Artificial Neural Network (ANN)

Saturday, December 29, 2018 15:43

For the glory of God

Introduction

- You may be familiar with the word 'Deep Learning'. If not, I would encourage you to take a look what it is from any websites if you are studying in Engineering.
- Over the post 5 years or so, deep learning has sky-rocketed from many research groups.
- Deep learning research now routinely appears in everywhere like as science, nature, engineering, and so forth.
- For example, you may hear about AlphaGo vs. Lee sedol, a.k.a Google DeepMind Challenge Match.

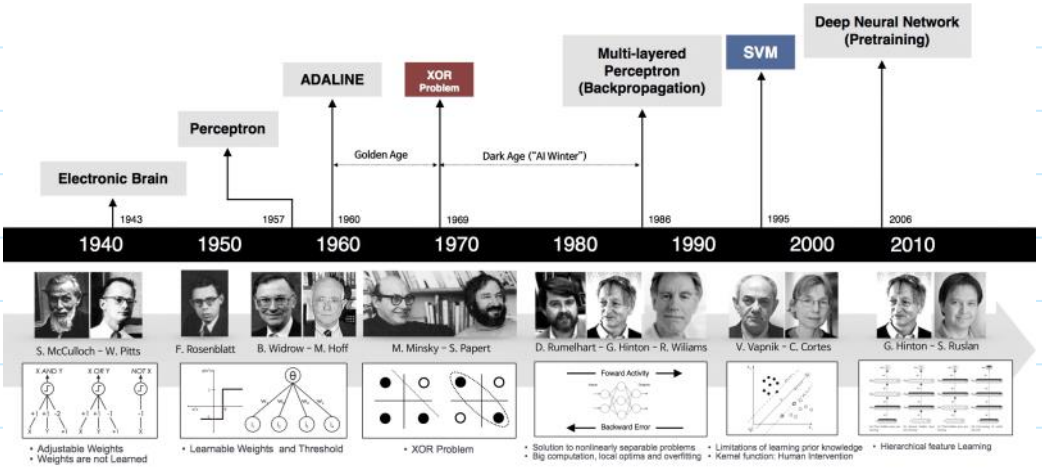
  - It was a five-game match between 18-time world champion Lee and AlphaGo program developed by Google Deepmind.
  - The result was astonishing the world ; the deep learning program won all but the fourth game.
- Deep learning is actually based on the theory of **Artificial Neural Network**.

  -> ANN is inspired by the structure of the human brain and constructed by complex connections between neurons.
- So, it seems that the main idea of deep learning has been in place for decades because ANN was a quite popular idea.
- **However**, it is worth walking through the history of neural nets and deep learning to see how we got here.

  ↳ This is one of reasons why I decided to summarize this hand-written note.

History of Artificial Neural Network

- Before we dive into details about ANN and deep learning, let us take a look milestones in the development of ANN ;



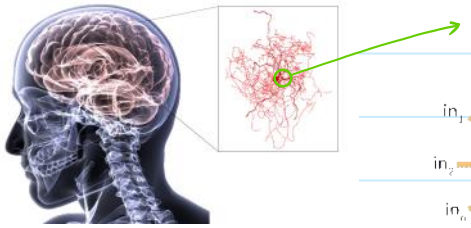
· From now on, we will start our journey to visit each development phase. So, buckle up !

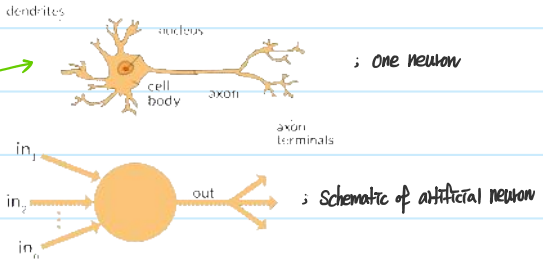
Neurons in the brain

· In fact, in order to understand ANN, the first thing that we have to understand is how our brain works.

↳ Again, this is because ANN is based on the theory of neurons in our brain. (praise the Lord !)

· Okay, let us study how our brain works then.





; one neuron

; Schematic of artificial neuron

- As can be seen, this is a structure of human brain with numerous neurons.
- It is known that the average brain has about 100 billion neurons.
- The process is as following ;

1) Signals are received from dendrites. (The signals are coming from other neurons)

2) Nucleus makes a summation for all incoming signals.

3) It makes a decision if the signal (Σ) is sent down the axon.

↳ The decision depends on the signal strength.

4) If the signal is enough, axon reacts. e.g. signal = see the car via eyes, reaction = recognize the car via brain

5) This outgoing signal is then used as another input for other neurons.

Artificial neuron

· We have talked about how neurons in the brain works.

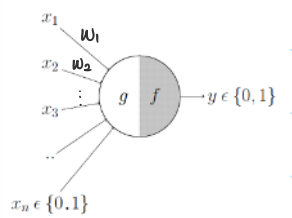
· We could actually mimic most of this process by coming up with ;

- A function that receives a list of weighted input signals and output signal if the sum of these weighted inputs reach a certain value.

· This idea was first invented by S. McCulloch and W. Pitts in 1943.

↳ It was the first mankind's mathematical model of a biological neuron.

· The main idea is as follows ;



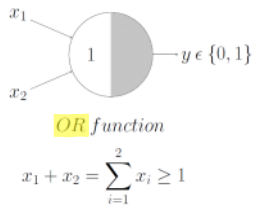
- There are inputs with the weights. (The weights are adjustable ; however, it's fixed once it is determined)

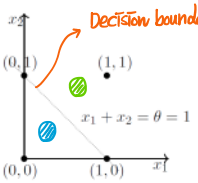
- g takes the inputs and performs aggregation with the weights.

- f make a decision { if the summation is greater than a threshold value, it returns y=1
  otherwise, y=0

· This idea could be used as a linear classifier.

· For example, let's say that we have boolean input which has four possible combinations. (0,0), (0,1), (1,0), (1,1)





all points in the green : output = 1

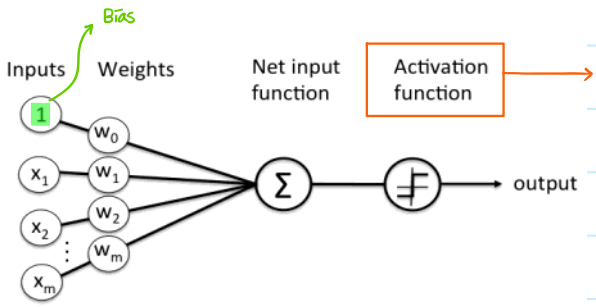
all points in the blue : output = 0

; Note that the weights are not learned but adjustable.

