEEE TEVC, 2019

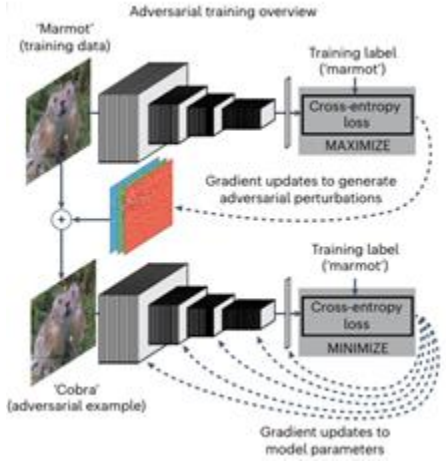


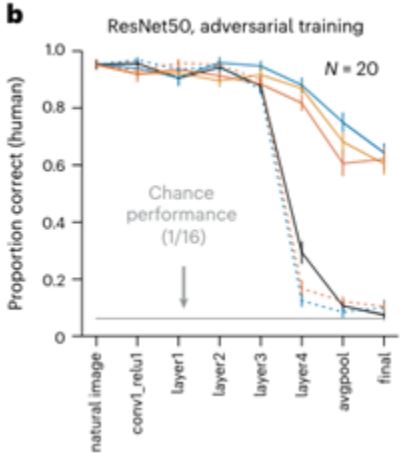
Szegedy et al., arXiv, 2013

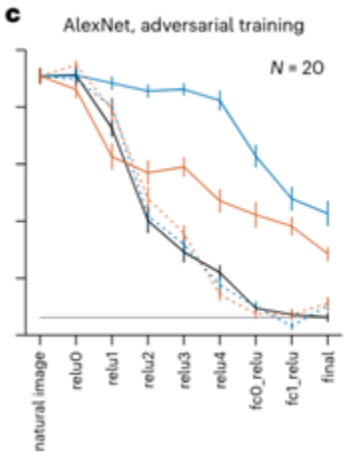
Similar for CNN is not similar for humans

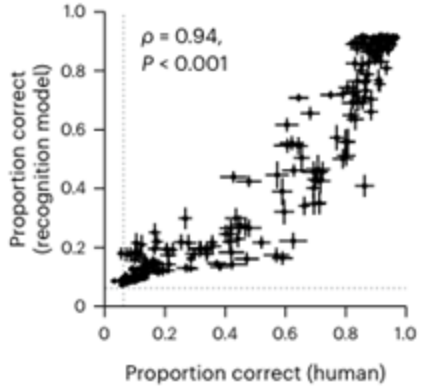
Feather et al. Nat. Neuroscience, 2023

Adversarial learning helps:









How to counter adversarial attacks

- Augment your training set with adversarial examples
- Adversarial examples are not robust: e.g. a slight change in the adversarial input will change their classification.
- Methods that estimate the uncertainty in the NN model prediction can detect adversarial attacks.
- Two approaches for uncertainty prediction:
  - Bayesian NN (dropout, data sampling)
    Gal & Ghahramani, PMLR, 2016 , Lakshminarayanan et al., NIPS, 2017
  - Evidential DL
    Sensoy et al., NIPS, 2018 (classification), Amini et al., NIPS, 2020 (regression)

Evidential deep learning (EDL) for classification

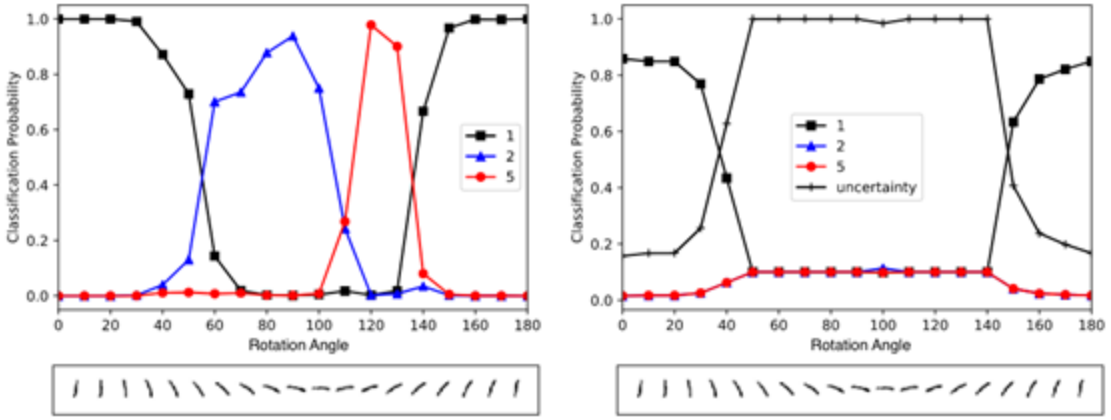
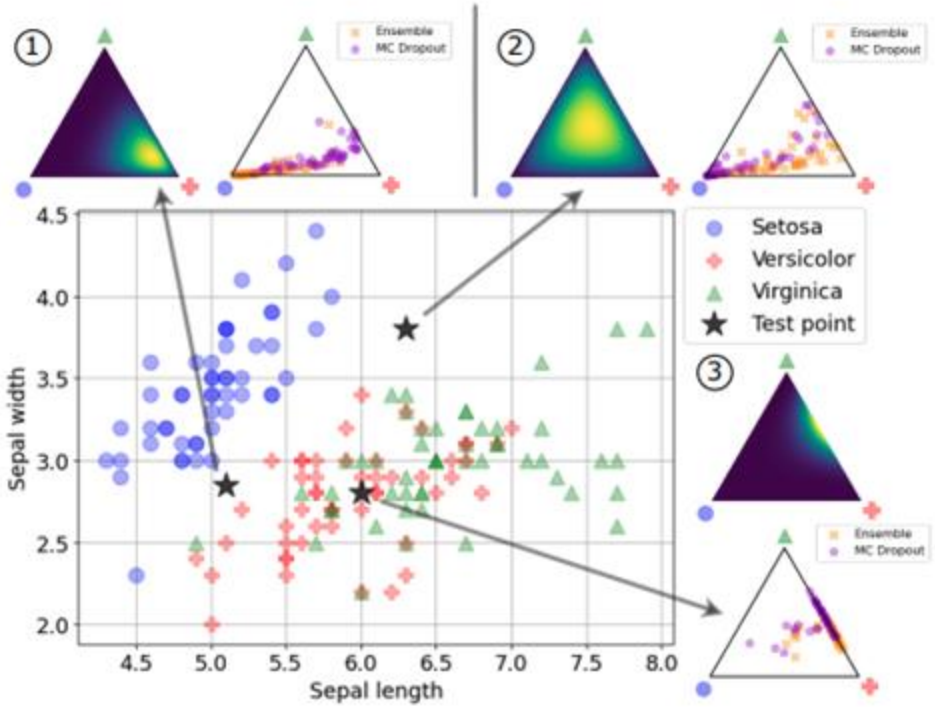


