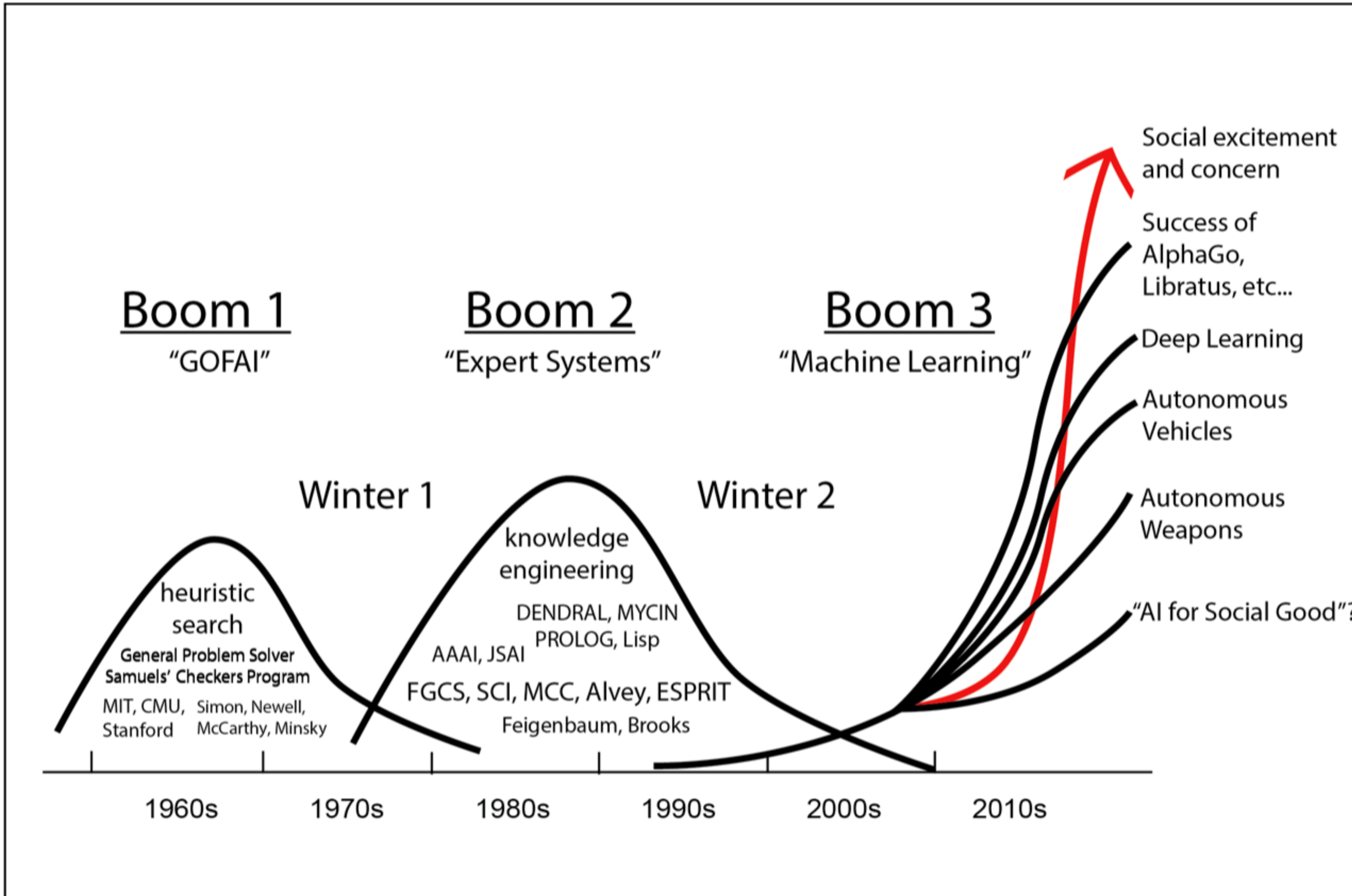
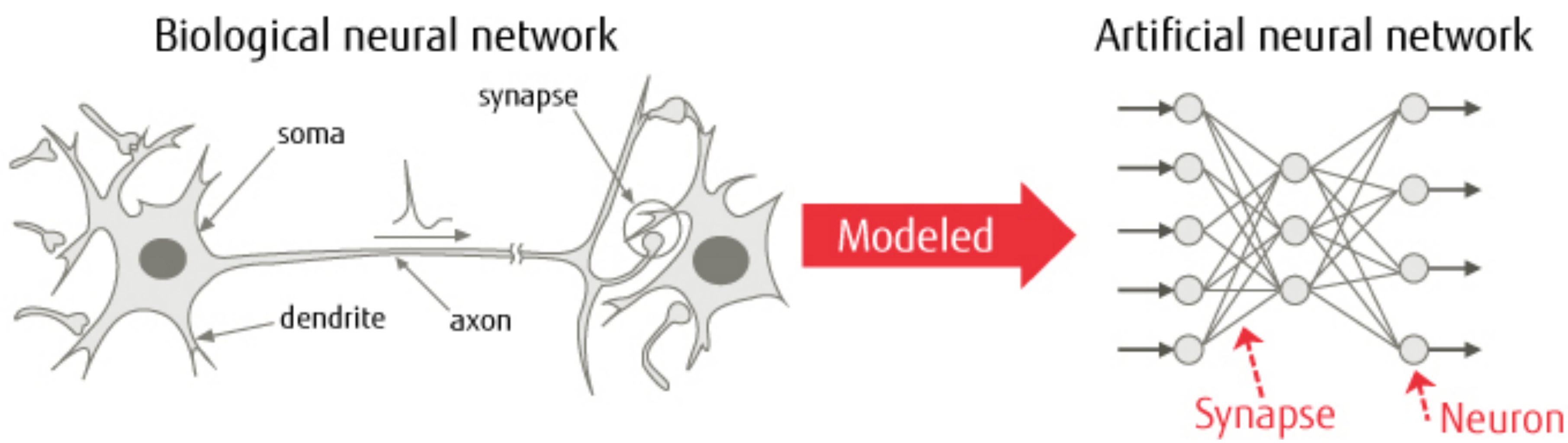
Thoughts on Deep Learning

모두의 연구소

- 2018. 10. 24 -


Boom 1
"GOFAI"
Boom 2
"Expert Systems"
Boom 3
"Machine Learning"
Winter 1
Winter 2
heuristic search
General Problem Solver
Samuels' Checkers Program
MIT, CMU, Stanford
Simon, Newell, McCarthy, Minsky
knowledge engineering
DENDRAL, MYCIN
PROLOG, Lisp
AAAI, JSAI
FGCS, SCI, MCC, Alvey, ESPRIT
Feigenbaum, Brooks
Social excitement and concern
Success of AlphaGo, Libratus, etc...
Deep Learning
Autonomous Vehicles
Autonomous Weapons
"AI for Social Good"?
1960s
1970s
1980s
1990s
2000s
2010s

Gets much deeper



	The second AI boom (1988)	The third AI boom (2015)
Layer	3 layers	7 layers
Neuron	29	1.1 million
Synapse	232	730 million

the neural network used in a mobile robot Fujitsu Laboratories
The object recognition network Fujitsu Laboratories developed in 2015

https://journal.jp.fujitsu.com/en/2016/02/09/01/

Representation Power

MLP1 - Universal Approximator:

It can approximate with any desired non-zero amount of error a family of functions that includes all **continuous function** on a closed and bounded subset of R^n, and **any function** mapping from any **finite** dimensional **discrete** space to another

Hornik (1989), Cybenko (1989)

Representation Power

Limit of theorem:

- Does not discuss the learnability of the neural network.
- Does not guarantee that a training algorithm will find the correct function generating training data.
- Does not state how large the hidden layer should be.