Perception

· In 1958, Frank Rosenblatt proposed 'Perception' which is more generalized computational model than the Artificial neuron where weights and thresholds can be learned over time. (His paper stunned the world)

· Let us see how it works.



This is a difference with Artificial neuron.

(It can be tanh, sigmoid, and so forth)

Schematic of Rosenblatt's perceptron.

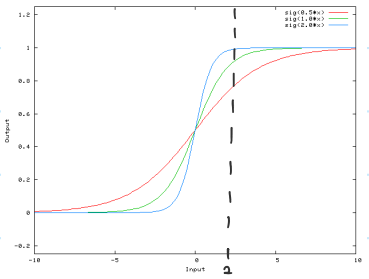
- There are inputs with a bias. (Here, please note that a single bias node is needed per layer)

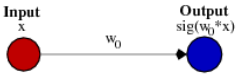
↳ Why do we need this? Actually, bias are almost always helpful. How come?

· It allows to shift the activation function to the left or right.

· This actually helps for successful learning.

· Let us think about a case without a bias; let's say $y = \text{sigmoid}(w \cdot x)$



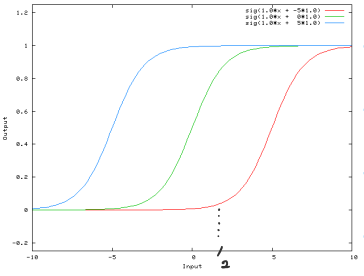


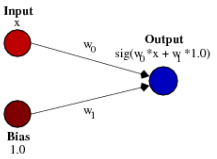
- What if we want the network to have output 0 when $x = 2$?

- Changing the steepness of the sigmoid function will not work.

- Changing the weight essentially changes the steepness only.

· We might want to shift the curve to satisfy the condition.





- Initialize the weights.

- Make a summation of all inputs and initialized weights.

- Use the result as an input for activation function.

↳ We mentioned that this is a difference between Artificial neural and perception.

Then, what is the significance of the activation function ?

- If we do not apply it, then the output signal would be a simple linear function.

: It has less power to learn complex functional mappings from data.

What are Activation functions and what are it uses in a Neural Network Model?

Activation** functions* are really important for a Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable. *They introduce non-linear properties to our Network.* ***Their main purpose is to convert a input signal of a node in a A-NN to an output signal. That output signal now is used as a input in the next layer in the stack.

- Calculate the output and compare it with the real output value.
- If the residual (real - predicted) is larger than ϵ ; convergence criteria, update the weights.
- Repeat the process until it is converged.

$$W_{next}(i) = W_{current}(i) + \alpha X_i (t - f_{net})$$

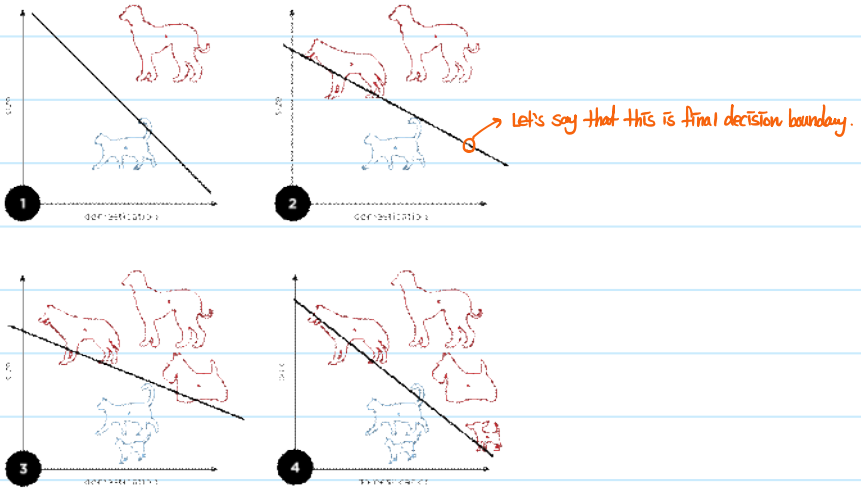
; where α = learning rate

t = target value

f_{net} = prediction

- Let us take an example : we want to classify whether it is cat or dog using two datasets.

↳ The basic idea is to generate a linear classifier by updating weights.



Let's say that this is final decision boundary.

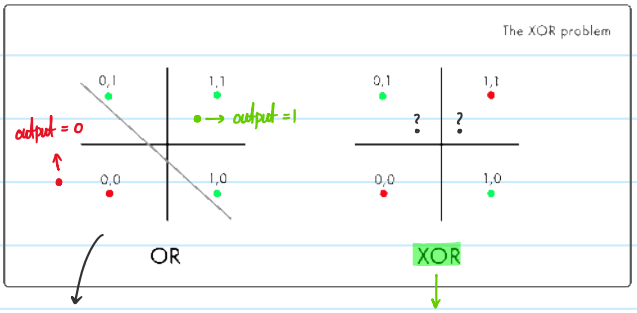
- After the perceptron was released to the world, the golden age of ANN started until...

XOR Problem

- Unfortunately, the Perceptron's popularity rapidly decreased as Marvin Minsky and Seymour papert published their paper in 1969.

· They pointed out that the perception is just a linear classifier and it's not able to classify XOR problems, which was mathematically validated in the paper.

↳ This invoked the dark age, namely AI winter.

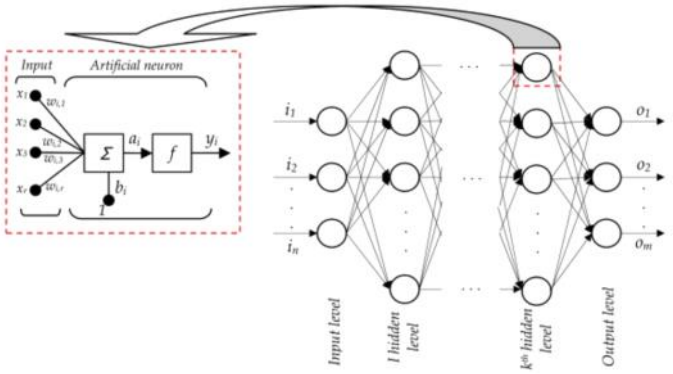


e.g. $x_1 + x_2 \geq 1$: output = 1 For this case, it's impossible to classify them with only one linear line.

