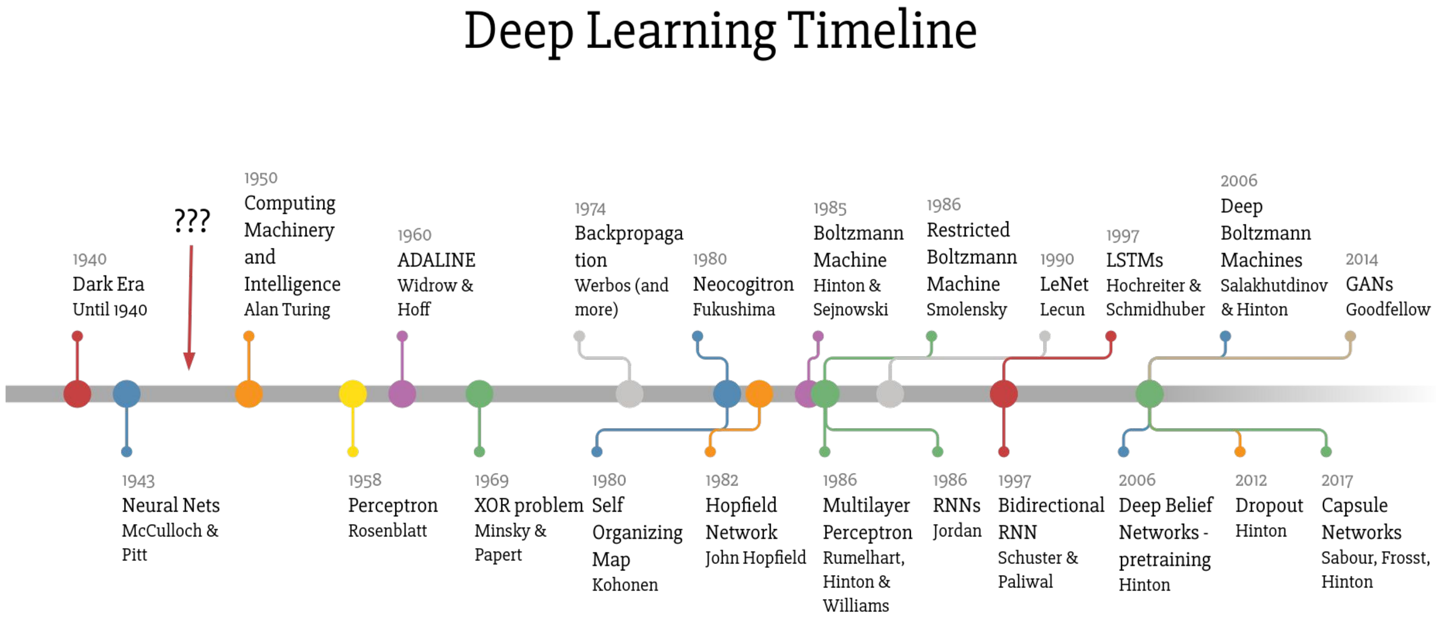
Applied Machine Learning for Business Analytics

Lecture 3: Neural Networks and Deep Learning

Lecturer: Zhao Rui

DL/NN is not New



Made by Favio Vázquez

Why DL is powerful now?

- Feature engineering require high-level expert knowledge, which are easily over-specified and incomplete.
- Large amounts of training data
- Modern multi-core CPUs/GPUs/TPUs
- Better deep learning ‘tricks’ such as regularization, optimization, transfer learning etc.

Deep learning myth: three steps

- To deploy deep learning (or other machine learning) systems

Collect the data

↓

Train the model

↓

Deploy the model

The truth

1. Select a metric for optimization 💪
2. Collect data 🙇
3. Train model 👩‍💻
4. Realize many labels are wrong 😱
5. Relabel data ✍️
6. Train model 👩‍💻
7. Model performs poorly on one class 🤦
8. Collect more data for that class 🙇
9. Train model 👩‍💻
10. Model performs poorly on most recent data 🤦
11. Collect more recent data 🙇
12. Train model 👩‍💻
13. Deploy model 👩‍🔧
14. Dream 🤑
15. Get a call at 3am about complaints that model is biased 🧑‍🦲
16. Revert to the older version
17. Collect more data, do more training and testing👨‍💻
18. Deploy model 👩‍🔧
19. Pray 🙆
20. Model performs well but revenue decreasing 🤷
21. Cry 😭
22. Choose a different metric 💪
23. Start over 👨‍🦲

Three steps in deep learning

To approximate the true function, define a function **space**

Neural network structures

Learning Representation

Need a **measure** to evaluate the quality of each potential function in the previous space

Evaluate the function

Objective Function

Search the function space to find the best function based on the measure.

Pick the best function

Optimization

Find model parameters:

$y=ax+b$ → $y=2x+1$

a=2 and b=1

function space → a function

Agenda

1. Linear Regression
2. Neural Networks
3. Evaluation of Functions
4. Optimization
5. Deep Representation Learning
6. Application of DL

1. Linear Regression

Linear regression (Single Variable)

- Model architecture: y=ax+b
- Objective function: Mean Squared Error Function

$$J(a,b) = \frac{1}{n}\sum_{i=0}^{n}(y_i - (ax_i + b))^2$$

- Optimization: Gradient Descent Algorithm

Gradient descent

Gradient for the total loss
function over parameters,

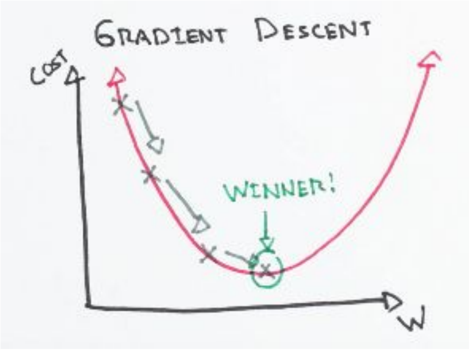
$$\theta_{t+1} = \theta_t - \alpha \boxed{\nabla J(\theta_t)}$$

New
Parameters
Guess

Current
Parameters
Guess

Learning
Rate

Like hiking down a mountain



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

Simple math

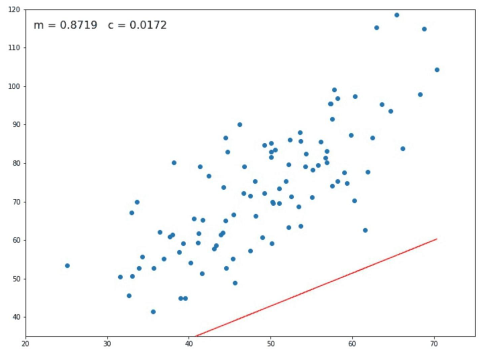
- Gradients for parameters:

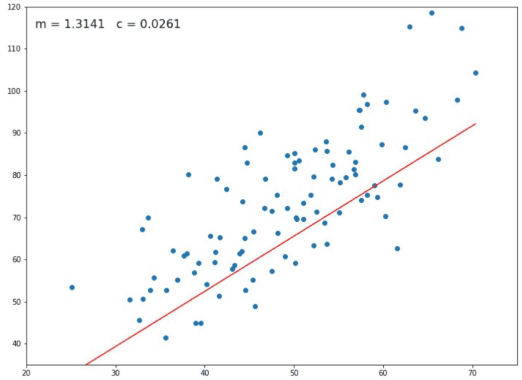
$$\frac{\partial}{\partial a}J(a,b) = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (ax_i + b))(-x_i)$$

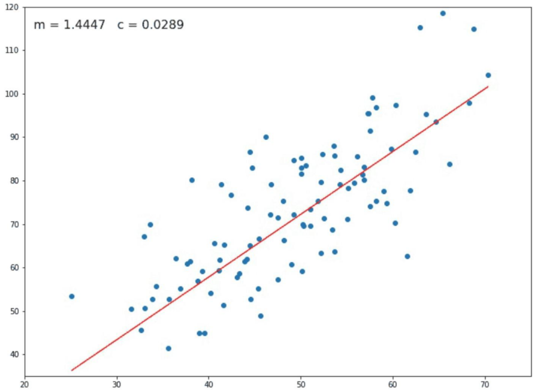
$$\frac{\partial}{\partial a}J(a,b) = \frac{-2}{n}\sum_{i=0}^{n}(y_i - y_i^{'})x_i$$

$$\frac{\partial}{\partial b}J(a,b) = \frac{-2}{n}\sum_{i=0}^{n}(y_i - y_i^{'})$$

Optimization

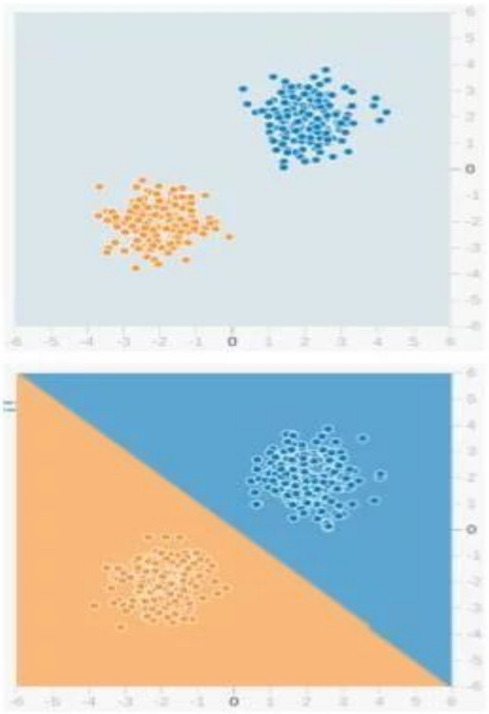


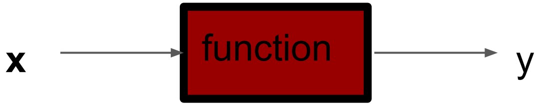




2. Neural Networks

A “simple” classification problem





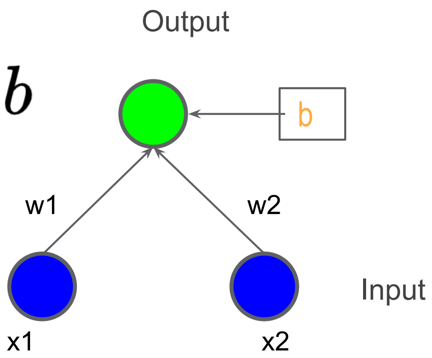
$$y = \mathbf{w}\mathbf{x} + b$$

A linear model

- Linear Regression if output is continuous
- Logistic Regression if output is discrete

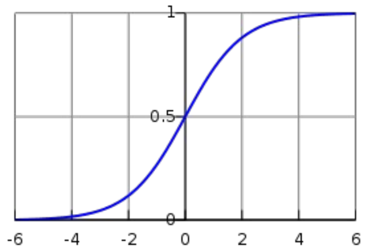
Linear Regression

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



Logistic Regression

$$y = \sigma(\mathbf{wx} + b)$$

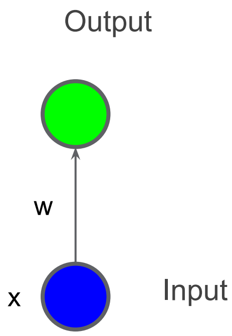


A linear model

- Linear Regression if output is continuous
- Logistic Regression if output is discrete