Figure 1: Classification of the rotated digit 1 (at bottom) at different angles between 0 and 180 degrees. **Left:** The classification probability is calculated using the *softmax* function. **Right:** The classification probability and uncertainty are calculated using the proposed method.

[Sensoy et al., NIPS, 2018]



[Ulmer et al. TMLR., 2023]

Evidential deep learning (EDL) for classification

- Instead of softmax class probabilities, EDL predicts for each training sample x_i the parameters $\alpha_i = (f_1(x_i|W) + 1, \dots, f_K(x_i|W) + 1)$ of a Dirichlet distribution $D(p_i|\alpha_i)$, which gives the class probabilities $p_{ik} = {}^{\alpha_{ik}}/_{E_i}$ and their uncertainties $\frac{p_{ik}(1-p_{ik})}{(E_i+1)}$ with evidence $E_i = \sum_{k=1}^{K} \alpha_{ik}$
- Instead of the cross-entropy loss, EDL trains a modified loss function:

$$\mathcal{L}(f(x_i; W), y_i) = \sum_{k=1}^{K} \left\{ (y_{ik} - \quad \frac{p_{ik}(1-p_{ik})}{+1)} \right\} + \lambda_t \sum_{i=1}^{N} KL(\quad \|\quad (1))$$

(a) Confident Prediction

(c) Out-of-distribution

- It also includes an unknow class
- All parameters are learned directly by back-propagation during training. No need for sampling as in Bayesian NN, which makes the method fast.

Outline

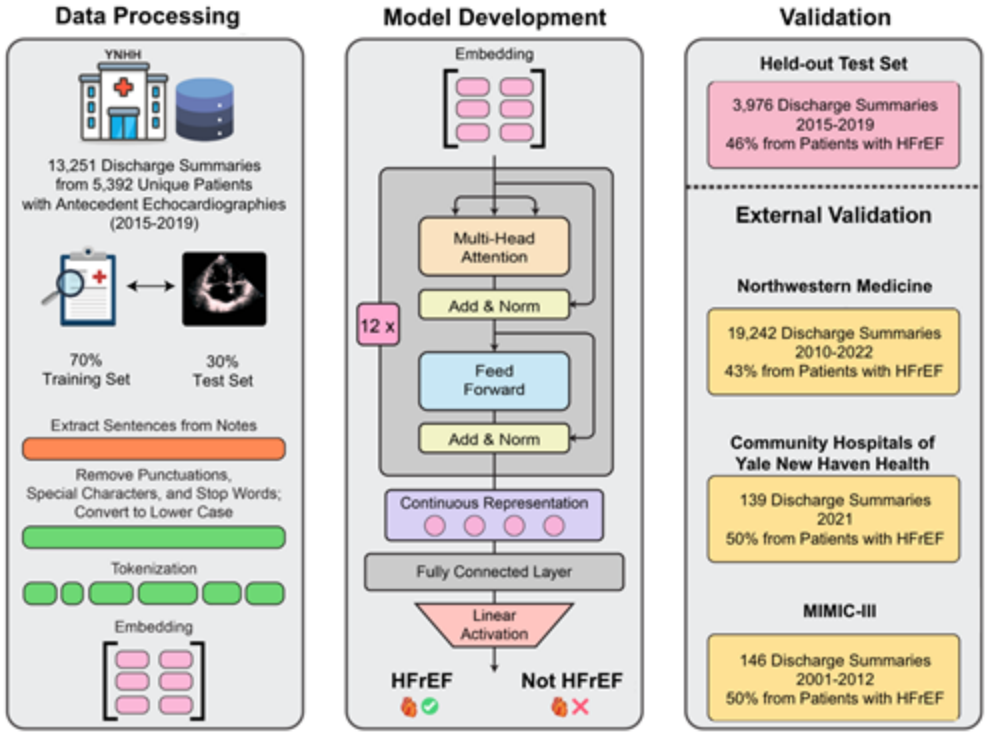
- History of deep learning
- Principles of deep learning
- How do deep neural networks learn?
- How do deep neural networks generalize?
- Biological and medical applications

When should I start using deep learning?

