# Dive into Deep Learning

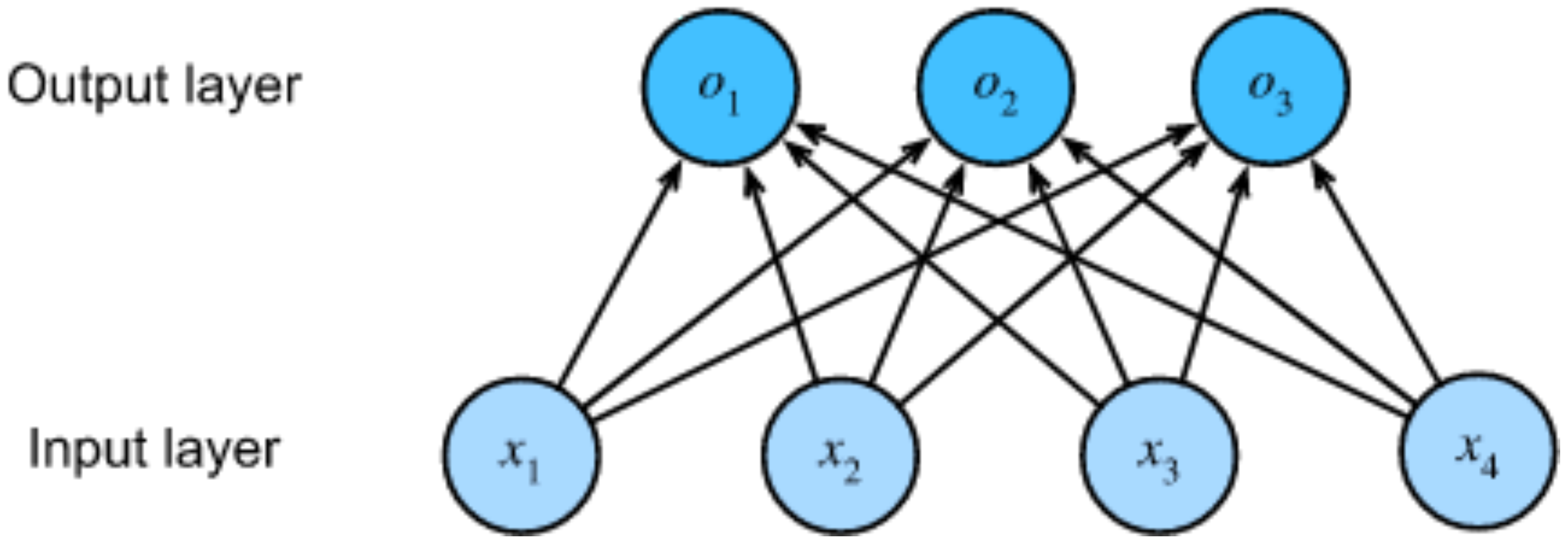
## Chapter 4. Multilayer Perceptrons

Wayne L, Oct 18, 2020

# Previously

## Perceptron ( Linear = Single = Simple = Feedforward )

- This model mapped our inputs directly to our outputs via single affine transformation, followed by a softmax operation.



$$y = \mathbf{w}\mathbf{x} + b$$

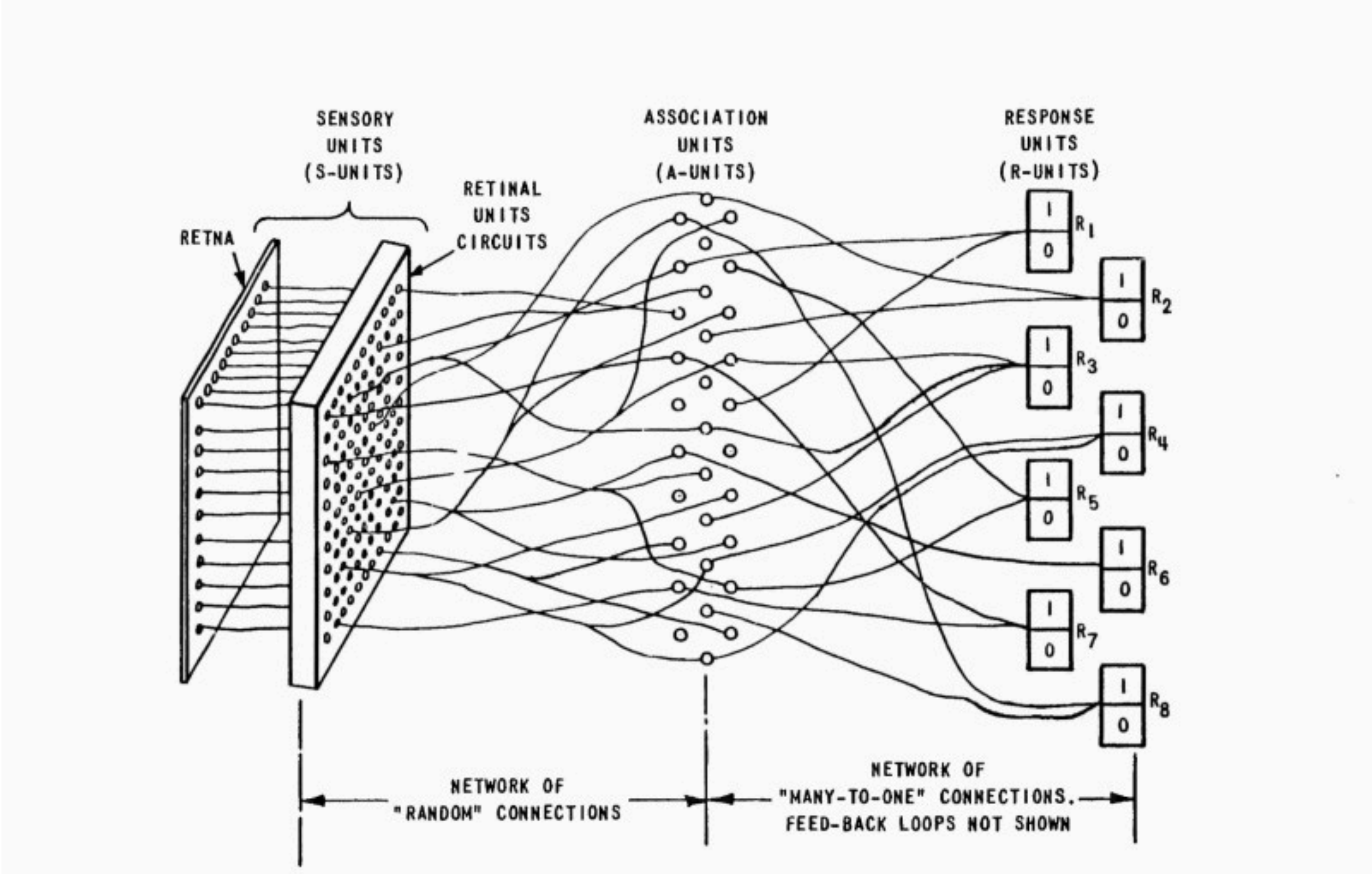
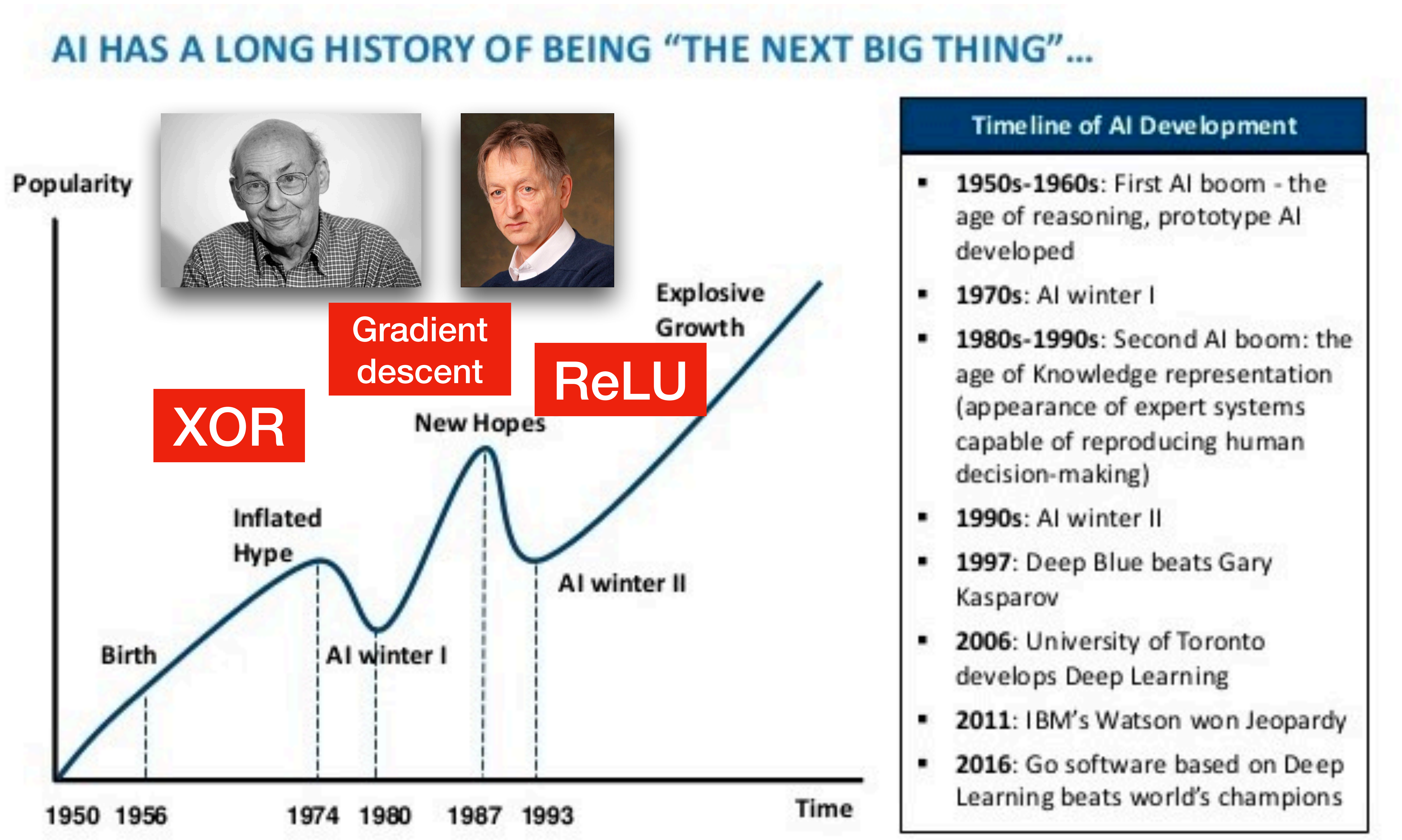

