# DEEP LEARNING

## FOR

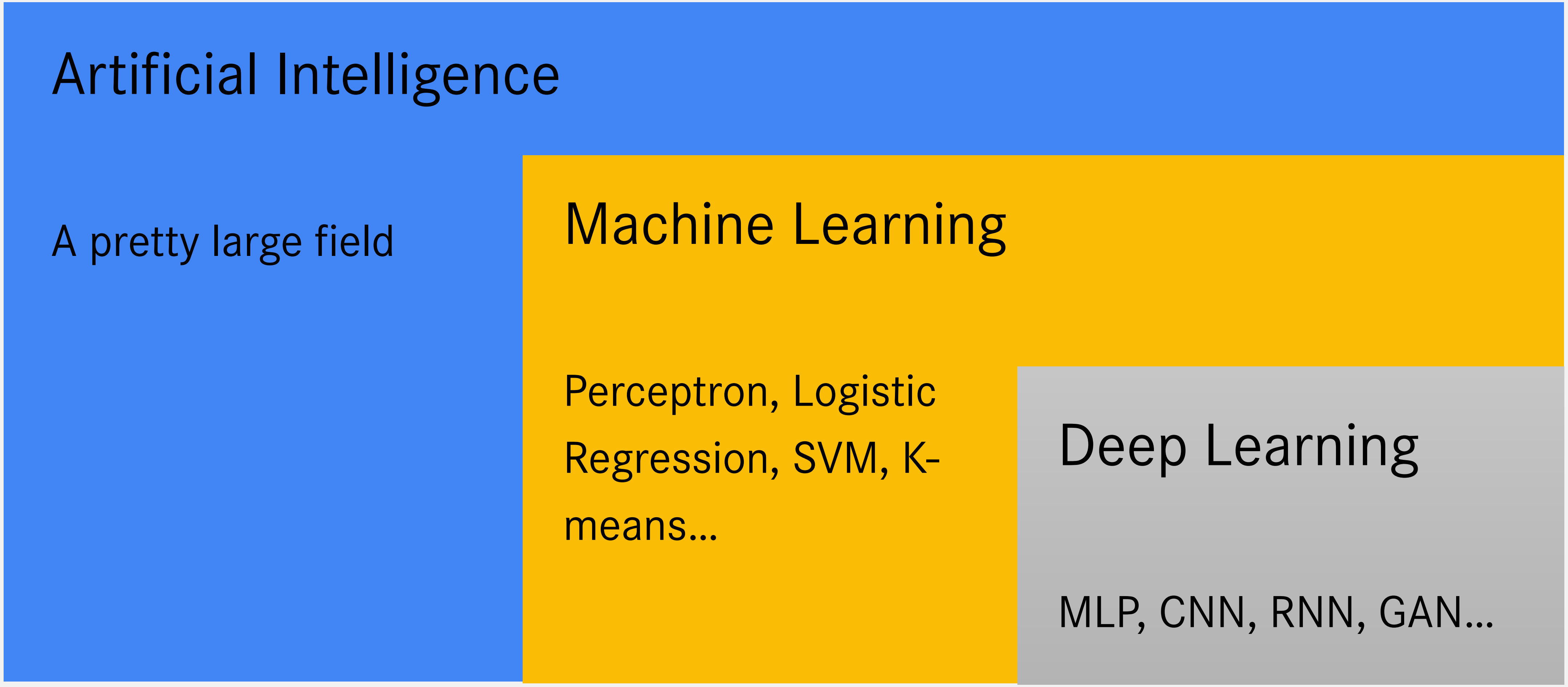
## NATURAL LANGUAGE PROCESSING

Junyang LIN

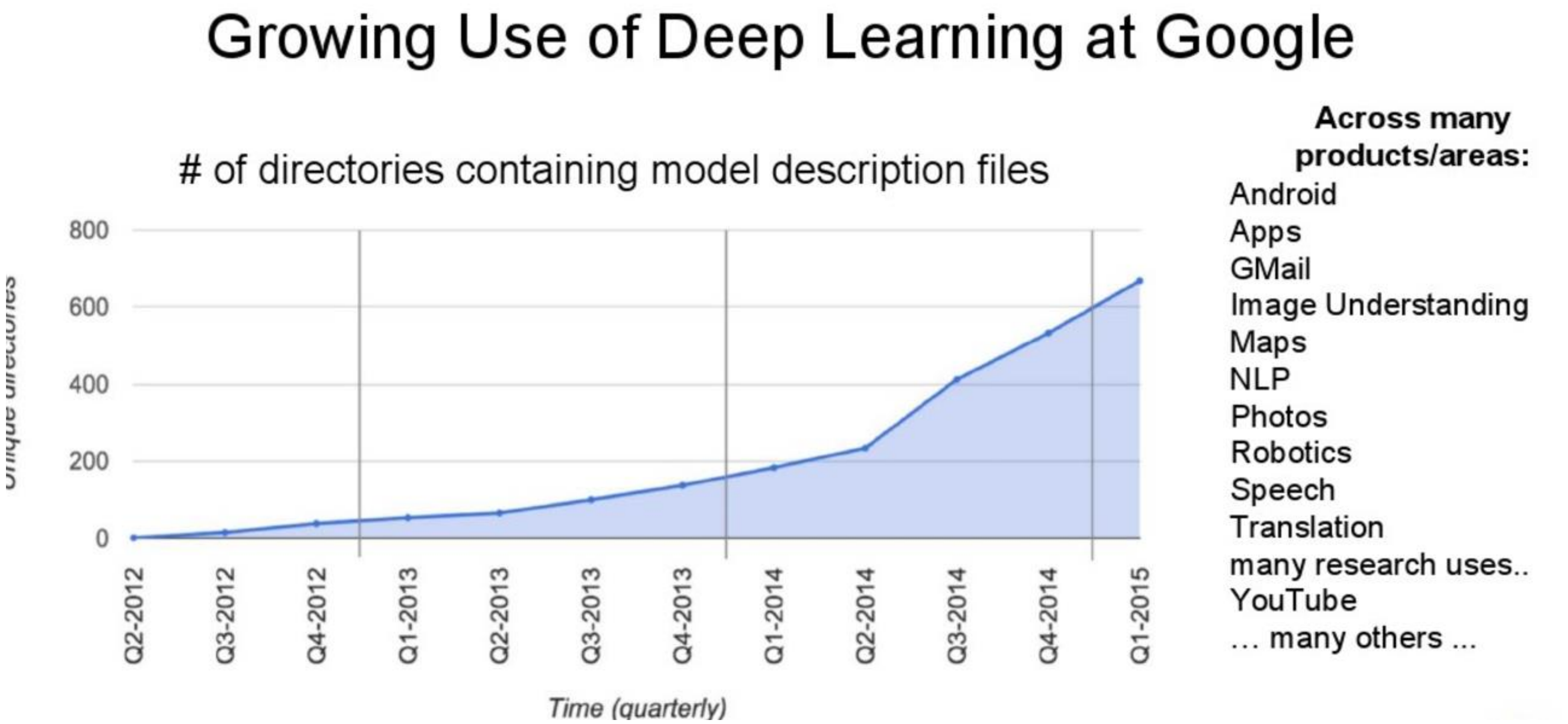
linjunyang@pku.edu.cn

https://justinlin610.github.io

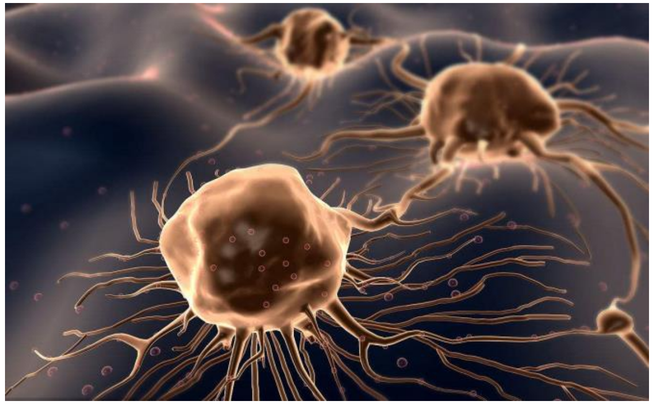
# Deep Learning: A Sub-field of Machine Learning

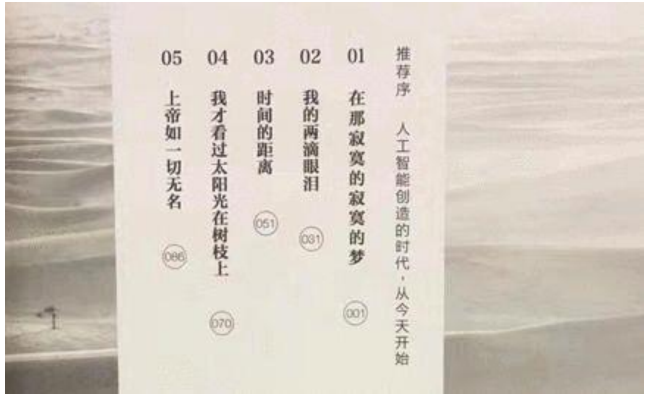


# Deep Learning is Becoming Popular



# Deep Learning is Powerful





推荐序 人工智能创造的时代，从今天开始

01 在那寂寞的寂寞的梦 001

02 我的两滴眼泪 031

03 时间的距离 051

04 我才看过太阳光在树枝上 070

05 上帝如一切无名 086

# Deep Learning is Powerful (Machine Translation)

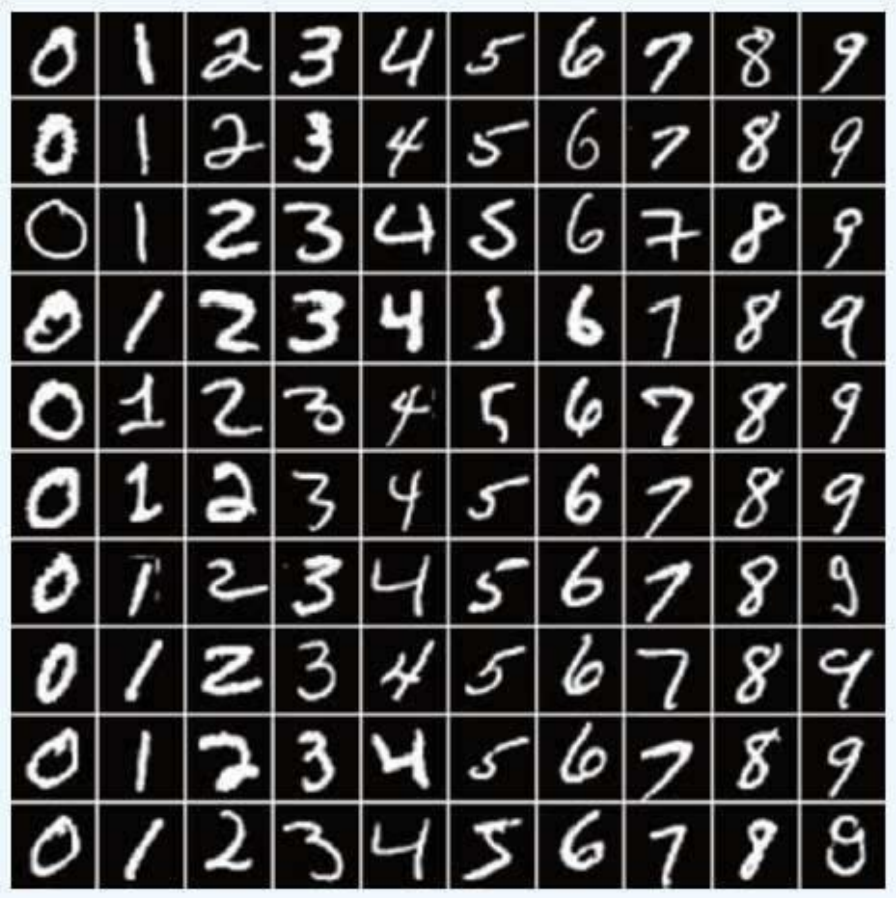
```
INFO:tensorflow:Restoring parameters from checkpoints/dev
Input
  Word Ids:      [185, 171, 4, 89, 136, 68, 114]
  English Words: ['he', 'saw', 'a', 'old', 'yellow', 'truck', '.']

Prediction
  Word Ids:      [209, 4, 130, 320, 229, 168, 308, 312, 1]
  French Words: il a vu un vieux camion jaune . <EOS>
```