Multi-layer Perceptions

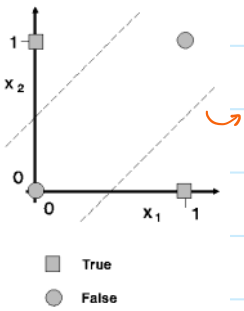
· After the AI winter, David E. Rumelhart and Geoffrey E. Hinton proposed an idea 'Multi-layer perceptions' in 1986 in order to resolve the XOR problem.

↳ They had actually shown that it is capable of approximating an XOR operator as well as many other non-linear functions by introducing the following structure.



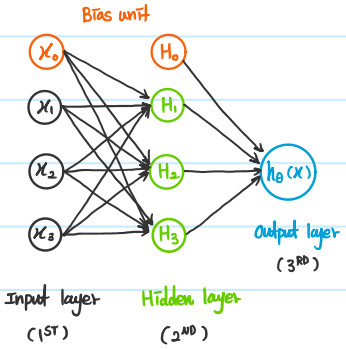
- As can be seen, a MLP is composed of more than one perception.
- They are composed of
  a) Input layer to receive the signal
  b) Output layer that makes a decision
  c) Hidden layer that is the true computational engine of the MLP

· With the MLP, the XOR problem was resolved by like drawing multiple decision boundaries :



↳ two linear classifiers

· Then, how does it work? The process is called as Forward Propagation.



; where Bias unit

↳ It depends on a problem (Not necessary to add on the model)

Here,

$$H_1^{(2)} = g(\Theta_{10}^{(1)}X_0 + \Theta_{11}^{(1)}X_1 + \Theta_{12}^{(1)}X_2 + \Theta_{13}^{(1)}X_3)$$

$$H_2^{(2)} = g(\Theta_{20}^{(1)}X_0 + \Theta_{21}^{(1)}X_1 + \Theta_{22}^{(1)}X_2 + \Theta_{23}^{(1)}X_3)$$

$$H_3^{(2)} = g(\Theta_{30}^{(1)}X_0 + \Theta_{31}^{(1)}X_1 + \Theta_{32}^{(1)}X_2 + \Theta_{33}^{(1)}X_3)$$

; where $H_i^{(j)}$ = activation of unit i in layer j

$\Theta^{(j)}$ = Matrix of weights controlling function mapping from layer j to layer $j+1$

Hence,

$$h_\theta(x) = g(\Theta_{10}^{(2)}H_0^{(2)} + \Theta_{11}^{(2)}H_1^{(2)} + \Theta_{12}^{(2)}H_2^{(2)} + \Theta_{13}^{(2)}H_3^{(2)})$$

; where g can be { Sigmoid function (= Logistic function), $g(x) = \frac{1}{1+e^{-\theta^T x}}$

Gaussian function, $g(x) = e^{-x^2}$

TanH function : scaled version of the sigmoid function, $g(x) = \frac{e^{2x}-1}{e^{2x}+1}$

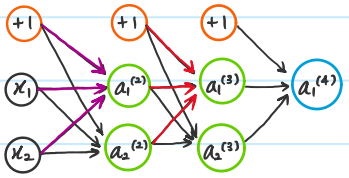
· Now, let's look at the procedure.

① Randomly initialize weights

- Initialize each $\Theta_{ij}^{(L)}$ to a random value in $[-\epsilon, \epsilon]$
- Note that the initialization can't be zero vector for neural network because hidden units become identical if zero initialization.

② Implement forward propagation to get $h_\theta(x^{(i)})$ for any $x^{(i)}$

- Let's take a look a diagram, so that we'll be able to figure out how it works.



Here,

$$h_\theta(x) = a_1^{(4)} = g(\theta_{10}^{(3)} \cdot 1 + \theta_{11}^{(3)} a_1^{(3)} + \theta_{12}^{(3)} a_2^{(3)})$$

"Hypothesis function"

$$a_1^{(3)} = g(\theta_{10}^{(2)} \cdot 1 + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)})$$

$$a_1^{(2)} = g(\theta_{10}^{(1)} \cdot 1 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2)$$

In doing so, recall that;

- We know the weights via initialization ($\theta_{ij}^{(l)}$ matrix)
- We know the input values (x_1 and x_2)
- Then, we can calculate $h_\theta(x)$ → Iterate the process with different training set ($x^{(i)}$)

③ Compute the cost function, $J(\theta)$

- We can measure the accuracy of our hypothesis function by using a cost function.
  ↔ Squared error function
- In general,

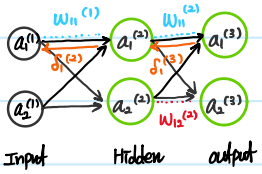
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$; which means difference between the predicted value and the actual value

  - $\hat{y}_i$ = the predicted value computed from forward propagation
  - y_i = the actual value given from output

Backpropagation algorithm

- As we discussed about forward propagation, it seemed that it resolves the problem by introducing MLP.
- However, it also had the problem that it's really hard to estimate weights and bias when the MLP becomes complex.
  - Eventually, we might want to solve non-linear complex problems.
  - The more complex, the more hidden layers and parameters.
  - It turned out that it's really challenging to achieve the goal that estimates weights and bias; such that we might be able to figure out the relationship between input and output signals.
- Again in 1986, they (Rumelhart, Hinton, and Williams) proposed the idea 'Backpropagation' to solve this problem.
- Then, how does it work? Let us take a simple example.

↳ Let's start with the diagram as below. (assume that learning rate = 1)



- First of all, let's say we've done the forward propagation. Suppose that we define an error with MSE.

$$J_1 = \frac{1}{2}(a_1^{(3)} - y)^2$$

$$J_2 = \frac{1}{2}(a_2^{(3)} - y)^2$$

; Note that we used initialized weight parameters.

- Second, we want to update the weight parameters using Backpropagation such that the error is minimized.

$$W_{j,\text{next}} = W_{j,\text{previous}} - \alpha \frac{d J_{\text{total}}}{d W_j}$$; where α = learning rate

- As a part of example, let's update $W_{11}^{(2)}$.

$$W_{11}^{(2),\text{new}} = W_{11}^{(2),\text{old}} - \frac{d J_1}{d W_{11}^{(2)}}$$; Note that α is assumed to be equal to 1.

