4/4

Neural Network

Forward Propagation and Back Propagation

Backward Propagation: $\frac{\partial C}{\partial b^{(2)}} = \frac{\partial C}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial b^{(2)}} = \frac{1}{n}\sum_{i=1}^{n} y_i - \sigma(u^{(2)})$

update w^2 and b^2: $\begin{bmatrix} w_{new}^2 \\ b_{new}^2 \end{bmatrix} = -\alpha \begin{bmatrix} \frac{\partial C}{\partial w^2} \\ \frac{\partial C}{\partial b^2} \end{bmatrix} + \begin{bmatrix} w^2 \\ b^2 \end{bmatrix}$

rule: go from backward and update each weight

loss: $cost(w, b)$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\sigma(-w^T h_i + b)) + (1-y_i)\log(1-\sigma(-w^T h_i + b))\right]$$

Tuning Parameters

8 steps:
1) step size α
2) number of Back Propagation iterations
3) Batch size
4) number of hidden layers
5) size of each hidden layer
6) activation function
7) Cost function
8) Regularization

Activation Function

introduce non-linearity

Tuning the activation function is equivalent to feature engineering

ex: "Relu" activation function. It's for approximating non-linear function

Challenges

1) overcome overfitting
2) when dimensionality increases, # of weights increase and gradient ↓

Regularization

① early stopping (i.e. early termination of weights)

② dropout: dropout neurons in the layers randomly

CNN

we want filter that reduce number of weights, capture features

use pooling to reduce weights

layers: convolutional layer w/ nxn kernal

max-pooling (to reduce/downsize feature map)

also, use stride > 1 can also reduce the sizes of input

→ downsize feature map

Application: Image Classification, Computer Vision

RNN

handling sequences of input

Application: Language translation, predicting next word

Speech Recognition, Video Tagging

ex: for hand gesture Recognition, given input: video clips

output: predicted class of hand gest

use CNN then RNN

image → CNN

video → RNN