Representation Power

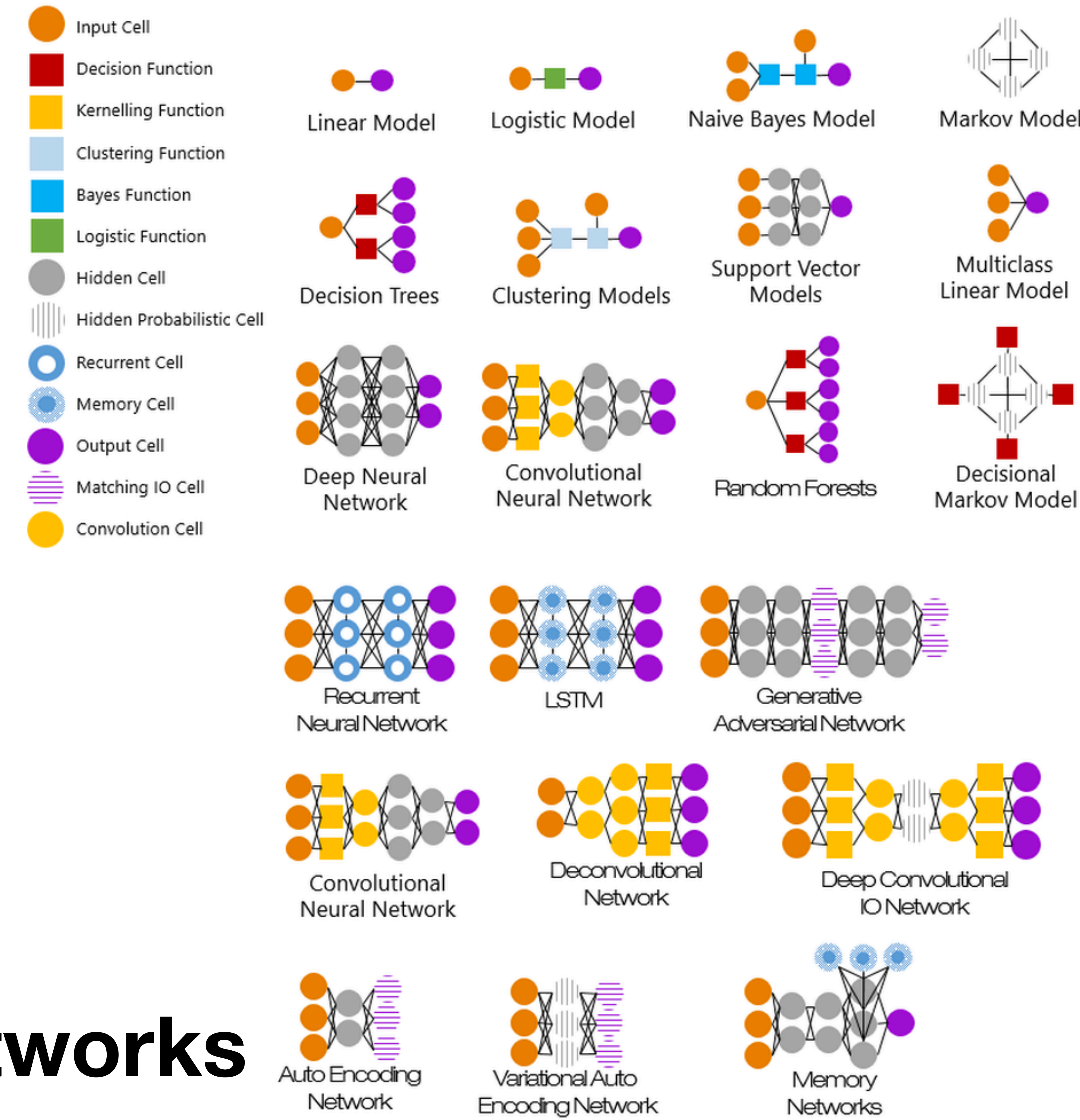
Why we go deep?

There exist neural networks with many layers of bounded size cannot be approximated by networks with fewer layers unless these layers are ***exponentially*** large.

Telgarsky[2016], https://arxiv.org/pdf/1509.08101.pdf

So there are

Various Networks



Why Deep Learning is so special?

Two Important Concepts:

Computation Graph
Auto-grad

Computation Graph Abstraction

Allows us us to ***easily*** *construct* ***arbitrary*** networks, ***evaluate*** their predictions for given inputs and compute gradients for their parameters with respect to ***arbitrary*** scalar losses.

Computation Graph Abstraction

a neural network = **An (arbitrary) DAG**

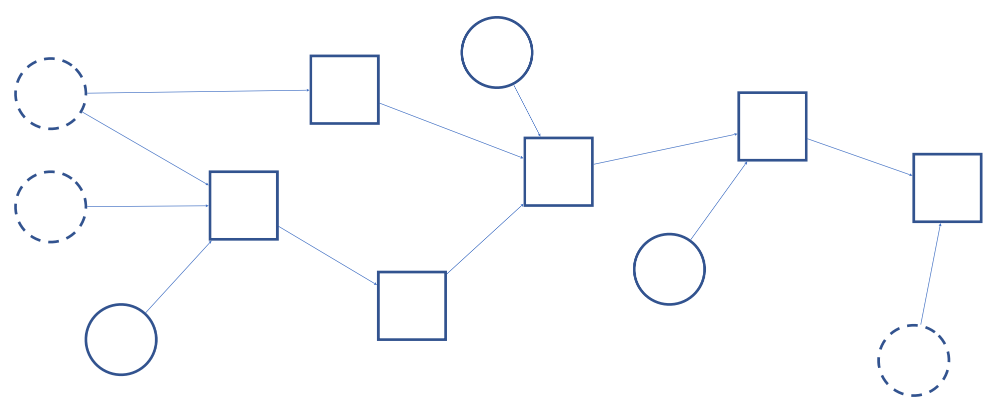
A computation graph is a representation of an arbitrary mathematical computation as a graph.

It is a directed acyclic graph(**DAG**) in which
- **nodes:** mathematical operations or (bound) variables
- **edges:** he flow of intermediary values between nodes

The graph structure defines the **order** of the computation in terms of the dependencies between the different component

Computation Graph Abstraction

a neural network = An (arbitrary) DAG



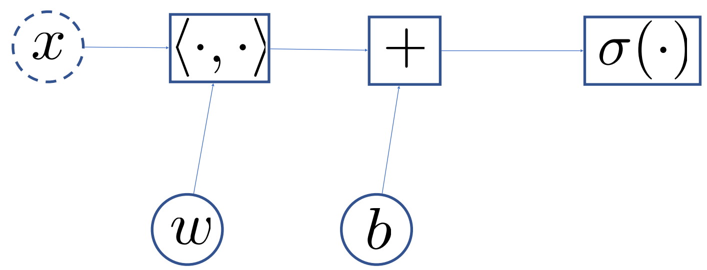
1. ***Solid Circles***: parameters (to be estimated or found)
2. ***Dashed Circles***: vector inputs/outputs (given as a training example)
3. ***Squares***: compute nodes (functions, often continuous/differentiable)

