INSTITUTO
METRÓPOLE
DIGITAL

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

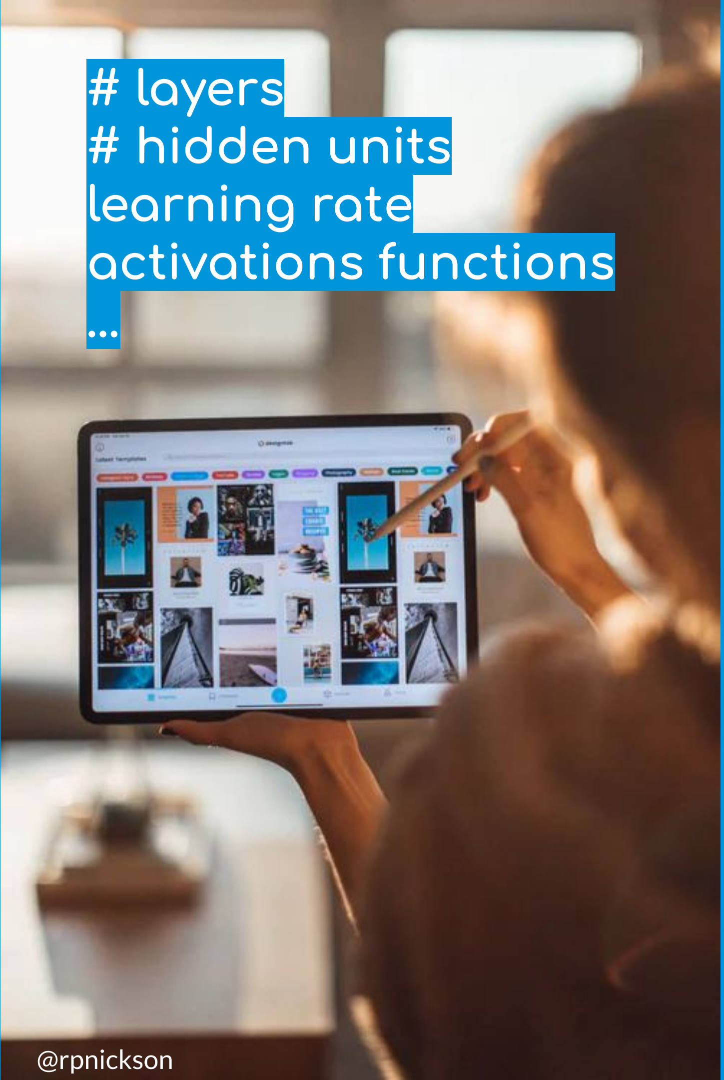
IMD1122
Special Topics in AI
Deep Learning

Networks in Practice

Lesson #05

Neural Networks in Practice #01

Splitting Data & Regularization



layers
hidden units
learning rate
activations functions
…

Applied NN is a highly iterative process

Idea
↓
Code
↓
Experiment

Train - Dev - Test Sets

Making good choices in how you set up your training, development, and test sets can make a huge difference in helping you quickly find a good high performance neural network.

Data	Train Set	Dev Set	Test Set

Holdout
Cross-Validation
Validation
Development

Previous ML era

- 70/30
- 60/20/20

Big Data era

- 98/1/1
- 99.5/0.25/0.25
- 99.5/0.4/0.1

Mismatched train/test distribution

Scenario: say you are building a cat-image classifier application that determines if an image is of a cat or not. The application is intended for users in rural areas who can take pictures of animals by their mobile devices for the application to classify the animals for them.

Scraped from Web Pages
100k images

Collected from Mobile Devices
<<target distribution>>
8k images

A possible option: shuffling the data

100k images
(from web)

8k images
(from target distribution)

