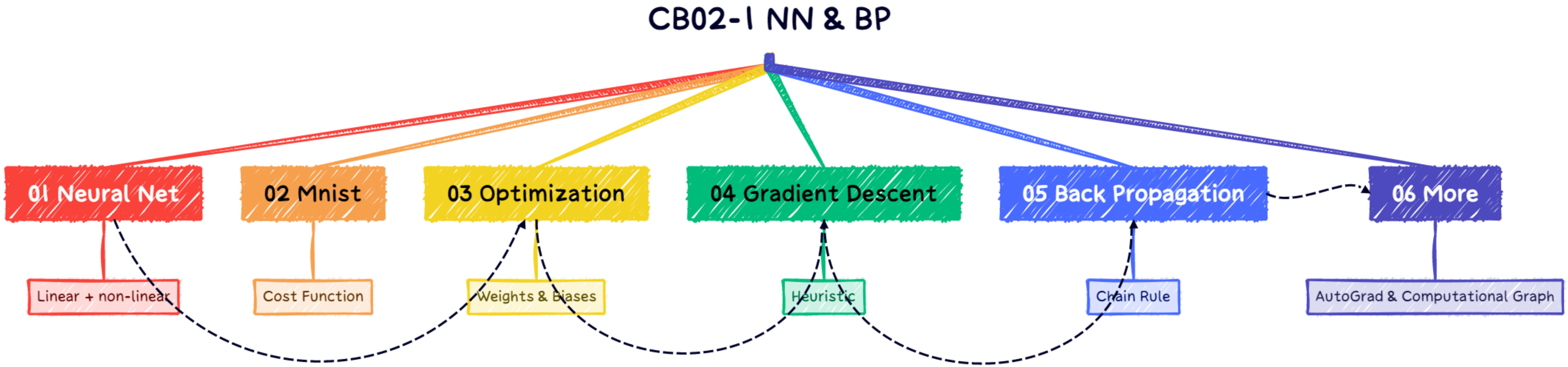
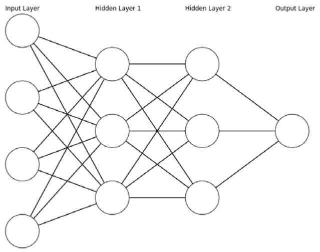
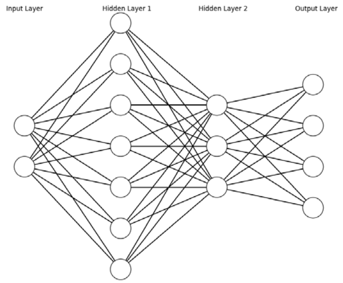
CB02-1 Neural Network & Back Propagation

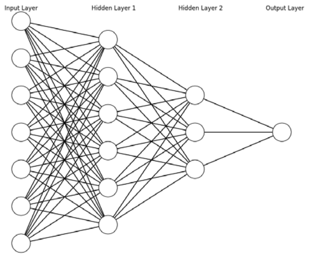
00 Outline


CB02-1 NN & BP
01 Neural Net
02 Mnist
03 Optimization
04 Gradient Descent
05 Back Propagation
06 More
Linear + non-linear
Cost Function
Weights & Biases
Heuristic
Chain Rule
AutoGrad & Computational Graph