Reference: cho, 2018

Computation Graph Abstraction

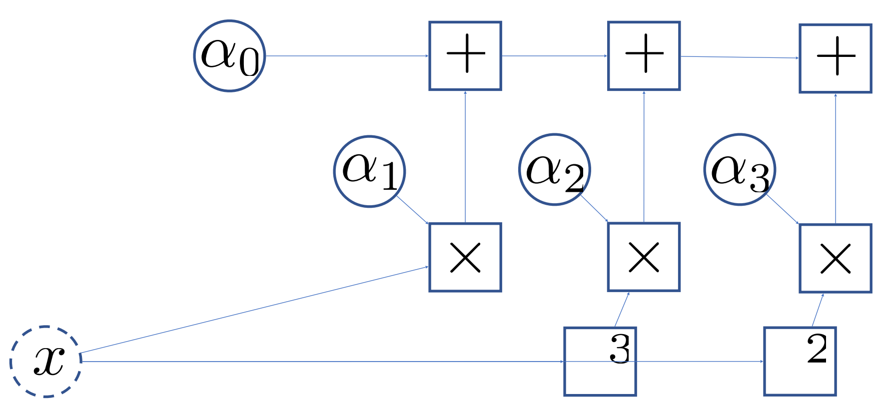
1. **Logistic regression**

$$p_\theta(y=1|x) = \sigma(w^\top x + b) = \frac{1}{1+\exp(-w^\top x - b)}$$



2. **3rd-order polynomial function**

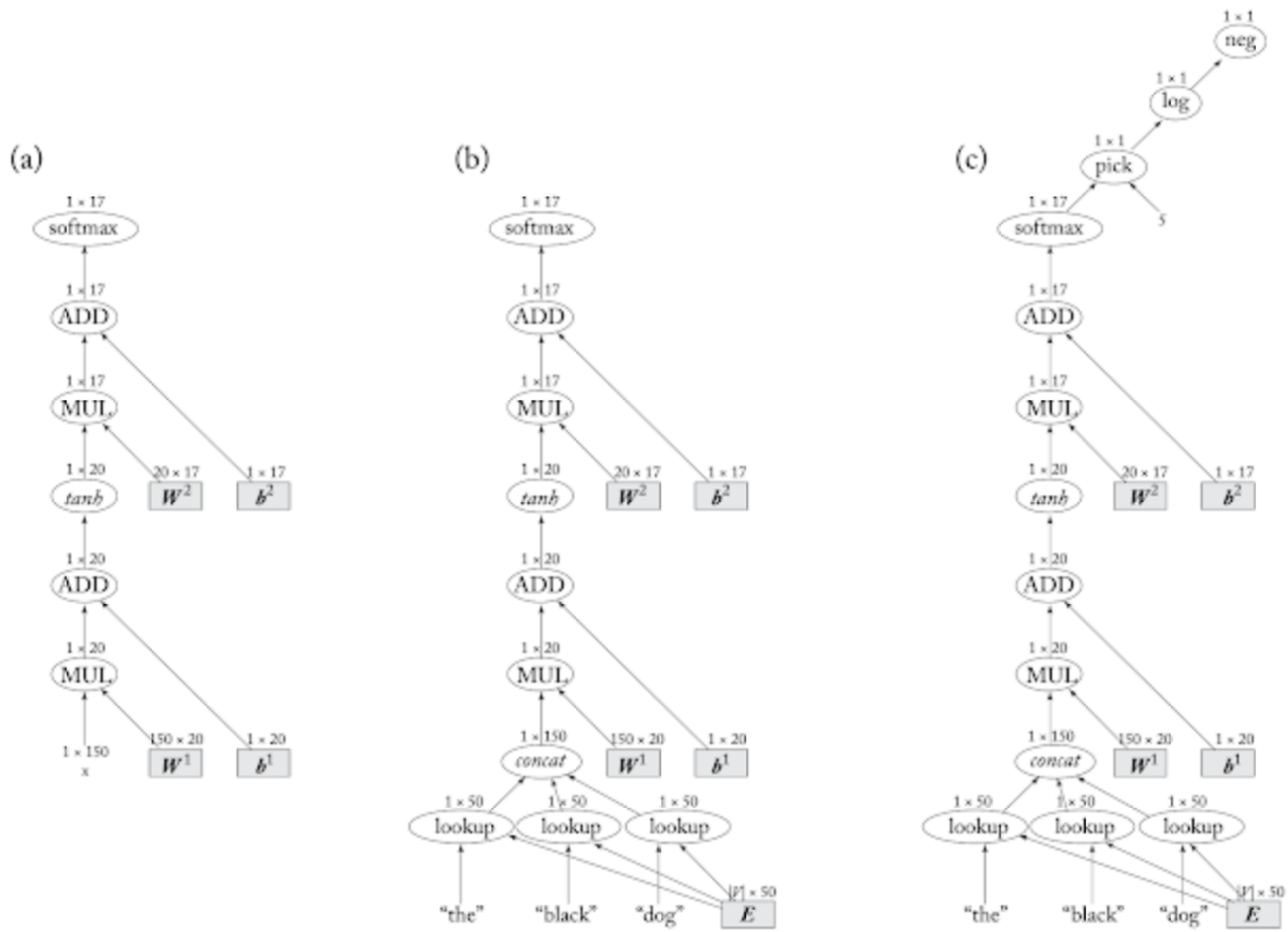
$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3$$



Reference: cho, 2018

Computation Graph Abstraction

Examples from Deep Neural Network



Computation Graph

Forward Computation

Backward Computation

Forward Computation

Algorithm 5.3 Computation graph forward pass.

```
for i = 1 to N do
    Let a_1, ..., a_m = π^{-1}(i)
    v(i) ← f_i(v(a_1), ..., v(a_m))
```

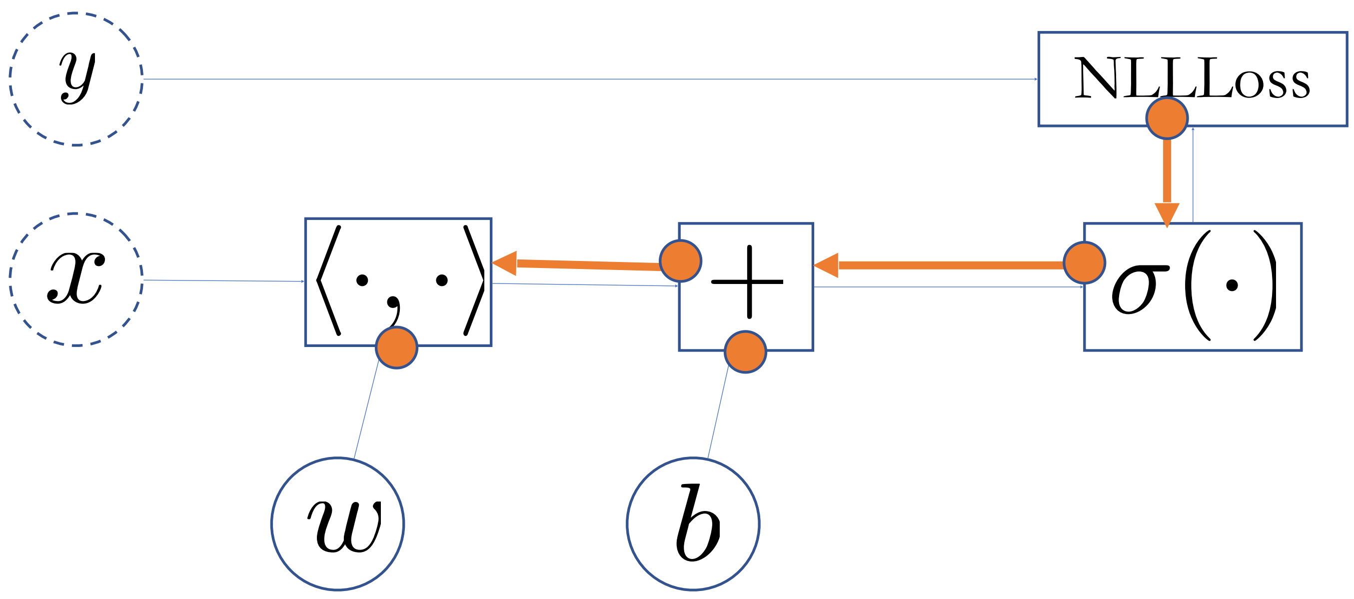
1: **for** i = 1 to N **do**

2: Let $a_1, \ldots, a_m = \pi^{-1}(i)$

3: $v(i) \leftarrow f_i(v(a_1), \ldots, v(a_m))$

Backward Computation

- Automatic differentiation (autograd)
  - Reverse-sweep the DAG starting from the loss function node.
    - Iteratively multiplies the Jacobian of each OP node until the leaf nodes of the parameters.
    - As expensive as forward computation with a constant overhead: O(N), where N: # of nodes.