SENSORY UNITS (S-UNITS)
RETINAL UNITS CIRCUITS
ASSOCIATION UNITS (A-UNITS)
RESPONSE UNITS (R-UNITS)
RETNA
$R_1$
$R_2$
$R_3$
$R_4$
$R_5$
$R_6$
$R_7$
$R_8$
1
0
NETWORK OF "RANDOM" CONNECTIONS
NETWORK OF "MANY-TO-ONE" CONNECTIONS. FEED-BACK LOOPS NOT SHOWN

Figure I ORGANIZATION OF THE MARK I PERCEPTRON

Mark 1 Perceptron (1959), Frank Rosenblatt

# Previously

## AI History



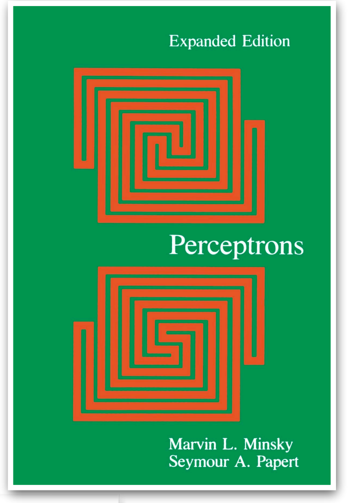
AI HAS A LONG HISTORY OF BEING "THE NEXT BIG THING"...

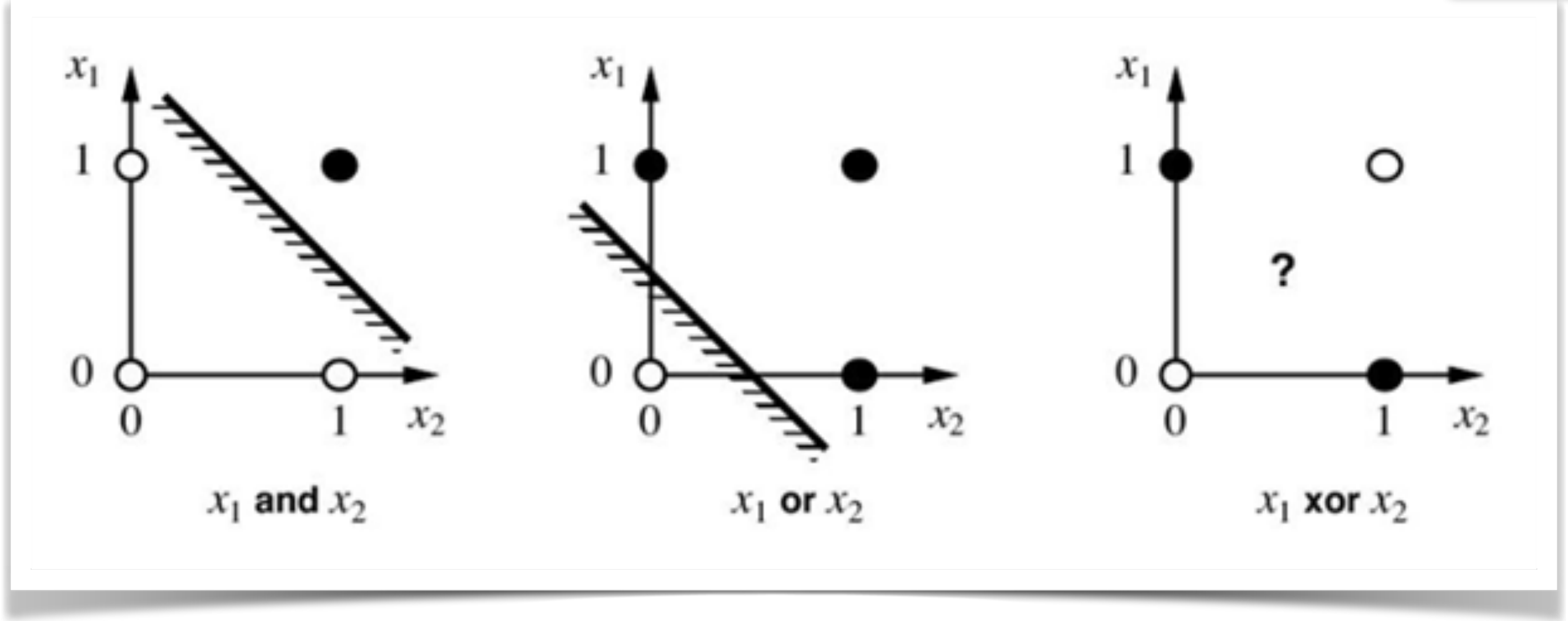
Timeline of AI Development

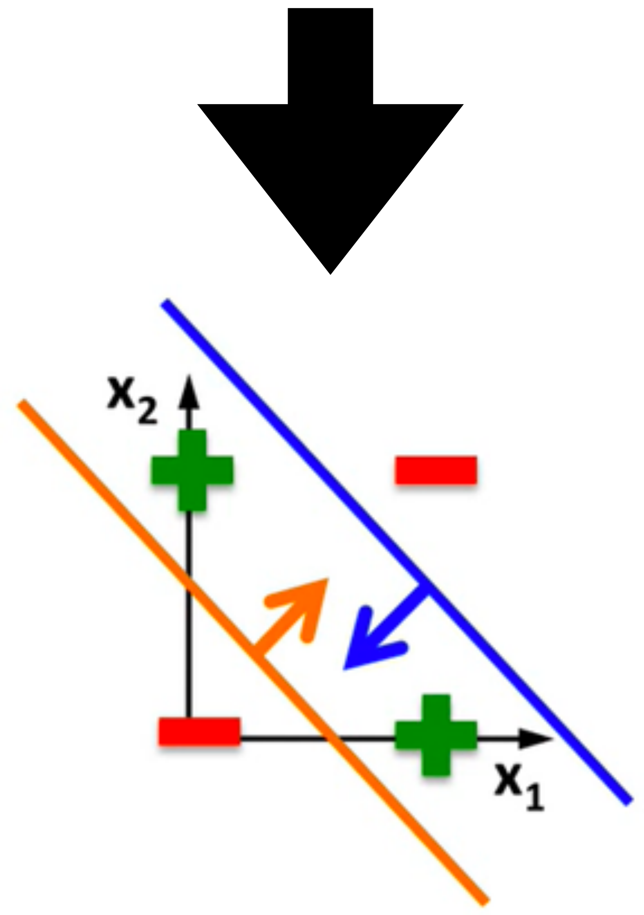
- **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
- **1970s**: AI winter I
- **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
- **1990s**: AI winter II
- **1997**: Deep Blue beats Gary Kasparov
- **2006**: University of Toronto develops Deep Learning
- **2011**: IBM's Watson won Jeopardy
- **2016**: Go software based on Deep Learning beats world's champions