01.1 Just Numbers!





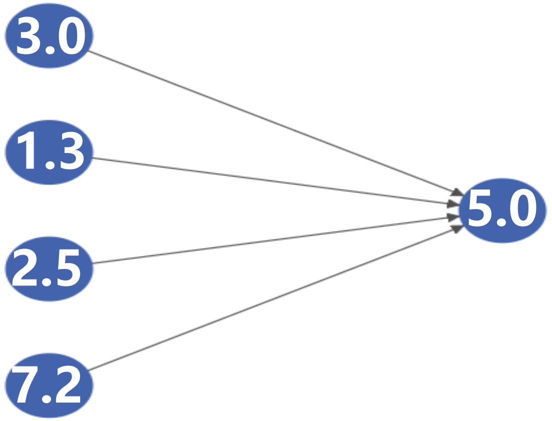


y = k x + b

k -> w : weights

y = y1 + y2 + y3 + y4

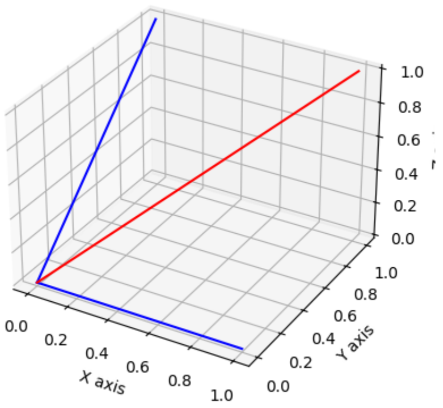
= sum(wi*xi)+ b

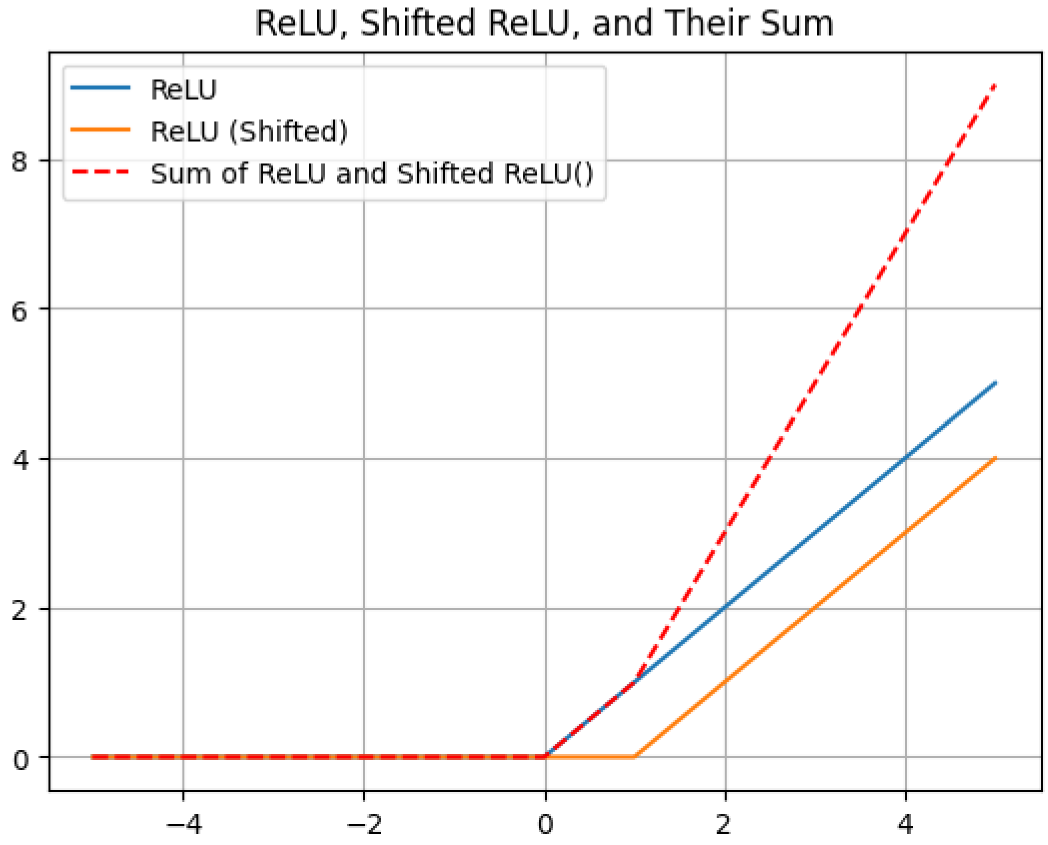


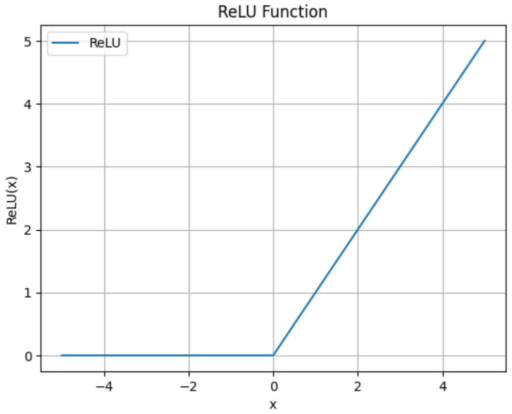
Weights & Biases

01.2 Linear & Nonlinear

NN = f(x)



