PRODUCT DEVELOPMENT LAB

OBJECT DETECTION
(APPLICATION: SELF DRIVING CAR)

Submitted by:

ARGHYA MAZUMDAR (GROUP LEADER) 115CS0580

ASHISH JHA 115CS0603

BHANU PRASAD USURUPATI 115CS0581

SANJAY HANSDAK 715CS1148

KULDEEP KURROLIYA 715CS2063

RAJ KAPOOR GUPTA 115CS0016

EKTA RAKESH 115CS0232

ABSTRACT

We built our own self-driving car. This is going to be a modelled version of a car (so it won't be driving on the streets of real cities) but still - it will learn how to drive itself. The key word here is *learn*, because the car will not be given any rules on how to operate in the environment before hand - it will have to figure everything out on its own. To achieve this, we will be using Deep Q-Learning. Deep Q-Learning is the result of combining Q-Learning with an Artificial Neural Network. The states of the environment are encoded by a vector which is passed as input into the Neural Network. Then the Neural Network will try to predict which action should be played, by returning as outputs a Q-value for each of the possible actions. Eventually, the best action to play is chosen by either taking the one that has the highest Q-value, or by overlaying a Softmax function.

ACKNOWLEDGEMENTS

We are indebted to the following resources and our project supervisor without whom this would not be possible.

- The Udacity NanoDegree Self Driving Car Course
- The Open Source Udacity Simulator
- Udemy Artificial Intelligence Course
- Prof Sambit Bakshi (Project Supervisor)

CONTENTS

UNDERLYING SOFTWARE / HARDWARE REQUIRED FOR THE PRODUCT

SOFTWARE REQUIREMENTS:

- PYTHON 3.5 /2.7
- UDACITY CAR SIMULATOR (UNITY GAME ENGINE)
- ANACONDA PACKAGE
- TENSORFLOW PLATFORM
- PYTORCH

HARDWARE REQUIREMENTS

- 8 GB RAM
- INTEL I7 PROCESSOR
- NVIDIA GTX /AMD GRAPHICS
- OPERATING SYSTEM
- WINDOWS 10 OS

All software and hardware are licensed by individual proprietors and have no violation of policy issues.

Introduction to different libraries used:

Pytorch: Pytorch is a python package that provides two high-level features:

1. Tensor computation (like numpy) with strong GPU acceleration
2. Deep Neural Networks built on a tape-based autograd system

Keras: Keras is an open source neural network library written in Python. It is capable of running on top of MXNet, Deeplearning4j, Tensorflow, CNTK or Theano.

Numpy: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

scikit-learn: Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms.

Pandas: In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

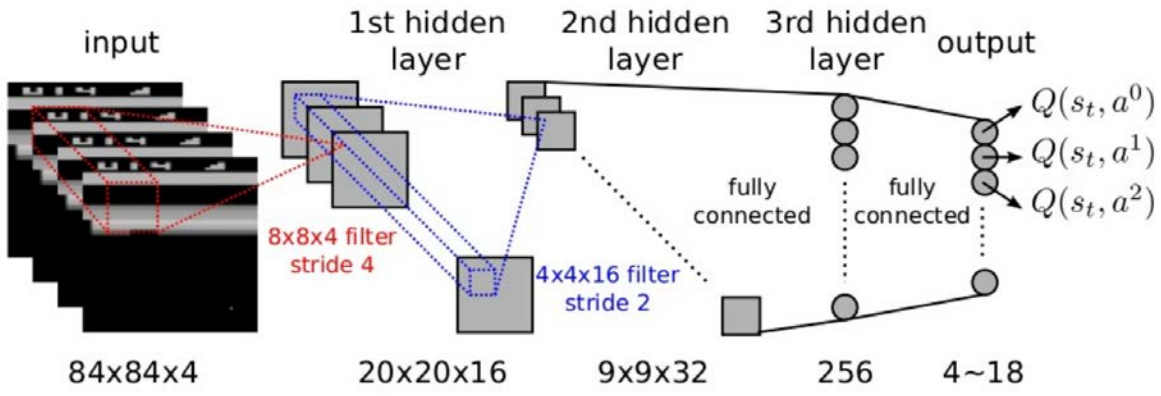
Kivy: Kivy is an open source Python library for developing mobile apps and other multitouch application software with a natural user interface. It can run on Android, iOS, Linux, OS X, and Windows.