XOR

Perceptrons (1969), Marvin minsky







No one on earth had found a viable way to train

# Multilayer Perceptrons

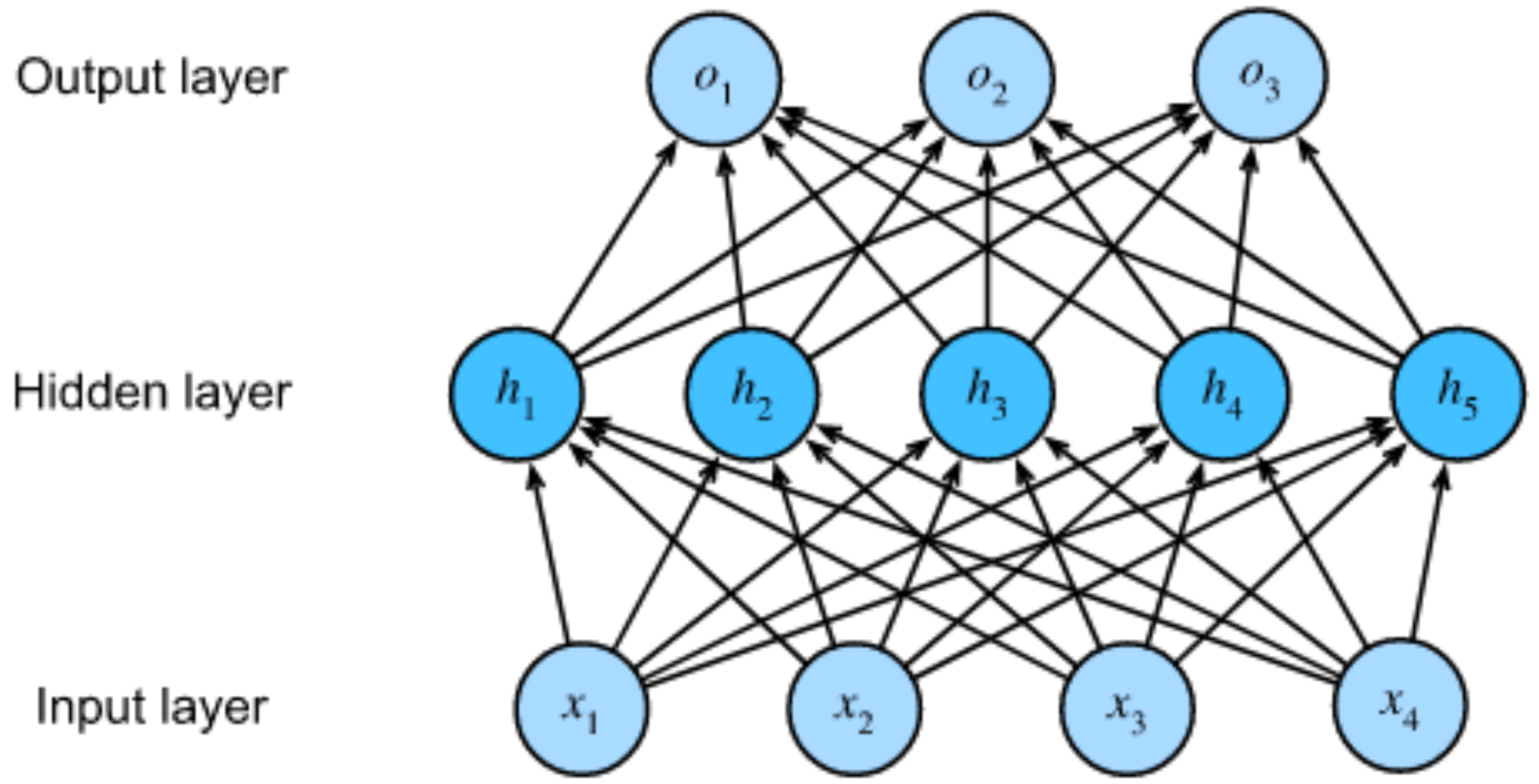
## Perceptrons ( Multi = Deep feedforward)

- The goal of a feedforward network is to approximate some function $f^*$.

- For example, for a classifier, $y = f^*(x)$ maps an input $x$ to a category $y$.
  A feedforward network defines a mapping $\mathbf{y} = f(x; \theta)$ and learns the value of the parameters $\theta$ that result in the best function approximation.

- ***Universal approximation theorem*** means that regardless of what function we are trying to learn, we know that a large MLP will be able to represent this function.

$$f(x) = f^*(x) \approx y$$



$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

$$\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$$

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

$$\mathbf{O} = (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W} + \mathbf{b}$$

Even though MLP is going deeper, it can be equivalent with single-layer model !

# Multilayer Perceptrons

## From Linear to Nonlinear

- In order to realize the potential of multilayer architectures, we need one more key ingredient: a nonlinear *activation function* $\sigma$ to be more ***expressive***.
- MLPs are ***universal approximators***, however, it doe not mean that we can solve all of problems with MLPs. In fact, we can approximate many functions much more compactly by using deeper (or wider) networks.
- Each neuron acts as a ***linear SVM***, however, …
  - its output is not interpreted immediately,
  - but it becomes a new feature,
  - to be forwarded to the next layer for further analysis **#SVMcascade**

