昆山杜克大学
DUKE KUNSHAN UNIVERSITY

# introduction

stats403_deep_learning

spring_2025

lecture_1

# logistics

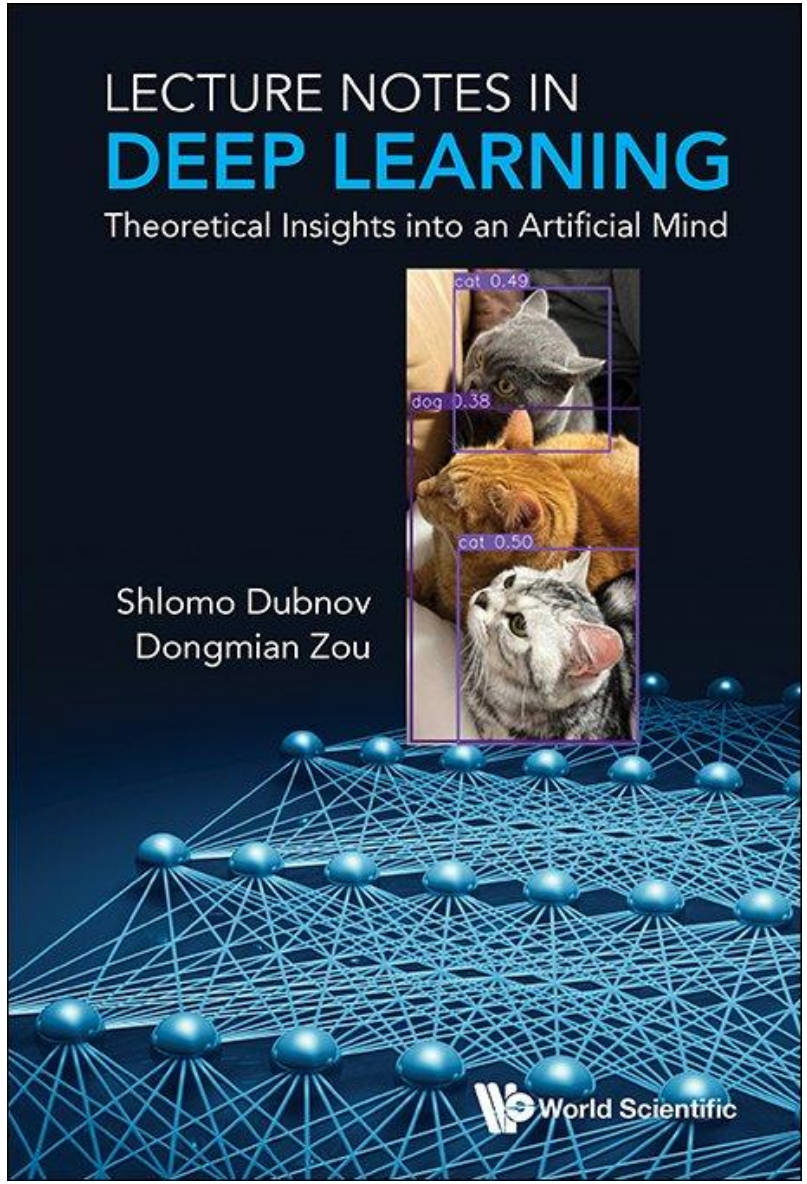
- lectures:
  - MoWe 8:30-11am @ WDR 1007

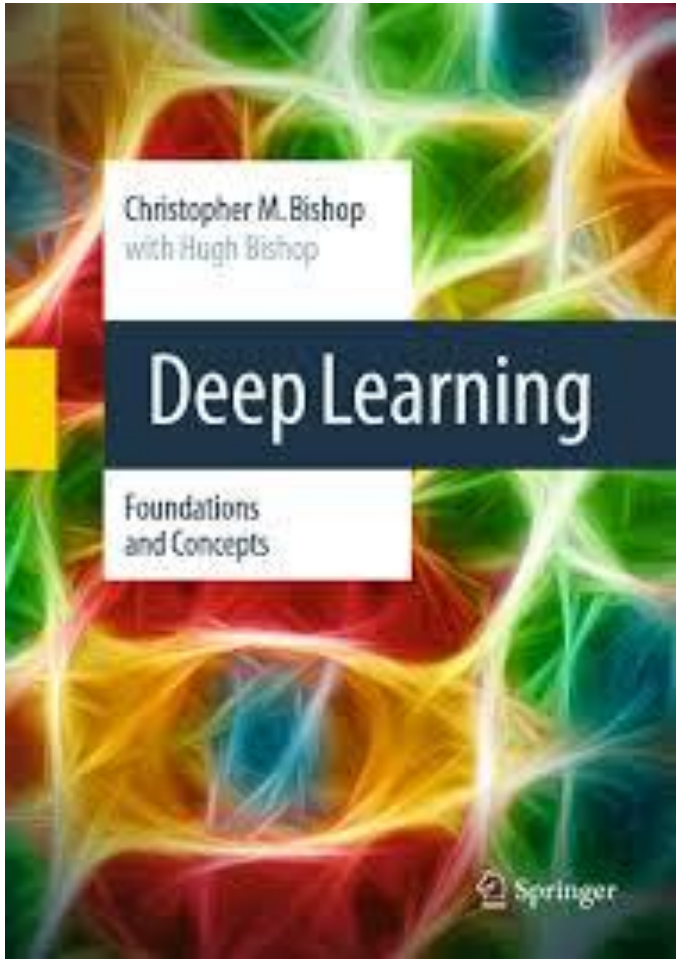
- Dongmian's office hour:
  - Email: dongmian.zou@dukekunshan.edu.cn
  - Office hour:
    - Mo 11am-noon @ WDR 3010
    - or by appointment

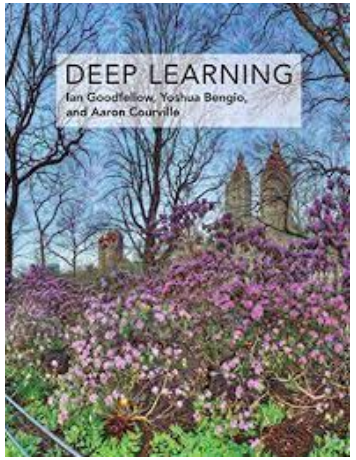
# logistics

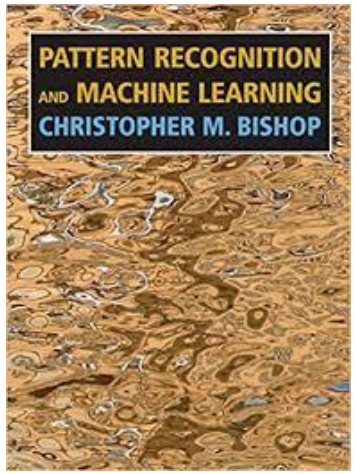
- lectures
- homework assignments: 2 in total
  - You can discuss with friends or consult with AI, but you must write up the solution by yourself.
- exams: 1 take-home
- course project: 1 individual project (proposal; presentation; report)

# textbook


LECTURE NOTES IN
DEEP LEARNING
Theoretical Insights into an Artificial Mind
cat 0.49
dog 0.38
cat 0.50
Shlomo Dubnov
Dongmian Zou
World Scientific


Christopher M. Bishop
with Hugh Bishop
Deep Learning
Foundations
and Concepts
Springer


DEEP LEARNING
Ian Goodfellow, Yoshua Bengio,
and Aaron Courville


PATTERN RECOGNITION
AND MACHINE LEARNING
CHRISTOPHER M. BISHOP

# course project

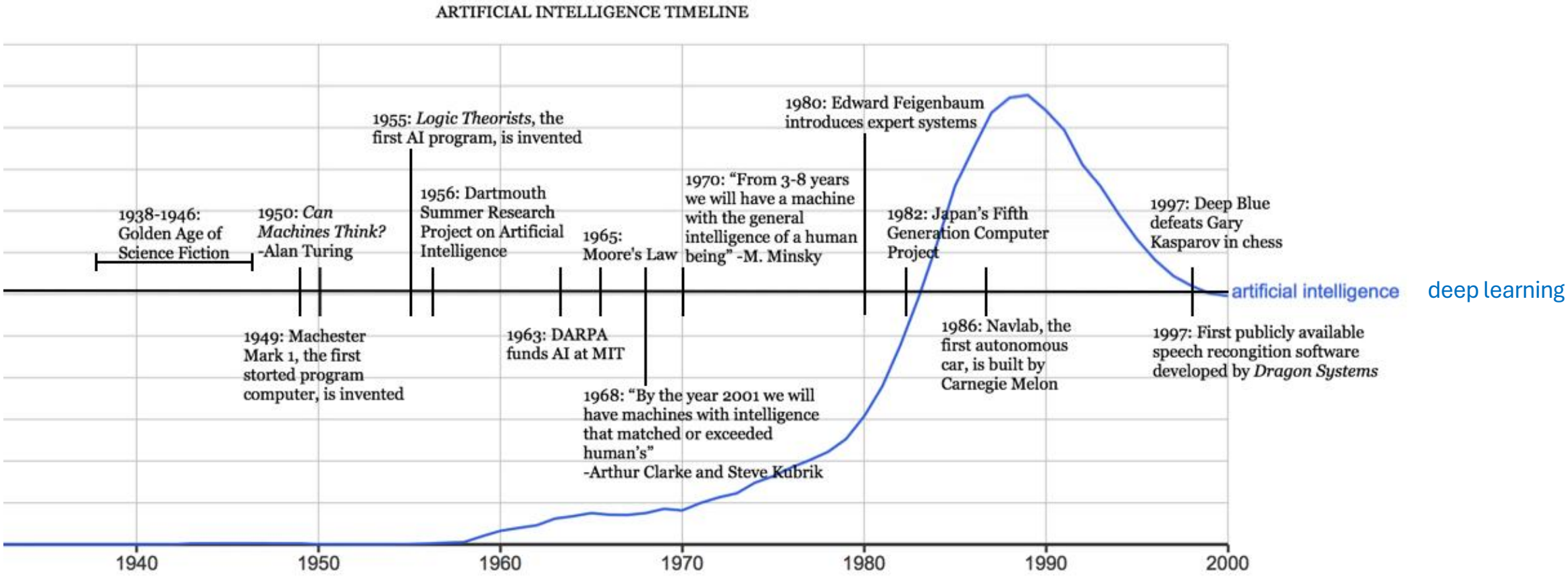
1. You will need to find a research paper on deep learning.
   - It should be published in one of the following conferences: NeurIPS, ICML, ICLR, AISTATS.
   - Also, it should be published during the last three years (2023-2025).
2. The paper should include an algorithm that you can implement (or has open-source code available that you can execute; be careful about the hardware requirement when choosing the paper) and of real interest to you (you should be really excited about it).
3. You need to study the methodology in the paper, including any theoretical guarantees.
4. You need to implement the algorithm on a dataset or application that is relevant and not considered in the paper.
5. You need to report your findings and discuss the performance and any unexpected outcomes.
6. (bonus) You will get bonus points if you extend or improve the methodology.

