Deep Learning

Chapter 2 Building Neural Network from Scratch

Dr. Van-Toi NGUYEN
EEE, Phenikaa University

1

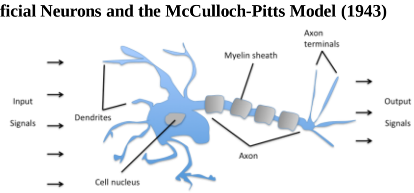
Chapter 2: Building Neural Network from Scratch

1. Shallow neural network
2. Deep neural network
3. Building neural network: step-by-step (modulation)
4. Regularization
5. Dropout
6. Batch Normalization
7. Optimizers
8. Hyper-parameters
9. Practice

2

Previous Lecture Overview

Artificial Neurons and the McCulloch-Pitts Model (1943)



Schematic of a biological neuron.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

3

Previous Lecture Overview

Vectorizing across multiple examples

```
for i = 1 to m:
```

$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$

$a^{[1](i)} = \sigma(z^{[1](i)})$

$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$

$a^{[2](i)} = \sigma(z^{[2](i)})$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$Z^{[1]} = W^{[1]}X + b^{[1]}$

$A^{[1]} = \sigma(Z^{[1]})$

$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$

$A^{[2]} = \sigma(Z^{[2]})$

Previous Lecture Overview

Dimensions of vectorized implementations

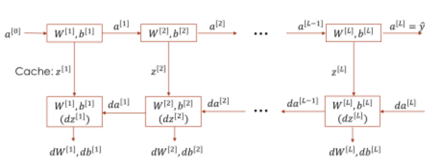
- For one single training example:
- $z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$
- $(n^{[l]}, 1) = (n^{[l]}, n^{[l-1]}) \times (n^{[l-1]}, 1) + (n^{[l]}, 1)$

- For a vectorized implementation over m examples
- $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$
- $(n^{[l]}, m) = (n^{[l]}, n^{[l-1]}) \times (n^{[l-1]}, m) + (n^{[l]}, m)$

Matrix	Dimensions
$Z^{[l]}, A^{[l]}, b^{[l]}, dZ^{[l]}, dA^{[l]}, db^{[l]}$	$(n^{[l]}, m)$
$W^{[l]}, dW^{[l]}$	$(n^{[l]}, n^{[l-1]})$

Previous Lecture Overview

Forward and backward functions



$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$

$b^{[l]} = b^{[l]} - \alpha db^{[l]}$

Previous Lecture Overview

Parameters vs. Hyperparameters

- Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \ldots$
- Hyperparameters
  - Learning rate: α
  - Number of iterations
  - Number of hidden layers or L
  - Number of hidden units in each layer: $n^{[1]}, n^{[2]}, \ldots$
  - Choice of activation function: sigmoid, ReLU, tanh, etc.
  - Momentum, mini-batch size, regularization parameters, … (in the next Chapter)

Previous Lecture Overview

Summary of Forward/Backward Computations

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}(Z^{[L]}) = \hat{Y}
\end{aligned}
$$

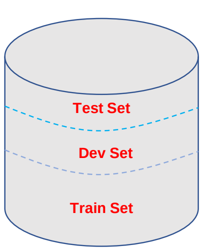
Forward Propagation

$$
\begin{aligned}
dZ^{[L]} &= A^{[L]} - Y \\
dW^{[L]} &= \frac{1}{m} dZ^{[L]} A^{[L]^T} \\
db^{[L]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[L]}, axis = 1, keepdims = True) \\
dZ^{[L-1]} &= dW^{[L]^T} dZ^{[L]} g'^{[L]}(Z^{[L-1]}) \\
&\vdots \\
dZ^{[1]} &= dW^{[L]^T} dZ^{[2]} g'^{[1]}(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} A^{[1]^T} \\
db^{[1]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[1]}, axis = 1, keepdims = True)
\end{aligned}
$$

Backward Propagation

x. Some concept

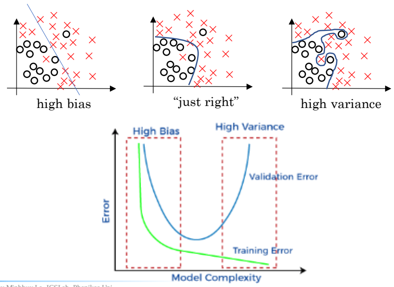
Training vs. Development vs. Test sets



- Traditionally best practice:
  - Train : Test = **70:30**
  - Train : Dev : Test = **60:20:20**
- Modern big data era:
  - Total dataset size: 1,000,000
  - Dev set: big enough to evaluate different algorithm choices, say 10,000 more than enough
  - Test set: big enough to test accuracy, say 10,000 more than enough
  - Thus, train : dev : test = **98 : 1 : 1**
  - Or even, **99.5 : 0.4 : 0.1**

x. Some concept

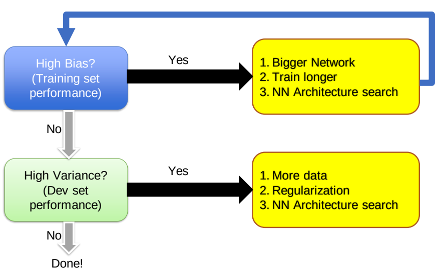
Bias vs. Variance



10

x. Some concept

Bias vs. Variance



11

4. Regularization

Overfitting

- Can be solved using Regularization or More Data
- Sometimes it is difficult to get more data, so regularization could be a good

