Deep Learning and its applications to robotics

Pinxin Long

Outline

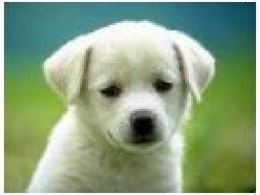
- **An Introduction to Deep Learning**

- **Deep Learning Libraries (Keras)**

- **Its Applications to Robotics**

Machine Learning

$$x \longrightarrow \boxed{F(x)} \longrightarrow y \begin{cases} \text{Class label} \\ \text{(Classification)} \\ \\ \text{Vector} \\ \text{(Estimation)} \end{cases}$$



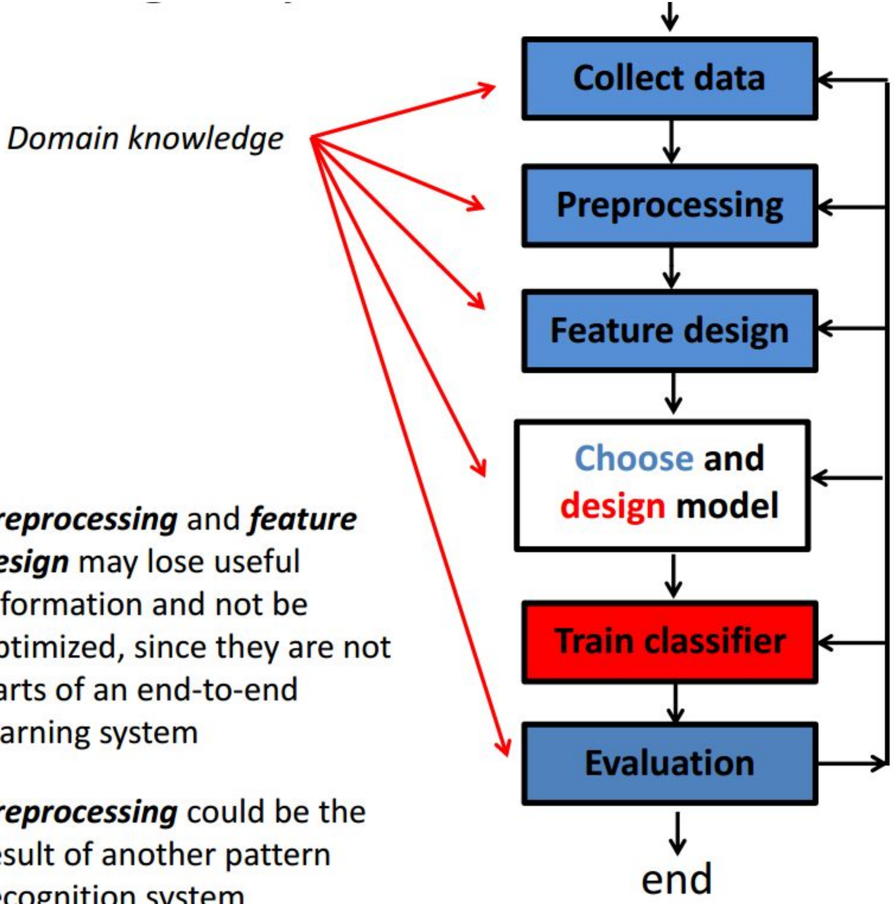
$\xrightarrow{\text{Object recognition}}$ {dog, cat, horse, flower, …}

deep *learning* of data representations

$$\underset{\text{input}}{x} \xrightarrow[\text{hierarchical}]{\mathcal{M}(x|\Theta)} \underset{\text{output}}{y}$$

joint learning
of representations with
increased levels of
abstraction
+ classification or
regression

Traditional Design Cycle



Preprocessing and ***feature design*** may lose useful information and not be optimized, since they are not parts of an end-to-end learning system

Preprocessing could be the result of another pattern recognition system

Interest of people working on **computer vision, speech recognition, medical image processing,…**

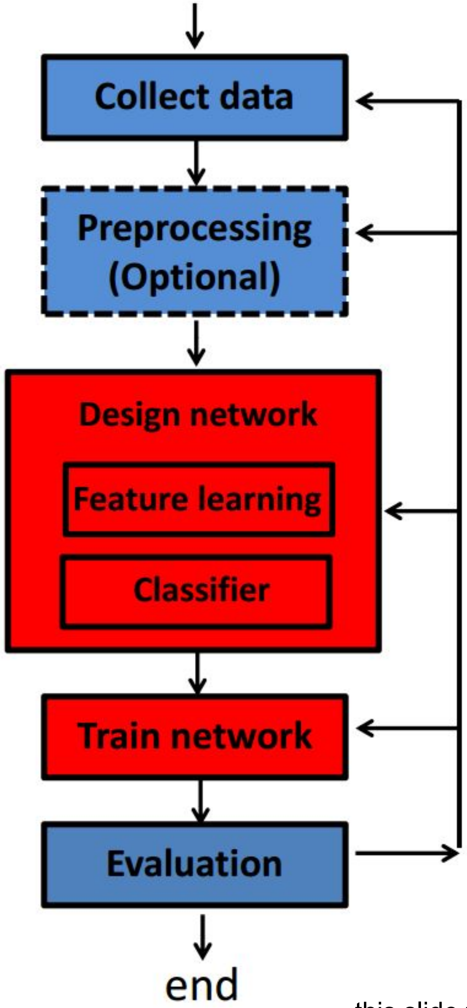
Interest of people working on **machine learning**

Interest of people working on **machine learning** and **computer vision, speech recognition, medical image processing,…**

this slide from Xiaogang Wang

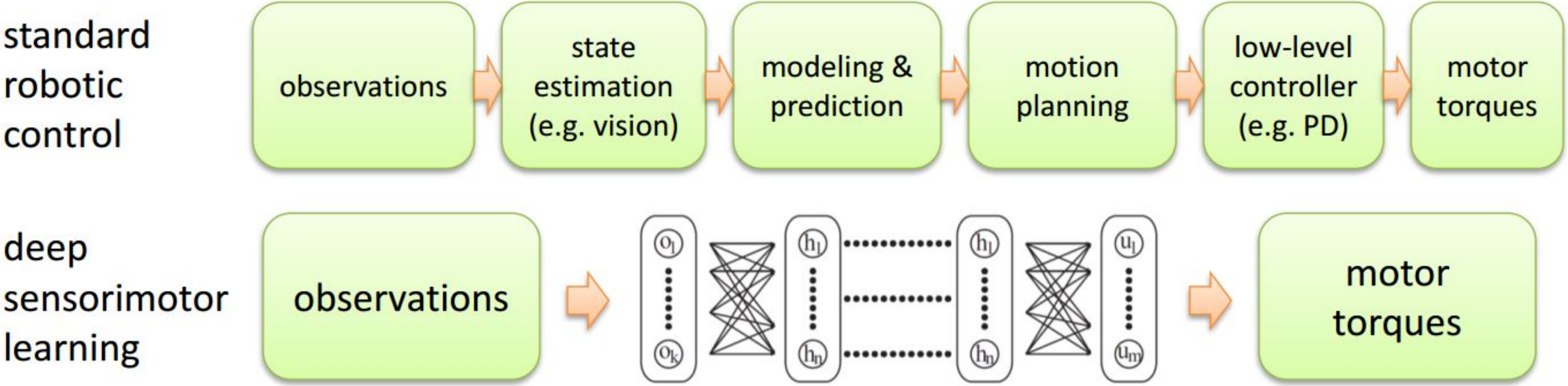
Design Cycle with Deep Learning

- Learning plays a bigger role in the design circle
- Feature learning becomes part of the end-to-end learning system
- Preprocessing becomes optional means that several pattern recognition steps can be merged into one end-to-end learning system
- Feature learning makes the key difference
- We underestimated the importance of data collection and evaluation



this slide from Xiaogang Wang

End-to-end robotic control



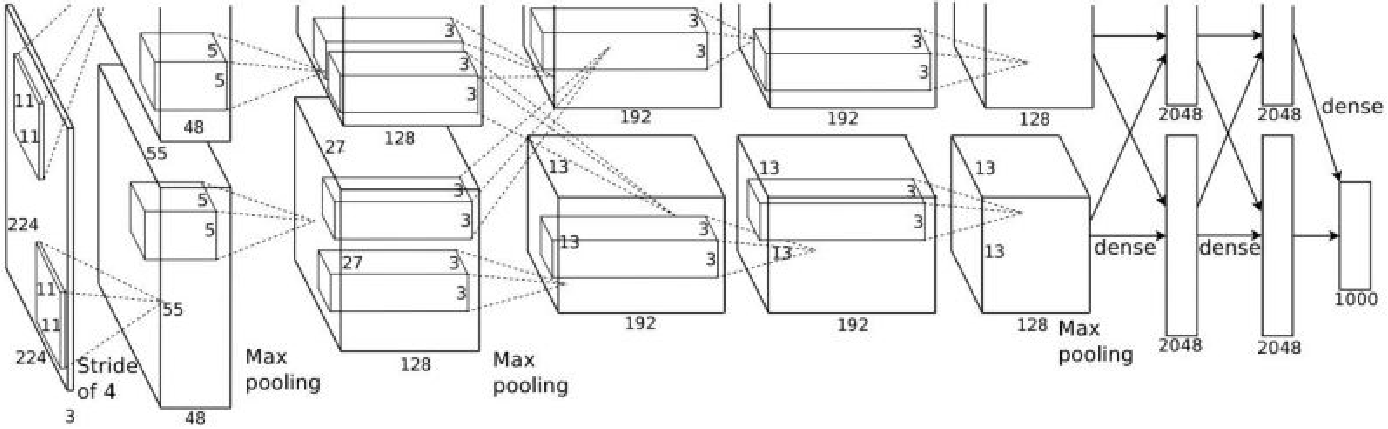
this slide from Sergey Levine

What is Deep Learning

In general, Deep Learning is Machine Learning algorithms that process data with hierarchical layers, for a non-linear mapping of data.