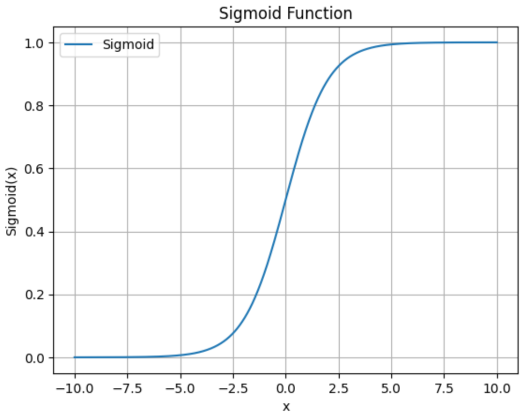




W&B

adjustment results in

desirable output



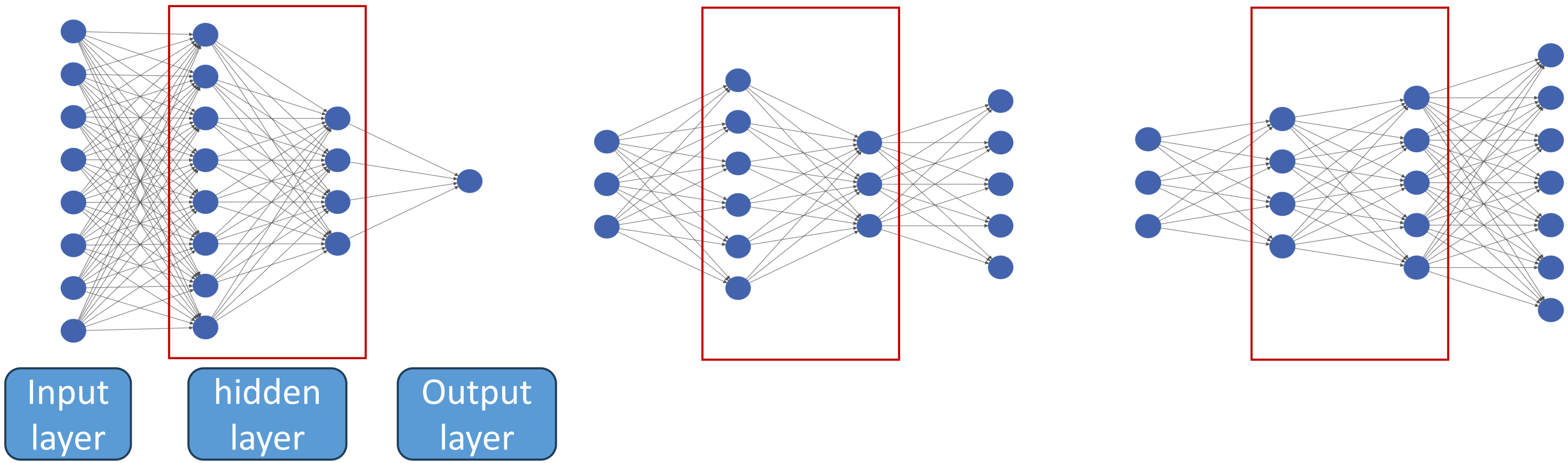
ref: [C4 of Neural networks and deep learning]

01.3 Feedforward NN

8,8,4,1->(8*8+8)+(8*4+4)+(4*1+1)

3,6,3,5->(51+15)