→ Note that $a_1^{(3)}$ doesn't propagate errors to anywhere

- As can be seen, it feels like that we need to calculate the derivative.

$$\frac{d J_{\text{total}}}{d W_{11}^{(2)}} = \frac{d J_1}{d a_1^{(3)}} \times \frac{d a_1^{(3)}}{d z_1^{(3)}} \times \frac{d z_1^{(3)}}{d W_{11}^{(2)}}$$; where $z_1^{(3)} = W_{11} a_1^{(2)} + b$ ~ not necessary for this case.

$$= (a_1^{(3)} - y_1) \times \left[\phi(z_1^{(3)})(1 - \phi(z_1^{(3)})) \right] \times a_1^{(2)}$$

How come ?

- $a_1^{(3)} = \phi(z_1^{(3)})$; note that I assume the activation function is Sigmoid.

$$= \frac{1}{1 + e^{-z_1^{(3)}}}$$

- $$\frac{d a_1^{(3)}}{d z_1^{(3)}} = -1 (1 + e^{-z_1^{(3)}})^{-2} (-e^{-z_1^{(3)}})$$

$$= \frac{e^{-z_1^{(3)}}}{(1 + e^{-z_1^{(3)}})^2}$$

$$= \frac{1}{1 + e^{-z_1^{(3)}}} \frac{e^{-z_1^{(3)}}}{1 + e^{-z_1^{(3)}}}$$

$$= \phi(z_1^{(3)}) \cdot (1 - \phi(z_1^{(3)}))$$

$$= (a_1^{(3)} - y_1) \times (a_1^{(3)} \cdot (1 - a_1^{(3)})) \times a_1^{(2)}$$

$$= \delta_1^{(3)} a_1^{(2)}$$

; where $\delta_1^{(3)} \equiv (a_1^{(3)} - y_1)(a_1^{(3)}(1 - a_1^{(3)}))$

- Hence, by rearranging the equation.

$W_{11}^{(2)}, new = W_{11}^{(2)}, old - \delta_1^{(3)} a_1^{(2)}$; where $\delta_1^{(3)} = a_1^{(3)} - y$ (It's an error, known)

$\therefore$ we'll be able to update the $W_{11}^{(2)}$

- In a similar way, we have ;

$W_{12}^{(2)}, new = W_{12}^{(2)}, old - \delta_1^{(3)} a_2^{(2)}$

$W_{21}^{(2)}, new = W_{21}^{(2)}, old - \delta_2^{(3)} a_1^{(2)}$

$W_{22}^{(2)}, new = W_{22}^{(2)}, old - \delta_2^{(3)} a_2^{(2)}$

· How about $W_{11}^{(1)}$ heading from hidden to input layer ?

$$W_{11}^{(1)}, new = W_{11}^{(1)}, old - \frac{\partial J_{total}}{\partial W_{11}^{(1)}}$$ → Note that $a_1^{(2)}$ does propagate error to both $a_1^{(3)}$ and $a_2^{(3)}$.

$$= W_{11}^{(1)}, old - \left(\frac{\partial J_{total}}{\partial a_1^{(2)}} \times \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial W_{11}^{(1)}} \right)$$

$$\frac{\partial J_1}{\partial a_1^{(2)}} + \frac{\partial J_2}{\partial a_1^{(2)}} = \frac{\partial J_1}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} + \frac{\partial J_2}{\partial z_2^{(3)}} \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}$$

$$= \delta_1^{(3)} W_{11}^{(2)} + \delta_2^{(3)} W_{21}^{(2)}$$

$$= W_{11}^{(1)}, old - \left\{ \left(\delta_1^{(3)} W_{11}^{(2)} + \delta_2^{(3)} W_{21}^{(2)} \right) \times a_1^{(2)} (1 - a_1^{(2)}) \times a_1^{(1)} \right\}$$

$$= W_{11}^{(1)}, old - \delta_1^{(2)} a_1^{(1)} \quad ; \text{ where } \delta_1^{(2)} \equiv \left(\delta_1^{(3)} W_{11}^{(2)} + \delta_2^{(3)} W_{21}^{(2)} \right) \times a_1^{(2)} (1 - a_1^{(2)})$$

↓ It's like $\delta_j = h'(a_j) \sum_k W_{kj} \delta_k$

· Based on the example, let us generalize the Backpropogation equation.

- Let's say we define an error with MSE.

$$E = \frac{1}{2} \sum_{j=1}^{J} (y_j - \hat{y}_j)^2$$

- Let's say we want to update weight parameters for minimizing the error function.

$$W_{new} = W_{old} - \Delta W_{kj} \quad ; \text{ where } \Delta W_{kj} = -\alpha \frac{\partial E}{\partial W_{kj}}$$

- In order to update it, we should calculate the derivative. Let's use chain rule.

$$\frac{\partial E}{\partial W_{kj}} = \frac{\partial E}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial x_j} \frac{\partial x_j}{\partial W_{kj}} \quad ; \text{ where } \frac{\partial x_j}{\partial W_{kj}} = \hat{y}_k$$

- Let's define the error term.

$$\delta_j \equiv - \frac{\partial E}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial x_j} \quad ; \text{ where } \frac{\partial \hat{y}_j}{\partial x_j} = \hat{y}_j (1 - \hat{y}_j) \text{ if the activation function is sigmoid}$$

- Finally, we use ; $\frac{\partial E}{\partial W_{kj}} = \delta_j \hat{y}_k$; note that δ_j will be different in case of output layer or hidden layer.

- Finally, we use ; $\frac{\partial E}{\partial W_{kj}} = \delta_j \hat{y}_k$; note that δ_j will be different in case of output layer or hidden layer.
- Backpropagation would be done by updating with all weight parameters. (It would be great if we perform gradient checking)
- Forward propagation would be implemented with the new parameters ; and repeat the process until $\varepsilon \simeq 0$

Backpropagation vs. Numerical Estimation

Deep Learning

- The more problems become complex, the more hidden layers and parameters.
- People might think that more hidden layers solve the complex problem as they experienced the power of MLP.

↳ Researchers started to call the structure as Deep neural networks or Deep learning.

e.g. More than two-hidden layers ⇔ Deep learning

- Unlike their expectations, the performance of deep learning was not as good as they expected because of ;
  - Vanishing gradient problem
  - Overfitting problem
  - Computational costs problem

For details, please take a look SVM hand-written note by me.