## course project (see syllabus for rubrics)

- proposal presentation (during week 2; ~5 min)
- final presentation (during week 7)
- report (due during the final week)

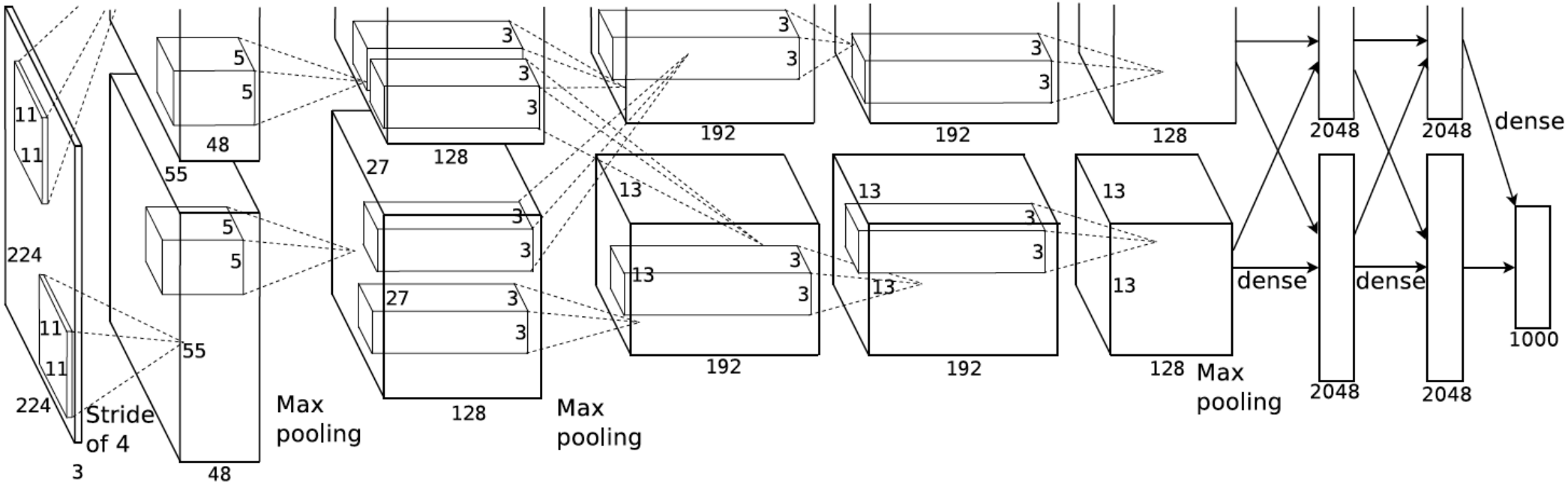
# 1.1 deep learning: what and why

# history of artificial intelligence

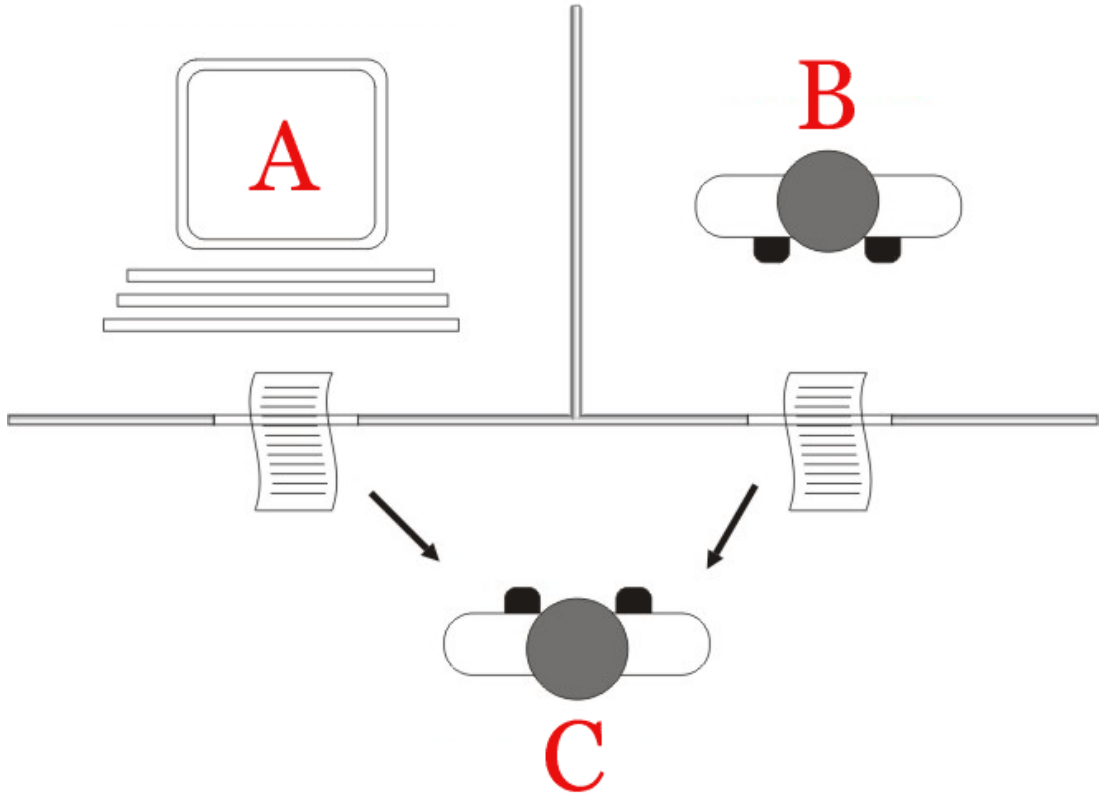


# AlphaGo (2014)

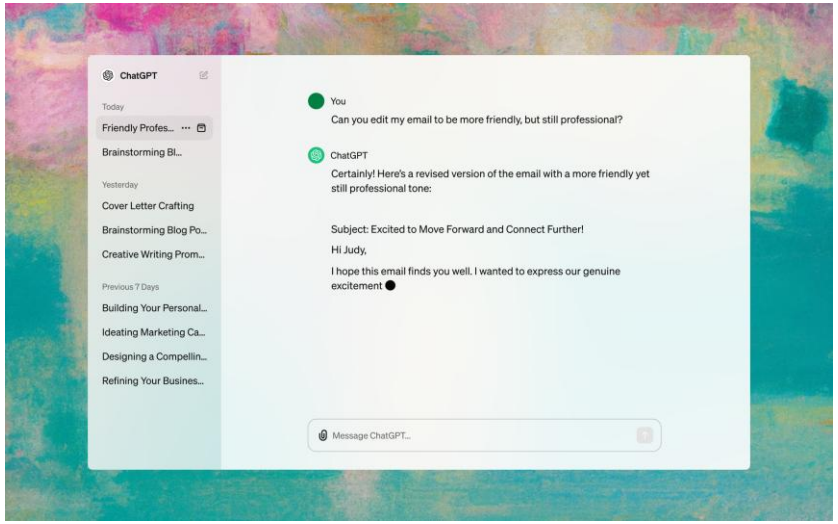
# AlexNet (2012)