Backward Computation

- Automatic differentiation (autograd)
  - Implement the Jacobian-vector product of each OP node:

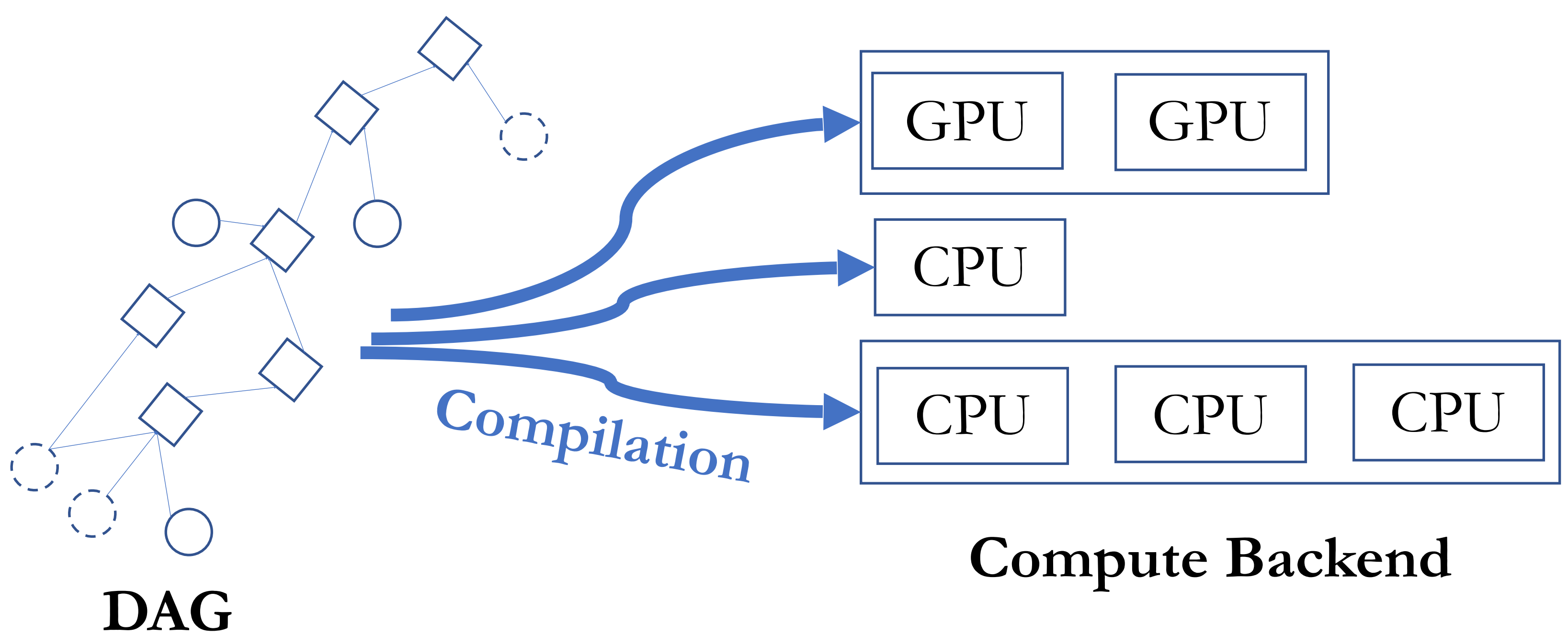
$$\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \vdots \\ \frac{\partial L}{\partial x_d} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_{d'}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_1}{\partial x_d} & \cdots & \frac{\partial F_{d'}}{\partial x_d} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial F_1} \\ \vdots \\ \frac{\partial L}{\partial F_{d'}} \end{bmatrix}$$

  - Can be implemented efficiently without explicitly computing the Jacobian.
  - The same implementation can be reused every time the OP node is called.

Reference: cho, 2018

Backward Computation

- Practical Implications – Automatic differentiation (autograd)
  - Unless a complete new OP is introduced, no need to manually derive the gradient
  - Nice de-coupling of specification (front-end) and implementation (back-end)
    1. [Front-end] Design a neural network by creating a DAG.
    2. [Back-end] The DAG is “compiled” into an efficient code for a target compute device.



Reference: cho, 2018

Backward Computation

Algorithm 5.4 Computation graph backward pass (backpropagation).

1: $d(N) \leftarrow 1$ $\quad\triangleright\ \frac{\partial N}{\partial N} = 1$

2: **for** i = N-1 to 1 **do**

3: $\quad d(i) \leftarrow \sum_{j \in \pi(i)} d(j) \cdot \frac{\partial f_j}{\partial i}$ $\quad\triangleright\ \frac{\partial N}{\partial i} = \sum_{j \in \pi(i)} \frac{\partial N}{\partial j} \frac{\partial j}{\partial i}$

Do you know deep learning?

$$C = \frac{1}{2|T|} \sum_{(X,t)\in T} \sum_{p} \left(n_p^3 - t_p\right)^2$$

$$\frac{\partial C}{\partial w_{00}^3} = \frac{1}{|T|} \sum_{(X,t)\in T} \sum_{p} \left(n_p^3 - t_p\right) \times \frac{\partial}{\partial w_{00}^3}\left(n_p^3\right)$$

$$= \frac{1}{|T|} \sum_{(X,t)\in T} \sum_{p} \left(n_p^3 - t_p\right) \times \frac{\partial}{\partial w_{00}^3}\left(a\left(z_p^3\right)\right)$$

$$= \frac{1}{|T|} \sum_{(X,t)\in T} \sum_{p} \left(n_p^3 - t_p\right) \times \left(n_p^3\left(1 - n_p^3\right)\right) \times \frac{\partial}{\partial w_{00}^3}\left(z_p^3\right)$$

$$= \frac{1}{|T|} \sum_{(X,t)\in T} \sum_{p} \left(n_p^3 - t_p\right) \times \left(n_p^3\left(1 - n_p^3\right)\right) \times \frac{\partial}{\partial w_{00}^3}\left(w_{0p}^3 n_0^2 + w_{1p}^3 n_1^2 + b_p^3\right)$$

Only when p is 0 will there be a w_{00}^3 term, otherwise the whole thing will be multiplied by 0

$$= \frac{1}{|T|} \sum_{(X,t)\in T} \left(n_0^3 - t_0\right) \times \left(n_0^3\left(1 - n_0^3\right)\right) \times \frac{\partial}{\partial w_{00}^3}\left(w_{00}^3 n_0^2 + w_{10}^3 n_1^2 + b_0^3\right)$$

$$= \frac{1}{|T|} \sum_{(X,t)\in T} \left(n_0^3 - t_0\right) \times \left(n_0^3\left(1 - n_0^3\right)\right) \times \left(n_0^2\right)$$

$$= \frac{1}{|T|} \sum_{(X,t)\in T} \frac{\partial C_{Xt}}{\partial n_0^3} \times \frac{\partial n_0^3}{\partial z_0^3} \times n_0^2$$