The framework contains all the elements for building an application such as:

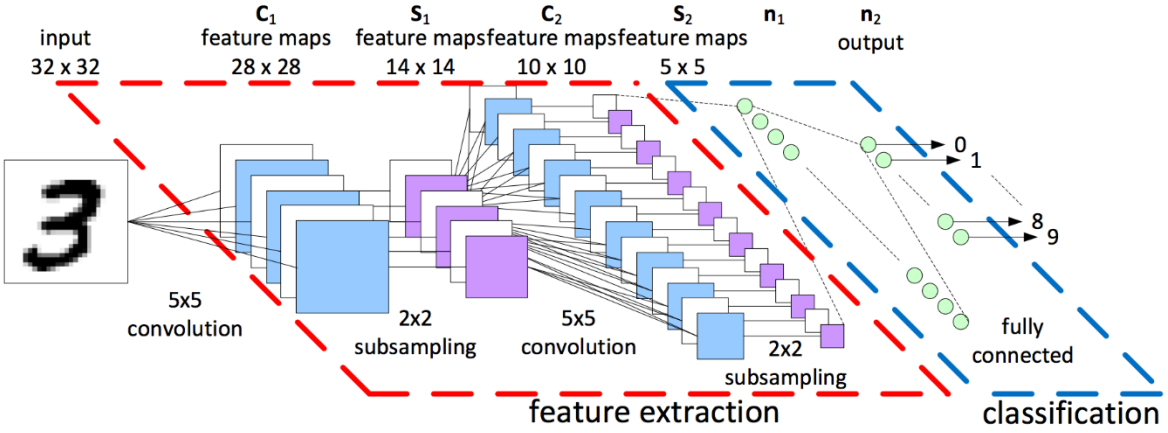
1. extensive input support for mouse, keyboard, TUIO, and OS-specific multitouch events,
2. a graphic library using only OpenGL ES 2, and based on Vertex Buffer Object and shaders,
3. a wide range of widgets that support multitouch,
4. an intermediate language (Kv) used to easily design custom widgets.

Theories and Logics involved:

Deep Q-Learning: Deep Q-Learning is the result of combining Q-Learning with an Artificial Neural Network. The states of the environment are encoded by a vector which is passed as input into the Neural Network. Then the Neural Network will try to predict which action should be played, by returning as outputs a Q-value for each of the possible actions. Eventually, the best action to play is chosen by either taking the one that has the highest Q-value, or by overlaying a Softmax function.



Convolutional neural network: In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery.CNNs use a variation of multilayer perceptrons designed to require minimal pre-processing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

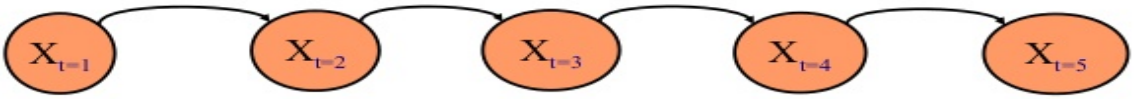


Markov decision processes (MDPs): It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying a wide range of optimization problems solved via dynamic programming and reinforcement learning.

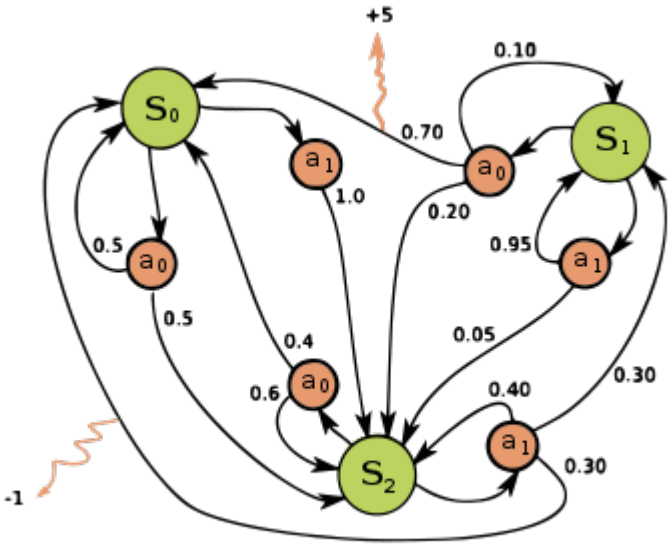
Markov Property

- **Markov Property: The state of the system at time t+1 depends only on the state of the system at time t**

$$P[X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_1 = x_1, X_0 = x_0] = P[X_{t+1} = x_{t+1} \mid X_t = x_t]$$