Logistic Regression

- Cost function: $\min_{w,b} J(w,b)$

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

- L2 regularization: $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$
- L1 regularization: $\|w\|_1 = \sum_{j=1}^{n_x} |w_j|$
- λ is the **regularization parameter**

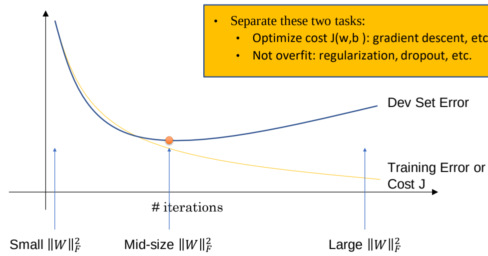
12

4. Regularization

- $J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|_F^2$
- **Frobenius** Norm: $\|w^{[l]}\|_F^2 = \sum_{i=1}^{n[l-1]}\sum_{j=1}^{n[l]}(w_{ij}^{[l]})^2$
- Back-propagation: $\frac{\partial J}{\partial W^{[l]}} = dW^{[l]} = \left(\frac{1}{m} dZ^{[l]} A^{[l]^T}\right) + \frac{\lambda}{m} W^{[l]}$
- Weight updates: $W^{[l]} = W^{[l]} - \alpha dW^{[l]}$
- L2 normalization is also called "weight decay" because
  - $W^{[l]} = W^{[l]} - \alpha\left[\left(\frac{1}{m} dZ^{[l]} A^{[l]^T}\right) + \frac{\lambda}{m} W^{[l]}\right]$
  - $= W^{[l]} - \frac{\alpha\lambda}{m} W^{[l]} - \alpha\left(\frac{1}{m} dZ^{[l]} A^{[l]^T}\right)$
  - $= \left(1 - \frac{\alpha\lambda}{m}\right) W^{[l]} - \alpha\left(\frac{1}{m} dZ^{[l]} A^{[l]^T}\right)$

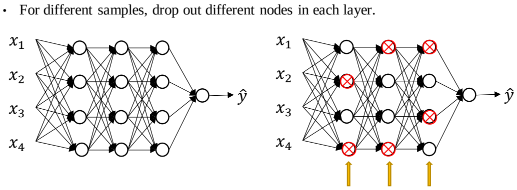
4. Regularization (Data Augmentation)

- Separate these two tasks:
  - Optimize cost J(w,b): gradient descent, etc.
  - Not overfit: regularization, dropout, etc.



5. Dropout

- Suppose dropout rate is 0.5, drop out 0.5 nodes in each layer for each sample
- For different samples, drop out different nodes in each layer.

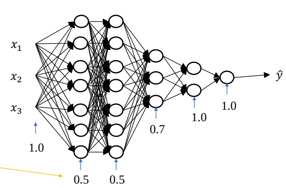


5. Dropout

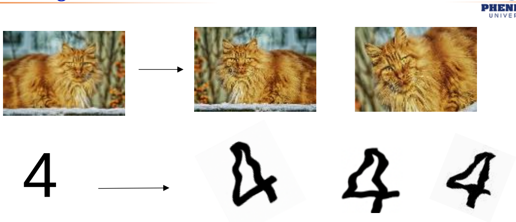
- Suppose dropout is applied to layer 3
- keep_prob = 0.8 (probability a node will be kept)
- $d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep\_prob$
  - A vector to decide which nodes to dropout
- $a3 = np.multiply(a3, d3)$
- $a3 /= keep\_prob$
  - Pump up the activation values by keep_prob to maintain the expected values
- $z^{[4]} = w^{[4]}.a^{[3]} + b^{[4]}$
- Example: 100 units ➔ 20 units shut off
- Dropout different hidden units in different iterations

5. Dropout

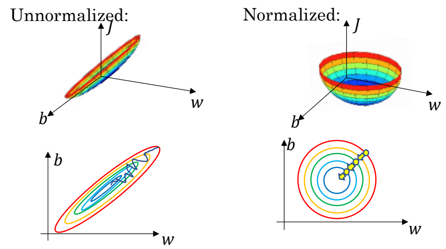
- No dropout during test time
  - Would add noise during predictions is dropout is used during test time
- Why dropout works?
  - Regularizes the network
  - Reduces the dependence on some particular feature (input node)
    - Dropout spreads out the weights
- Can use different dropout keep_probs for different layers
- Cost function not well-defined because of the weights randomly changed



x. Data Augmentation



6. Batch Normalization



19

6. Batch Normalization

- Normalizing inputs

$$x_{j,std} = \frac{x_j - \mu_j}{\sigma_j}$$

where μ_j is the sample mean of the feature x_j and σ_j the standard deviation.

- After normalization, the inputs will have unit variance and centered around mean zero.

