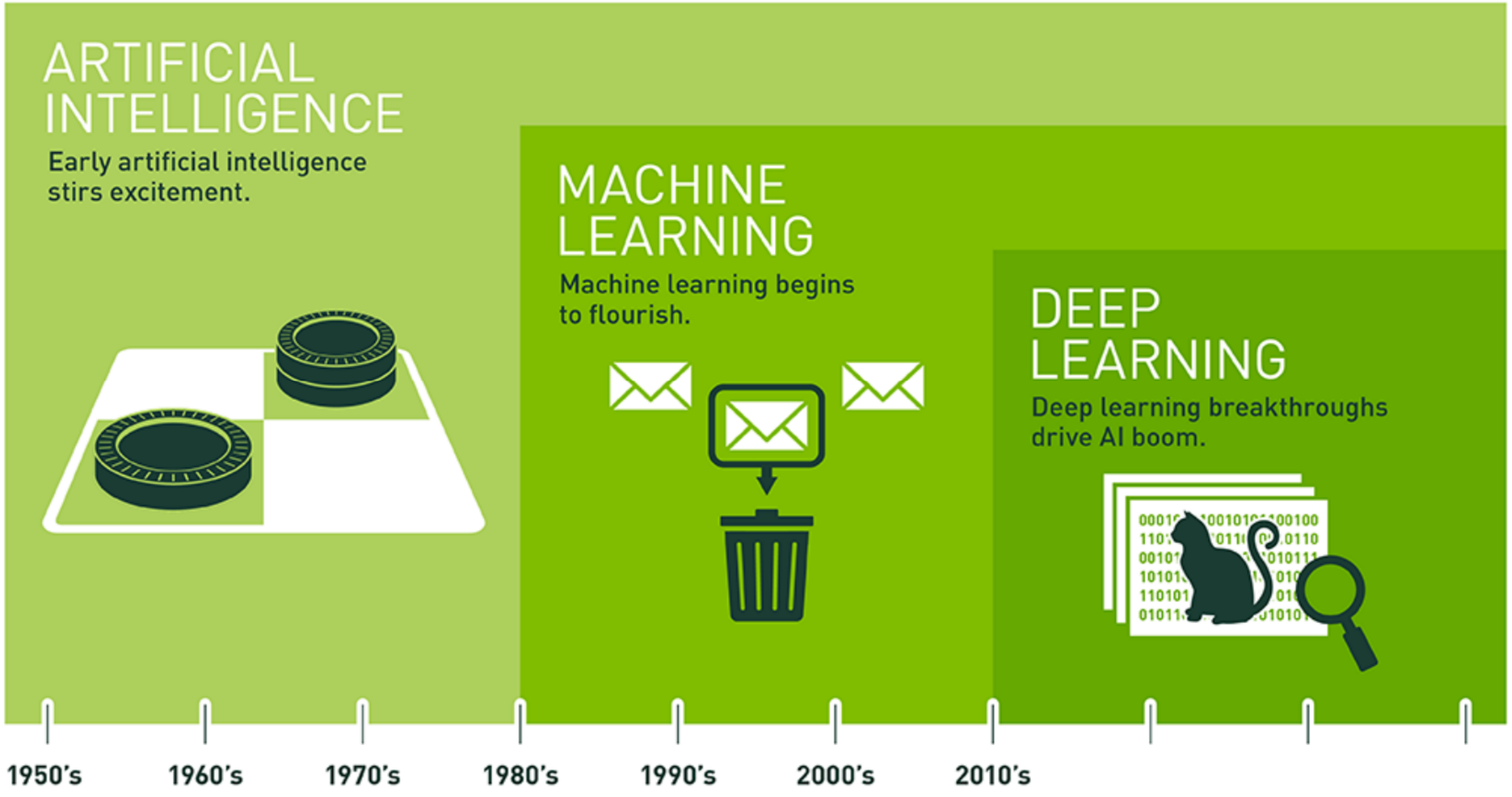
Introduction to deep learning

From an implementation view under Pytorch

Songpeng Zu @ 2020.11.17

AI, ML and DL



Artificial Intelligence
Artificial Intelligence is human intelligence exhibited by machines

Machine Learning
Field of study that gives computers the ability to learn without being explicitly programmed.