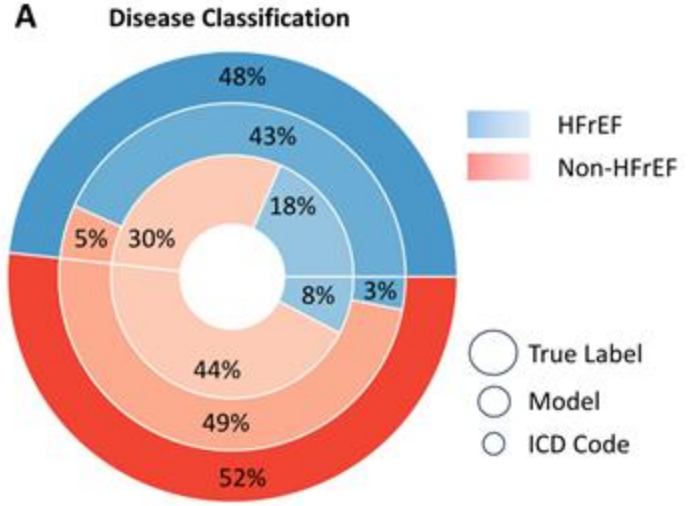
- Training from scratch
  - Availability of large datasets (>10000 items per class)
- Few shot learning, transfer learning or fine tuning
  - Pretrained foundation model adapted to the problem
  - Specialized dataset (can be fairly small)
- Zero shot learning
  - Pretrained foundation model adapted to the problem
- Datasets are well annotated and labeled for supervised learning
- Ideally, data should be homogenous, i.e. consisting of many multidimensional feature vectors with strong correlations and hierarchical structure (sequences, 3D structures, images, text, …)
- Computational resources and hardware (GPU, TPU) should be available
- Time to experiment
- The problem should not be too simple

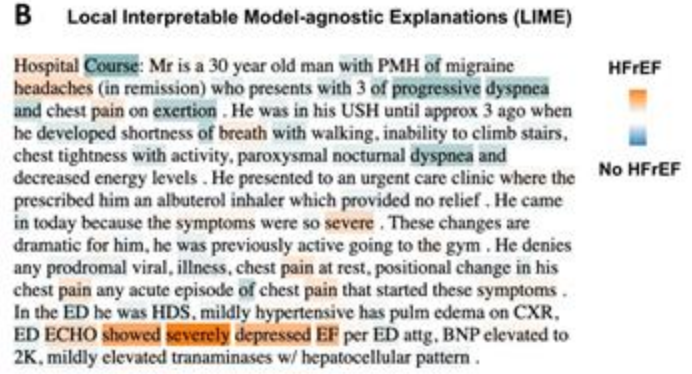
Using a language model to detect heart failure by parsing medical discharge summaries

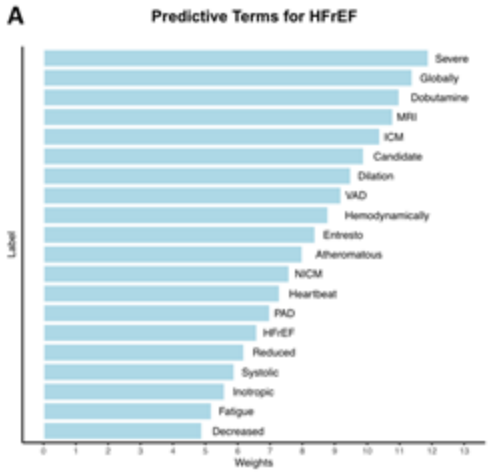
Longformer
[Beltagy et al. arxiv, 2020]



[Nargesi et al., medRxiv, 2023]