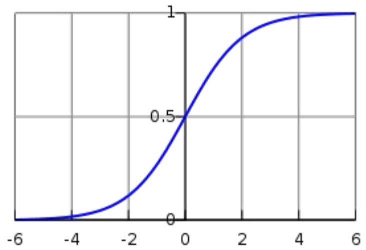
Linear Regression

$$\hat{y} = w * x$$



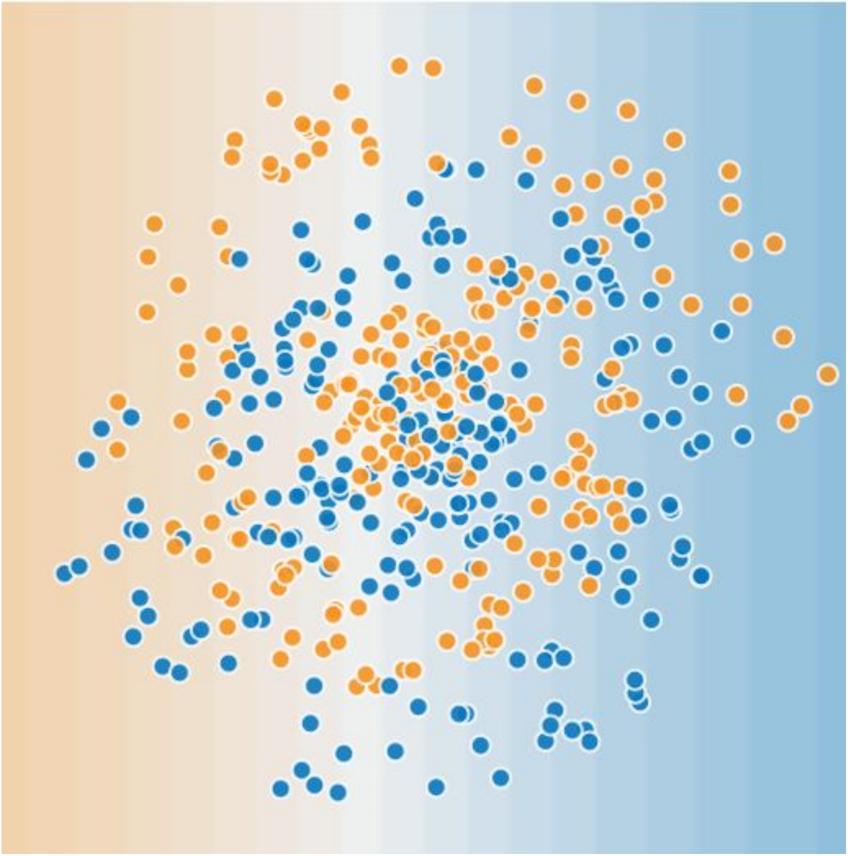
Logistic Regression

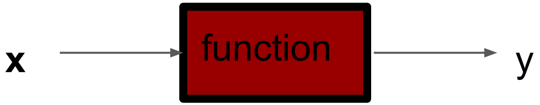
$$y = \sigma(\mathbf{wx} + b)$$



How about this classification problem?

Linear model can not solve it





We need **non-linear functions**

Add complexity

For Simplicity, the bias term is ignored here.

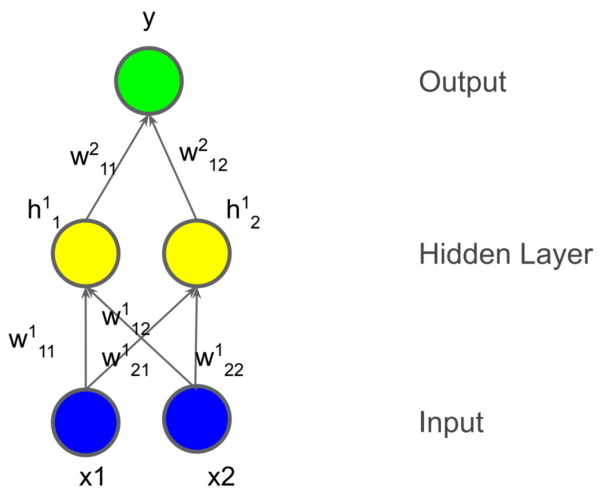
$$h^1_1 = w^1_{11}x1 + w^1_{12}x2$$

$$h^1_2 = w^1_{21}x1 + w^1_{22}x2$$

$$y = w^2_{11}h^1_1 + w^2_{12}h^1_2 = y = \mathbf{wx}$$

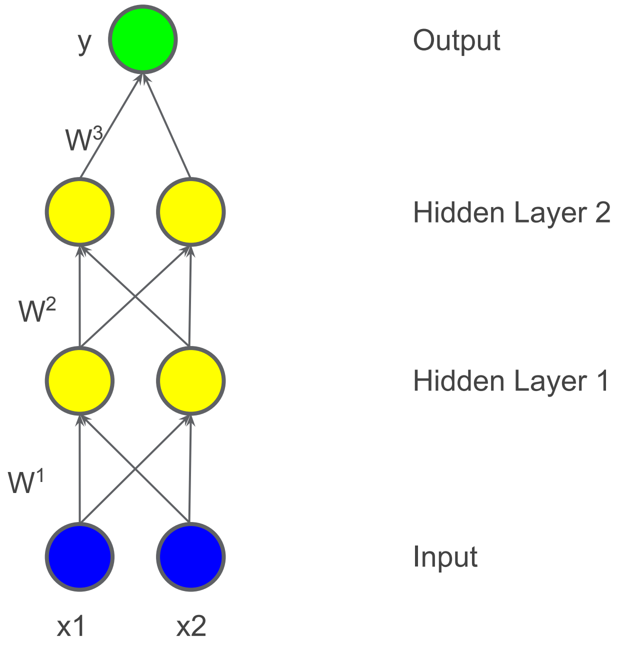
$$\Downarrow$$

$$y = (w^2_{11}w^1_{11} + w^2_{21}w^1_{12})x1 + (w^2_{12}w^1_{12} + w^2_{12}w^1_{22})x2$$



Add complexity

$$y = \mathbf{W}^3\mathbf{W}^2\mathbf{W}^1 \begin{bmatrix} x1 \\ x2 \end{bmatrix} = (\mathbf{W}^3\mathbf{W}^2\mathbf{W}^1) \begin{bmatrix} x1 \\ x2 \end{bmatrix}$$



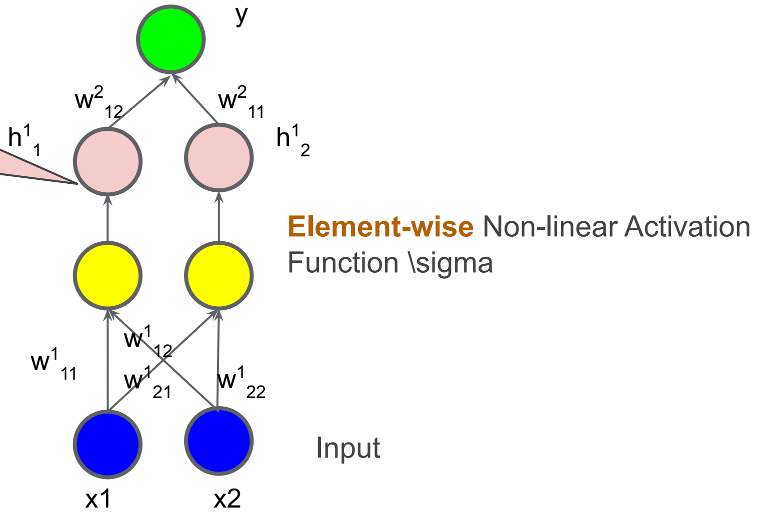
Make it non-linear

$$h_1^1 = \sigma(w_{11}^1 x1 + w_{12}^1 x2)$$

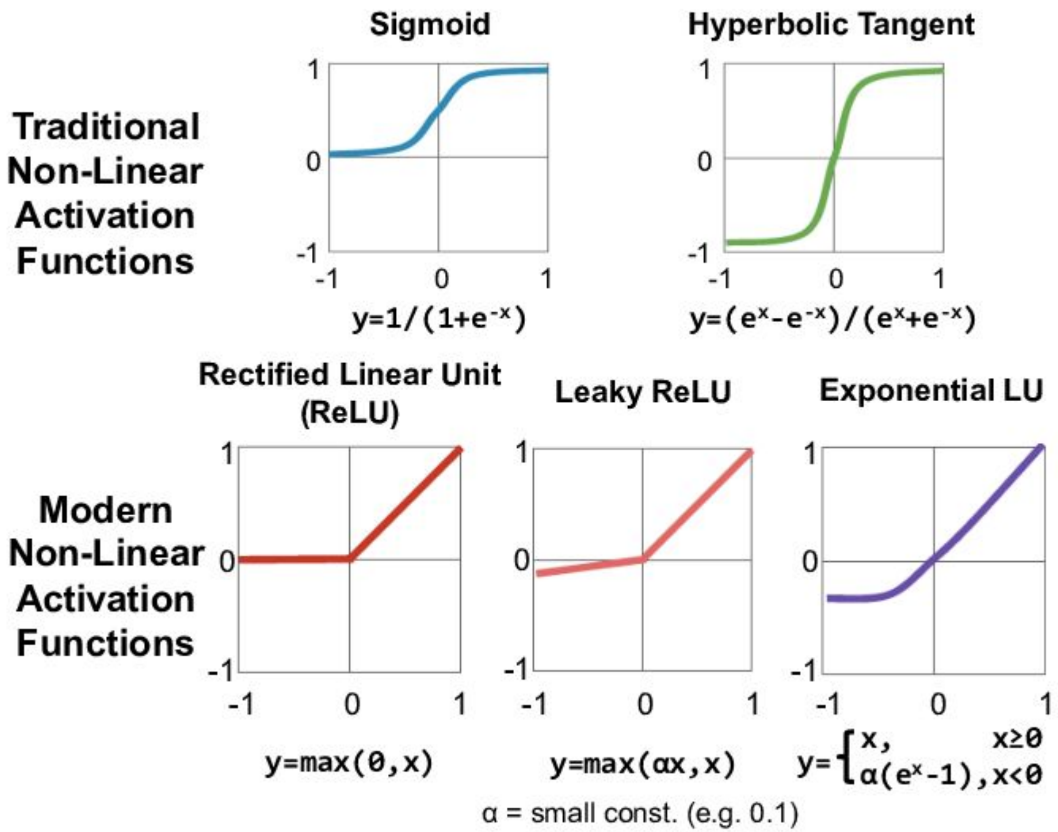
$$h_2^1 = \sigma(w_{21}^1 x1 + w_{22}^1 x2)$$

$$y = w_{11}^2 h_1^1 + w_{12}^2 h_2^1 \neq y = \mathbf{wx}$$

We Usually Don't Draw Non-Linear Transforms



Non-linear activation functions



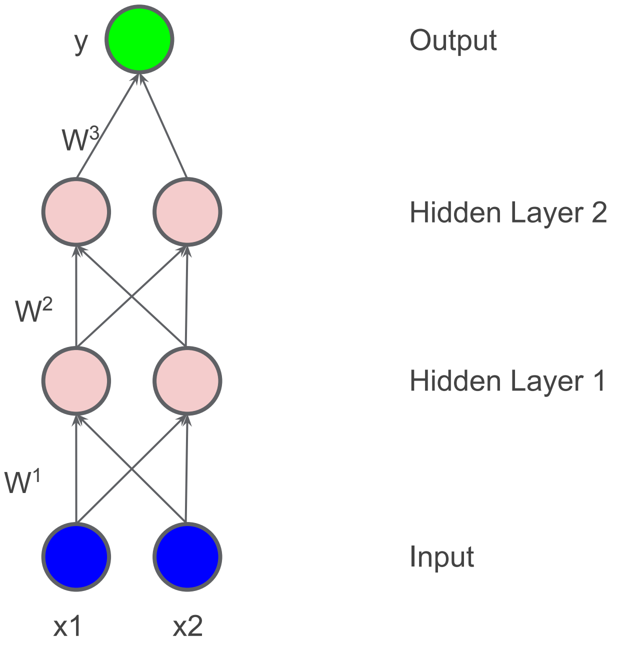
Add non-linear activation function

$$y = \mathbf{W}^3\sigma(\mathbf{W}^2\sigma(\mathbf{W}^1\mathbf{x}))$$

$$y = \mathbf{W}^3\mathbf{h}^2$$

$$\mathbf{h}^2 = \sigma(\mathbf{W}^2\mathbf{h}^1)$$

$$\mathbf{h}^1 = \sigma(\mathbf{W}^1\mathbf{x})$$



Why non-linear activation

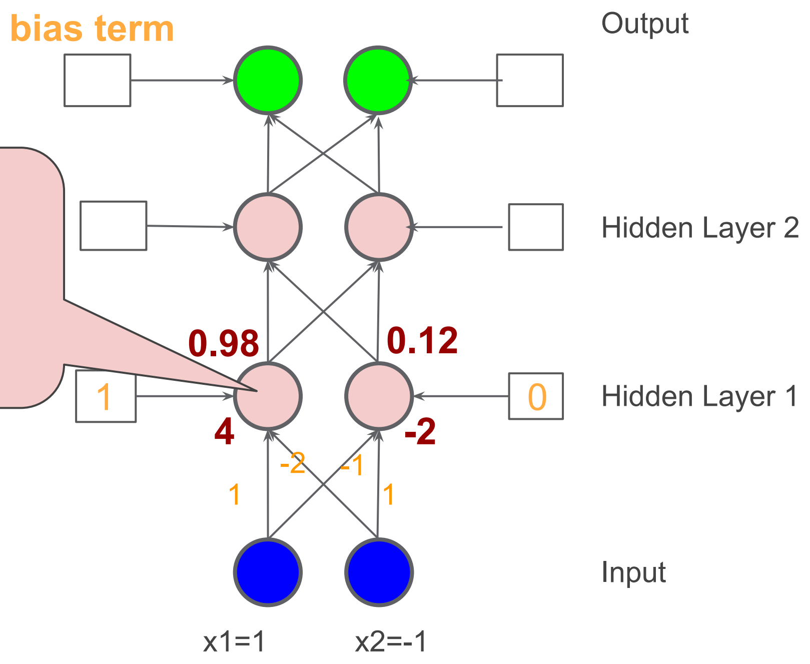
- The non-linearities activation function increases the capacity of model
- Without non-linearities, deep neural networks is meaningless: each extra layer is just one linear transform.
- How to select activation functions?