Deep Learning
Using deep neural networks to implement machine learning

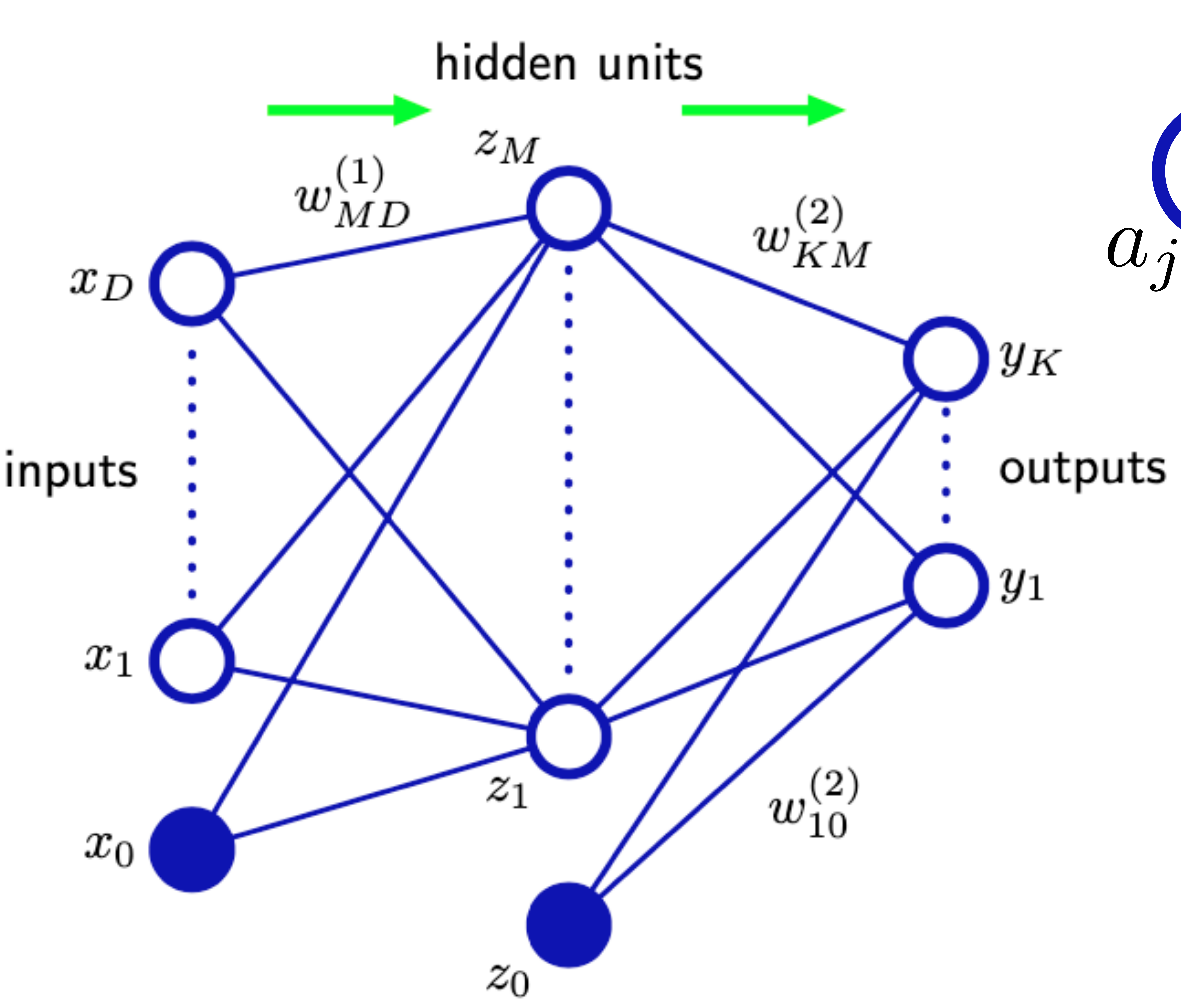
Deep Learning

Empirical Risk Minimization

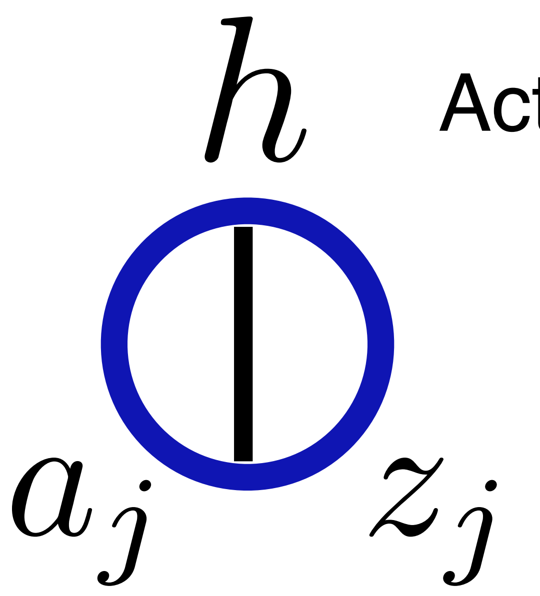
$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

*In deep learning: the space of **f** is the multiple-layer neural network*

A Typical Structure of Neural Network (NN)



Neural network or feed forward network or multilayer perceptron



Activation function

$$a_j = \sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0}$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=1}^{M} w^{(2)}_{kj} z_j + w^{(2)}_{k0}$$

$$y_k = \sigma(a_k)$$

Pattern Recognition and Machine Learning, Chapter 5, Neural Networks

Content

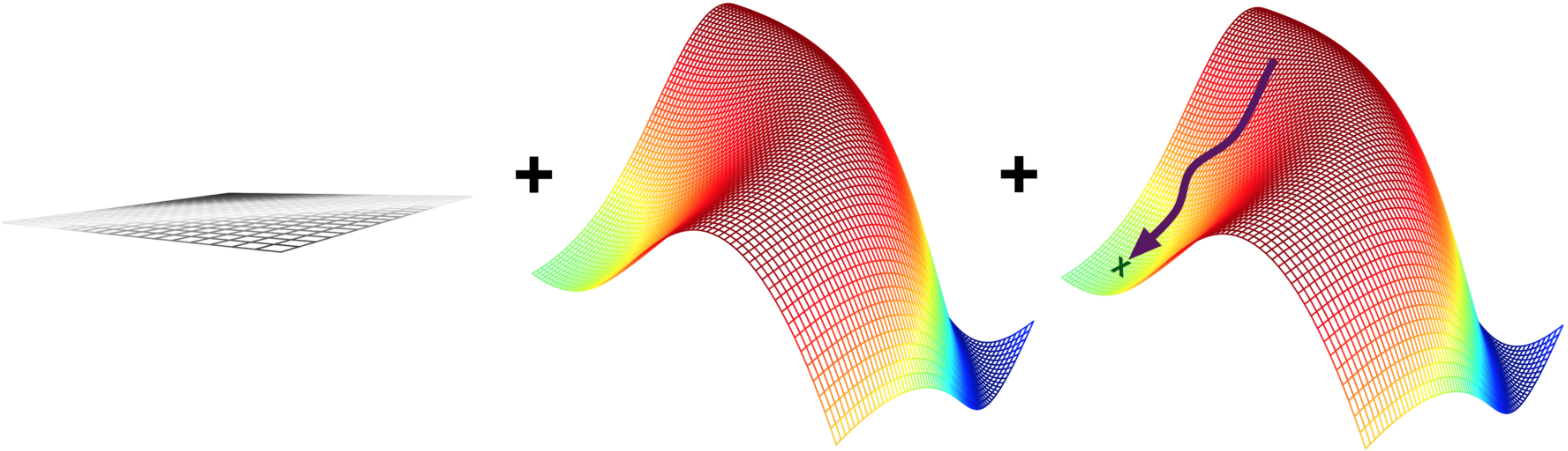
- Optimization in deep learning:
    - *Backpropagation (BP)*
    - *mini-batch gradient descent*
    - Momentum, adaptive learning rate
    - Tricks
- The implementation of deep learning
    - A deterministic, directed acyclic, computational graph
    - Elements: data organization; definition of neural network; and the optimization module
- Example: an implementation of VAE on MNIST

