Linear regression → Logistic regression

$\sigma(z) = \frac{1}{1+e^{-z}}$ logistic activation (sigmoid)

Activation functions

- Sigmoid — (two-class) → multiclass : softmax $\frac{e^i}{\sum_{i=j} e^i}$ ($j=2$)
- tanh
- ReLU : $\max(0,z)$ — Rectified linear unit
- Leaky ReLU — $f(y)=y$ ($a \neq 0.01$: randomised ReLU); $f(y)=ay$

Gradient descent

Loss function $\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ e.g. logistic loss $= -[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})]$

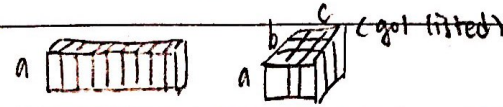
Cost function $J(w,b) = \frac{1}{m}\sum^{m}\mathcal{L}$

Python

rank 1 array : column/row vector

```
flatten_x = x.reshape(x.shape[0], -1).T
```

a x b c d ← a x b x c x d



Max likelihood estimation

	High bias (underfit)	→ ✓ →	High variance (overfit)
model complexity	low		high
training error	high	reasonably low	low
validation error	(high)	reasonably low	high
solutions	bigger network longer training		regularization more data

Gradient checking

Regularization ① $J(w,b) = \frac{1}{m}\sum^{m}\mathcal{L} + \sum R(w_j)$

l_1 regularization $\frac{\lambda}{2m}\|w\|$ $\quad \|w\| = \sum w_j$

l_2 regularization $\frac{\lambda}{2m}\|w\|_2^2$ $\quad \|w\|_2^2 = \sum w_j^2 = w^T w$

→Direct effect : smaller / less w_j → less complex ← thus

i.e. as $\lambda\uparrow$ wt $z = wa+b\downarrow$ with activation function, every layer ≈ linear

② Dropout keep-prob (=1 is no dropout)

inverted dropout { train time : reassign probability
test time : no change

向矩阵*=(np.random.rand(向shape)<p)/p

③ Early stopping

Weight Initialization of W (partially helps vanishing gradient problem)

$W^{[l]} = np.random.rand(shape) * \sqrt{\quad}$

- $\tanh$: $\sqrt{\frac{1}{n^{[l-1]}}}$ (Xavier init)
- $\sqrt{\frac{2}{n^{[l]}+n^{[l-1]}}}$
- $\sqrt{\frac{2}{n^{[l]}}}$ → which give $var(w_j) = \frac{2}{n}$

Optimization

Mini-batch Gradient Descent

$X = [x^{(1)} x^{(2)} \cdots x^{(n)} | \cdots | \cdots]$

$x^{\{1\}}$ $x^{\{2\}}$ ... $x^{\{n\}}$

total = a (epoch) × n (mini-batch)

↳ $m \leq 2000$: batch GD, size a = all

↳ usually 2^k

{ a = 1 : stochastic (one by one)

Exponentially weighted average $\bar{X}_n = \beta \bar{X}_{n-1} + (1-\beta)\theta_n$ (old average, new input)

for approx over $\frac{1}{1-\beta}$ day

→ bias correction, for warm-up

① GD with momentum $\begin{cases} V_{dW} = \beta_1 V_{dW} + (1-\beta_1) dW \\ V_{db} = \beta_1 V_{db} + (1-\beta_1) db \end{cases}$ usually 0.9

then update $\begin{cases} W = W - \alpha V_{dW} \\ b = b - \alpha V_{db} \end{cases}$ to be tuned

intuitive: controlling learning rate (vertically ↓, horizontally ↑)

b ↑ → w ↑ more

② RMSprop (root mean square) usually 0.999

$\begin{cases} S_{dW} = \beta_2 S_{dW} + (1-\beta_2)(dW)^2 \\ S_{db} = \beta_2 S_{db} + (1-\beta_2)(db)^2 \end{cases}$ then update $\begin{cases} W = W - \alpha \frac{dw}{\sqrt{S_{dW}} + \varepsilon} \\ b = b - \alpha \frac{db}{\sqrt{S_{db}}} \end{cases}$

usually 10^{-8} (to avoid 0 denominator)

