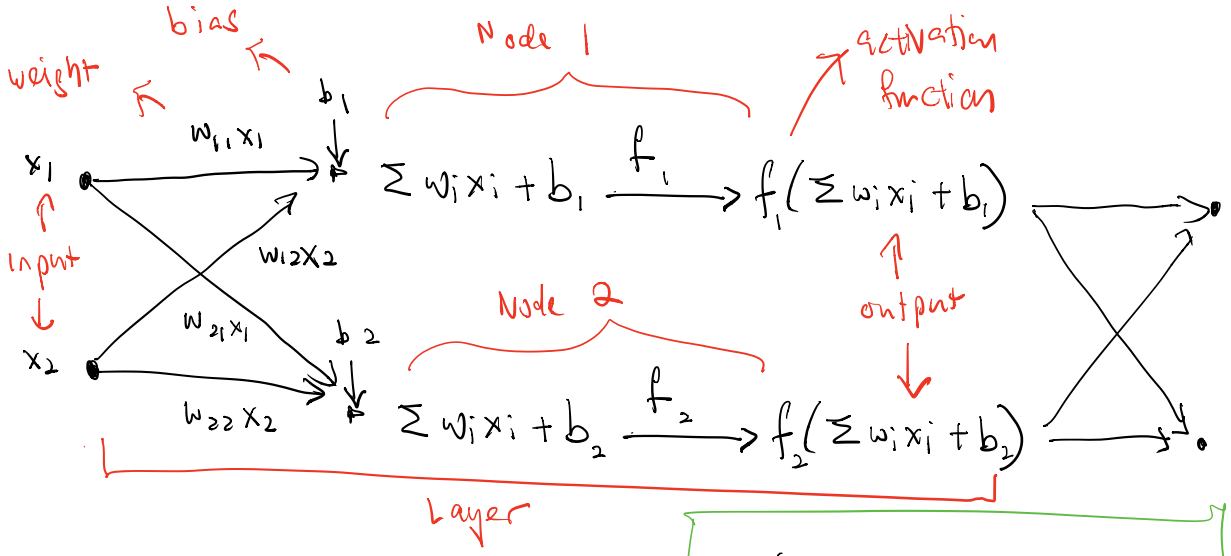
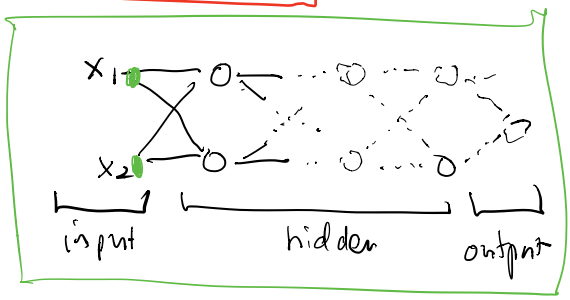
Anatomy





Each layer implements this function

$$[y] = f([W][x] + [b])$$

affine transformation (linear + translation)

Affine can be expressed as an augmented matrix

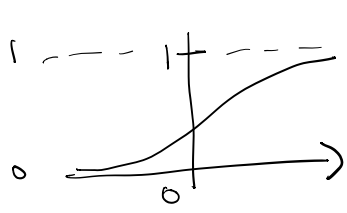
$$\begin{bmatrix} ax_1 + bx_2 + c \\ dx_1 + ex_2 + f \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