Optimization in deep learning

Learning and Optimization

Learning = Representation + Evaluation + Optimization

- Representation: Hypothesis space
- Evaluation: Objective/Loss function



Domingos, Pedro. Communications of the ACM, 55.10(2012): 78-87

Gradient-based optimization

First-order method

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

- Approximation of $J(\cdot)$ at θ_t

$$J(\boldsymbol{\theta}_t + \boldsymbol{v}) \approx \hat{J}(\boldsymbol{\theta}_t + \boldsymbol{v}) = J(\boldsymbol{\theta}_t) + \nabla J(\boldsymbol{\theta}_t)^T \boldsymbol{v}$$

- Minimize the surrogate function

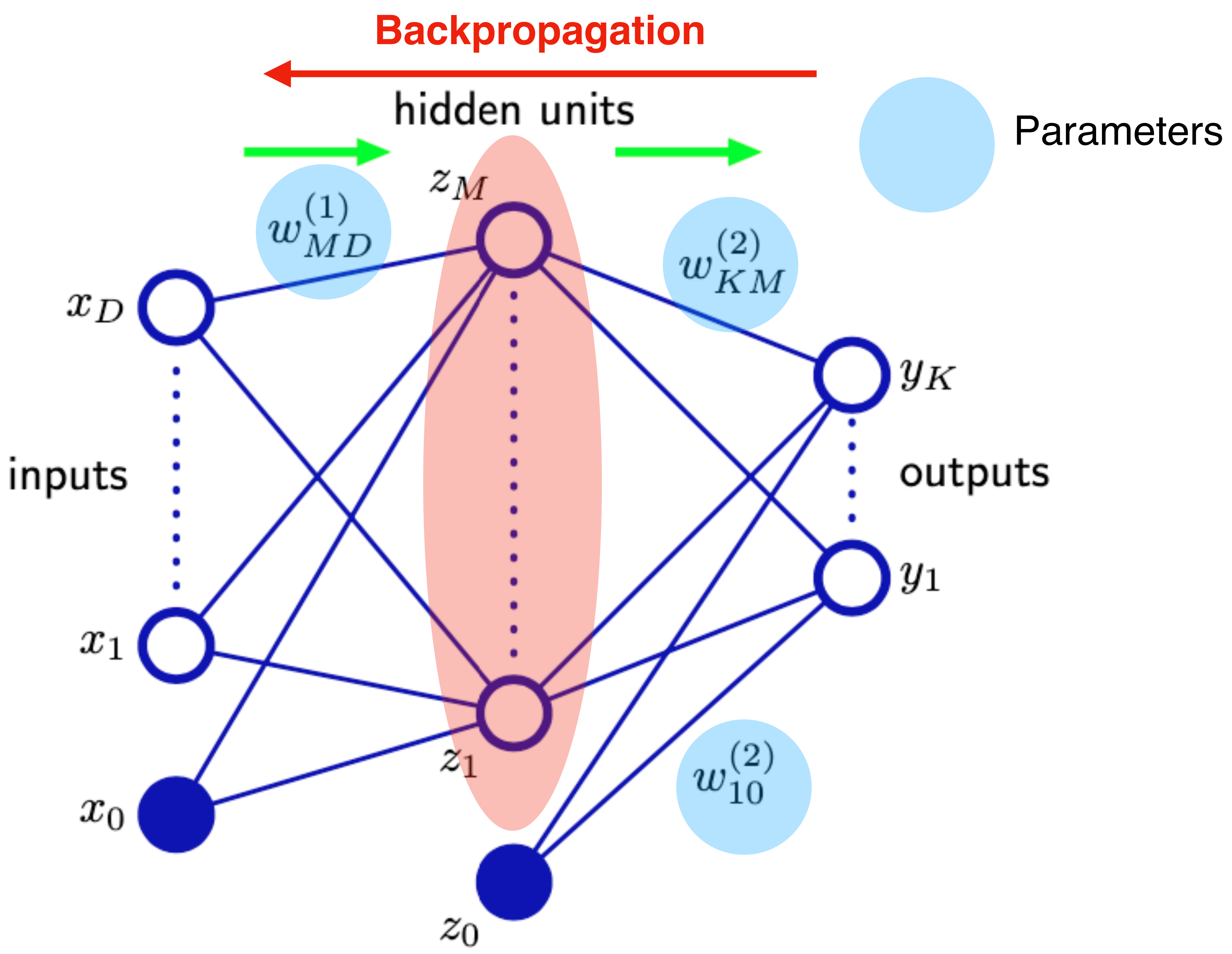
$$\boldsymbol{v}^* = \arg\min J(\boldsymbol{\theta}_t) + \nabla J(\boldsymbol{\theta}_t)^T \boldsymbol{v} + \frac{1}{2\alpha}\|\boldsymbol{v}\|_2^2 = -\alpha \nabla J(\boldsymbol{\theta}_t)$$

Initialization: $\boldsymbol{\theta}_0$
for $t = 0,1,\cdots$
$\quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla J(\boldsymbol{\theta}_t)$
until stopping criterion is satisfied

Gradient descent

Backpropagation (BP)