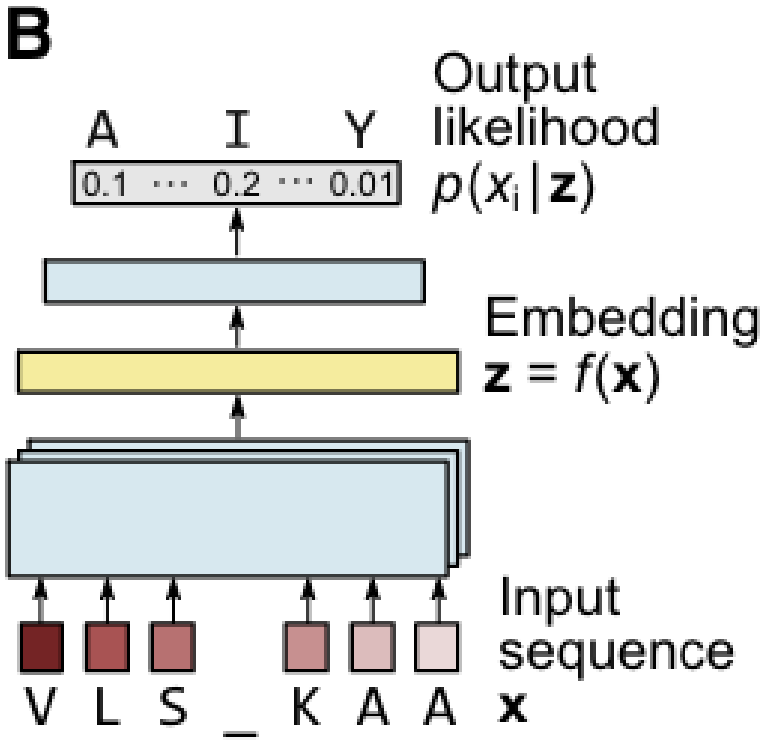




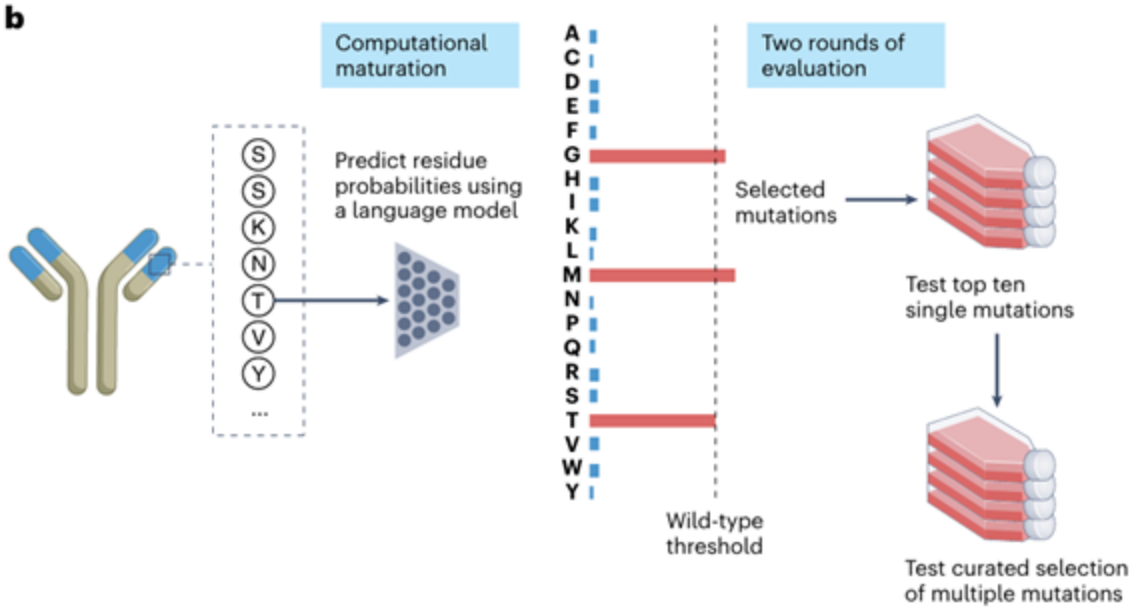


LIME: [Ribeiro et al. KDD, 2016]

Zero-shot learning from protein LLMs



ESM-1b ,ESM-1v
Rives et al. PNAS, 2021



$$\mathcal{M}(p_j) = \left\{ i \in [N], x_i' \in \mathcal{X} : \frac{p_j(x_i' \mid \mathbf{x})}{p_j(x_i \mid \mathbf{x})} > \alpha \right\},$$

Hie et al. Nat. Biotech. 2023

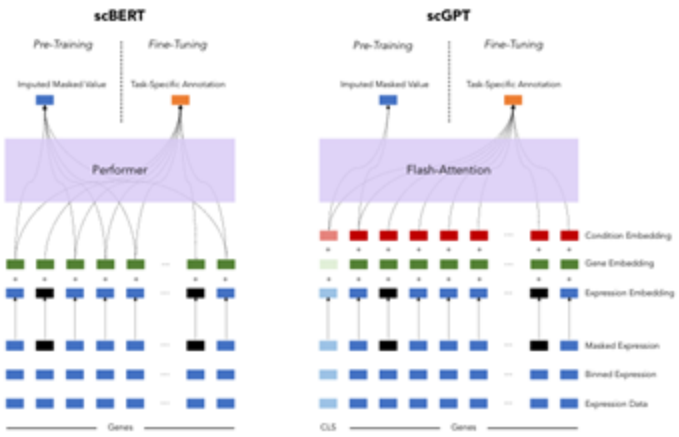
Few-shot learning with single cell foundation models