# Deep Learning is Powerful (Summarization)

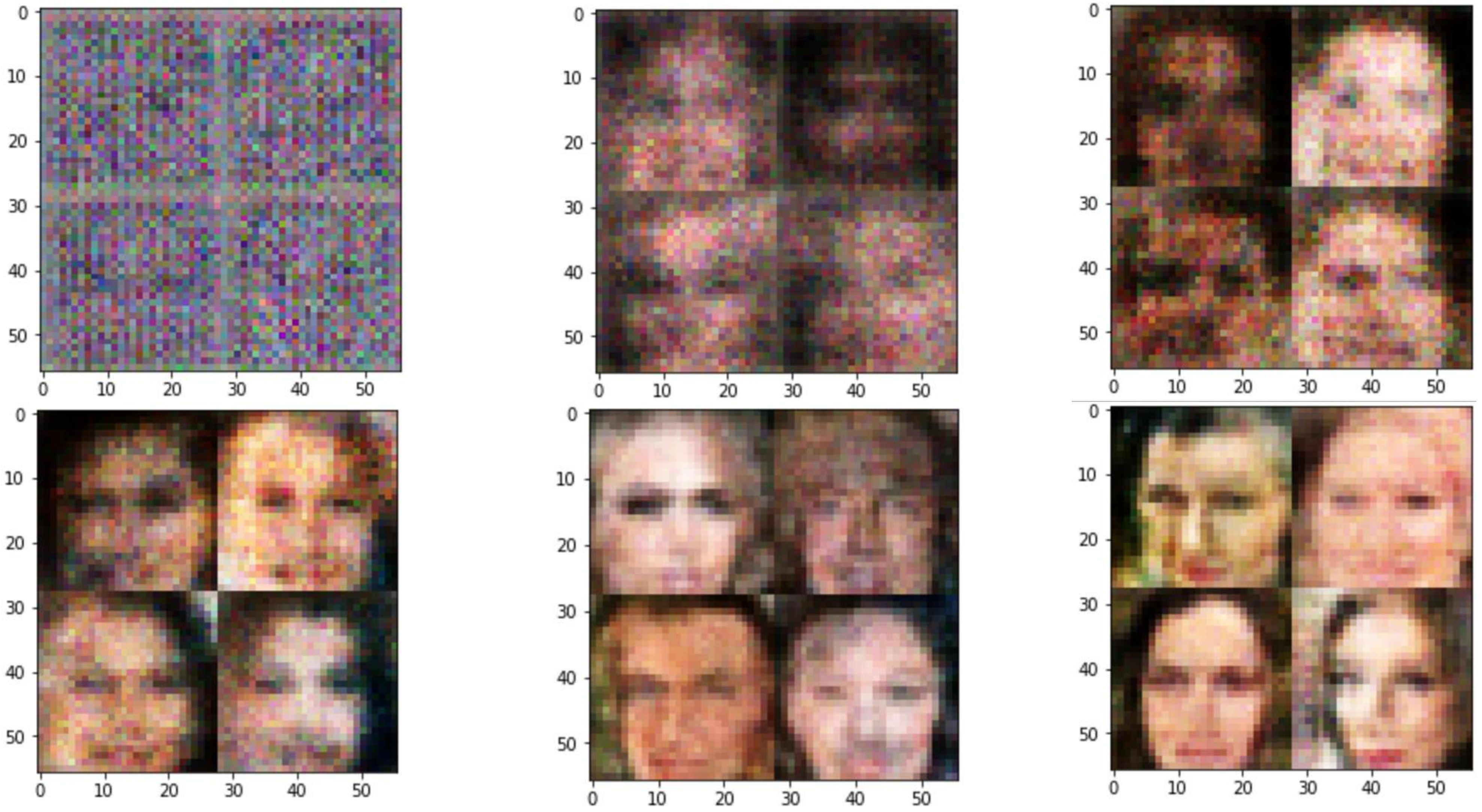
| | |
|---|---|
| Text | 仔细一算，上海的互联网公司不乏成功案例，但最终成为BAT一类巨头的几乎没有, 这也能解释为何纳税百强的榜单中鲜少互联网公司的身影。有一类是被并购，比如：易趣、土豆网、PPS、PPTV、一号店等；有一类是数年偏安于细分市场。<br>With careful calculation, there are many successful Internet companies in Shanghai, but few of them becomes giant company like BAT. This is also the reason why few Internet companies are listed in top hundred companies of paying tax. Some of them are merged, such as Ebay, Tudou, PPS, PPTV, Yihaodian and so on. Others are satisfied with segment market for years. |
| Reference | 为什么上海出不了互联网巨头?<br>Why Shanghai comes out no giant company? |
| Seq2seq-A | 上海的互联网巨头。<br>Shanghai's giant company. |
| SRB | 上海鲜少互联网巨头的身影。<br>Shanghai has few giant companies. |

https://arxiv.org/abs/1706.02459

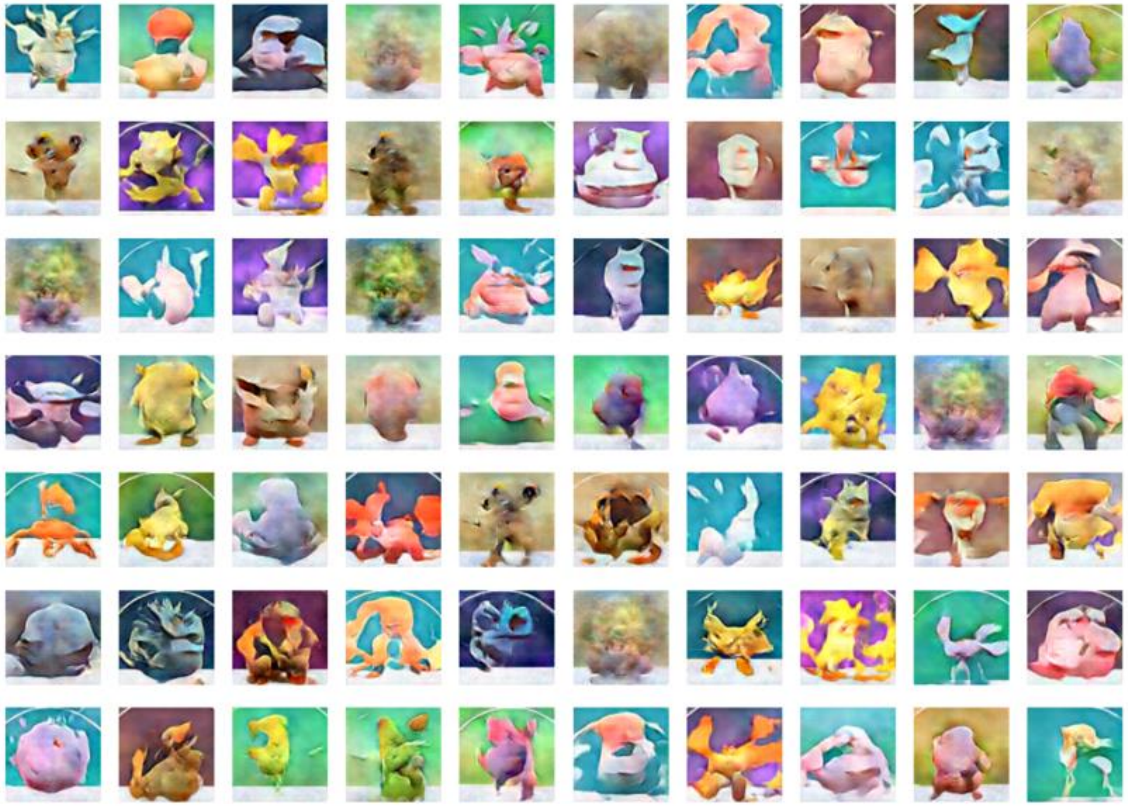
# Deep Learning is Powerful (Object Recognition)



# Deep Learning is Powerful (Face Generation)



# Deep Learning is Powerful (Pokemon Generation)



# Deep Learning is Powerful (Cat Generation)

# Deep Learning is Powerful (Cat Generation)

# History of Deep Learning

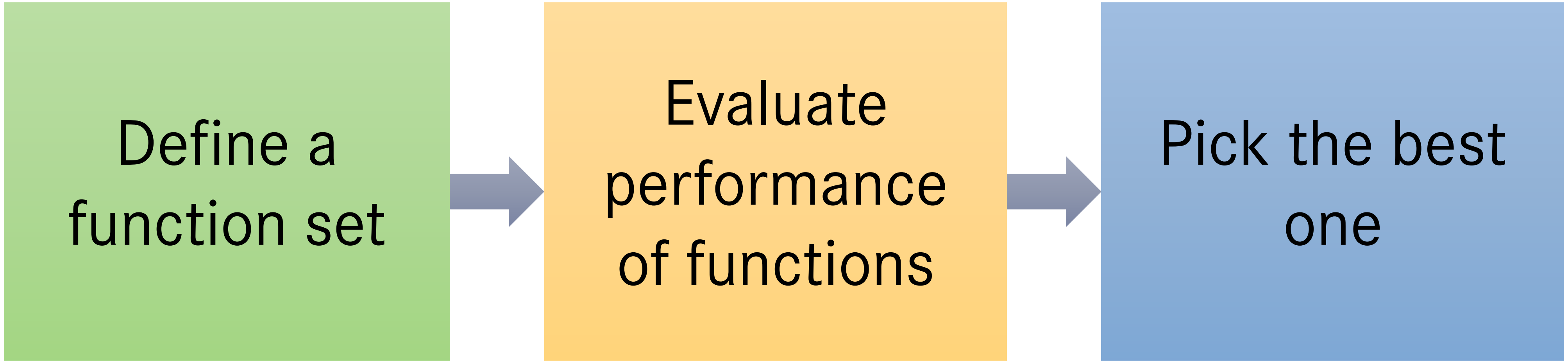
# Ups and Downs

- 1958: Perceptron Learning Algorithm (Rosenblatt, linear model, limited)
- 1980s: Multi- layer Perceptron (MLP, non-linear, not fancy)
- 1986: Backpropagation (G. Hinton et al., but not efficient when deep)
- 1990s: SVM vs Neural Network (Yann LeCun, CNN)
- 2006: RBM Initialization (G. Hinton et al., Breakthrough)
- 2009: GPU
- 2011: Started to be popular in Speech Recognition
- 2012: AlexNet won ILSVRC (Deep Learning Era started)
- 2014: Started to become very popular in NLP (Y. Bengio, RNN…)

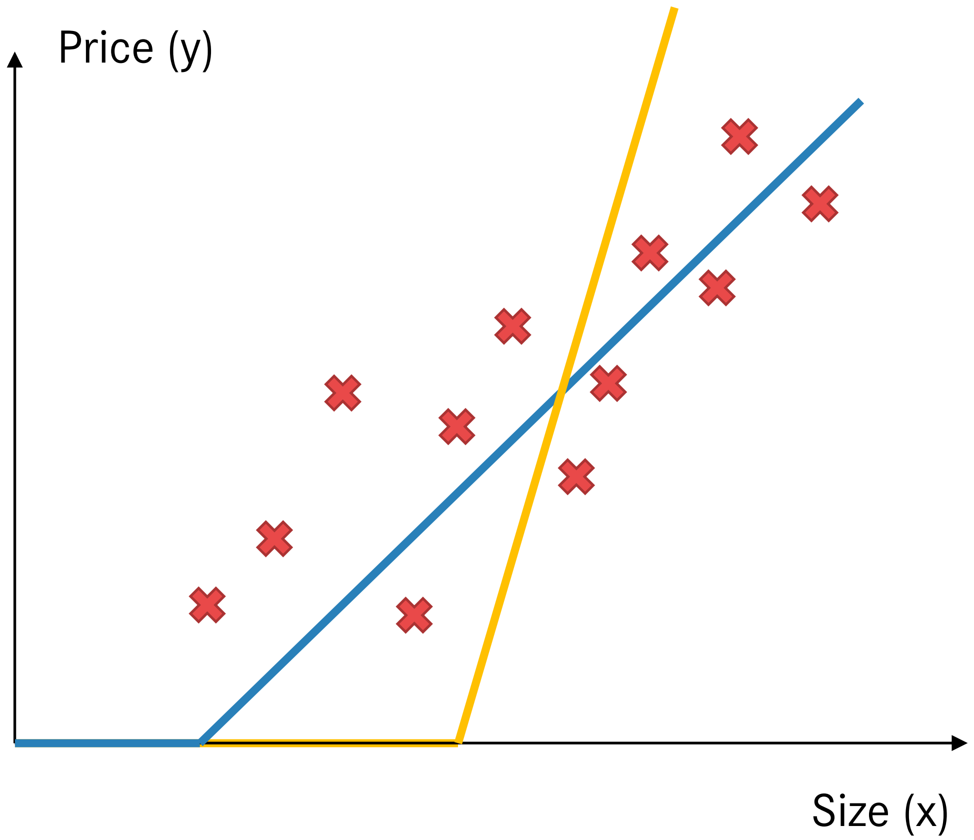
# Great Figures

# What is Deep Learning?

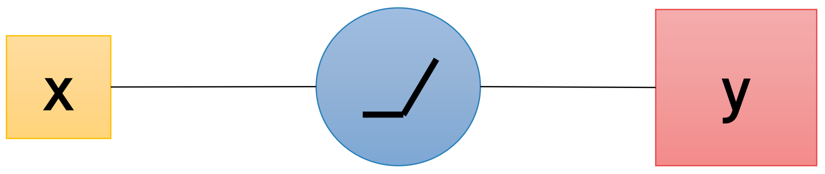
# The Essence of Machine Learning



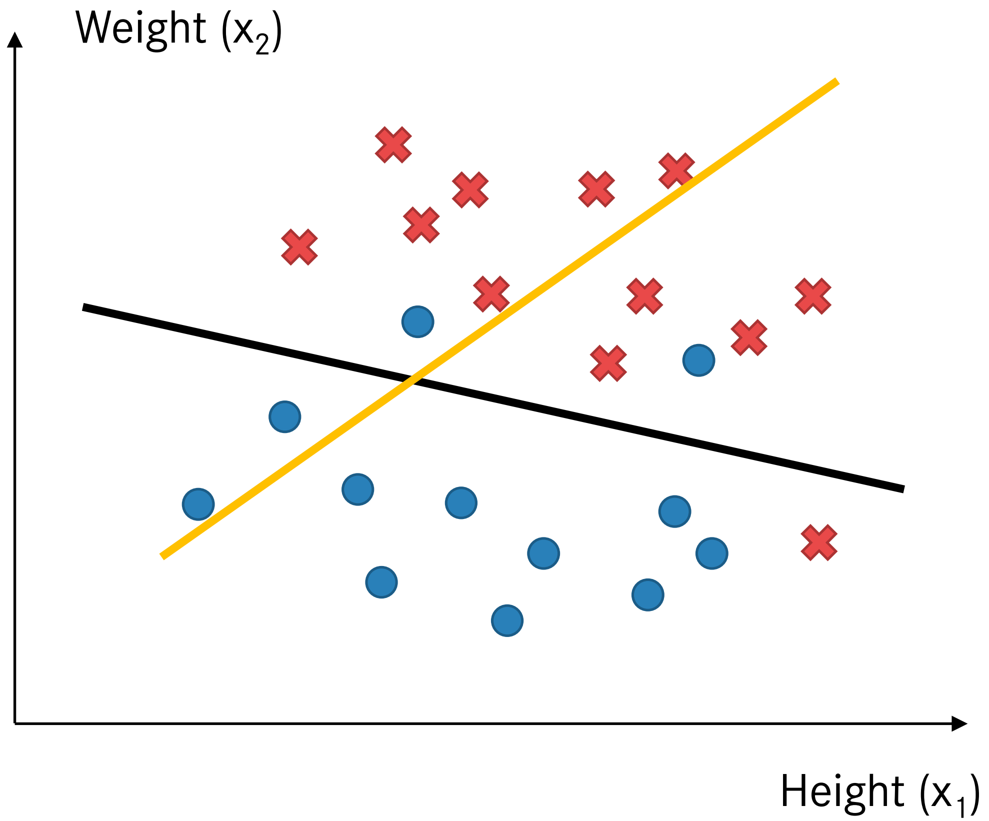
# Linear Regression (Housing Price)



