Course 2: Improving Deep Neural Networks:

Hyperparameter Tuning, Regularization and Optimization

Content:

Week 1: Practical aspects of Deep learning

Week 2: Optimization algorithms

Week 3: Hyperparameter tuning, Batch normalization and programming Frameworks

Coursera

Deep Learning Specialization

Train / Dev / Test sets

① dataset size : 100 ~ 1000000 → 60/20/20

small data

② size : 1000000 ~ ∞ → 98/1/1

large data

or 99.5/0.25/0.25

Bias / variance

overfitting underfitting underfit overfit

error	high var	high bias	high bias, high var	low bias, low var
Train	1%	15%	15%	0.5%
Dev	11%	16%	30%	1%

Best

1) high bias → more training won't help!

get bigger network, more data

assumption: human has 0% error

if high bias:

- Try to make NN bigger
- Try different model
- Train longer
- Try different optimization algorithms.

if high var:

- more data
- Try regularization
- Try different model

* Train bigger network never hurts

Reg for logistic reg.

Regularization → reduce variance (overfitting)

$$J(w,b) = \frac{1}{m}\sum_{i} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2$$

(L2): most common type $\|w\|_2^2 = w^T w$

(L1): sparse! lots zeros $\|w\|_1 = \sum_{j=1}^{n_x} |w_j|$

Reg. for N.N:

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w^{[l]}\|_F^2$$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$$

"Frobenius" norm.
NOT L_2 norm.

why Regularization?

as $\lambda \to$ so, weight matrix $\to 0$.

→ two out a lot of hidden units

→ simplify NN.

→ reduce overfitting.

Dropout regularization → apply both during forward & backward Propagation

use ONLY in training!
↑ if use in test, it would add noise.

→ "drop" some neurons on each iteration based on a probability. e.g. on each iteration, work with a smaller neural network.

* spread out weights

Optimization algorithms

notation: (i) → ith training sample

$[l]$ → lth layer

$\{t\}$ → tth mini batch

Batch gradient descent: run gradient descent on whole data

mini batch gradient descent: run gradient descent on mini data

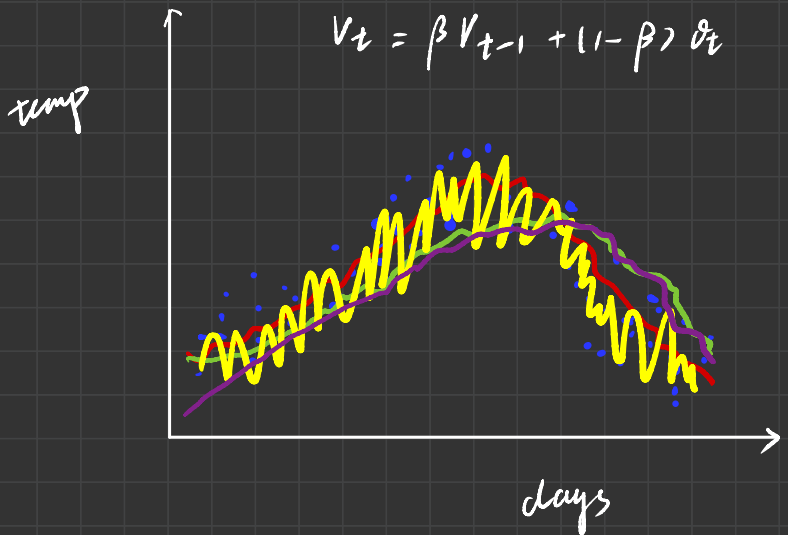
= m → Batch gradient descent (if small training set < 2000) → cost fnc decreases on each iter.

= 1 → stochastic gradient descent → Too long / iter → cost fnc oscillates, can be extremely noisy., won't converge, lose speedup from vectorization.

(use smaller learning rate)

btt 1 & m → mini bach gradient descent → fast, don't always converge.

(has to be power of 2: fits CPU / GPU env.)



note: when $\beta = 0.98$. won't actually have green curve (see purple curve instead). A lower starting.

Bias correction

$v_1 = 0.02\theta_1 \rightarrow 0.02\theta_1 / (1-0.98) = \theta_1$

$v_2 = 0.0196\theta_1 + 0.02\theta_2 \rightarrow (0.0196\theta_1 + 0.02\theta_2)/(1-0.98^2)$

Gradient descent with momentum

on iteration t:

compute dw, db on current mini batch

0.9 often used

$v_{dw} = \beta v_{dw} + (1-\beta)dw$

$v_{db} = \beta v_{db} + (1-\beta)db$

$w = w - \alpha v_{dw}$, $b = b - \alpha v_{db}$

RMSprop

computes the exponentially weighted averages of the squared gradients

Adam optimization algorithm → Adaptive moment Estimation.

putting momentum & RMSprop together ↓ combines the effect of gradient descent with momentum together with gradient descent with RMSprop.

CODE see lab

hyper parameter:	α (learning rate)	tune
	β_1	0.9
	β_2	0.999
	ε	10^{-8}

$\alpha = (1 / (1 + \text{decay\_rate} * \text{epoch\_num}))\,\alpha_0$ ← *

$\alpha = 0.95^{\text{epoch\_num}}\,\alpha_0$

$\alpha = k / \text{epoch\_num}\ \ \alpha_0$

Tuning

1. α.
2. β. mini batch size, # of hidden units
3. # of layers, α decay, Adam β_1, β_2, ε.

Choose points at random.

in practice: Panda vs. Caviar → training models in parallel

↓ not enough computional capacity: baby sit one model

Batch normalization

Softmax regression

TensorFlow

| see lab.