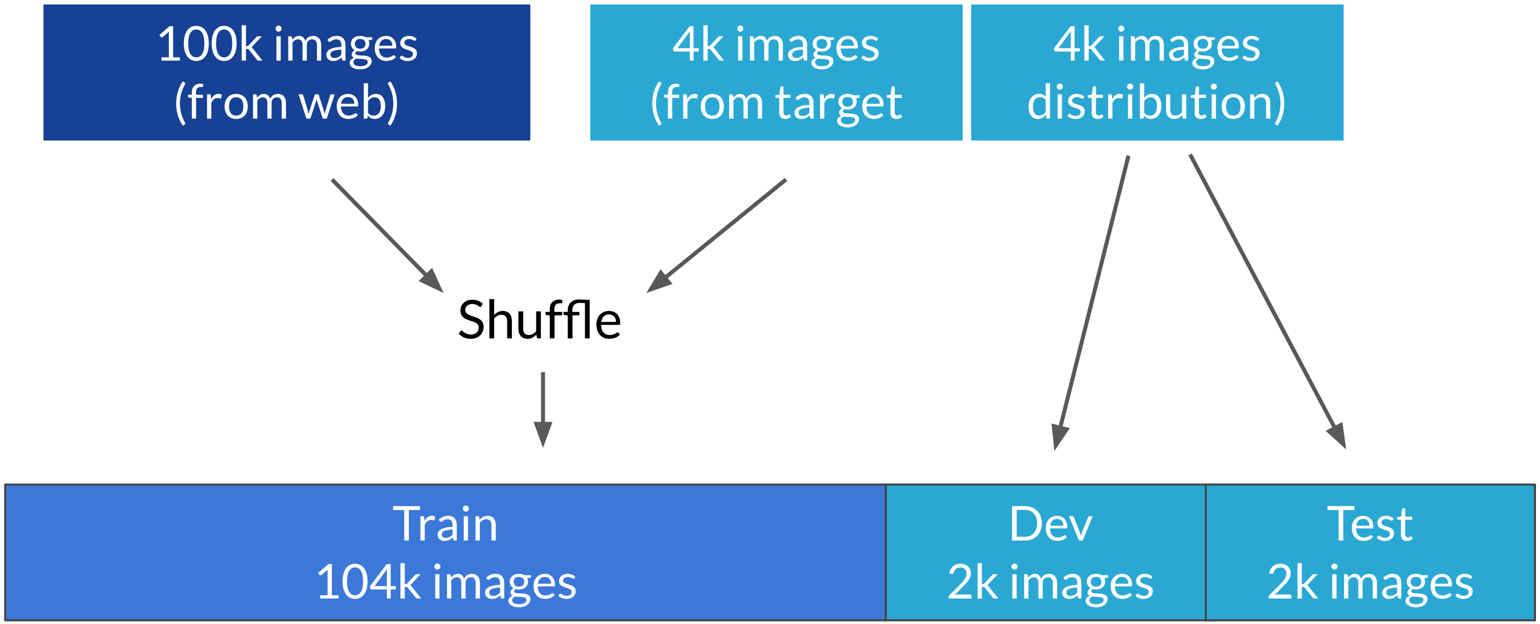
Shuffle

Train 104k images	Dev 2k images	Test 2k images

There a big drawback here!!

only 148 images come from the target distribution

A better option

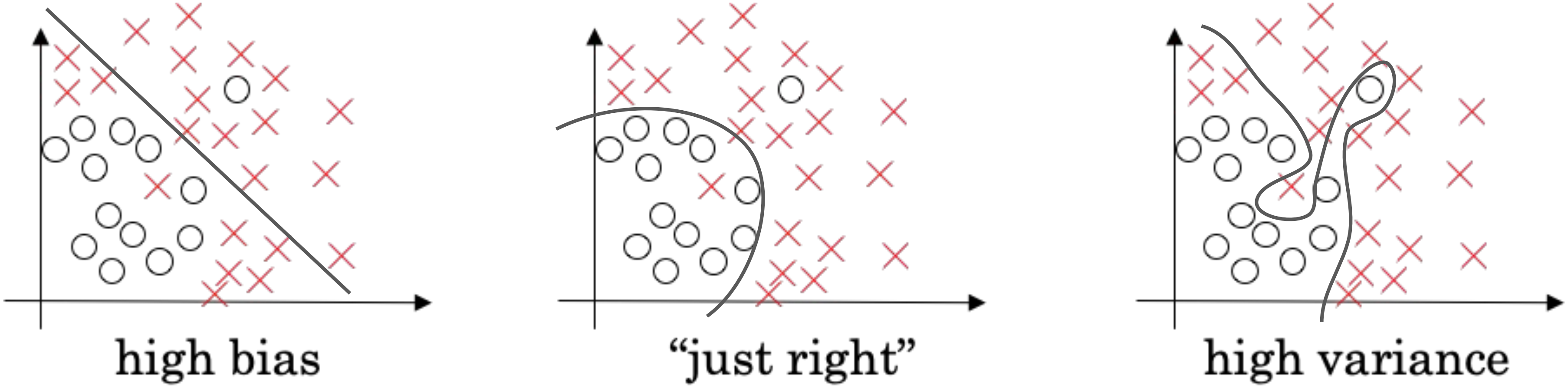


Rule of the thumb

>> make sure that the dev and test sets come from the same distribution

Not having a test set might be okay. (Only dev set)