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \qquad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \qquad \text{(BP2)}$$

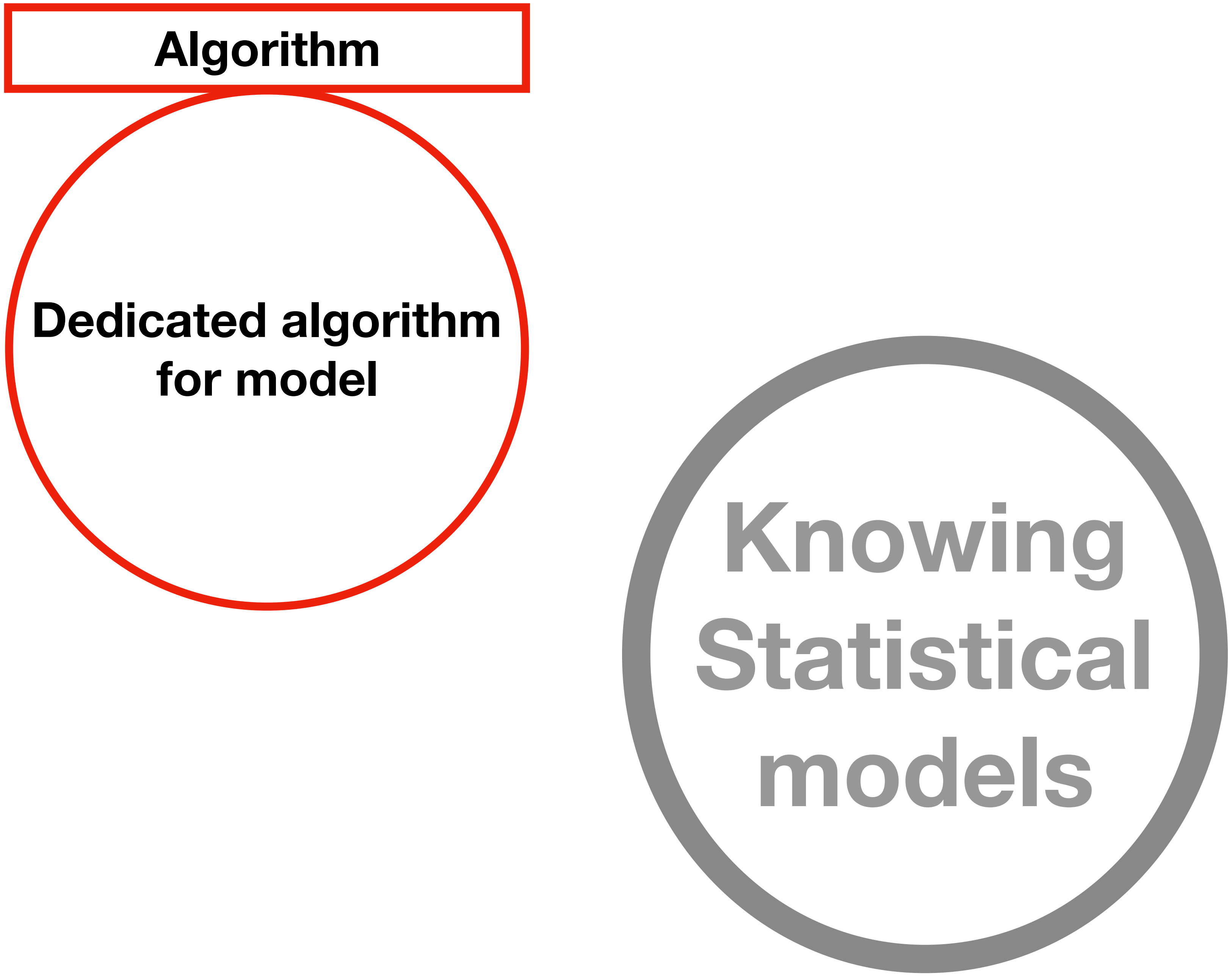
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad \text{(BP3)}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \qquad \text{(BP4)}$$

Backprop?

Deep learning Frameworks!!