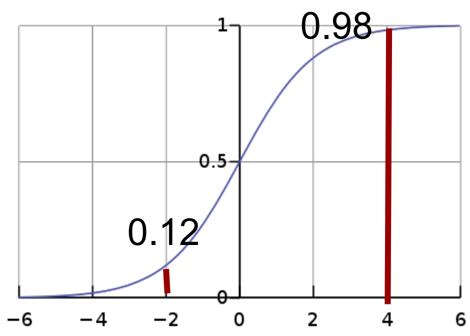
  You can select an activation function which will approximate the distribution faster leading to faster training process.

Forward computation

Sigmoid function:

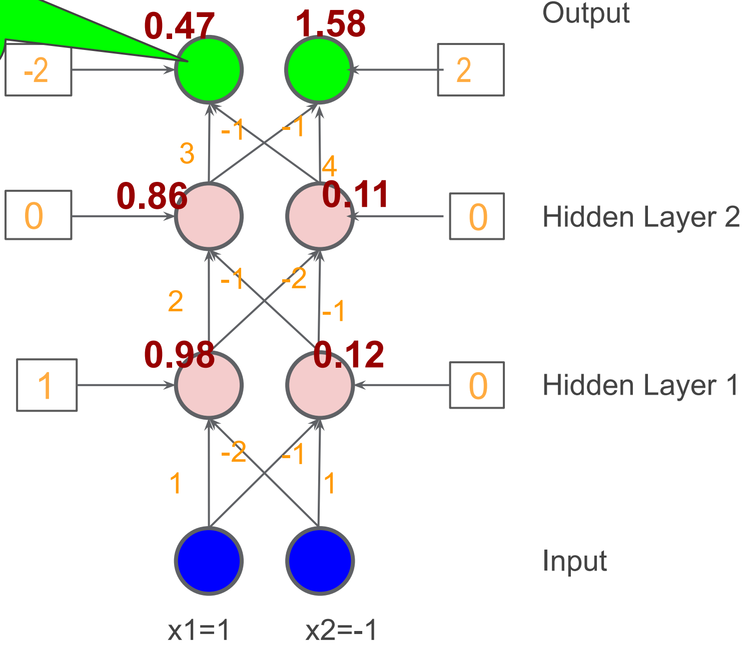
$$h = \sigma(z) = \frac{1}{1+e^{-z}}$$





Forward computation

Identity Function. It can be non-linear functions specified by applications.



Forward computation

1. Neural Network acts as a function that transforms the input vector into the output vector (target)

$$\begin{bmatrix} 0.47 \\ 1.58 \end{bmatrix} = f_\theta(\begin{bmatrix} 1 \\ -1 \end{bmatrix})$$

$$\begin{bmatrix} 0.04 \\ 1.7 \end{bmatrix} = f_\theta(\begin{bmatrix} 0 \\ 0 \end{bmatrix})$$

$$\mathbf{W}^1 = \begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \quad \mathbf{b}^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mathbf{W}^2 = \begin{bmatrix} 2 & -1 \\ -2 & -1 \end{bmatrix} \quad \mathbf{b}^2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

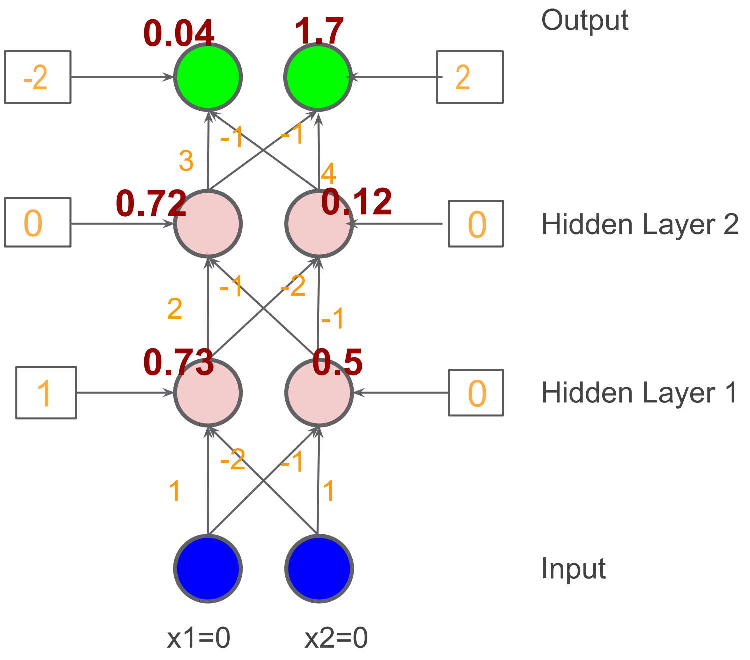
$$\mathbf{W}^3 = \begin{bmatrix} 3 & -1 \\ -1 & 4 \end{bmatrix} \quad \mathbf{b}^3 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$

one param. set

2. It is actually a function space parameterized by weights matrices and bias vectors.

$$\theta = \{\mathbf{W}^1, \mathbf{b}^1, \mathbf{W}^2, \mathbf{b}^2, \mathbf{W}^3, \mathbf{b}^3\}$$

Parameter space

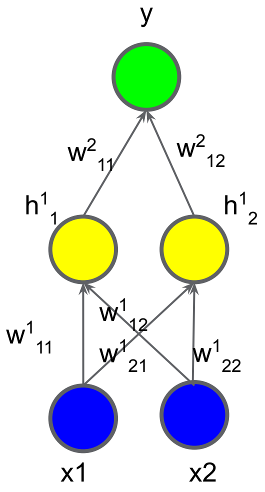


Add complexity

Associative Law

$$h_1^1 = w_{11}^1 x1 + w_{12}^1 x2$$
$$h_2^1 = w_{21}^1 x1 + w_{22}^1 x2$$

$$y = w_{11}^2 h_1^1 + w_{12}^2 h_2^1$$



Matrix Format

Output

$$\begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \end{bmatrix}$$

Hidden Layer

$$y = \begin{bmatrix} w_{11}^2 & w_{12}^2 \end{bmatrix} \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix}$$

Input

$$y = W^2 W^1 \begin{bmatrix} x1 \\ x2 \end{bmatrix} = (W^2 W^1) \begin{bmatrix} x1 \\ x2 \end{bmatrix} = W \begin{bmatrix} x1 \\ x2 \end{bmatrix}$$

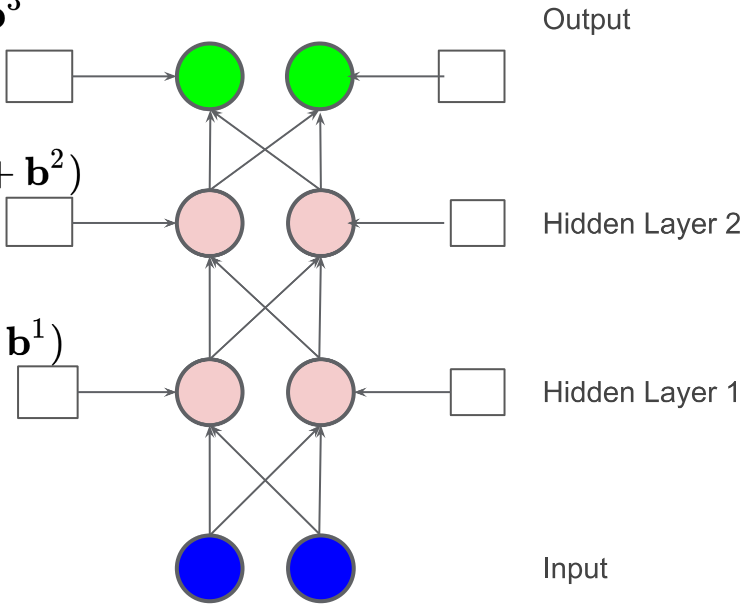
Forward computation

$$y = \mathbf{W}^3\sigma(\mathbf{W}^2\sigma(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3$$

$$y = \mathbf{W}^3\mathbf{h}^2 + \mathbf{b}^3$$

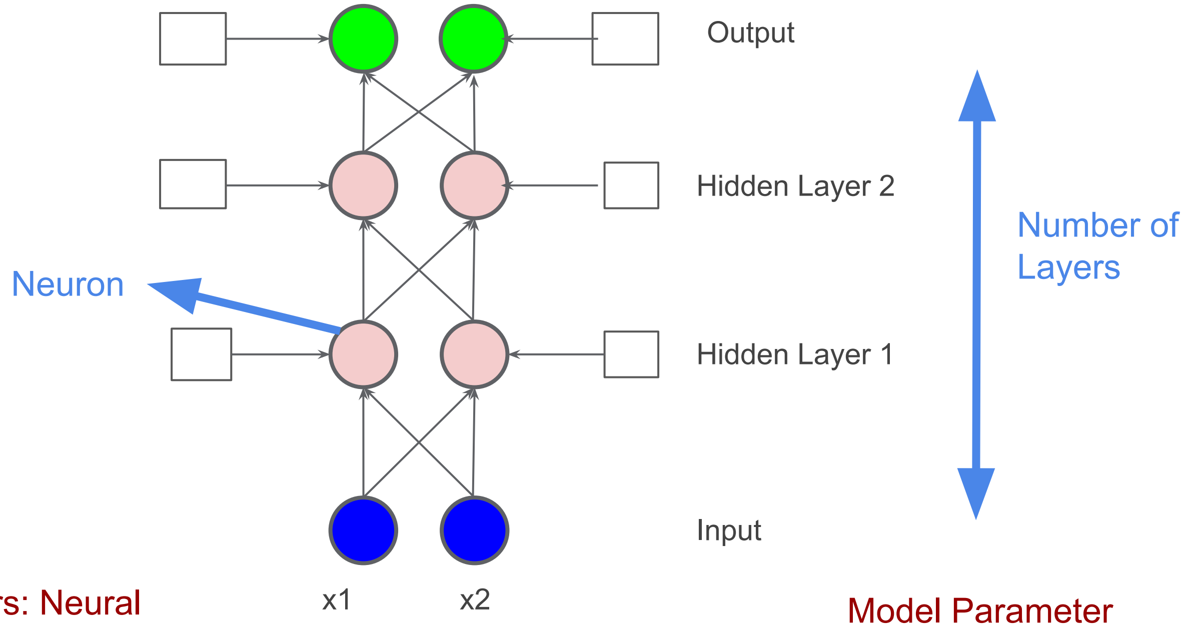
$$\mathbf{h}^2 = \sigma(\mathbf{W}^2\mathbf{h}^1 + \mathbf{b}^2)$$

$$\mathbf{h}^1 = \sigma(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1)$$



1. Neural Network is a model that **recursively** applies the matrix multiplication and non-linear activation function.
2. Parallel computing techniques can be used to speed up matrix operation.