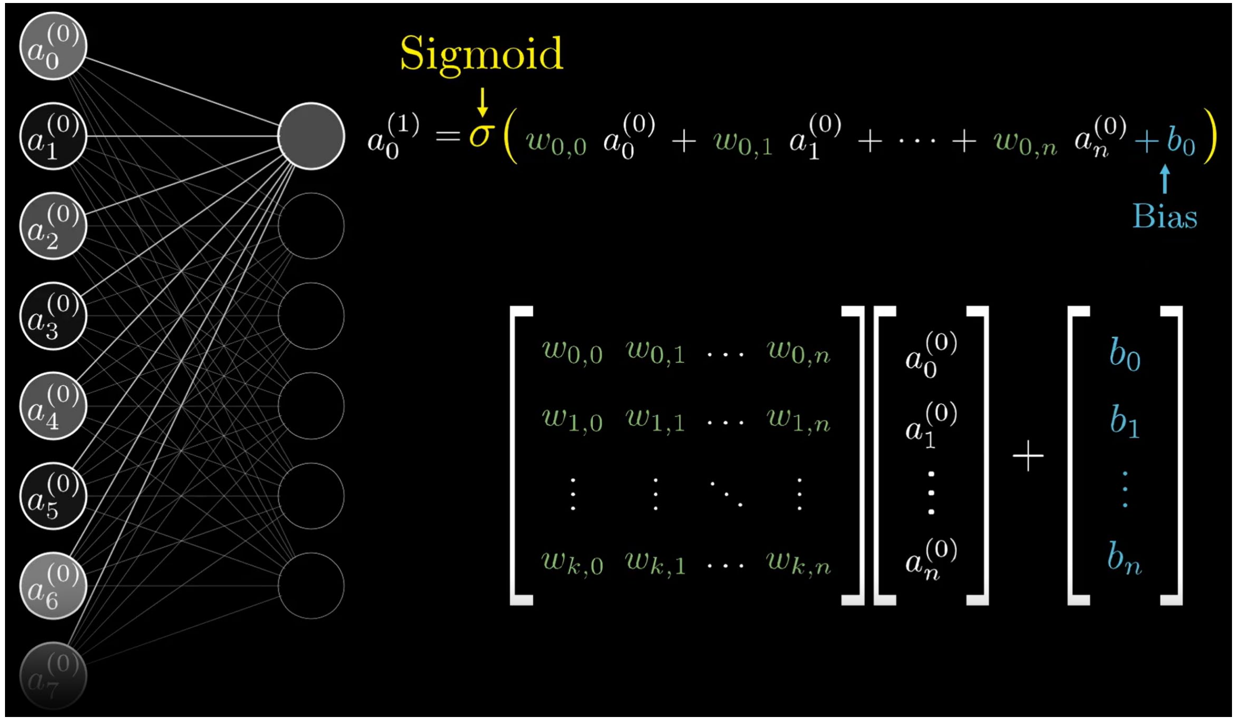
3,4,5,7->67+16

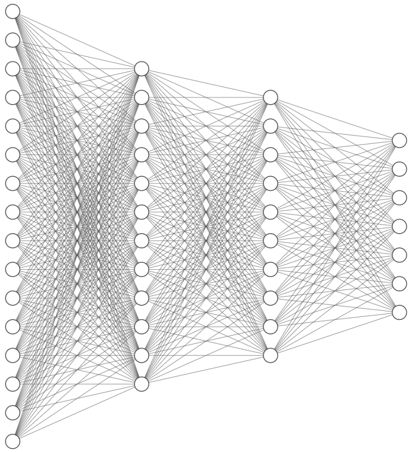


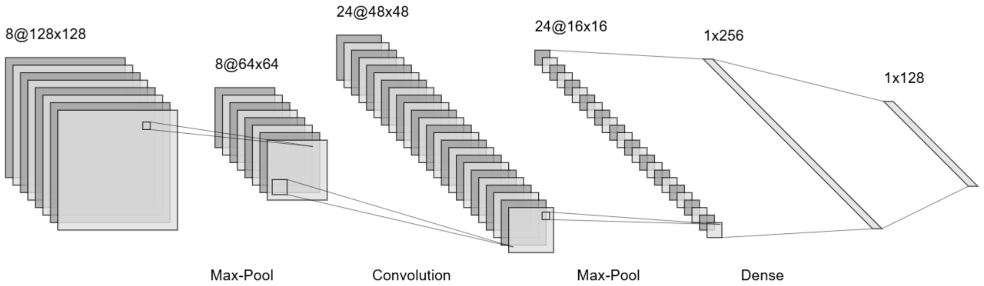
[Ref: NN created via NN SVG (alexlenail.me)]