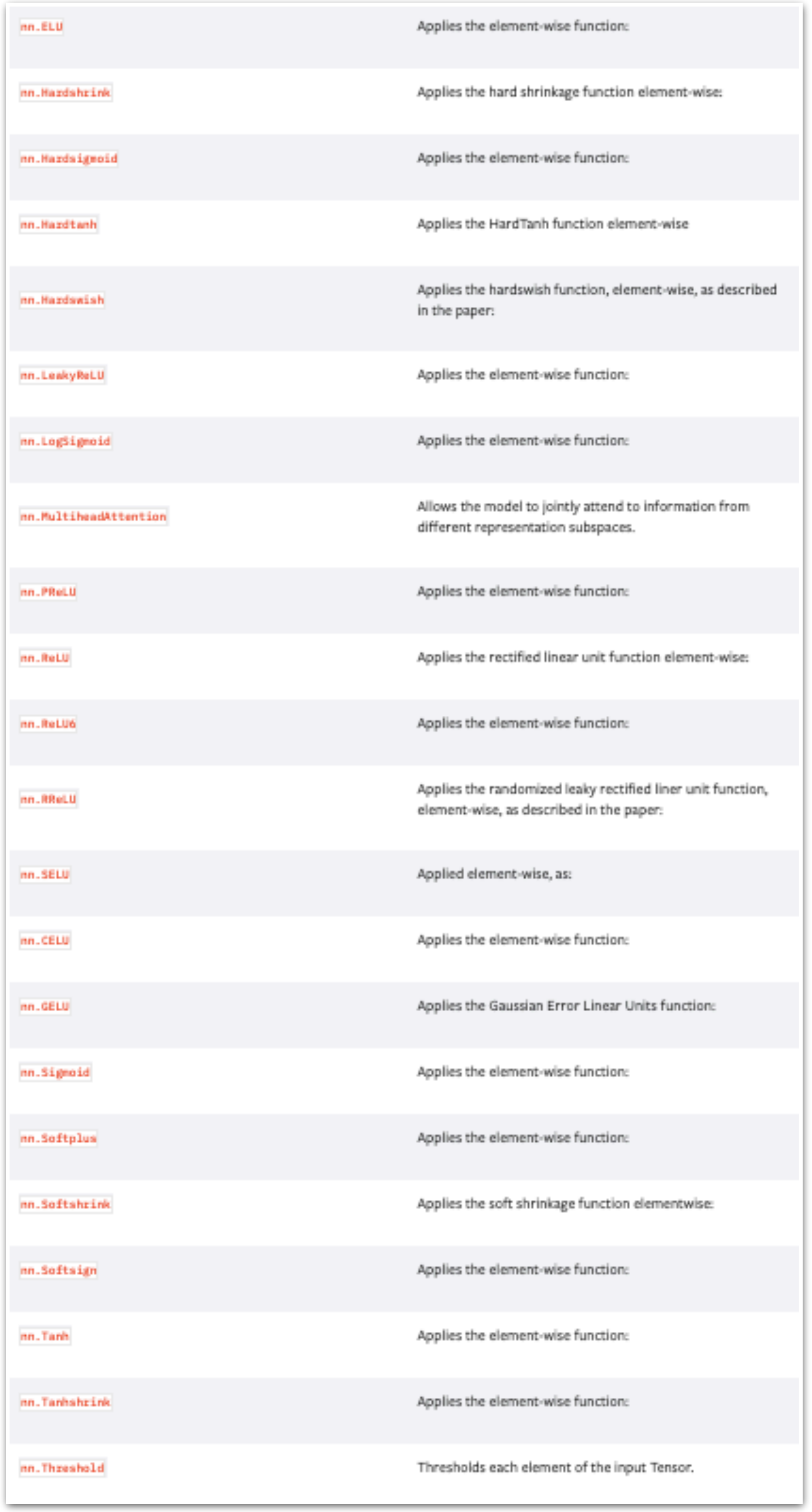
$$\mathbf{H}^{(1)} = \sigma_1(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$$

$$\mathbf{H}^{(2)} = \sigma_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)})$$

# Multilayer Perceptrons

## Activation function

- ***Activation function*** decide whether a neuron should be activated or not by calculating the weighted sum and further adding bias with it.
- They are ***differentiable*** operators to transform input signal to outputs, while most of them add non-linearity.



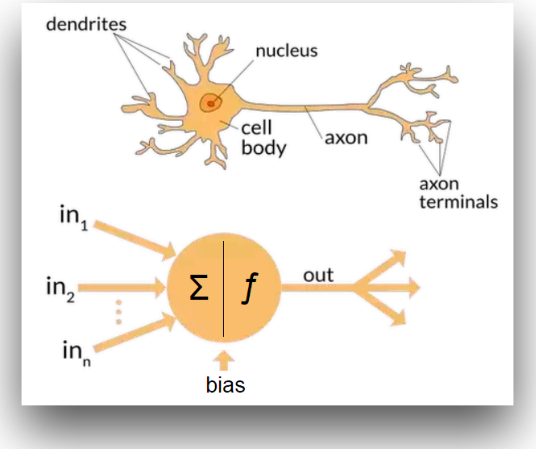
| | |
|---|---|
| `nn.ELU` | Applies the element-wise function: |
| `nn.Hardshrink` | Applies the hard shrinkage function element-wise: |
| `nn.Hardsigmoid` | Applies the element-wise function: |
| `nn.Hardtanh` | Applies the HardTanh function element-wise |
| `nn.Hardswish` | Applies the hardswish function, element-wise, as described in the paper: |
| `nn.LeakyReLU` | Applies the element-wise function: |
| `nn.LogSigmoid` | Applies the element-wise function: |
| `nn.MultiheadAttention` | Allows the model to jointly attend to information from different representation subspaces. |
| `nn.PReLU` | Applies the element-wise function: |
| `nn.ReLU` | Applies the rectified linear unit function element-wise: |
| `nn.ReLU6` | Applies the element-wise function: |
| `nn.RReLU` | Applies the randomized leaky rectified liner unit function, element-wise, as described in the paper: |
| `nn.SELU` | Applied element-wise, as: |
| `nn.CELU` | Applies the element-wise function: |
| `nn.GELU` | Applies the Gaussian Error Linear Units function: |
| `nn.Sigmoid` | Applies the element-wise function: |
| `nn.Softplus` | Applies the element-wise function: |
| `nn.Softshrink` | Applies the soft shrinkage function elementwise: |
| `nn.Softsign` | Applies the element-wise function: |
| `nn.Tanh` | Applies the element-wise function: |
| `nn.Tanhshrink` | Applies the element-wise function: |
| `nn.Threshold` | Thresholds each element of the input Tensor. |

https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity

# Multilayer Perceptrons

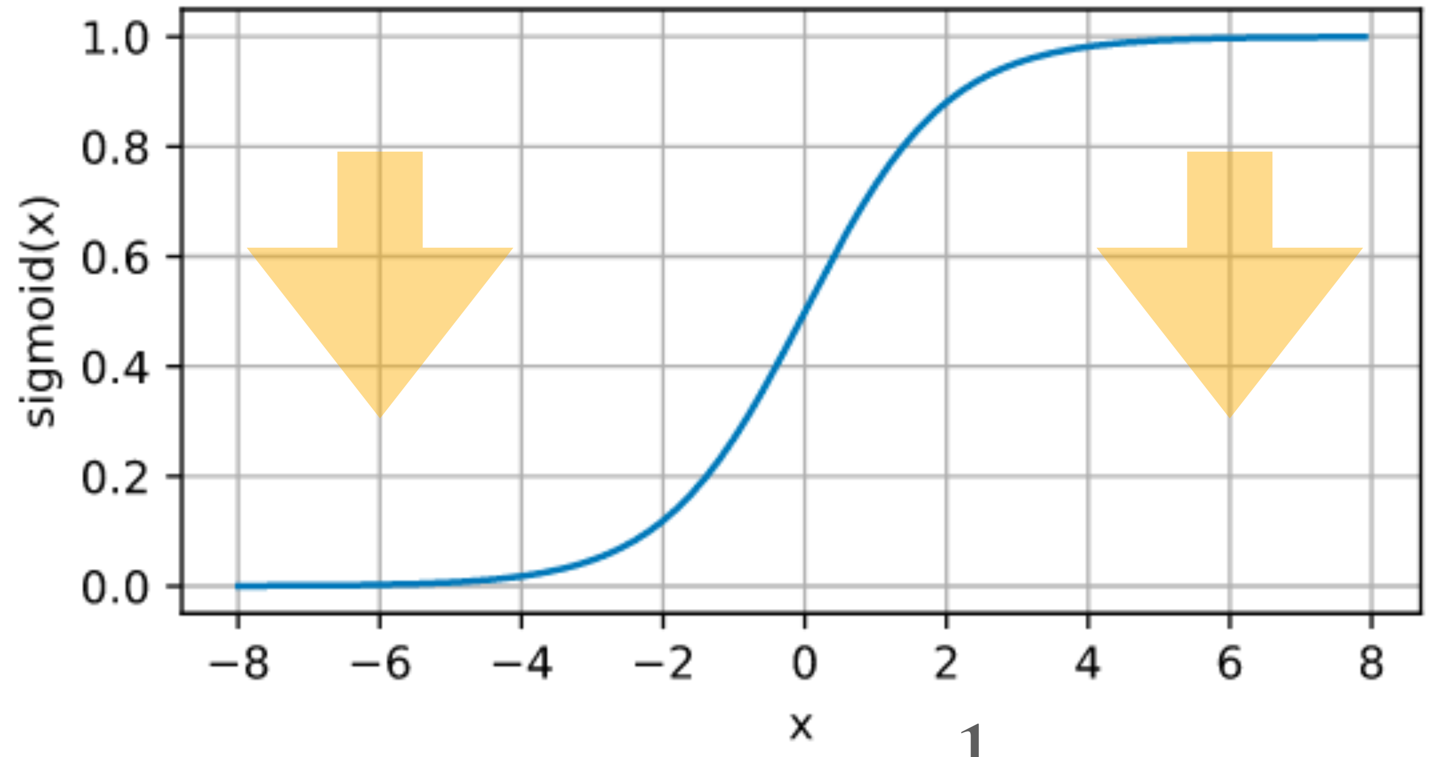
## Sigmoid

- In the earliest neural networks, scientists were interested in modeling biological neurons which either fire or do not fire.