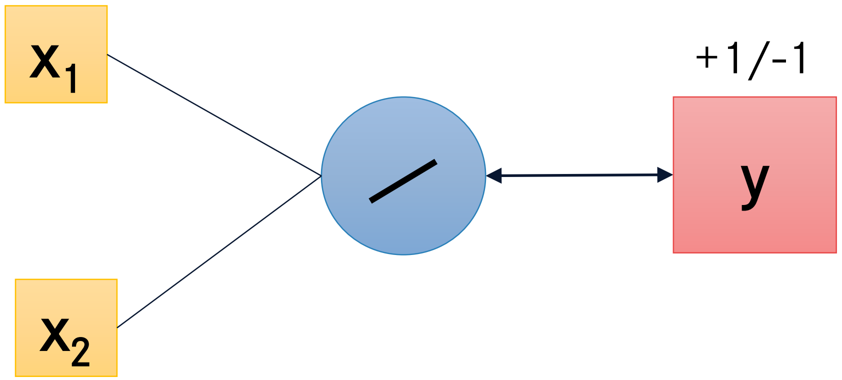

$$z = wx + b$$
$$f(z) = \max(0, z)$$



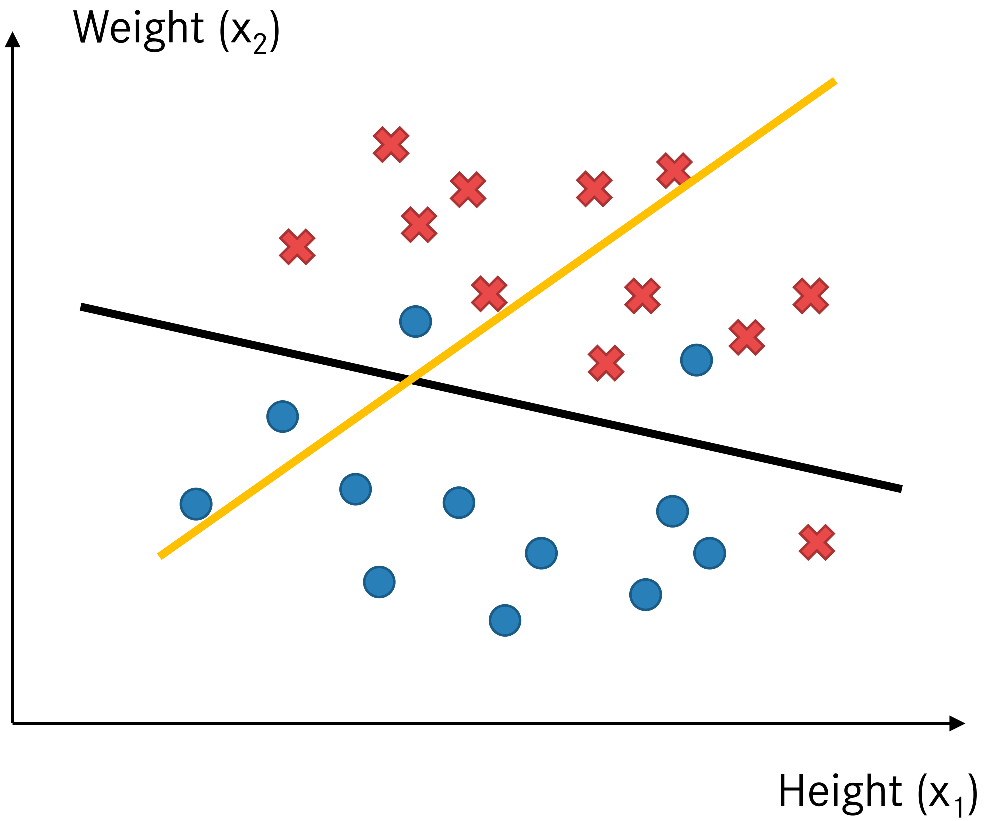
# Perceptron




$$z = w_1x_1 + w_2x_2 + b = w^Tx + b$$
$$f(z) = sign(z)$$

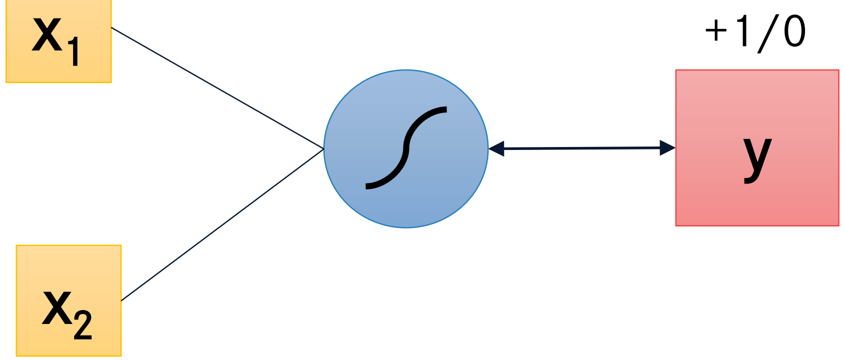


# Logistic Regression

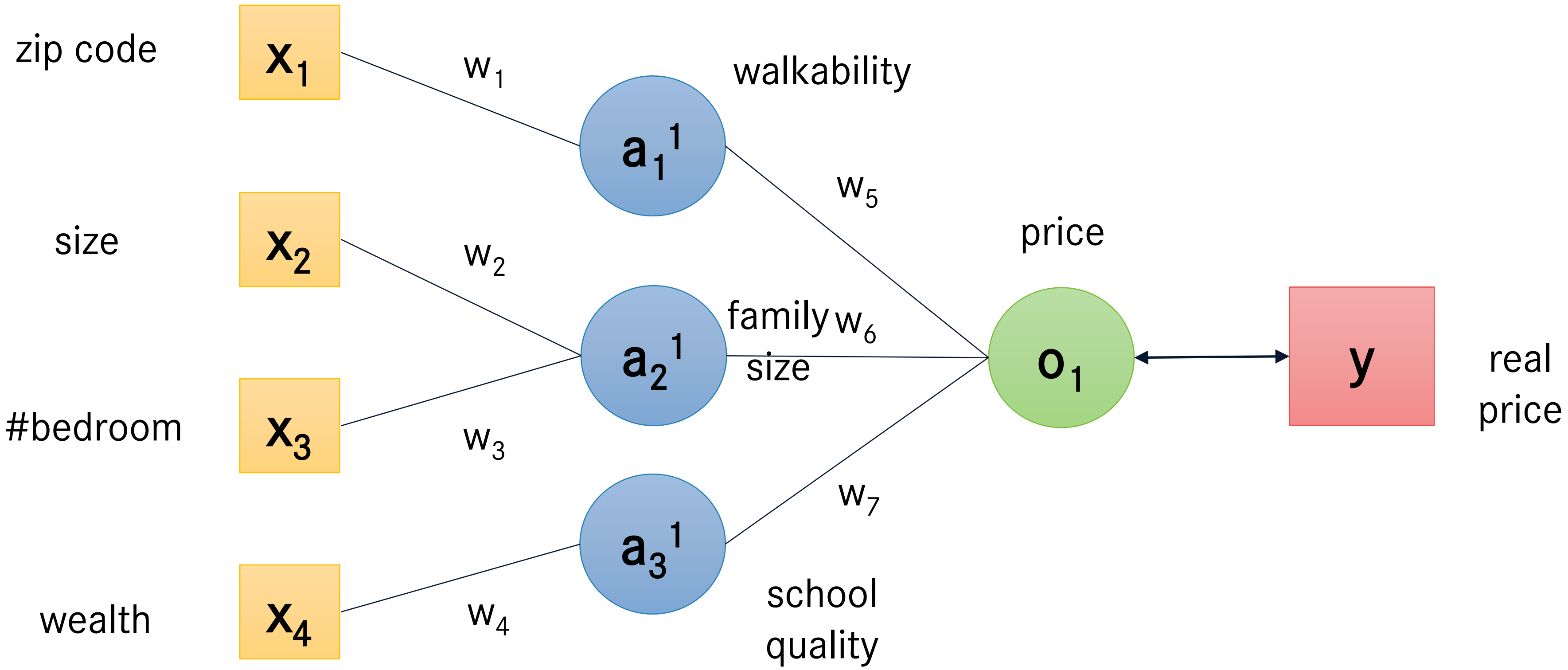



$$z = w^T x + b$$

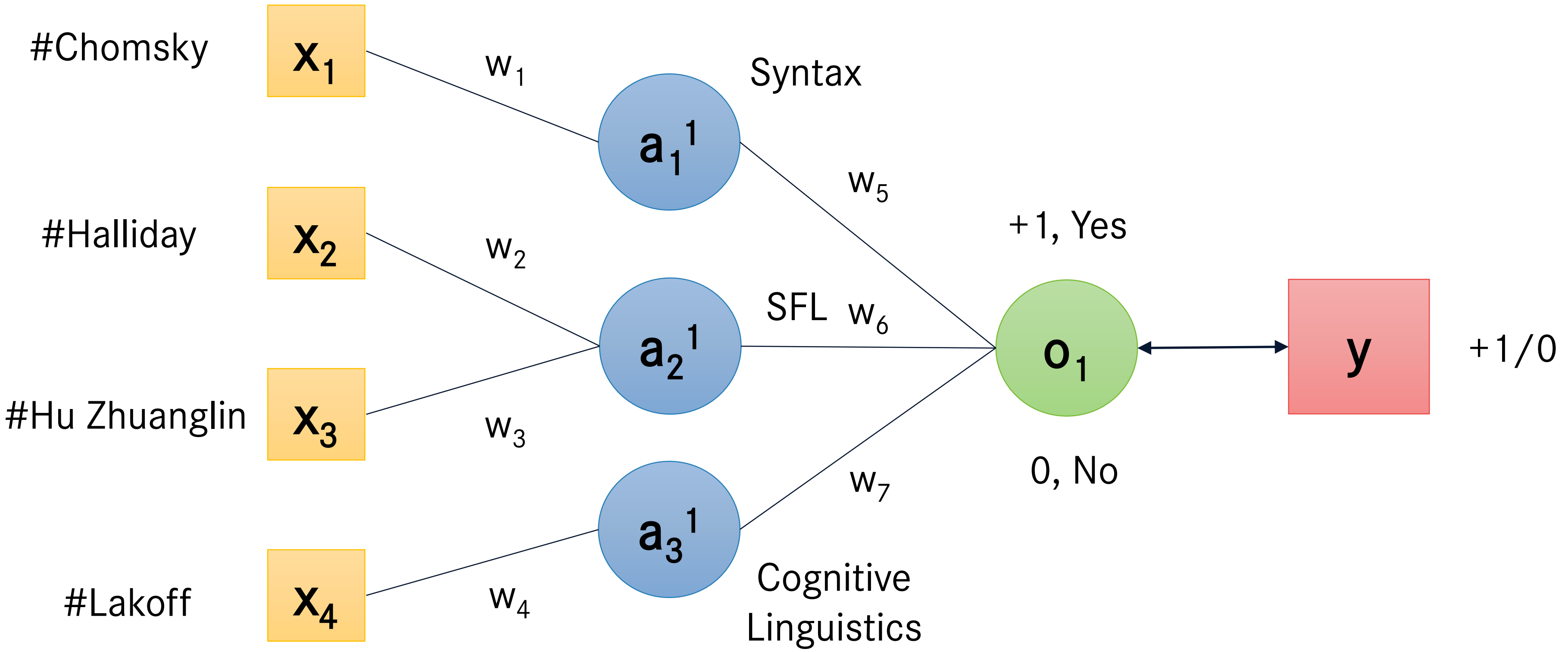
$$g(x) = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (g(z) \in (0,1))$$