01.4 Key Ideas

$$\sigma(w \cdot x + b)$$

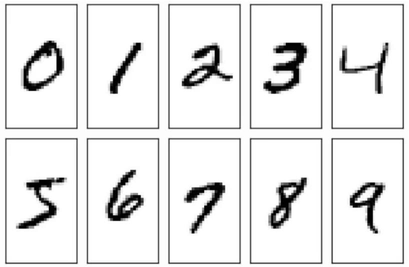




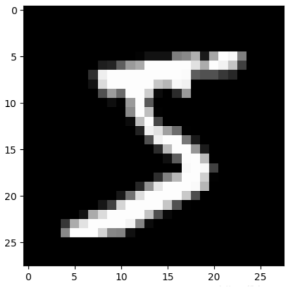


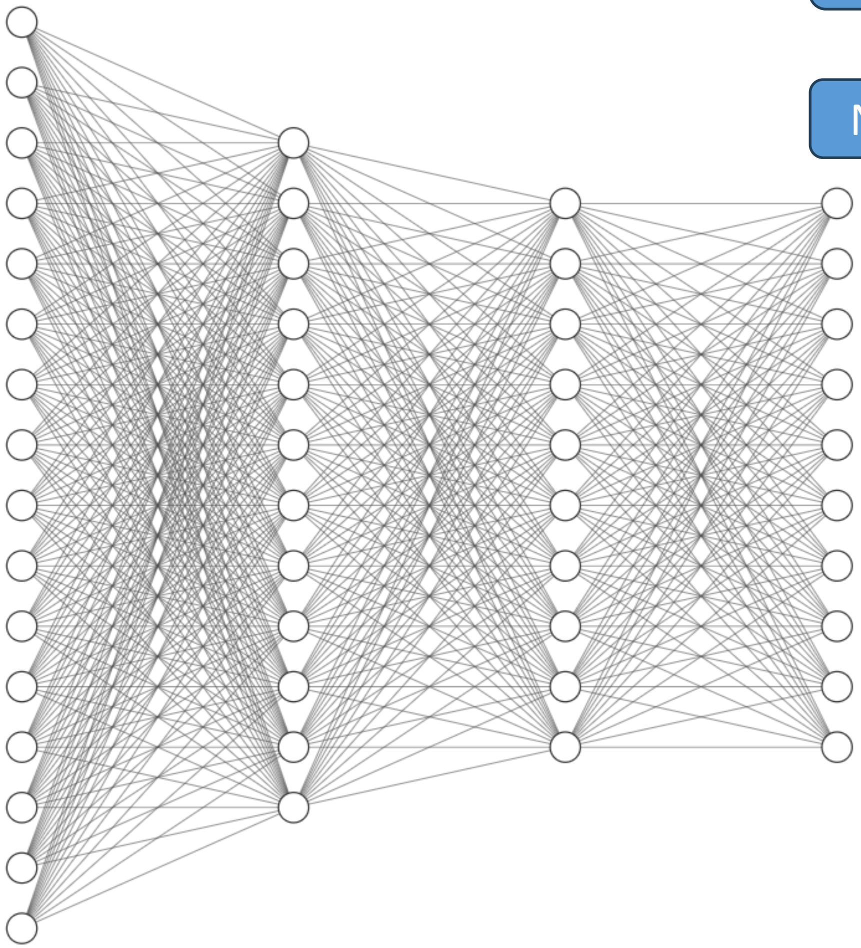
ref: [3Blue1Brown - But what is a Neural Network?]

02 Mnist & Cost Function



28*28





Cost Function：a和y之间的差距

NN输出a	真实结果y
0.01	0
0.01	0
0.01	0
0.01	0
0.21	0
0.71	1
0.01	0
0.01	0
0.01	0
0.01	0