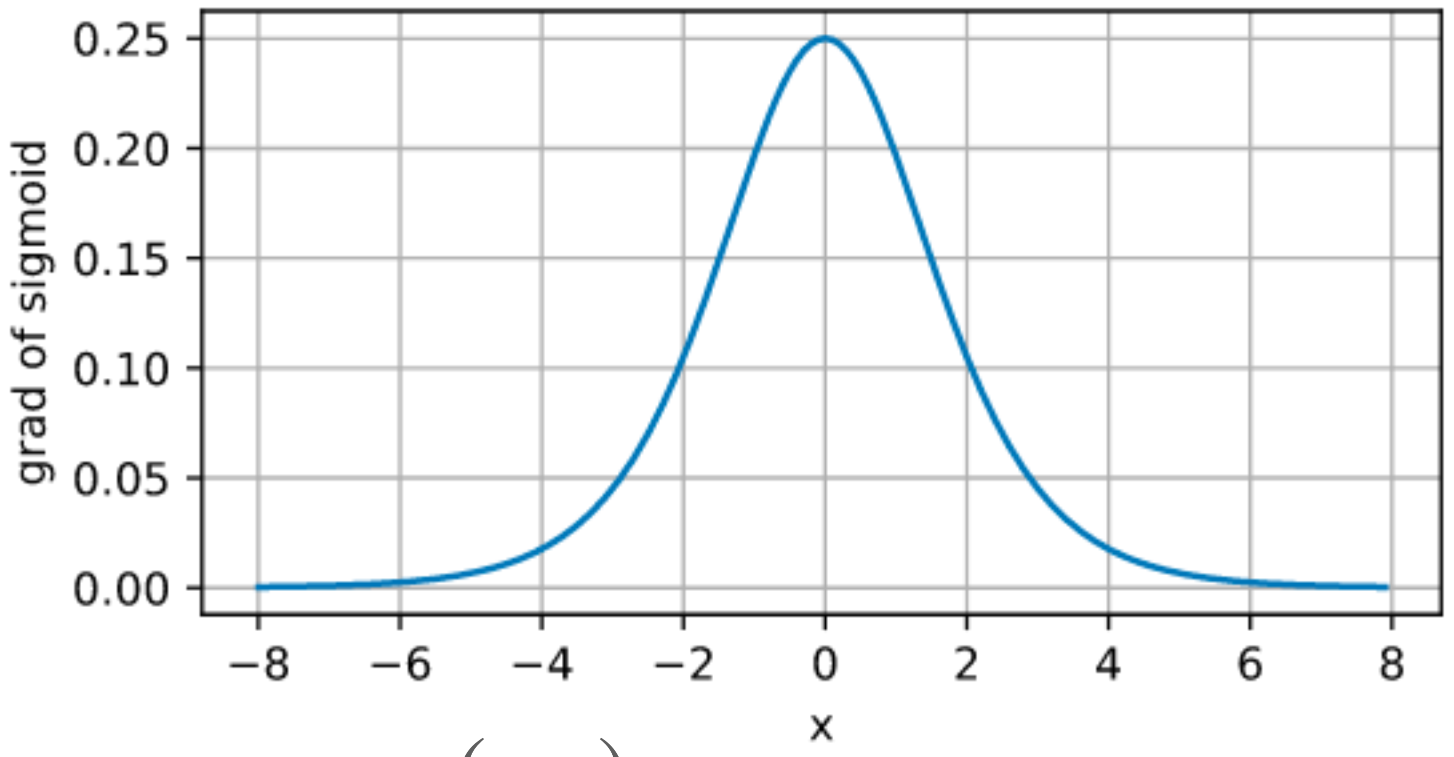


- When attention shifted to gradient based learning, the sigmoid funciton was a natural choice because it is a smooth, differentiable approximation to a thresholding unit.

- widely used as activation functions on the output units, when we want to interpret the outputs as probabilities for binary classification problems (special case of the softmax).



$$sigmoid(x) = \frac{1}{1 + exp(-x)}$$



vanishing gradient

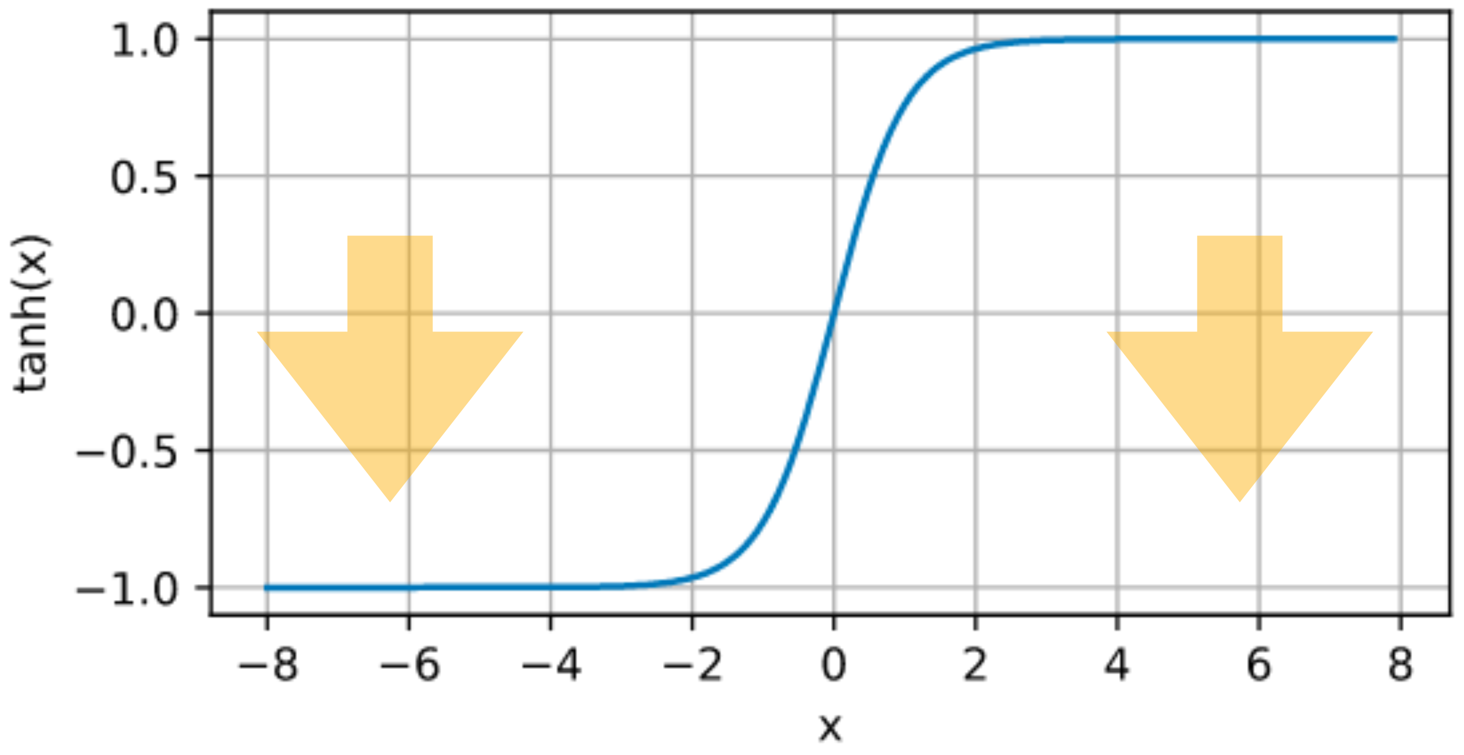
$$\frac{d}{dx} sigmoid(x) = \frac{exp(-x)}{(1 + exp(-x))^2} = sigmoid(x)(1 - sigmoid(x))$$