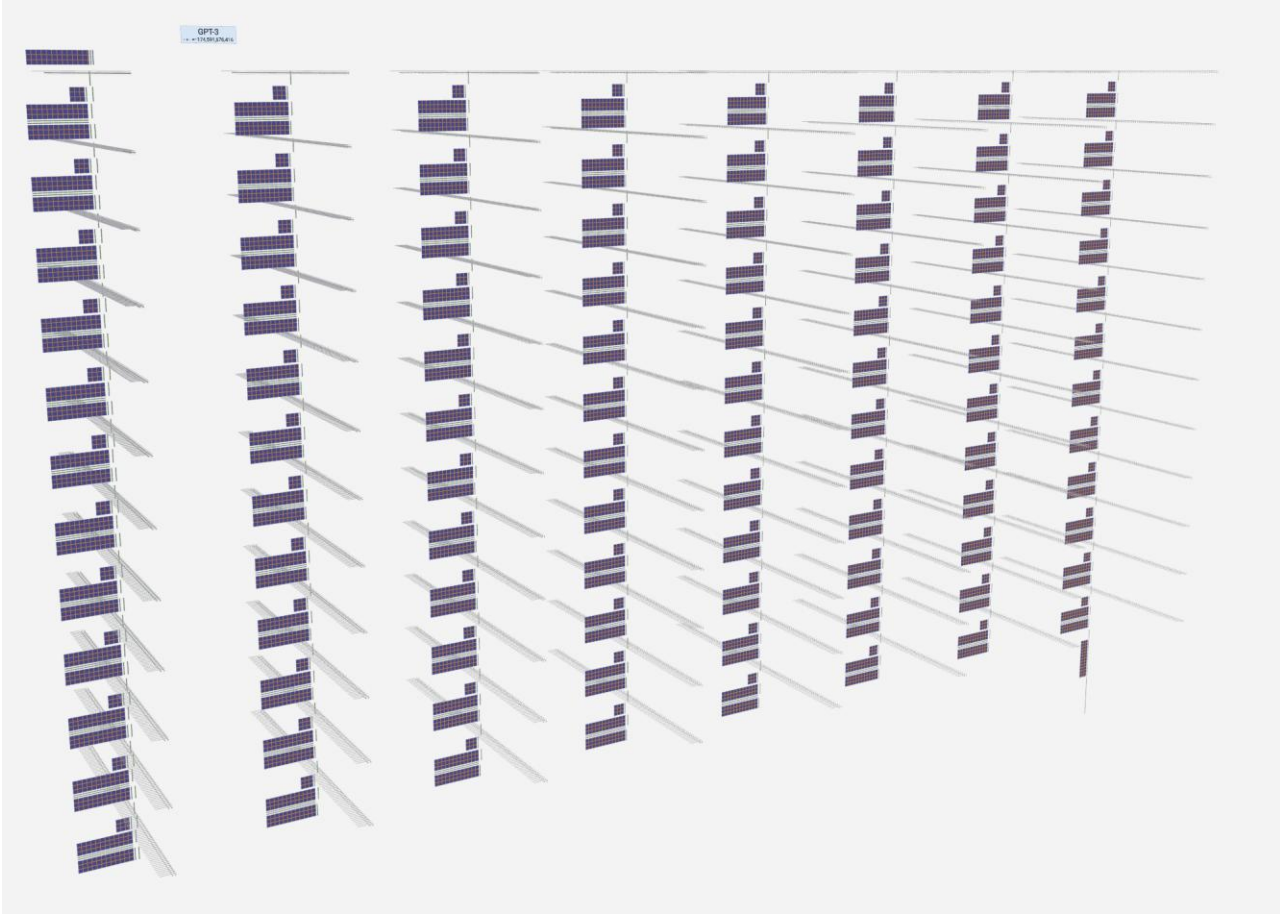
# AI driven by deep learning




ChatGPT



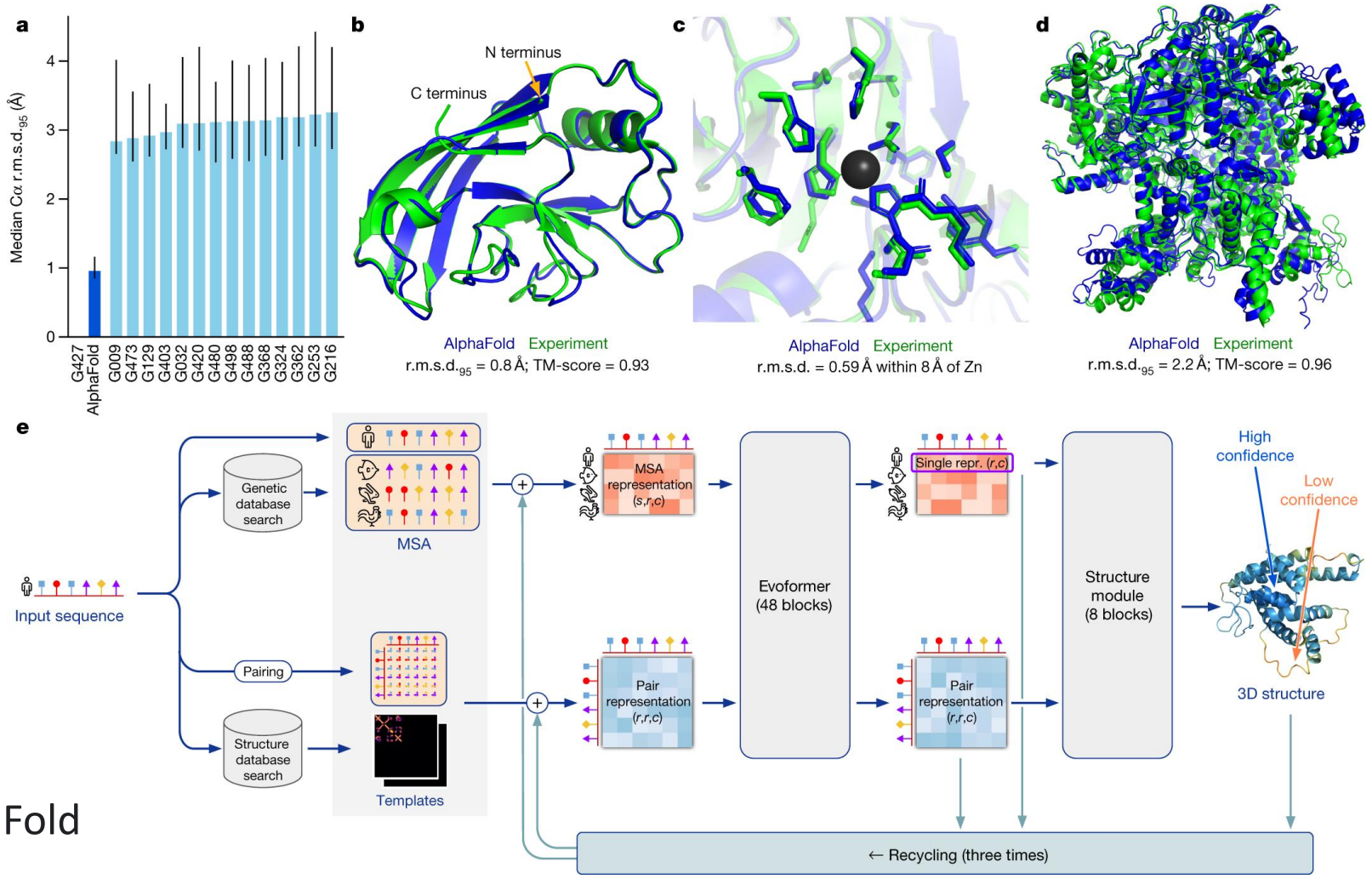
# AI driven by deep learning



# AI driven by deep learning

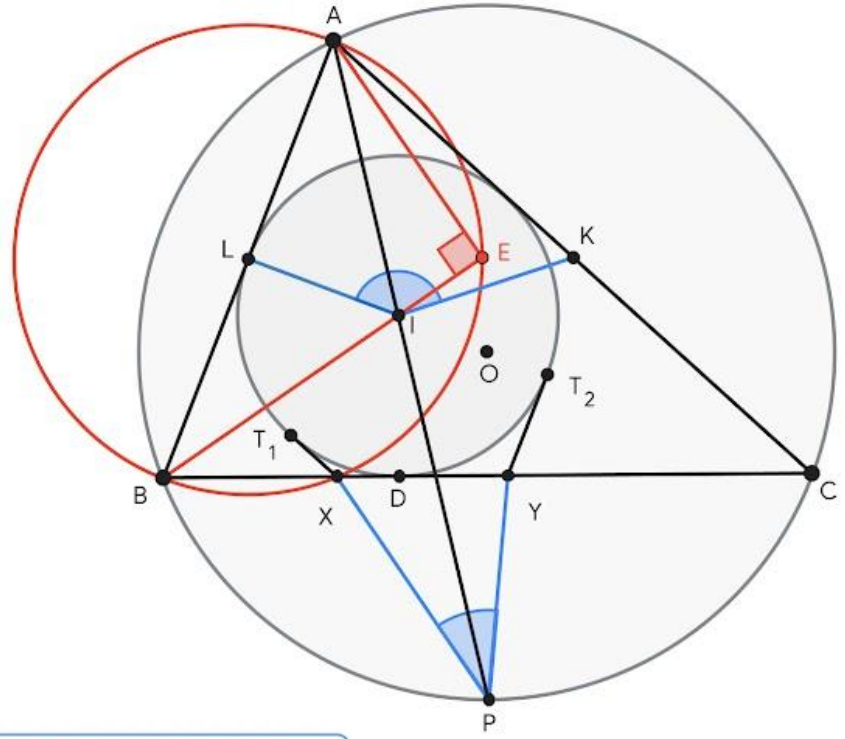
# AI driven by deep learning

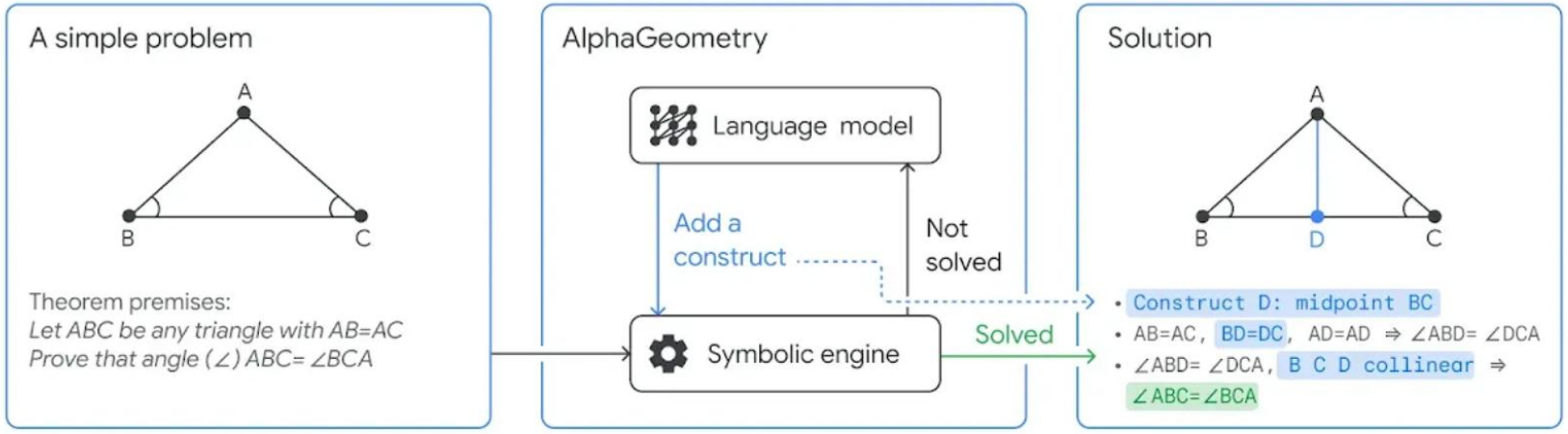

# AI driven by deep learning




AlphaFold

# AI driven by deep learning



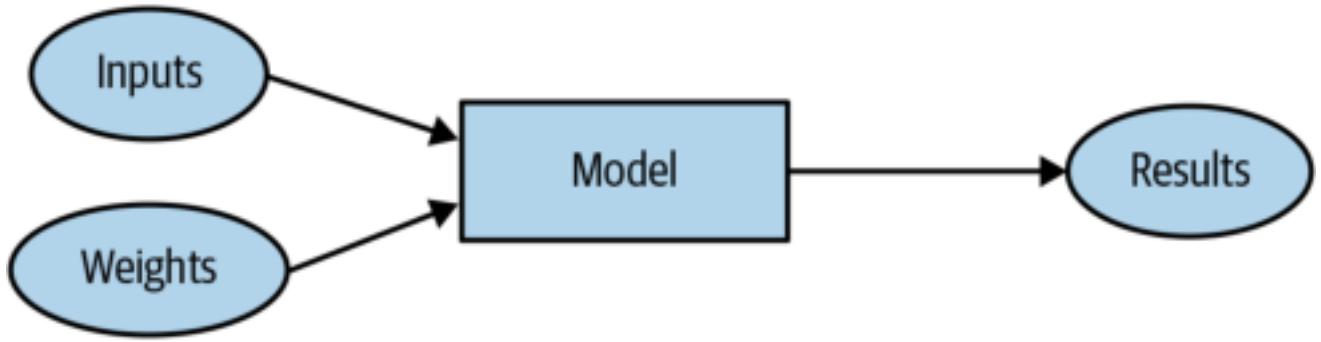



# AI driven by deep learning

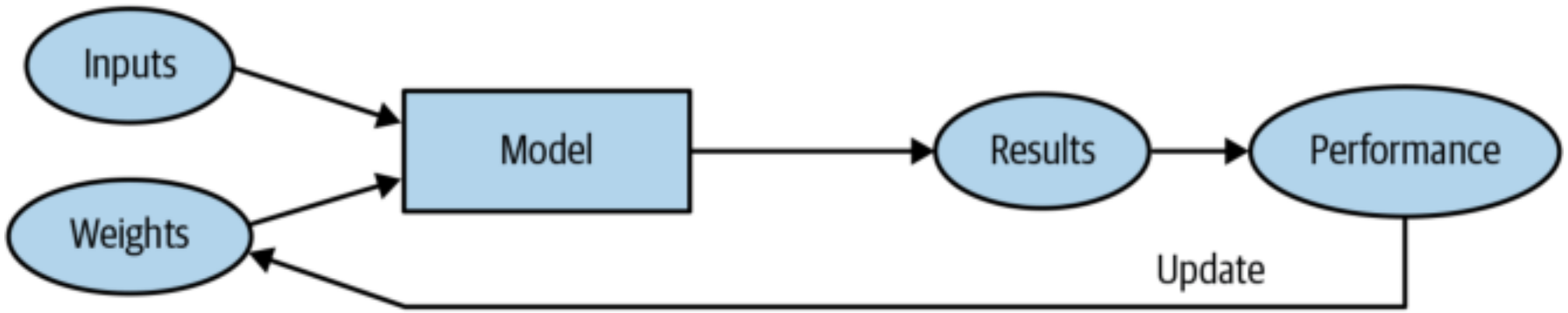
# AI driven by deep learning

# 1.2 from linear models to neural networks

# a typical machine learning program



# a typical training procedure

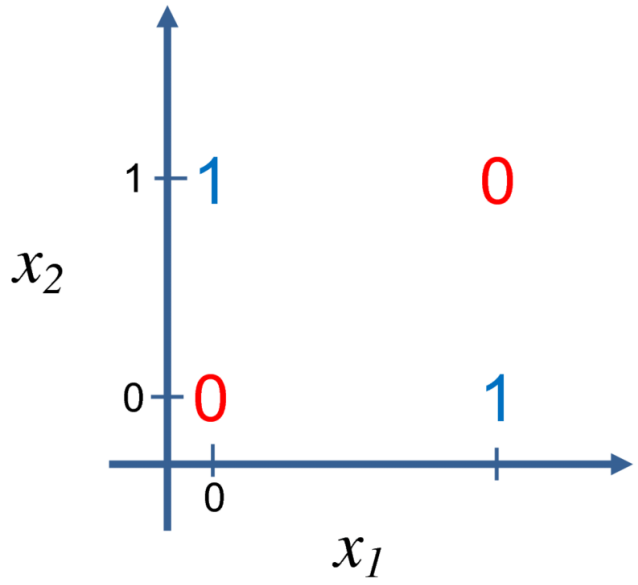


learn “exclusive OR”