Neural network: function set



Hyperparameters: Neural Network Structure

Model Parameter

1.Number of Layers
2.Number of neurons in each layer
3.Non-linear Activation function in each layer

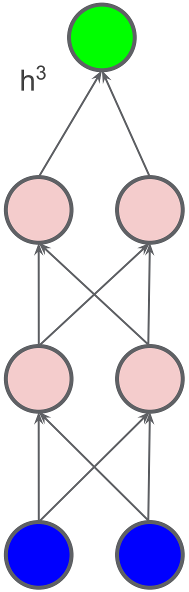
A function space containing various functions

A specific function mapping from input data to targets.

Output layer

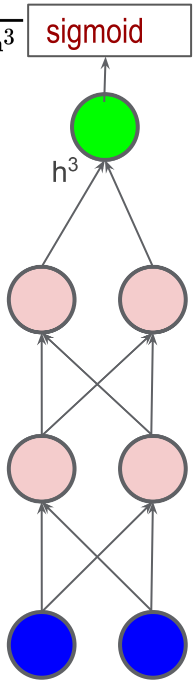
Regression

$$\tilde{y} = \mathbf{h}^3$$



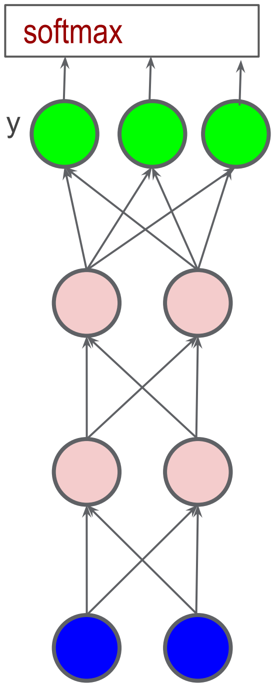
Binary Classification

$$\tilde{y} = \frac{1}{1+e^{-\mathbf{h}^3}}$$



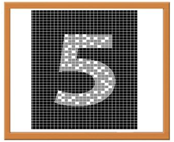
Multi-class Classification

$$\tilde{\mathbf{y}} = \left[\frac{e^{h_1^3}}{e^{h_1^3}+e^{h_2^3}+e^{h_3^3}}, \frac{e^{h_2^3}}{e^{h_1^3}+e^{h_2^3}+e^{h_3^3}}, \frac{e^{h_3^3}}{e^{h_1^3}+e^{h_2^3}+e^{h_3^3}}\right]$$



$$\tilde{y}_i = \frac{e^{h_i^3}}{\sum_{j=1}^{K} e^{h_j^3}}$$

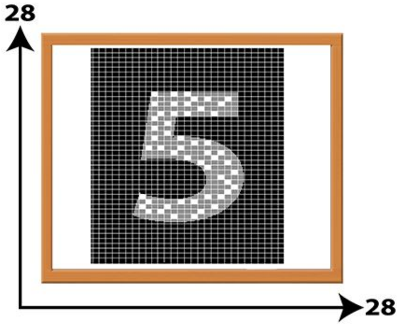
Example: MNIST dataset



Machine Learning Model

Output: 5

Input Vector



28*28=784

Pixel
Pixel
Pixel
Pixel
Pixel
⋮
Pixel

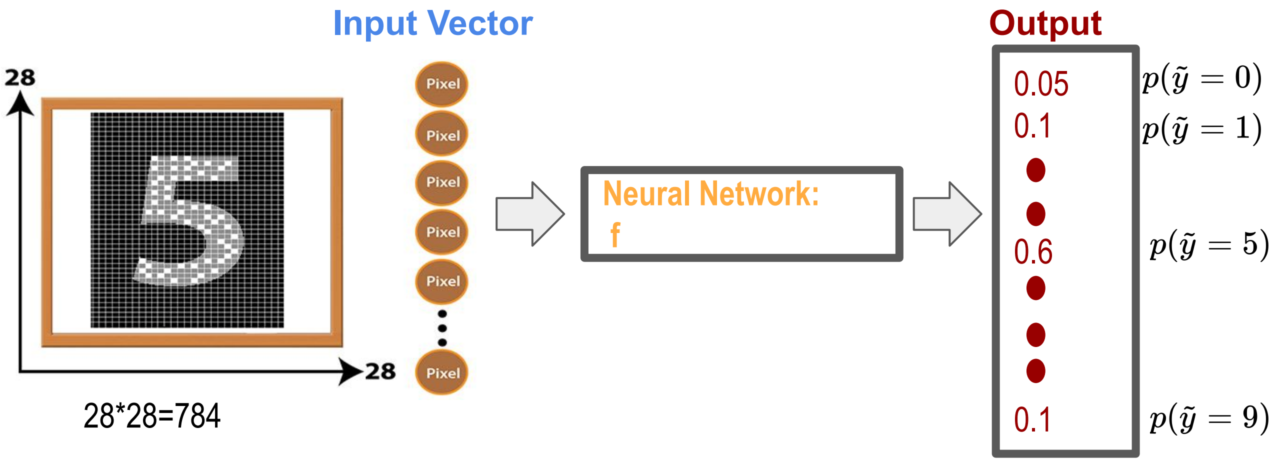
Neural Network: f

Output

0.05	$p(\tilde{y}=0)$
0.1	$p(\tilde{y}=1)$
●	
●	
0.6	$p(\tilde{y}=5)$
●	
●	
●	
0.1	$p(\tilde{y}=9)$

The probability distribution over the ten digits

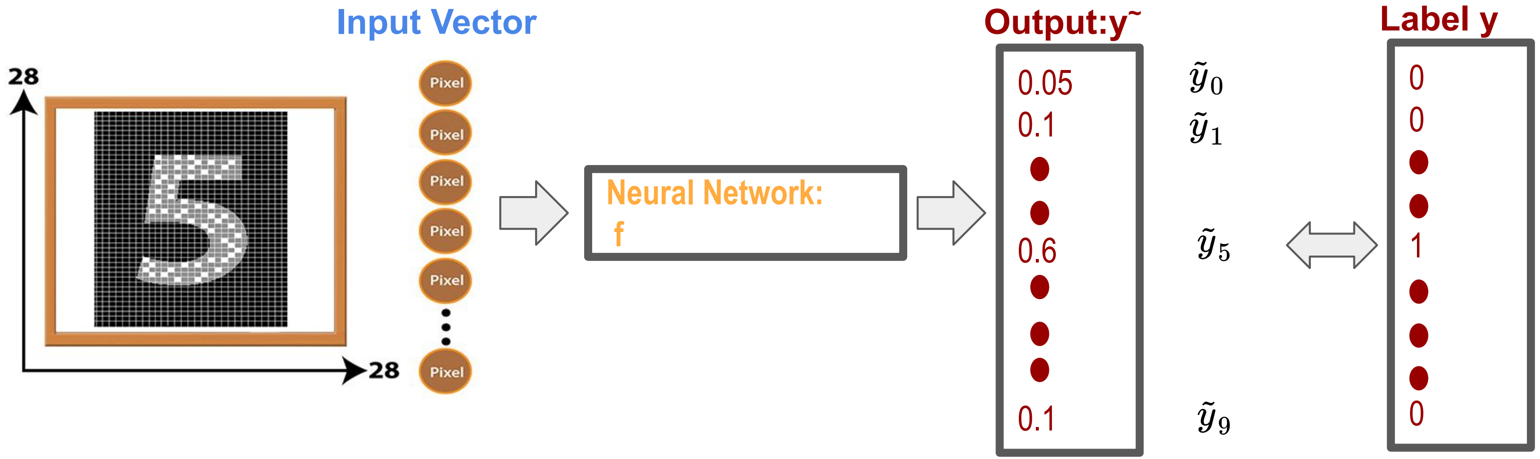
Example: MNIST dataset



1. In this task, the neural network is a function mapping from the input 784-dim vector to the output 10-dim vector.
2. The neural network structure should be decided to make sure the best function exists in the function set.

3. Evaluation of Functions

Cross-Entropy loss



Given a set of parameters and one training sample,

$$loss(\tilde{\mathbf{y}}, \mathbf{y}) = -\sum_{i=0}^{9} y_i ln(\tilde{y}_i)$$

Total loss

- Training dataset contains N training samples
- The total loss is: $J = \sum_{n=1}^{N} loss(\tilde{\mathbf{y}}_n, \mathbf{y}_n)$
- Find a function is the function set that minimizes the total loss J
- Find the network parameters θ that minimizes the total loss J. Modern

For loss function, training data are fixed and model parameters are unknown.

$$argmin_{\theta} J$$

4. Optimization

Gradient descent

Gradient for the total loss function over parameters, which computed by Backpropagation algorithm

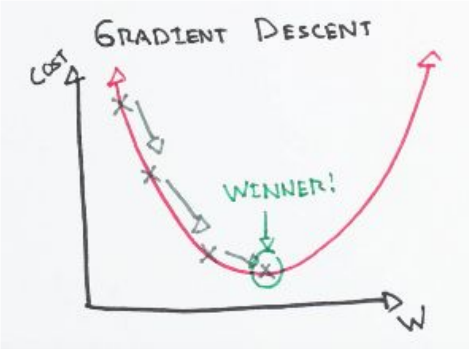
$$\theta_{t+1} = \theta_t - \alpha \boxed{\nabla J(\theta_t)}$$

New Parameters Guess

Current Parameters Guess

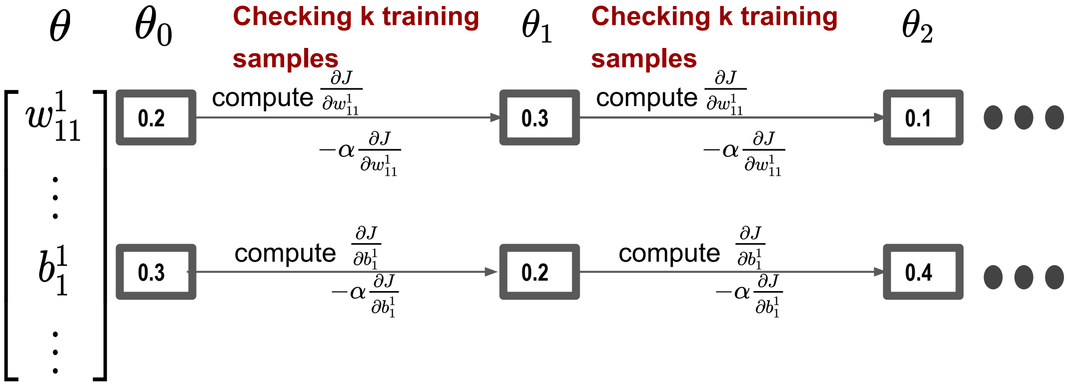
Learning Rate

Like hiking down a mountain



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

Gradient descent



Backpropagation is used to compute gradients in an efficient way. $\frac{\partial J}{\partial \theta}$

Batch size: **k**

A dataset is [1,2,3,4,5,6] and the batch size is 2, one batch shuffle could be: batch0=[2,1], batch1=[3,6], batch2=[4,5]

Backpropagation

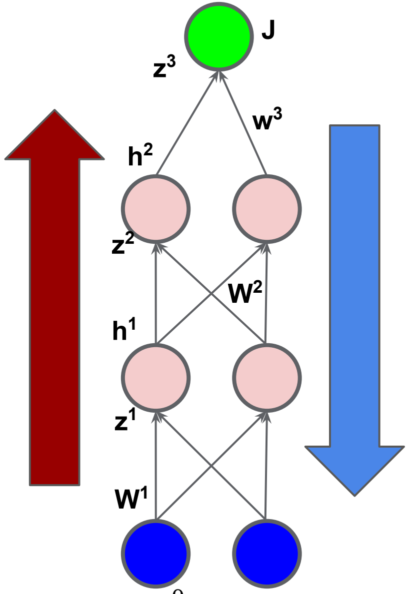
Definition（from wiki):

By computing the gradient of the loss function with respect to each weight by the **chain rule**, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule

Predictions flows forward

$$h^l = \sigma(\mathbf{z}^l)$$

$$\mathbf{z}^l = \mathbf{W}^l h^{l-1} + \mathbf{b}^l$$



$$\mathbf{x} = \mathbf{h}^0$$

Error flows backward

$$\frac{\partial J}{\partial \mathbf{w}^3} = \frac{\partial J}{\partial \mathbf{z}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{w}^3}$$

$$\frac{\partial J}{\partial \mathbf{W}^2} = \underbrace{\frac{\partial J}{\partial \mathbf{z}^3}}_{\text{From } \mathbf{w}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{z}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{W}^2}$$

$$\frac{\partial J}{\partial \mathbf{W}^1} = \underbrace{\frac{\partial J}{\partial \mathbf{z}^3}}_{\text{From } \mathbf{w}^3} \underbrace{\frac{\partial \mathbf{z}^3}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{z}^2}}_{\text{From } \mathbf{W}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{h}^1} \frac{\partial \mathbf{h}^1}{\partial \mathbf{z}^1} \frac{\partial \mathbf{z}^1}{\partial \mathbf{W}^1}$$

More details:
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Batch size

Three approaches to select batch sizes:

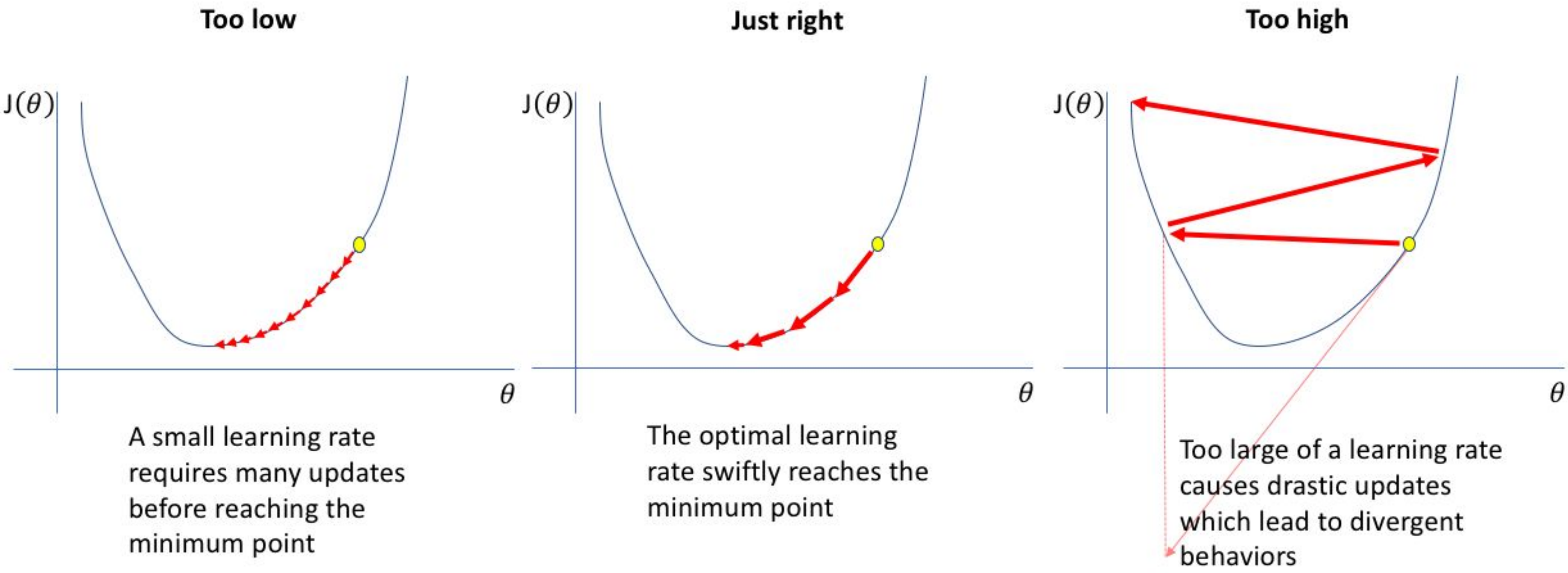
1. Batch Gradient Descent — ***batch size*** *= Number of training data*
2. Mini-batch Gradient Descent — 1<***batch size***< *number of training data*
3. Stochastic Gradient Descent — ***batch size*** *= 1*

Training process

- Initialize neural network randomly
- For _ in range(number of epoch):
    - Shuffle all the training dataset into a list of batches
    - For _ in range(number of batches)
        - Get output with the input data in the batch
        - Compare outputs with ground truth in training data
        - Compute loss function with the batch data
        - Update weights with backpropagation and gradient descent algorithm

Iteratively perform

Learning rate



https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Except SGD

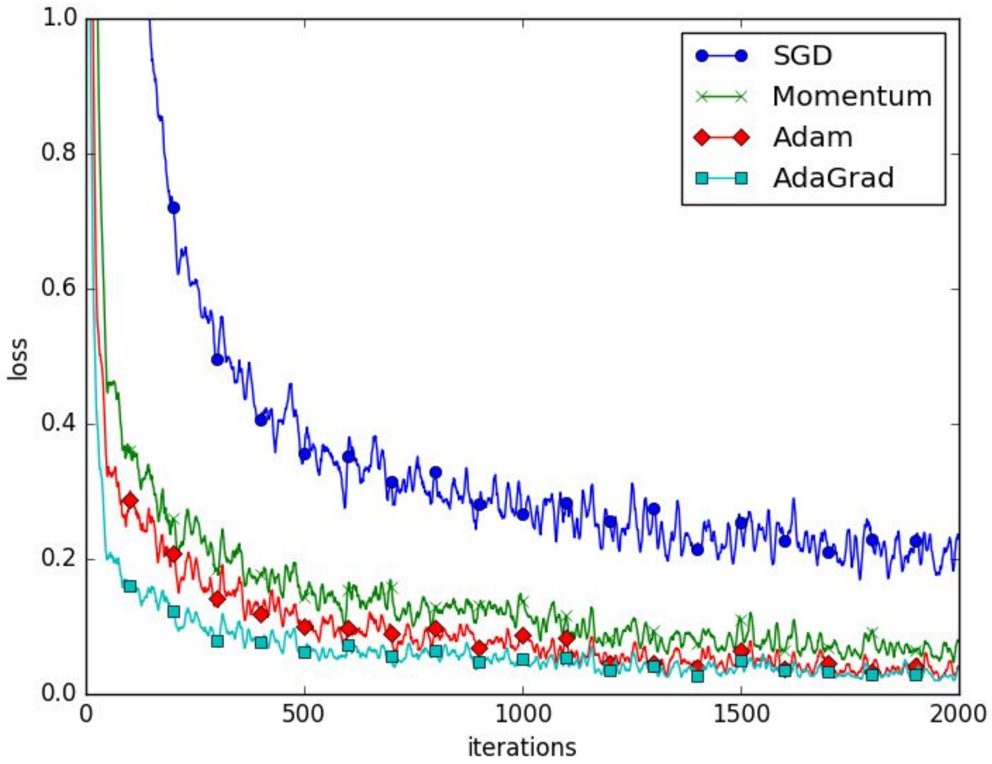
SGD

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \bigtriangledown f(\mathbf{x}_n)$$

Different Variants

Momentum, Adam, AdaGrad, RMSProp

Auto-tune learning rates



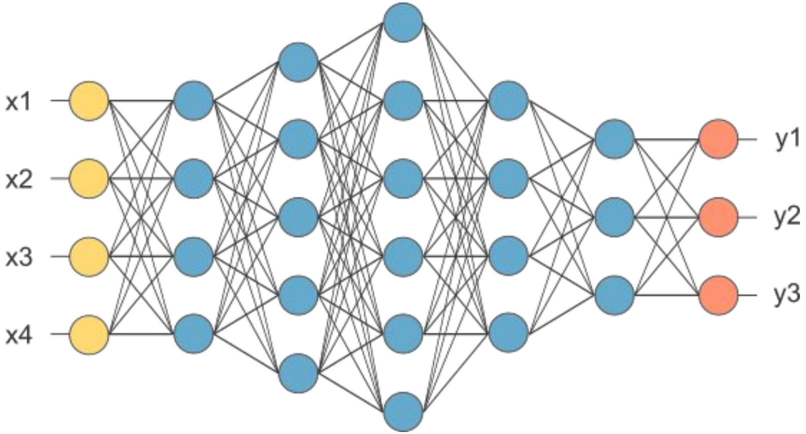
Neural network visualization

[Playground]

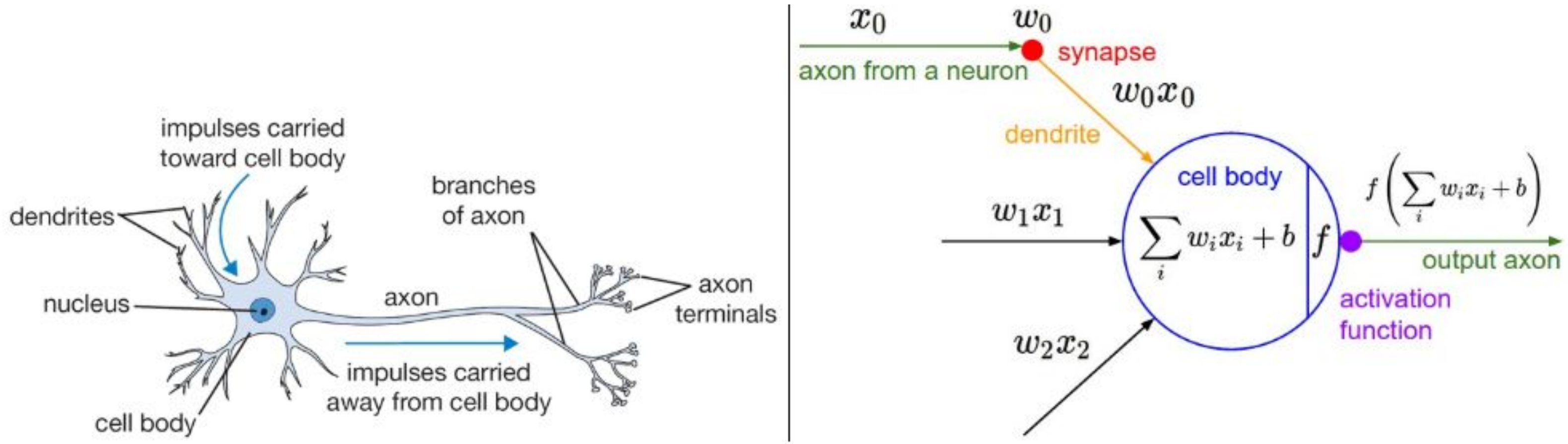
5. Deep Representation Learning

Neural network

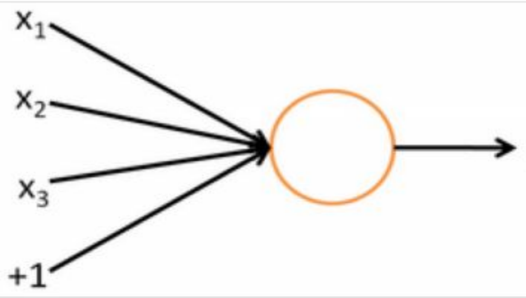
- From Wiki:
  - NN is based on a collection of connected units of nodes called artificial neurons which loosely model the neurons in a biological brain.
- From another way:
  - NN is running several ‘logistic regression’ at the same time (expanding at width and depth dimensions).