Until now, most of DL application are using multi-layer neural network.

$$y = f(x)$$



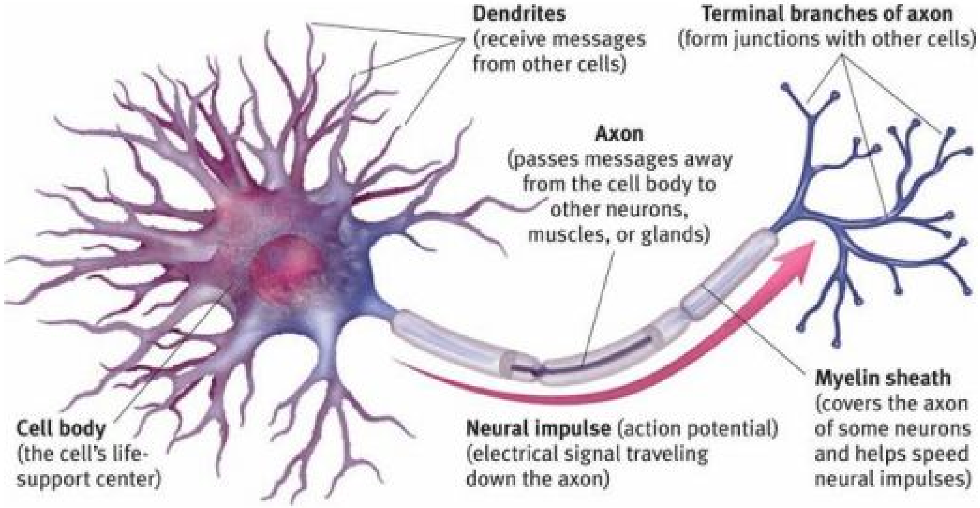
Black box model, but works impressively well.

this slide is from Qinrui Yan

Neuron System

First!! Aritificial neural network is far away from real neurons!!!!!!

Much more sophisticated: Hodgkin-Huxley model



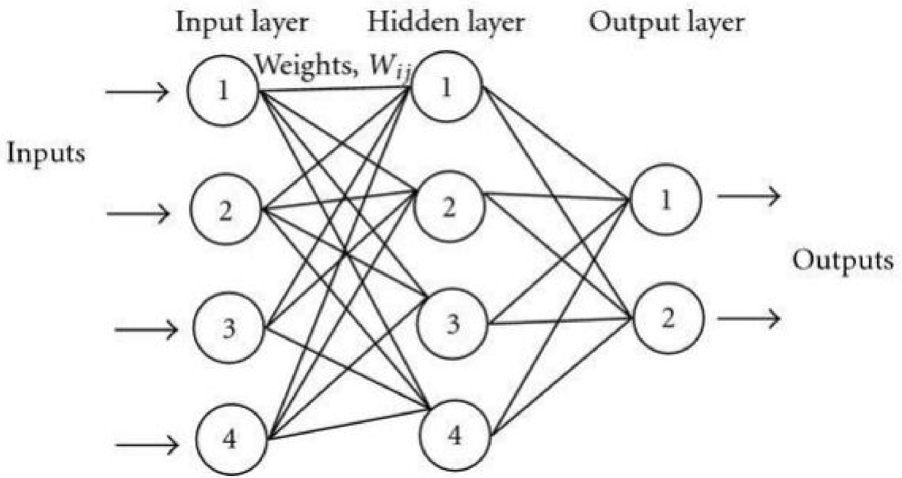
this slide is from Qinrui Yan

Artificial Neural Network

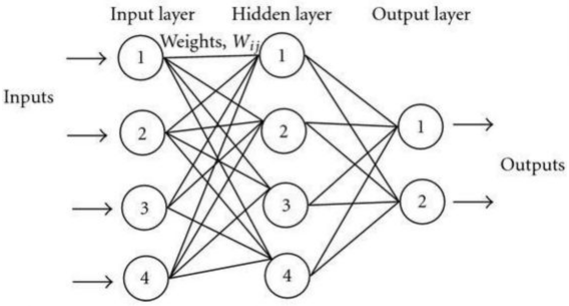
Although ANN can be used as a **regression** or **clustering** algorithm, it was initially created for **classification**.

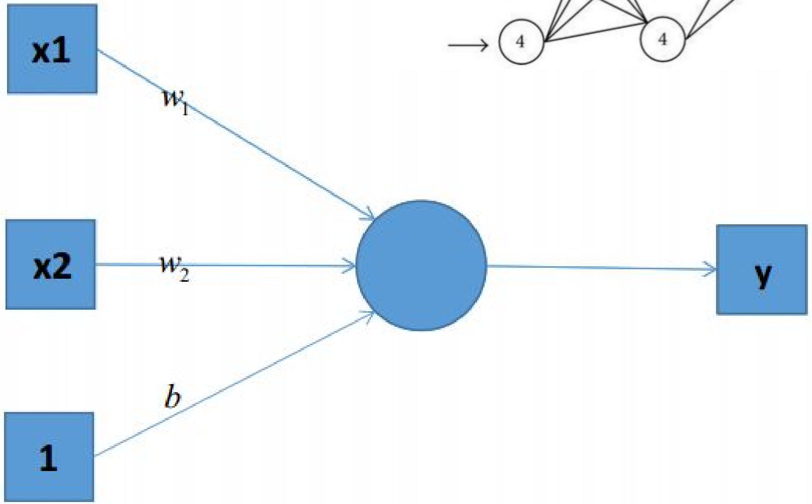
$$E = \sum_k (h_k - y_k)^2$$

h is the output under current weights.
y is the labelled data of current input.

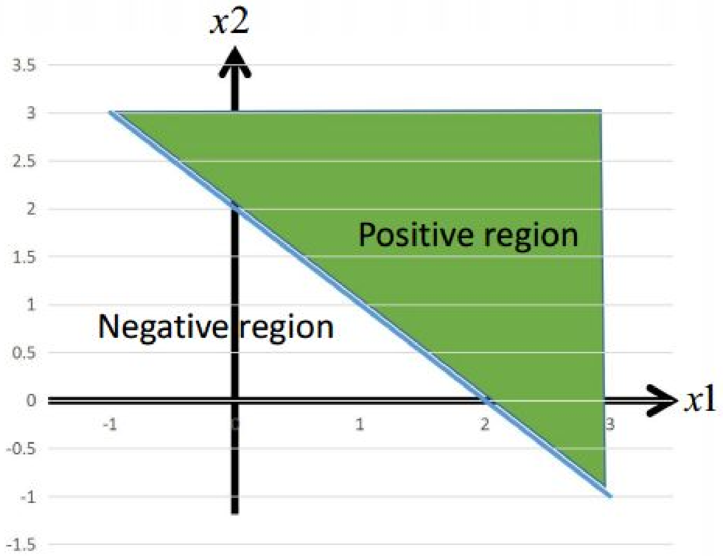


Perceptron





$$y = w_1 x_1 + w_2 x_2 + b$$

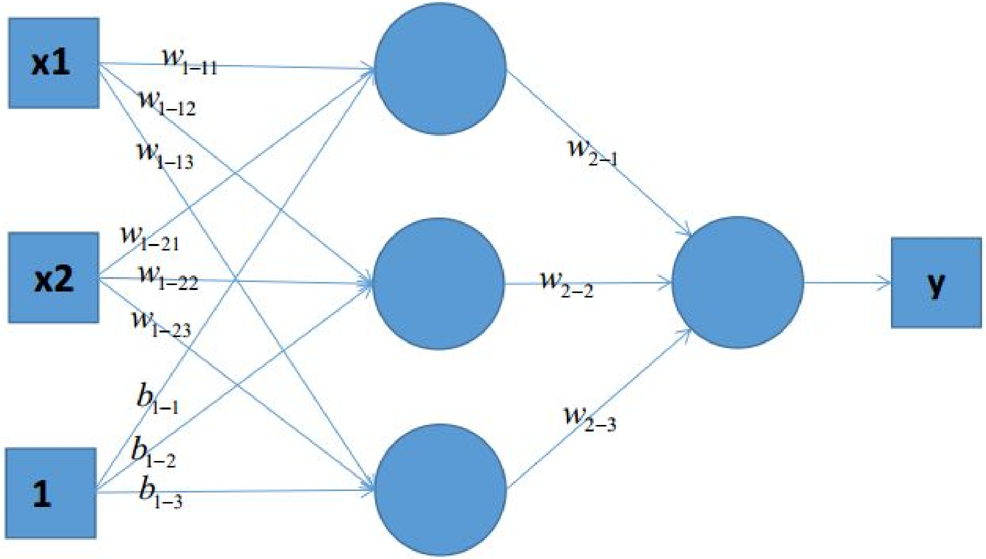


$$w_1 = 1, w_2 = 1, b = -2$$

single layer perceptron is a linear classifier

this slide is from Qinrui Yan

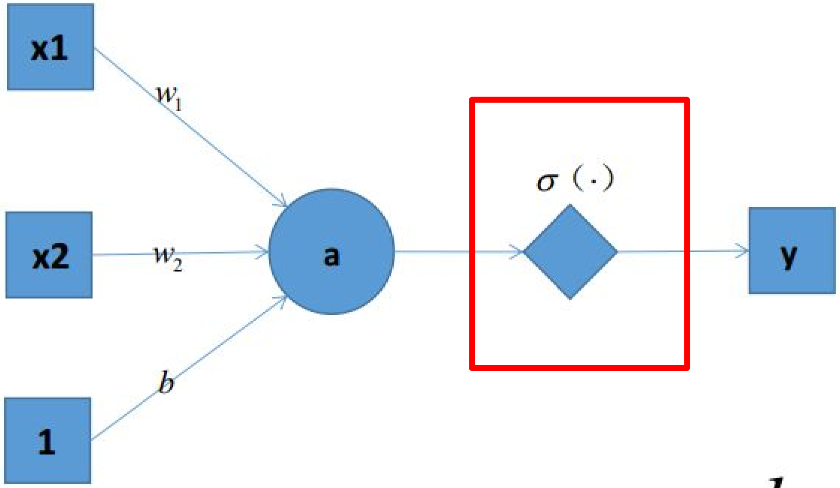
Perceptron with one hidden layer



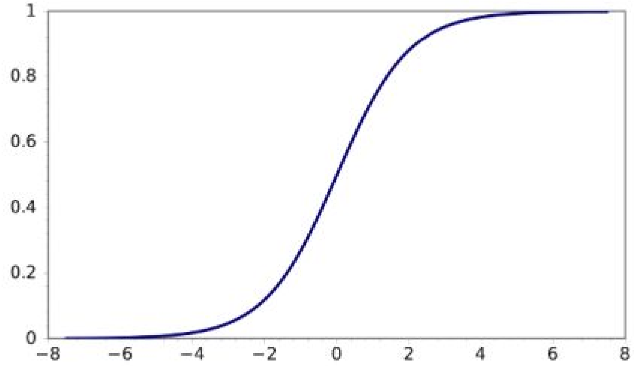
$$
\begin{aligned}
y &= w_{2\text{-}1}(w_{1\text{-}11}x_1 + w_{1-21}x_2 + b_{1\text{-}1}) \\
&+ w_{2\text{-}2}(w_{1\text{-}12}x_1 + w_{1-22}x_2 + b_{1\text{-}2}) \\
&+ w_{2\text{-}3}(w_{1\text{-}13}x_1 + w_{1-23}x_2 + b_{1\text{-}3})
\end{aligned}
$$