4



Uses Of Self Driven Cars

Self Driven Cars will be important for the following reasons:
Safety. Self-driving cars will remove a major danger when it comes to driving—human error. Over 90% of all car accidents are caused by human error. For example, over 1,37,000 people were killed in road accidents in 2013 alone in India, that is more than the number of people killed in all our wars put together.

Get to where you're going faster. Self-driving autos will be able to drive closer together and will remove distracted drivers slowing down traffic. This also ties back into safety (20% of injuries in the U.S involve distracted drivers).
Reduce governmental costs. Having self-driving cars on the road will mean fewer collisions, reduced traffic congestion, lower fuel costs and overall less time wasted on the road.
Car ownership. For most of us, the amount of time actually spent driving is very little. Most of the time, our cars are idly parked. This is why services like Uber and Lyft are so popular. Imagine the same service; but the car is now self-driving. This would reduce the costs, improve passenger safety, and lessen the need for individual car ownership.

SOURCE CODE

DRIVE CODE:

```
#parsing command line arguments
import argparse
#decoding camera images
import base64
#for frametimestamp saving
from datetime import datetime
#reading and writing files
import os
#high level file operations
import shutil
#matrix math
import numpy as np
#real-time server
import socketio
#concurrent networking
import eventlet
```

```
#web server gateway interface
import eventlet.wsgi
#image manipulation
from PIL import Image
#web framework
from flask import Flask
#input output
from io import BytesIO

#load our saved model
from keras.models import load_model

#helper class
import utils

#initialize our server
sio = socketio.Server()
#our flask (web) app
app = Flask(__name__)
#init our model and image array as empty
model = None
prev_image_array = None

#set min/max speed for our autonomous car
MAX_SPEED = 25
MIN_SPEED = 10

#and a speed limit
speed_limit = MAX_SPEED

#registering event handler for the server
@sio.on('telemetry')
```

```
def telemetry(sid, data):
    if data:
        # The current steering angle of the car
        steering_angle = float(data["steering_angle"])
        # The current throttle of the car, how hard to push peddle
        throttle = float(data["throttle"])
        # The current speed of the car