20

6. Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

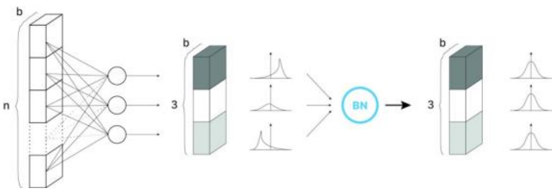
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

21

6. Batch Normalization

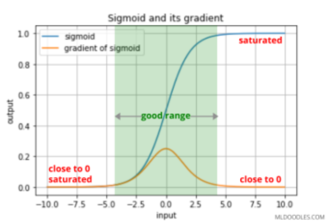


22

6. Batch Normalization

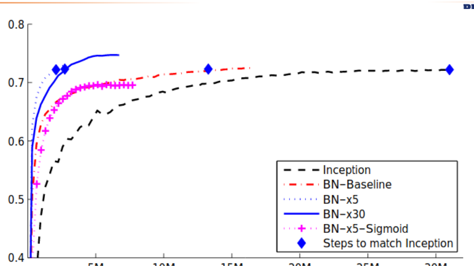
Gradient Vanishing

By applying batch normalization, we can make sure the input stays in the steep portion, also called as the good range. When the input stays in the good range, the derivative is also bigger and does not vanish.



23

6. Batch Normalization



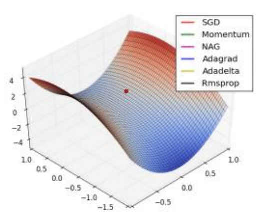
24

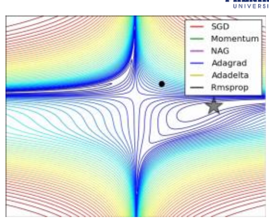
7. Optimizers

https://ml-cheatsheet.readthedocs.io/en/latest/optimizers.html

Students Read & Discussion

7. Optimizers

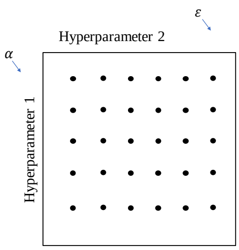




8. Hyper-parameters

- Learning rate: α
- Momentum: β
- RMPprop: $\beta_2 = 0.999$ (usually not tuned)
- Adam: $\beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$ (usually not tuned)
- #layers
- #hidden units
- Learning rate decay
- Mini-batch size

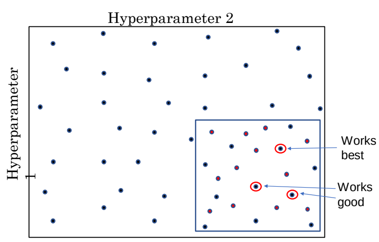
8. Hyper-parameters



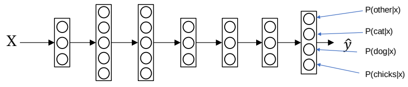
- Some parameters are more important than the others
- Take for example: α, ε
- α is more important than ε
- Grid search would result in searching through only 5 different important values (α)
- Thus, a **random search** would be better!

8. Hyper-parameters

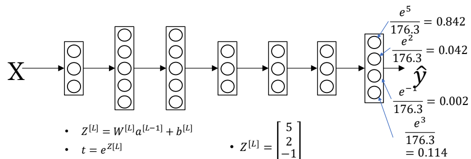
Coarse to fine



x. Softmax



x. Softmax



- $Z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]}$
- $t = e^{Z[L]}$
- $a^{[L]} = \frac{e^{Z^{[L]}}}{\sum_{i=1}^{n_L} t_i}$

- $Z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$
- $t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$ $\quad \sum_{j=1}^{4} t_j = 176.3$
- $a^{[L]} = \frac{t}{176.3}$

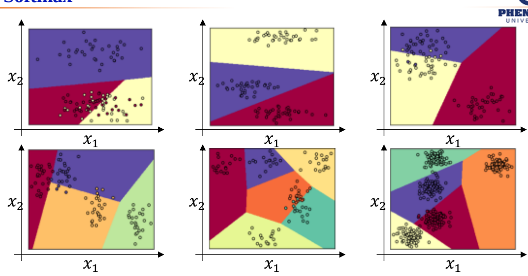
31

x. Softmax

- $Z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$ $\quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$
- $a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5 + e^2 + e^{-1} + e^3) \\ e^2/(e^5 + e^2 + e^{-1} + e^3) \\ e^{-1}/(e^5 + e^2 + e^{-1} + e^3) \\ e^3/(e^5 + e^2 + e^{-1} + e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$
- Softmax because each class has a probability, whereas hardmax would give 1 to the class with highest probability and 0 to the rest $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$.
- If C = 2, softmax reduces to logistic regression.

32

x. Softmax



33

Conclusion

- Has review some commons techniques in neural network
- Prevent overfitting using Regularization, Dropout
- Fast training, higher accuracy, prevent gradient vanishing using Batch Norm
- Optimizers improve accuracy (toward global minimum)
- Multiclassification using Softmax
- Hyper-parameters turning takes time and generally hard to apply in practice. Usually choose common params from published papers (experiences)