Bias vs Variance



Underfitting

Overfitting

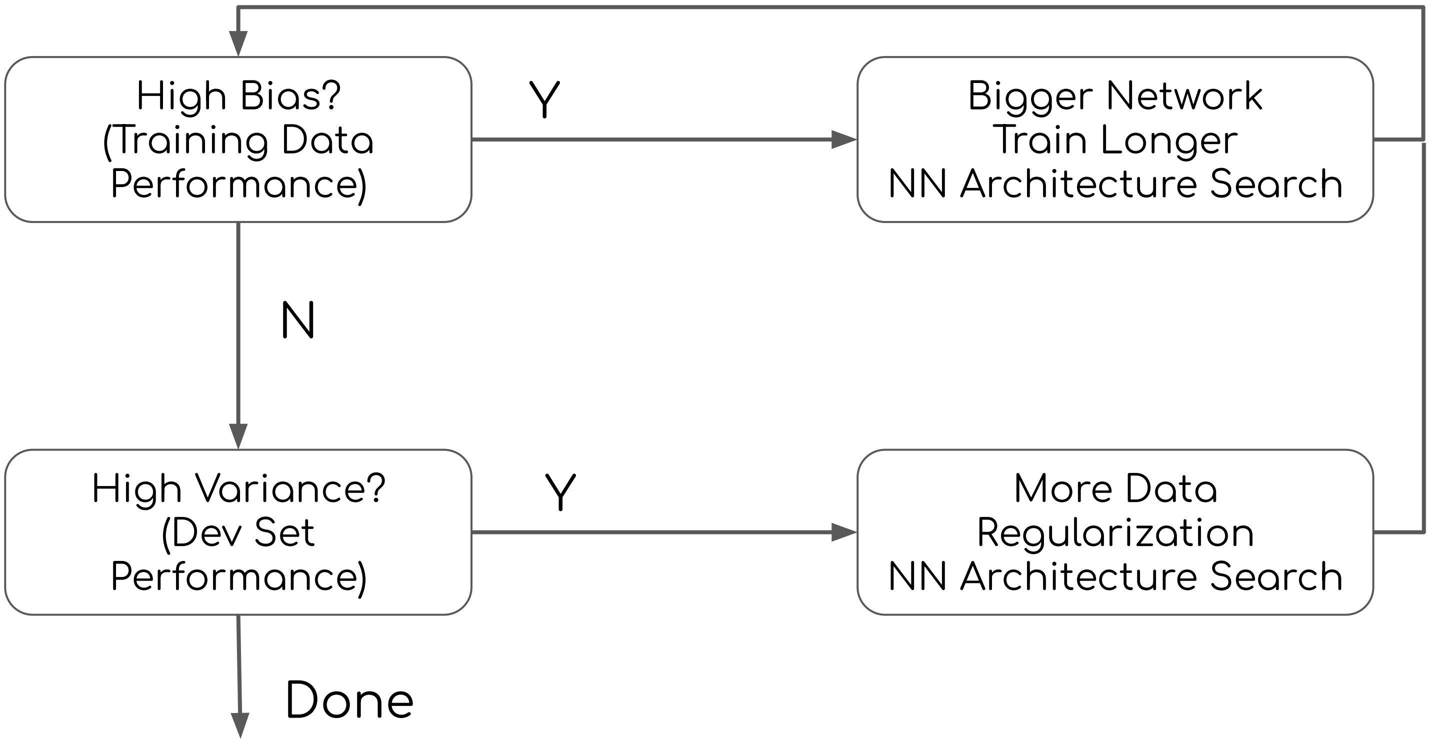
Bias vs Variance

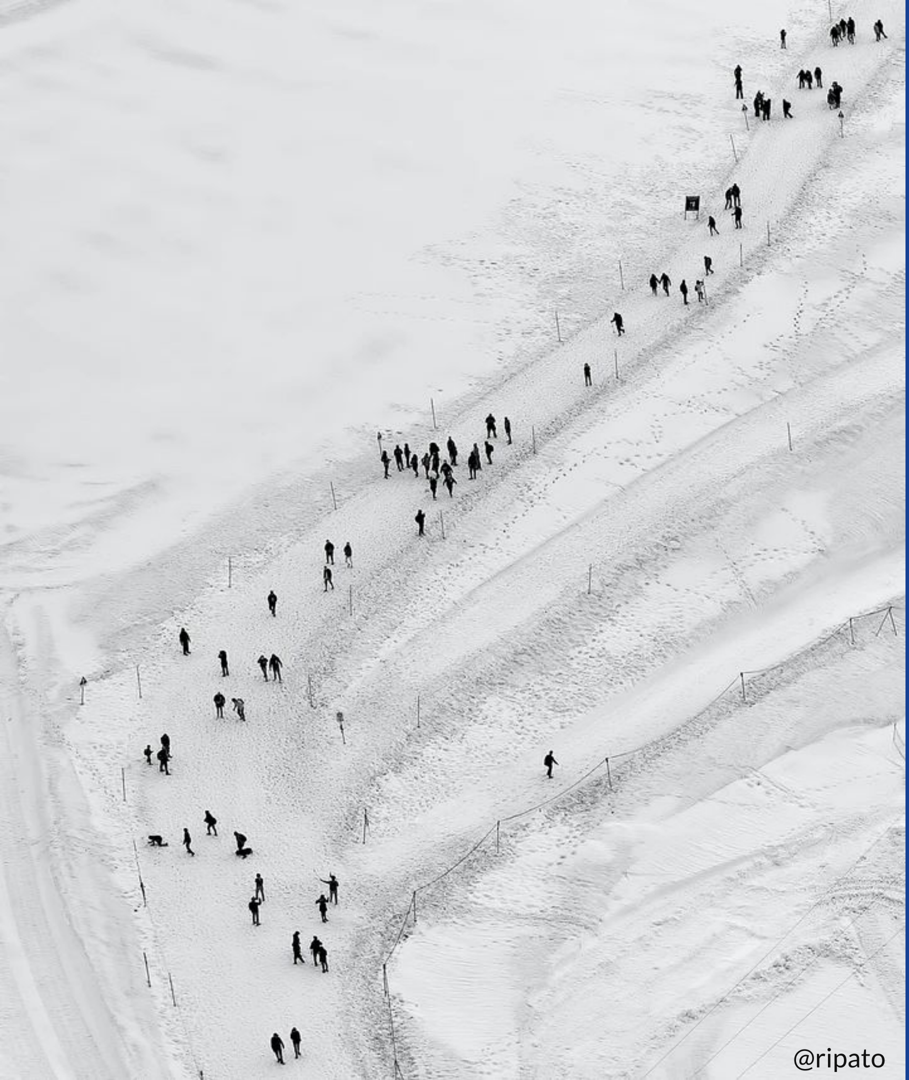
Cat Classification

	Scenario #01	Scenario #02	Scenario #03	Scenario #04
Train Set Error	1%	15%	15%	0.5%
Dev Set Error	16%	16%	30%	1%

	Scenario #01	Scenario #02	Scenario #03	Scenario #04
	Low Bias High Variance	High Bias Low Variance	High Bias High Variance	Low Bias Low Variance