this slide is from Qinrui Yan

Perceptron with non-linear activation function



$$a = w_1 x_1 + w_2 x_2 + b$$
$$y = \sigma(a)$$

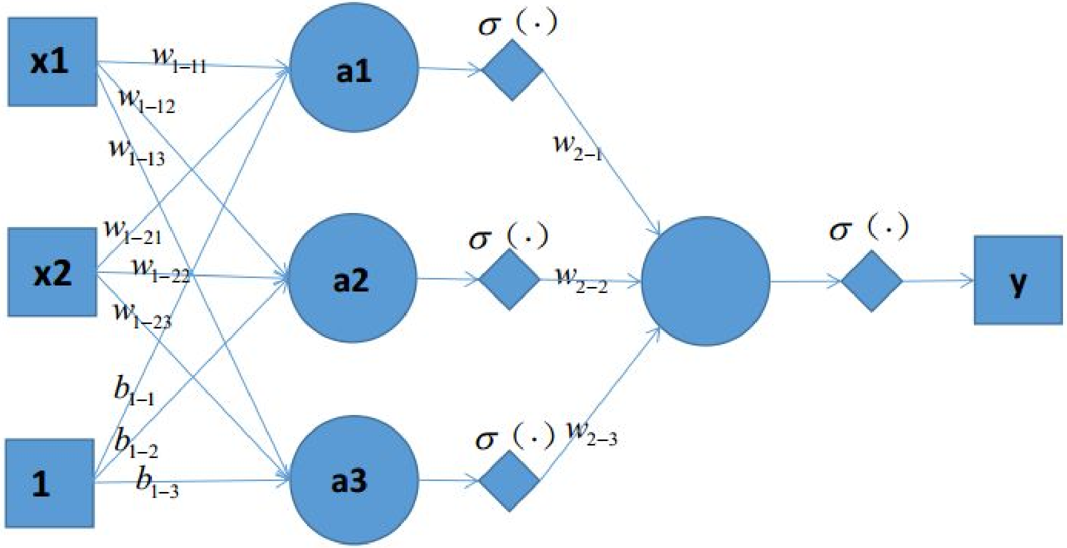


$\sigma(\cdot)$ is a non-linear activation function, sigmoid was the most popular one,

$$\sigma(y) = \frac{1}{1+e^{-y}}$$

this slide is from Qinrui Yan

Perceptron with non-linear activation function



$$a1 = w_{1\text{-}11}x_1 + w_{1-21}x_2 + b_{1\text{-}1}$$

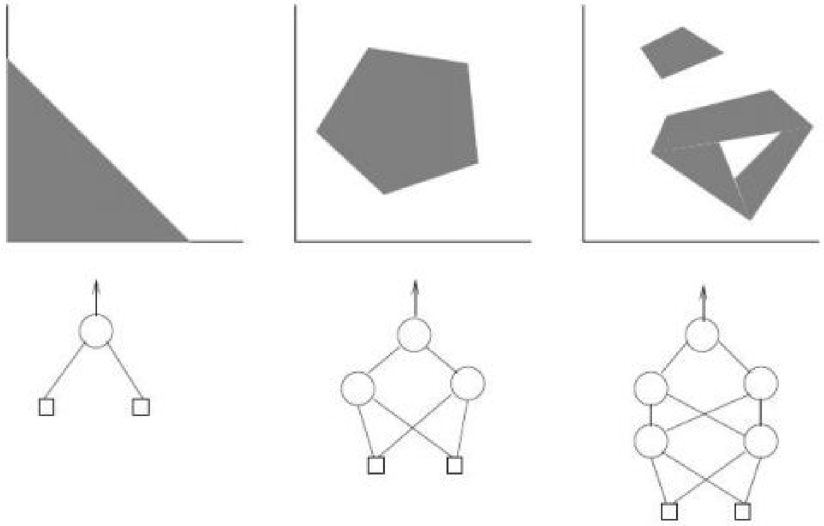
$$a2 = w_{1\text{-}12}x_1 + w_{1-22}x_2 + b_{1\text{-}2}$$

$$a3 = w_{1\text{-}13}x_1 + w_{1-23}x_2 + b_{1\text{-}3}$$

$$y = \sigma(w_{2-1}\sigma(a1) + w_{2-2}\sigma(a2) + w_{2-3}\sigma(a3))$$

this slide is from Qinrui Yan

Power of 'deep' structure



One neuron (perceptron): Linear separation
One hidden layer: Realization of convex regions
Two hidden layers: Realization of non-convex regions

Problem of 'deep' structure

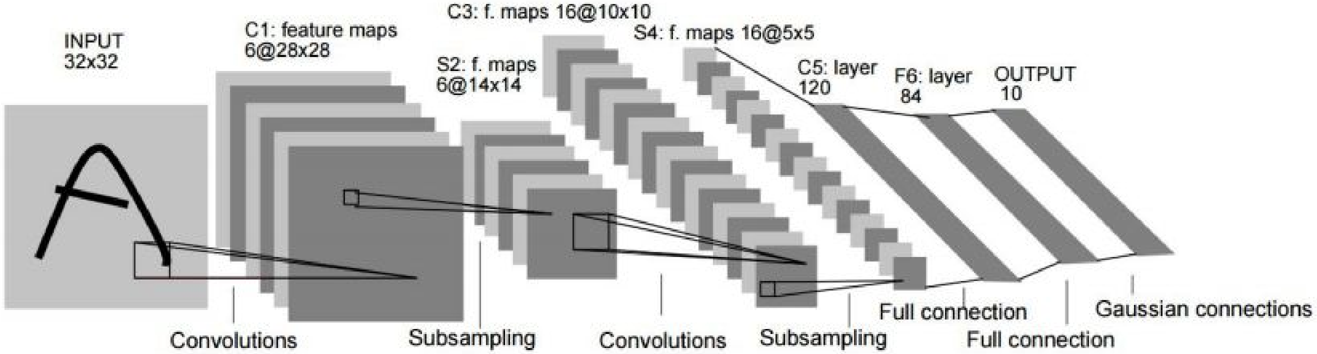
In general, the more layers a neural network has, the more representative ability it has.

- Gradient diffusion: Errors are difficult to back propogate.

- Overfitting: Too many parameters, easy to drop into local minimal.

Convolution neural network (CNN)

- Designed for 2-dimensional object recognition, take the spatial information into account.



- Basic types of layers:
  1. convolution layer: for feature extraction
  2. sub-sampling layer: for simplifying feature, prevent overfitting.
  3. fully connected layer: for final classification.

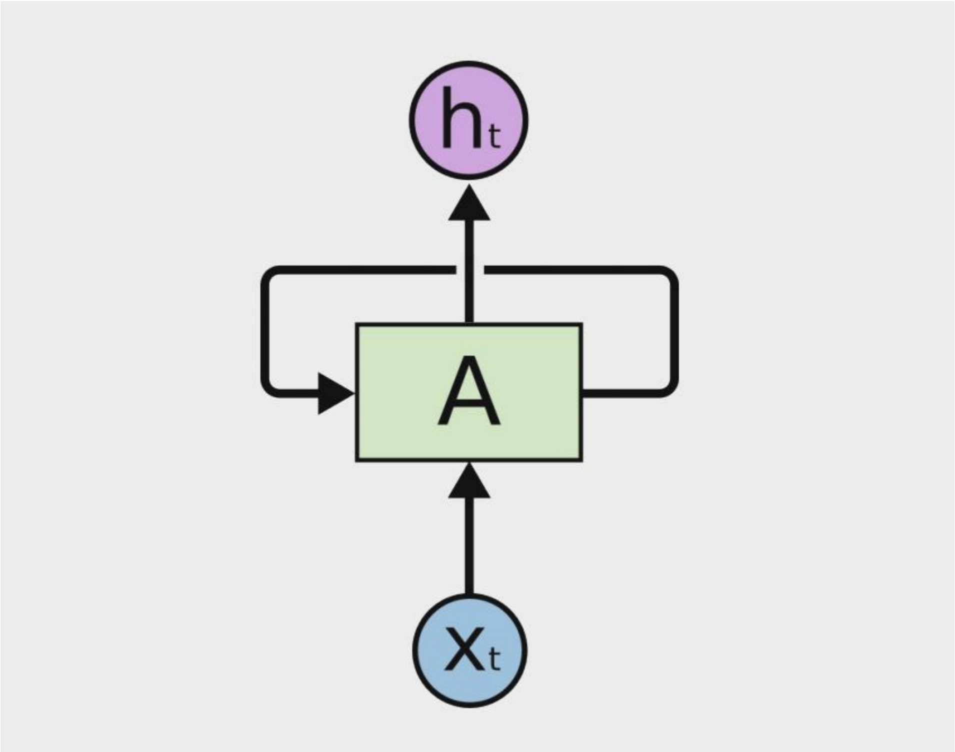
this slide is from Qinrui Yan

Classic Networks:

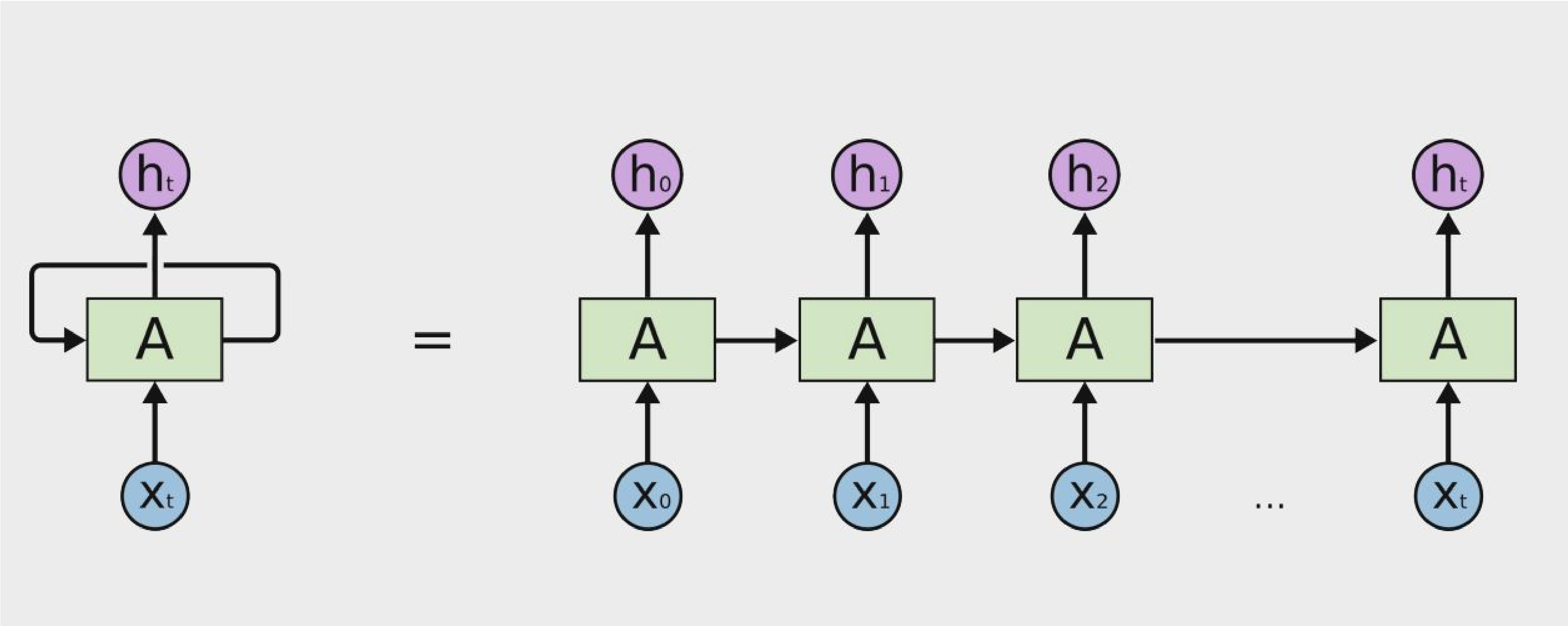
1. AlexNet(2012): A.Krizhevsky & G.Hinton(U Toronto)
2. GoogleNet(2014): C.Szegedy & etc (Google, Umich, UNC)
3. VGG(2014): K.Simonyan & A.Zisserman (Oxford)
4. SPP-Net(2014): He Kaiming & etc(MSRA)
5. Deep residual network(2015): He Kaiming & etc(MSRA)

this slide is from Qinrui Yan

Recurrent Neural Networks (RNNs)



http://colah.github.io/posts/2015-08-Understanding-LSTMs/



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

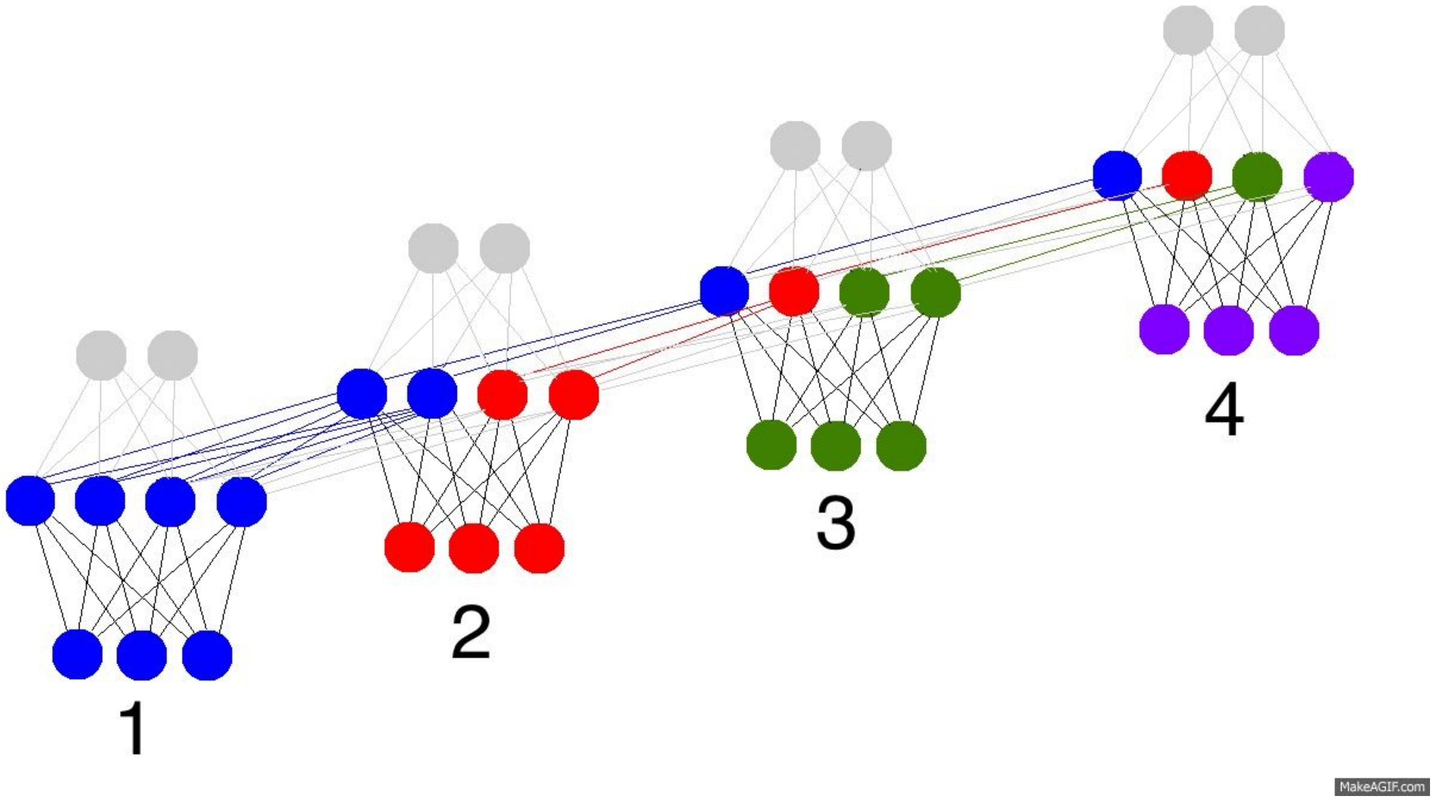
with **hidden layer recurrence:**

(input + empty_hidden) -> hidden -> output
(input + prev_hidden) -> hidden -> output
(input + prev_hidden) -> hidden -> output
(input + prev_hidden) -> hidden -> output

.... and 4 timesteps with **input layer recurrence....**

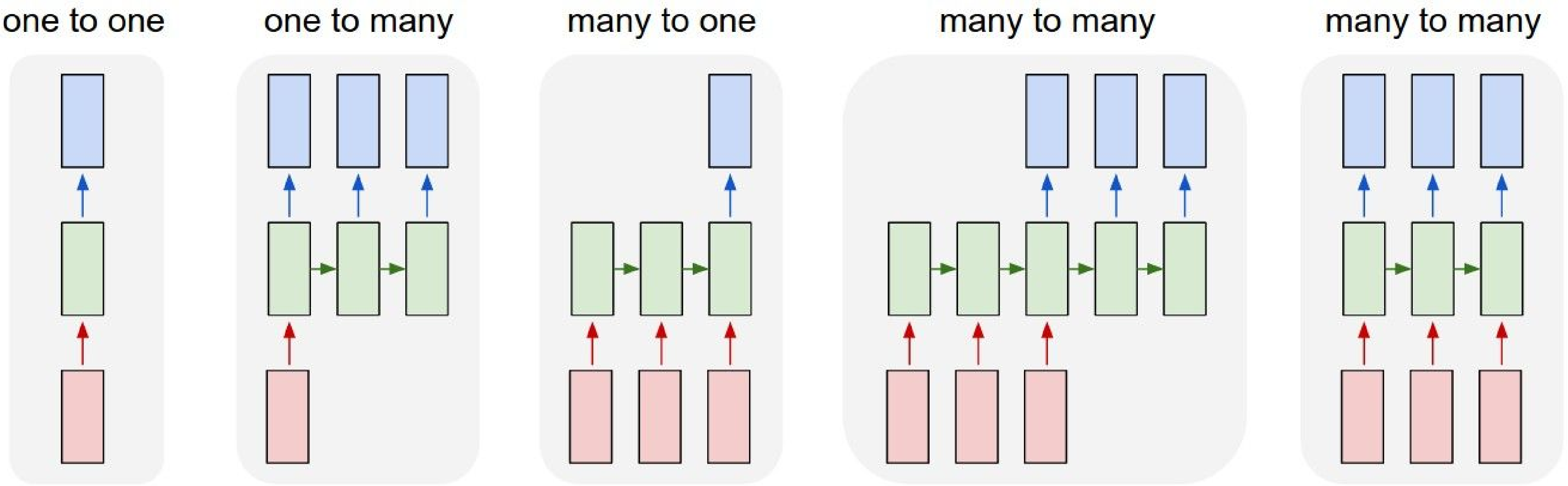
(input + empty_input) -> hidden -> output
(input + prev_input) -> hidden -> output
(input + prev_input) -> hidden -> output
(input + prev_input) -> hidden -> output

https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/



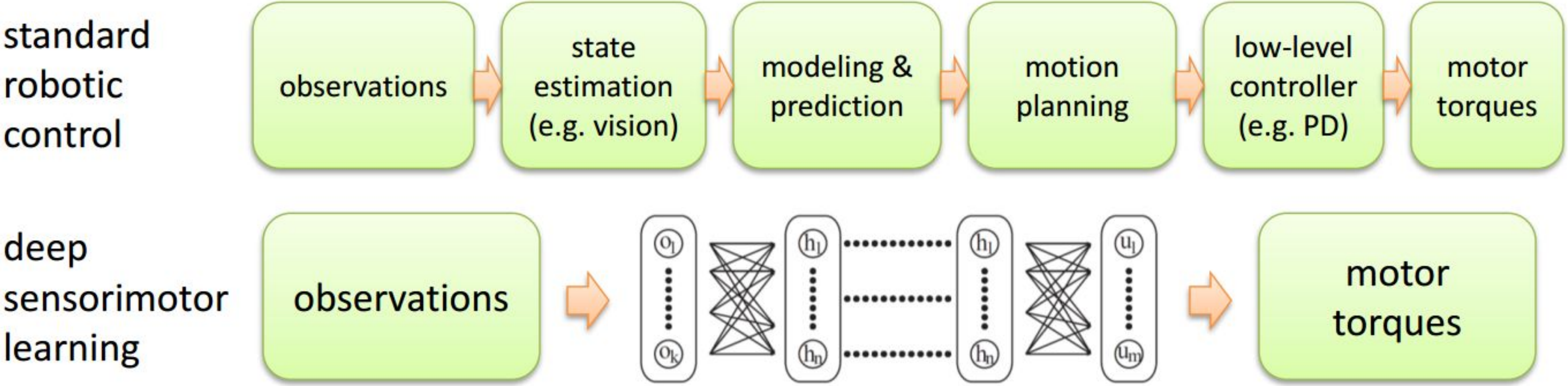
https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/