# Multilayer Perceptrons

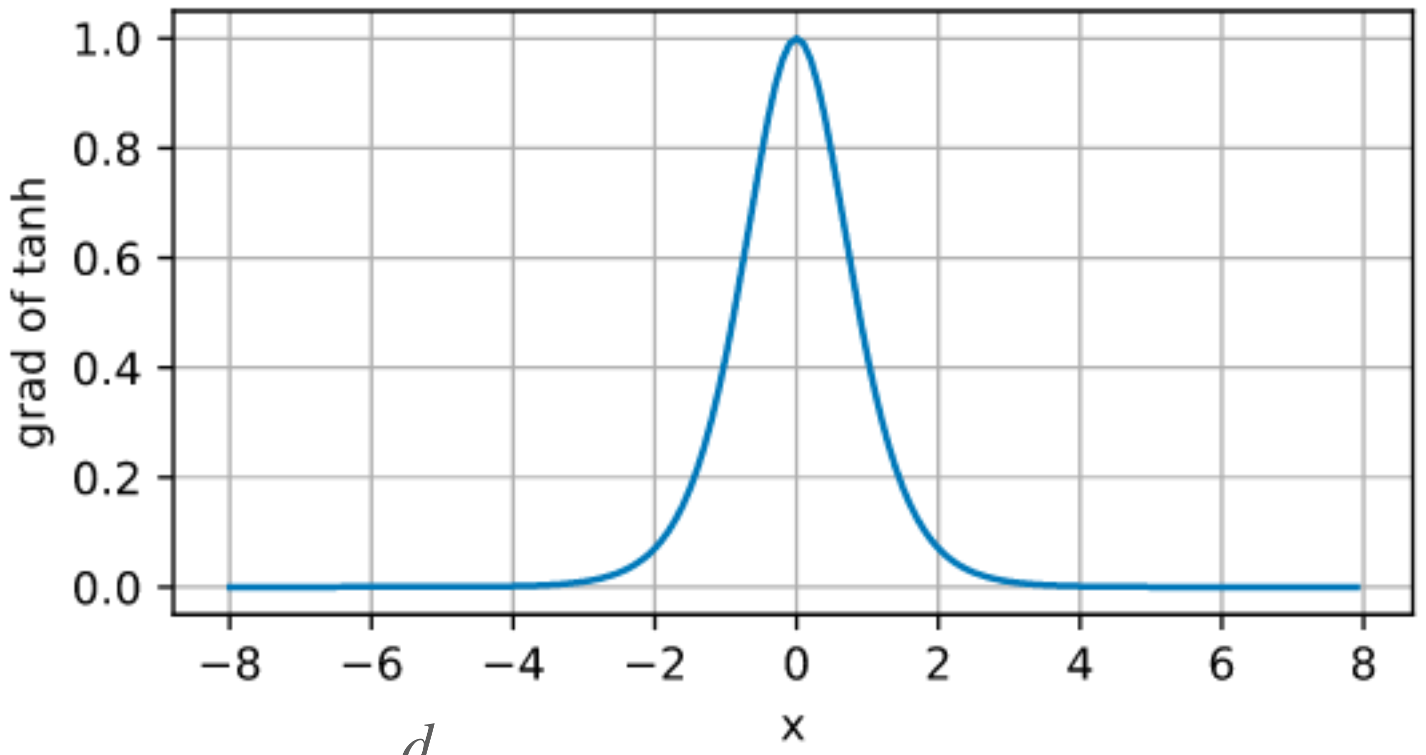
## Tanh (hyperbolic tangent)

- Like the sigmoid function, the tanh function also squashes its inputs, transforming them into elements on the interval between -1 and 1.

- Although the shape of the function is similar to that of the sigmoid function, the tanh function exhibits point symmetry about the origin of the coordinate system.