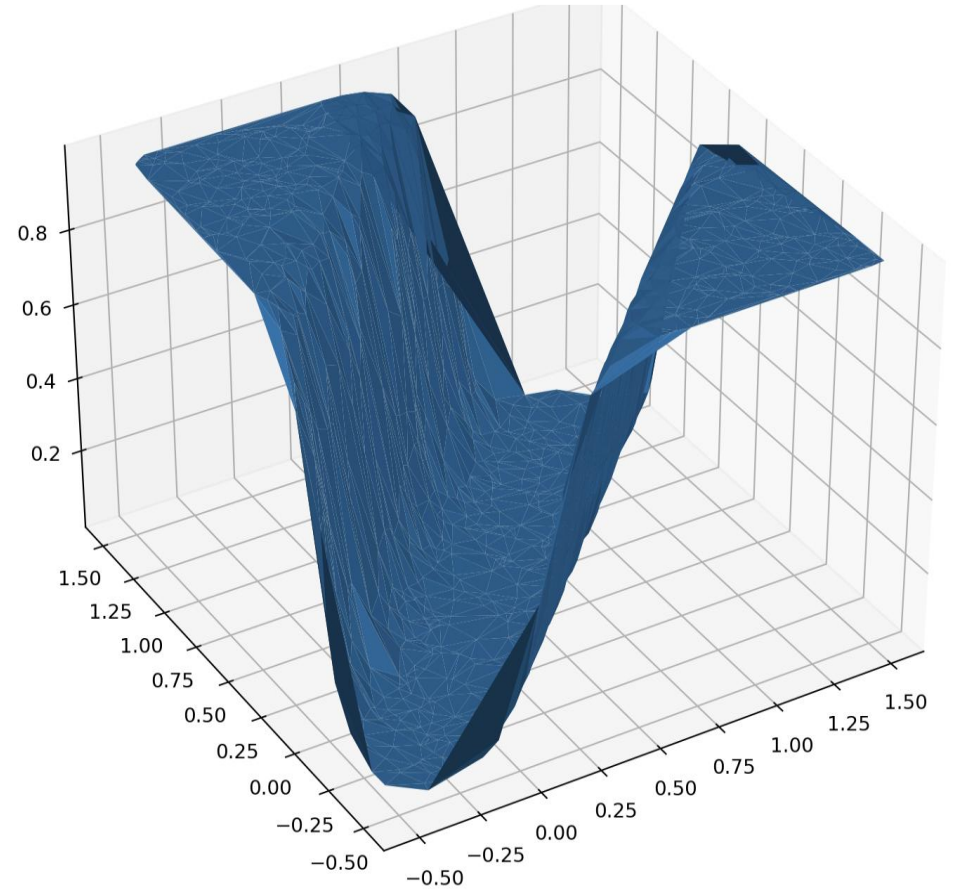
$$f^*(0,0) = f^*(1,1) = 0$$
$$f^*(0,1) = f^*(1,0) = 1$$

# learn “exclusive OR”

- dataset: $\mathbb{X} = \{(0,0)^{\mathrm{T}}, (0,1)^{\mathrm{T}}, (1,0)^{\mathrm{T}}, (1,1)^{\mathrm{T}}\}$



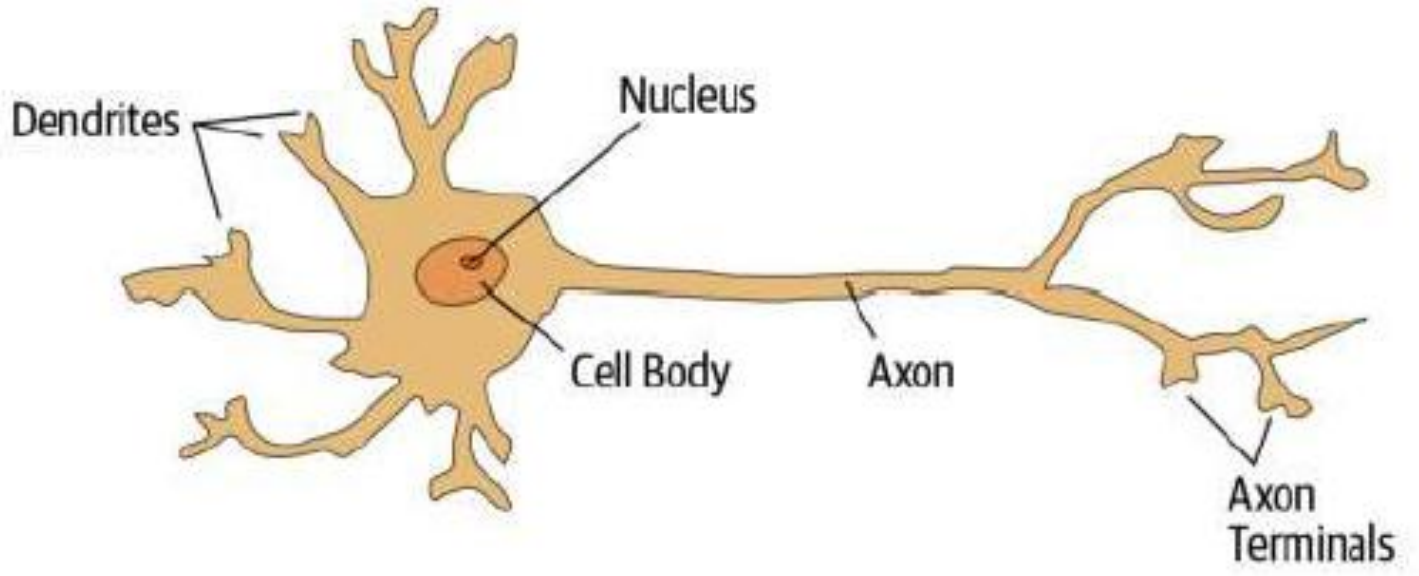
# learn “exclusive OR”




need: a big hypothesis class

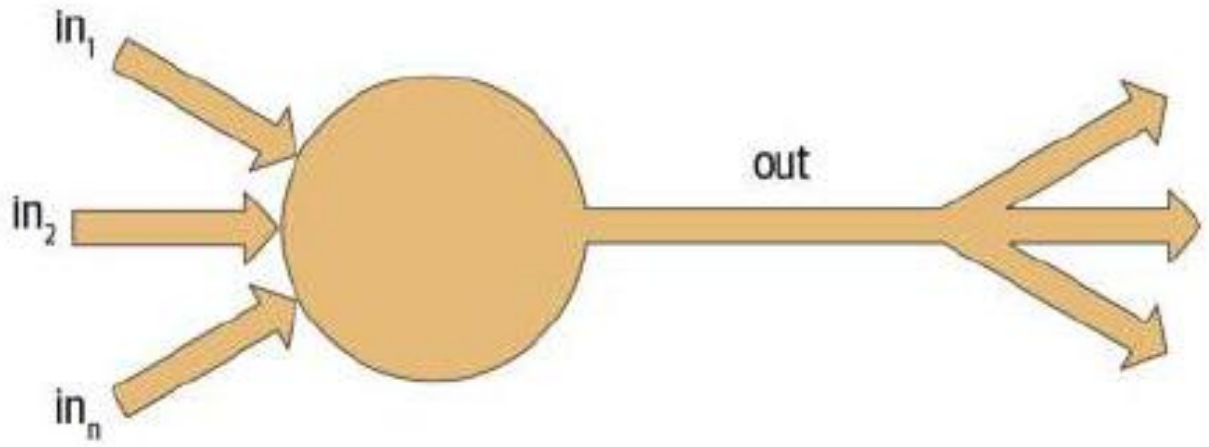
# neural networks

To illustrate, let's look at a simplest structure of a "neural network", which tries to mimic how human brains work.



# neural networks

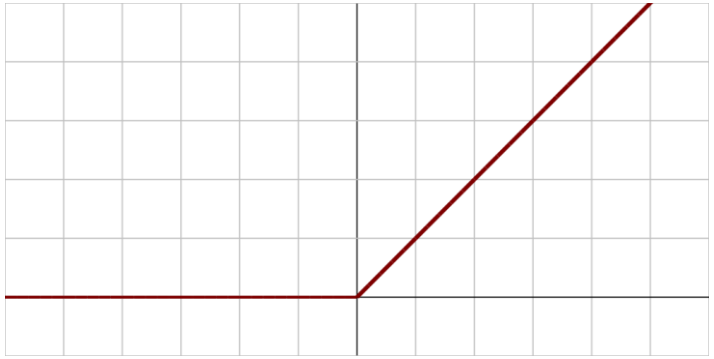
To illustrate, let's look at a simplest structure of a "neural network", which tries to mimic how human brains work.



# activation functions

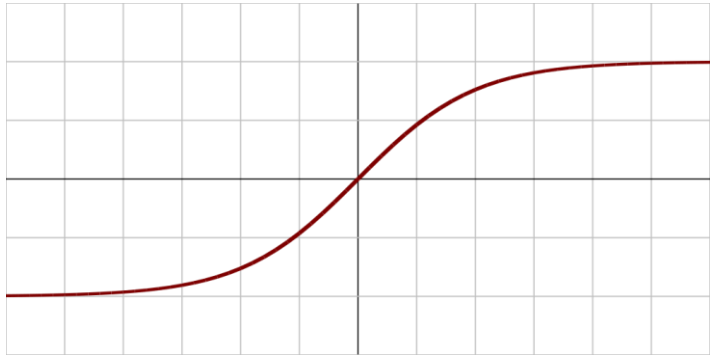
- ReLU (rectified linear unit)

$$\mathrm{ReLU}(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{otherwise.} \end{cases}$$



- tanh (hyperbolic tangent)
  - (similarly) sigmoid

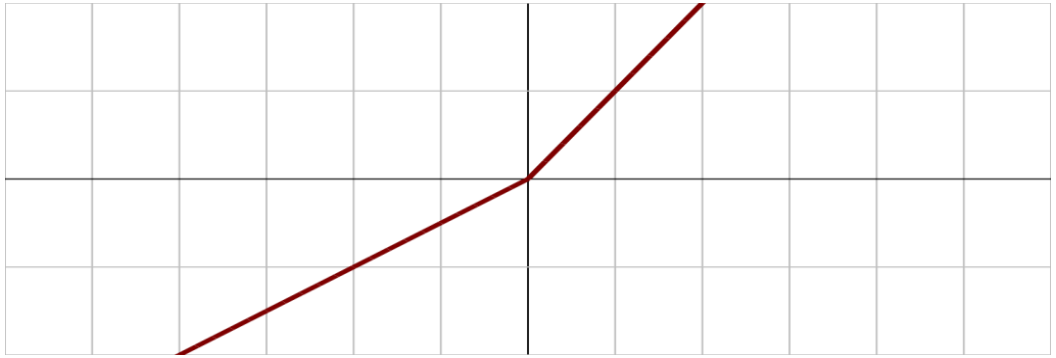
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \qquad \mathrm{sigmoid}(x) = \frac{1}{1 + e^{-x}}.$$