RNNs



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

End-to-end robotic control



Drawbacks of deep learning

1. Computation is expensive
2. It is very difficult and labour intensive to get labelled data.
3. Brute force.

We have to realize, deep neural network is not the final solution of Artificial Intelligence. Actually for human, most of knowledge comes from unsupervised learning, so a long way need to go.
How to bring prior knowledge in to the model is still a important issue.

this slide is from Qinrui Yan

Conclusion

1. Deep Learning is a very **simple but powerful** tool of feature learning, especially for perception in Robotics.

2. Where is the training data from?

3. How to simplify data processing according to the specify task of robot.

4. Deep Learning is changing lots of things, sometimes even over our expectations. We should pay attention to its development.

this slide is from Qinrui Yan

Deep Learning Libraries

tensorflow/tensorflow — C++ ★ 28,686 · 11,773

Computation using data flow graphs for scalable machine learning

BVLC/caffe — C++ ★ 11,462 · 6,910

Caffe: a fast open framework for deep learning.

fchollet/keras — Python ★ 7,167 · 2,096

Deep Learning library for Python. Convnets, recurrent neural networks, and more. Runs on Theano or TensorFlow.

dmlc/mxnet — C++ ★ 4,510 · 1,671

Lightweight, Portable, Flexible Distributed/Mobile Deep Learning with Dynamic, Mutation-aware Dataflow Dep Scheduler; for Python, R, Julia, Scala, Go, Javascript and more

More info...https://github.com/zer0n/deepframeworks

Keras: An Introduction

What is Keras?

- Neural Network library written in Python
- Designed to be minimalistic & straight forward yet extensive
- Built on top of either Theano or newly TensorFlow

Why use Keras?

- Simple to get started, simple to keep going
- Written in python and highly modular; easy to expand
- Deep enough to build serious models

General Design

General idea is to based on layers and their input/output

- Prepare your inputs and output tensors
- Create first layer to handle input tensor
- Create output layer to handle targets
- Build virtually any model you like in between

Layers and Layers

Keras has a number of pre-built layers.

- Regular dense, MLP type

```
keras.layers.core.Dense(output_dim,
                        init='glorot_uniform',
                        activation='linear',
                        weights=None,
                        W_regularizer=None, b_regularizer=None, activity_regularizer=None,
                        W_constraint=None, b_constraint=None,
                        input_dim=None)
```

- Recurrent layers, LSTM, GRU, etc.

```
keras.layers.recurrent.GRU(output_dim,
        init='glorot_uniform', inner_init='orthogonal',
        activation='sigmoid', inner_activation='hard_sigmoid',
        return_sequences=False,
        go_backwards=False,
        stateful=False,
        input_dim=None, input_length=None)
```

- 1D Convolutional layers

```
keras.layers.convolutional.Convolution1D(nb_filter, filter_length,
        init='uniform',
        activation='linear',
        weights=None,
        border_mode='valid',
        subsample_length=1,
        W_regularizer=None, b_regularizer=None,
        W_constraint=None, b_constraint=None,
        input_dim=None, input_length=None)
```

- 2D Convolutional layers

```
keras.layers.convolutional.Convolution2D(nb_filter, nb_row, nb_col,
        init='glorot_uniform',
        activation='linear',
        weights=None,
        border_mode='valid',
        subsample=(1, 1),
        W_regularizer=None, b_regularizer=None,
        W_constraint=None,
        dim_ordering='th')
```

Other types of layer include:

- **Dropout**
- Noise
- Pooling
- **Normalization (BatchNormalization)**
- Embedding
- Flatten & **Merge**
- And many more...

Activations

More or less all your favourite activations are available:

- Sigmoid, tanh, **ReLu**, softplus, hard sigmoid, linear
- Advanced activations implemented as a layer (after desired neural layer)
- Advanced activations: LeakyReLu, PReLu, ELU, Parametric, Softplus, Thresholded linear and Thresholded Relu

Objectives and Optimizers

Objective Functions:

- Error loss: rmse, **mse**, mae, mape, msle
- Hinge loss: squared hinge, hinge
- Class loss: binary crossentropy, **categorical crossentropy**

Optimization:

- Provides **SGD**, Adagrad, Adadelta, **Rmsprop and Adam**
- All optimizers can be customized via parameters

Parallel Capabilities

- Training time is drastically reduced thanks to Theano’s GPU support
- Theano compiles into CUDA, NVIDIA’s GPU API
- Currently will only work with NVIDIA cards but Theano is working on OpenCL version

- TensorFlow has similar support

Architecture/Weight Saving and Loading

- Model architectures can be saved and loaded

```
# save as JSON
json_string = model.to_json()
# save as YAML
yaml_string = model.to_yaml()

# model reconstruction from JSON:
from keras.models import model_from_json
model = model_from_json(json_string)

# model reconstruction from YAML
model = model_from_yaml(yaml_string)
```

- Model parameters (weights) can be saved and loaded

```
model.save_weights('my_model_weights.h5')
model.load_weights('my_model_weights.h5')
```

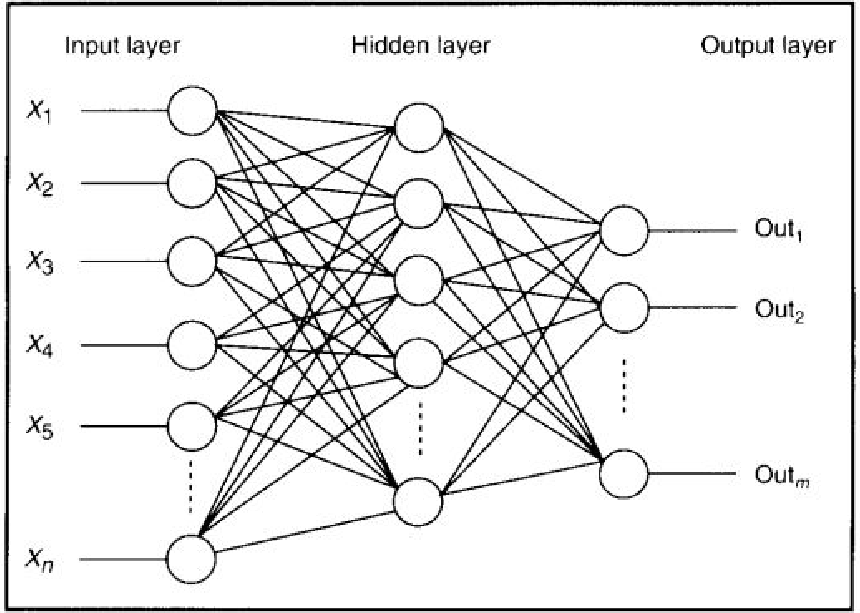
Callbacks

Allow for function call during training

- Callbacks can be called at different points of training (batch or epoch)
- Existing callbacks: Early Stopping, weight saving after epoch, **learning rate**
- Easy to build and implement, called in training function, **fit ()**

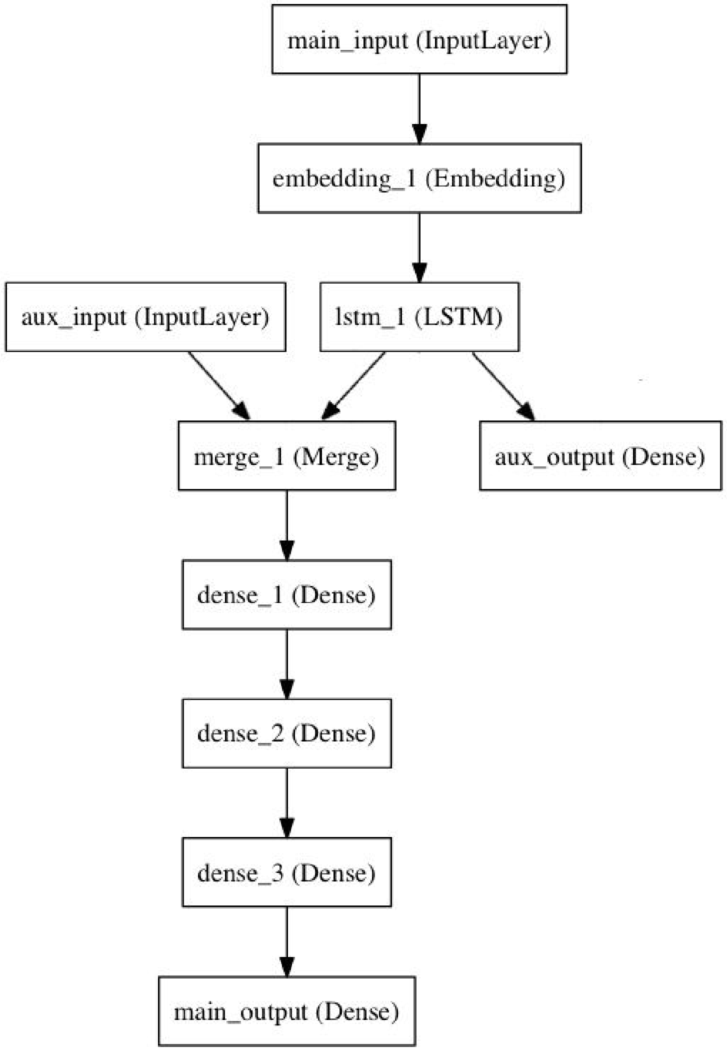
Model Type: Sequential

- Sequential models are linear stack of layers
- The model we all know and love
- Treat each layer as object that feeds into the next