Knowing Statistical models


Algorithm
Dedicated algorithm for model
Knowing Statistical models

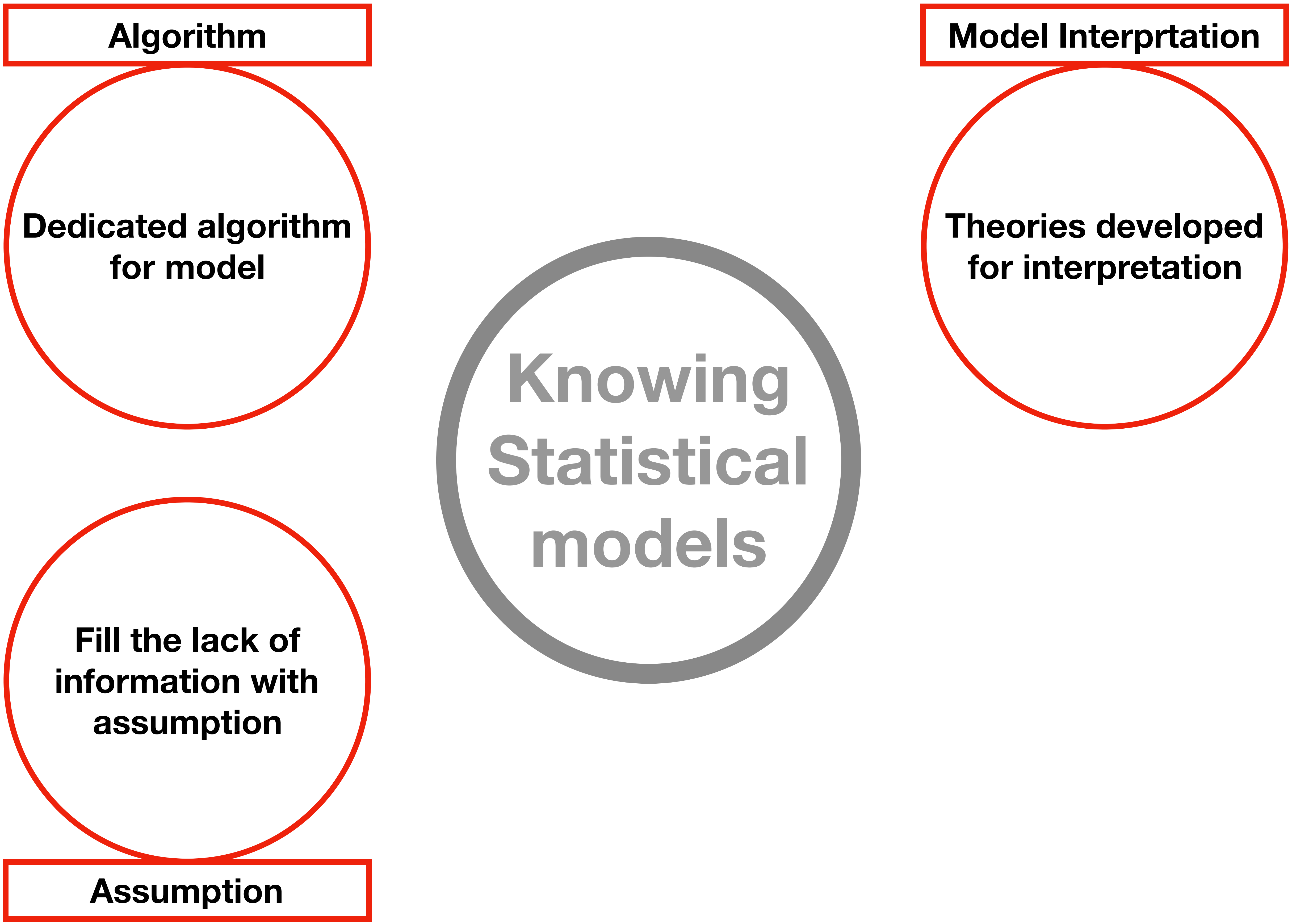
Algorithm

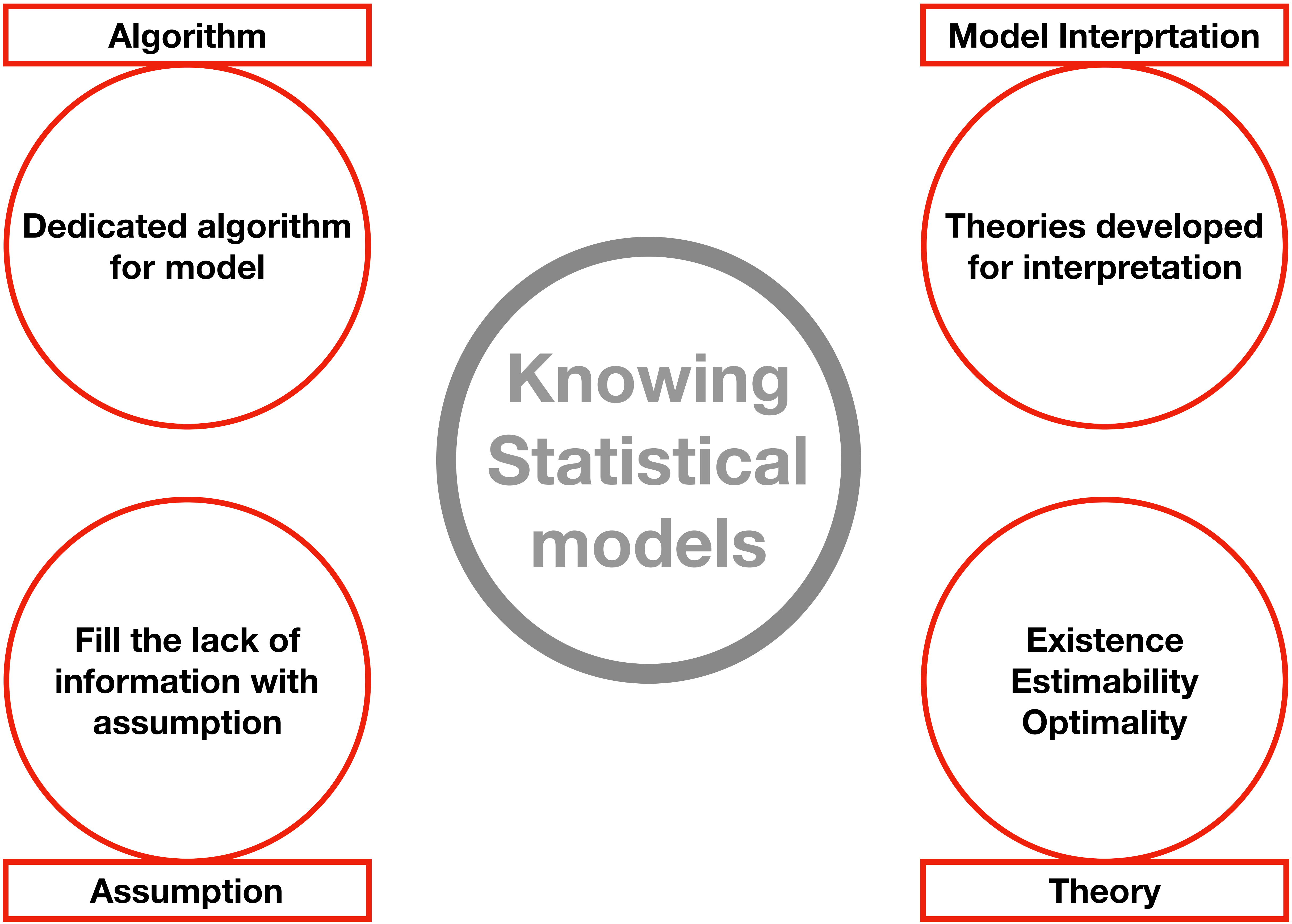
Dedicated algorithm for model

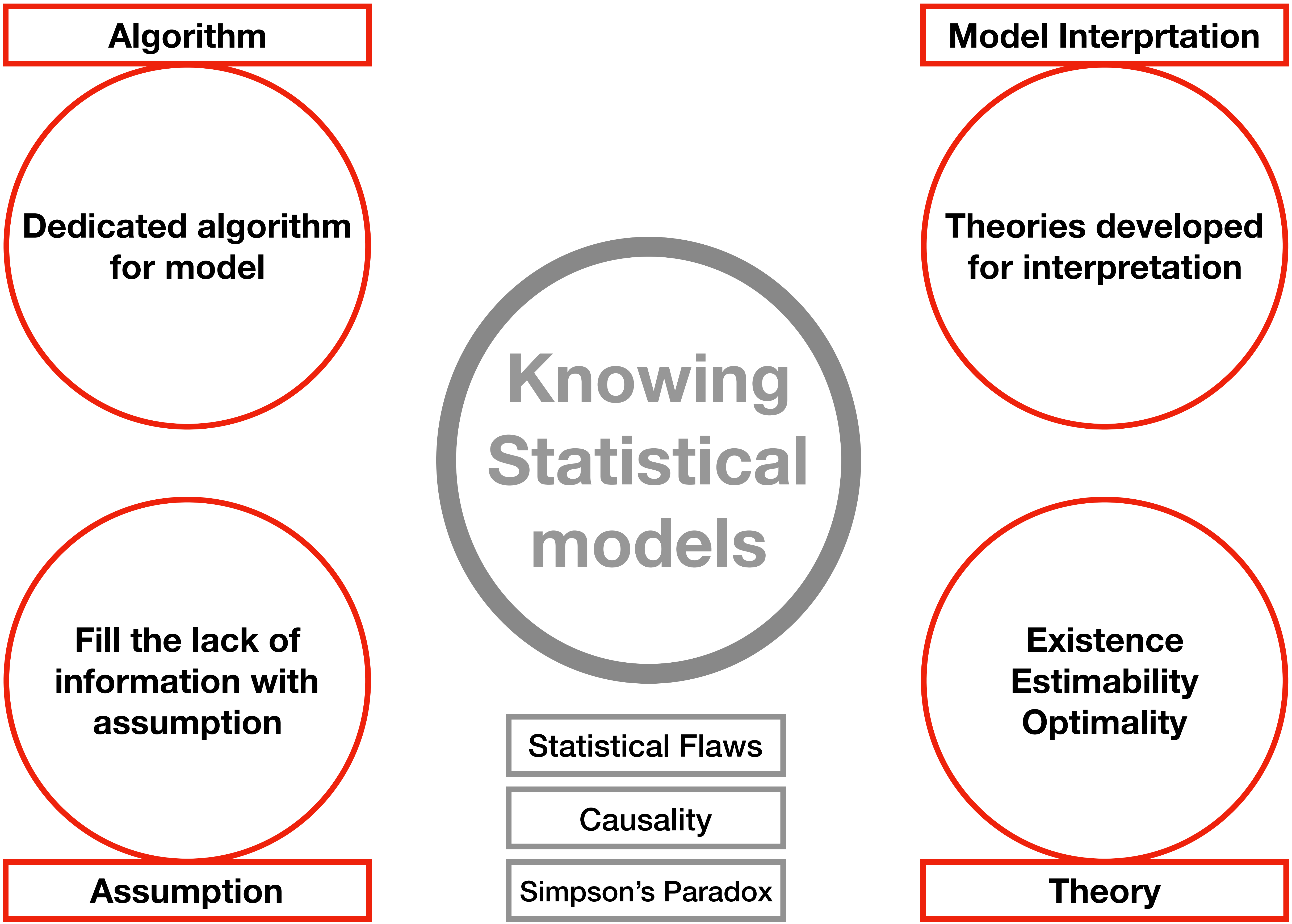
Model Interprtation

Theories developed for interpretation

Knowing Statistical models


Algorithm
Dedicated algorithm for model
Model Interprtation
Theories developed for interpretation
Knowing Statistical models
Fill the lack of information with assumption
Assumption


