IMPROVING DEEP NEURAL NETWORKS: HYPERPARAMETER TUNING REGULARIZATION AND OPTIMIZATION

Setup Data — Idea → Code → Experiment (cycle); Domain

	Some			
Training	70	60	99	✓
Development / Hold out, Cross Validation	30	20	0.05	✓
Test		20	0.05	✗ Unbiased estimate

Fitting — Over / Under; Decision Boundary; Dimension of Data

Error — Bias — Size of network / neuron, Architecture → Deep Learning
— Variance — Data; Network — Regularization, Architecture

Bias Variance Trade off?

Regularization

Decision boundary → parameters — b - single number, w > loss function → cost function

Frobenium norm: $\frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|^2 = \frac{\lambda}{2m}\sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}}(w_{ij}^{[l]})^2$

Type (b - single no.)
- L_2 norm: $+\frac{\lambda}{2m}\|w\|_2^2$
- L_1 norm: $+\frac{\lambda}{m}|w| = \frac{\lambda}{m}\|w\|_1$ — lots of 0 in model → sparse model, compress model

Implementation
- forward →
- backward ← back prop $+\frac{\lambda}{m}w^{[l]}$
- gradient $w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$

why work?
1) weight decay $\downarrow(1-\alpha\frac{\lambda}{m})$
2) simple model — weights → 0, linear form of equation — decision boundary

→ Network: eliminate (weight 0) node — each training example; computer vision

Implementation: Inverted Dropout
1) Bool (P of keep) / Layer × activation
2) Scale up to match expected value → Test Dataset

1. reduced network — smaller
2. Cannot rely on one feature — spread out weight, ~ adaptive L_2 regularization

Other Technique
- Data augmentation — computer vision ← Data
- Early stopping vs orthogonalization ← Regularization

Input: Normalize — mean, variance; Same scale — circular contour / elongated " — optimize quickly

Vanishing & Exploding gradient — slope, derivative
Deep Network: $w>1$ explode, $w<1$ vanish
barrier — partial solution: Initialization (scale), control $Var(w_i)$

Scaling parameter:
- $Var(w_i) = \frac{1}{n}$
- RELU $= \frac{2}{n}$, $\frac{2}{n^{[l-1]}}$
- Tanh $= \sqrt{\frac{1}{n^{[l-1]}}}$ Xavier Initialization
- $= \frac{2}{n^{[l-1]}+n^{[l]}}$

Gradient Checking
Numerical approximation of gradient — Order of error

$\frac{f(\theta+\epsilon)-f(\theta-\epsilon)}{2\epsilon} \approx g(\theta)$ — $O(\epsilon^2)$

$\frac{f(\theta+\epsilon)-f(\theta)}{\epsilon} \approx g(\theta)$ — $O(\epsilon)$

Check weight, derivative — vector relative diff: $\frac{\|d\theta_{diff} - d\theta\|_2}{\|d\theta_{aprox}\|_2 + \|d\theta_2\|_2}$ — correct 10^{-7}, 10^{-3}

OPTIMIZATION ALGORITHMS

method — Mini batch gradient descent

Gradient update — Entire set — vectorization, slow update / time per iteration
— Mini batch — batch size ← 2^n computer memory & CPU/GPU; Cost — fluctuate, wonder around min

epoch — single pass through entire training set

Exponentially weighted moving average: $V_t = \beta V_{t-1} + (1-\beta)\theta_t$
$= \beta V_{t-1} + \beta(1-\beta)V_{t-2} + \beta(1-\beta)^2 V_{t-3} + \ldots$ coeff → exponentially decay

— average over last $\frac{1}{1-\beta}$ unit; $(1-\epsilon)^{1/\epsilon} = 1/e \approx 0.35$ average over $\frac{1}{\epsilon}$ unit

Implementation — memory efficient
Initial phase → Bias correction $V_t \to \frac{V_t}{1-\beta^t}$

Optimize
- Gradient descent with momentum — Gradient descent → step oscillation — ↓vertical, ↑horizontal → minima
  - $V_{dw} = \beta V_{dw} + (1-\beta)dw$
  - $V_{db} = \beta V_{db} + (1-\beta)db$
  - to update velocity, friction, acceleration
- RMSprop — Root mean square prop
  - $S_{dw} = \beta_2 S_{dw} + (1-\beta_2)dw^2$ — elementwise
  - $S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$
  - $w := w - \alpha\frac{dw}{\sqrt{S_{dw}}+\epsilon}$ — relatively small
  - $b := b - \alpha\frac{db}{\sqrt{S_{db}}+\epsilon}$ — relatively big
- Adam — Adaptive moment estimation = Gradient descent with moment + RMSprop + bias correction

learning rate decay

step size ↓ gradually

$$\alpha = \frac{1}{1 + \text{decay rate} * \text{epoch num}} \times \alpha_0$$

$$\alpha = 0.95^{\text{epoch\_num}} \times \alpha_0 \quad \text{exponential decay}$$

$$\alpha = \frac{k}{\sqrt{\text{epoch num}}} \times \alpha_0$$

Descrete decay function

Manual decay

Local Optima

0 gradient — local minima

saddle point — high dimension — problem of plateaus

Batch Normalization

Additional hyperparameter → robust network

→ speed ↑ : bigger network

Normalization of input in between layers

① μ, $\sigma^2(\epsilon)$ - for each layer (mini batches)

② $z^{(i)}_{\text{normal}} \times \gamma + \beta$

not applicable for b — learning parameter

learning on shifted dist^n / Covariate shift

~ decoupling between layers → robust

add some regularization effect

mini batch - μ, σ^2 - randomness

Test data

→ exponentially weighted average over mini batch

Hyperparameter tuning

parameter

- uniform random number
- appropriate scale - log scale

power → random

Grid vs random

↓

assume one constant

more diverse initialization

coarse to fine

Modeling

Babysitting one model

Training models - parallel

Deep learning framework

Binary

Multiclass } classification

Softmax Activation function $t = e^{(z^{[L]})}$ $a^{[L]} = \frac{e^{z^{[L]}}}{\sum t_i}$

Several linear decision boundary

Hardmax → (max → 1 else 0)

loss function $L(y, \hat{y}) = -\sum_{j=1}^{c} y_j \log \hat{y}_j$

Software framework

1. ease
2. speed
3. truly open