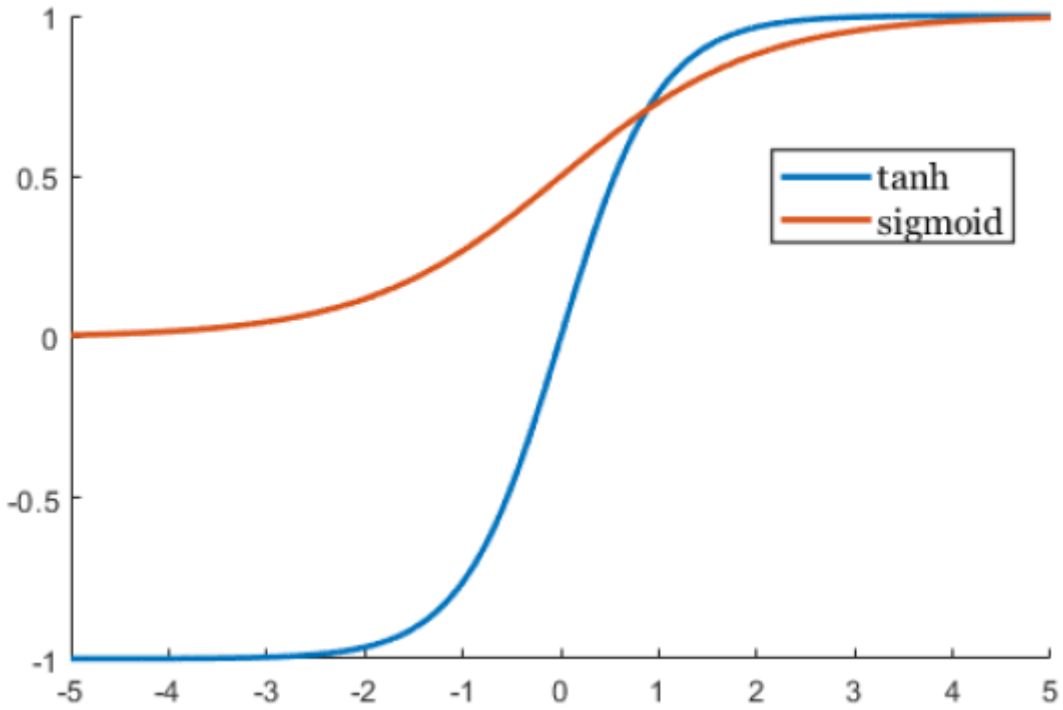
- leaky ReLU

given $0 < \alpha < 1$,

$$\mathrm{LeakyReLU}(x) = \begin{cases} \alpha x, & \text{if } x \leq 0 \\ x, & \text{otherwise.} \end{cases}$$



# activation functions

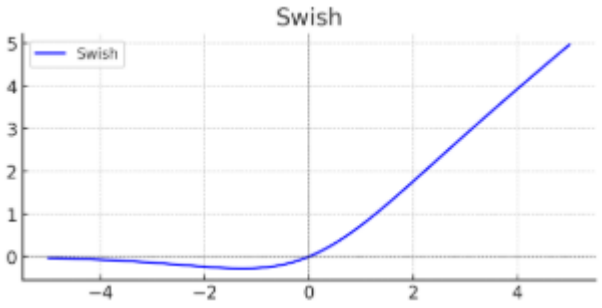


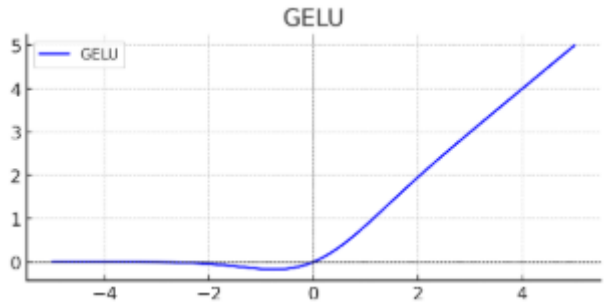
# activation functions that have gained popularity recently

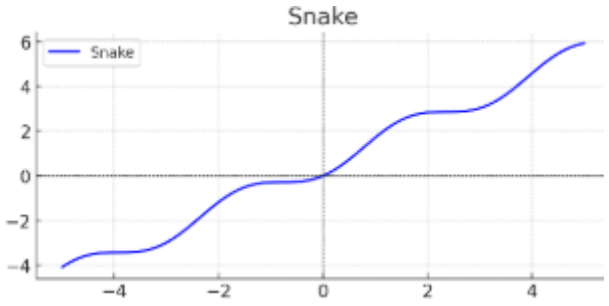
- Swish : $\text{swish}(x) = x \cdot \text{sigmoid}(x)$
  - also known as SiLU

- GELU: $\text{GELU}(x) = x \cdot \Phi(x)$ where $\Phi$ is the cdf of $N(0,1)$

- snake: $\text{snake}(x) = x + \frac{1}{\alpha}\sin^2(\alpha x)$



# activation functions that have gained popularity recently

## Dynamic Tanh (DyT)

$$\mathrm{DyT}(\boldsymbol{x}) = \boldsymbol{\gamma} * \tanh(\alpha \boldsymbol{x}) + \boldsymbol{\beta}$$

### Transformers without Normalization

**Jiachen Zhu**[1,2], **Xinlei Chen**[1], **Kaiming He**[3], **Yann LeCun**[1,2], **Zhuang Liu**[1,4,†]

[1]FAIR, Meta, [2]New York University, [3]MIT, [4]Princeton University
[†]Project lead

Normalization layers are ubiquitous in modern neural networks and have long been considered essential. This work demonstrates that Transformers without normalization can achieve the same or better performance using a remarkably simple technique. We introduce Dynamic Tanh (DyT), an element-wise operation $\mathrm{DyT}(\boldsymbol{x}) = \tanh(\alpha \boldsymbol{x})$, as a drop-in replacement for normalization layers in Transformers. DyT is inspired by the observation that layer normalization in Transformers often produces tanh-like, $S$-shaped input-output mappings. By incorporating DyT, Transformers without normalization can match or exceed the performance of their normalized counterparts, mostly without hyperparameter tuning. We validate the effectiveness of Transformers with DyT across diverse settings, ranging from recognition to generation, supervised to self-supervised learning, and computer vision to language models. These findings challenge the conventional understanding that normalization layers are indispensable in modern neural networks, and offer new insights into their role in deep networks.

**Date:** March 14, 2025
**Project page and code:** jiachenzhu.github.io/DyT
**Correspondence:** jiachen.zhu@nyu.edu, zhuangl@princeton.edu

Meta

**Algorithm 1** Pseudocode of DyT layer.

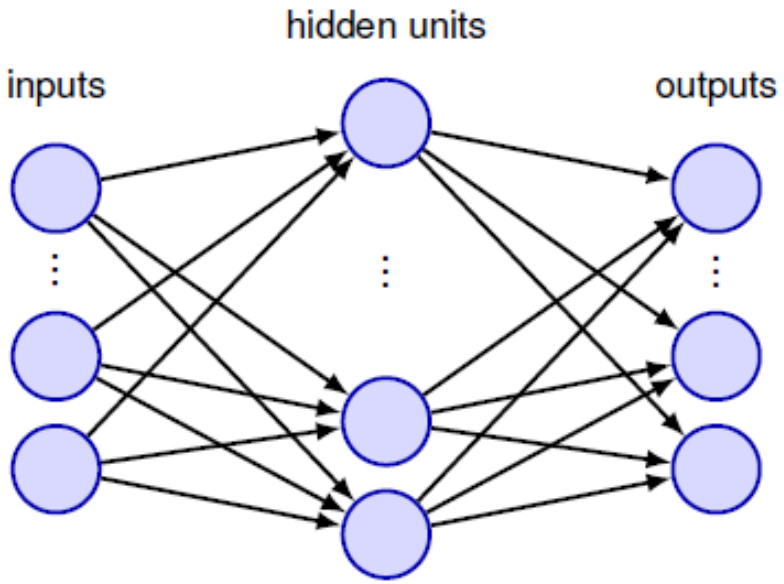
```
# input x has the shape of [B, T, C]
# B: batch size, T: tokens, C: dimension

class DyT(Module):
    def __init__(self, C, init_α):
        super().__init__()
        self.α = Parameter(ones(1) * init_α)
        self.γ = Parameter(ones(C))
        self.β = Parameter(zeros(C))

    def forward(self, x):
        x = tanh(self.alpha * x)
        return self.γ * x + self.β
```

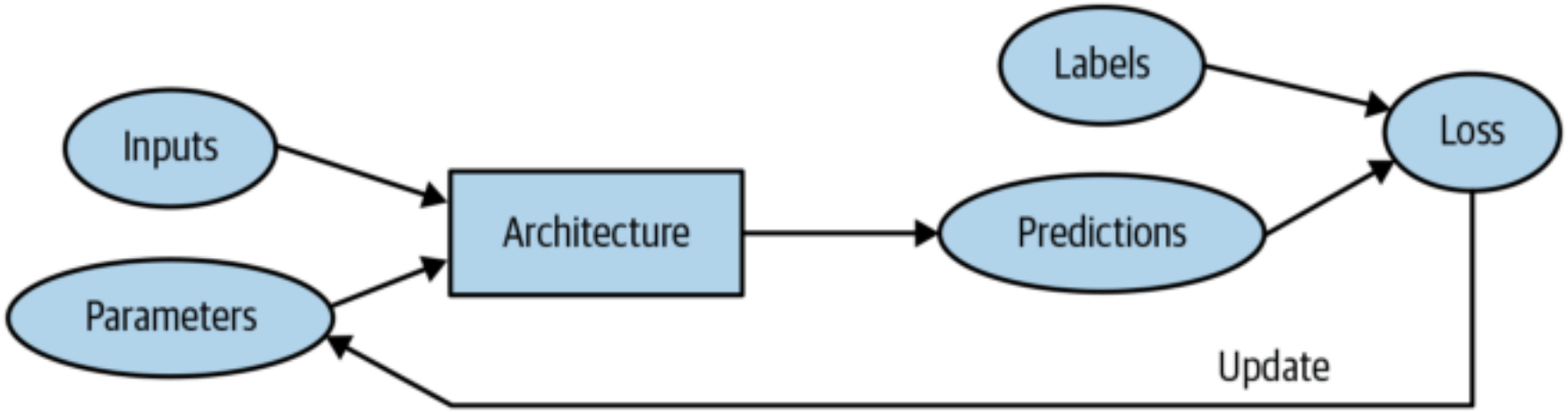
# neural networks

- a neural network can have more hidden layers if necessary
- a deep neural network refers to a neural network that has multiple hidden layers
- names for the simplest NN:
  - ❑ fully connected layers/networks
  - ❑ multi-layer perceptron (MLP)
  - ❑ dense layers/networks



# 1.3 training neural networks

# training loop



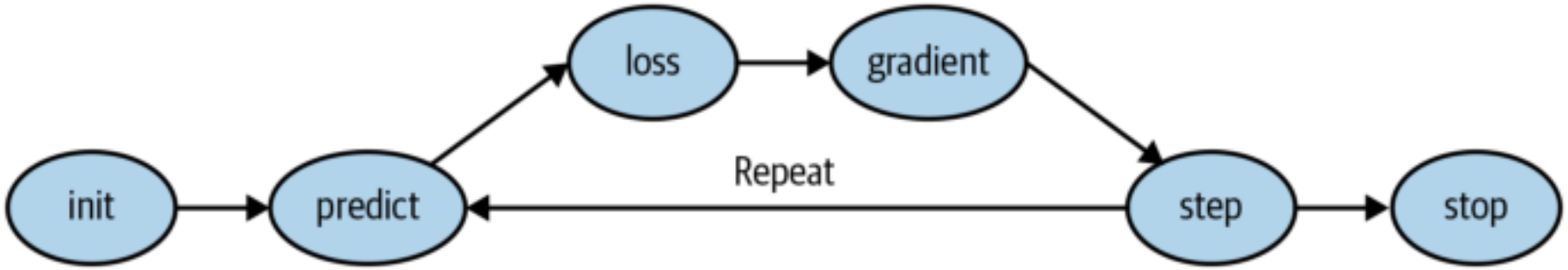
# gradient descent

- the loss function has “fastest decay” along its “– gradient”
- we would take

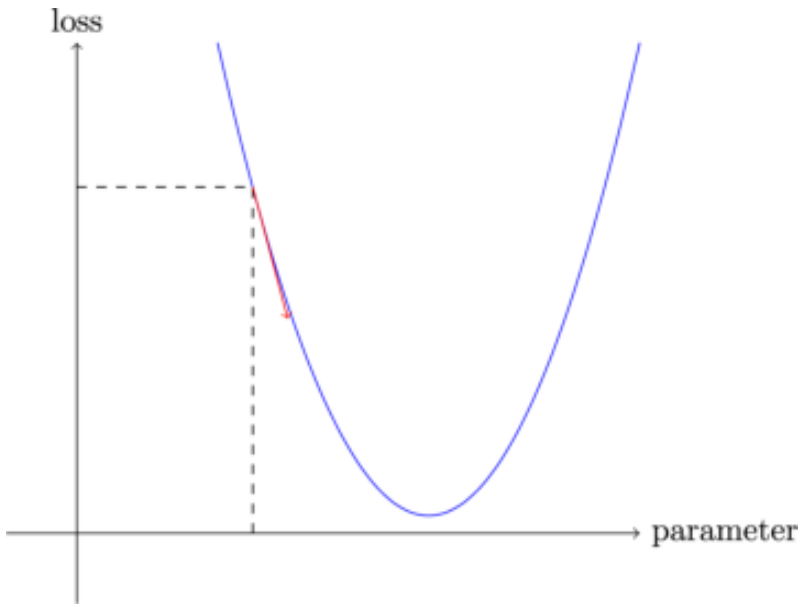
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} \ell(\theta^{(t)})$$