        speed = float(data["speed"])
        # The current image from the center camera of the car
        image = Image.open(BytesIO(base64.b64decode(data["image"])))
        try:
            image = np.asarray(image)       # from PIL image to numpy array
            image = utils.preprocess(image) # apply the preprocessing
            image = np.array([image])       # the model expects 4D array

            # predict the steering angle for the image
            steering_angle = float(model.predict(image, batch_size=1))
            # lower the throttle as the speed increases
            # if the speed is above the current speed limit, we are on a downhill.
            # make sure we slow down first and then go back to the original max speed.
            global speed_limit
            if speed > speed_limit:
                speed_limit = MIN_SPEED  # slow down
            else:
                speed_limit = MAX_SPEED
            throttle = 1.0 - steering_angle**2 - (speed/speed_limit)**2

            print('{} {} {}'.format(steering_angle, throttle, speed))
            send_control(steering_angle, throttle)
        except Exception as e:
            print(e)
```

```
        # save frame
        if args.image_folder != '':
            timestamp = datetime.utcnow().strftime('%Y_%m_%d_%H_%M_%S_%f')[:-3]
            image_filename = os.path.join(args.image_folder, timestamp)
            image.save('{}.jpg'.format(image_filename))
    else:

        sio.emit('manual', data={}, skip_sid=True)


@sio.on('connect')
def connect(sid, environ):
    print("connect ", sid)
    send_control(0, 0)


def send_control(steering_angle, throttle):
    sio.emit(
        "steer",
        data={
            'steering_angle': steering_angle.__str__(),
            'throttle': throttle.__str__()
        },
        skip_sid=True)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Remote Driving')
    parser.add_argument(
        'model',
        type=str,
        help='Path to model h5 file. Model should be on the same path.'
```

```
)
parser.add_argument(
    'image_folder',
    type=str,
    nargs='?',
    default='',
    help='Path to image folder. This is where the images from the run will be saved.'
)
args = parser.parse_args()

#load model
model = load_model(args.model)

if args.image_folder != '':
    print("Creating image folder at {}".format(args.image_folder))
    if not os.path.exists(args.image_folder):