Differentiation using Chain Rule



$$l = g(y_1, \cdots, y_k)$$

$$\nu = \left(\frac{\partial l}{\partial y_1}, \cdots, \frac{\partial l}{\partial y_K}\right)^t \quad J = \begin{bmatrix} \frac{\partial y_1}{\partial z_1}, & \dots, & \frac{\partial y_1}{\partial z_M} \\ \vdots, & \ddots, & \vdots \\ \frac{\partial y_K}{\partial z_1}, & \dots, & \frac{\partial y_K}{\partial z_M} \end{bmatrix}$$

$$\frac{\partial l}{\partial z} = J^t \cdot \nu$$

Mini-batch optimization

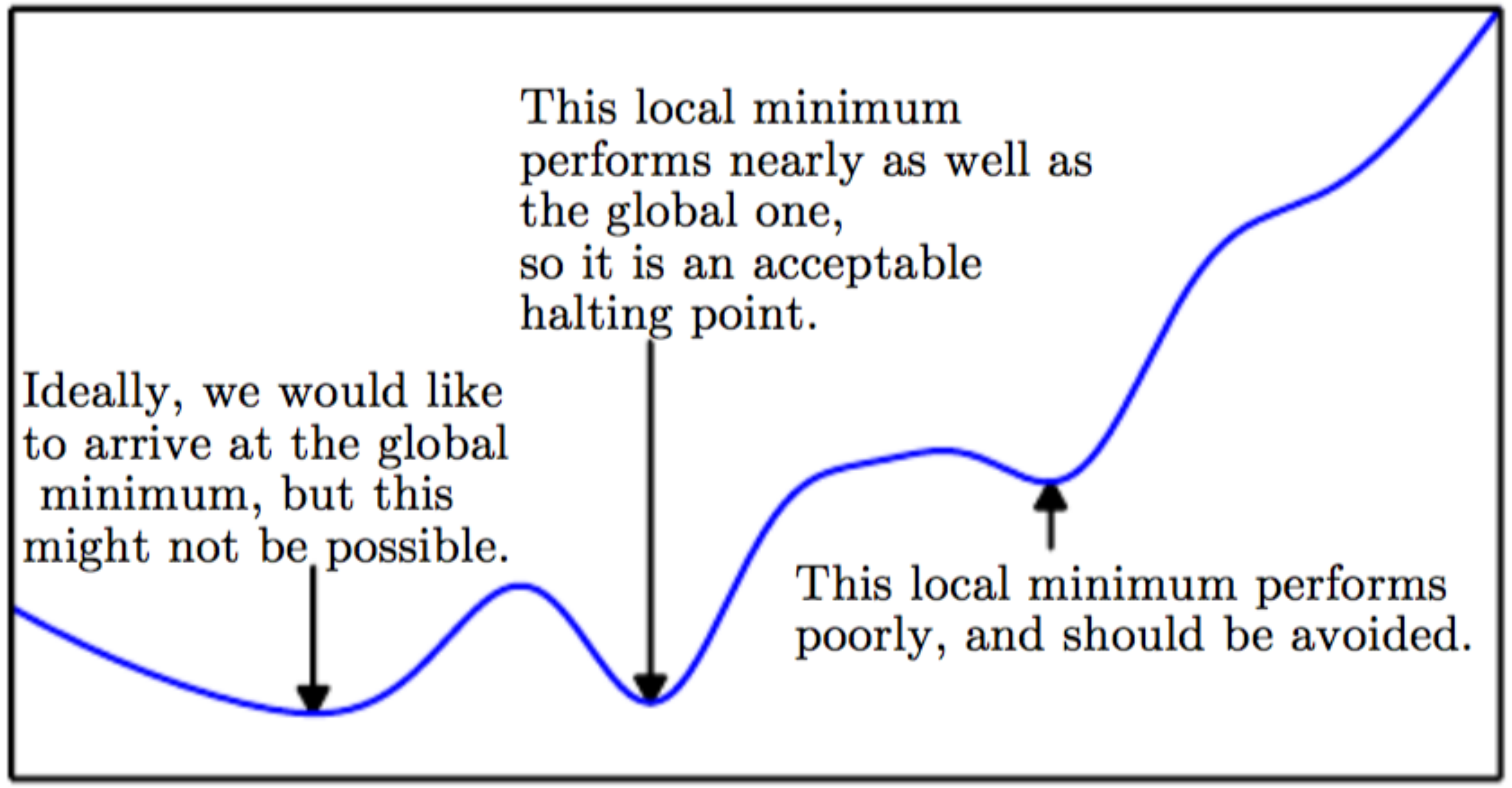
When we handle large scale of data set

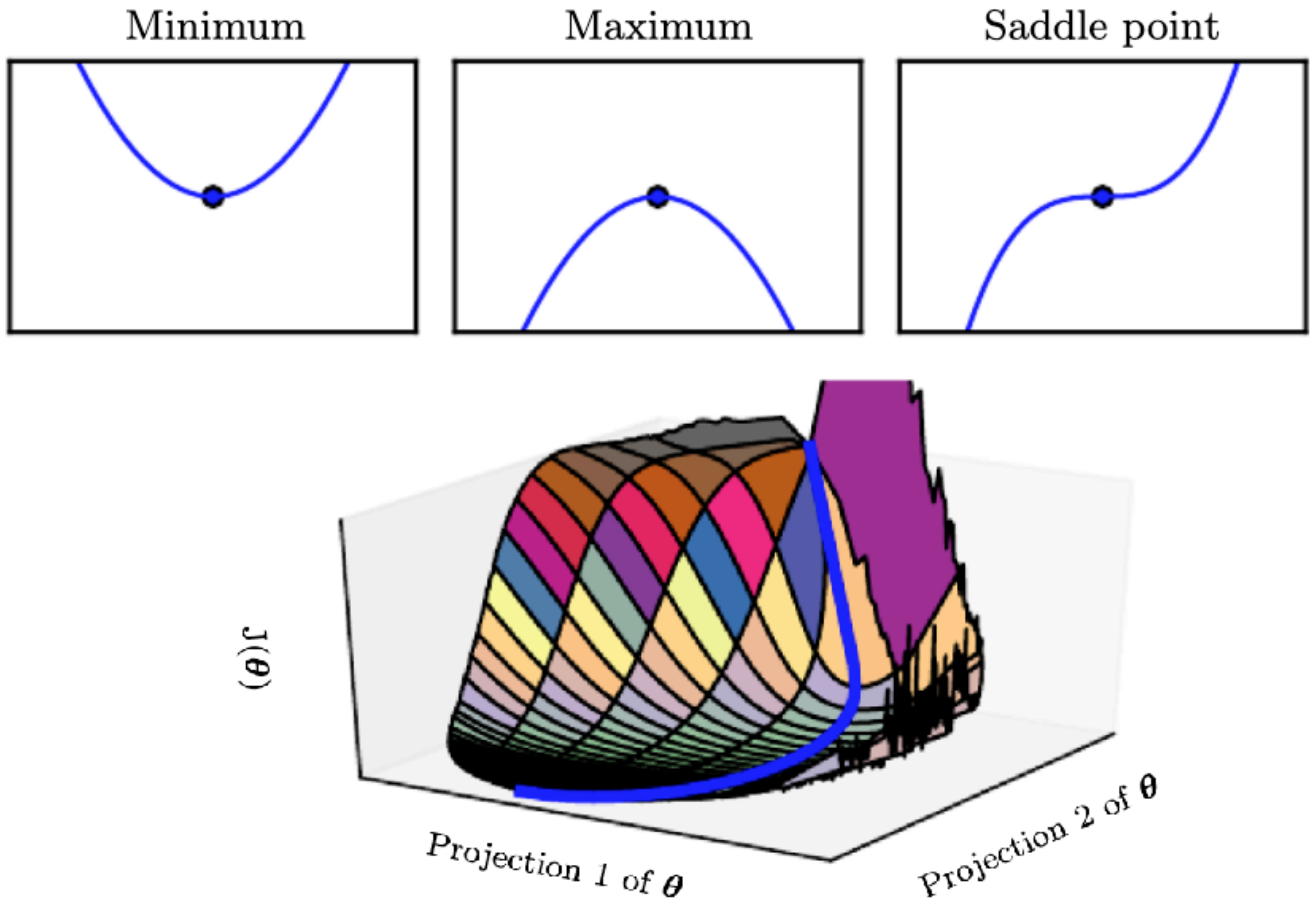
$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} \log p_{\text{model}}(\boldsymbol{x}, y; \boldsymbol{\theta})$$

- Batch/deterministic method: process all the samples simultaneously
- Stochastic/online method: use only a single example at a time
  - Generalization error is often best.
  - Estimation is noisy, and need to carefully choose the learning rate.
- Mini-batch: use small part of data sampled from the entire data set
  - Standard error: less than linear returns (square root of n).
  - Small batches can offer a regularizing effect (perhaps due to the noise).
  - The noise is reduced (compared with stochastic method).
  - Hardware consideration:
    - Memory cost scales with the batch size.
    - Extremely small batches are usually underutilized by multicore architectures.
    - Trick: power of 2 batch sizes usually offer better runtime when using GPU.

Challenges in NN optimization

- **Local minima** :
  - non-convex optimization
  - lots of local minima
- **Saddle points** in high-dim spaces
  - More common than local minima
  - Gradient-based methods seem to be able to escape empirically