$$tanh(x) = \frac{1 - exp(-2x)}{1 + exp(-2x)}$$



$$\frac{d}{dx}tanh(x) = 1 - tanh^2(x)$$

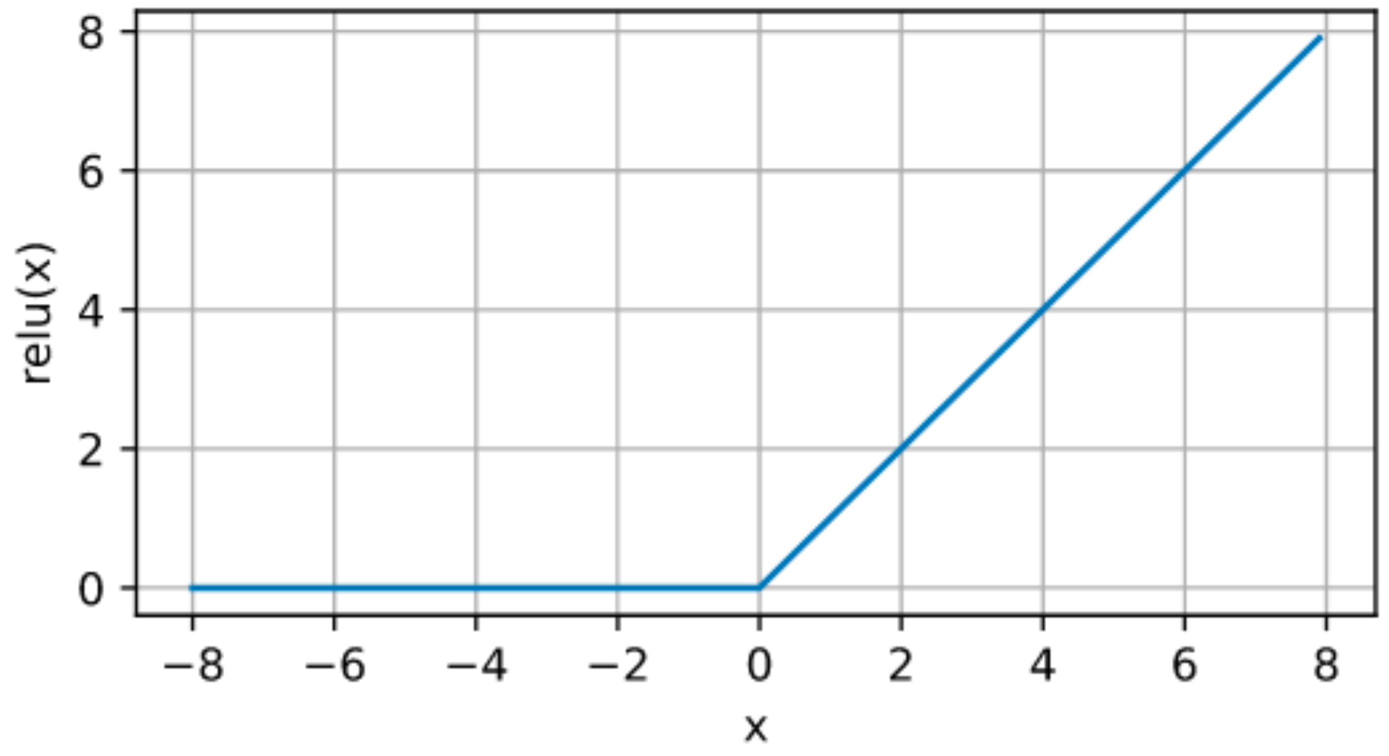
vanishing gradient

# Multilayer Perceptrons

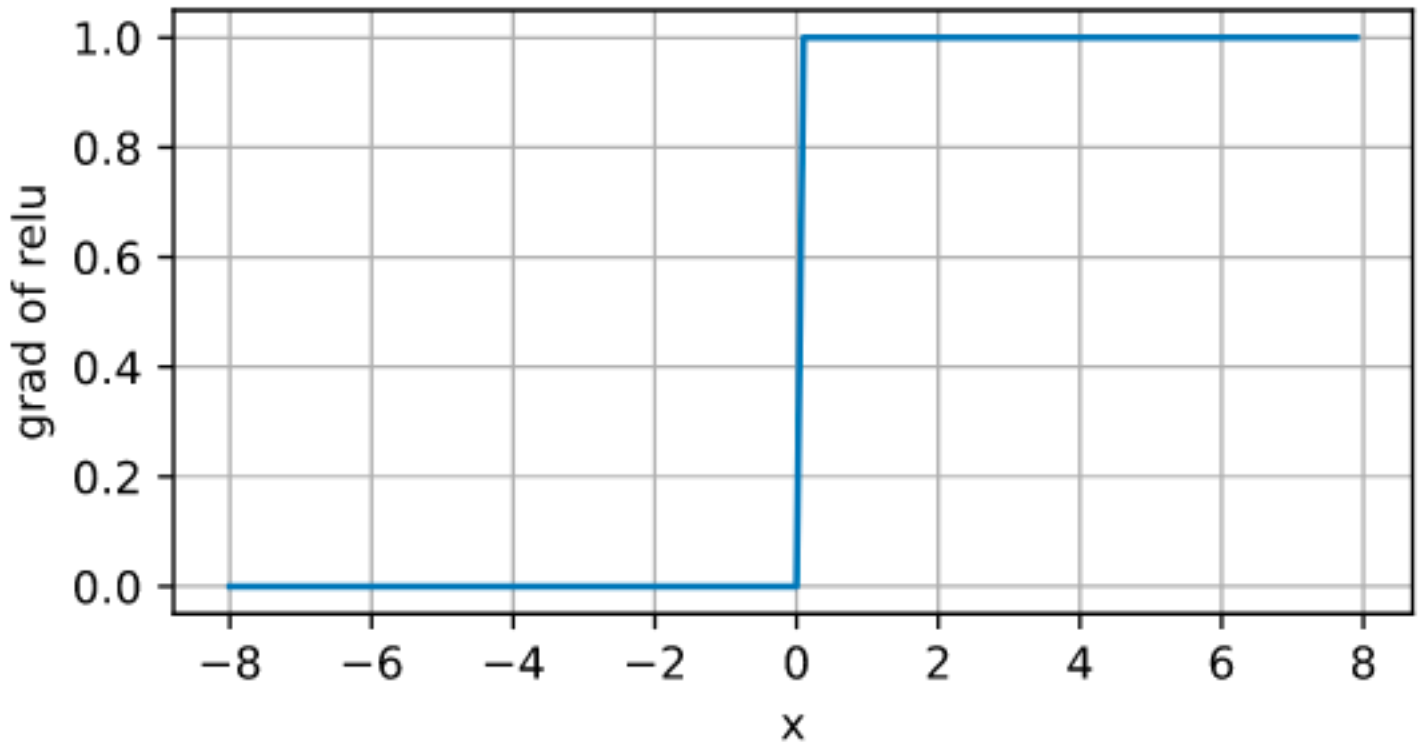
## ReLU (rectified linear unit)

- The reason for using ReLU is that its derivates are particularly well behaved: either they vanish or they just let the argument through.
- This makes optimization better behaved and it mitigated the well-documented problem of vanishing gradients that plagued previous versions of neural networks.

$$pReLU(x) = max(0,x) + \alpha min(0,x)$$



$$ReLU(x) = max(x,0)$$

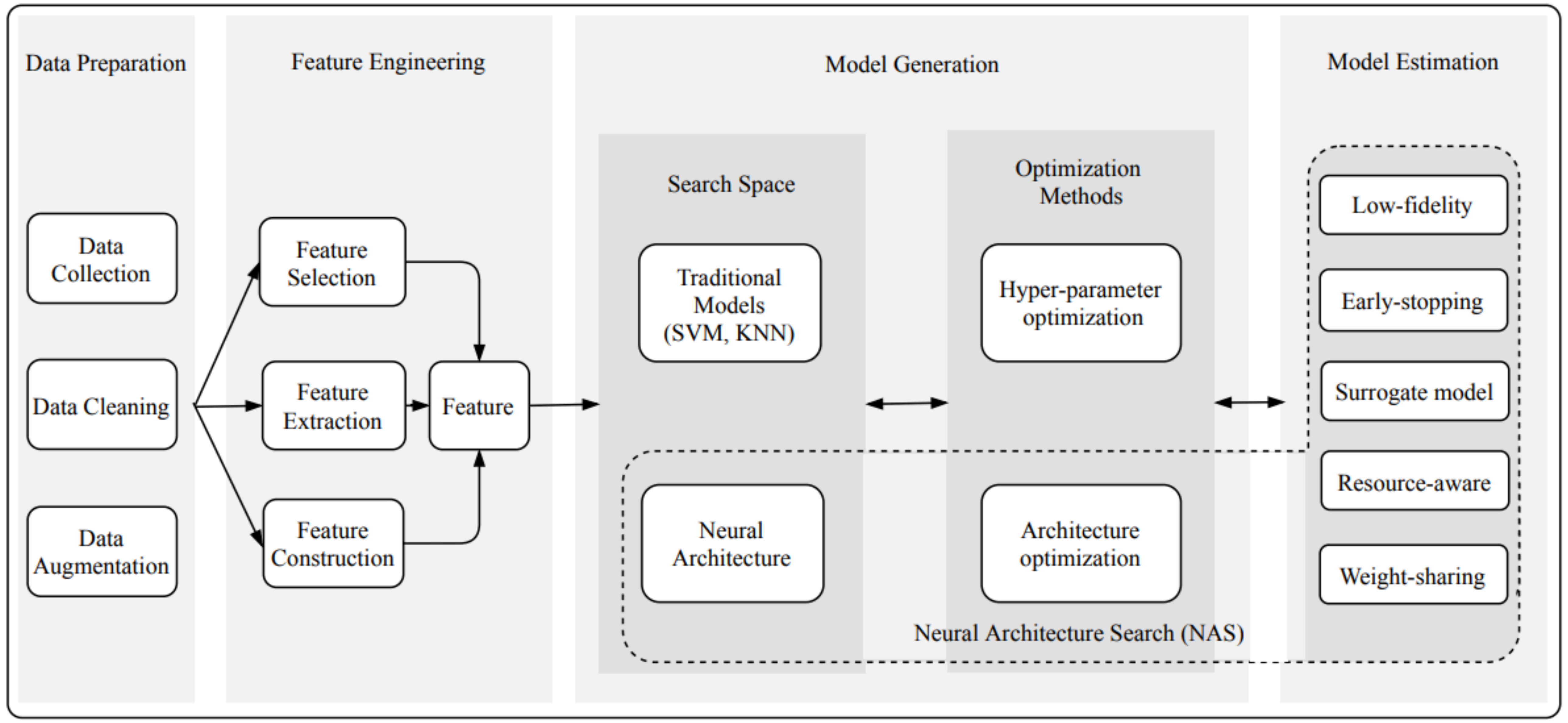


“Predicting in advance which will work best is usually impossible.”