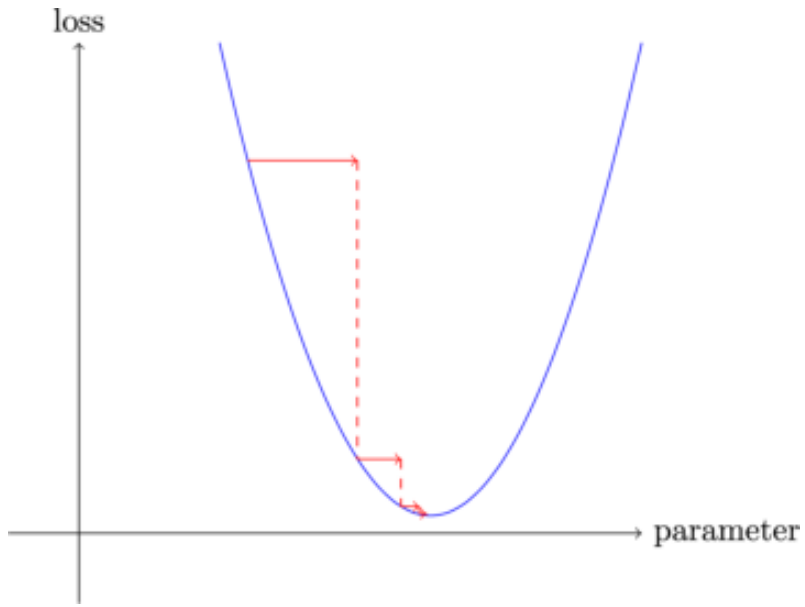
learning rate

# gradient descent



# gradient descent



# stochastic gradient descent

- note that our loss function is essentially an expected loss with respect to the data distribution

$$\ell(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\mathcal{E}(\mathbf{x};\boldsymbol{\theta})\right]$$

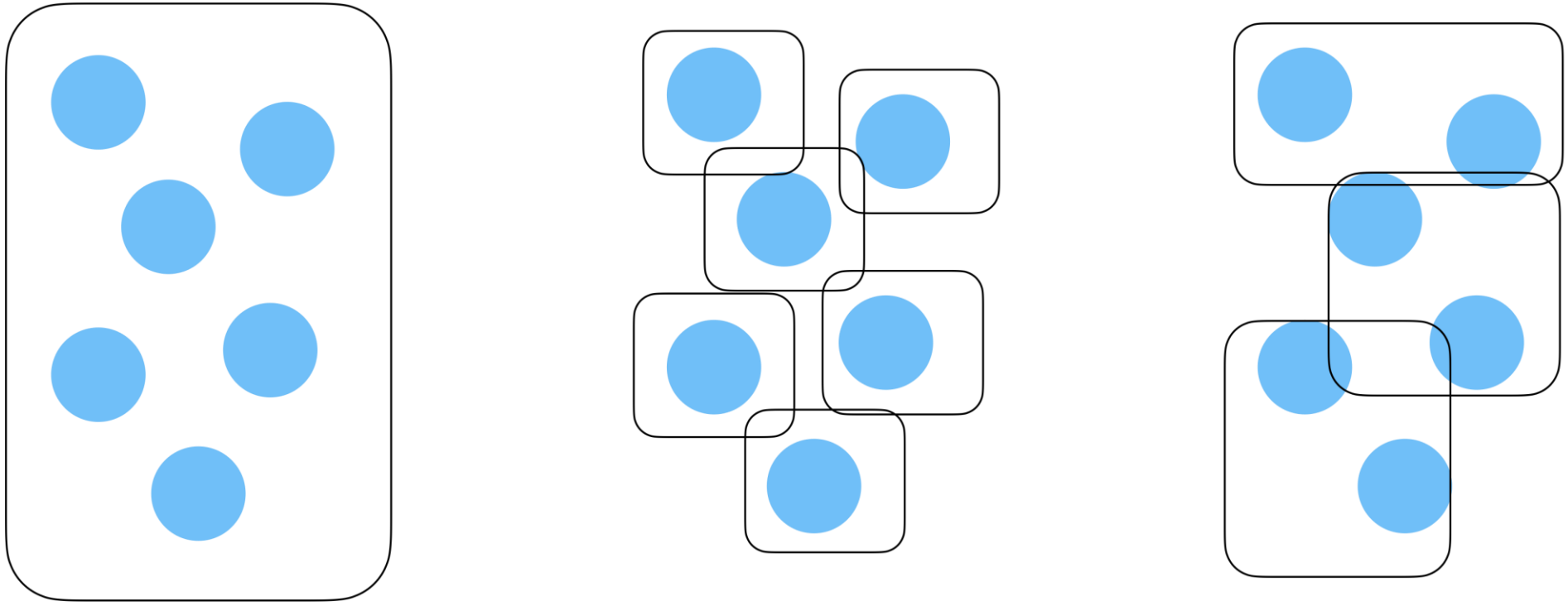
error function

- therefore,

$$\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\nabla_{\boldsymbol{\theta}}\mathcal{E}(\mathbf{x};\boldsymbol{\theta})\right]$$

- in deep learning, usually for each “gradient step”, the expectation is taken over a “mini-batch”

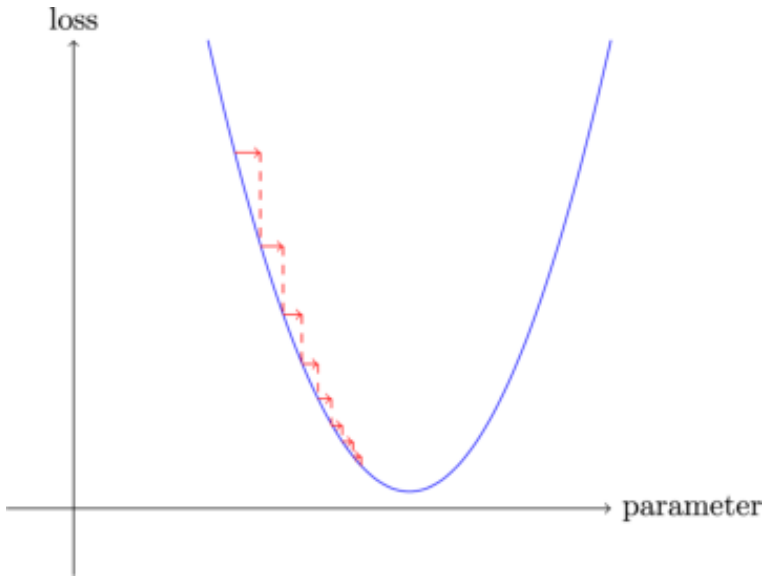
# mini-batch



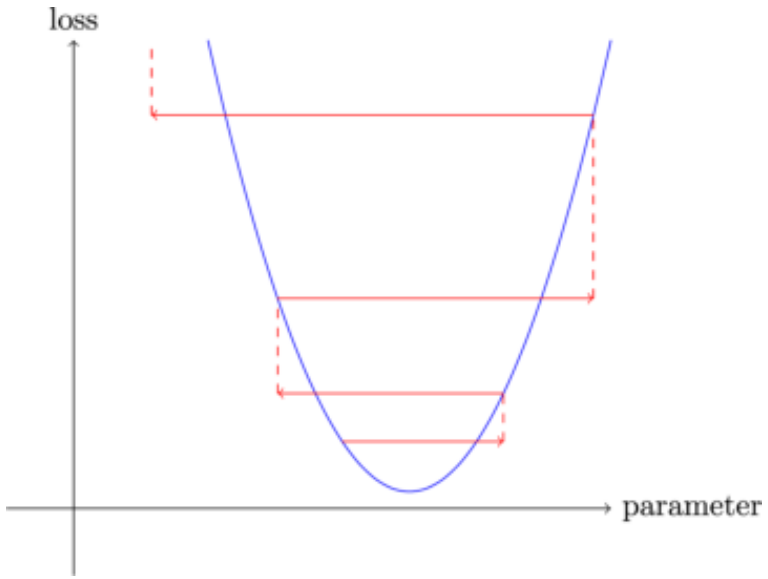
batch online mini-batch

we say we have trained one "epoch" if we have updated the parameters according to all the training examples once

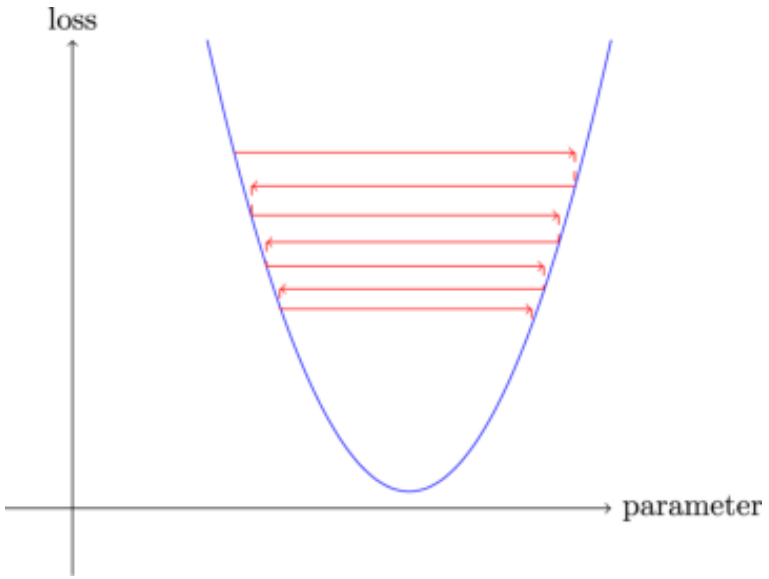
# learning rate




low LR




high LR




bouncy LR

# learning rate

- What is a good learning rate?
  - neither too low (slow convergence) nor too high (divergent)

- learning rate finder `torch.optim.lr_scheduler.LambdaLR`
  - start with a very very small learning rate
  - iterate the following
    - use the current learning rate for one mini-batch
    - then increase the learning rate by a certain percentage
    - if the loss gets worse, freeze the learning rate

# learning rate

- “1 cycle” training: a schedule separated into two phases
  - warmup: LR grows from min to max
  - annealing: LR decreases back to min