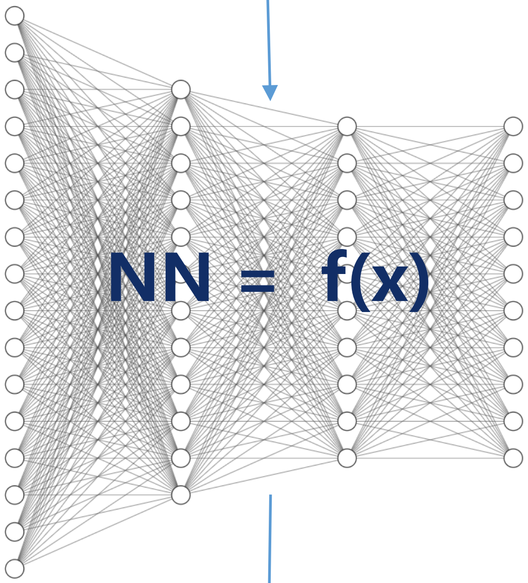
差距

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - a_i)^2$$

Ref：MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges

03 Convert to An Optimization Problem

输入



输出

第一次抽象

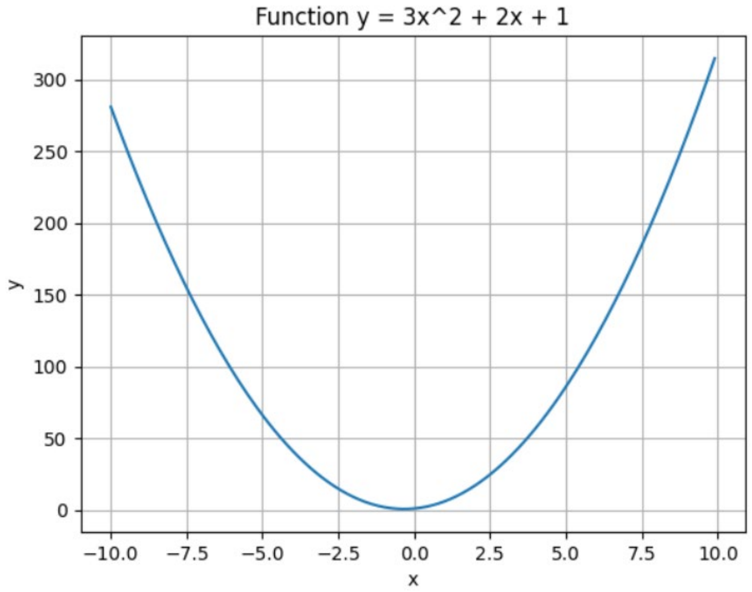
Weights & Biases

Cost Function->Minimum

WhySAI

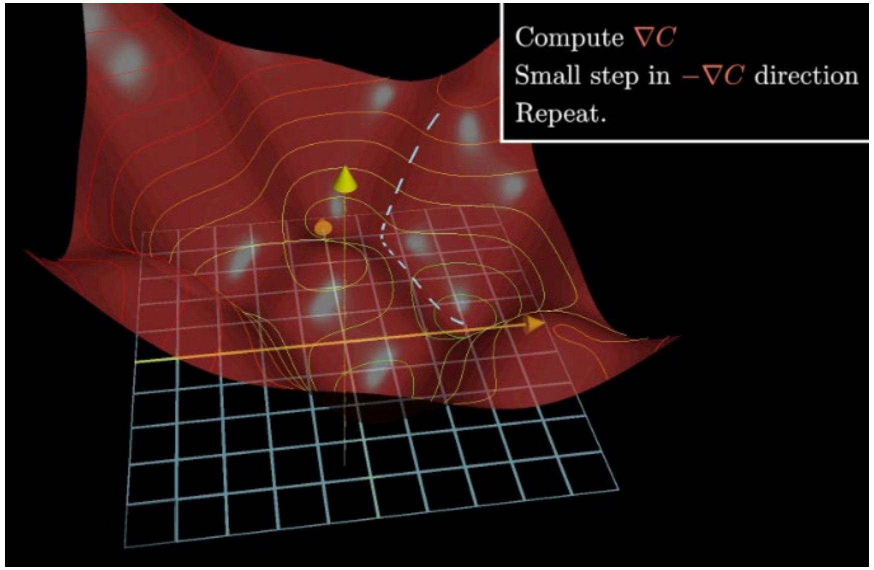
第二次抽象
启发式方法

沿着Slope方向移动



沿着梯度方向移动

$$\nabla f(w_1, w_2, \ldots, w_n, b_1, b_2, \ldots, b_n) = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \ldots, \frac{\partial f}{\partial w_n}, \frac{\partial f}{\partial b_1}, \frac{\partial f}{\partial b_2}, \ldots, \frac{\partial f}{\partial b_n}\right)$$



