Setting up your ML application

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

\[
\begin{align*}
\text{Idea} & : \\
\text{Experiment} & : \\
\text{Code} & : \\
\end{align*}
\]

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

...
Train/dev/test sets

Data

- Training set

- Dev set
  - Hold-out cross validation
  - Development set "dev"

- Test set

Data

- Small data: 70/30/1
  - 100 - 1500 - 10000

- Big data: 1000,000

- 98/1/1

- 99.5/25/25
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

Bias/Variance
deeplearning.ai
Bias and Variance

- **High Bias**: Underfitting
- **Just Right**: "Just right"
- **High Variance**: Overfitting
Bias and Variance

Cat classification

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

High variance ↑ high bias ↑
High bias & high variance

Human: 80 %
Optimal (Bayes) error: \frac{1}{2} \times 15 %
Blurry images

Andrew Ng
High bias and high variance
Setting up your ML application

Basic “recipe” for machine learning
Basic recipe for machine learning

- High bias? (training data problem) → Bigger network → Turn larger (NN architecture search)
- High variance? (data set problem) → More data → Regularization (NN architecture search)
- Done

Bias vs. Variance tradeoff
Regularizing your neural network

Regularization
deeplearning.ai
Logistic regression

\[
\min_{w,b} J(w, b) \quad w \in \mathbb{R}^n, \quad b \in \mathbb{R}
\]

\[
J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-w^T x_i}) + \frac{\lambda}{2m} \|w\|_2^2
\]

L2 regularization
\[
\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w
\]

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

\(
\lambda = \text{regularization parameter}
\)

\(
\text{lambda}
\)

\(
\text{not}
\)

\[
\frac{1}{m} \sum_{i=1}^{m} y_i w^T x_i + \frac{\lambda}{2m} w^T w
\]

\[
\frac{1}{m} \sum_{i=1}^{m} \left( y_i \log(\frac{1}{1+e^{-w^T x_i}}) + (1 - y_i) \log(1 - \frac{1}{1+e^{-w^T x_i}}) \right)
\]

\[
\frac{1}{m} \sum_{i=1}^{m} y_i w^T x_i + \frac{\lambda}{2m} w^T w
\]

\[
\frac{1}{m} \sum_{i=1}^{m} y_i w^T x_i + \frac{\lambda}{2m} \|w\|_2^2
\]

\[
\frac{1}{m} \sum_{i=1}^{m} y_i w^T x_i + \frac{\lambda}{2m} \|w\|_1
\]

w will be sparse
Neural network

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

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

"Frobenius norm"

\[ \| \cdot \|_2 \quad \| \cdot \|_F \]

\[ dw^{[l]} = \text{from backprop} + \frac{\lambda}{m} w^{[l]} \]

\[ w^{[l]} := w^{[l]} - \alpha dw^{[l]} \]

"Weight decay"

\[ w^{[l]} := w^{[l]} - \alpha \left[ \text{from backprop} + \frac{\lambda}{m} w^{[l]} \right] \]

\[ = w^{[l]} - \frac{\alpha \lambda}{m} w^{[l]} - \alpha \text{ from backprop} \]

\[ = (1 - \frac{\alpha \lambda}{m}) w^{[l]} - \alpha \text{ from backprop} \]

Andrew Ng
Regularizing your neural network

Why regularization reduces overfitting
How does regularization prevent overfitting?

- High bias
- "Just right"
- High variance

\[ \mathcal{L}(\mathbf{w}^{\text{lin}}, b) = \frac{1}{m} \sum_{i=1}^{m} L(y_i, \hat{y}_i) + \frac{\lambda}{2m} \sum_k \|\mathbf{w}^{(k)}\|_F^2 \]

\( \mathbf{w} \approx 0 \)
How does regularization prevent overfitting?
Regularizing your neural network

Dropout regularization
Dropout regularization

\[ \hat{y} \]

\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]

\[ \hat{y} \]

\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]

\[ \uparrow \quad \uparrow \quad \uparrow \]
\[ 0.5 \\ 0.5 \\ 0.5 \]
Implementing dropout (“Inverted dropout”)

Illustrate with layer $l=3$. $\text{keep-prob}=0.8$ $\frac{\text{random}}{x}$ $0.2$

$\rightarrow d_3 = \text{np.random.rand}(a_3.\text{shape}[0], a_3.\text{shape}[1]) < \text{keep-prob}$