If activation function is linear → Matrix multiplication

$$y = A_5 A_4 A_3 A_2 A_1 x$$

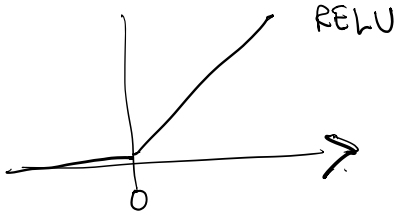
$$= Bx$$

Activation functions

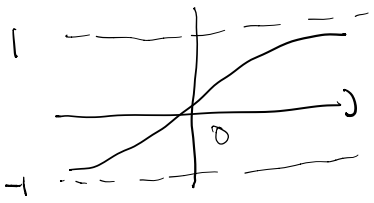


Sigmoid

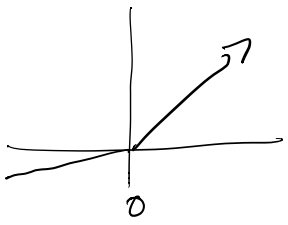
$\frac{1}{1+e^{-z}}$



RELU



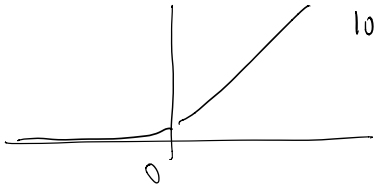
tanh

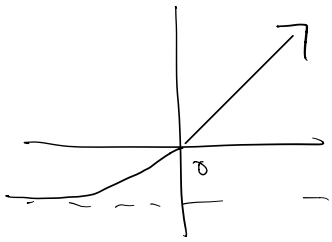


Leaky RELU

softplus

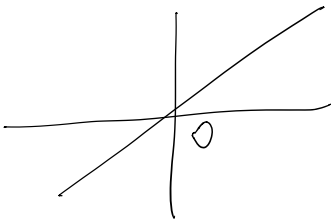
$\log(1+e^{x})$





ELU

linear



Training

θ = Collection of weights w_{ij}^k & biases b_i^k

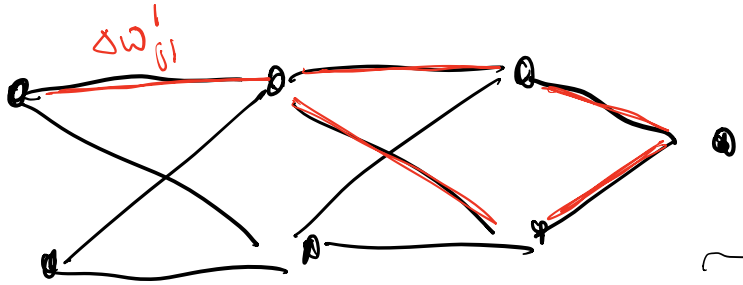
Loss with batch data & params

$$\theta_{k+1} = \theta_k - \alpha \frac{\partial L(X, \theta_k)}{\partial \theta}$$

This is just gradient descent

How is gradient calculated?

Sketch of main idea of backpropagation



$$\Delta L \approx \frac{\partial L}{\partial w^l_{jk}} \Delta w^l_{jk}$$

$$a^L_m = f(wx+b)$$

activation

layer = L

node = m

For one path

$$\Delta L \approx \frac{\partial L}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^1_j}{\partial a^1_j} \frac{\partial a^1_j}{\partial w^1_{jk}} \Delta w^1_{jk}$$

chain rule

For all paths

$$\Delta L \approx \sum \frac{\partial L}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^1_j}{\partial a^1_j} \frac{\partial a^1_j}{\partial w^1_{jk}} \Delta w^1_{jk}$$

Expression for gradient

$$\frac{\partial L}{\partial w^L_{jk}} \approx \sum \frac{\partial L}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^1_j}{\partial a^1_j} \frac{\partial a^1_j}{\partial w^1_{jk}}$$

Automatic differentiation

Tensorflow uses reverse mode autodiff

Autodiff → rules & differentiate simple expressions by substitution eg. $\frac{d}{dx} x^n \rightarrow n x^{n-1}$