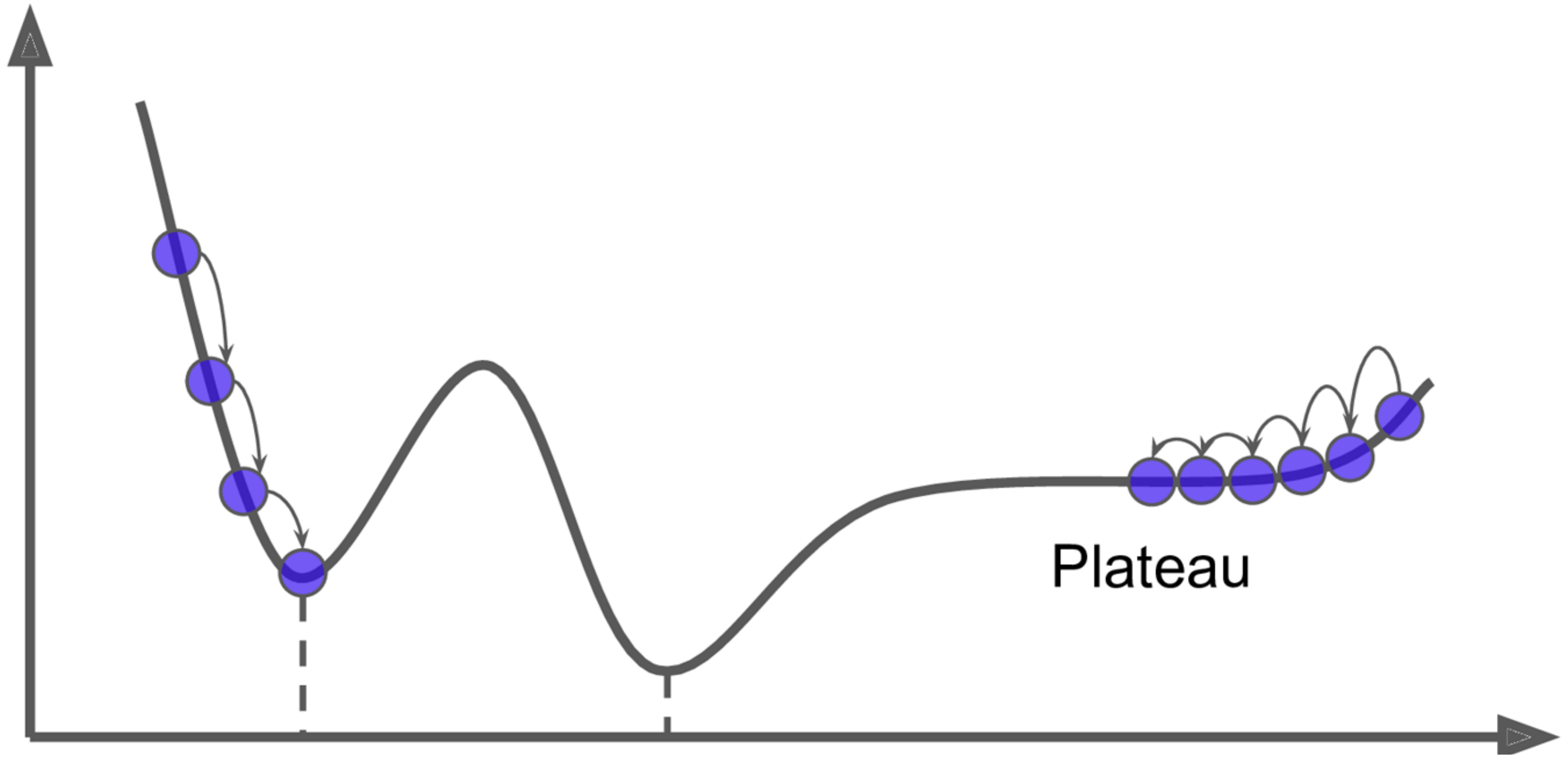


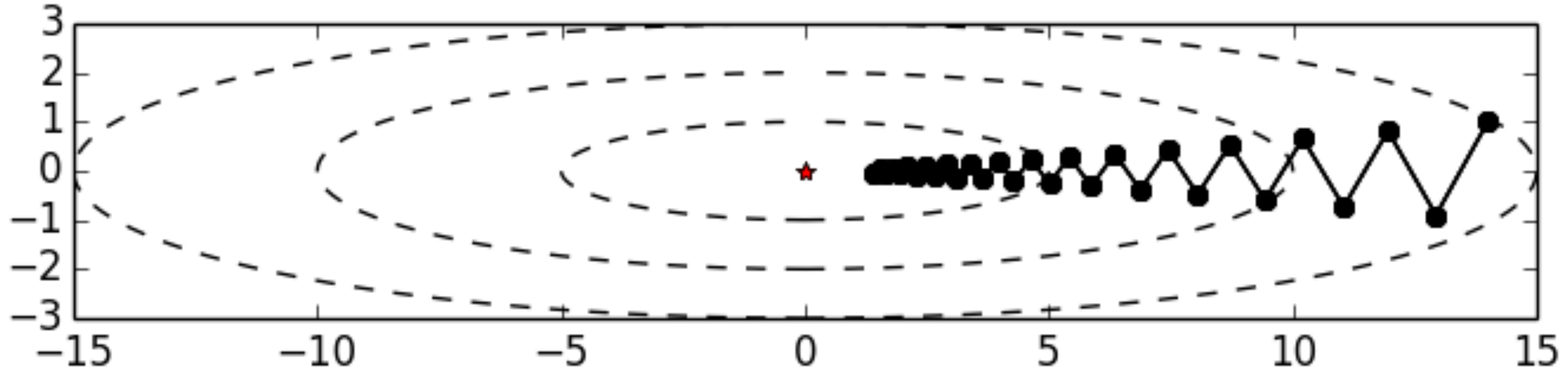


Deep Learning Book, Chapter 4 and 8

Challenges in NN optimization