Neural computation



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

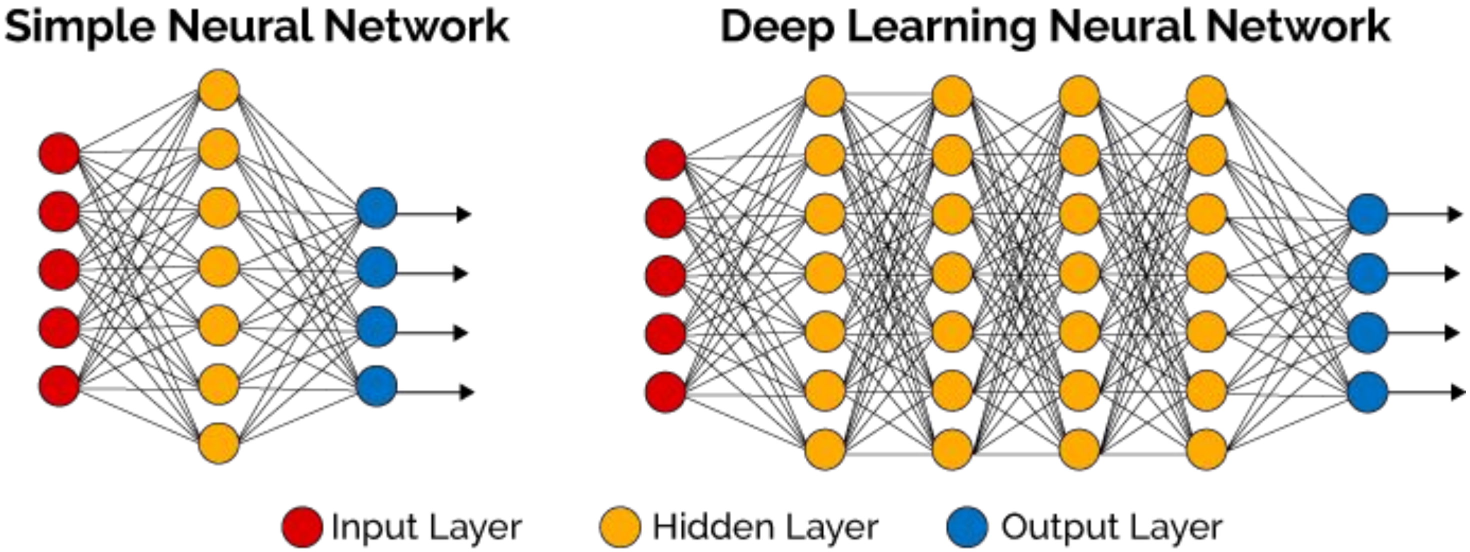


The fact that a neuron is essentially a logistic regression unit:

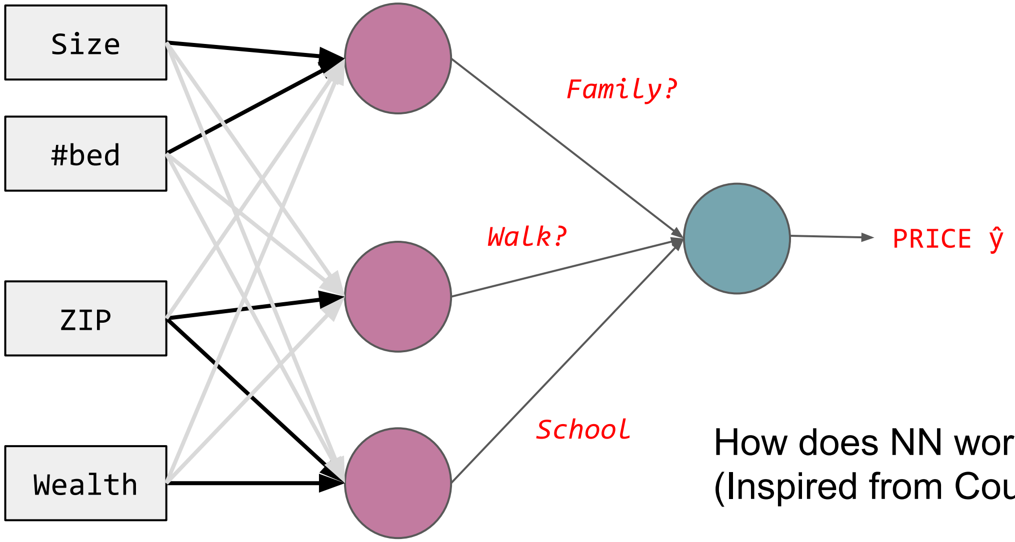
1 performs a dot product with the input and its weights

2 adds the bias and apply the non-linearity

Shallow vs Deep

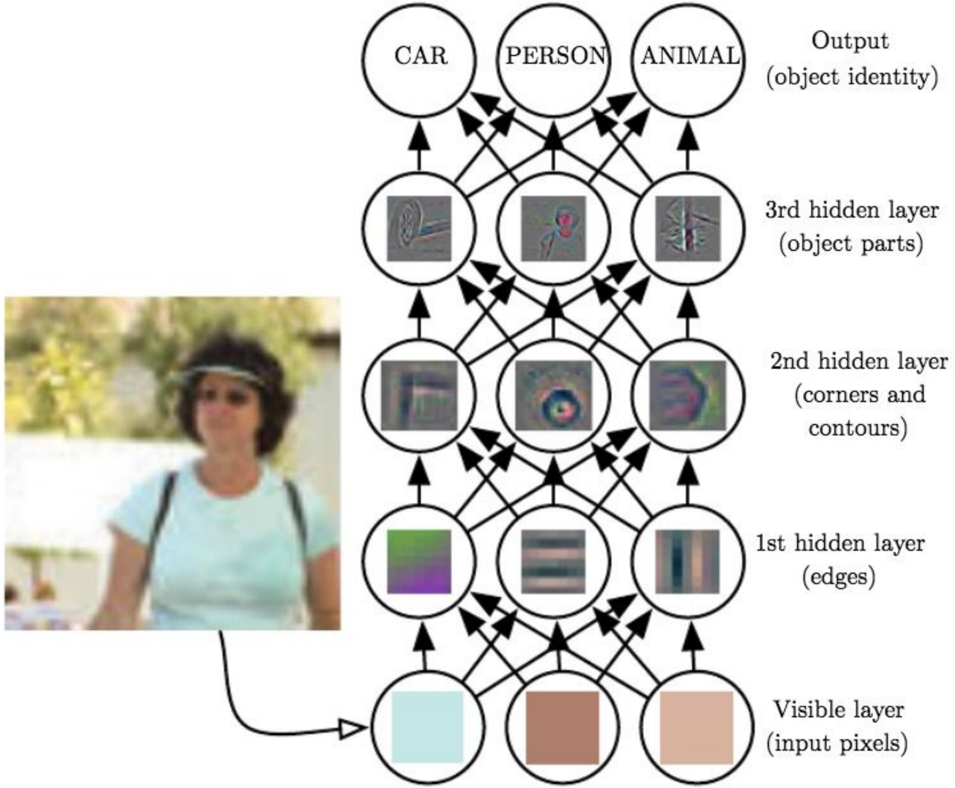


Representation learning in DL



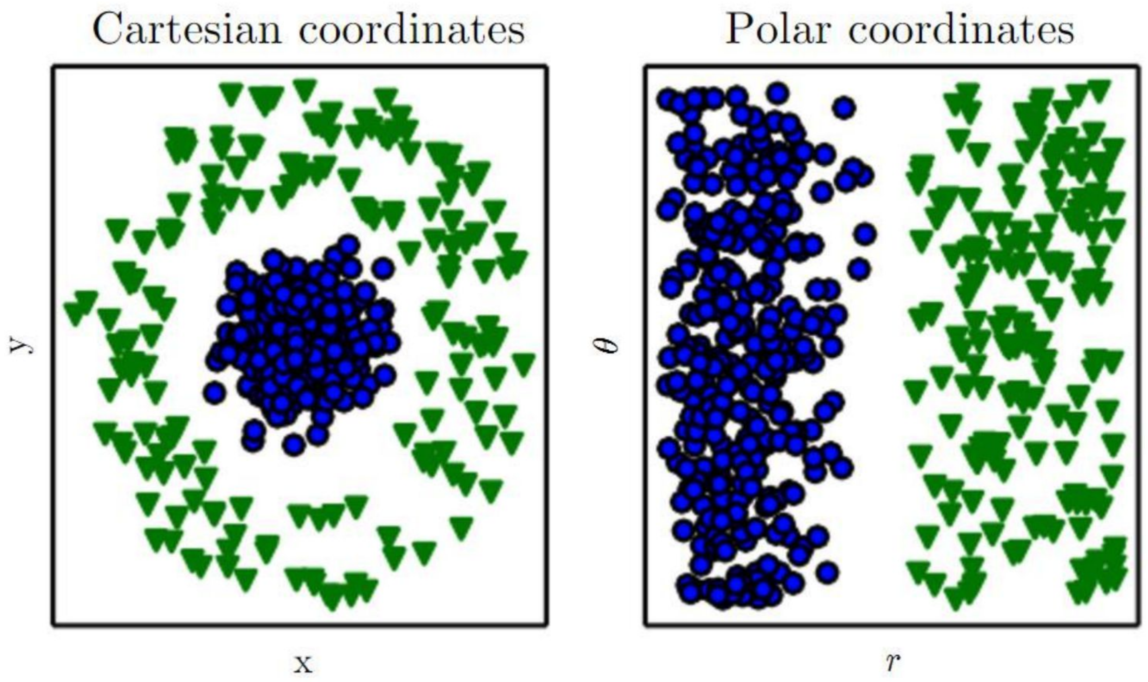
How does NN work
(Inspired from Coursera)

Representation learning in DL

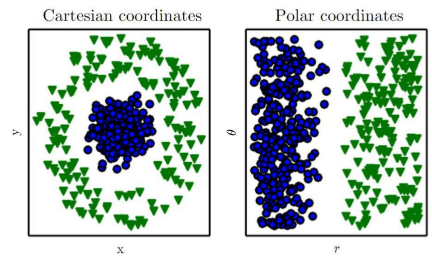


From Deep Learning (Goodfellow)

Representation matters

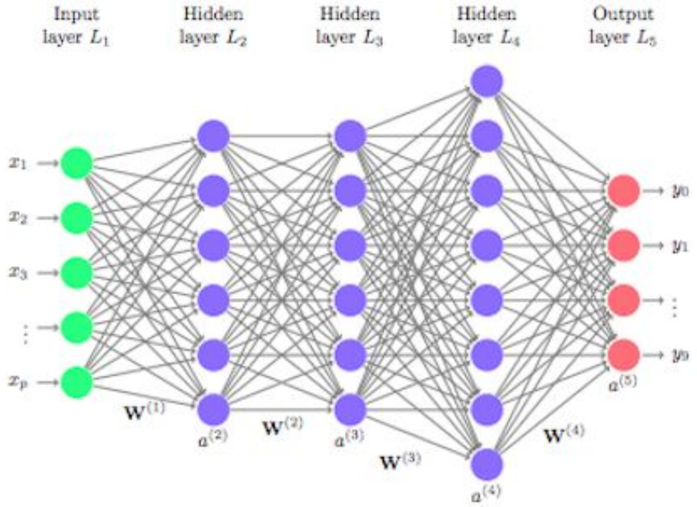


Task: Draw a line to separate the **green triangles** and **blue circles**.

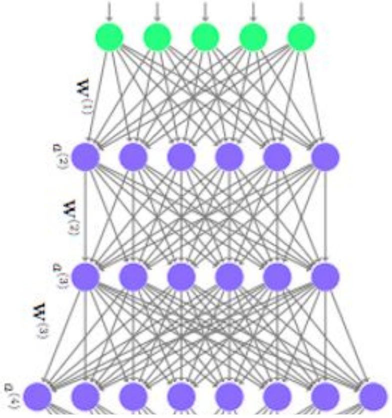


We want to project the data into the new feature/vector space that data is linearly separated

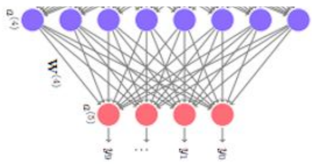
Hidden representation in deep learning



Low-dim, Original Space

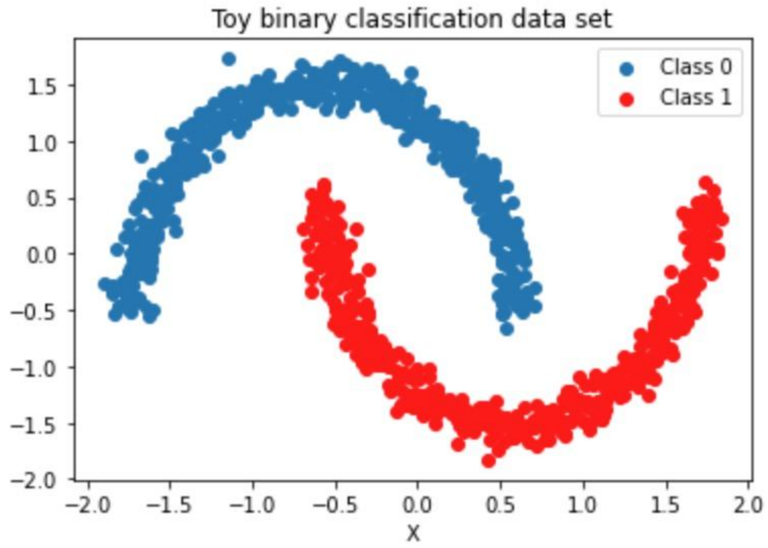


High-dim, **Linearly Separated** Space



Softmax Classifier
(Linear Model)