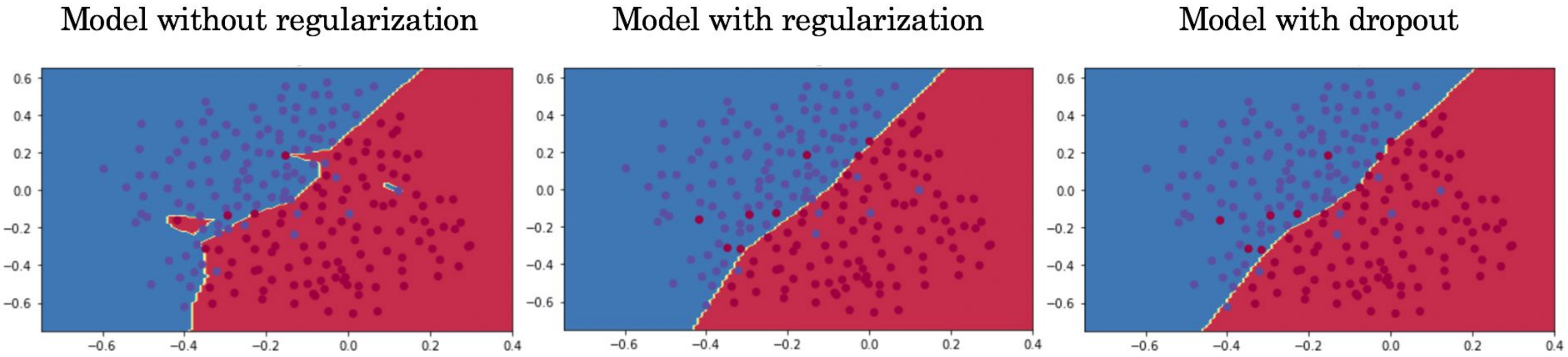
Basic Recipe for Machine Learning





Regularizing your Neural Network

What if we penalize complexity?



It is very important that you regularize your model properly because it could dramatically improve your results

L2 Regularization (Frobenius Norm)

$$J = -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log\left(a^{[L](i)}\right) + (1-y^{(i)}) \log\left(1-a^{[L](i)}\right)\right)$$

$$J_{regularized} = \underbrace{-\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log\left(a^{[L](i)}\right) + (1-y^{(i)}) \log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m} \frac{\lambda}{2} \sum_l \sum_k \sum_j W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

L2 Regularization (Frobenius Norm)

Impact on Gradient Descent

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_l\sum_k\sum_j W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

$$\frac{\partial J}{\partial W^{[l]}} = dW^{[l]} = \{from\ backprop.\} + \frac{\lambda}{m}W^{[l]}$$

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

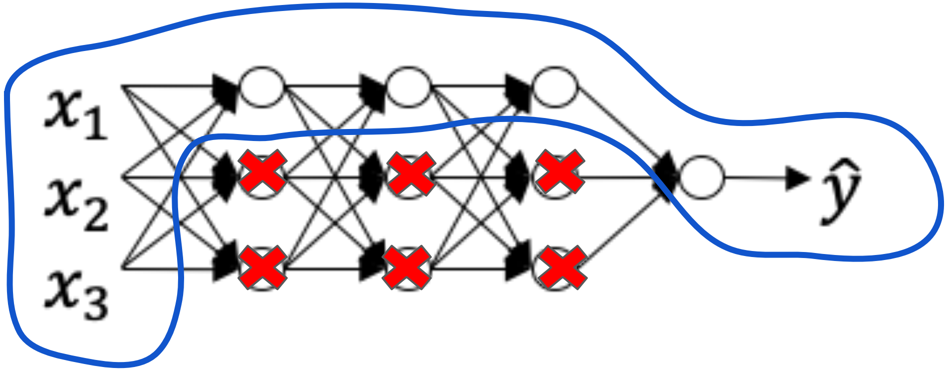
$$W^{[l]} = W^{[l]} - \alpha\left[\{from\ backprop.\} + \frac{\lambda}{m}W^{[l]}\right]$$

"Weight Decay"

$$W^{[l]} = (1 - \frac{\alpha\lambda}{m})W^{[l]} - \alpha\left[\{from\ backprop.\}\right]$$

How does regularization prevent overfitting?