ADAM optimization (adaptive moment estimation)

① $V_{dW} = S_{dW} = V_{db} = S_{db} = 0$ (init)

② mini-batch iteration: $V_{dW}^{corrected} = \frac{V_{dW}}{1-\beta_1^t}$ $V_{db}^{corrected} = \frac{V_{db}}{1-\beta_1^t}$

$S_{dW}^{corrected} = \frac{S_{dW}}{1-\beta_2^t}$ $S_{db}^{corrected} = \frac{S_{db}}{1-\beta_2^t}$

$W = W - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}} + \varepsilon}$ $b = b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$

Learning rate delay

$\alpha = \frac{1}{1 + \text{decayrate} * \text{epochnum}} \alpha_0$

$0.95^{\text{epoch num}} \alpha_0$ (exponentially)

$\frac{k}{\sqrt{\text{e.n.}}} \alpha_0$

$\frac{k}{\sqrt{t}} \alpha_0$

stair case

or manually

Saddle point, plateau (rather than local optima)

Hyperparameter tuning

Panda vs Caviar (many in parallel) model

Batch normalization $z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$ for non-input layer(s)

* but don't want norm distribution at all time $\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$

pro: covariate shift (not quite matching train ↔ test)

con: like dropout, will introduce noise

and at test time:

use μ, σ^2 from exponentially weighted average (across mini-batches), ie. "running average"

→ Week3 intro to TensorFlow

Orthogonalization : one variable tuned at a time

Evaluation metric

Performance

Human level
↕ avoidable bias → bigger model, longer training, better optimization algorithms
training ε
↕ high variance → NN architecture / param search
train-dev ε
↕ data mismatch
dev & test ε.

Transfer learning vs. multi-task learning

eg. pretrained XXY

A&B same input	data A>>B	low level features recyclable	A&B data	shared low level similar features	→ can train (joint) NN to optimise for all tasks

End-to-end: no intermediate nodes / constructs

audio → ~~features~~ → ~~phonemes~~ → ~~words~~ → transcript

CNN

Case studies

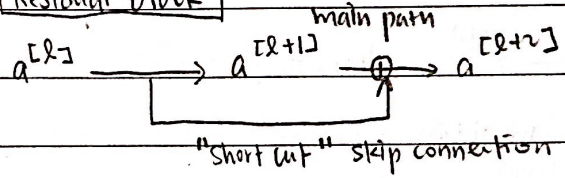
LeNet-5

AlexNet

VGG

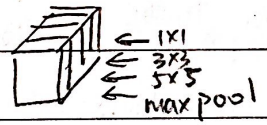
ResNet

Residual block



Inception

1×1 convolution : 加和 flatten

Inception 

(to shrink size without losing info)

Data Augmentation

- Mirroring
- Random cropping
- Rotation
- Shearing 
- Local warping

Colour shifting e.g. PCA (principles component analysis)

(Computer vision, neural style transfer)

Object Localization

Landmark Detection

Sliding window

Bounding box — YOLO = you only look once

intersection over union = $\frac{\text{intersection}}{\text{union}}$ ≥0.5 - correct

non-max suppression

anchor boxes for each cell

region proposals : segmentation algorithm
- only run sliding windows on interesting blobs/regions

Face verification vs. recognition

1:1 1:K

much more difficult

One shot learning : input = 1 image

Siamese network

Triplet loss

Anchor – Positive → small

Negative → large

$d(A,p) - d(A,N) + \alpha \leq 0$

$\|f(A) - f(p)\|^2 - \|f(A) - f(N)\|^2$ ↑ margin params

$\mathcal{L} = [\quad , 0]_{max}$

Neutral Style Transfer

content (C) + style (S) → generated picture (G)

cost function

$J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$

content cost function Style cost function

difference in k

activation function channel-channel similarity

$\|a^{(1)} - a^{(2)}\|^2$ $n_c \times n_c$