        os.makedirs(args.image_folder)
    else:
        shutil.rmtree(args.image_folder)
        os.makedirs(args.image_folder)
    print("RECORDING THIS RUN ...")
else:
    print("NOT RECORDING THIS RUN ...")

# wrap Flask application with engineio's middleware
app = socketio.Middleware(sio, app)

# deploy as an eventlet WSGI server
eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```

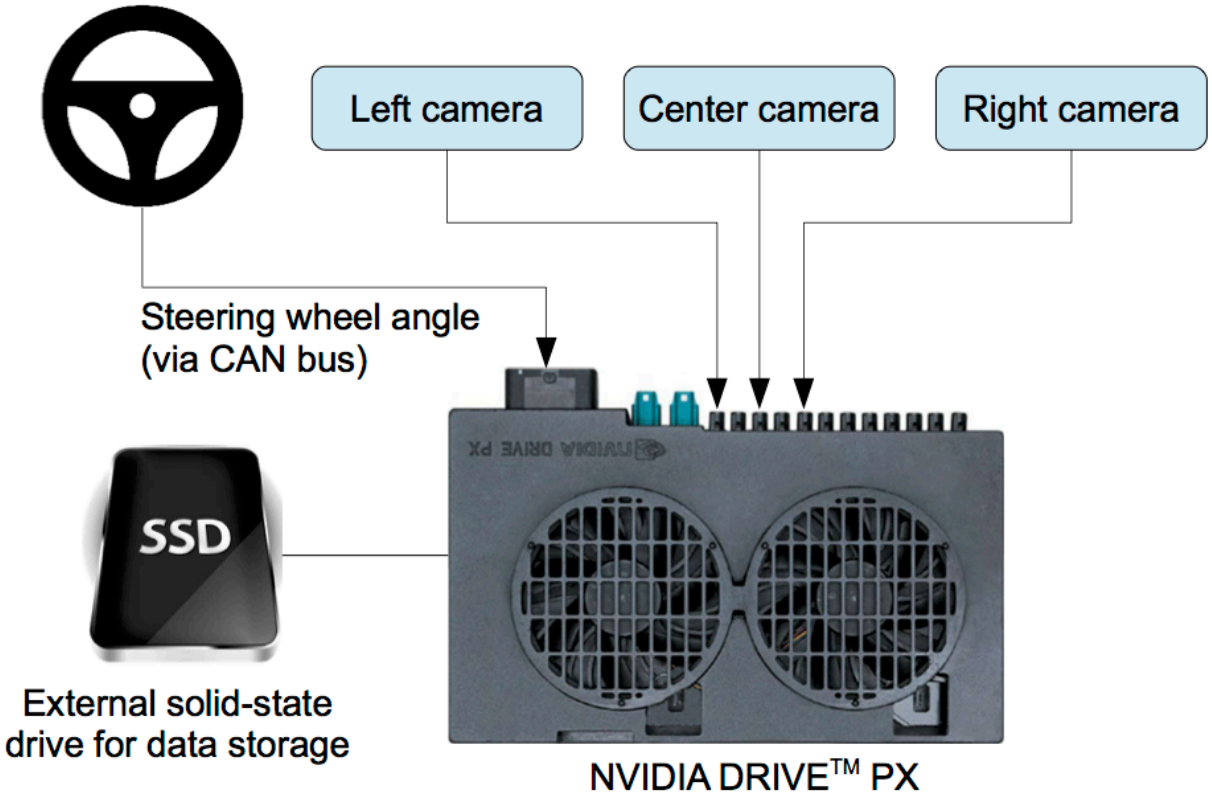
APPLICATION

Data Generation

- Records images from center, left, and right cameras w/ associated steering angle, speed, throttle and brake.
- saves to CSV
- ideally you have a joystick, but keyboard works too

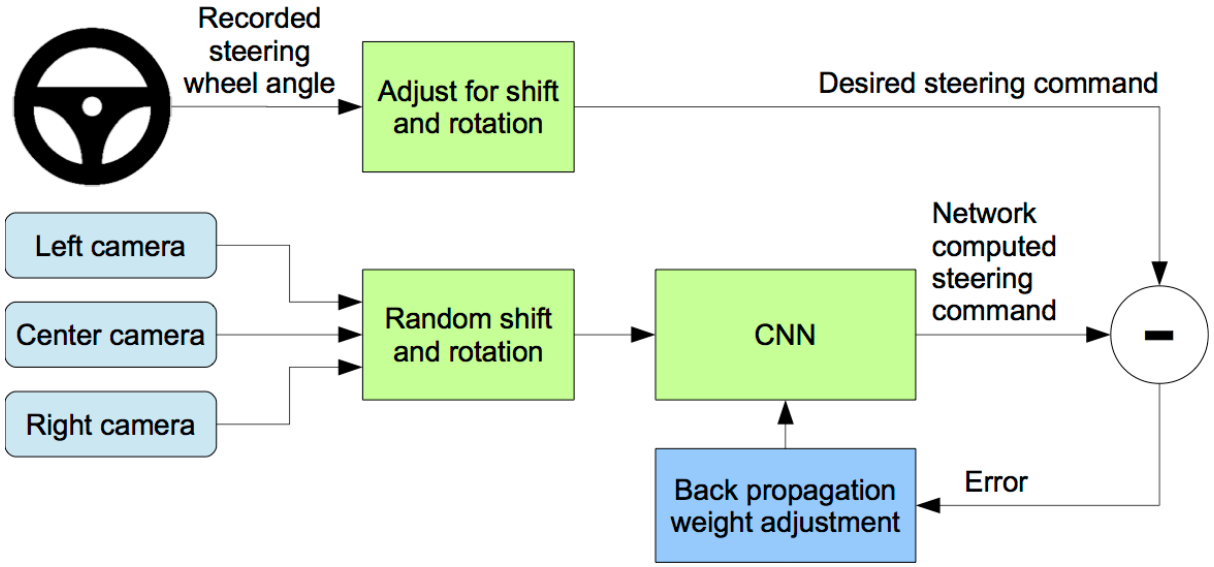
Training Mode - Behavioural cloning

We use a 9 layer convolutional network, based off of Nvidia's end-to-end learning for self-driving car paper. 72 hours of driving data was collected in all sorts of conditions from human drivers



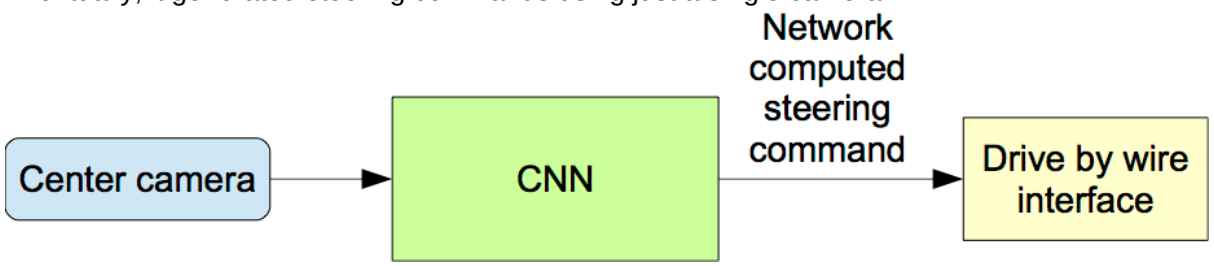
In order to make the system independent of the car geometry, the steering command is 1/r, where r is the turning radius in meters. 1/r was used instead of r to prevent a singularity when

driving straight (the turning radius for driving straight is infinity). 1/r smoothly transitions through zero from left turns (negative values) to right turns (positive values).



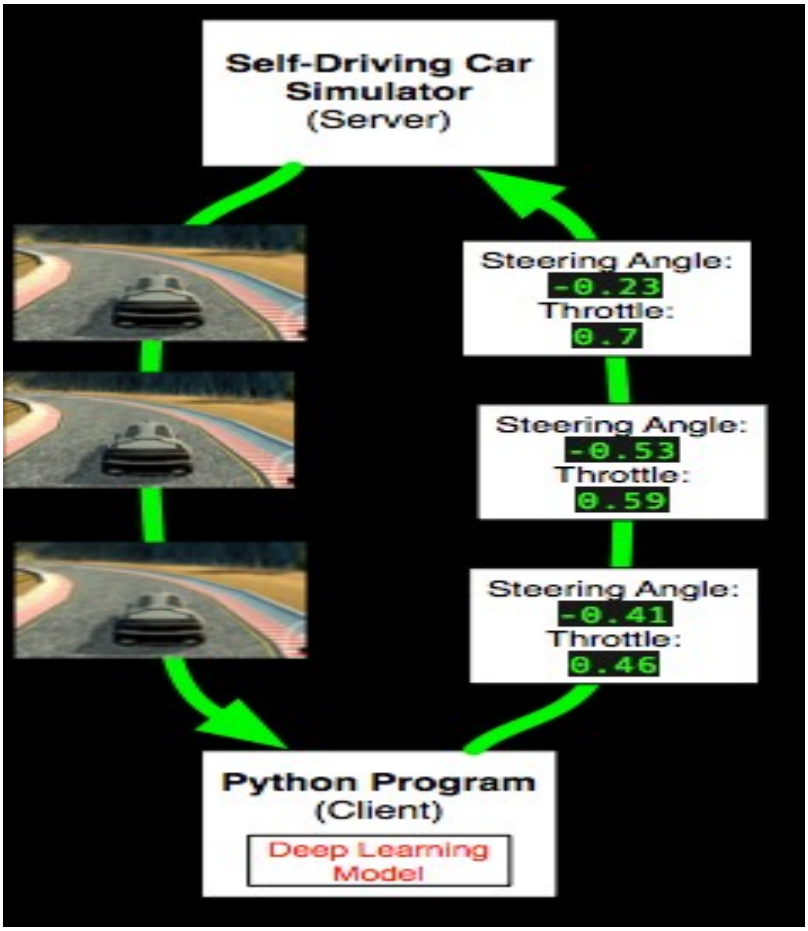
Images are fed into a CNN that then computes a proposed steering command. The proposed command is compared to the desired command for that image, and the weights of the CNN are adjusted to bring the CNN output closer to the desired output. The weight adjustment is accomplished using back propagation

Eventually, it generated steering commands using just a single camera



Testing mode

We will just run autonomous mode, then run our model and the car will start driving



CONCLUSION:

The project is composed of two modules which consist of the object detector simulator developed in Pytorch and Kivy framework. The other module was developed using Udacity's open source Car Simulator in which the training and test script were written using the Tensorflow and Qt framework by Unity's Game Engine.

SOURCES AND REFERENCES

Books

- We had read with interest the Elements of Statistical Learning and Murphy's Machine Learning - a Probabilistic Perspective

Neural Networks and Deep Learning is a free online book. The book teach us about:

- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data