Intuition #01

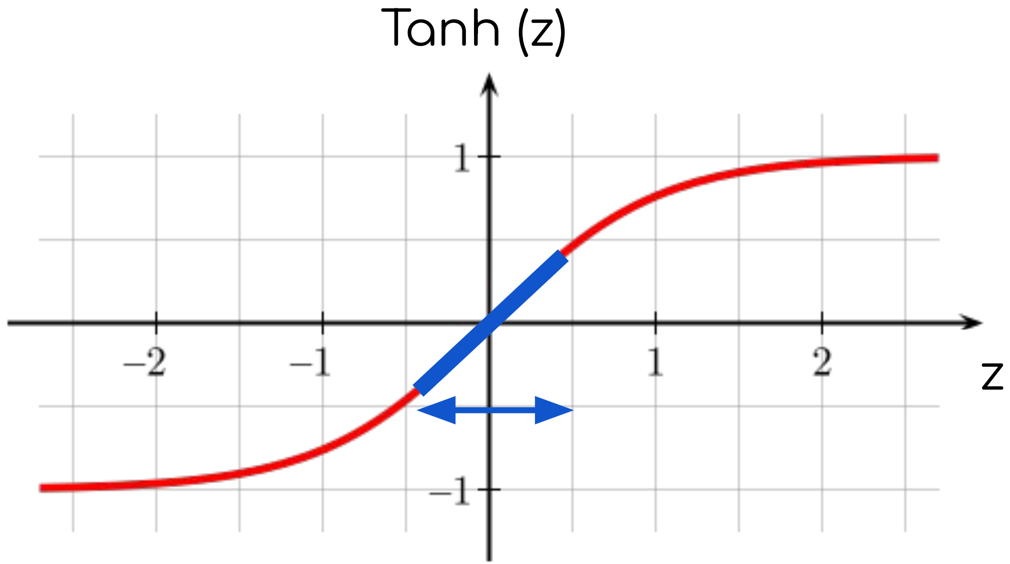


$$W^{[l]} = \left(1 - \frac{\alpha\lambda}{m}\right)W^{[l]} - \alpha\,[\{from\ backprop.\}$$

$$\lambda \uparrow \quad \Rightarrow \quad W^{[l]} \approx 0$$

How does regularization prevent overfitting?

Intuition #02



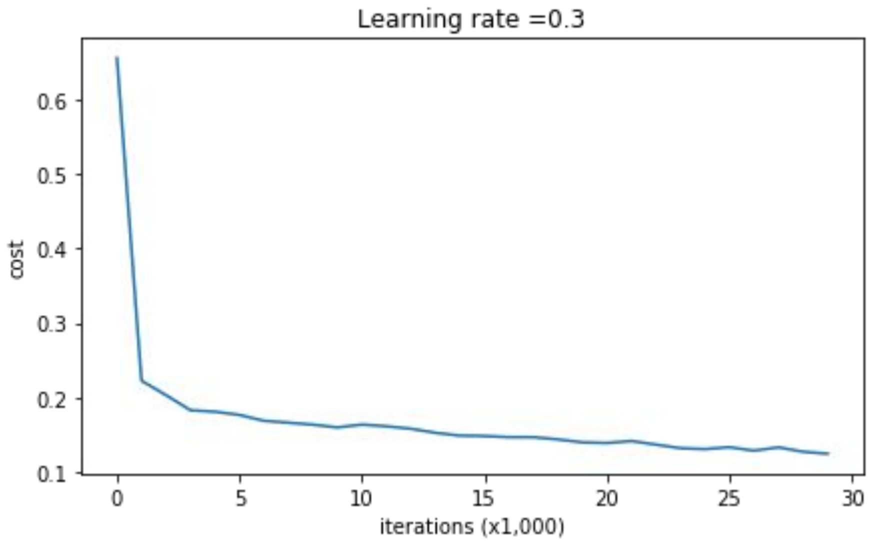
$$\lambda \uparrow \quad \Rightarrow \quad W^{[l]} \approx 0$$

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

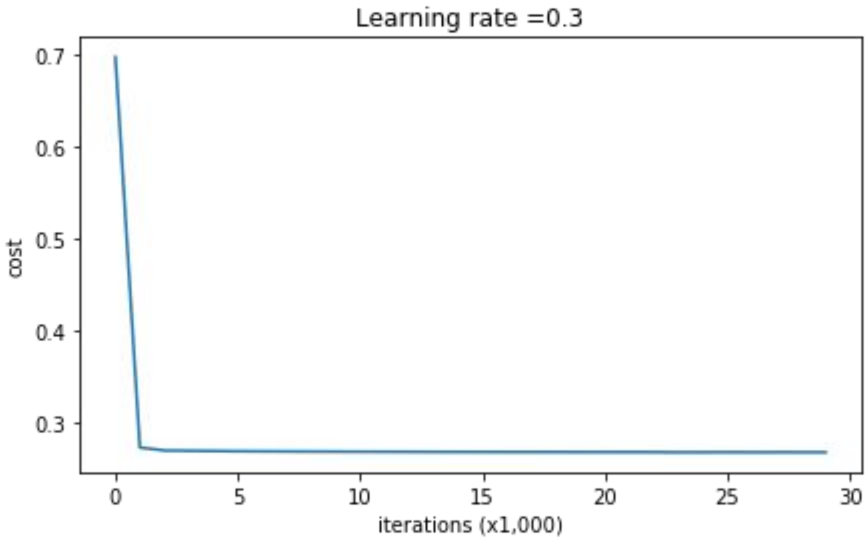
How does regularization prevent overfitting?