Algorithm
Dedicated algorithm for model
Model Interprtation
Theories developed for interpretation
Knowing Statistical models
Fill the lack of information with assumption
Assumption
Existence
Estimability
Optimality
Theory


Algorithm
Dedicated algorithm for model
Model Interprtation
Theories developed for interpretation
Knowing Statistical models
Fill the lack of information with assumption
Assumption
Existence
Estimability
Optimality
Theory
Statistical Flaws
Causality
Simpson's Paradox

Algorithm

Back Propagation

Knowing Deep Learning

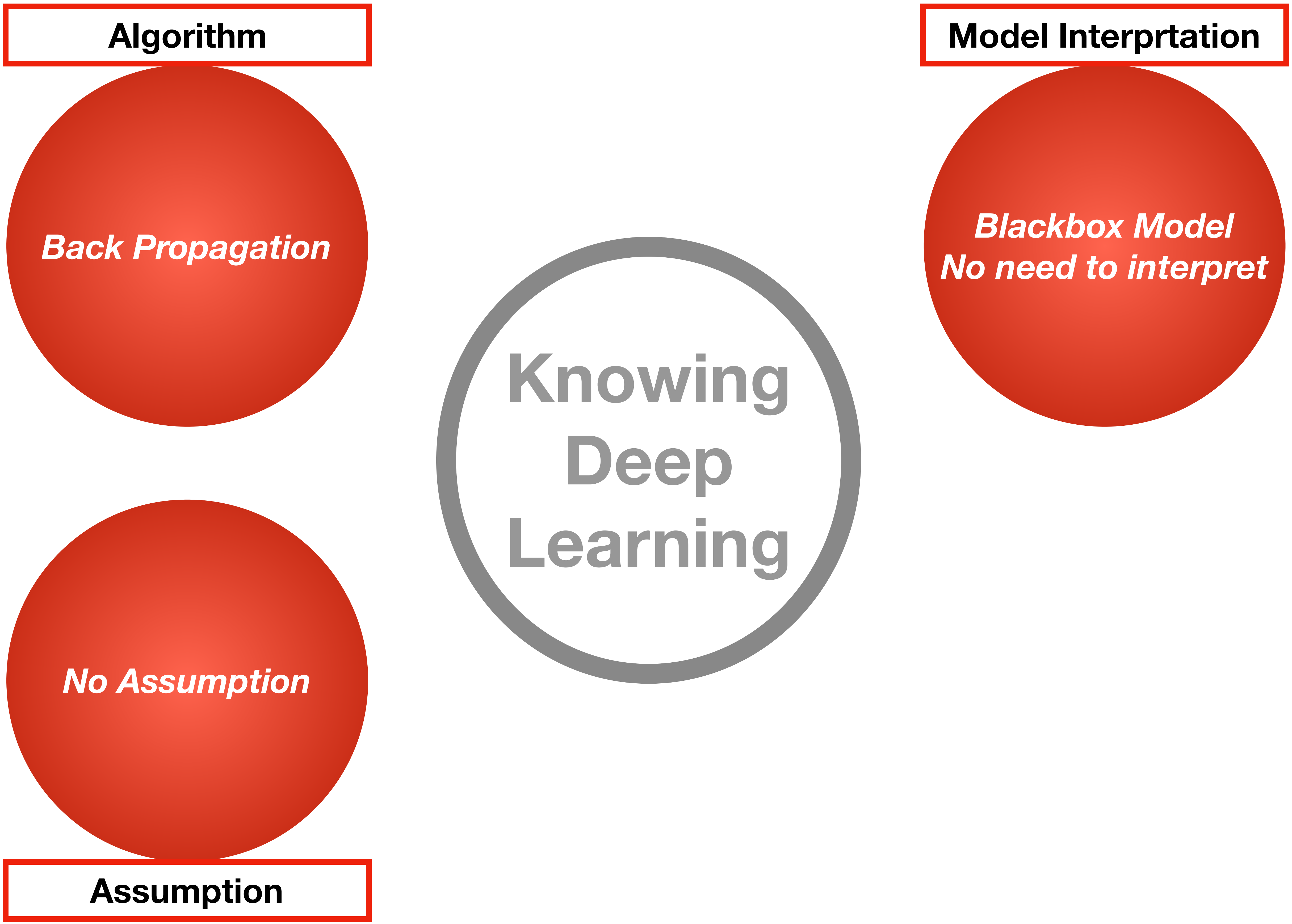
Algorithm

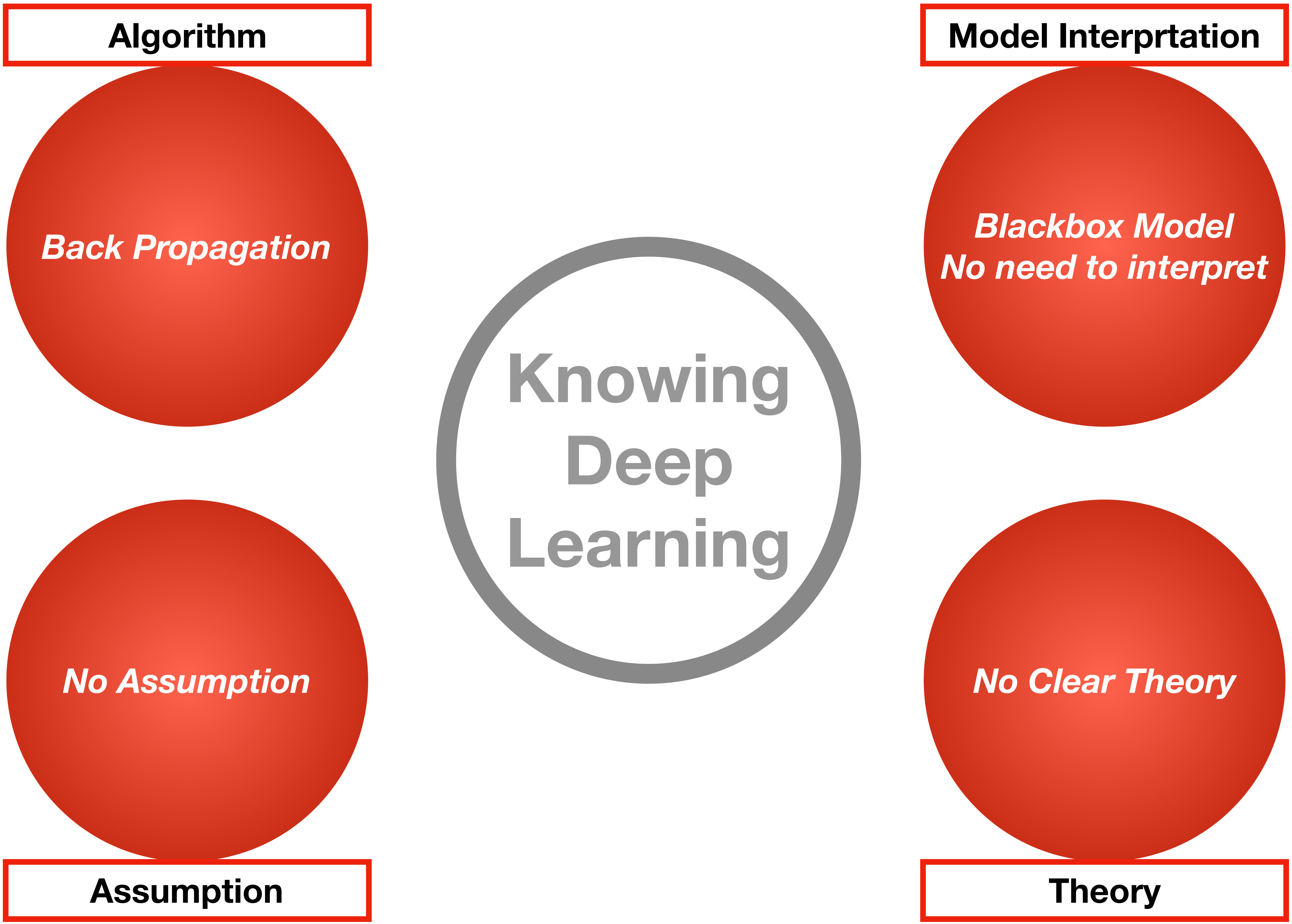
Back Propagation

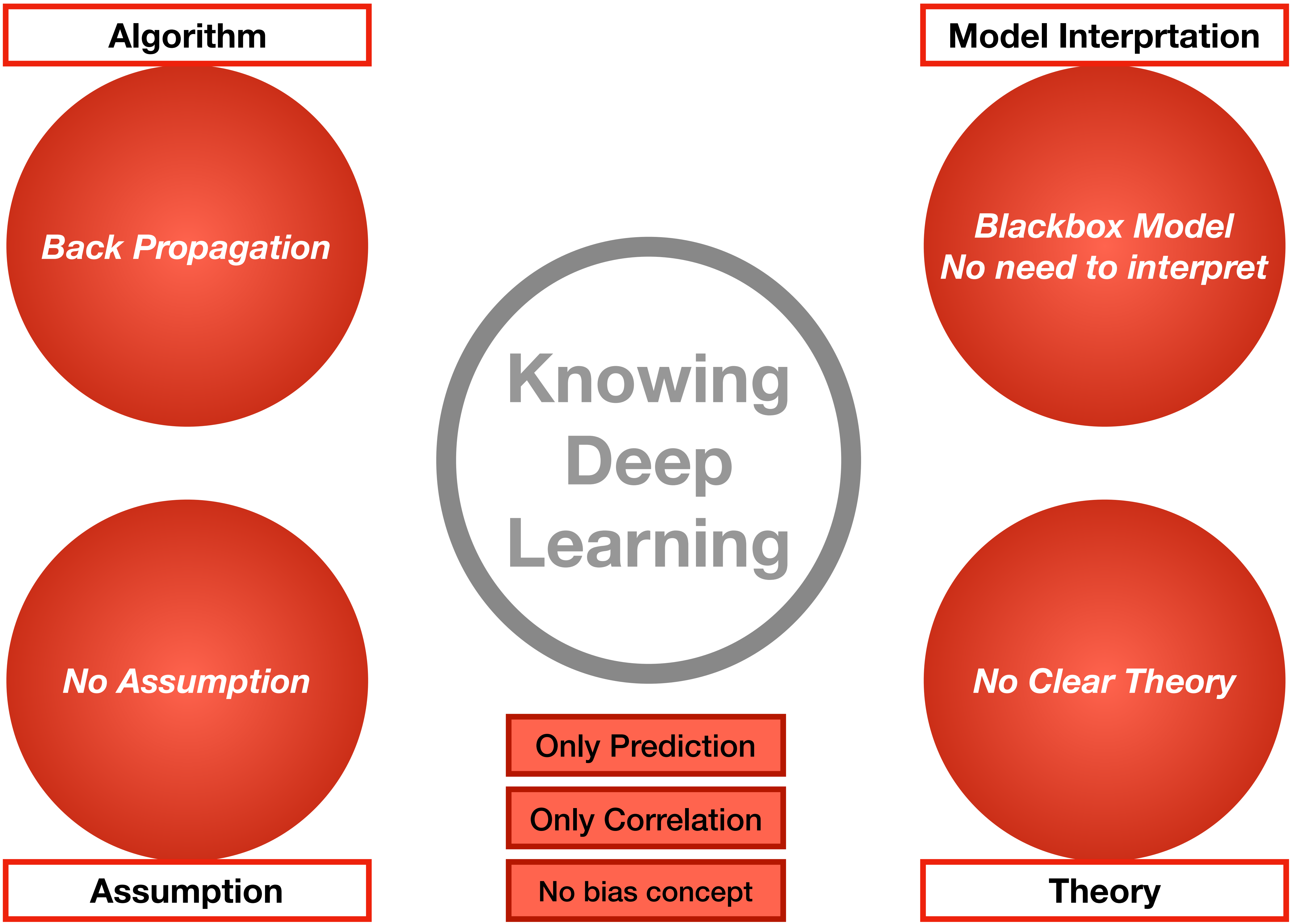
Model Interprtation

Blackbox Model
No need to interpret

Knowing Deep Learning


Algorithm
Back Propagation
Model Interprtation
Blackbox Model
No need to interpret
Knowing
Deep
Learning
No Assumption
Assumption


Algorithm
Back Propagation
Model Interprtation
Blackbox Model
No need to interpret
Knowing
Deep
Learning
No Assumption
Assumption
No Clear Theory
Theory


Algorithm
Back Propagation
Model Interprtation
Blackbox Model
No need to interpret
Knowing
Deep
Learning
No Assumption
Assumption
Only Prediction
Only Correlation
No bias concept
No Clear Theory
Theory

What is *IMPORTANT* is matrix operation!!

Understanding network	=	Know every details about input & output

Batch normalization

Fully Connected

Pooling

Layer normalization

Dropout