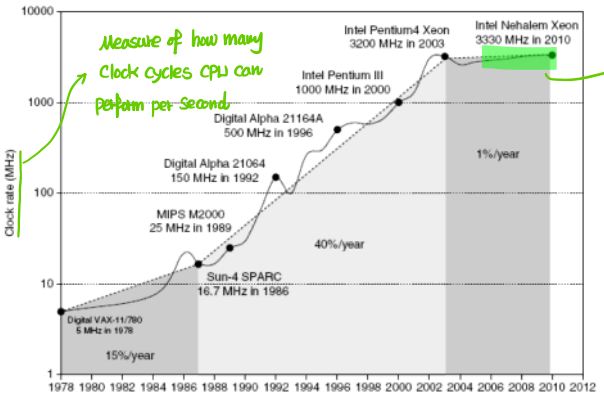
- For these problems, ANN method gave the most popularity to another methods such as SVM in 1996.
- However, the Deep learning is currently taking the seat back by solving all the problems that we mentioned.
- From now on, we are going to walk through how they actually solve the problems.

a) Computational costs problem

- Due to the heavy computation with a bunch of parameters, it might be true that they couldn't handle it.

↳ This was one of reasons why the fancy idea is not able to be more popular.

- As computer is surprisingly developed, the problem was naturally solved. In addition, people started again to think about it.



Measure of how many clock cycles CPU can perform per second

< Single processor clock rate >

Actually, the performance could go further.

However, there was an issue, aka power wall

Unable to dissipate the heat produced by circuit

↳ For this reason, parallel computing has been introduced.

b) Overfitting problem

- As the number of hidden layer increases in deep learning
  1) It might be able to model a complex non-linear equation.
  2) At the same time, it might confront overfitting problem.

· As the number of hidden layer increases in deep learning { 1) It might be able to model a complex non-linear equation.
2) At the same time, it might confront overfitting problem.

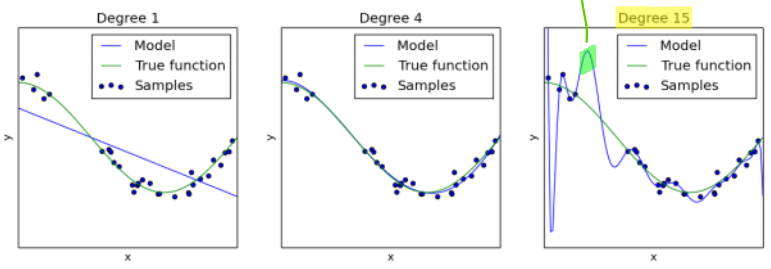
· Then, what is overfitting ?

- In machine learning, overfitting refers to a model that models the training data too well.
- It happens when a model learns too much details in the training data.
- It negatively impacts the performance of the model on new data.

· For example.

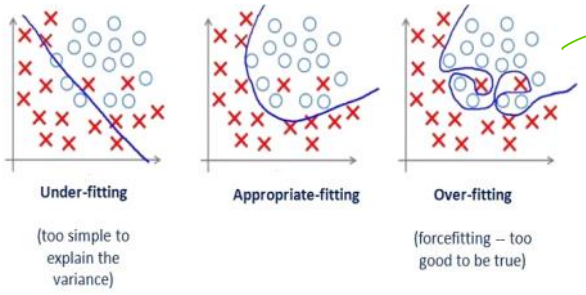
a) overfitting in regression

If we have a new data point at this location, the estimation will be ..



<Under-fitting> <Just right> <overfitting>

b) overfitting in classification



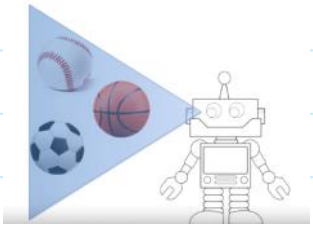
→ Yes, we've done well to classify this (only) example.

⇓

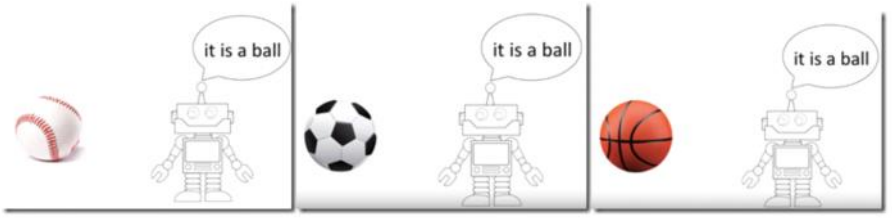
It's hard to say that this is a generalization model

· Let us take a simple example in order to understand the difference between underfitting and overfitting.

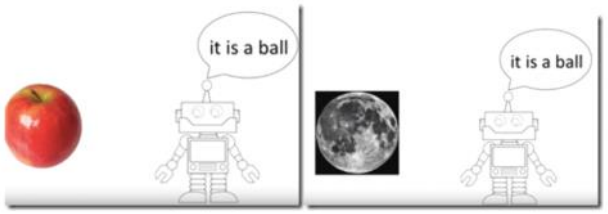
- Let's say that we are going to train 'If you have circle-rounded shapes, it's a ball' to a machine.



- The machine might answer correctly with these items.



- However, the machine might say that those are also a ball.



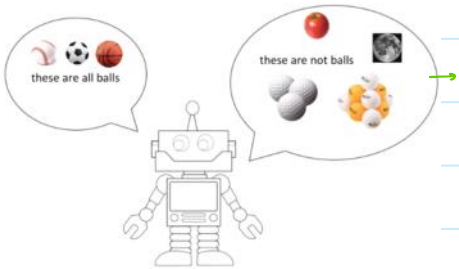
- We know that the machine is wrong in terms of the apple and the earth.

- In this case, we would say that the model is trained too bias, namely under-fitting.

· Let's say that we trained the machine again with a multiple features to overcome under-fitting problem.

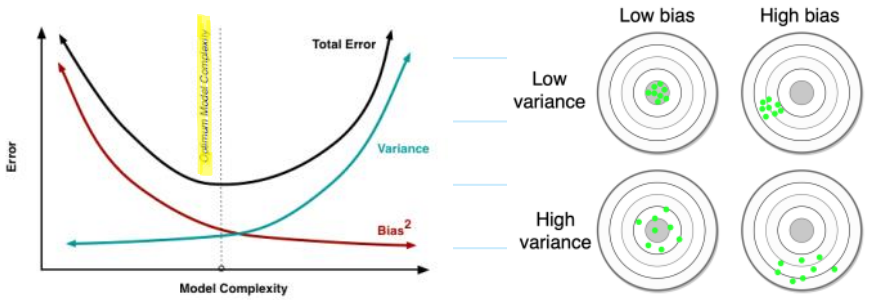
- It might work well with training datasets.