**Deep Learning (Chapter 6) , Ian Goodfellow et al.**

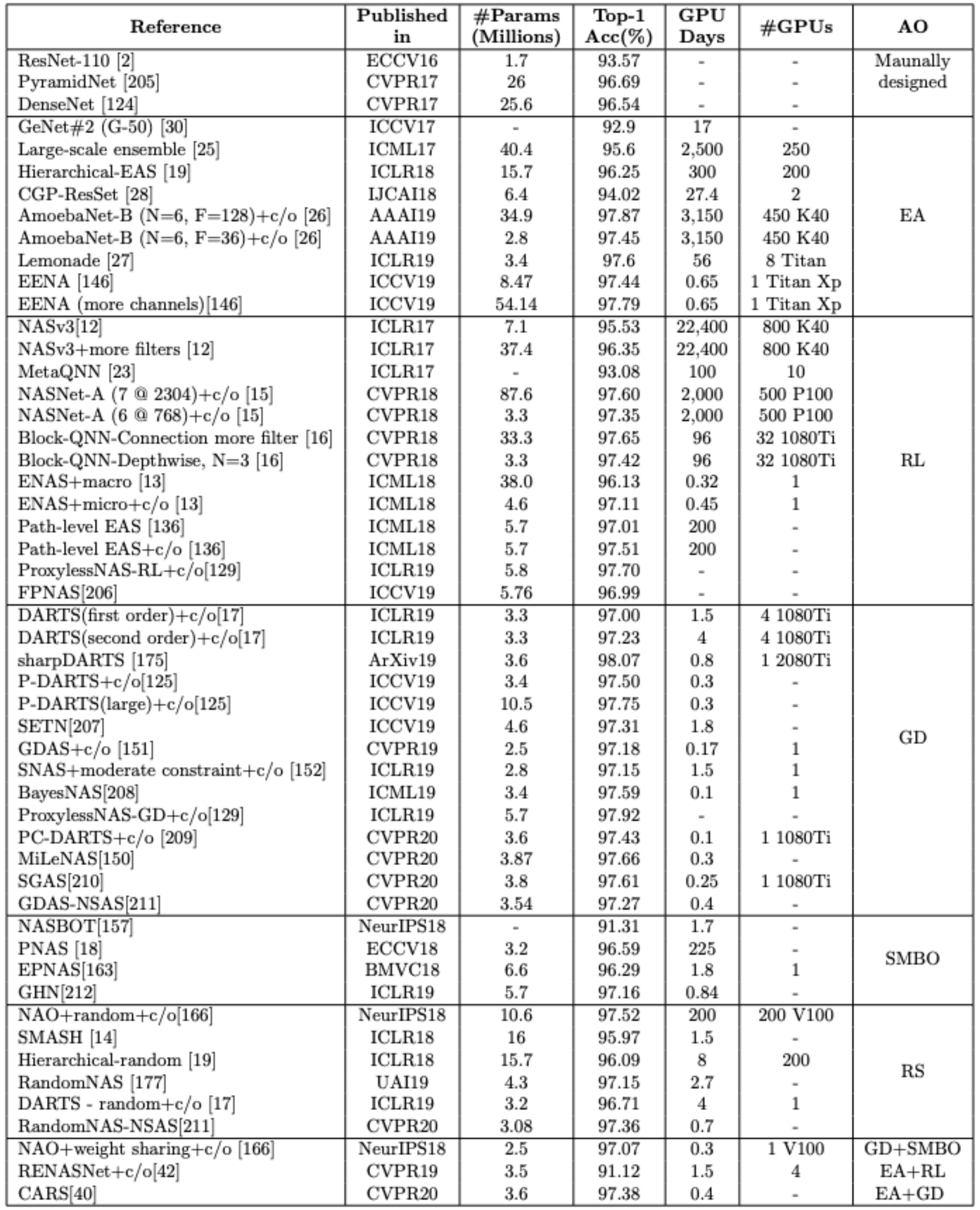
# Auto ML

**#AutoML: automated machine learning**
**#NAS: neural architecture search**
**#HPO: hyper-parameter optimization**



(1) https://github.com/microsoft/nni
(2) https://github.com/automl/auto-sklearn
(3) https://github.com/google/automl

CIFAR 10 (https://www.cs.toronto.edu/~kriz/cifar.html)



| Reference | Published in | #Params (Millions) | Top-1 Acc(%) | GPU Days | #GPUs | AO |
|---|---|---|---|---|---|---|
| ResNet-110 [2] | ECCV16 | 1.7 | 93.57 | - | - | Maunally designed |
| PyramidNet [205] | CVPR17 | 26 | 96.69 | - | - | |
| DenseNet [124] | CVPR17 | 25.6 | 96.54 | - | - | |
| GeNet#2 (G-50) [30] | ICCV17 | - | 92.9 | 17 | - | EA |
| Large-scale ensemble [25] | ICML17 | 40.4 | 95.6 | 2,500 | 250 | |
| Hierarchical-EAS [19] | ICLR18 | 15.7 | 96.25 | 300 | 200 | |
| CGP-ResSet [28] | IJCAI18 | 6.4 | 94.02 | 27.4 | 2 | |
| AmoebaNet-B (N=6, F=128)+c/o [26] | AAAI19 | 34.9 | 97.87 | 3,150 | 450 K40 | |
| AmoebaNet-B (N=6, F=36)+c/o [26] | AAAI19 | 2.8 | 97.45 | 3,150 | 450 K40 | |
| Lemonade [27] | ICLR19 | 3.4 | 97.6 | 56 | 8 Titan | |
| EENA [146] | ICCV19 | 8.47 | 97.44 | 0.65 | 1 Titan Xp | |
| EENA (more channels)[146] | ICCV19 | 54.14 | 97.79 | 0.65 | 1 Titan Xp | |
| NASv3[12] | ICLR17 | 7.1 | 95.53 | 22,400 | 800 K40 | RL |
| NASv3+more filters [12] | ICLR17 | 37.4 | 96.35 | 22,400 | 800 K40 | |
| MetaQNN [23] | ICLR17 | - | 93.08 | 100 | 10 | |
| NASNet-A (7 @ 2304)+c/o [15] | CVPR18 | 87.6 | 97.60 | 2,000 | 500 P100 | |
| NASNet-A (6 @ 768)+c/o [15] | CVPR18 | 3.3 | 97.35 | 2,000 | 500 P100 | |
| Block-QNN-Connection more filter [16] | CVPR18 | 33.3 | 97.65 | 96 | 32 1080Ti | |
| Block-QNN-Depthwise, N=3 [16] | CVPR18 | 3.3 | 97.42 | 96 | 32 1080Ti | |
| ENAS+macro [13] | ICML18 | 38.0 | 96.13 | 0.32 | 1 | |
| ENAS+micro+c/o [13] | ICML18 | 4.6 | 97.11 | 0.45 | 1 | |
| Path-level EAS [136] | ICML18 | 5.7 | 97.01 | 200 | - | |
| Path-level EAS+c/o [136] | ICML18 | 5.7 | 97.51 | 200 | - | |
| ProxylessNAS-RL+c/o[129] | ICLR19 | 5.8 | 97.70 | - | - | |
| FPNAS[206] | ICCV19 | 5.76 | 96.99 | - | - | |
| DARTS(first order)+c/o[17] | ICLR19 | 3.3 | 97.00 | 1.5 | 4 1080Ti | GD |
| DARTS(second order)+c/o[17] | ICLR19 | 3.3 | 97.23 | 4 | 4 1080Ti | |
| sharpDARTS [175] | ArXiv19 | 3.6 | 98.07 | 0.8 | 1 2080Ti | |
| P-DARTS+c/o[125] | ICCV19 | 3.4 | 97.50 | 0.3 | - | |
| P-DARTS(large)+c/o[125] | ICCV19 | 10.5 | 97.75 | 0.3 | - | |
| SETN[207] | ICCV19 | 4.6 | 97.31 | 1.8 | - | |
| GDAS+c/o [151] | CVPR19 | 2.5 | 97.18 | 0.17 | 1 | |
| SNAS+moderate constraint+c/o [152] | ICLR19 | 2.8 | 97.15 | 1.5 | 1 | |
| BayesNAS[208] | ICML19 | 3.4 | 97.59 | 0.1 | 1 | |
| ProxylessNAS-GD+c/o[129] | ICLR19 | 5.7 | 97.92 | - | - | |
| PC-DARTS+c/o [209] | CVPR20 | 3.6 | 97.43 | 0.1 | 1 1080Ti | |
| MiLeNAS[150] | CVPR20 | 3.87 | 97.66 | 0.3 | - | |
| SGAS[210] | CVPR20 | 3.8 | 97.61 | 0.25 | 1 1080Ti | |
| GDAS-NSAS[211] | CVPR20 | 3.54 | 97.27 | 0.4 | - | |
| NASBOT[157] | NeurIPS18 | - | 91.31 | 1.7 | - | SMBO |
| PNAS [18] | ECCV18 | 3.2 | 96.59 | 225 | - | |
| EPNAS[163] | BMVC18 | 6.6 | 96.29 | 1.8 | 1 | |
| GHN[212] | ICLR19 | 5.7 | 97.16 | 0.84 | - | |
| NAO+random+c/o[166] | NeurIPS18 | 10.6 | 97.52 | 200 | 200 V100 | RS |
| SMASH [14] | ICLR18 | 16 | 95.97 | 1.5 | - | |
| Hierarchical-random [19] | ICLR18 | 15.7 | 96.09 | 8 | 200 | |
| RandomNAS [177] | UAI19 | 4.3 | 97.15 | 2.7 | - | |
| DARTS - random+c/o [17] | ICLR19 | 3.2 | 96.71 | 4 | 1 | |
| RandomNAS-NSAS[211] | CVPR20 | 3.08 | 97.36 | 0.7 | - | |
| NAO+weight sharing+c/o [166] | NeurIPS18 | 2.5 | 97.07 | 0.3 | 1 V100 | GD+SMBO |
| RENASNet+c/o[42] | CVPR19 | 3.5 | 91.12 | 1.5 | 4 | EA+RL |
| CARS[40] | CVPR20 | 3.6 | 97.38 | 0.4 | - | EA+GD |

https://arxiv.org/pdf/1908.00709.pdf