scGPT

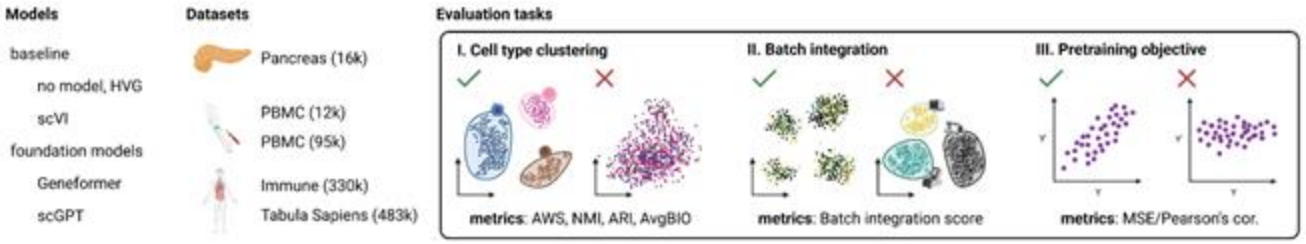
scBERT

Geneformer

Gene Embedding



> 10 million cells



"Our results indicate that both Geneformer and scGPT exhibit limited reliability in zero-shot settings and often underperform compared to simpler methods."

Kedzierska et al., arXiv, 2023

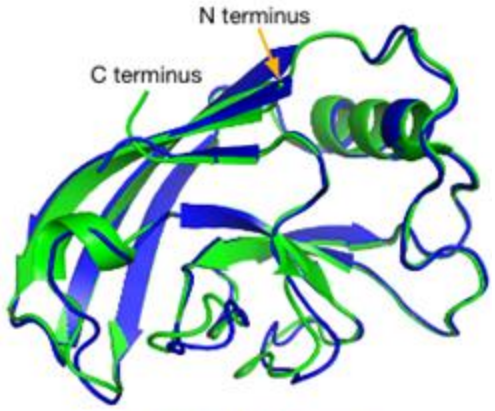
Model	Accuracy (↑)	Macro F1 (↑)	Accuracy (↑): 'hard to predict'	Macro F1 (↑): 'hard to predict'
scBERT (reported)	0.759	0.691	0.801	0.788
scBERT (reproduced)	0.766 ± 0.012	0.675 ± 0.012	0.765 ± 0.030	0.782 ± 0.013
L1 logistic regression	**0.811**	**0.707**	**0.848**	**0.828**

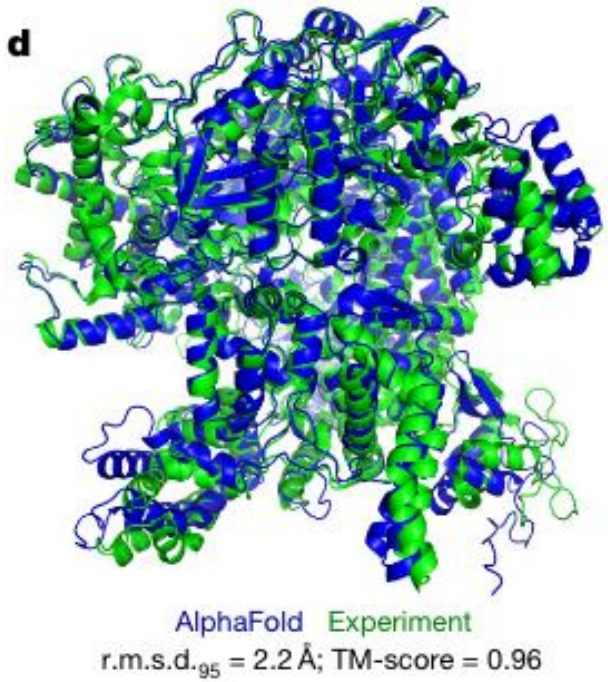
"Taken together, our results highlight the importance of rigorously testing foundation models against well established baselines, establishing challenging fine-tuning tasks on which to benchmark foundation models"

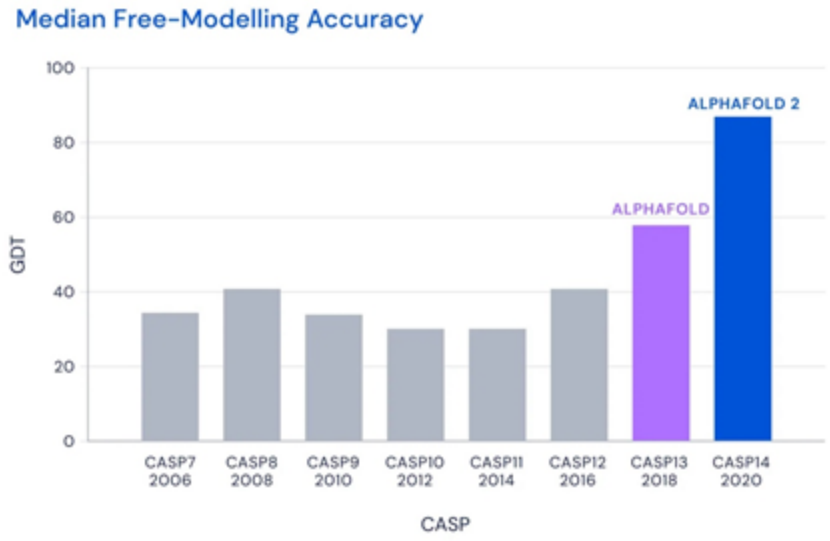
Boiarsky et al. , arXiv, 2023

AlphaFold2

- Highly accurate protein structure prediction
- Fast prediction from sequence to 3D structure (end-to-end)
- Turning sequences into structures and opening new paths to understand biology
- Extensions able to predict protein complexes (AlphaFold-Multimer)
- Emphasizes the need for curated data repositories (PDB)