- However, it might misclassify with some testing datasets because it was trained too variance, namely overfitting.



→ The machine was incorrect w.r.t. golf balls.

· So, when we are doing Machine learning, it is important to recognize the effects of bias and variance as below;



· Then, how would you shoot the overfitting problem?

- To begin with, it is not likely to have under-fitting problem unless you are not an expert in machine learning.

- However, the overfitting problem is always everywhere in machine learning area, which means we have to resolve it.

· A few methods have been proposed to solve the overfitting problem.

① Data augmentation

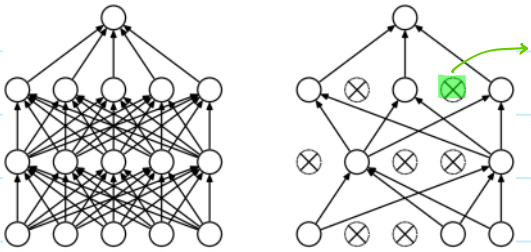
- We can actually use more training data such that the possibility to have overfitting in the model may reduce.

② Dropout (2012)

- This is a simple but powerful method to prevent overfitting in deep learning.

- Then, what is it?

: Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random.



(a) Standard Neural Net (b) After applying dropout.

(Fully-connected)

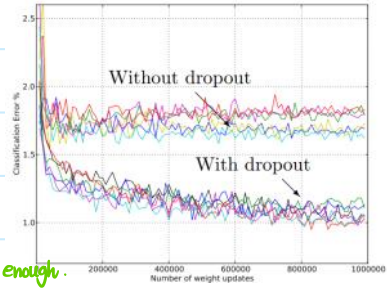
→ we do not exclude the neuron itself; however, we set the value equal to zero when fdward propagation is performed.

- It looks really simple but works well. (The team from Canada actually demonstrated it from their papers)

↳ well, how could this possible be a good idea?

: The basic idea is to focus a few features as less as possible.

e.g. We don't need so many experts to judge something!



We might not need it to classify a cat. The other info. may be enough.

③ Regularization

- As we discussed, regularization is one of techniques to prevent a model from overfitting.

- The main idea is 'Let us not have too big numbers in the weight'

- Let us think about the cost function as below;

Cost function (J): $\min_{w} \sum_{i=1}^{m} (y_i - w^T \phi(x_i))^2$

- Here, we may want to find weight parameters in which the cost function is minimized.

- In doing so, some of weight parameters may have very large numbers, which results in overfitting.

↳ Very steep lines..

- To avoid the problem, we also may want to minimize the weights as less as possible.

- It could be done by introducing one additional term in the equation.

$$J: \min_{w} \sum_{i=1}^{n} (y_i - w^T \phi(x_i))^2 + \lambda \|w\|_2^2 \quad ; \text{ where } \lambda \text{ is lambda}$$

- The equation means that we are trying to minimize cost function as well as weight parameters.

  (In the meantime, we will try to estimate values for the weight, though)

- In terms of λ, note that $\begin{cases} \lambda = \text{too low}, & \text{it doesn't do anything.} \\ \lambda = \text{too high}, & \text{it causes us to underfit. e.g. } \lambda \uparrow, \text{ to minimize J, } \vec{w} \downarrow \end{cases}$

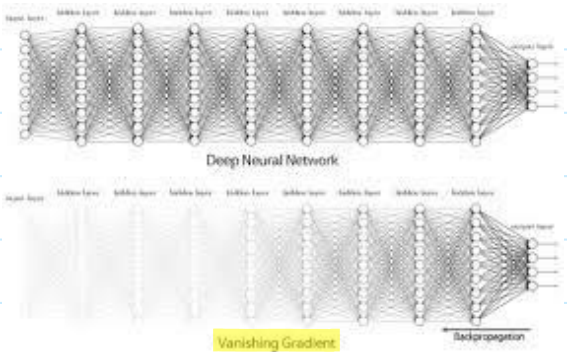
- Therefore, we have to choose the value λ properly.

c) Vanishing gradient problem

· As the number of hidden layer increases, there was another issue in deep learning society, namely vanishing gradient problem.

· Then, what is it?

- we may be able to recall that we used gradient-based optimization in Backpropagation of ANN.
- The vanishing gradient problem occurs when we try to train a model with Backpropagation algorithm.
- Generally, adding more hidden layers tends to make the network be able to model a complex arbitrary function.
- When we keep on adding more and more hidden layers in the model, the gradients calculated from Backpropagation algorithm tends to get smaller and smaller as we keep on moving back to the model.
- This results in the gradients are not quite reached to earlier layers. Even if is reached, it might be wrong information.



· Because of this problem, Deep learning met the second AI winter.

· In order to resolve the problem, a few methods had been proposed by AI researchers, especially by prof. Hinton in 2006.

- Pretraining (RBM) ; the prof. mentioned that we initialized the weight in a stupid way.
- ReLU activation function ; the prof. mentioned that we used wrong type of non-linearity.
- Recurrent Neural Network (RNN) → Vanishing gradient problem → Long Short-Term Memory network

d) Activation function

· We may be wondering ;

- Why do we have so many activation functions ?
- How do we know which one to use ?
- Why is that one works better than the other ?

· To answer the questions, it would be great to walk through the activation functions.

· Before we dive into them, let us recall the purpose of activation function.

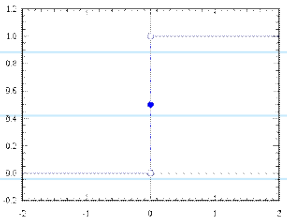
- First of all, it was introduced to describe non-linearity between inputs and output in a neural network model.
- This was actually more similar to real brain than original artificial neuron methodology.

↔ once it gets all incoming signals, it makes a decision if it is activated or not.

① Step function

· The first thing that comes to our minds is how about a threshold based activation function.

e.g. If the value of y is above a certain value, declare it activated.



· It would work well if we are only expected to classify either 0 or 1.

· However, if we want something intermediate activation values, it will not work.

② Linear function

· This is a straight line function where activation is proportional to input.

· This way, it gives a range of activations ; so, it's not binary activation anymore.

· However, we may have to notice that it has a constant gradient.