- Deep learning, a powerful set of techniques for learning in neural networks
- Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. This book will teach you many of the core concepts behind neural networks and deep learning.

Free Online Books

1. Deep Learning by Yoshua Bengio, Ian Goodfellow and Aaron Courville (05/07/2015)
2. Neural Networks and Deep Learning by Michael Nielsen (Dec 2014)
3. Deep Learning by Microsoft Research (2013)
4. Deep Learsning Tutorial by LISA lab, University of Montreal (Jan 6 2015)
5. neuraltalk by Andrej Karpathy : numpy-based RNN/LSTM implementation
6. An introduction to genetic algorithms
7. Artificial Intelligence: A Modern Approach
8. Deep Learning in Neural Networks: An Overview

- **Review Papers**
  - **Representation Learning: A Review and New Perspectives**, Yoshua Bengio, Aaron Courville, Pascal Vincent, Arxiv, 2012.
  - The monograph or review paper **Learning Deep Architectures for AI** (Foundations & Trends in Machine Learning, 2009).
  - Deep Machine Learning – A New Frontier in Artificial Intelligence Research – a **survey paper** by Itamar Arel, Derek C. Rose, and Thomas P. Karnowski.

- Graves, A. (2012). *Supervised sequence labelling with recurrent neural networks*(Vol. 385). Springer.
- Schmidhuber, J. (2014). Deep Learning in Neural Networks: An Overview. 75 pages, 850+ references, **http://arxiv.org/abs/1404.7828**, PDF & LATEX source & complete public BIBTEX file under **http://www.idsia.ch/~juergen/deep-learning-overview.html**.
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. **"Deep learning."** *Nature* 521, no. 7553 (2015): 436-444.

- **Reinforcement Learning**
  - Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).
  - Volodymyr Mnih, Nicolas Heess, Alex Graves, Koray Kavukcuoglu. "**Recurrent Models of Visual Attention**" ArXiv e-print, 2014.

Deep- and Machine-Learning Fora

- Google + Deep Learning Group
- KDNuggets: Data Science Hub
- Datatau: Hacker News for Data Science
- r/MachineLearning
- Deeplearning.net: A Portal for Theano/PyLearn
- Gitter Channel for Deeplearning4j

Other Resources

- Open Data for Deep Learning
- Machine Learning: Generative and Discriminative Models (Power Point); Sargur N. Srihari
- Neural Networks Demystified (A seven-video series)
- A Neural Network in 11 Lines of Python
- A Step-by-Step Backpropagation Example
- Generative Learning algorithms; Notes by Andrew Ng
- Calculus on Computational Graphs: Backpropagation