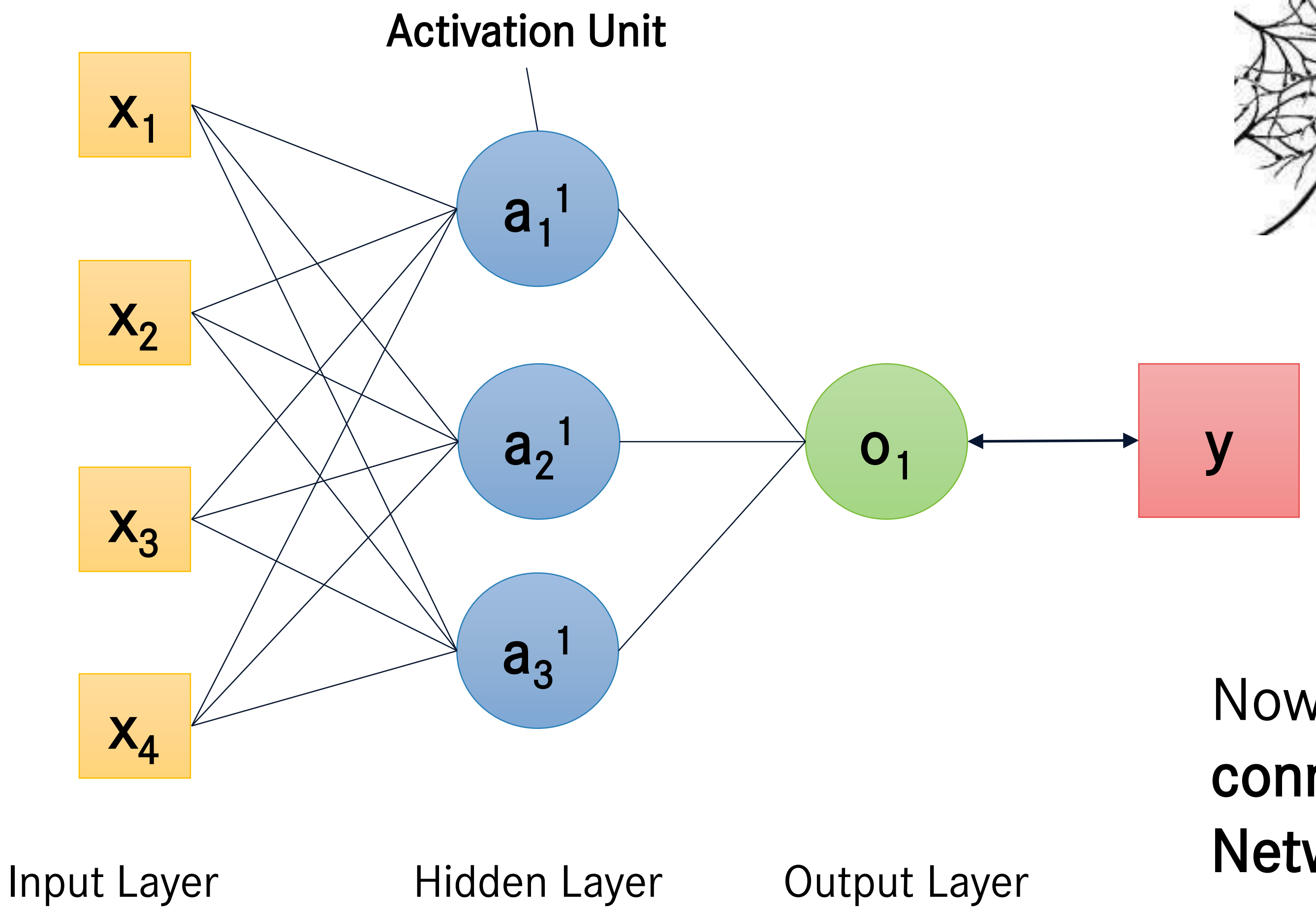
# Housing Price Prediction

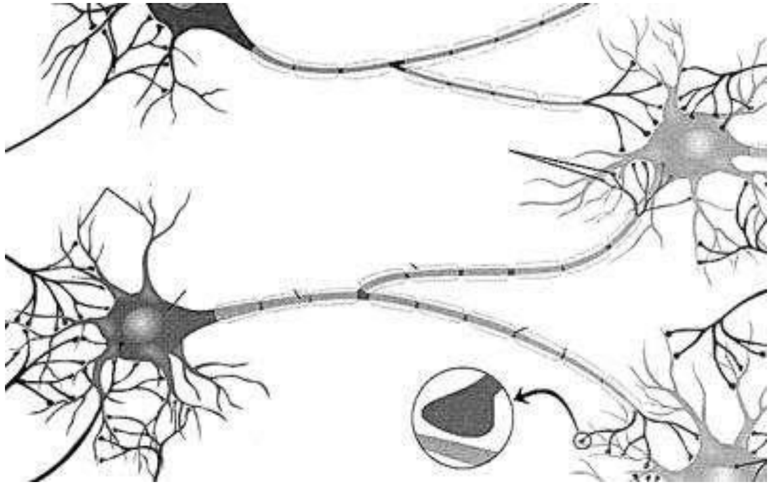


# Should you study linguistics?



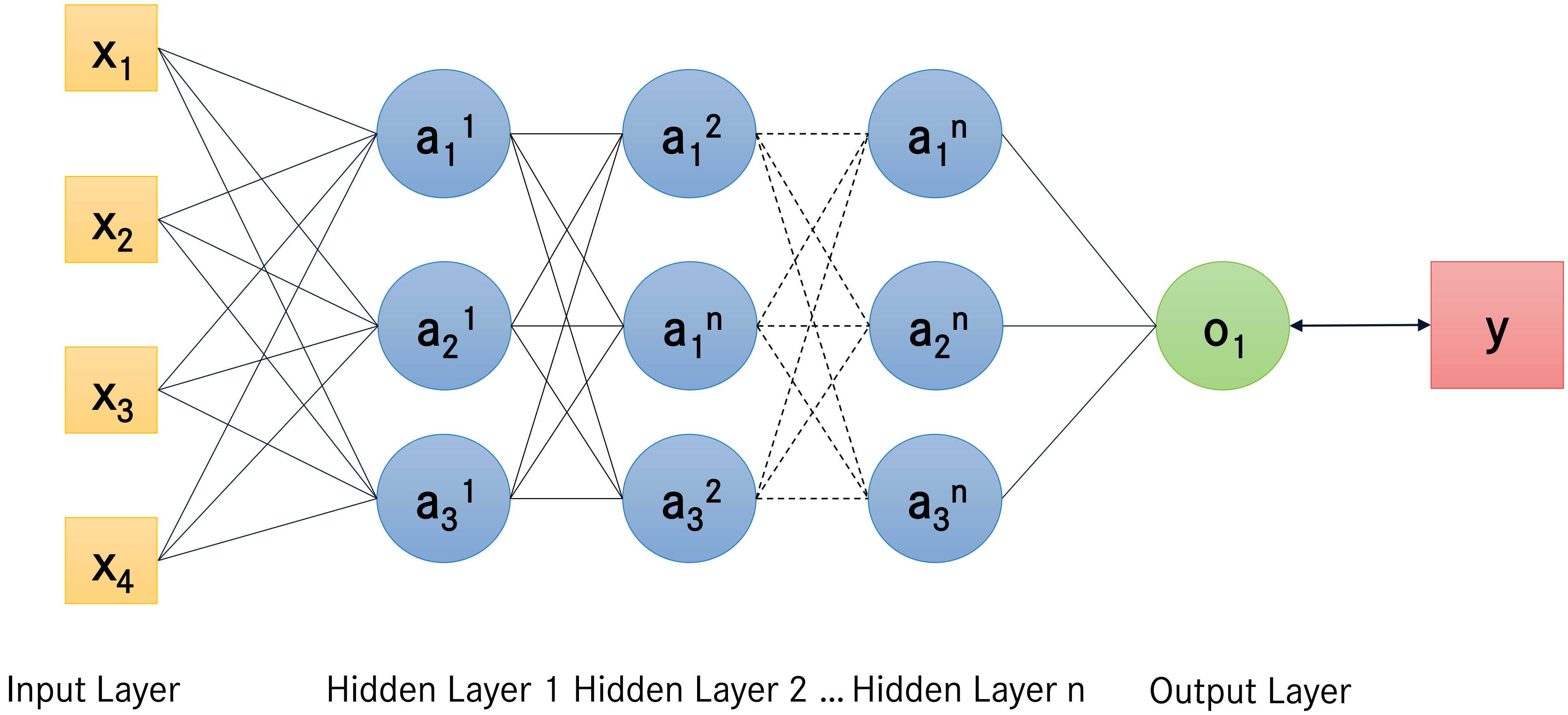
# Standard NN (MLP)



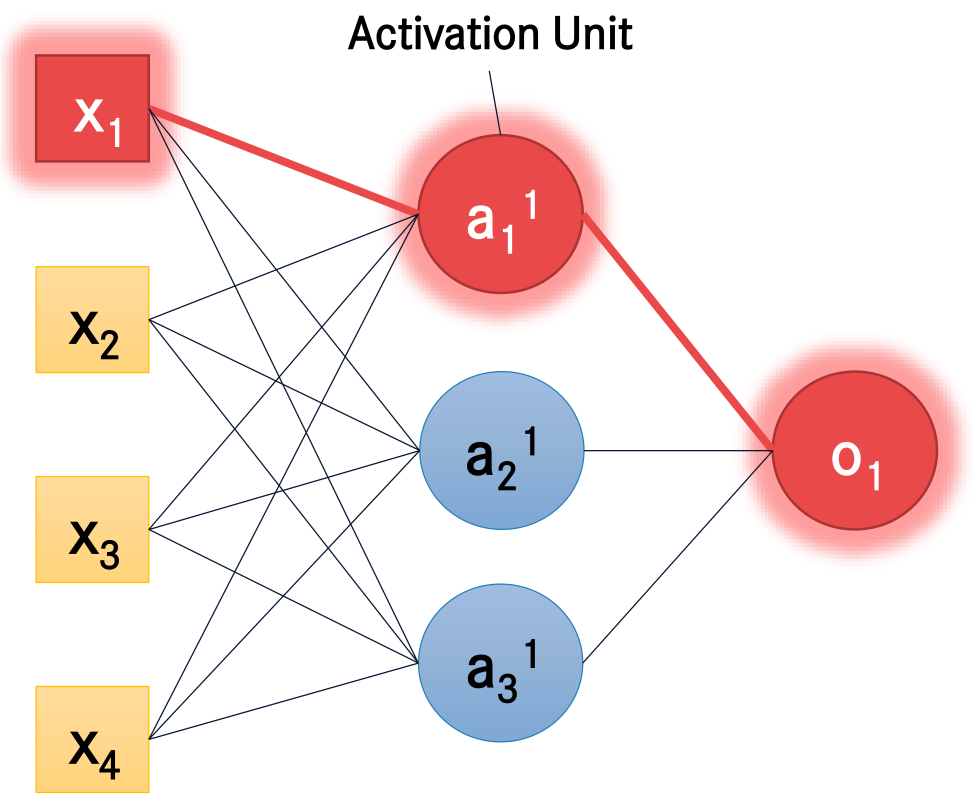



Now we have defined a **Fully-connected Feedforward Neural Network**, in fact, a function set.

# “Deep” means many hidden layers



# Activation Unit

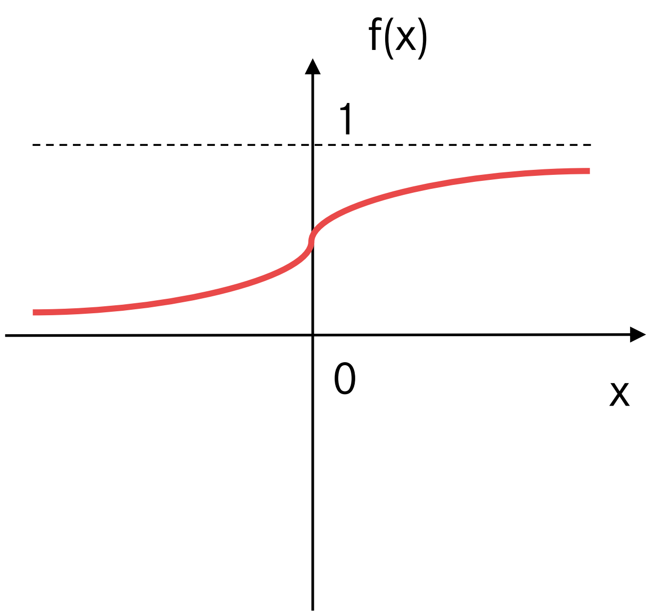



If there is no operation in the activation unit, the whole model will be a linear model.

Therefore, the effects of multi-layer NN will be equivalent to those of single-layer NN. This is why we need non-linear activation function in the activation unit.