Plateaus and ill-condition





Solutions: momentum and adaptive learning rate

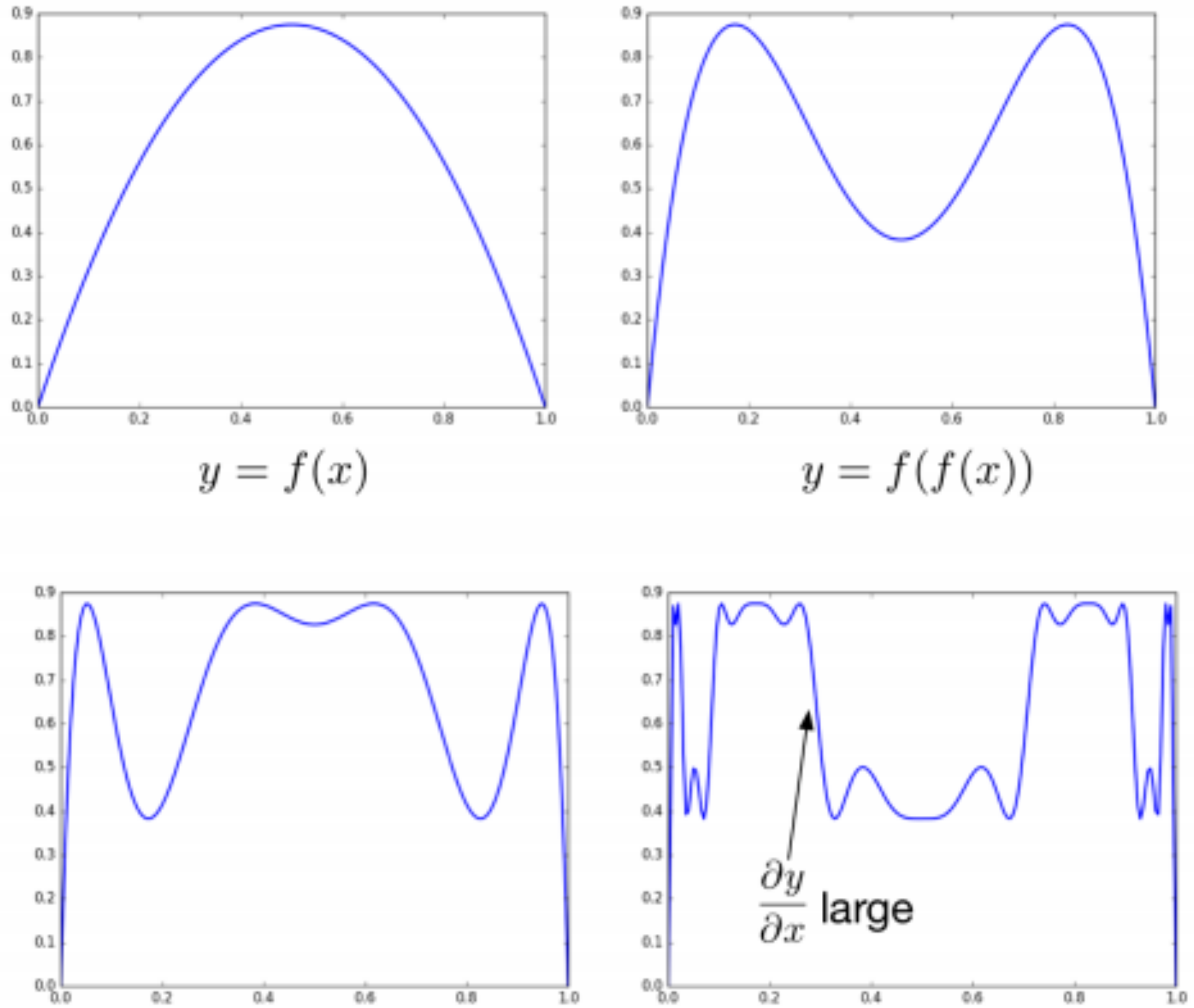
See details in “Deep Learning Book Chapter 8”