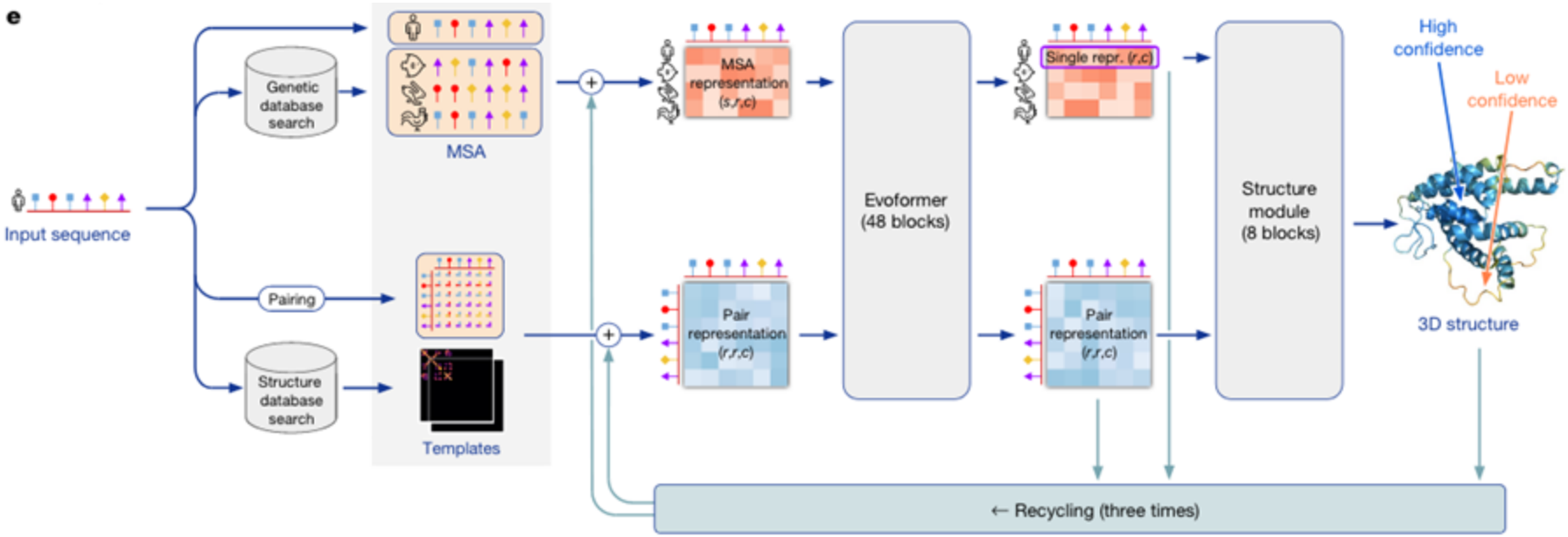
LASKER FOUNDATION

2023 Winners 2023 WINNERS

AlphaFold—for predicting protein structures 2023 BASIC
Demis Hassabis, John Jumper

[Review: DL & protein structure prediction AlQuraishi, Curr. Op. Chem. Biol., 2021]

AlphaFold2



Jumper et al. , Nature, 2021

John Jumper's talk on youtube

- end-to-end learning
- Physical and computational constraints built into the core of the network
- Easier to train on a limited set of 200'000 annotated PDB structures
- Iteratively learns distance matrix using MSA and Transformers
- The loss function compares predicted atom positions to true positions.
- Backbone 3D structure and side chain rotation are predicted by DNNs
- Peptide bond geometry is completely unconstrained (residue gaz). Constraints are enforced by loss penalty, and, after learning, exact constraints are enforced by Amber force field relaxation.

Some AlphaFold2 Results

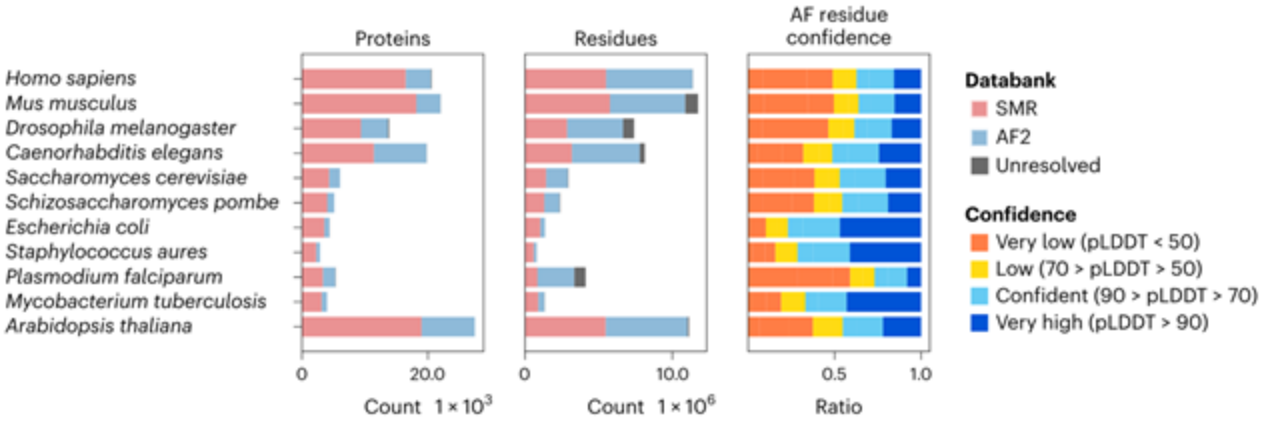
Human proteins 98.5% covered by AF2 (36% highly confident, 58% confident) compared to 17% with exp. structure