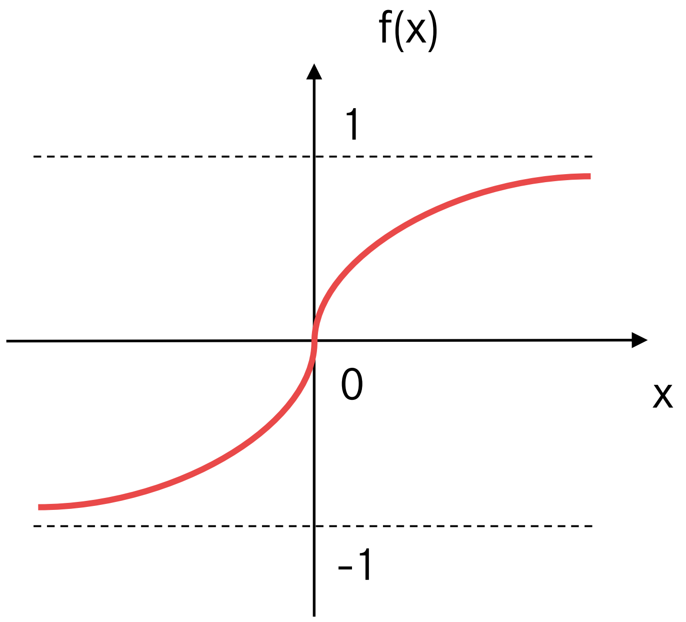
# Activation Function

Preferable



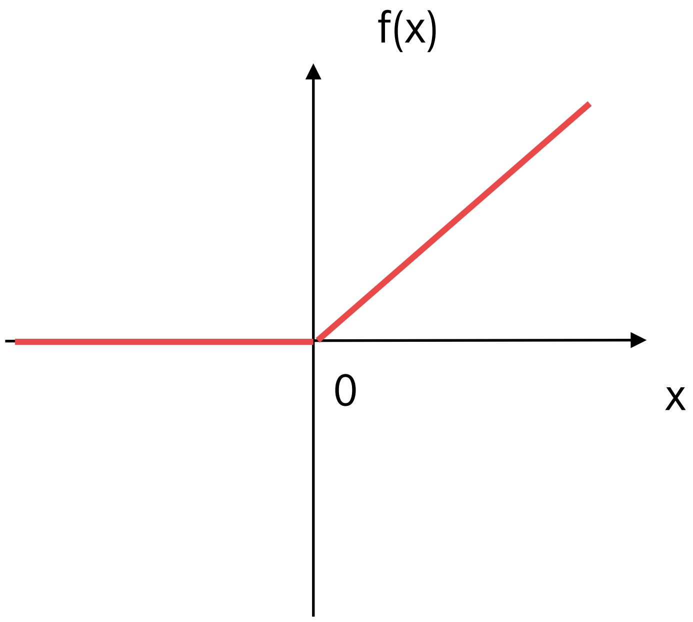

*Sigmoid function*

$$f(x) = \sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}$$




*Tanh function*

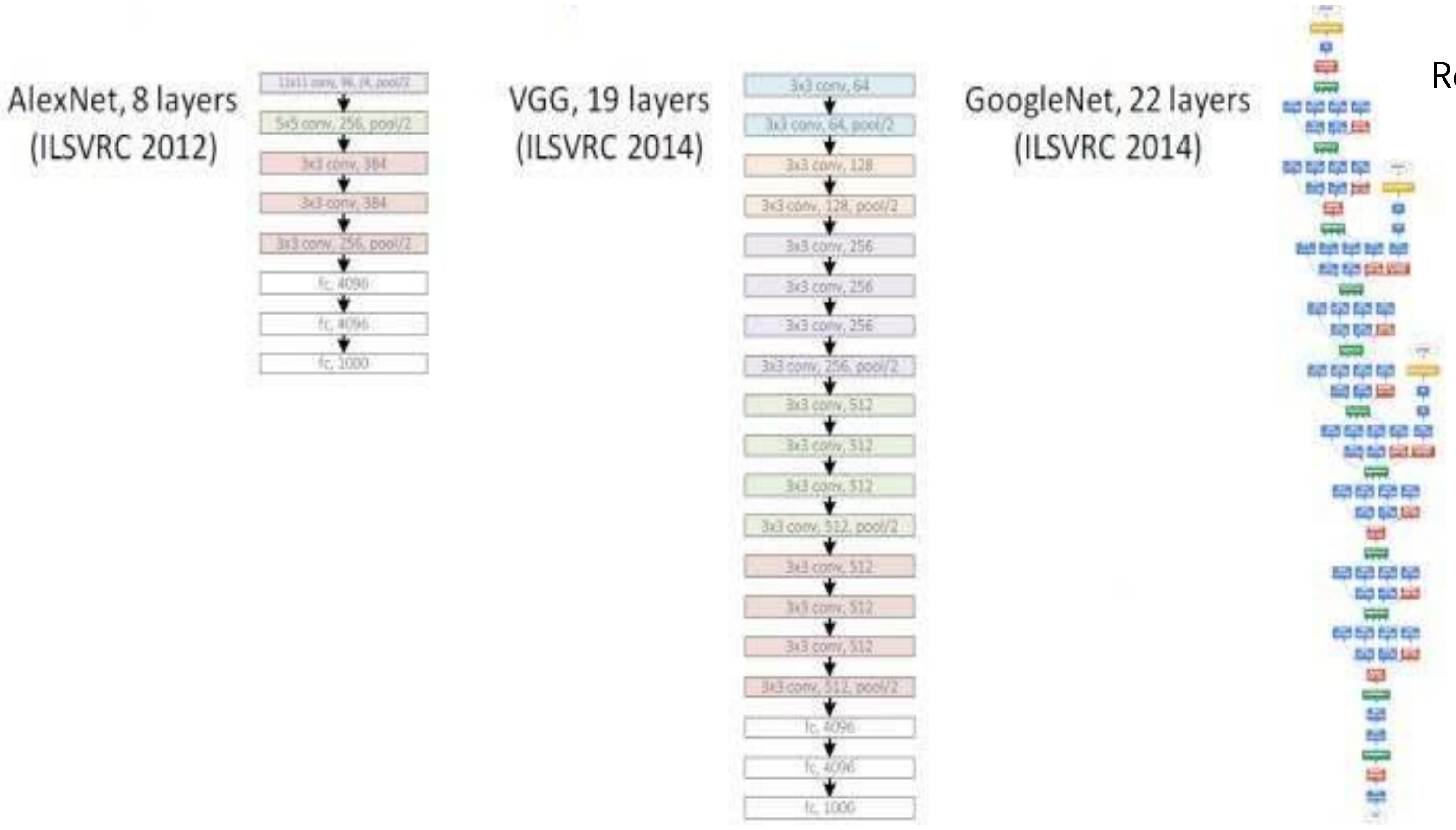
$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



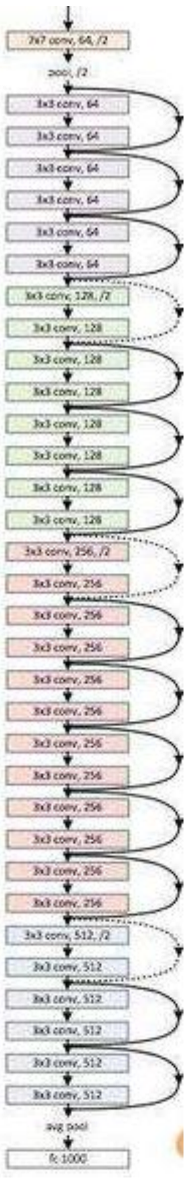

*ReLU function*

$$f(x) = ReLU(x) = \max(0, x)$$

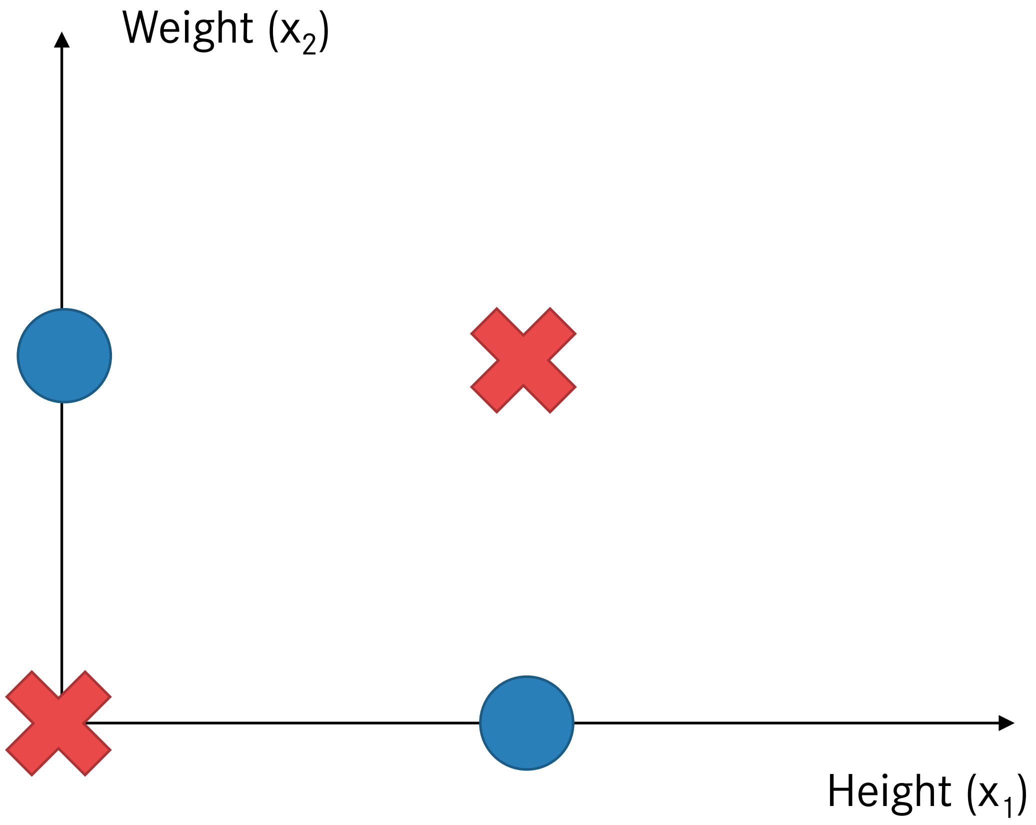
# “Deep” means many hidden layers




ResNet, 152 layers



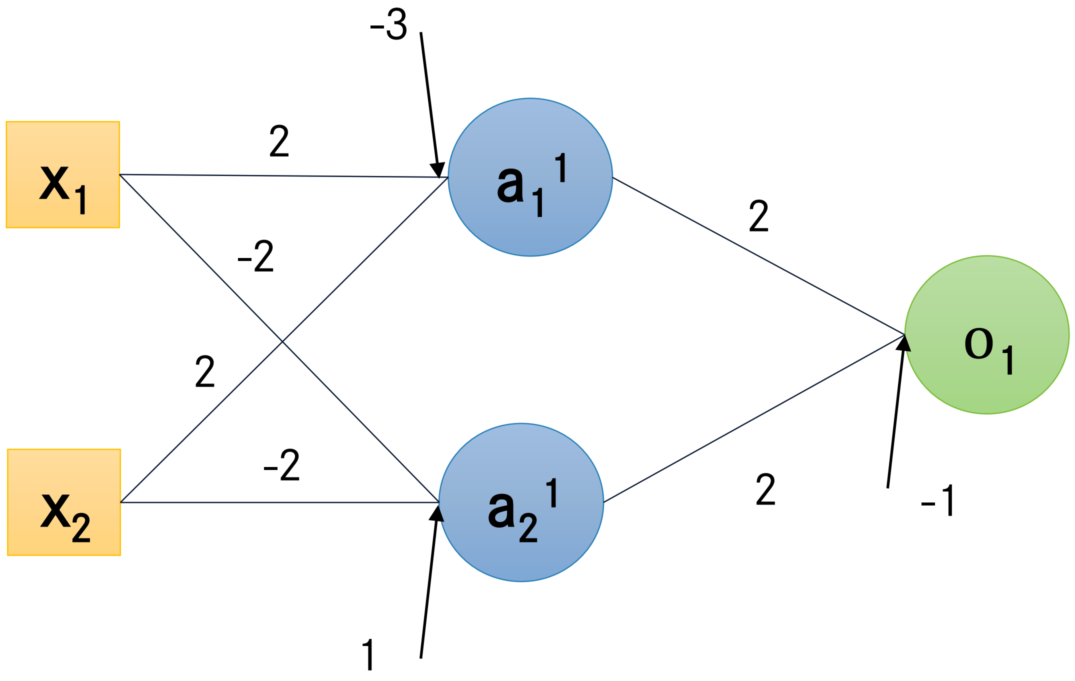
# How can you find the best function?




Oh my god!
No line can best separate the data!

Don't worry!
Neural Network can help you solve the problem!

# XNOR




| $x_1$ | $x_2$ | $a_1^1$ | $a_2^1$ | $o_1$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

# Gradient Descent


Loss Function:

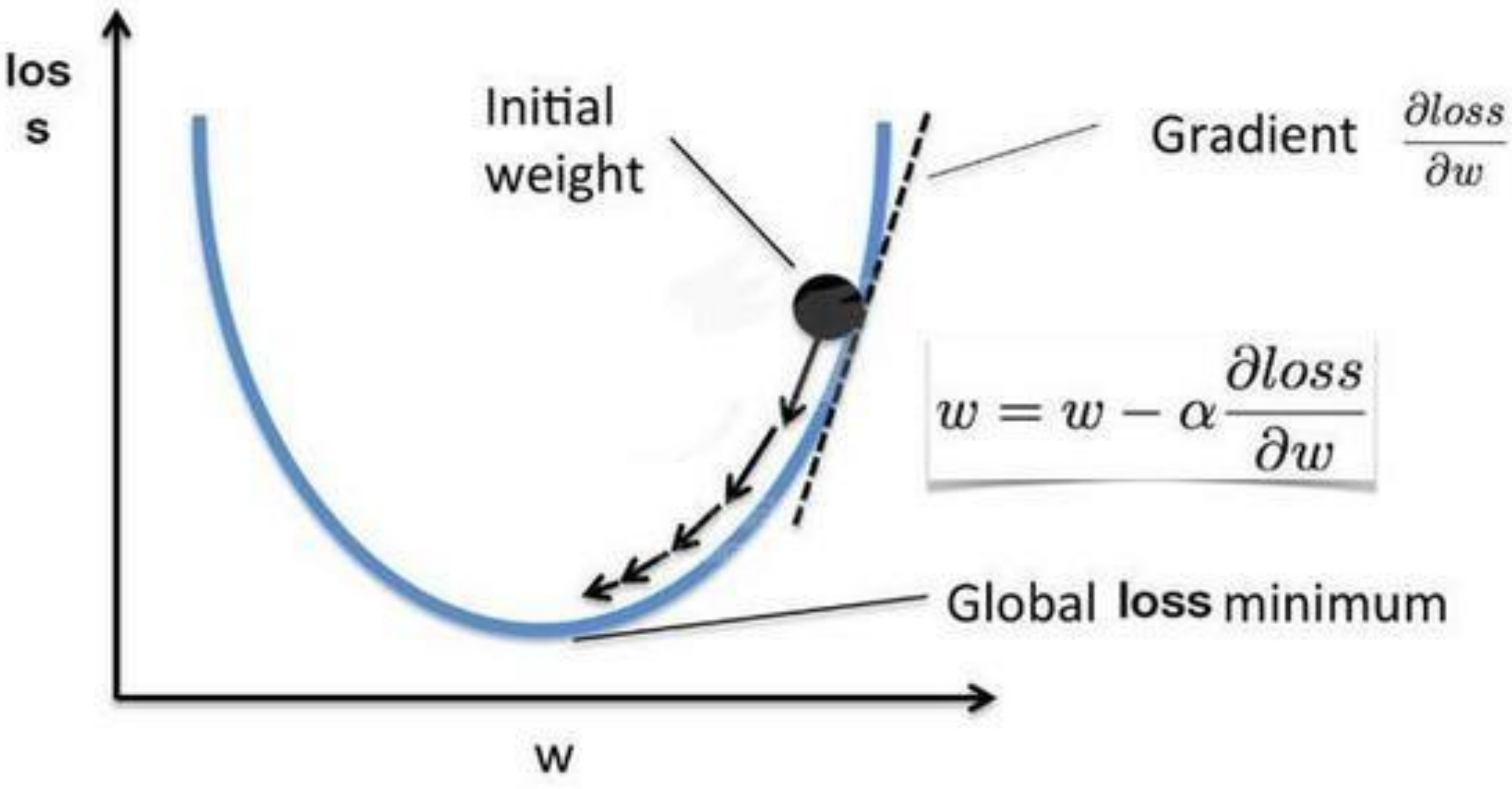
$$L = \frac{1}{N} \sum_{N}^{i} \frac{1}{2} (y_i - \hat{y}_i)^2$$