`torch.optim.lr_scheduler.OneCycleLR`

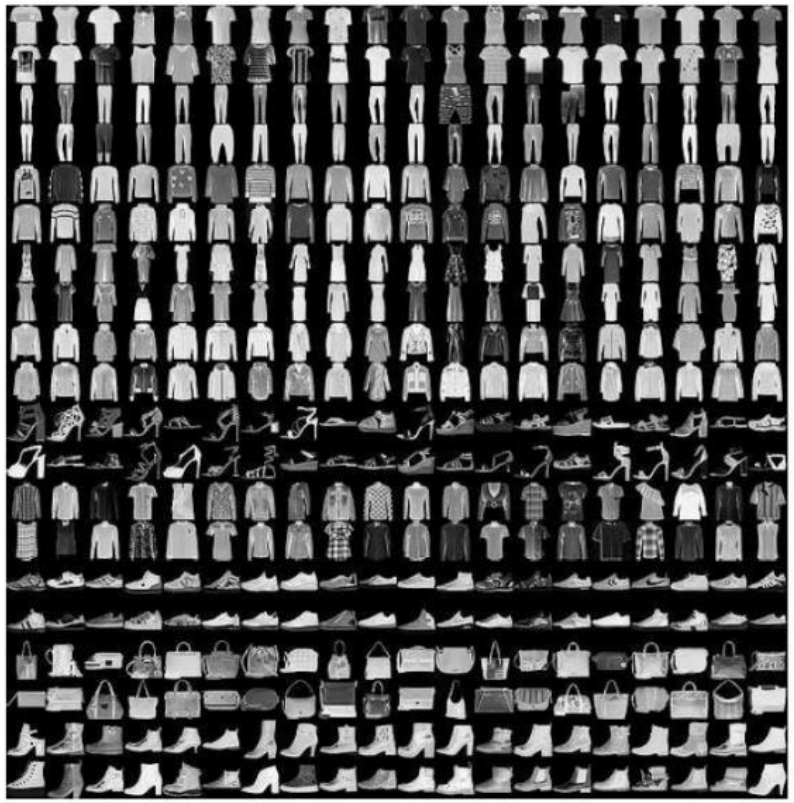
- intuition
  - at the beginning, we use small LR to avoid divergence
  - at the end, we want to find the location carefully with small LR
  - in the middle, we want to move quickly with large LR

# 1.4 PyTorch: an example

# playgrounds


# Let's familiarize ourselves with NN!

- We will train a fully connected network with FashionMNIST. (click here)

| Label | Description | Examples |
|---|---|---|
| 0 | T-Shirt/Top |  |
| 1 | Trouser | |
| 2 | Pullover | |
| 3 | Dress | |
| 4 | Coat | |
| 5 | Sandals | |
| 6 | Shirt | |
| 7 | Sneaker | |
| 8 | Bag | |
| 9 | Ankle boots | |

# 1.5 differentiation

# backpropagation

- for scalars $x, y$: if $y = g(x)$ and $z = f\big(g(x)\big) = f(y)$, then
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- for vectors $\boldsymbol{x}, \boldsymbol{y}$: if $\boldsymbol{y} = g(\boldsymbol{x})$ and $z = f(\boldsymbol{y})$, then
$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

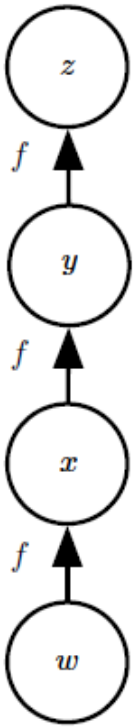
or equivalently,

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} z$$

- for general tensors X, Y: if $\mathrm{Y} = g(\mathrm{X})$ and $z = f(\mathrm{Y})$, then
$$\nabla_{\mathrm{X}} z = \sum_j (\nabla_{\mathrm{X}} \mathrm{Y}_j)\frac{\partial z}{\partial \mathrm{Y}_j}$$

# backpropagation

- symbol-to-number
  differentiation methods take the input of the computational graph and numerical values and return a set of numerical values.

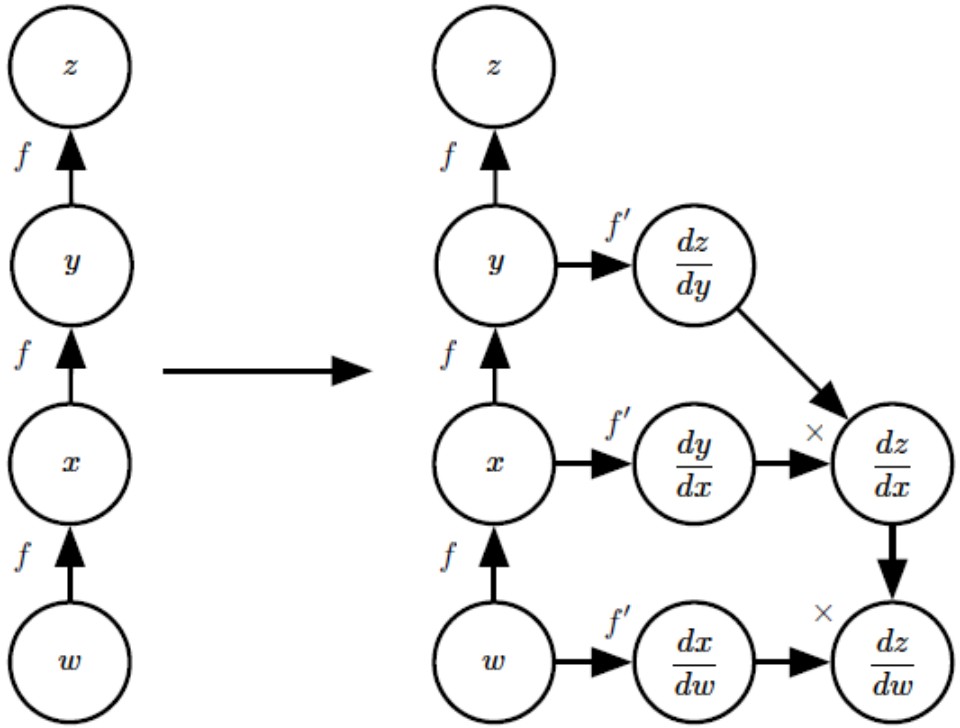



$$\frac{\partial z}{\partial w}$$
$$= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$
$$= f'(y)f'(x)f'(w)$$
$$= f'(f(f(w)))f'(f(w))f'(w).$$

PyTorch

# backpropagation

- symbol-to-symbol differentiation creates a symbolic description of the derivatives.

# 1.6 two important regularization methods

# batch normalization

- the activation values from the second last layer of a neural network

# batch normalization

- idea of fixing this: maintain a good distribution of activations throughout training

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

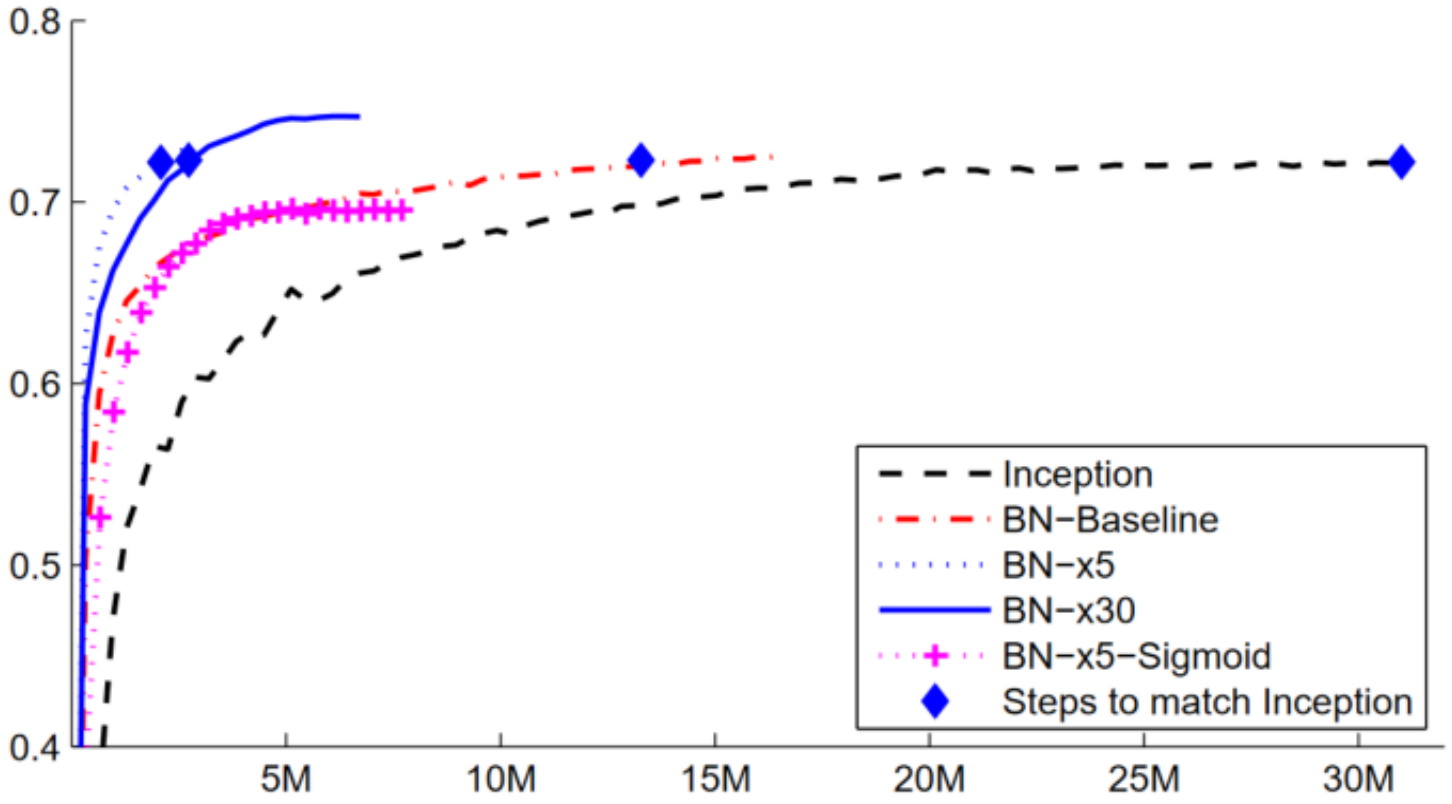
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