Ref: 3Blue1Brown - Gradient descent, how neural networks learn

WhysAI

GD

假设100个输入

计算100个∇C，取平均值

对所有W&B，更新$(-\eta\nabla C)$

SGD

随机取1个输入

计算1个∇C

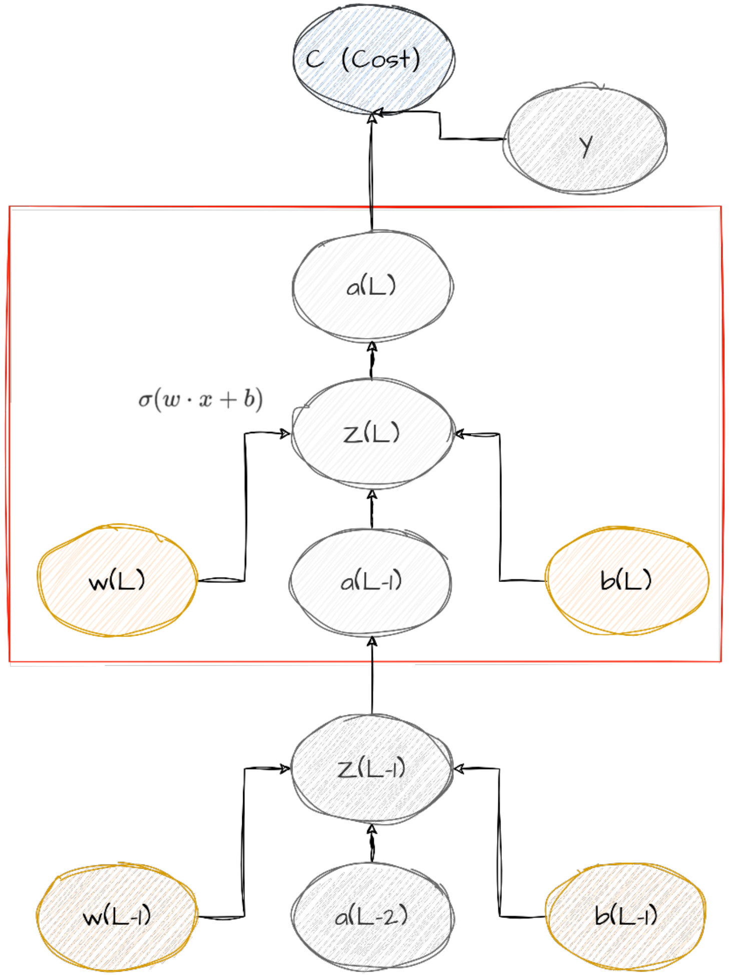
对所有W&B，更新$(-\eta\nabla C)$

Mini-batch SGD：取n个更新一次$-\eta\nabla C$

WhysAI

第三次抽象：核心任务是计算C(对于W&B的)梯度，也就是C对于每一个weights & Biases的偏导数

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}$$



C->a的偏导数

a->Z的偏导数

Z->W和B的偏导数：
需要上一层的输出a(L-1)

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

05.2 BP = Forward Pass+Backpropagate (Optional)

1. **Input x:** Set the corresponding activation a^1 for the input layer.
2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

Summary: the equations of backpropagation

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

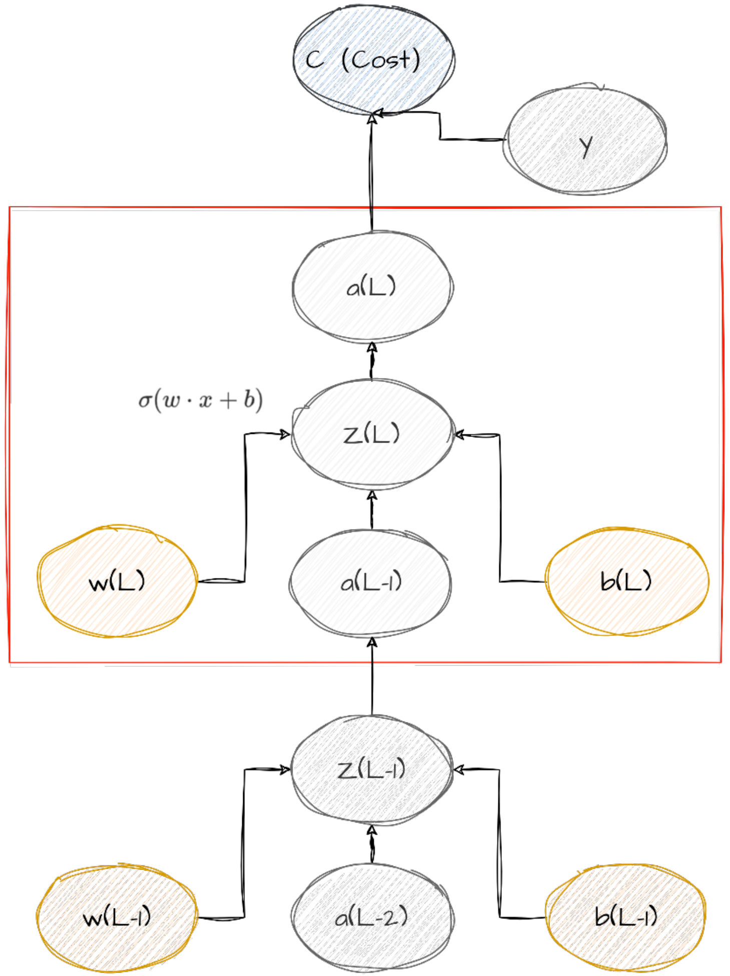
$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