[Tunyasuvunakool et al. Nature, 2021]

https://alphafold.ebi.ac.uk/

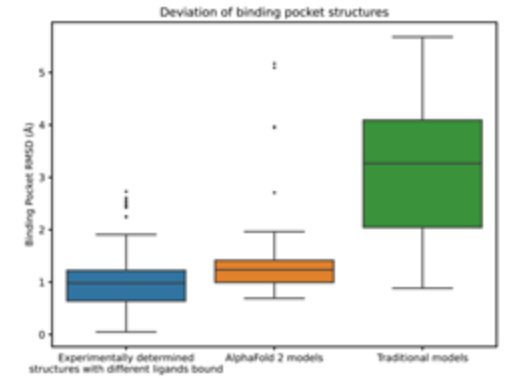
25% more confidently predicted residues compared to homology modeling. New domains discovered.

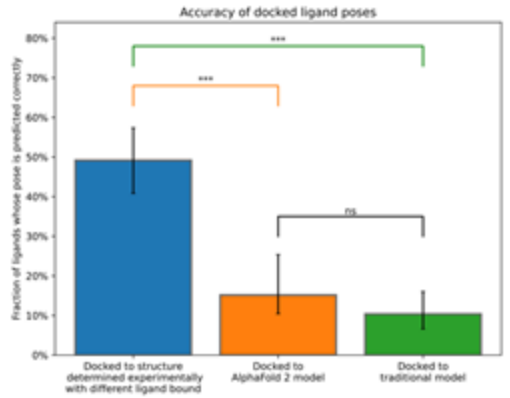
[Akdel et al., Nature, 2022]



G protein-coupled receptors are much better predicted by AF2 compared to homology models. However, molecular docking is not predicted more accurately by AF2.

[Karelina et al., bioRxiv, 2023]





Improvements after AlphaFold2

- De novo sequences (no MSA)
- Ultra-high residue accuracy (< 0.5A)
- Impact of minor sequence changes (PTMs, variants)
- Multidomain proteins
- Protein complexes (AlphaFold-Multimer)
- Protein conformational space (entropy)
- Folding trajectories

Prosit – MS/MS spectrum prediction

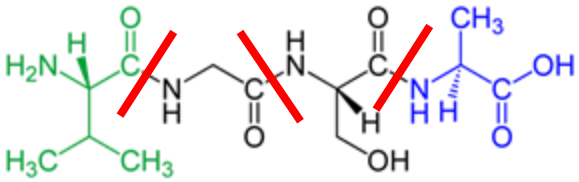
Protein

Sample preparation
proteolysis

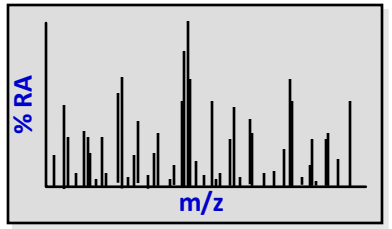
Protein *fragmented* into peptides

Sample introduction

Mass (m/z) measurement:
- MS + MS/MS



Match experimental MS/MS spectrum against protein sequence database



Experimental MS/MS spectrum

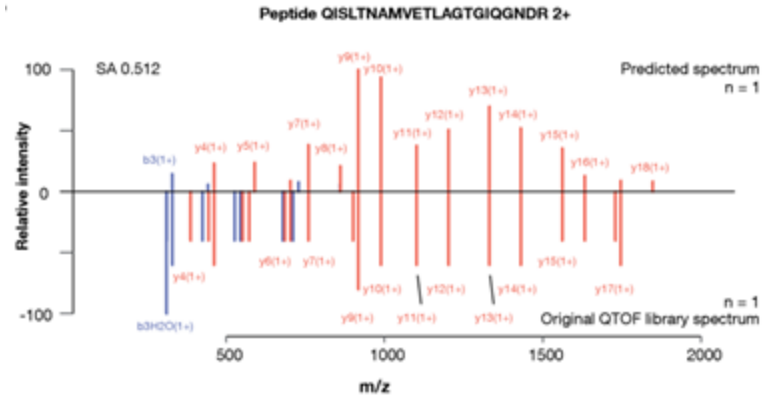
Plan quantitative DIA analysis using spectrum library