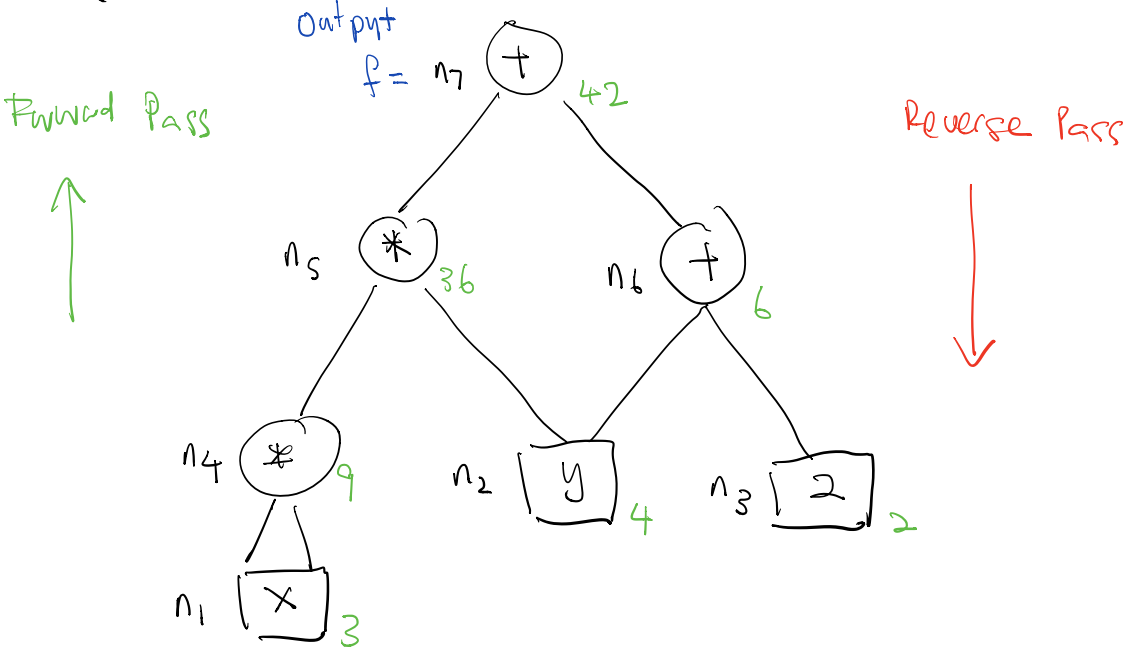
Example

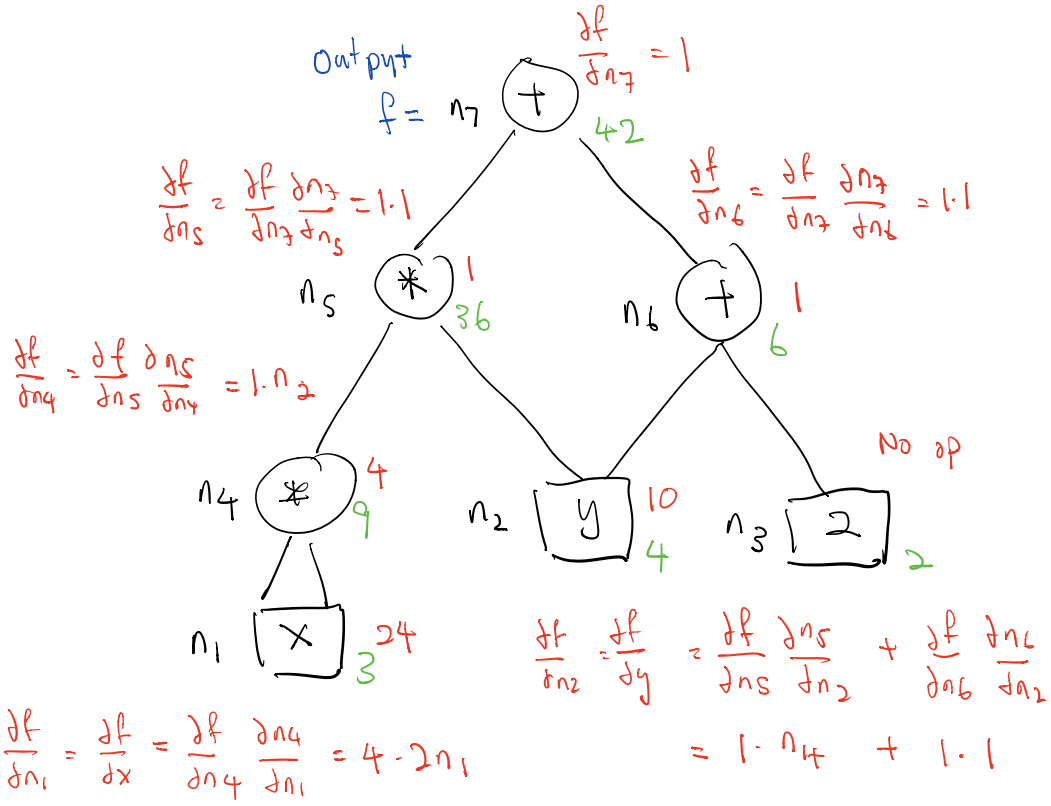
$f(x, y) = x^2 y + y + 2$ Inputs $x = 3$, $y = 4$ } variables; $2 = 2$ } constant