Functional API

- Optimized over all outputs
- Graph model allows for two or more independent networks to diverge or merge
- Allows for multiple separate inputs or outputs
- Different merging layers (sum, concat, elem-wise mult, ave, dot product, cos proximity)



In Summary

Pros:

- Easy to implement
- Lots of choice
- Extendible and customizable
- GPU
- High level
- Active community
- keras.io

Cons:

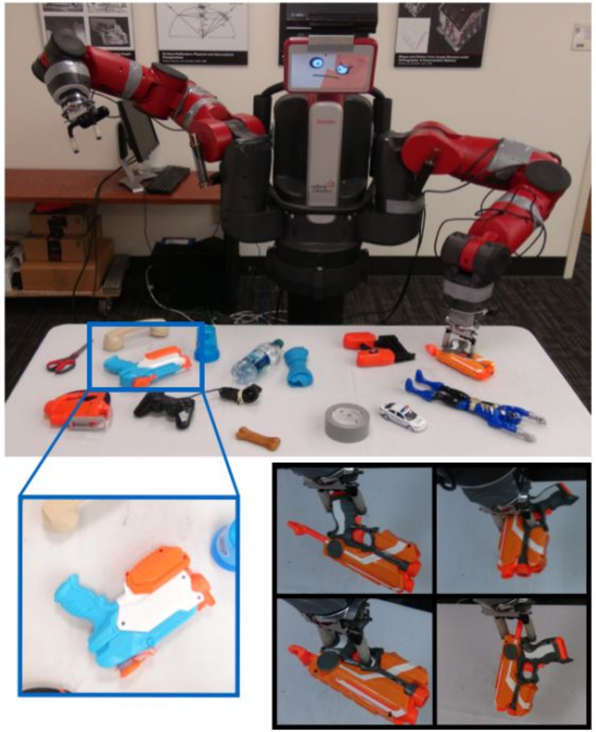
- Lack of generative models
- High level

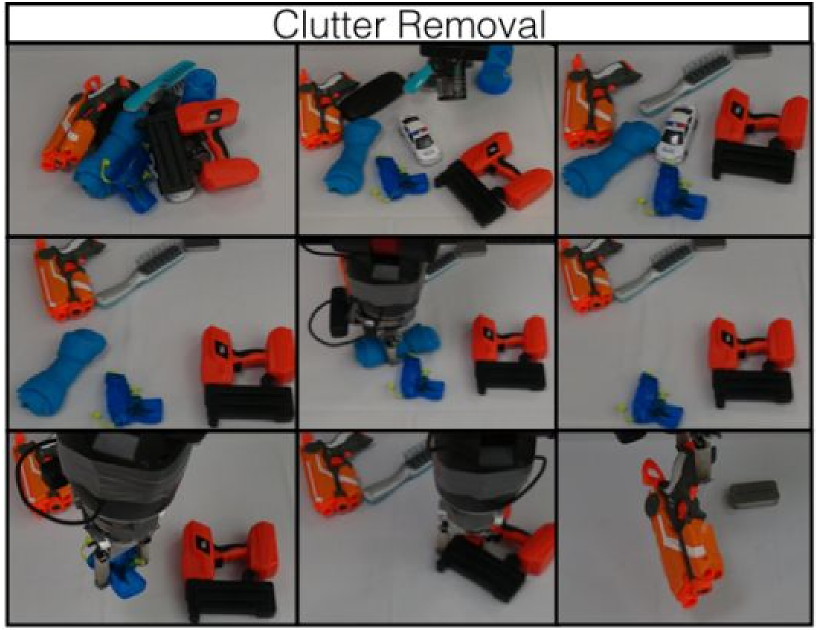
Its Applications to Robotics

- Problem
- Data
- Model

Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours (**ICRA 2016 Best student Paper Award**)

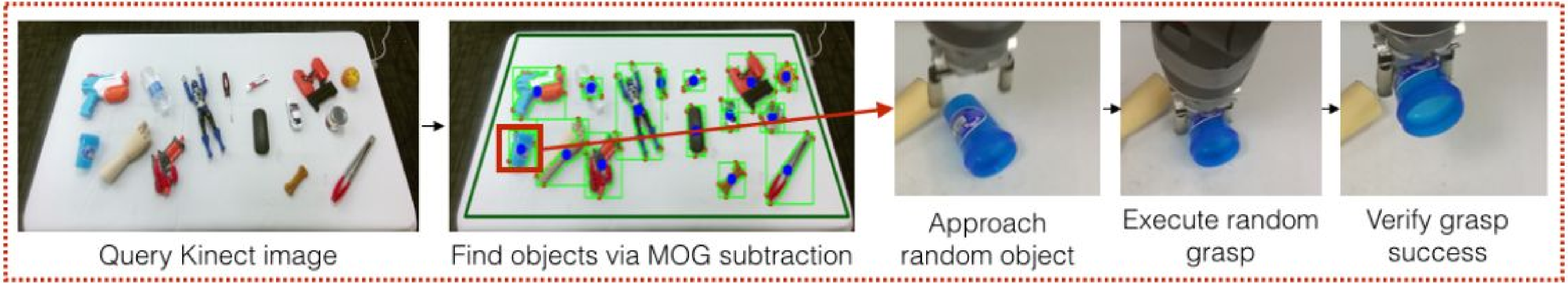
- Problem:



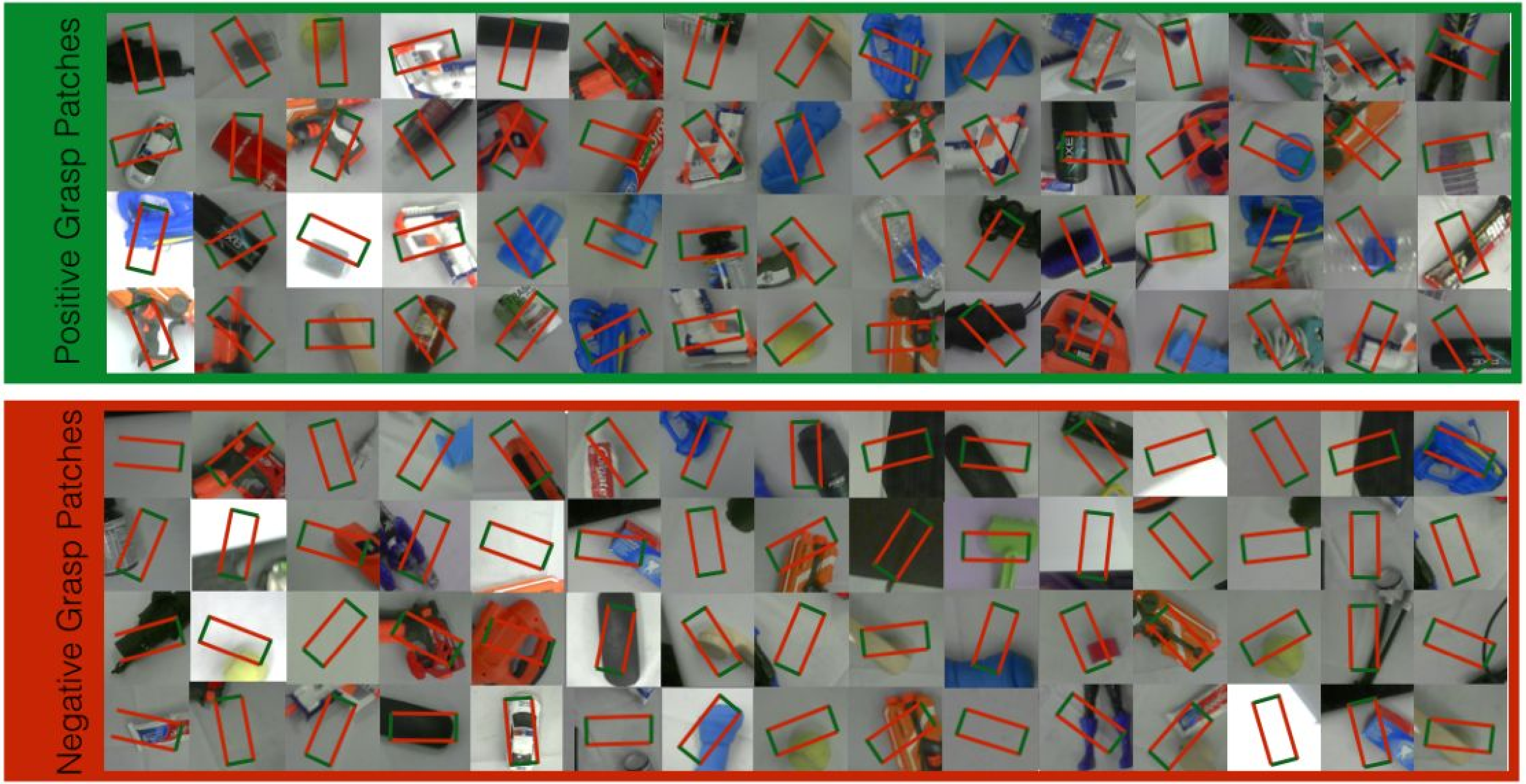



1X

- **Data:**


Query Kinect image
Find objects via MOG subtraction
Approach random object
Execute random grasp
Verify grasp success