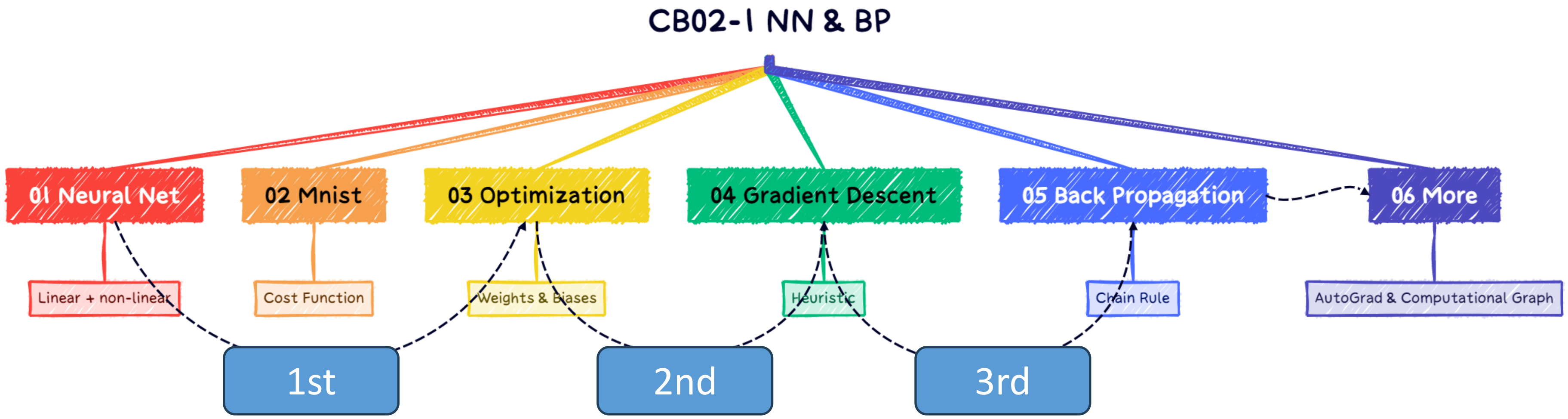
Ref: C2 of Neural networks and deep learning

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix}$$



Review & Next Video

WhySAI




```python
def backprop(x, y):
    # Feedforward Pass
    activations, linear_outputs = feedforward(x)

    # Backward Pass
    delta_w = [np.zeros(w.shape) for w in network.weights]
    delta_b = [np.zeros(b.shape) for b in network.biases]

    delta = cost_derivative(activations[-1], y) * \
        (opt.sigmoid_prime(linear_outputs[-1]))
    delta_w[-1] = np.dot(delta, activations[-2].transpose()) # Last/Output Layer
    delta_b[-1] = delta # Last/Output Layer

    for l in range(2, network.num_layers): # only index of layer changes
        delta = np.dot(network.weights[-l+1].transpose(), delta) * \
            opt.sigmoid_prime(linear_outputs[-l])
        delta_w[-l] = np.dot(delta, activations[-l-1].transpose())
        delta_b[-l] = delta
    return (delta_w, delta_b)
```

```
Epoch 0: 9248 / 10000, precision is 92.48%
Epoch 1: 9256 / 10000, precision is 92.56%
Epoch 2: 9247 / 10000, precision is 92.47%
Epoch 3: 9274 / 10000, precision is 92.74%
Epoch 4: 9255 / 10000, precision is 92.55%
Epoch 5: 9282 / 10000, precision is 92.82%
Epoch 6: 9232 / 10000, precision is 92.32%
Epoch 7: 9272 / 10000, precision is 92.72%
Epoch 8: 9302 / 10000, precision is 93.02%
Epoch 9: 9287 / 10000, precision is 92.87%
Epoch 10: 9282 / 10000, precision is 92.82%
Epoch 11: 9285 / 10000, precision is 92.85%
Epoch 12: 9275 / 10000, precision is 92.75%
Epoch 13: 9287 / 10000, precision is 92.87%
Epoch 14: 9301 / 10000, precision is 93.01%
Epoch 15: 9311 / 10000, precision is 93.11%
Epoch 16: 9325 / 10000, precision is 93.25%
```

Refs & Materials

Main Refs:

1. Michael A. Nielsen, Neural networks and deep learning, github repo (github.com)

2. The first 3/4 videos in 3B1B's "Neural networks") series: 3Blue1Brown个人主页 (bilibili.com)

3. Video Lecture on MicroGrad By Andrej Karpathy: Andrej Karpathy | 详解神经网络和反向传播

4. Computational Graph on BP from Chris Olah: Calculus on Computational Graphs

Others:

1. Chapters 3-5 of the d2l.ai: Dive into Deep Learning (d2l.ai)

2. Neural Networks from Scratch in Python Book : (nnfs.io)

3. CS231n's Python Numpy Tutorial Python Numpy Tutorial (with Jupyter and Colab)

4. Numpy Tutorial: 《编程不难》--C13-C18: Visualize-ML/Book1_Python-For-Beginners: Book_1

5. An efficient pure-PyTorch implementation of KAN: efficient-kan

6. NN SVG (alexlenail.me)