Setting up your ML application

Train/dev/test sets
Applied ML is a highly iterative process

- # layers
- # hidden units
- learning rates
- activation functions
- ...

Idea -> Experiment -> Code

NLP, Vision, Speech, Structural data
Ads, Search, Security, Logistic, ...
Train/dev/test sets

Data: 1,000,000

Train set: 70/30/1
100 - 1,000 - 10,000

Dev set: 10,000

Test set: 10,000

- Hold-out cross validation
- Development set "dev"
Mismatched train/test distribution

Training set: Cat pictures from webpages

Dev/test sets: Cat pictures from users using your app

→ Make sure dev and test come from same distribution.

Not having a test set might be okay. (Only dev set.)
Setting up your ML application

Basic "recipe" for machine learning
Basic recipe for machine learning

- High bias? (training data poor)
  - Yes
    - Bigger network
    - (NN architecture search)
  - No
    - More data
    - (NN architecture search)
- High variance? (test set poor)
  - Yes
    - Regularization
  - No
  - Done

"Bias vs. variance trade-off"
Regularizing your neural network

Why regularization reduces overfitting
How does regularization prevent overfitting?

\[
\mathcal{L}(\omega, \theta) + \frac{\lambda}{2m} \sum_{k=1}^{l} \| \omega_k \|_F^2
\]

\(\omega \approx 0\)

- High bias
- “Just right”
- High variance
How does regularization prevent overfitting?

\[ g(z) = \tanh(z) \]

\[
\mathbf{y}^{[2]} \quad \mathbf{W}^{[2]} \quad \mathbf{z}^{[2]} = \mathbf{W}^{[2]} \mathbf{a}^{[1]} + \mathbf{b}^{[2]}
\]

Every layer is linear.

\[
J(\cdots) = \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 + \frac{\lambda}{2} \sum_{j=0}^{l} || \mathbf{W}^{[j]} ||^2_F
\]
Regularizing your neural network

Other regularization methods
Data augmentation
Early stopping

- Optimize cost function $J$
  - Gradient, ...
  - Not overfit
  - Regularization, ...

$J(w, b)$

dev set error

test error or $J$

$\|w\|_2^2$

mid-size

$w \approx 0$

Andrew Ng
Setting up your ML application

Bias/Variance

deeplearning.ai
Bias and Variance

- **High bias**: Underfitting
- **“Just right”**: Appropriately fit
- **High variance**: Overfitting
Bias and Variance

Cat classification

Train set error: 1%  15% <= 15%  0.5%
Dev set error: 11%  16% <= 30%  1%

Human: 30%
Optimal (Bayes) error: <= 15%
Blurry images

Andrew Ng
High bias and high variance
Setting up your optimization problem

Normalizing inputs
Normalizing training sets

\[ x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \]

**Subtract mean:**

\[ \mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \]

\[ x := x - \mu \]

**Scale variance**

\[ \sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^T (x^{(i)} - \mu) \]

\[ x := \frac{x}{\sigma^2} \]

Use \( \mu, \sigma^2 \) to normalize test set.

Andrew Ng
Why normalize inputs?

\[ J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \]

Unnormalized:

Normalized:
Setting up your optimization problem

Numerical approximation of gradients
Checking your derivative computation

\[ f(\theta) = \theta^3 \]

\[ g(\theta) = \frac{d}{d\theta} f(\theta) = f'(\theta) \]

\[ g(\theta) = 3\theta^2 \]

\[ g(\theta) = 3 - (1)^2 = 2 \quad \text{when} \quad \theta = 1 \]

\[ \frac{f(\theta + \epsilon) - f(\theta)}{\epsilon} \approx g(\theta) \]

\[ \frac{(1.01)^3 - 1^3}{0.01} = \frac{2.0301 - 1}{0.01} \approx 2 \]

\[ \theta = 1 \]
\[ \theta + \epsilon = 1.01 \]
\[ \epsilon = 0.01 \]
Checking your derivative computation

\[ f(\theta) = \theta^3 \]

\[ \frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta) \]

\[ \frac{(1.01)^3 - (0.99)^3}{2(0.01)} = \frac{3.0001}{0.02} = 150.005 \approx 150 \]

\[ g(\theta) = 3\theta^2 = 3 \]

Approx. error: 0.0001

(P Slide: 3.0301, Error: 0.03)

\[ f'(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \]

\[ \frac{0.01}{0.0001} \]

\[ \frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon} \text{ error: } O(\varepsilon^2) \]

\[ \frac{0.01}{0.0001} \]
Regularizing your neural network

Understanding dropout
Why does drop-out work?