Find $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$

$\frac{\partial f}{\partial x} = 2xy = 24$ $\frac{\partial f}{\partial y} = x^2 + 1 = 10$

Computation Graph



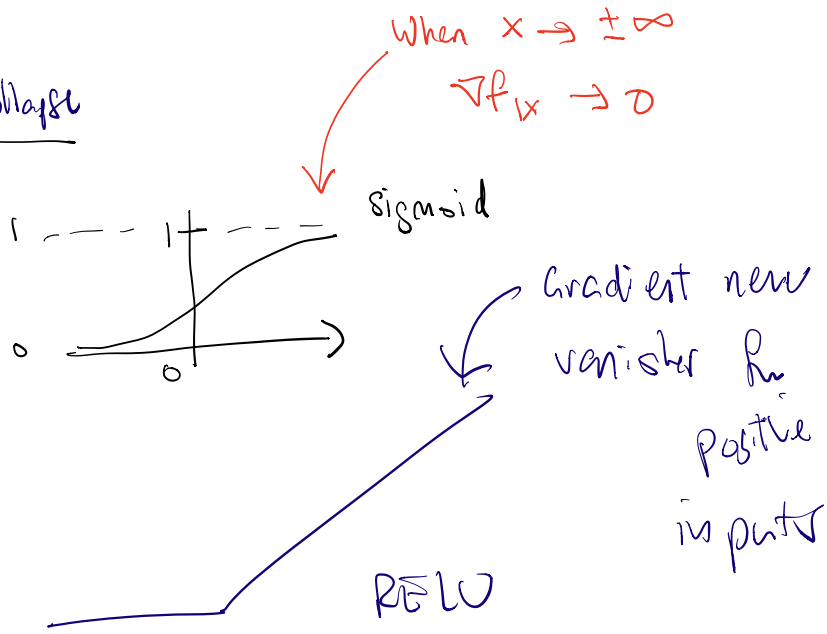


Regardless of # variables, only 2 passes needed to calculate all partial derivatives.

Gradient issues

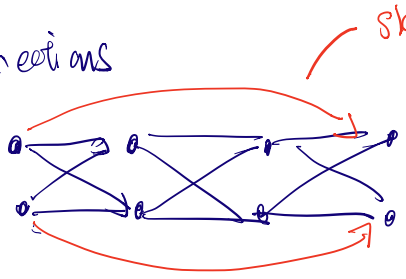
Deep networks have many layers.

Gradient collapse



①

② Skip connections



Gradient explosion

① Gradient clipping

② Batch normalization

Batch normalization

μ_B = batch mean

σ_B = batch standard deviation

] Estimated from inputs

$$\hat{x} = \frac{x - \mu_B}{\sigma_B + \epsilon}$$

$$z = \gamma x + \beta$$

Learned during backprop

Learns optimal scaling for each layer.

Batch, mini-batch & Stochastic gradient descent

Deep learning often uses large training sets – e.g. 10^6 images

Training size 10^6
Batch size 10^2
Iteration per epoch 10^4

} With batch size of 10^2, we have 10^4 gradient updates instead of 1 to same work

Gradient $\approx 10^2$ likely to be good approximation of gradient $\approx 10^6$

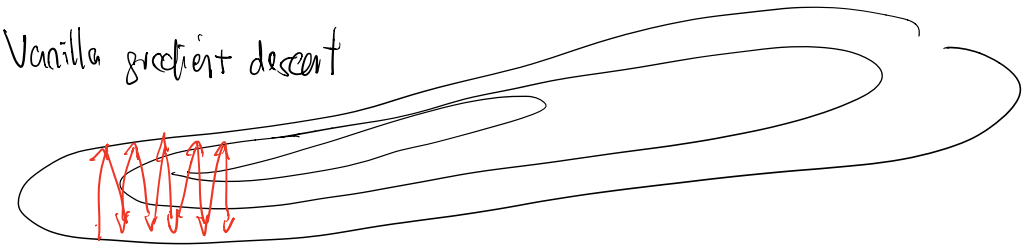
When batch size = 1 ⟶ Stochastic gradient descent
SGD