Challenges in NN optimization

Vanishing and exploding gradients

- When a neural network is too deep

$$\boldsymbol{W}^t = \left(\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}$$

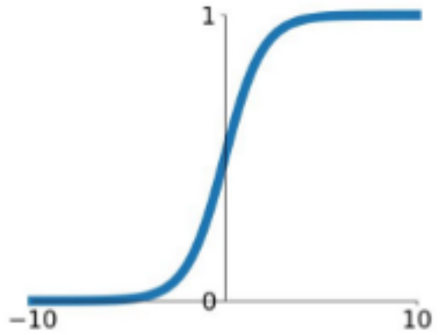


$y = f(x)$ $\quad$ $y = f(f(x))$

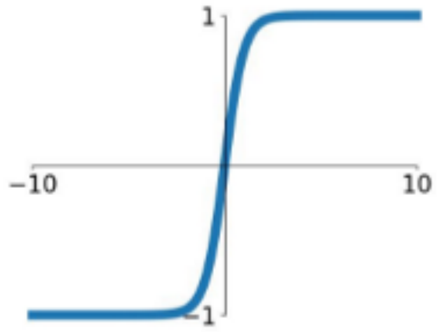
$y = f(f(f(x)))$ $\quad$ $y = \underbrace{f \circ \cdots \circ f}_{6 \text{ times}}(x)$

- When using sigmoid activation function

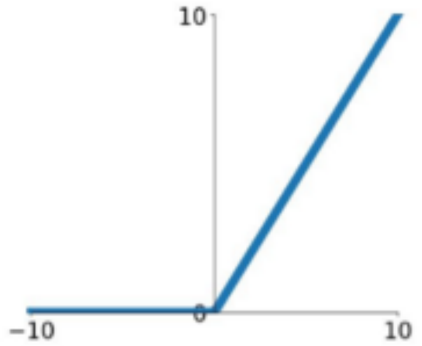
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



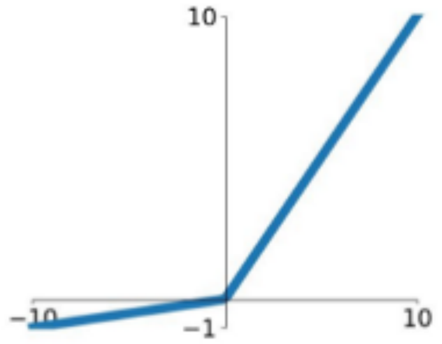
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

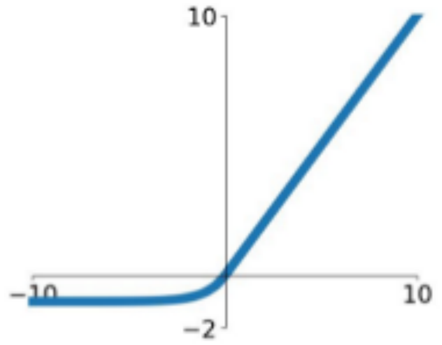


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Solutions:
Initialization; Gradient clipping; ResNets/LSTM; **Batch normalization**

arxiv.org › cs
Batch Normalization: Accelerating Deep Network Training by ...
Feb 11, 2015 — **Batch Normalization** allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some ...
by S Ioffe · 2015 · Cited by 22440 · Related articles

Optimization in deep learning

BP and beyond it

- BP: automation differentiation in neural network
- Mini-batch stochastic optimization
  - Mini-batch: dueling with big data
  - Stochastic: finding the “global” optimal points
- Gradient vanish or explosion
- Tricks
  - Batch normalization
  - Dropout
  - Regularization
  - Others: initializations, gradient clip, early stopping, …

Implementation of deep learning

Elements in deep learning

- *Data representation and organization*
- *Definition of the neural networks*
- *Optimization*
- Model store and reuse
- CPU - GPU
- Visualization
- C++ API…

Elements in deep learning

Data representation and organization

- *Data representation*
  - *Audio, language, images, parameters, …*
  - *transformations*
- *Data organization*
  - *Train, validation, test*
  - *Mini-batch optimization*
    - *Batch, mini-batch*
    - *Step, epoch*