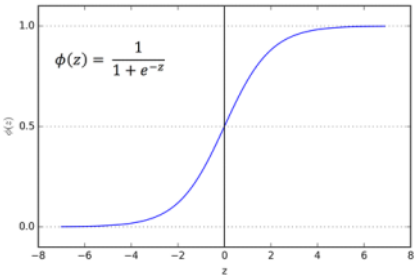
$$A = cx \rightarrow \text{Gradient} = c$$

· When we think about the gradient based method in training, it would be easy to realize that this isn't good.

③ Sigmoid function

· In general, this function is one of the most widely used activation functions today because ;

- It is nonlinear in nature (Recall the purpose of activation function)
- It is not binary activation.
- The output of activation function is always going to be in range (0,1).
- It has a smooth gradient.

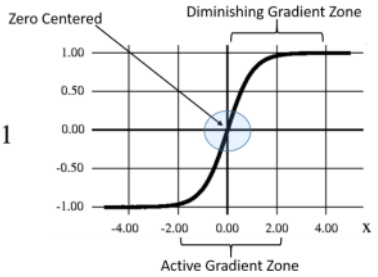


- However, there is a problem of vanishing gradient.
  - Towards either end of the sigmoid function, the Y values tend to respond very less to change in X.
  - It means the gradient at the region is going to be small.
  - It cannot make significant change in Backpropagation ; hence, the network refuses to learn further in deep neural network.

④ TanH function

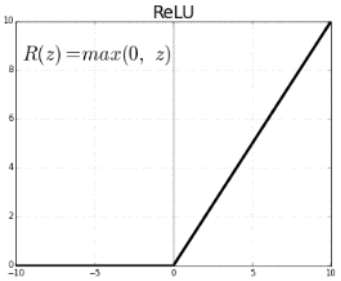
- In fact, this is a scaled Sigmoid function.
- Therefore, this has characteristics similar to sigmoid function that we discussed.
- One point to mention is that the gradient for tanH is stronger than sigmoid.
- However, it also has the vanishing gradient problem.

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



⑤ ReLU (Rectified Linear Unit) function

- Basically, this function was introduced to shoot vanishing gradient problem.
- It gives an output x if x is positive and 0 otherwise.



- It is nonlinear in nature.
- It resolves the vanishing gradient problem.
- It is less computationally expensive than other sigmoid functions. (Especially good for Deep learning)

Note that they are exponential-based functions.

· However, it also has a few drawbacks.

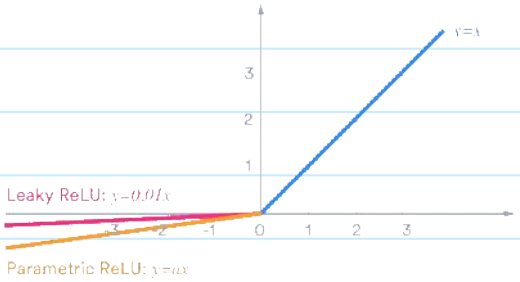
- The range is [0, inf] which means it can blow up the activation.

- Because of the horizontal line (Negative x), gradients will be zero.

↳ This can cause several neurons to just die and not respond to the network.

(It is called as 'Dying ReLU problem') → This is sometimes helpful to reduce the computational cost.

of. Leaky ReLU solved the dying ReLU problem. (parametric ReLU as well)



⑥ Summary

· So, which activation function should we pick?

· Well, it depends on your problem.

· In general, a sigmoid function works well for a classifier.

· ReLU works most of the time as a general approximator.

· We can actually use our own custom functions if necessary.

e) Pretraining with Restricted Boltzmann Machine (RBM)

· It is well known that we have to initialize the weight parameters in Artificial Neural Network structure.

· People might have initialized them randomly.

· Professor Hinton pointed out that we have initialized the weight parameters in a stupid way.

↳ He believed that this causes the problem in Deep neural network. Hence, it might turn in vanishing gradient problem.

· So, he proposed the method, called as RBM, for pretraining the weight parameters.

· How was it possible? what was the main idea?

- we typically solve the structure shown in below in ANN.

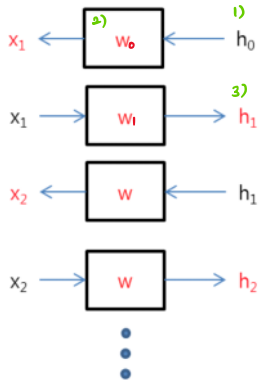
Input → weight → output <Neural network system> ; weight is only unknown

- In his idea, we may need to think in opposite to estimate the weight parameters.

Input ← weight ← hidden value ···· → Output < Deep belief network >

- Here, we know that it's impossible to predict weight and hidden value at the same time.

- In order to solve them simultaneously, he suggested 'Restricted Boltzmann machine'. The main idea is as follows ;



1) Initialize h_0

2) Train W_0 to minimize (Predicted - X_1) using gradient descent method

3) Estimate h_1 using X_1 real input and W_1 weight information

4) Repeat the process until we get optimized W and h.

5) Freeze the weight information and use it for network training in the two layers.

* Note that we need to repeat this when we have new layers, like step by step.

Long Short Term Memory (LSTM) network

· As we discussed, many researchers had tried to mimic human brain by using a concept of Artificial neural network.

· However, humans don't start their thinking from scratch every second. Their thoughts have persistence.

↳ Unfortunately, the traditional neural network couldn't mimic this sequential type.

- In the traditional network, we assume that all inputs and outputs are independent of each other.

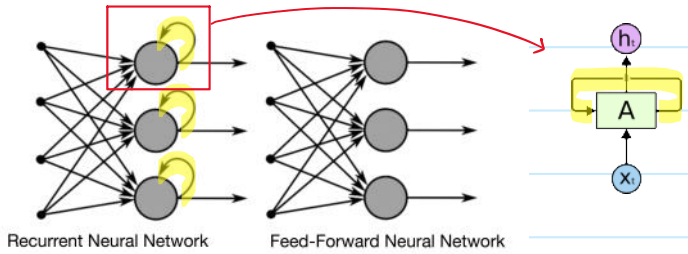
- In reality, some of tasks are actually sequential. e.g. Stock, weather, language, movie, ···

- For example, what if you want to predict the next word in a sentence you know better which words came before it.

· Recurrent Neural Network (RNN) actually did address this issue.

↳ What is this ? How does it look like ? Let us talk about it before LSTM because LSTM is a type of RNN.