Optimal batch size is tunable hyperparameter

depends on variability of training set,
hardware specs (e.g. amount of GPU RAM)

Optimizing gradient descent

Vanilla gradient descent



① Momentum

$$\text{Vanille} \quad \theta_{k+1} = \theta_k - \alpha \nabla f(\theta_k)$$

* In physical systems → gradient effects velocity ~ momentum, not position directly.

"momentum" "friction" ~ $\beta = 0.9$

$$m_{k+1} = \beta m_k - \alpha \nabla f_{\theta_k}$$

$$\theta_{k+1} = \theta_k + m_{k+1}$$

① $\beta = 0.9$ accumulates history of gradients opposite cancels out same reinforces ↓ oscillation

Suppose ∇f is constant, say $\nabla f = 1$

$$m_0 = 0$$

$$m_1 = \beta(0) + \eta = \eta$$

$$m_2 = \beta\eta + \eta = \eta(1+\beta)$$

$$m_3 = \beta^2\eta + \beta\eta + \eta = \eta(1+\beta+\beta^2)$$

$$m_4 = \beta^3\eta + \beta^2\eta + \beta\eta + \eta = \eta(1+\beta+\beta^2+\beta^3)$$

$$m_\infty = \frac{1}{1-\beta}\eta$$

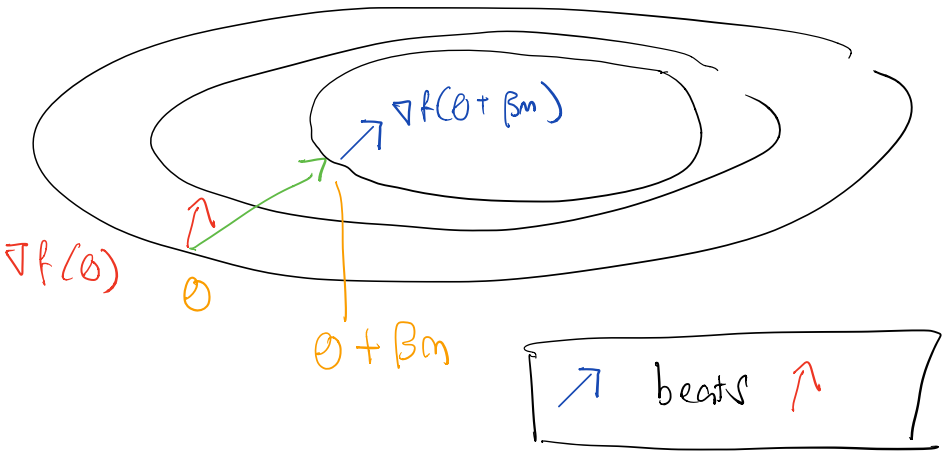
② If $\beta = 0.9$ terminal velocity is 10× more than vanilla.

Nesterov accelerated gradient

$$m_{k+1} = \beta m_k - \nabla f(\theta_k + \beta m_k)$$

$$\theta_{k+1} = \theta_k + m_{k+1}$$

Basic idea → evaluate gradient not at current position, but somewhere along direction of momentum.