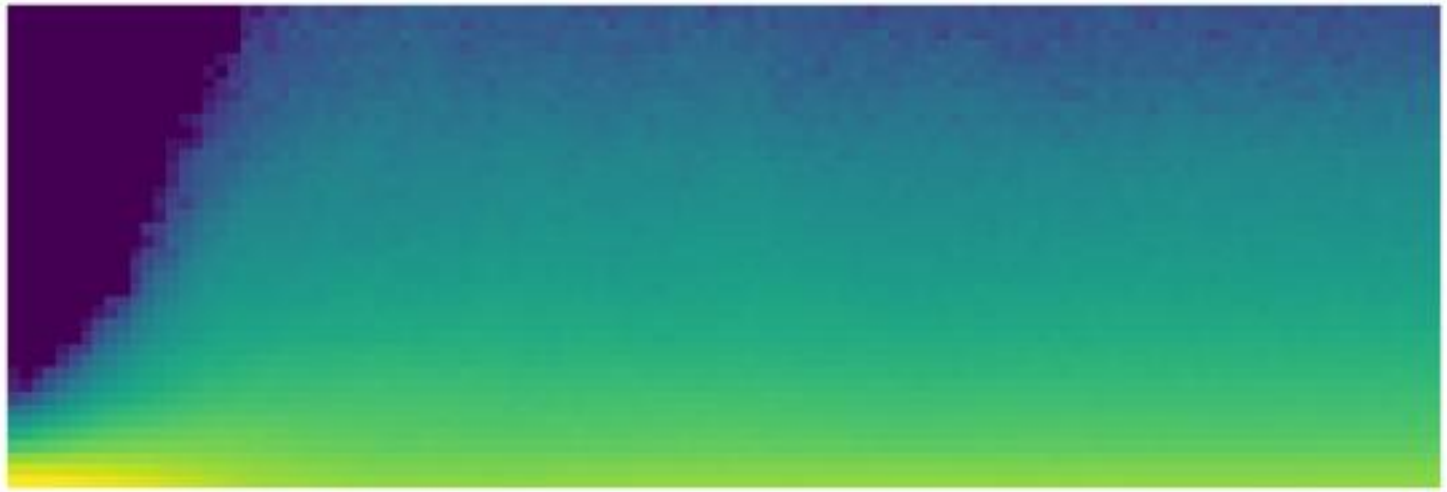
# batch normalization



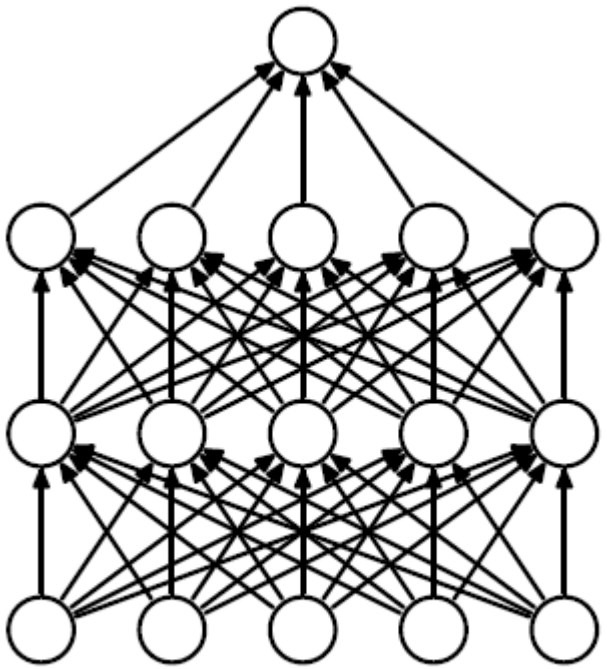

validation accuracy of different models on ImageNet dataset

# batch normalization

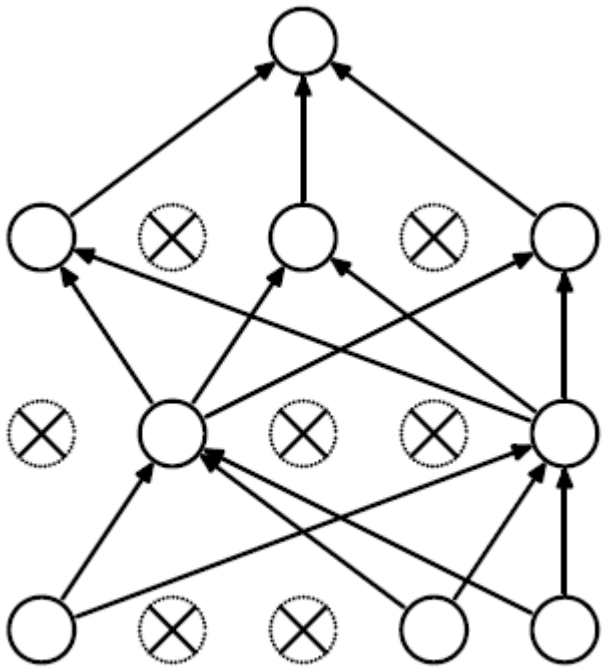
- after the batch normalization layer is added



# dropout

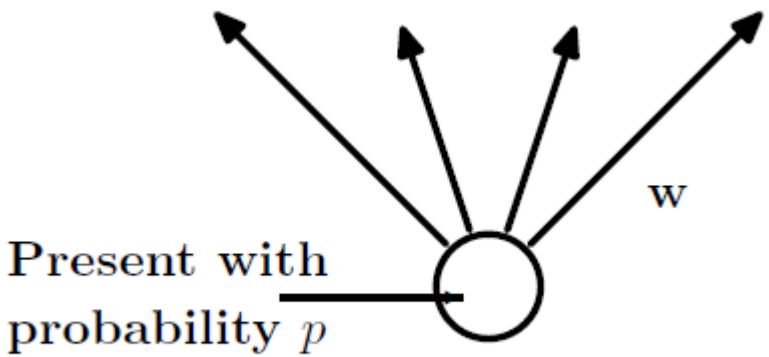


(a) Standard Neural Net

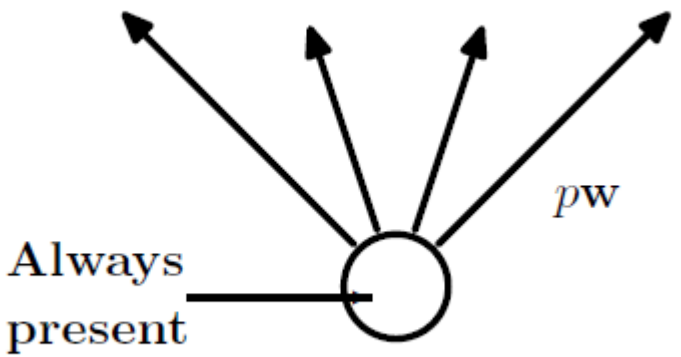


(b) After applying dropout.

# dropout




(a) At training time



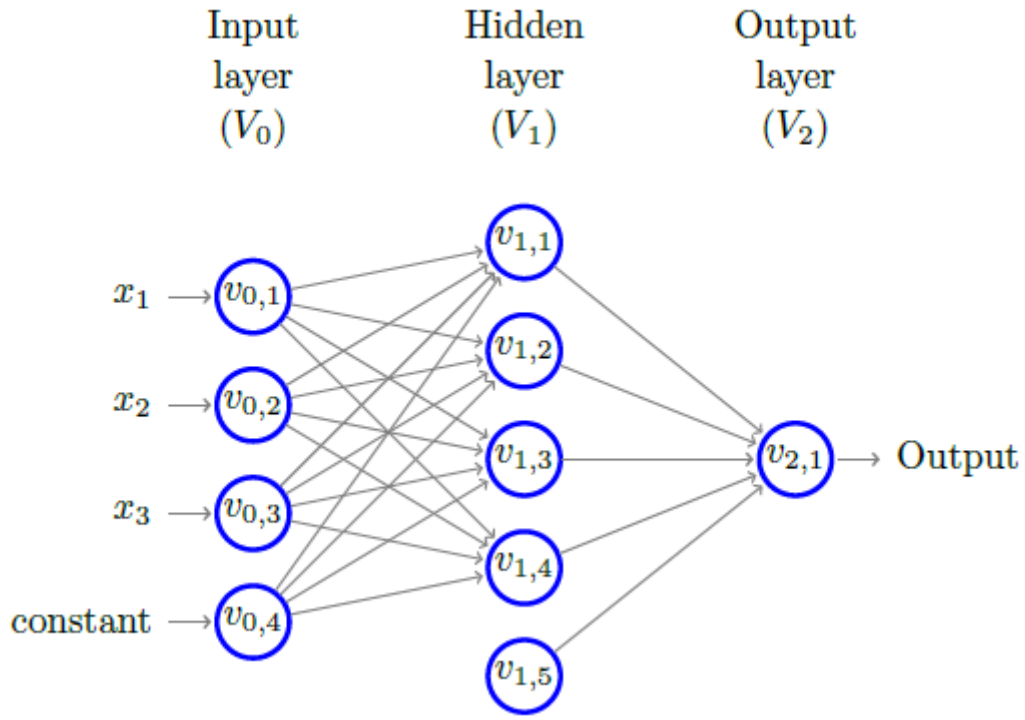

(b) At test time

# Let’s look at the performance of BN and dropout!

- click here

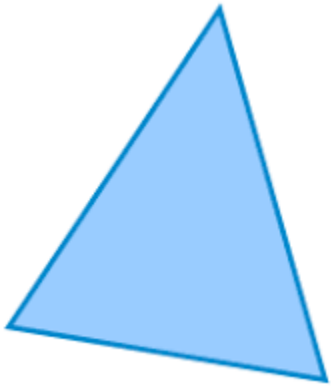
# 1.7 universal approximation (why can NN approximate the fitting function?)

# hypothesis class of NN

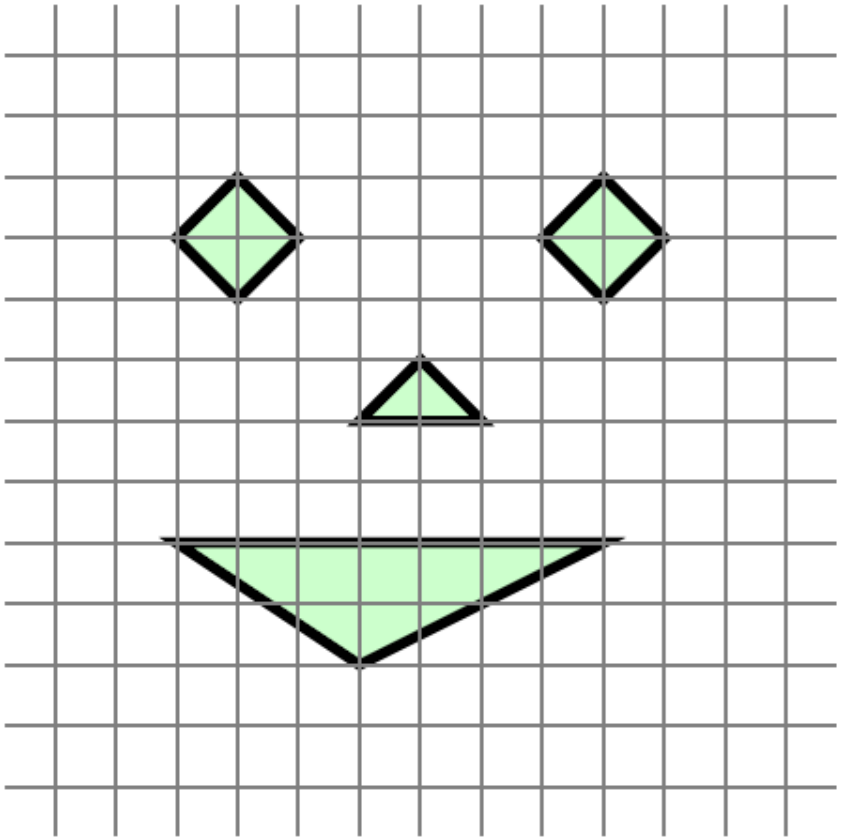



given the architecture of a fully-connected network, it represents a hypothesis class whose members depend on the weights of the network

# What NN can represent these functions?



# What NN can represent these functions?



# Let's start simple…

Claim:

For every $n$, there exists a hypothesis class of fully-connected networks that contains all functions from $\{\pm 1\}^n$ to $\{\pm 1\}$.

# universal approximation

Let $f: [-1,1]^n \to [-1,1]$ be a Lipschitz function. Given any $\epsilon > 0$, there is a neural network $N: [-1,1]^n \to [-1,1]$ with a sigmoid activation function, such that for every $\boldsymbol{x} \in [-1,1]^n$, it holds that

$$|f(\boldsymbol{x}) - N(\boldsymbol{x})| \leq \epsilon.$$

# Thank you!

# Reference


- Ch 6, 7.12, 8.7.1, *Deep Learning*.
- Ch 5.1, 5.2, *Pattern Recognition and Machine Learning*.
- Ch 20.3, *Understanding Machine Learning*.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4), 303-314.
- https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html
- https://github.com/KushajveerSingh/Deep-Learning-Notebooks/blob/master/Leslie%20N.%20Smith%20paper%20notebooks%20(fastai)/main.ipynb