Objective: minimize the total loss

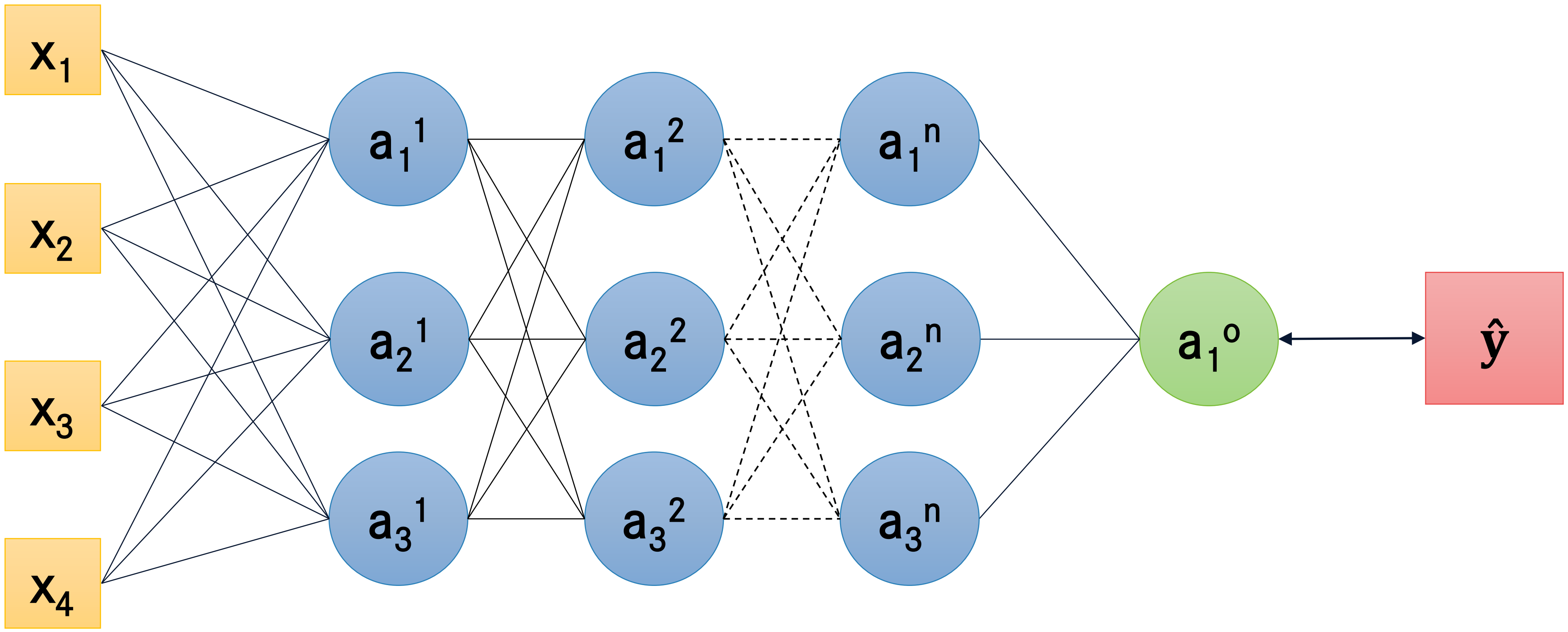
$$w \leftarrow w - \alpha \cdot \frac{\partial L}{\partial w}$$

(Here α is learning rate, which controls the range of each step)

# Gradient Descent



# Backpropagation (Chain Rule)



# Deep Learning Frameworks

TensorFlow

PYTORCH

Keras

mxnet

torch

A SCIENTIFIC COMPUTING FRAMEWORK FOR LUAJIT

Caffe2

# Word Embedding

# Discrete Representation

- Commonest Linguistic Idea: *signifier* and *signified* (Saussure)

- One-Hot Encoding can represent word. It is a vector with only one 1 and a lot of 0s.

- For example: *Hotel*

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

http://web.stanford.edu/class/cs224n

# Problems with Discrete Representation

- It has no relation to the meaning of word

- Similar word vectors should have large inner product. But

$$\text{motel}\ [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$$
$$\text{hotel}\ [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] = 0$$

- We need a better solution to represent word meaning

http://web.stanford.edu/class/cs224n

# Distributed Representation

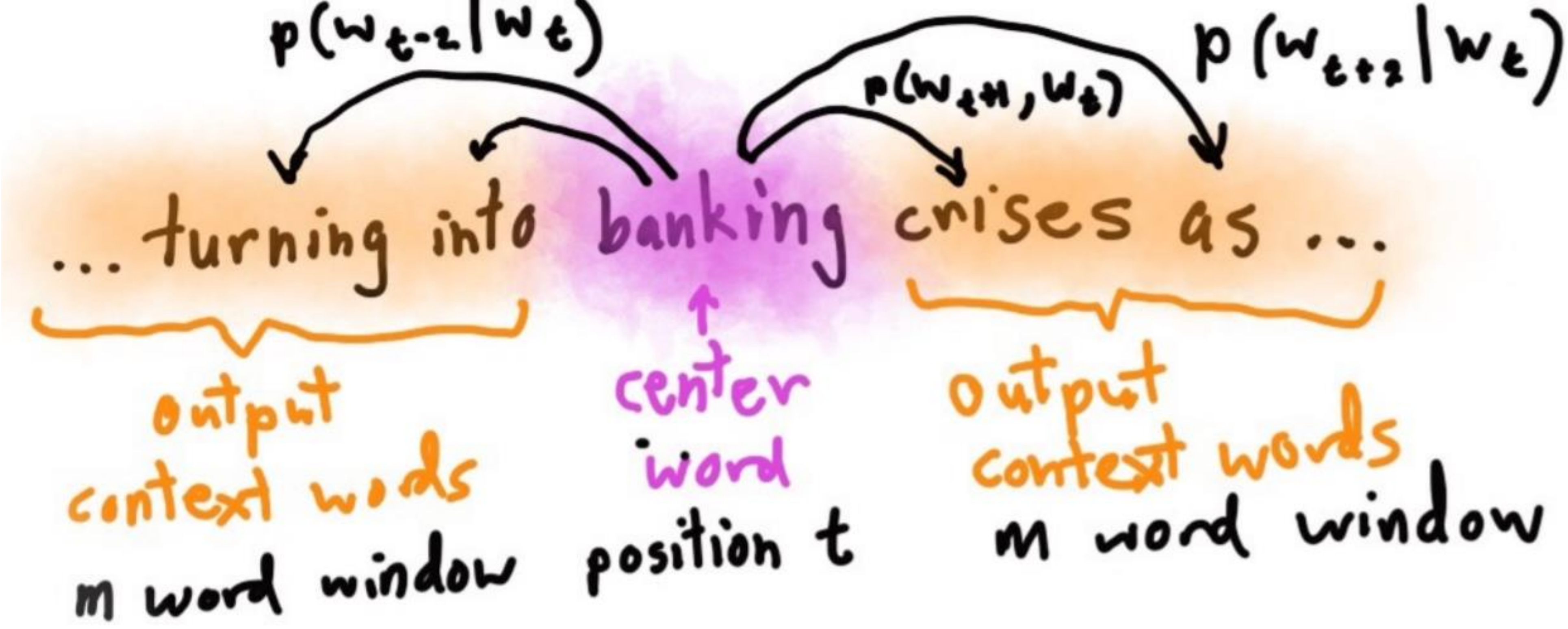
- *You shall know a word by the company it keeps.*

(J. R. Firth, 1957)

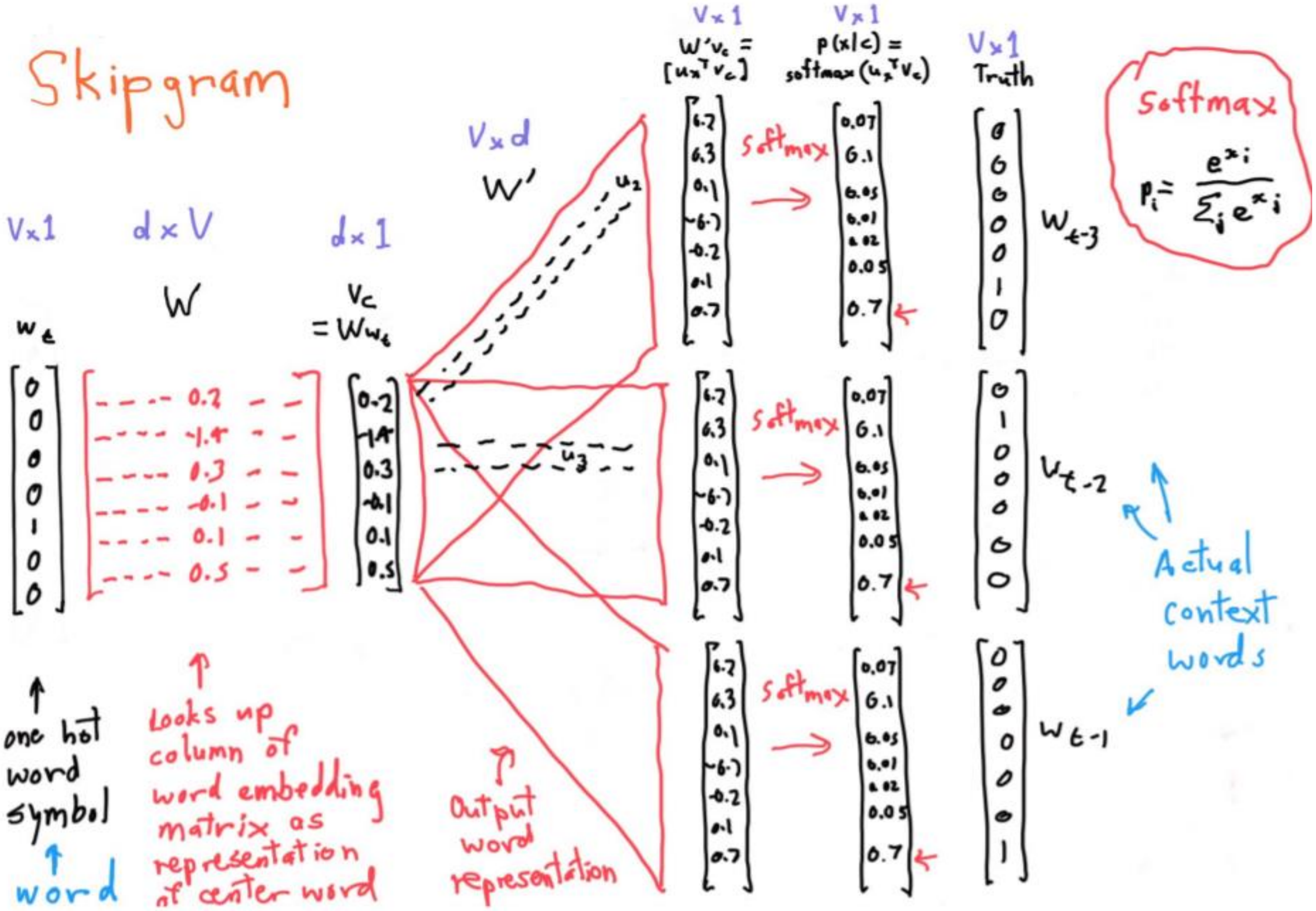
- Word Embedding can build distributed representations for words.

- Two of the most famous word embedding methods are:
- Word2Vec (Skip-Grams, CBOW)
- GloVe (Global Vector)