RMSProp

decay $\gamma = 0.9$

element-wise multiplication

$$s \leftarrow \gamma s + \nabla f(\theta) \otimes \nabla f(\theta)$$

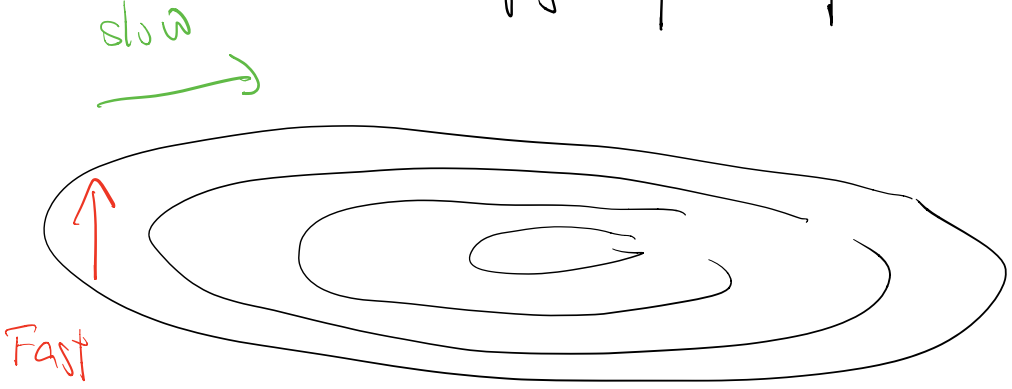
$$\theta \leftarrow \theta - \alpha \nabla f(\theta) \oslash s$$

element wise division

Basic idea

Reduce gradient in directions where it is changing the fastest

Increase gradient in direction where it is changing very slowly



Helps escape long narrow valleys

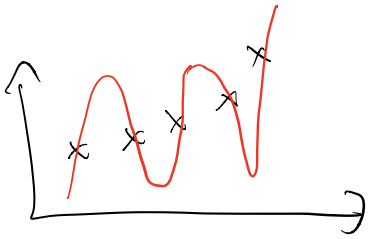
ADAM (momentum + RMSprop)

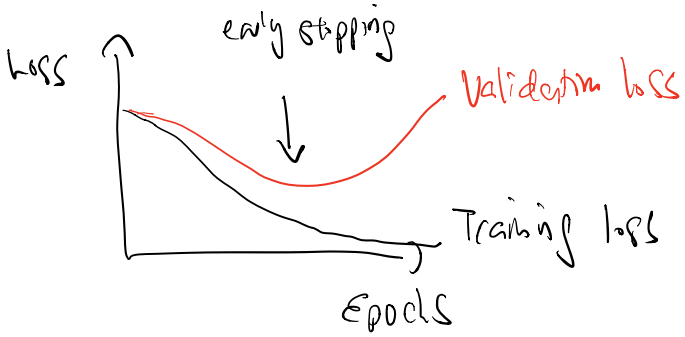
NADAM (+ Nesterov)

ADAptive Moment estimation

Regularization

Overfitting



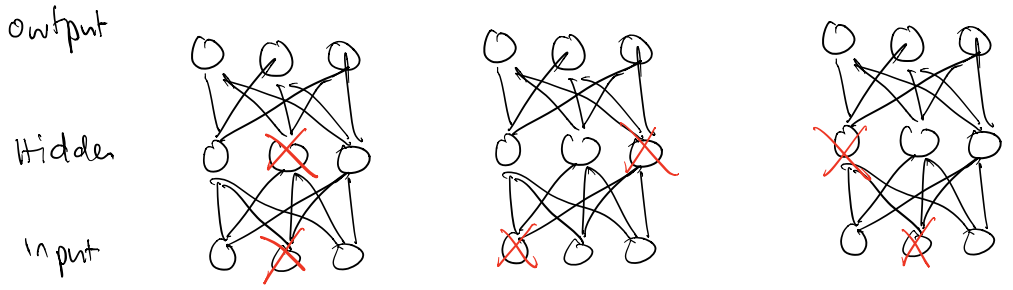


Remedies

1. ↑ data
2. Early stopping
3. L1 or L2
4. Dropout

Dropout

During train



Iteration 1 Iteration 2 Iteration 3

At every step → a fraction p of nodes is inactivated (including input but excluding output)

(1) Each step train a DIFFERENT neural network
→ Ensemble perspective

(2) No single node can be very important
→ Regularization perspective

During test

All nodes are active

weights are multiplied by $1-p$

Review

① Anatomy of neural network

② What a single node does $z = f(\Sigma wx + b)$

③ Update parameters by gradient descent

i) gradient calculated by backpropagation

ii) implemented by reverse mode auto differentiation

④ Collapsing / exploding gradients

i) choice of activation function

ii) skip connections

iii) gradient clipping

iv) batch normalization

⑤ Optimizing gradient descent

i) momentum

ii) Nesterov

iii) RMSProp

iv) ADAM

⑥ Overfitting / regularization

i) Dropout