- Understanding LSTM Networks
- Probability Cheatsheet

Research Papers referred

Convolutional Neural Network Models

- **Rethinking the inception architecture for computer vision** (2016), C. Szegedy et al. [pdf]
- **Inception-v4, inception-resnet and the impact of residual connections on learning** (2016), C. Szegedy et al. [pdf]
- **Identity Mappings in Deep Residual Networks** (2016), K. He et al. [pdf]

- **Deep residual learning for image recognition** (2016), K. He et al. [pdf]
- **Spatial transformer network** (2015), M. Jaderberg et al., [pdf]
- **Going deeper with convolutions** (2015), C. Szegedy et al. [pdf]
- **Very deep convolutional networks for large-scale image recognition** (2014), K. Simonyan and A. Zisserman [pdf]
- **Return of the devil in the details: delving deep into convolutional nets** (2014), K. Chatfield et al. [pdf]
- **OverFeat: Integrated recognition, localization and detection using convolutional networks** (2013), P. Sermanet et al. [pdf]
- **Maxout networks** (2013), I. Goodfellow et al. [pdf]
- **Network in network** (2013), M. Lin et al. [pdf]
- **ImageNet classification with deep convolutional neural networks** (2012), A. Krizhevsky et al.

Optimization / Training Techniques

- **Training very deep networks** (2015), R. Srivastava et al. [pdf]
- **Batch normalization: Accelerating deep network training by reducing internal covariate shift** (2015), S. Loffe and C. Szegedy [pdf]
- **Delving deep into rectifiers: Surpassing human-level performance on imagenet classification** (2015), K. He et al. [pdf]
- **Dropout: A simple way to prevent neural networks from overfitting** (2014), N. Srivastava et al. [pdf]
- **Adam: A method for stochastic optimization** (2014), D. Kingma and J. Ba [pdf]
- **Improving neural networks by preventing co-adaptation of feature detectors** (2012), G. Hinton et al. [pdf]
- **Random search for hyper-parameter optimization** (2012) J. Bergstra and Y. Bengio [pdf]

Unsupervised / Generative Models

- **Pixel recurrent neural networks** (2016), A. Oord et al. [pdf]
- **Improved techniques for training GANs** (2016), T. Salimans et al. [pdf]
- **Unsupervised representation learning with deep convolutional generative adversarial networks** (2015), A. Radford et al. [pdf]
- **DRAW: A recurrent neural network for image generation** (2015), K. Gregor et al. [pdf]
- **Generative adversarial nets** (2014), I. Goodfellow et al. [pdf]