- **Data**



- Model

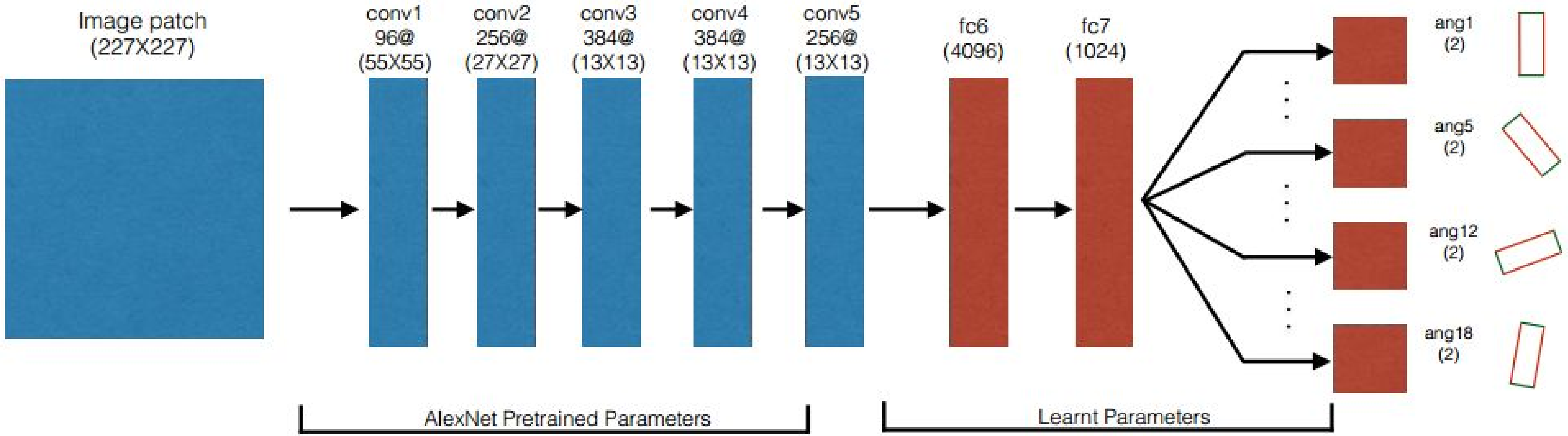


Image patch
(227X227)
conv1
96@
(55X55)
conv2
256@
(27X27)
conv3
384@
(13X13)
conv4
384@
(13X13)
conv5
256@
(13X13)
fc6
(4096)
fc7
(1024)
ang1
(2)
ang5
(2)
ang12
(2)
ang18
(2)
AlexNet Pretrained Parameters
Learnt Parameters

A Machine Learning Approach to **Visual Perception** of Forest Trails for Mobile Robots (RAL 2016)

- **Problem**

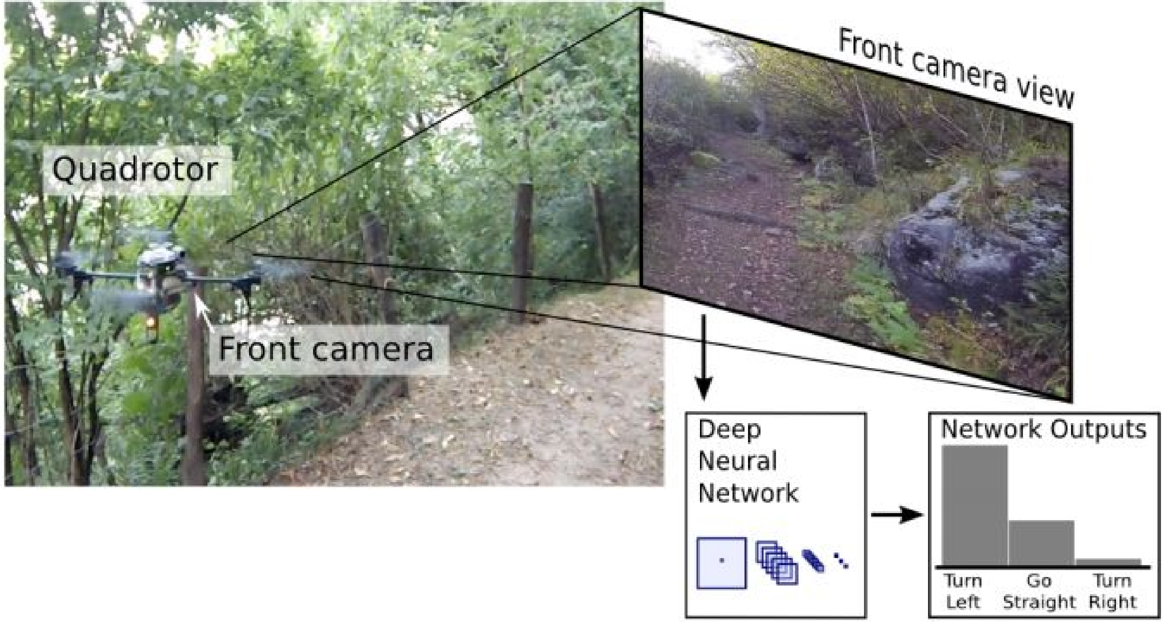
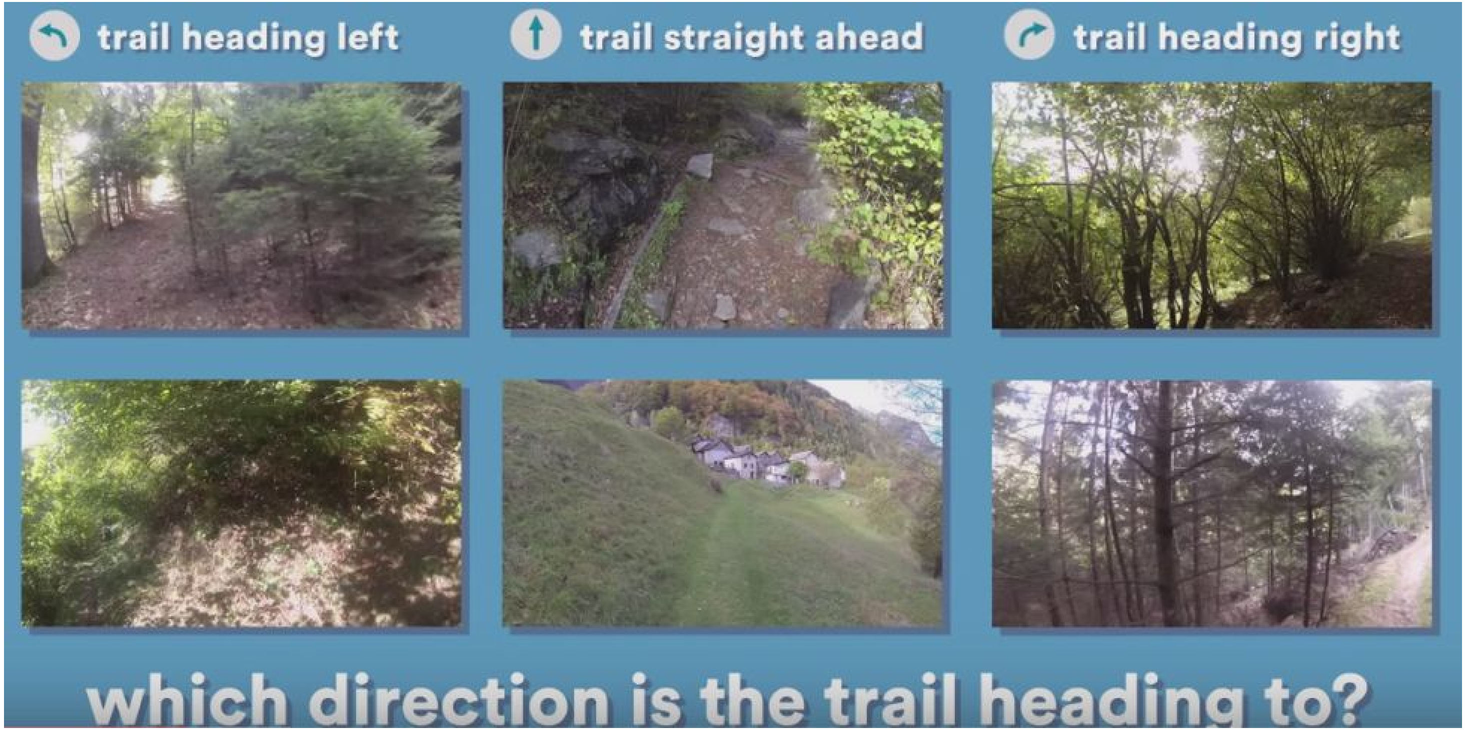
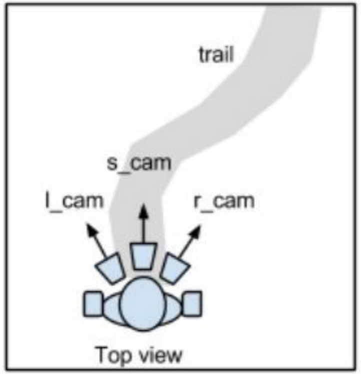


Fig. 1: Our quadrotor acquires the trail images from a forward-looking camera; a Deep Neural Network classifies the images to determine which action will keep the robot on the trail.

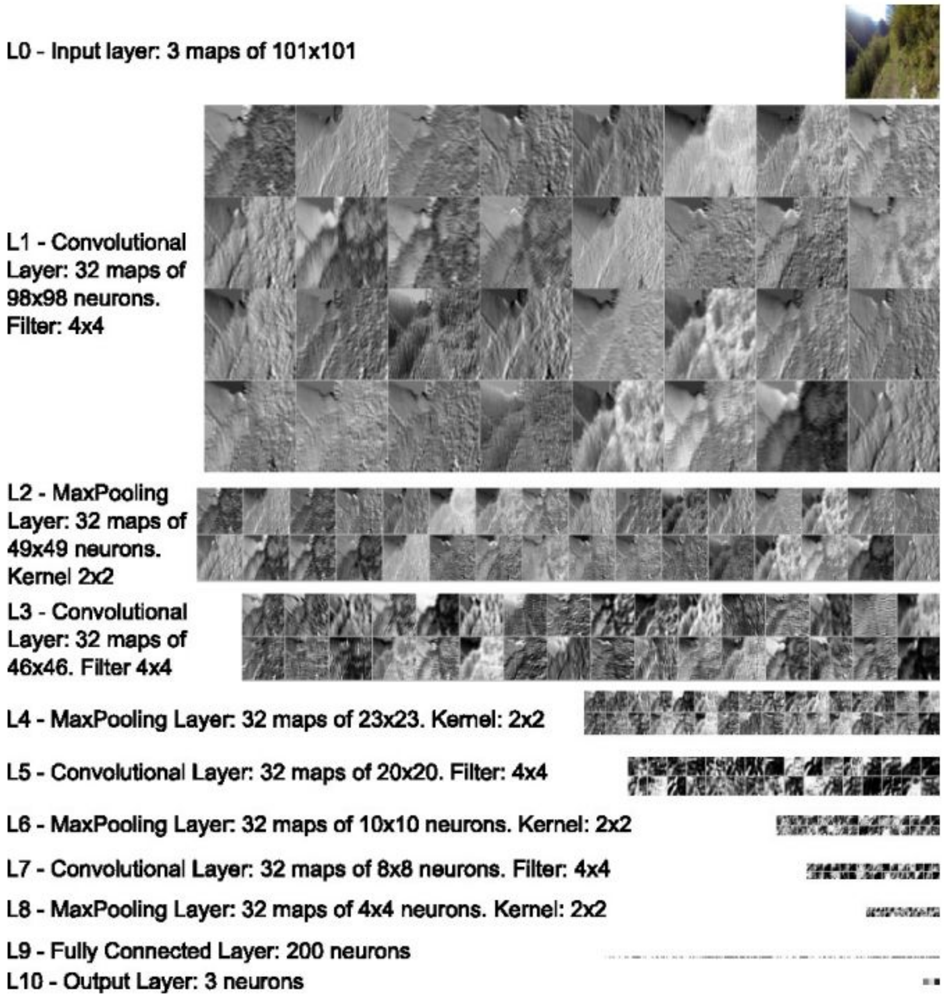
- **Problem**


trail heading left
trail straight ahead
trail heading right
which direction is the trail heading to?