Import torch

- *torch.tensor*
  - *Multi-dim matrix*
  - *Rich of functions for transformations, composition, change of shape…*
  - *Store the gradients*
- *torch.utils.data.[Dataset, DataLoader]*
  - *Dataset: __getitem__ and __len__*
  - *DataLoader: iterator*
    - *Shuffle dataset*
    - *Get Mini-batch dataset*

Elements in deep learning

Define neural networks

- *Neural networks*
  - *DNN, CNN, RNN, GNN*
  - *DIY network structure*
  - *Popular or latest structures in the community*

Import torch

- *torch.nn.Module*
  - *class YourNN(torch.nn.Module)*
  - *def forward(self, x)*
  - *Automatically store the parameters*
  - *Support composition*
  - *nn.Conv2d, nn.LSTM, nn.Embedding, …*

Elements in deep learning

Optimization

- *Automatic differentiation*
- *Optimizer*
  - *SGD*
  - *Adam*
  - *Adagrad*
  - *LBFGS*
  - *RMSprop, …*
- *Optimization tricks*
  - *Batch normalization*
  - *Dropout*
  - *Initialization …*

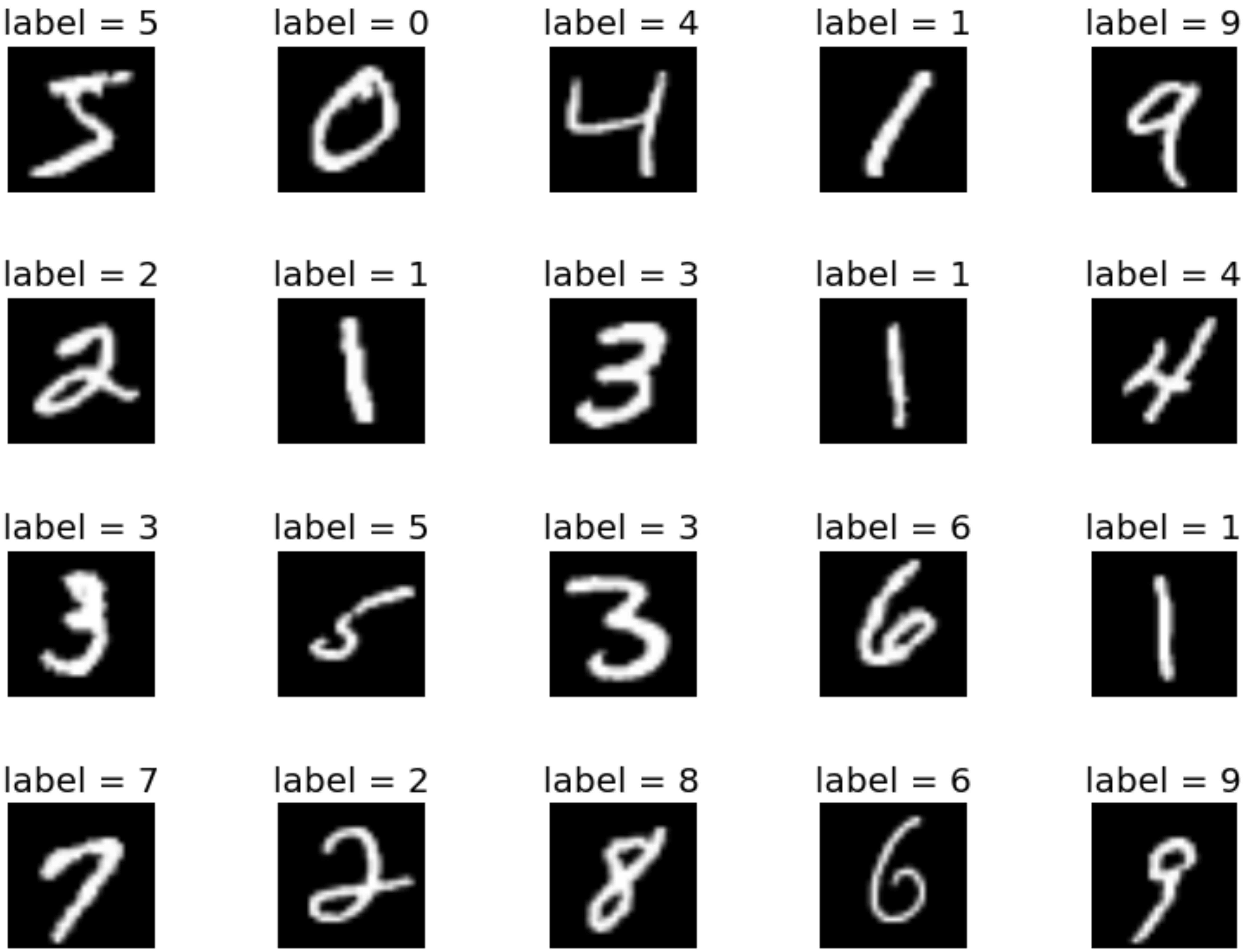
Import torch

- *torch.autograd, torch.Function, torch.Tensor*
  - *Autograd:*
    - *Backpropagation*
    - *Define-by-run*
  - *Tensor:*
    - *require_grad = True*
    - *.grad: accumulated the gradient*
    - *.grad_fn: refers to the torch.Function*
    - *.backward(): get the derivatives*
- *torch.optim*
  - *torch.optim.[optimizer_name](model.parameters(), other_arguments)*
- *torch.nn.[Normalization Layers]*
- *torch.nn.[Dropout Layers]*
- *torch.nn.init.[Methods]*

Example: VAE on MNIST

Example: VAE on MNIST

Non-linear low-dimensional representation learning



The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

https://github.com/beyondpie/intro_nn_with_torch/tree/main/homework