$\rightarrow a_3 = \text{np.multiply}(a_3, d_3)$ # $a_3 \times d_3$

$\rightarrow a_3 /= \text{keep-prob}$

50 units $\rightarrow 10$ units shut off

$z^{[4]} = \text{W}^{[4]}a^{[3]} + b^{[4]}$

$J$ \text{ reduced by 20\%} \quad \text{Test}$

$J /= 0.8$
Making predictions at test time

\[ a^n = X \]

No drop out.

\[ z^n = W^{[n]} a^n + b^{[n]} \]

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

\[ z^{[n]} = W^{[n]} a^{[n]} + b^{[n]} \]

\[ a^{[n]} = \ldots \]

\[ y = \text{keep-pool} \]
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

Other regularization methods
Data augmentation

Andrew Ng
Early stopping

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

$J(w,b)$

desired cost error

tuning error or $J$

mid-size $\|w\|_2$

# iterations

$\uparrow w \approx 0$

$\uparrow$ large $w$
Setting up your optimization problem

Normalizing inputs
Normalizing training sets

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

- Subtract mean:
  \[ \mu = \frac{1}{n} \sum_{i=1}^{n} x^{(i)} \]
  \[ x := x - \mu \]
  Use sum \( \mu \) to normalize test set.

- Normalize variance:
  \[ \sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2 \]
  \[ x' = \frac{x}{\sigma} \]
Why normalize inputs?

Unnormalized:

Normalized:

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

Setting up your optimization problem
Vanishing/exploding gradients

\[ y = \mathbf{w}^T \mathbf{a} \]

\[ a_{137} = g(z_{137}) = g(\mathbf{w}_{13}^T \mathbf{a}_{17}) \]

\[ z_{17} = (\mathbf{w}_{17} \mathbf{x}) \]

\[ a_{13} = g(z_{13}) = g(\mathbf{w}_{13}^T \mathbf{a}_{17}) \]

\[ a_{12} = g(z_{12}) = g(\mathbf{w}_{12}^T \mathbf{a}_{17}) \]

\[ \mathbf{w} = \begin{bmatrix} 1.5 & 0 \\ 0 & 0.7 \end{bmatrix} \]

\[ \mathbf{a}_{17} = \begin{bmatrix} 0.5 \\ 0.3 \end{bmatrix} \]

\[ 1.5 \]

\[ 0.5 \]

Andrew Ng
Single neuron example

\[ a = g(z) \]

\[ z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \]

- Large \( n \) \( \Rightarrow \) Smaller \( w_i \)

\[ \text{Var}(w_i) = \frac{1}{n} \frac{2}{n} \]

\[ W = \text{np} \cdot \text{random} \cdot \text{randn} \left( \text{shape} \right) \times \text{np}\text{sqrt} \left( \frac{2}{[1 \ldots 1]} \right) \]

\[ \text{ReLU} \quad g^{\text{ReLU}}(z) = \text{ReLU}(z) \]
Setting up your optimization problem

Numerical approximation of gradients
Checking your derivative computation

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

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

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

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

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

\[ \frac{(1.01)^3 - 1^3}{0.01} = \frac{3.0301}{0.01} \approx 2 \]

\[ \theta = 1 \]

\[ \theta + \epsilon = 1.01 \]

\[ g(\theta) = 3 \]

Andrew Ng
Checking your derivative computation

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

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

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

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

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

Approx. error: 0.0001

(pre slide: 3.0301, error: 0.03)
Setting up your optimization problem

Gradient Checking
Gradient check for a neural network

Take \( W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]} \) and reshape into a big vector \( \theta \).

\[
J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = J(\theta)
\]

Take \( dW^{[1]}, db^{[1]}, \ldots, dW^{[L]}, db^{[L]} \) and reshape into a big vector \( d\theta \).

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

For each $i$:

$$
\frac{d\theta_{approx}[i]}{\theta} = \frac{J(\theta_1, \theta_2, ..., \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, ...) - J(\theta_1, \theta_2, ..., \theta_{i-1}, \theta_i - \varepsilon, \theta_{i+1}, ...)}{2\varepsilon}
$$

$$
\approx \frac{d\theta[i]}{\theta} = \frac{2J}{\partial \theta_i}
$$

Check

$$
\frac{||d\theta_{approx} - d\theta||_2}{||d\theta_{approx}||_2 + ||d\theta||_2} \approx 10^{-7}
$$

$\varepsilon = 10^{-7}$
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.