- **Data**

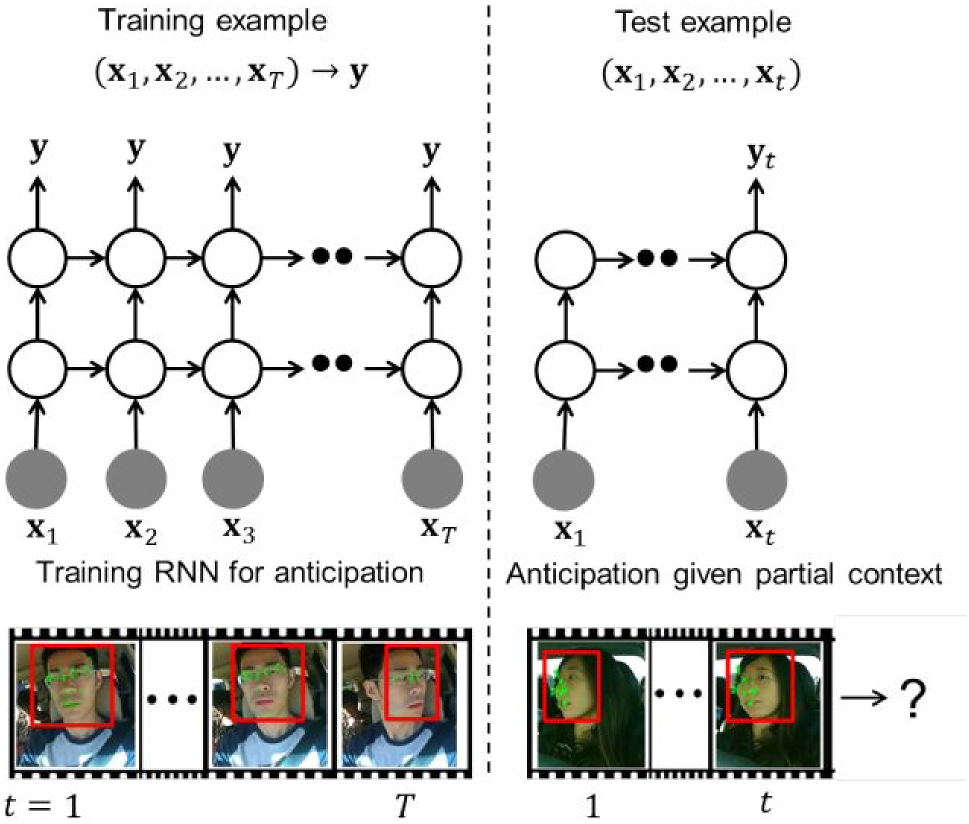


- **Model**

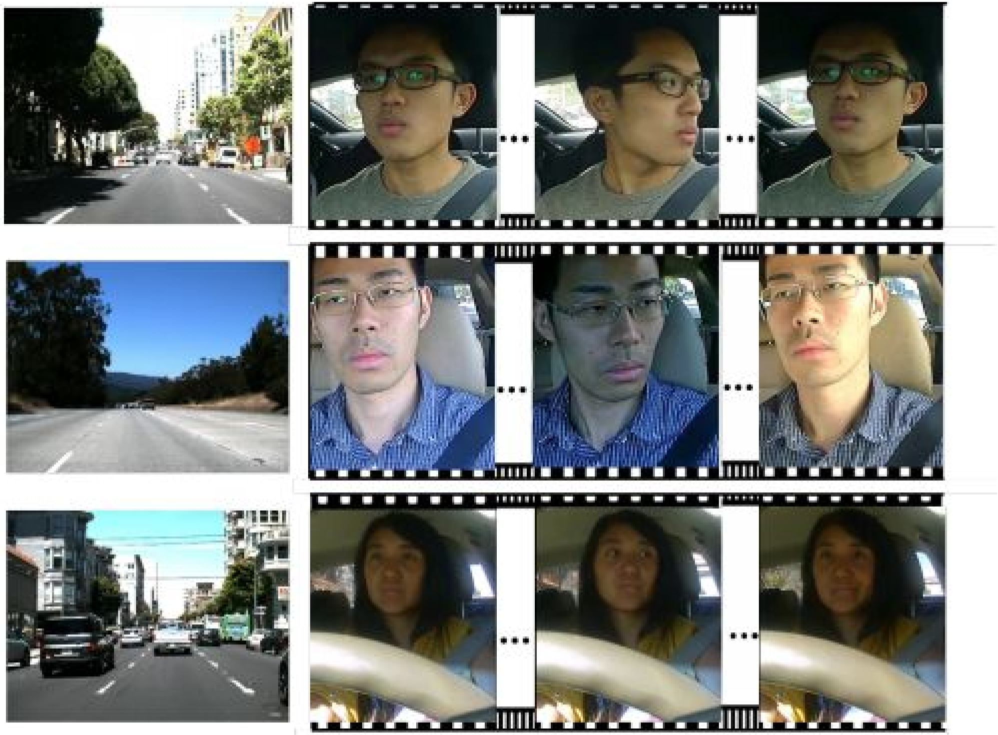

L0 - Input layer: 3 maps of 101x101
L1 - Convolutional Layer: 32 maps of 98x98 neurons. Filter: 4x4
L2 - MaxPooling Layer: 32 maps of 49x49 neurons. Kernel 2x2
L3 - Convolutional Layer: 32 maps of 46x46. Filter 4x4
L4 - MaxPooling Layer: 32 maps of 23x23. Kernel: 2x2
L5 - Convolutional Layer: 32 maps of 20x20. Filter: 4x4
L6 - MaxPooling Layer: 32 maps of 10x10 neurons. Kernel: 2x2
L7 - Convolutional Layer: 32 maps of 8x8 neurons. Filter: 4x4
L8 - MaxPooling Layer: 32 maps of 4x4 neurons. Kernel: 2x2
L9 - Fully Connected Layer: 200 neurons
L10 - Output Layer: 3 neurons

Recurrent Neural Networks for Driver Activity Anticipation via Sensory-Fusion Architecture (ICRA2016)

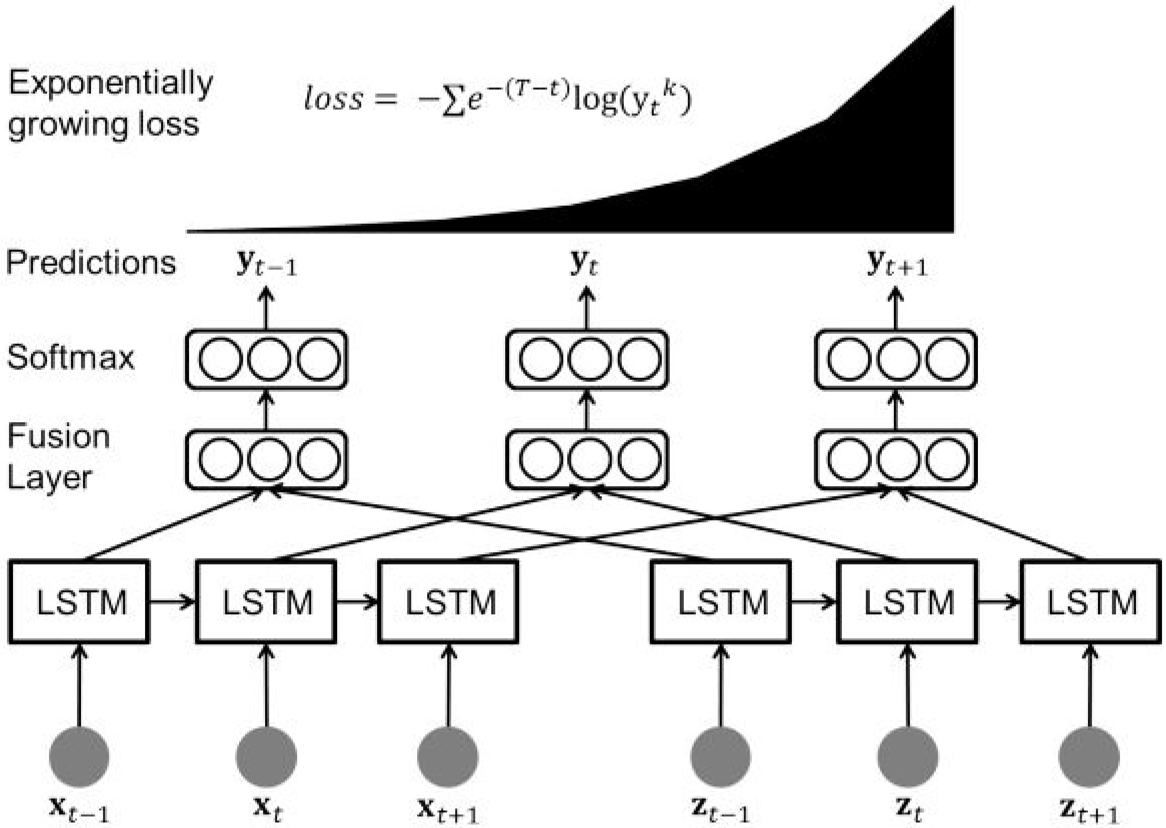
- Problem



- Data



- Model


Exponentially growing loss
$loss = -\sum e^{-(T-t)} \log(y_t{}^k)$
Predictions
$\mathbf{y}_{t-1}$
$\mathbf{y}_t$
$\mathbf{y}_{t+1}$
Softmax
Fusion Layer
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
$\mathbf{x}_{t-1}$
$\mathbf{x}_t$
$\mathbf{x}_{t+1}$
$\mathbf{z}_{t-1}$
$\mathbf{z}_t$
$\mathbf{z}_{t+1}$

Bin-picking Robot Deep Learning