Intuition: Can’t rely on any one feature, so have to spread out weights.
Regularizing your neural network

Dropout regularization
Dropout regularization
Implementing dropout ("Inverted dropout")

Illustrate with layer $l=3$. $\text{keep\_prob} = 0.8$ $\Rightarrow$ $0.2$

$\Rightarrow [d3] = \text{np.random.rand}(a3.\text{shape}[0], a3.\text{shape}[1]) < \text{keep\_prob}$

$a3 = \text{np.multiply}(a3, d3)$  # $a3 \times d3$

$\Rightarrow a3 /= \text{np.copy}(\text{keep\_prob})$

50 units $\Rightarrow$ 10 units shut off

$Z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

$\Rightarrow$ reduced by 20%  Test

$/ = 0.8$
Making predictions at test time

\[ a^n = X \]

No drop out.

\[ z^n = w \frac{a^n}{(z^n)^2} + b \]

\[ a^n = g(z^n)(z^n) \]

\[ z^n = w_0 a^n + b \]

\[ a^n = \ldots \]

\[ \downarrow \]

\[ \uparrow y \]

\[ /= \text{keep prob} \]
Vanishing/exploding gradients

Setting up your optimization problem
Vanishing/exploding gradients

\[ l = 150 \]

\[ \hat{y} = \mathbf{W}^{[L]} \mathbf{a}^{[L-1]} \]

\[ \mathbf{w}^{[0]} > I \]
\[ \mathbf{w}^{[0]} < I \begin{bmatrix} 0.9 & 0.9 \end{bmatrix} \]

\[ \mathbf{w} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \]

\[ g(z) = \begin{cases} 2 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \]

\[ a^{[0]} = g(z^{[0]}) = 2 \]

\[ a^{[1]} = g(z^{[1]}) = g(w^{[1]} a^{[0]}) \]

\[ a^{[L]} = g(z^{[L]}) = g(w^{[L]} a^{[L-1]}) \]

\[ \mathbf{W}^{[L]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \]

\[ \mathbf{W}^{[L-1]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \]

\[ \mathbf{W}^{[L-2]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \]

\[ \mathbf{W}^{[L-3]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \]

Andrew Ng
Single neuron example

\[ a = g(z) \]

\[ z = W_1 x_1 + W_2 x_2 + \cdots + W_n x_n \]

large \( n \) \( \rightarrow \) Smaller \( W_i \)

\[ \text{Var}(w_i) = \frac{2}{n} \frac{2}{n} \]

\[ W_{\text{init}} = \text{np}.\text{random.} \cdot \text{random} \cdot \text{softmax} \cdot \text{np}.\text{sqrt} \left( \frac{2}{n-1} \right) \]

\[ \text{ReLU}(x) = \text{ReLU}(x) \]

Other notes:

- Xavier initialization
- \( \frac{2}{\sqrt{n-1}} + v \)
- \( \text{ReLU} \)
Setting up your optimization problem

Gradient Checking
Gradient check for a neural network

$J([w^1, b^1], ..., [w^n, b^n], \theta)$

Is $\theta$ the gradient of $J(\theta)$?
Gradient checking (Grad check)

\[ J(\theta) = J(\theta_0, \theta_1, \ldots) \]

For each \( i \):

\[ \Delta \theta \approx \frac{J(\theta_0, \theta_1, \ldots, \theta_i + \varepsilon, \ldots) - J(\theta_0, \theta_1, \ldots, \theta_i - \varepsilon, \ldots)}{2\varepsilon} \]

\[ \approx \frac{2J}{\partial \theta_i} \]

Check \( \| \Delta \theta \|_2, \| \Delta \theta \|_2 + \| \Delta \theta \|_2 \)

\[ \varepsilon = 10^{-7} \]

\( \varepsilon = 10^{-3} \) - great!

\( \varepsilon = 10^{-5} \) - worry.

Andrew Ng
Regularizing your neural network

Regularization
deeplearning.ai
Logistic regression

\[
\min_{w,b} J(w, b)
\]

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

- \( L(y, \hat{y}) = \log(1 + e^{-y\hat{y}}) \)

\( \lambda \) = regularization parameter

- L2 regularization
  \( \|w\|_2 = \sum_{j=1}^{n_x} w_j^2 = w^T w \)

- L1 regularization
  \[
  \frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m} \|w\|_1
  \]

\( w \) will be sparse

Andrew Ng
Neural network

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

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

"Frobenius norm" \[ \| \cdot \|_2 \quad \| \cdot \|_F \]

\[ dW^{(l)} = (\text{from backprop}) + \frac{\lambda}{m} W^{(l)} \]

\[ \Rightarrow W^{(l)} := W^{(l)} - \alpha dW^{(l)} \]

"Weight decay" \[ W^{(l)} := W^{(l)} - \alpha \left[ (\text{from backprop}) + \frac{\lambda}{m} W^{(l)} \right] \]

\[ = W^{(l)} - \frac{\alpha \lambda}{m} W^{(l)} - \alpha \left( \text{from backprop} \right) \]

\[ = \left( 1 - \frac{\alpha \lambda}{m} \right) W^{(l)} - \alpha \left( \text{from backprop} \right) \]
How does regularization prevent overfitting?

\[ x_1, x_2, x_3 \rightarrow \hat{y} \]
Setting up your optimization problem

Gradient Checking implementation notes
Gradient checking implementation notes

- Don’t use in training – only to debug

- If algorithm fails grad check, look at components to try to identify bug.

- Remember regularization.

- Doesn’t work with dropout.

- Run at random initialization; perhaps again after some training.

\[ d\theta_{\text{optmz.}[i]} \leftarrow \frac{d\theta}{\leftarrow} \]

\[ \frac{db}{\rightarrow} \quad \frac{dw}{\rightarrow} \]

\[ \begin{align*}
\ell(\theta) &= \frac{1}{n} \sum_{i=1}^{n} \ell(y, \hat{y}) + \frac{1}{2m} \sum_{k=1}^{K} \| \omega_{k} \|^2 \\
\delta &= \text{gradient of } \ell \text{ wrt. } \theta
\end{align*} \]

\[ \text{keep-prob} = 1.0 \]