Intuition #03

Without Regularization



With Regularization

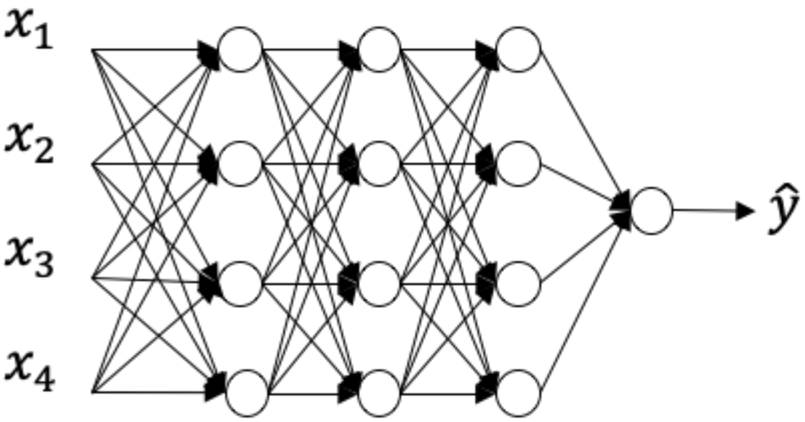


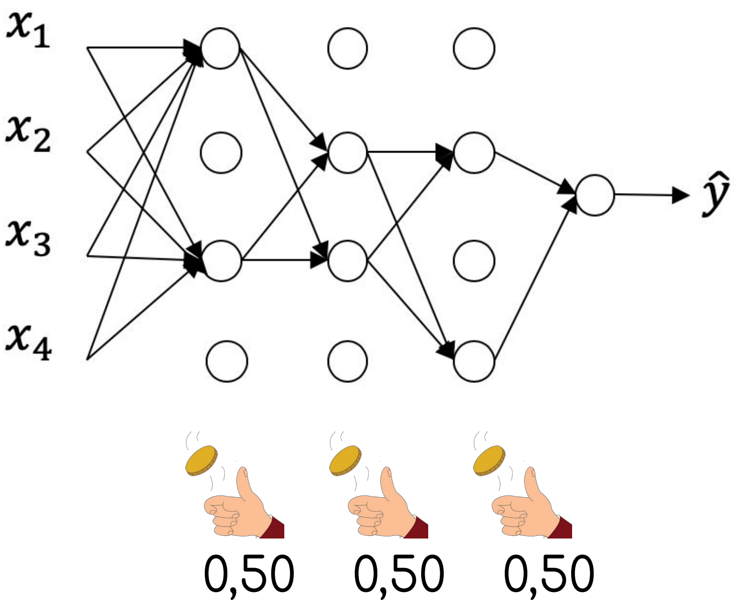
Putting it all together

```
model = tf.keras.Sequential([
                             tf.keras.layers.Dense(20, activation=tf.nn.relu),
                             tf.keras.layers.Dense(3,activation=tf.nn.relu,
                                                   kernel_regularizer=tf.keras.regularizers.l2(l=0.01)),
                             tf.keras.layers.Dense(1, activation = tf.nn.sigmoid)
                             ])
```

Dropout Regularization

You implement Dropout regularization only while training the network. You do not apply it while running your testing data through it as you do not want any randomness in your predictions





Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

Implementing Dropout ("Inverted Dropout")

Illustrate with layer "l" = 3

$$\begin{cases} keep\_prob & = 0.8 \\ 1 - keep\_prob & = 0.2 \end{cases}$$

$$d3 = np.randon.rand(a3.shape[0], a3.shape[1]) < keep\_prob$$

$$a3 = np.multiply(a3, d3)$$

$$a3 / = keep\_prob$$

$$Z^{[4]} = \alpha^{[3]} W^{[4]} + b^{[4]}$$

It is necessary not to impact the value of Z.

Applying dropout for a input

```
import tensorflow as tf
import numpy as np

tf.random.set_seed(0)
layer = tf.keras.layers.Dropout(.2, input_shape=(2,))
data = np.arange(10).reshape(5, 2).astype(np.float32)
print(data)

[[0. 1.]
 [2. 3.]
 [4. 5.]
 [6. 7.]
 [8. 9.]]
```

keep_prob=0.8

```
outputs = layer(data, training=True)
print(outputs)

tf.Tensor(
[[ 0.    1.25]
 [ 2.5   3.75]
 [ 5.    6.25]
 [ 7.5   8.75]
 [10.    0.  ]], shape=(5, 2), dtype=float32)
```

output = output /keep_prob

Putting it all together

```
model_dropout = tf.keras.Sequential([
                             tf.keras.layers.Dense(20, activation=tf.nn.relu),
                             tf.keras.layers.Dropout(0.3),
                             tf.keras.layers.Dense(3, activation=tf.nn.relu),
                             tf.keras.layers.Dropout(0.3),
                             tf.keras.layers.Dense(1, activation = tf.nn.sigmoid)
                             ])
```

Rule of the thumb

>> You implement Dropout regularization only while training the network.

