Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode
Setting up your ML application

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

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

NLP, Vision, Speech, Structural data

Ads / Search / Security / Logistic...
Train/dev/test sets

Data

Train set

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

Test set

Pron smaller: 70/30/1
100 - 1000 - 10000

Big data: 1,000,000

10,000

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
Bias and Variance

high bias

just right

high variance
Bias and Variance

Cat classification

Train set error: 1%  
Dev set error: 11%  

Human: 80%

Optimal (Bayes) error: 2% to 15%

y=1

y=0

High bias & high variance

Low bias & low variance

0.5%
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)
  - Yes
    - Bigger network
      - Too large
        - (NN architecture search)
  - No
    - High variance? (data set problem)
      - Yes
        - More data
          - (NN architecture search)
      - No
        - Done

Bias \rightarrow Variance \rightarrow Tradeoff
Regularizing your neural network

Regularization
Logistic regression

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

\(L_2\) regularization

\(\|w\|^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w \leq \lambda\)

\(L_1\) regularization

\(\frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m} \|w\|_1 \leq \lambda\)
Neural network

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

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

"Frobenius norm"

\[
\frac{\partial J}{\partial w^{(l)}} = \text{(from backprop)} + \frac{\lambda}{m} w^{(l+1)}
\]

\[
\Rightarrow w^{(l+1)} := w^{(l+1)} - \alpha \frac{\partial J}{\partial w^{(l)}},
\]

"Weight decay"

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

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

\[
= \left( 1 - \frac{\alpha \lambda}{m} \right) w^{(l+1)} - \alpha \text{ (from backprop)}
\]
Regularizing your neural network

Why regularization reduces overfitting
How does regularization prevent overfitting?

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

\[ \mathbf{w}^m \approx \mathbf{0} \]

- **High bias**
- **“Just right”**
- **High variance**
How does regularization prevent overfitting?
Regularizing your neural network

Dropout regularization
Dropout regularization

\[ \hat{y} \]

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

\[ \hat{y} \]

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

\[ \uparrow \uparrow \uparrow \]
\[ 0.5 \quad 0.5 \quad 0.5 \]
Implementing dropout ("Inverted dropout")

Illustrate with layer $l=3$, keep-prob = 0.8

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

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

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

50 units $\Rightarrow$ 10 units shut off

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

$J$ reduced by 20%.  Test

$\Rightarrow = 0.8$
Making predictions at test time

\[ a^\text{test} = X \]

No drop out.

\[ z^\text{test} = W^T a^\text{test} + b^\text{test} \]
\[ a^\text{test} = g(z^\text{test}) \]
\[ e^\text{test} = W^T a^\text{test} + b^\text{test} \]
\[ a^\text{out} = \ldots \]

\( /= \text{keep-prob} \)
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
Early stopping

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

$J(w, b)$

$\|w\|_2$ (mid-size)

$\|w\|_2^2$ (large $w$)

# iterations

dev. set error

test error or $J$
Setting up your optimization problem

Normalizing inputs
deeplearning.ai
Normalizing training sets

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

Subtract mean:

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

\[ x := x - \mu \]

Normalize variance:

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

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

Use sum $\mu \sigma^2$ to normalize test set.

Andrew Ng
Why normalize inputs?

Unnormalized:

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

Normalized:
Vanishing/exploding gradients

Setting up your optimization problem
Vanishing/exploding gradients

$y = W [1] \circ \ldots \circ W [l - 1] x$

$g(z) = \frac{1}{1 + e^{-z}}$

$b^{l - 1} = 0$

$\hat{y} = W [l] \circ \ldots \circ W [1] x$

$w^{100} > I$

$w^{100} < I [0.9, 0.9]$

$W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Andrew Ng
**Single neuron example**

\[
\begin{align*}
\hat{y} &= a = g(z) \\
    z &= w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \\
    \text{large } n &\Rightarrow \text{ smaller } w_i \\
    \text{Var}(w_i) &= \frac{1}{n} \left[ \frac{2}{n} \right] \\
    W &= \text{np.random.randn}(\text{shape}) \times \text{np.sqrt}(\frac{2}{n}) \\
    \text{ReLUs} &\quad g^{\text{ReLU}}(z) = \text{ReLU}(z)
\end{align*}
\]
Setting up your optimization problem

Numerical approximation of gradients

deeplearning.ai
Checking your derivative computation

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

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

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

When \( \theta = 1 \)

\[ g(\theta) = 3 - (1)^2 = 2 \]

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

\[ (1.01)^3 - 1^3 = 3.0301 \approx 3 \]

\[ \theta = 1 \]

\[ \theta + \varepsilon = 1.01 \]

\[ \varepsilon = 0.01 \]
Checking your derivative computation

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

\[
\frac{f(1.01) - f(0.99)}{2 \times 0.01} \approx 3 \]

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

approx error: 0.0001

(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 $i$:

$\nabla \theta[i] = \frac{J(\theta_1, \theta_2, \ldots, \theta_i + \varepsilon, \ldots) - J(\theta_1, \theta_2, \ldots, \theta_i - \varepsilon, \ldots)}{2 \varepsilon}$

$\nabla \theta[i] \approx \frac{2J}{2\theta[i]}$

$\nabla \theta_{\text{approx}} \approx \nabla \theta$

Check:

$\frac{\|\nabla \theta_{\text{approx}} - \nabla \theta\|_2}{\|\nabla \theta\|_2 + \|\nabla \theta_{\text{approx}}\|_2}$

$\varepsilon = 10^{-7}$

$10^{-7}$ - great!

$10^{-5}$

$10^{-3}$ - worry.
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.