# Skip-Grams




http://web.stanford.edu/class/cs224n

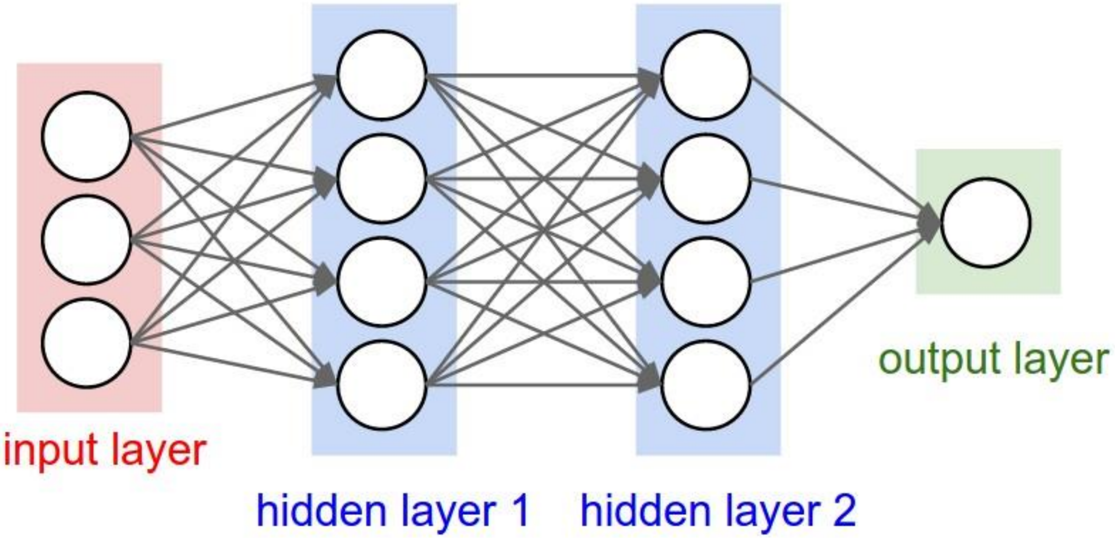



http://web.stanford.edu/class/cs224n

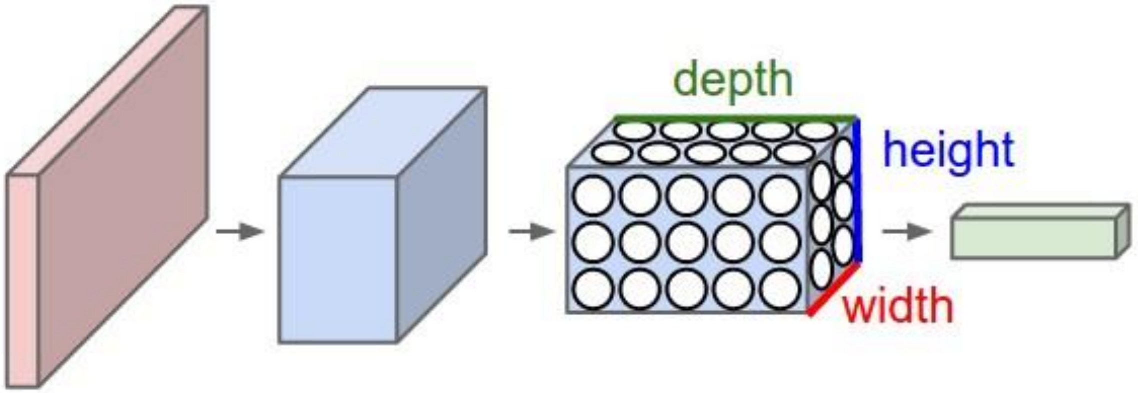
# Popular Networks

# Convolutional Neural Network

# Convolutional Neural Network (CNN)



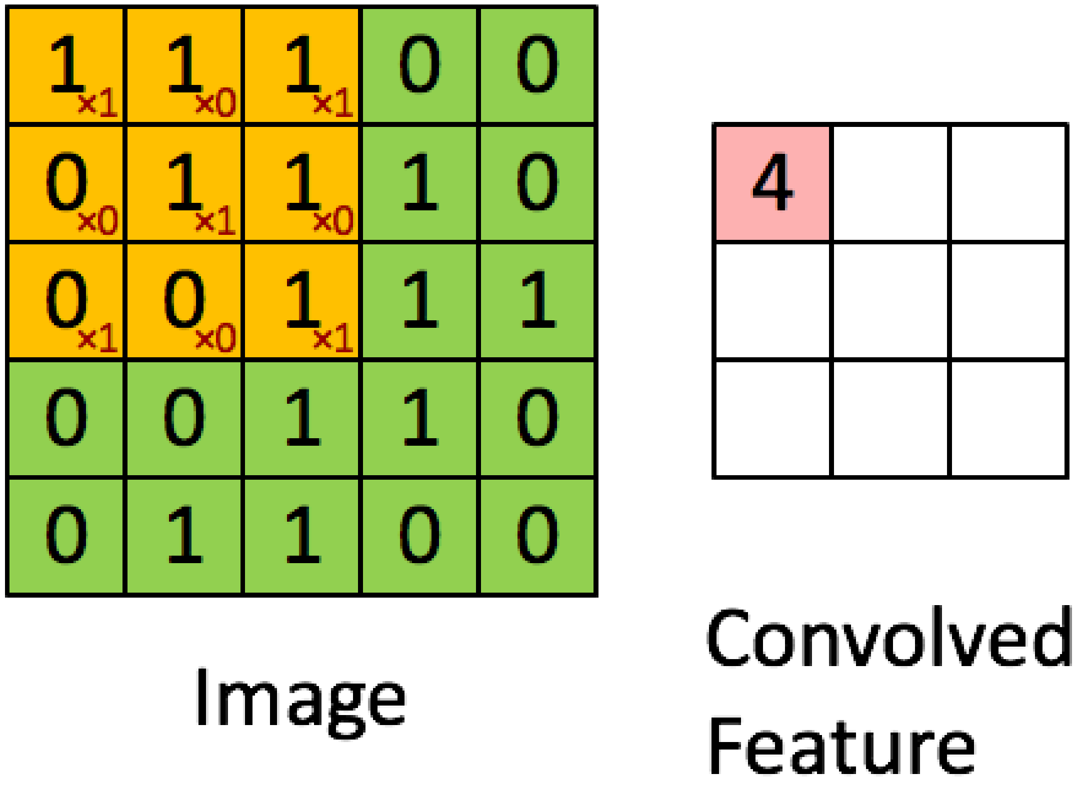

Fully-connected Feedforward Neural Network




Convolutional Neural Network

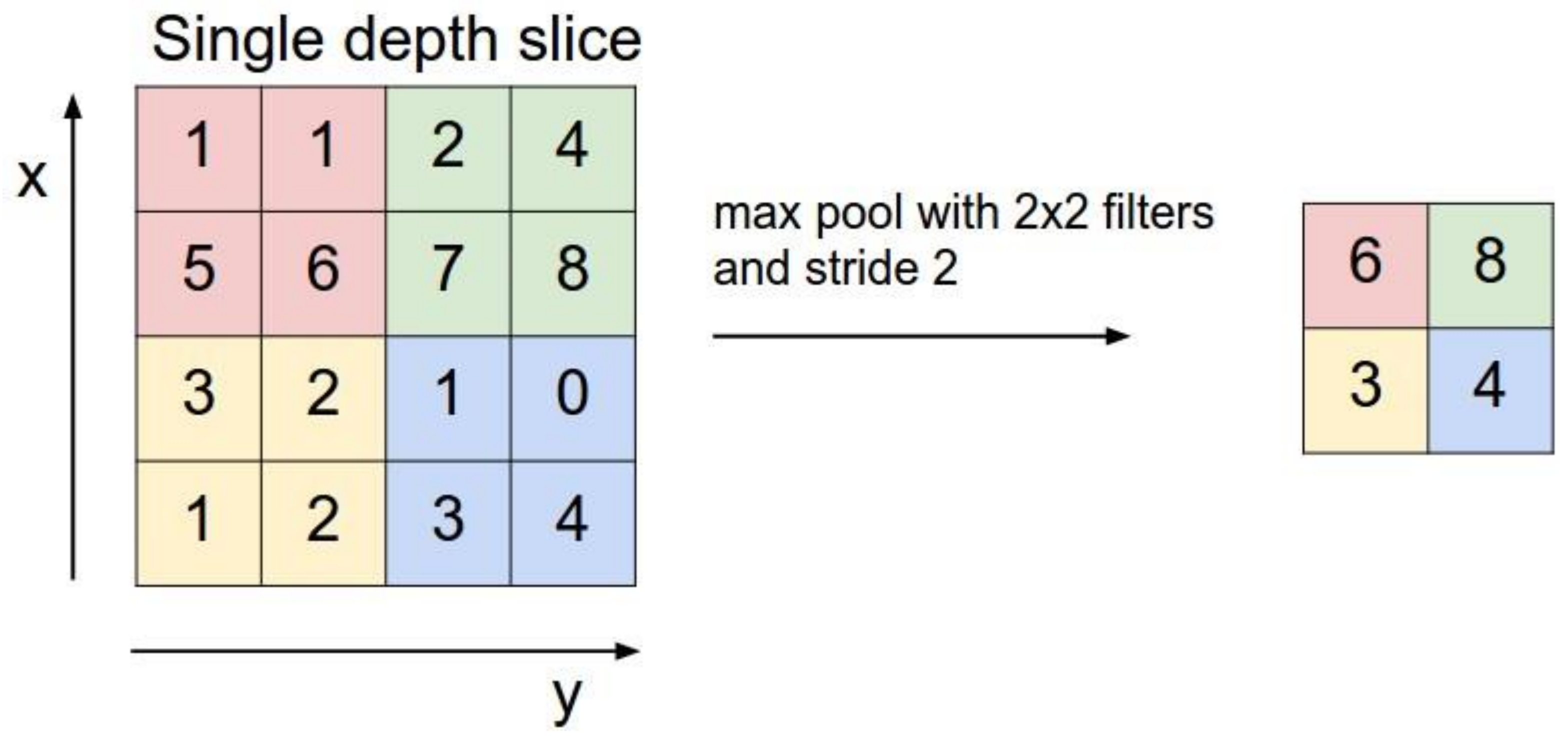
http://cs231n.github.io/convolutional-networks/#comp