· Basically, RNN has networks with loops in them allowing information to persist.

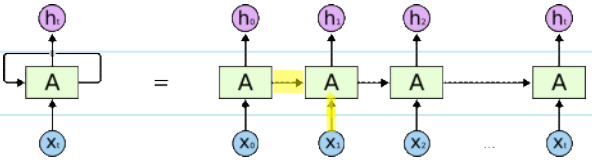


· So, we see the new loops that we couldn't see in traditional neural network.

- Then, what is the significance of the loop ?

: Actually, it allows the network to be related to sequences.

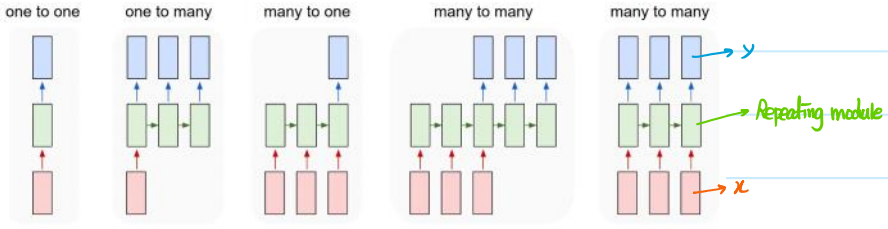
· A RNN can be thought of as multiple copies of the same network, each passing a message to a successor.



- As can be seen, this chain-like nature revealed that RNN is able to handle a sequential data.
- In fact, there had been incredible success applying RNNs to a variety of problems such as language modeling.

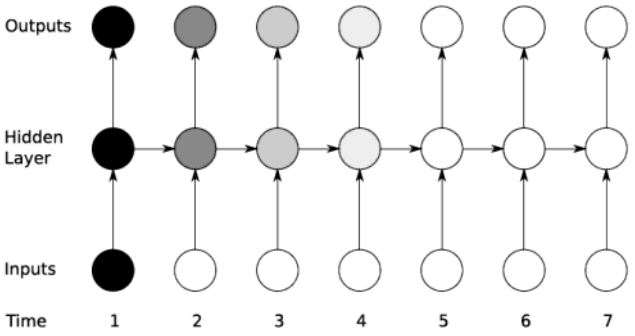
  (In the mean-time, CNN did an amazing job in image recognition stuffs)

- In addition, a RNN can have a variety of structure types for their purpose;



e.g. structure for translator

- So, it looks pretty. Isn't it?
- Unfortunately, however, a RNN has also the problem like as vanishing gradient problem.
  - Yes, RNN works well if the gap between previous information and the present work is small.
  - However, as the gap grows, RNNs become unable to learn to connect the information.
  - In theory, several researchers found that RNNs are not capable of handling such long-term dependencies.



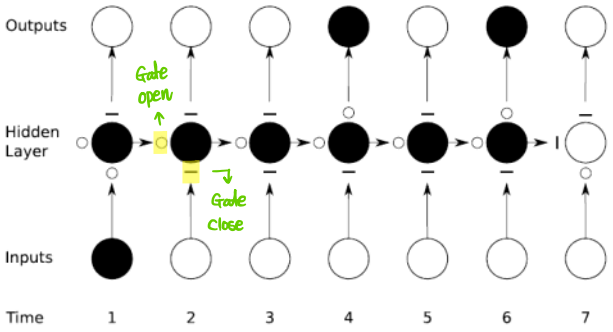
In Backpropagation,

we still use (x) multiply.

- In order to shoot this problem, finally, the LSTM was introduced.
- Again, the LSTM is a special kind of RNN which is capable of handling long-term dependencies.

  ↳ It's actually now widely used because it works tremendously well.

- For example, the vanishing gradient problem mentioned above was resolved by LSTM structure as below;

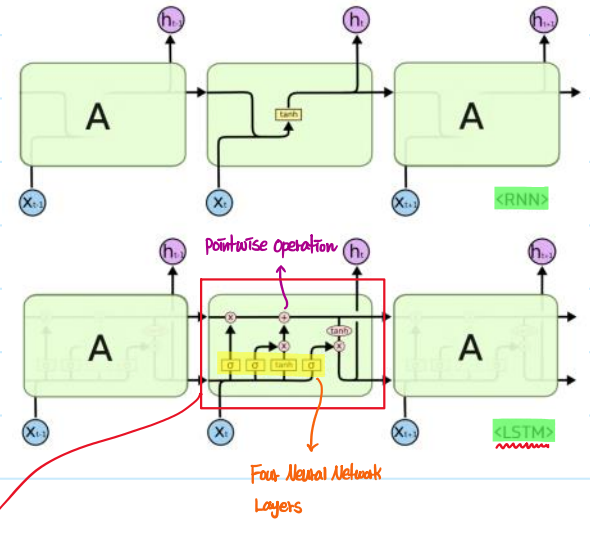


In Backpropagation,

We use (+) addition

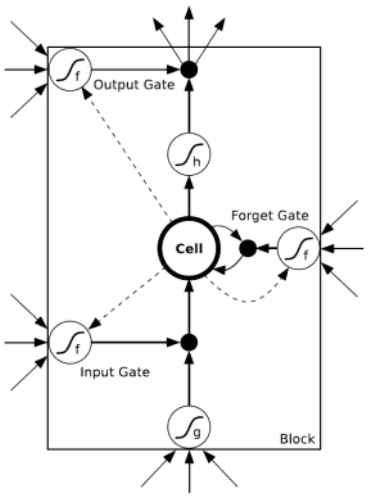
instead of (x) multiplication

- So, how was it possible ? how does LSTM structure look like ?
- Basically, LSTM networks also have the chain-like structure as RNNs do.
  - ↳ However, the repeating module has a different structure.
    - Instead of having a single neural network layer, there are four, interacting in a very special way.



Let us see this block in detail.

- The LSTM does have the ability to remove or add information, which is carefully regulated by the gates.
- Here, the gates are a way to optionally let information through.
- They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.
- Since the sigmoid layer outputs are between 0 and 1,
  - It describes how much of each component should be let through.
  - 0 means let nothing through.
  - 1 means let everything through.
- Thus, the LSTM network has three types of these gates (Input, output, and forget) to protect and control the cell state.



< Memory block of LSTM network >

- Because of the characteristic, the LSTM network has been widely used to handle sequential data.
- The best example is to predict weather. Let me attach one of my paper results, NASA MERRA-2 weather prediction using LSTM.