Spectral normalization

Masking

Convolution

Skip Connection

Padding

Dilated convolution

Variational Inference

Adversarial Network

Inverse autoregression

Re-parametrization trick

GRU

AND

Various Ideas!!

Memory Cell

Attention mechanism

Self attention

Residual Block

Understanding network = **Know every details about input & output**

In practice, there are techniques like…

Reference: cho, 2018

Practicalities

- Optimization algorithm:
  - Adam(Kingma and Ba, 2014): Effective & Robust to the choice of learning rate

- Initializer: magnitude of random variable matters
  - Xavier Initializer: generally good.

$$W \sim U\left(-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, \frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}\right)$$

  - He: good for deep network

$$W \sim N\left(0, \frac{2}{d_{in}}\right)$$

Ref: Deep Learning Book

Practicalities

- Restart & Ensemble: Do if resource allows

- Vanishing gradients:
  - Shallower network
  - Batch normalization
  - Special architecture such as LSTM or GRU

- Exploding gradients:
  - Gradient clipping

- Saturated neurons:
  - Change initializer
  - Change learning rate
  - Scale input values
  - Layer normalization

$$g_h = \frac{\tanh(\mathbf{h})}{\|\tanh(\mathbf{h})\|}$$

Practicalities

- Dead neurons: (especially for ReLU)
  - Reduce learning rate
- Shuffling
  - Crucial and double check if done when training
- Learning rate scheduling
  - Reduce learning rate