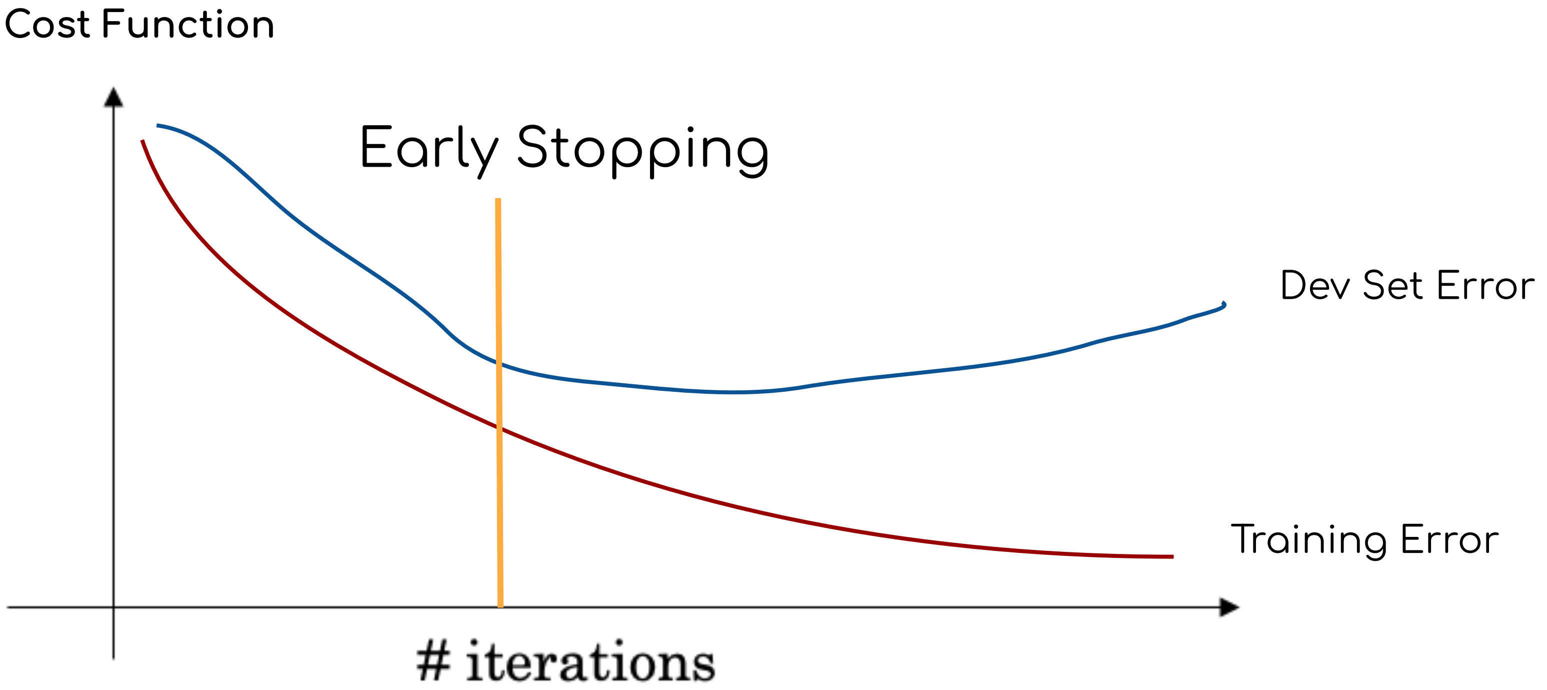
>> You do not apply it while running your testing data through it as you do not want any randomness in your predictions.

Other Regularization Methods

Original Data

Data Augmentation

Other Regularization Methods



Neural Networks in Practice #02

Normalize Inputs,
Vanishing/Exploding Gradients
and Weight Initialization

Next…..