Moons Dataset




```
# fit a logistic regression model to classify this data set as a benchmark
simple_model = LogisticRegression()
simple_model.fit(X_train, Y_train)
print('Train accuracy:', simple_model.score(X_train, Y_train))
print('Test accuracy:', simple_model.score(X_test, Y_test))
```

```
Train accuracy: 0.89
Test accuracy: 0.88
```

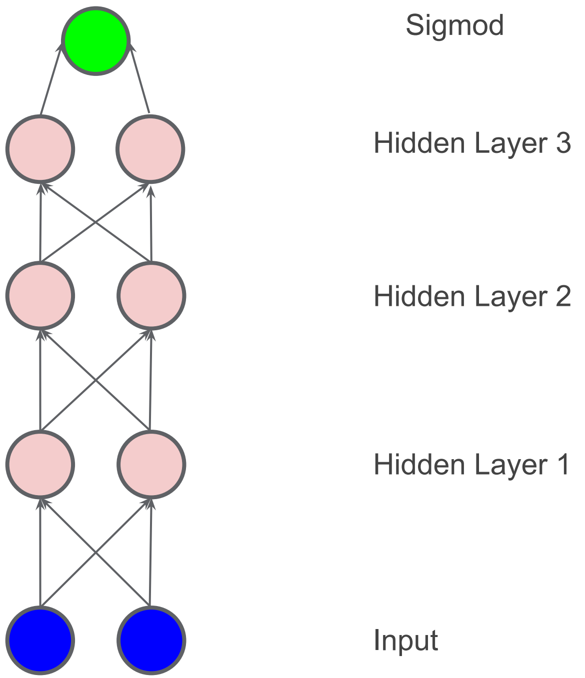
Fully-Connected Neural Network

```
# fix a width that is suited for visualizing the output of hidden layers
H = 2
input_dim = X.shape[1]

# create sequential multi-layer perceptron
model = Sequential()

# Then, use add() to insert layers into the container
model.add(Input(shape=(input_dim,)))
model.add(Dense(H,activation='tanh'))
model.add(Dense(H, activation='tanh'))
model.add(Dense(H, activation='tanh'))
#binary classification, one output
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              metrics=['acc'])
```



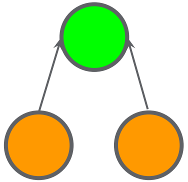
Fully-Connected Neural Network

```
# evaluate the training and testing performance of your model
# note: you should extract check both the loss function and your evaluation metric
score = model.evaluate(X_train, Y_train, verbose=0)
print('Train loss:', score[0])
print('Train accuracy:', score[1])
```

```
Train loss: 0.0007340409210883081
Train accuracy: 1.0
```