# Convolutional Layer



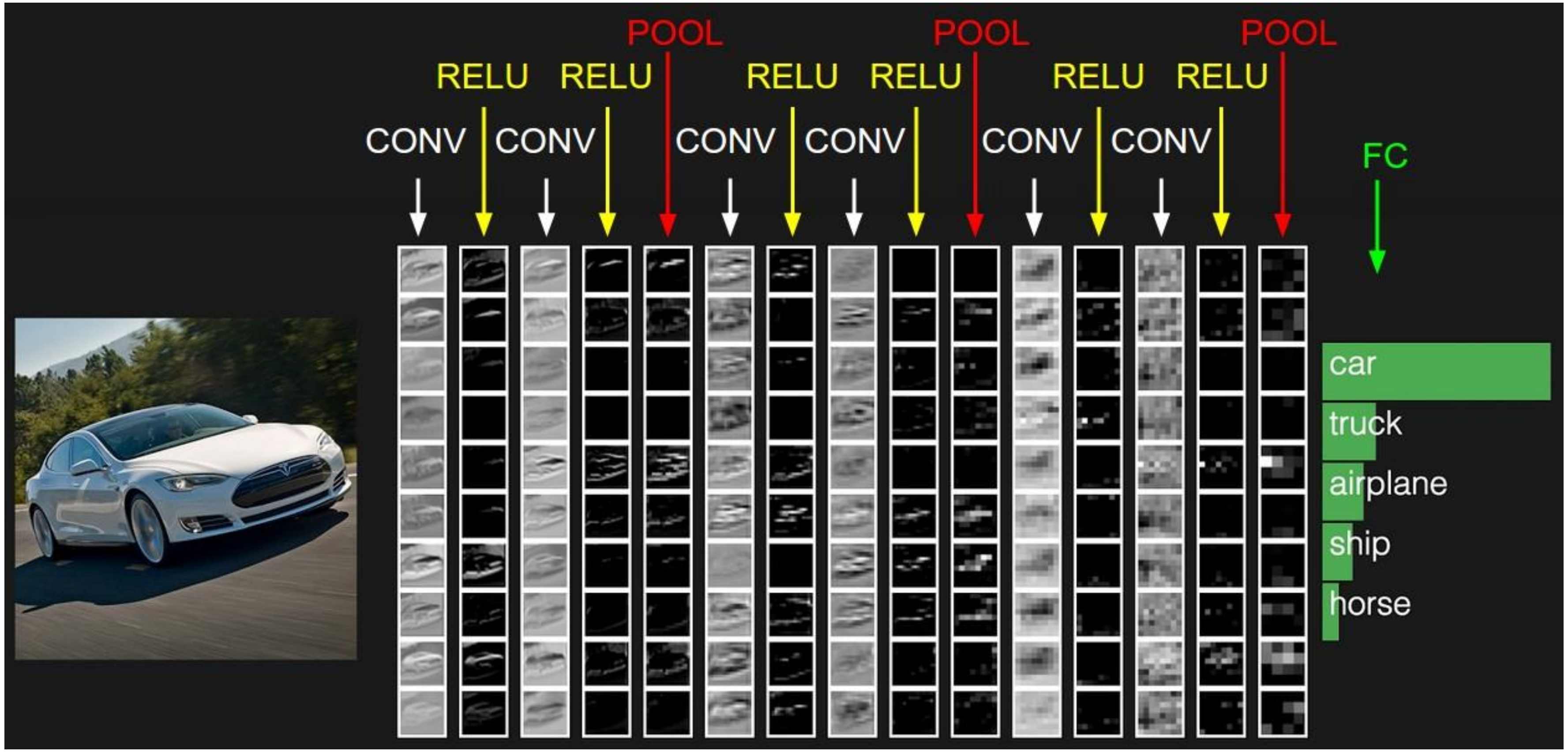

http://cs231n.github.io/convolutional-networks/#comp

# Max Pooling




http://cs231n.github.io/convolutional-networks/#comp

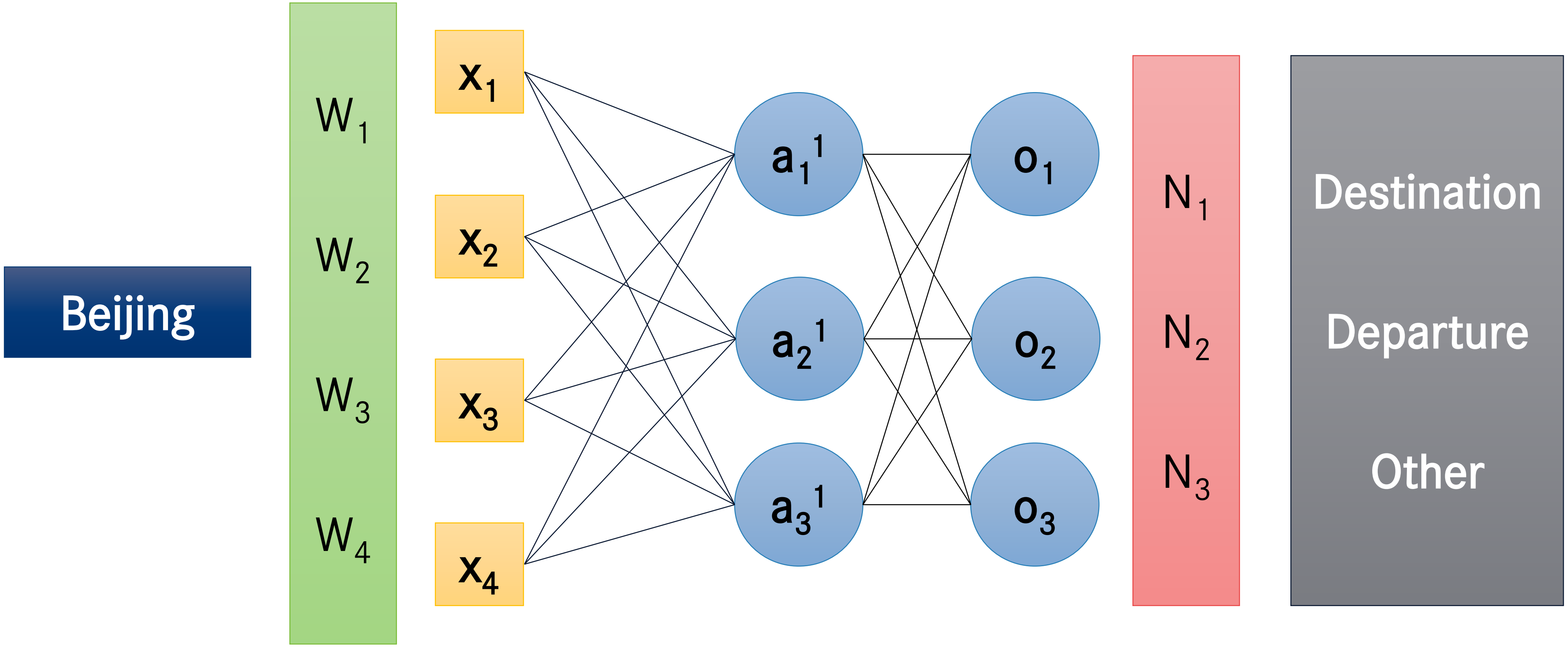
# Activations of CNN



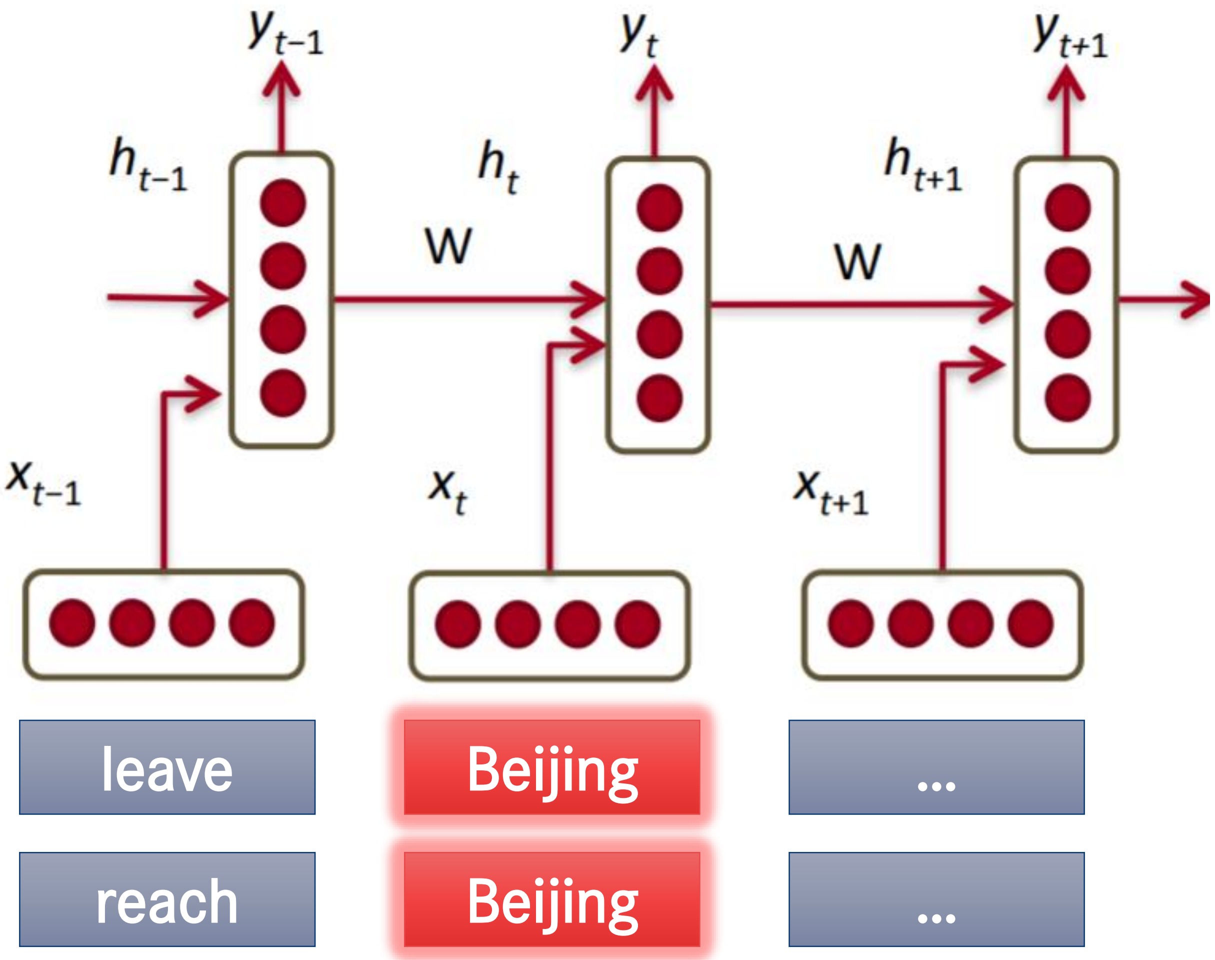

http://cs231n.github.io/convolutional-networks/#comp

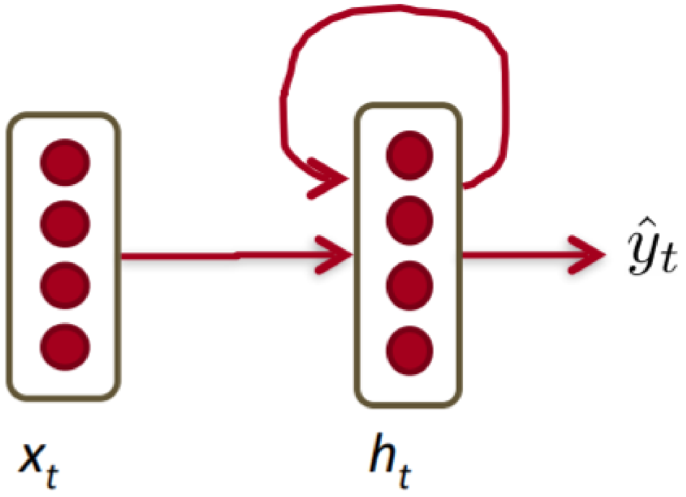
# Recurrent Neural Network

# Any Problem in Fully-connected Network?



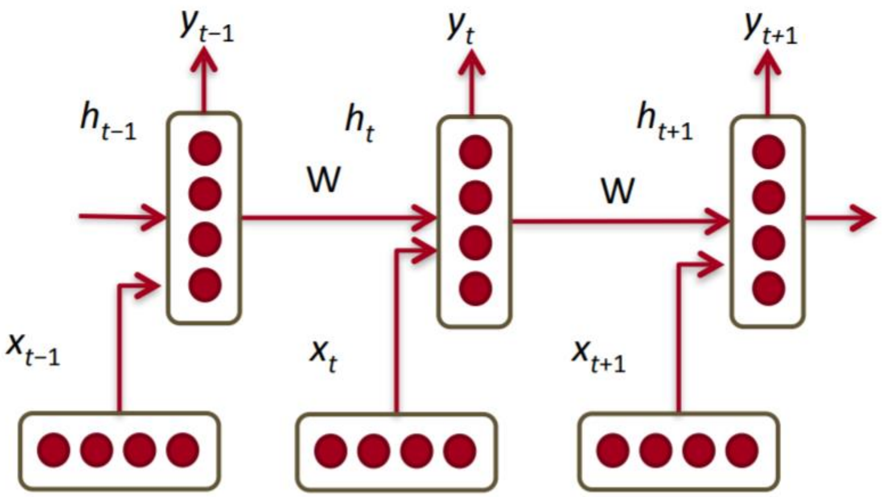
# Recurrent Neural Network (RNN)




http://web.stanford.edu/class/cs224n

# RNN vs Language Model




$$h_t = \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right)$$

$$\hat{y}_t = \text{softmax}\left(W^{(S)}h_t\right)$$

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

http://web.stanford.edu/class/cs224n

# Why Deep Learning?

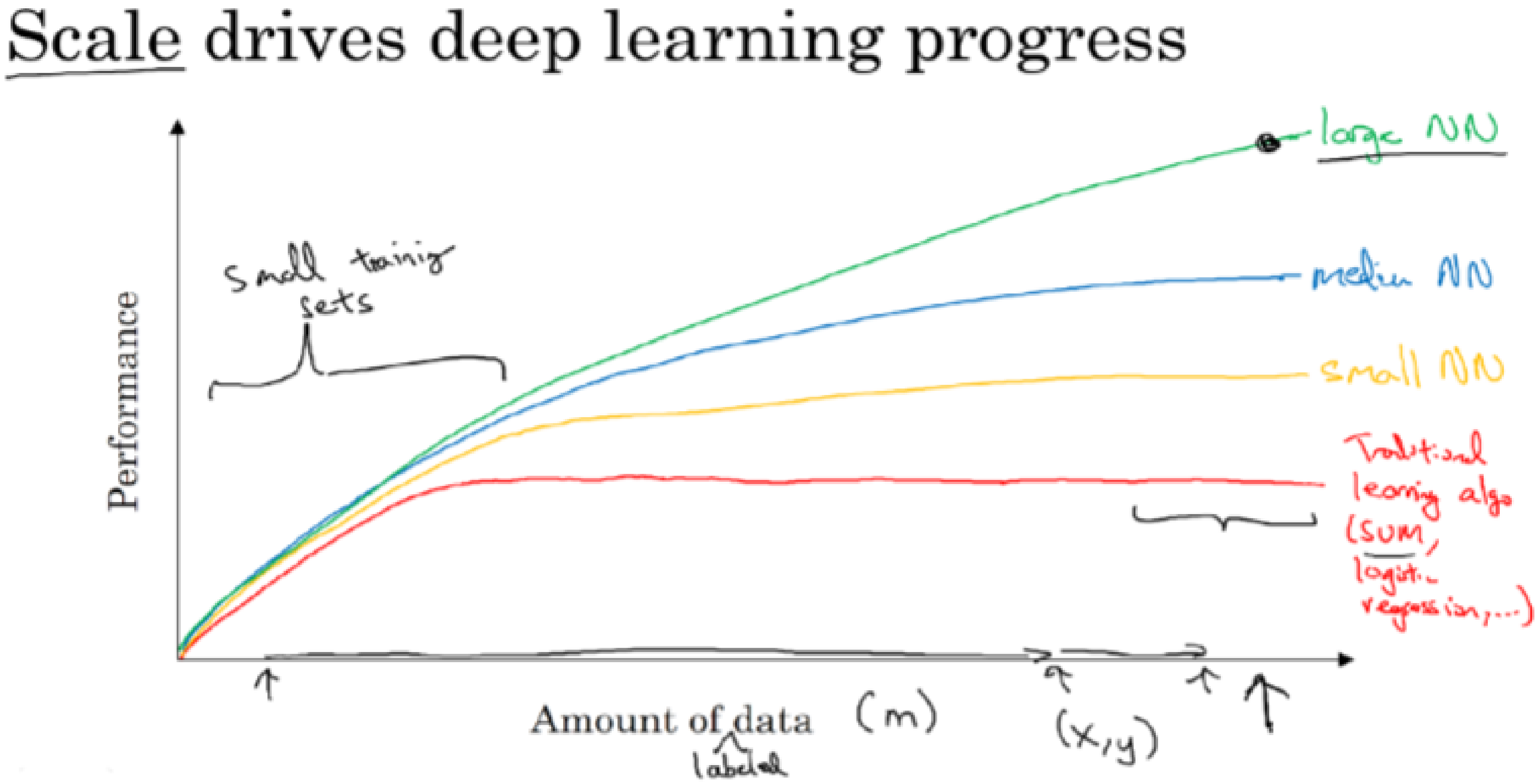
# Machine Learning vs Deep Learning

| Machine Learning | Deep Learning |
| --- | --- |
| Human-designed representations<br>+<br>Input features<br>+<br>Pick the best weights | Representation Learning<br>+<br>Raw Inputs<br>+<br>Pick the best algorithm |

# Advantages of Deep Learning

- Feature engineering is hard and to some extent, ineffective, incomplete or over-specified and it is really a hard work!
- Deep learning can learn features, which are easy to adapt and fast to learn.
- Flexible, universal and learnable
- More data and more powerful machines

# Advantages of Deep Learning




From Andrew Ng's course "Deep Learning"

# Future for Deep Learning?

- Unsupervised learning may be the most important research area in the future since it is pretty easy to achieve a large amount of unlabeled data while labelled data are far fewer and pretty expensive.

- Transfer Learning can help us transfer the task to pre-trained models

- Generative Model, such as GAN (Generative Adversarial Network)

- Abandon it?! (Well, Hinton said we should drop BP…)

# Personal Ideas About What We Can Do

- It seems that now linguists' contribution to NLP becomes trivial and deep learning does not really need us, but things may not be that bad.

- Still, we find the effects of many NLP tasks, like machine translation, not satisfactory, and machines cannot really understand semantic meaning, let alone pragmatic.

- More significant problems for scientists to solve in today's world, instead of improving the performance of algorithms, which are though vital.

# References

Book:

Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT press, 2016.

http://www.deeplearningbook.org/

https://github.com/exacity/deeplearningbook-chinese

Article:

LeCun Y, Bengio Y, Hinton G. Deep learning[J]. Nature, 2015, 521(7553): 436-444.

Schmidhuber J. Deep learning in neural networks: An overview[J]. Neural networks, 2015, 61: 85-117.

Goldberg Y. A Primer on Neural Network Models for Natural Language Processing[J]. J. Artif. Intell. Res.(JAIR), 2016, 57: 345-420.

Talk is Cheap,

Show me the Code!

```python
class NeuralNetwork(object):
    def sigmoid(self, x):
        return 1/(1 + np.exp(-x))

    def __init__(self, input_nodes, hidden_nodes, output_nodes, learning_rate):
        # Set number of nodes in input, hidden and output layers.
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes

        # Initialize weights
        self.weights_input_to_hidden = np.random.normal(0.0, self.hidden_nodes**-0.5,
                                        (self.hidden_nodes, self.input_nodes))

        self.weights_hidden_to_output = np.random.normal(0.0, self.output_nodes**-0.5,
                                        (self.output_nodes, self.hidden_nodes))
        self.lr = learning_rate

        # Activation function is the sigmoid function
        self.activation_function = self.sigmoid
```

```python
def train(self, inputs_list, targets_list):
    # Convert inputs list to 2d array, column vector
    inputs = np.array(inputs_list, ndmin=2).T
    targets = np.array(targets_list, ndmin=2).T

    #### Implement the forward pass here ####
    ### Forward pass ###
    #Hidden layer
    hidden_inputs = np.dot(self.weights_input_to_hidden, inputs)
    hidden_outputs = self.activation_function(hidden_inputs)

    #Output layer
    final_inputs = np.dot(self.weights_hidden_to_output, hidden_outputs)
    final_outputs = final_inputs

    #### Implement the backward pass here ####
    ### Backward pass ###

    # 1 is the gradient of f'(x) where f(x) = x
    output_delta = (targets - final_outputs) * 1

    hidden_delta = np.dot(self.weights_hidden_to_output.T, output_delta) * hidden_outputs * (1-hidden_outputs)

    # TODO: Update the weights
    self.weights_hidden_to_output += self.lr * np.dot(output_delta, hidden_outputs.T)
    self.weights_input_to_hidden += self.lr * np.dot(hidden_delta, inputs.T)
```

```
#predict with a inputs_list
def run(self, inputs_list):
    # Run a forward pass through the network
    inputs = np.array(inputs_list, ndmin=2).T

    #### Implement the forward pass here ####
    #Hidden layer
    hidden_inputs = np.dot(self.weights_input_to_hidden, inputs)
    hidden_outputs = self.activation_function(hidden_inputs)

    #Output layer
    final_inputs = np.dot(self.weights_hidden_to_output, hidden_outputs)
    final_outputs = final_inputs

    return final_outputs
```