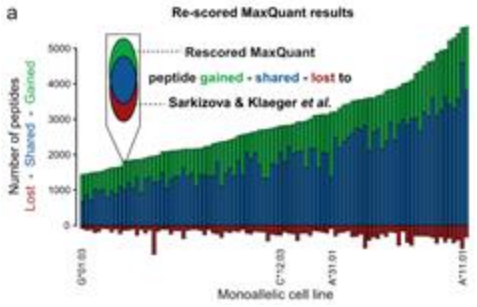
Prosit – MS/MS spectrum prediction

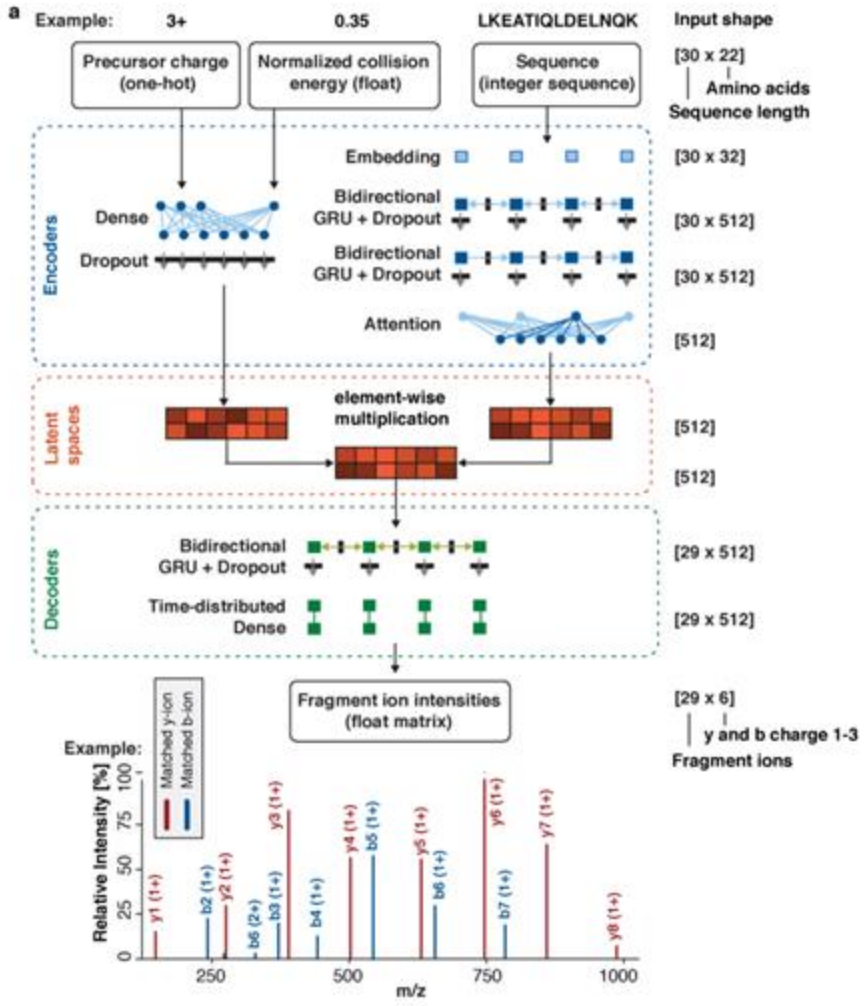
PROTEOMICSDB

Data from ProteomicsDB.org:

155,437,510 pepide spectrum matches







Gessulat et al, Nature, 2019

Tabular data: DL or gradient boosting

- Numerical, ordinal, and categorical features
- Missing values, data sparcity
- Often no comparable data and lack of prior knowledge
- Recently DL solutions (TabNet, NODE, DNF-Net, 1D-CNN) were proposed
- Tree-ensemble boosting algorithms such as XGBoost outperform these DL architectures for tabular data (Shwartz-Ziv & Armon, arXiv, 2021)

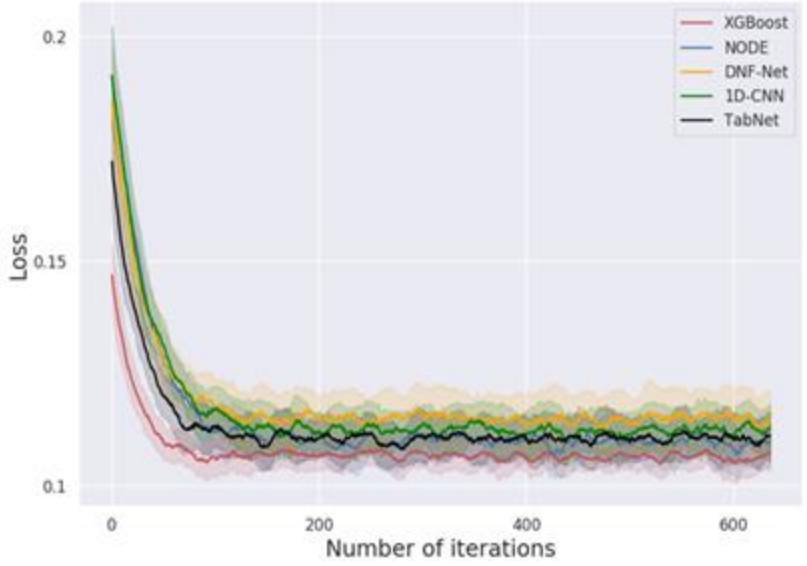


Figure 2: **The Hyper-parameters optimization process for different models.**