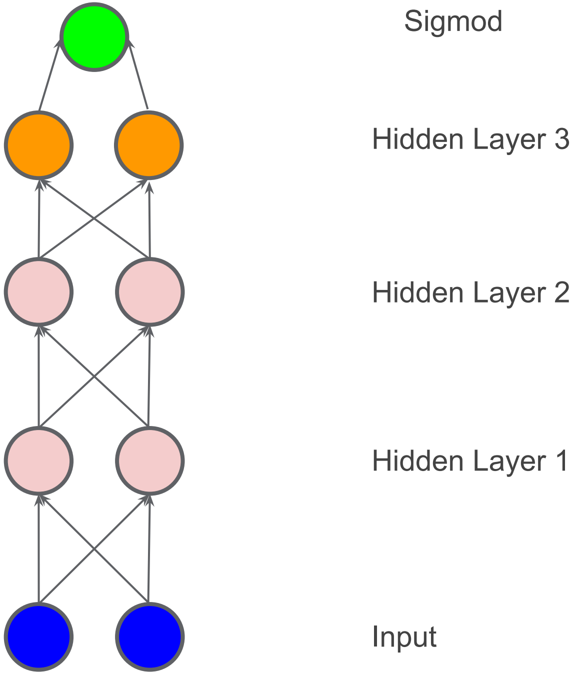
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.0008793871384114027
Test accuracy: 1.0
```

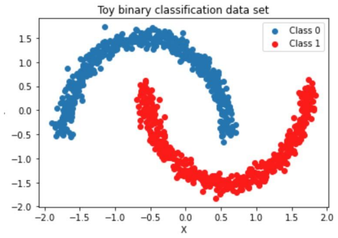


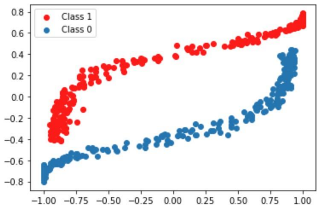
1. In forward computation, the output of hidden layer 3 is feed into “logistic regression” to predict labels.
2. Since the train and test accuracy are both 1, it means the hidden layer 3’ output are linearly separated.

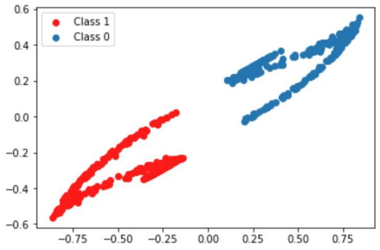
Let us visualize those outputs!

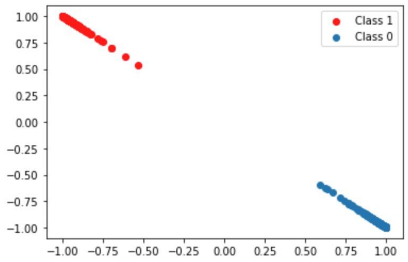


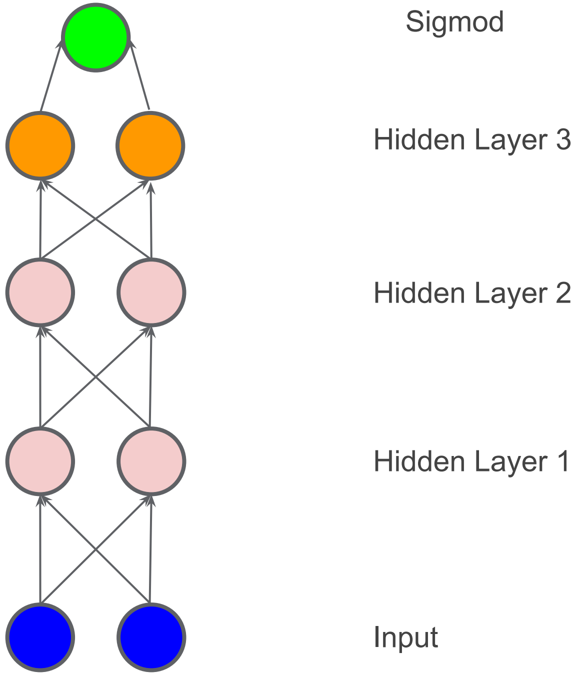
Fully-Connected Neural Network







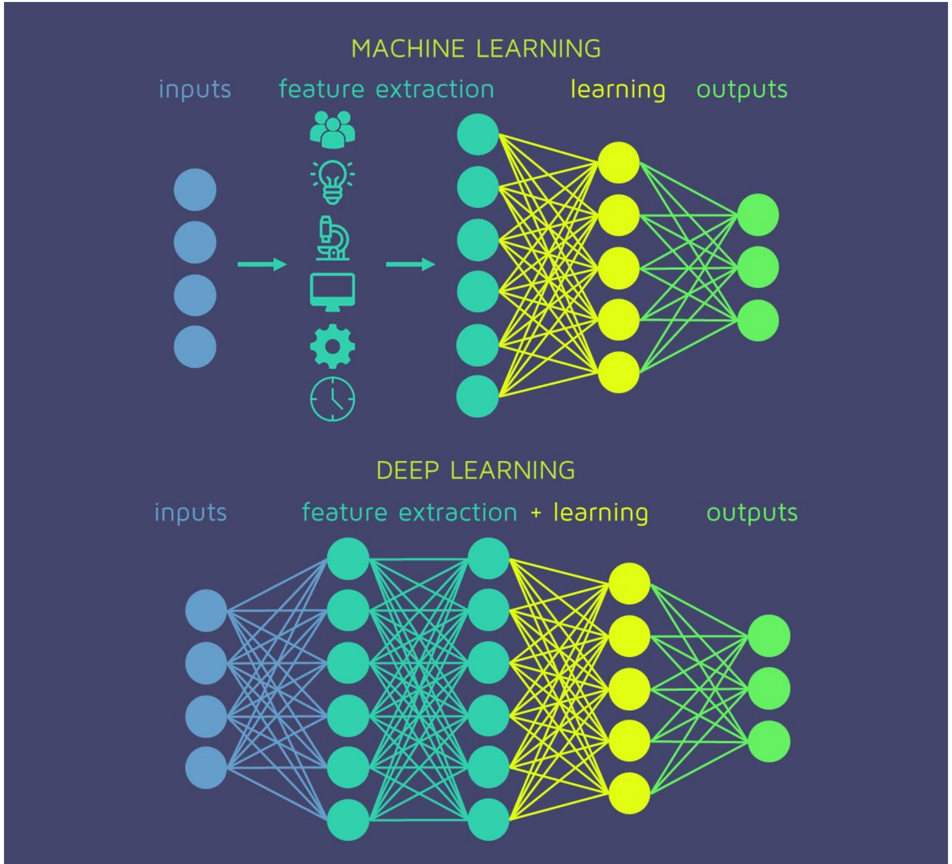




Representation Learning

https://github.com/rz0718/BT5153_2023/blob/main/codes/lab_lecture03/Representation_Learning.ipynb

End-to-end learning

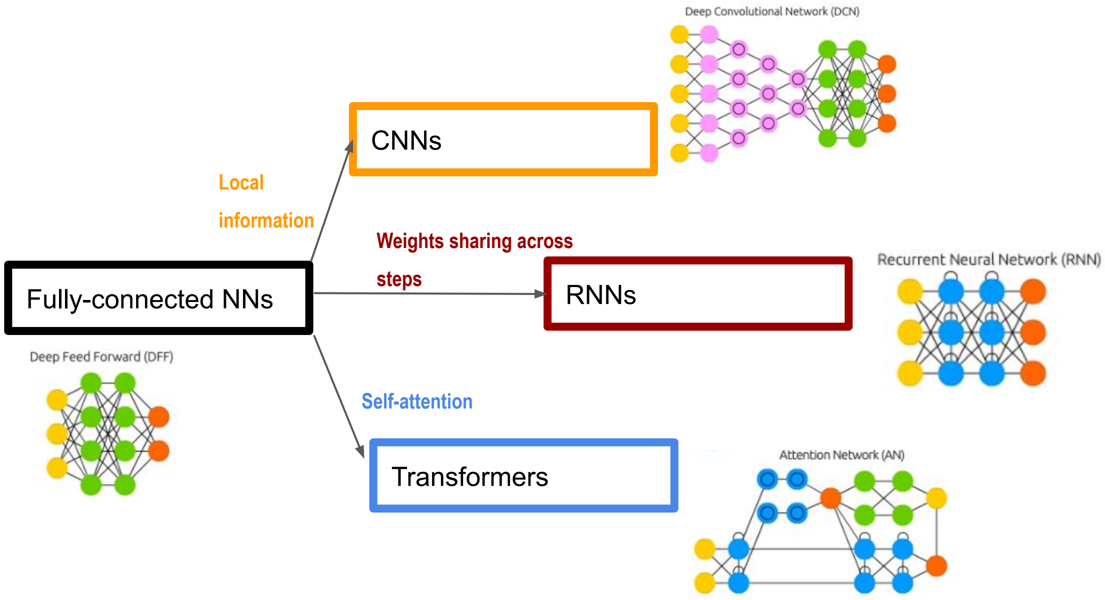


From Aporras

Representation Learning in Neural Networks

- Outputs of each hidden layer of an neural network is a non-linear transformation of the input data into a feature space. Each hidden layer should transform the input so that it is more linearly separable
- we are more interested in learning the latent representation of the data rather than perfecting our performance in a single task (such as classification).
  - We do not need to preprocess the data to add non-linear features. The neural network will learn the most suitable non-linear transformations to the input (to achieve the best classification)

Deep learning structures

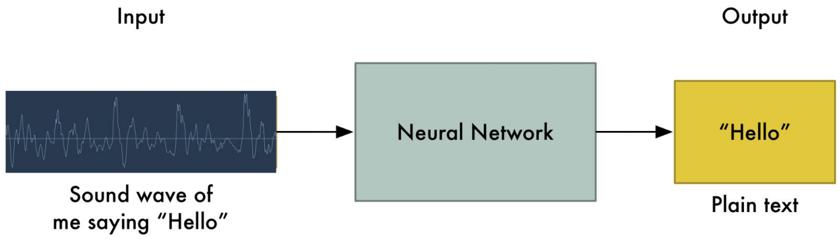


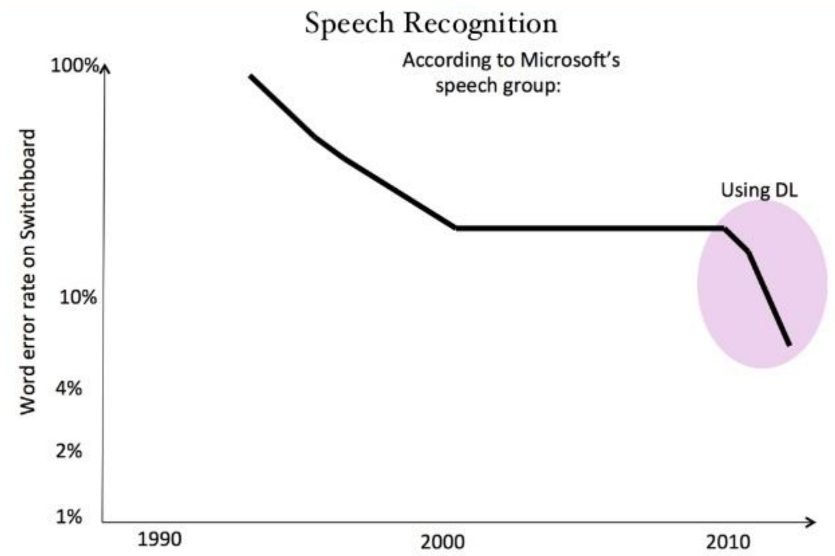
https://www.asimovinstitute.org/author/fjodorvanveen/

6. Applications of DL

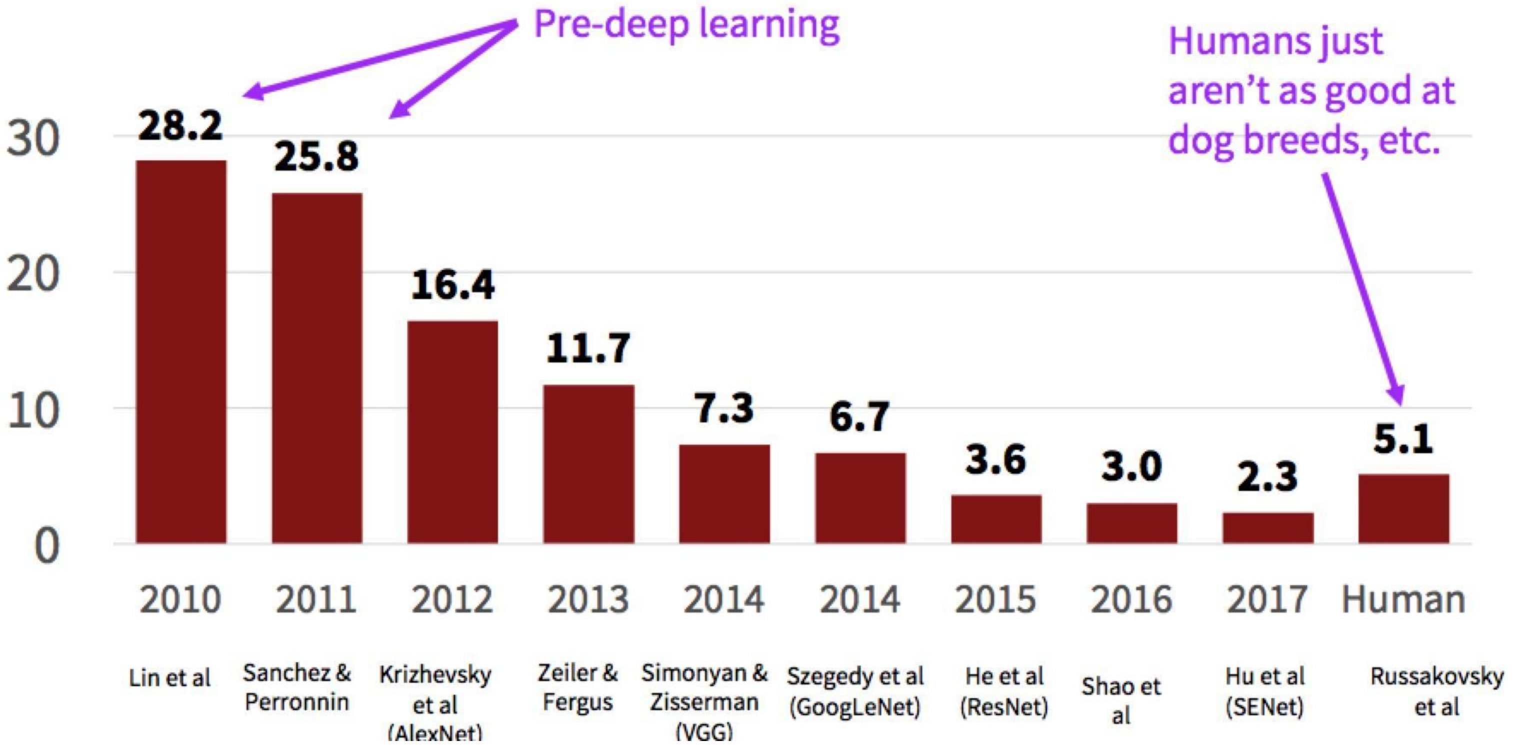
Deep learning for speech

The first real-world tasks addressed by deep learning is speech recognition

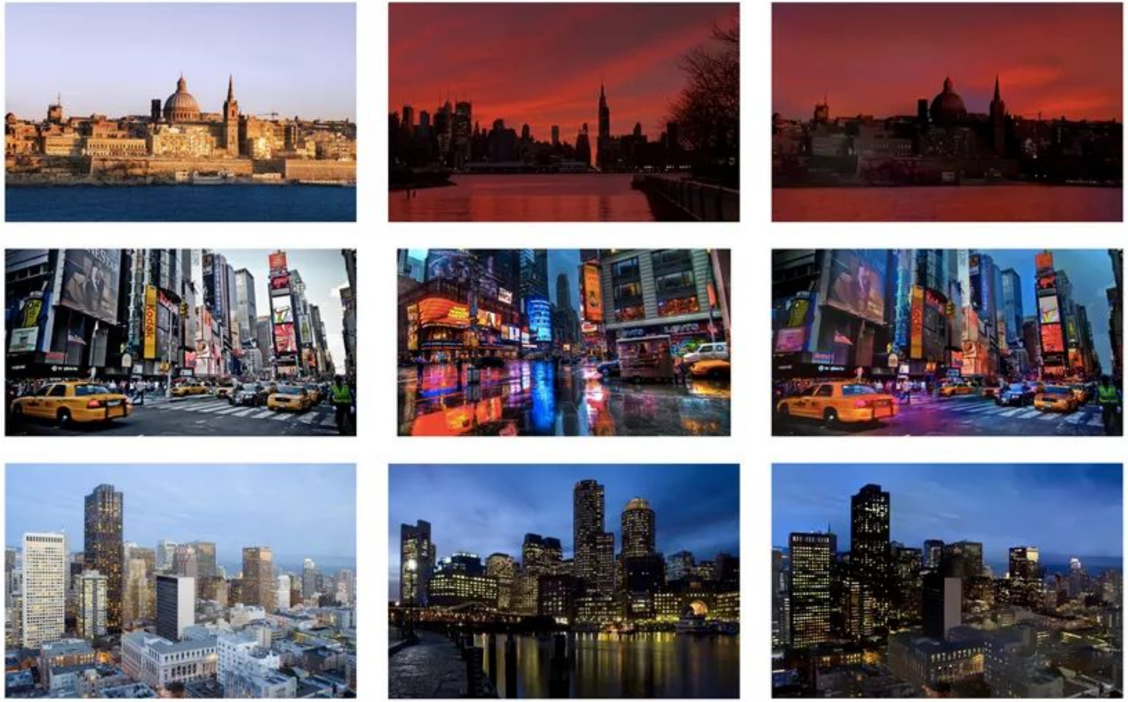




Deep learning for computer vision



Deep learning for arts

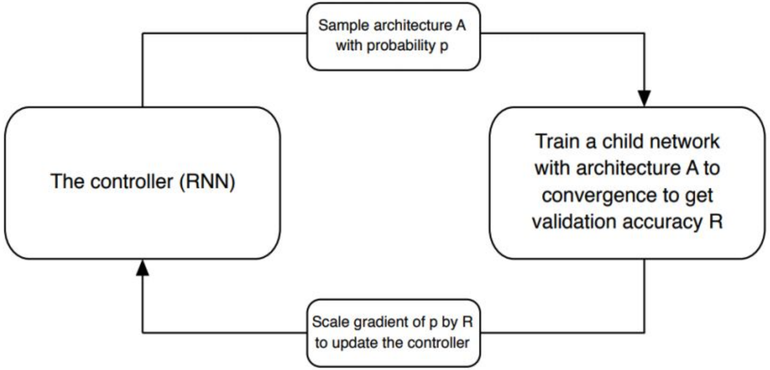


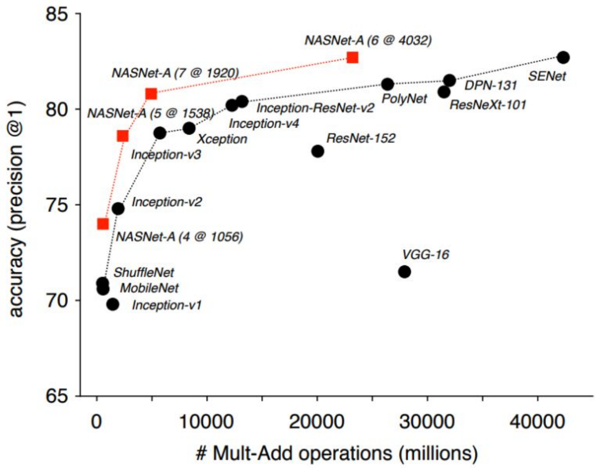
Original photo Reference photo Result

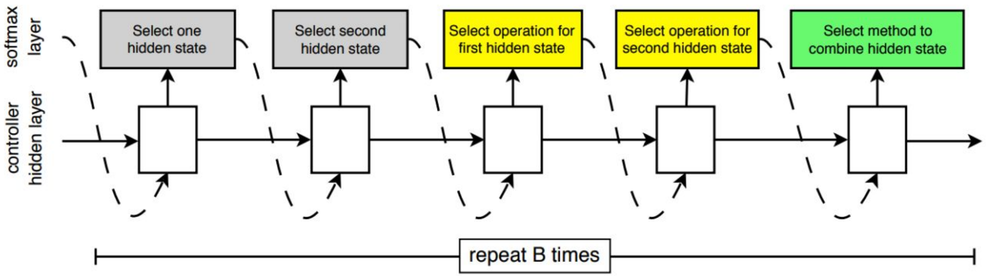
Deep learning for data generations

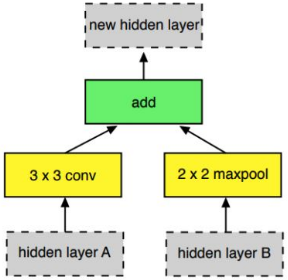
- Given training data, generate new samples from same distribution

AutoML and neural architecture search









Source:Lex Fridman

Next Class: Deep Learning Practice