

Introduction to Deep Learning and Its Application to Hearing Research

Basic of Neural Network
Tensorflow
Convolutional Neural Network,
Autoencoder,
Recurrent Neural Network,
Generative Adversarial Network

Prof. Ahn Kang-hun

Department of Physics,
Chungnam National University

ahnkanghun@gmail.com

Colloquium at Dept. of Physics, Seoul National University, March 14, 2018

arXiv:1712.06340

Accepted in ICASSP 2018 Lecture Session

LANGUAGE AND NOISE TRANSFER IN SPEECH ENHANCEMENT GENERATIVE ADVERSARIAL NETWORK

Santiago Pascual¹, Maruchan Park², Joan Serrà³, Antonio Bonafonte¹, Kang-Hun Ahn²

¹ Universitat Politècnica de Catalunya, Barcelona, Spain

² Chungnam National University, Daejeon, Republic of Korea

³ Telefónica Research, Barcelona, Spain

ABSTRACT

Speech enhancement deep learning systems usually require large amounts of training data to operate in broad conditions or real applications. This makes the adaptability of those systems into new, low resource environments an important topic. In this work, we present the results of adapting a speech enhancement generative adversarial network by fine-

In previous work, we proposed an end-to-end speech enhancement system [6] based on a generative adversarial network [7] (GAN), namely speech enhancement generative adversarial network (SEGAN). SEGAN was proposed in the pursuit of end-to-end speech processing, where signal is enhanced at the raw waveform level, with a one-shot, non-recursive structure. It showed the applicability of latest deep

Training Korean speech & Result

Training data

Pre-trained with English (86epochs)
Clean korean speech : 200m
Noisy korean speech : 200m
30 epochs

Test data

Unseen noisy korean speech
(Different noise, speaker, sentence)

Result

noisy



enhanced



noisy



enhanced



Erasing artillery sound

Data set

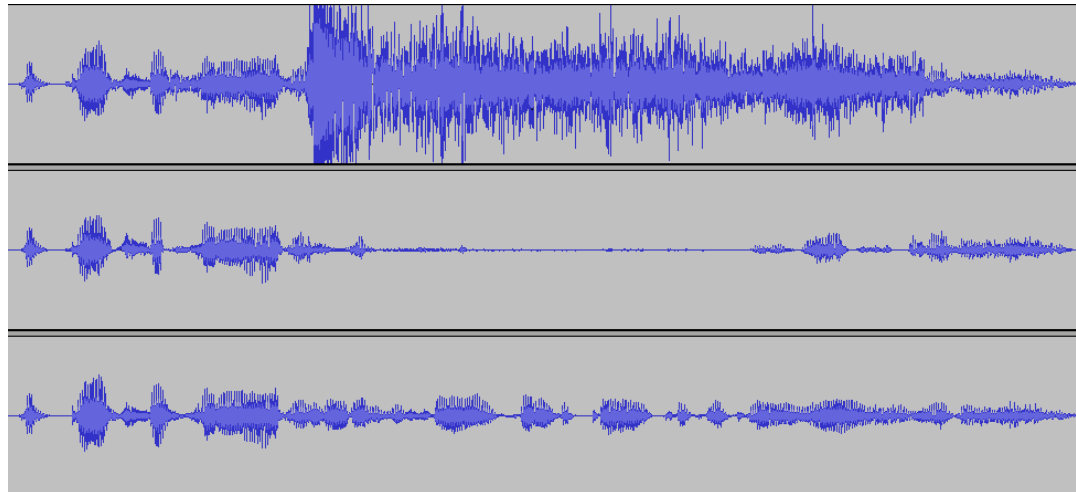
speech + artillery sound

54 Training data

4 Test data

noisy

enhanced



Types of Learning

Supervised (inductive) learning

- Training data includes desired outputs

Unsupervised learning

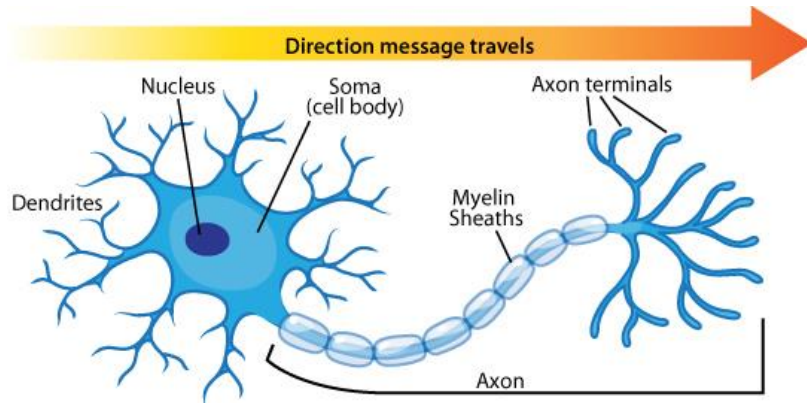
- Training data does not include desired outputs

Semi-supervised learning

- Training data includes a few desired outputs

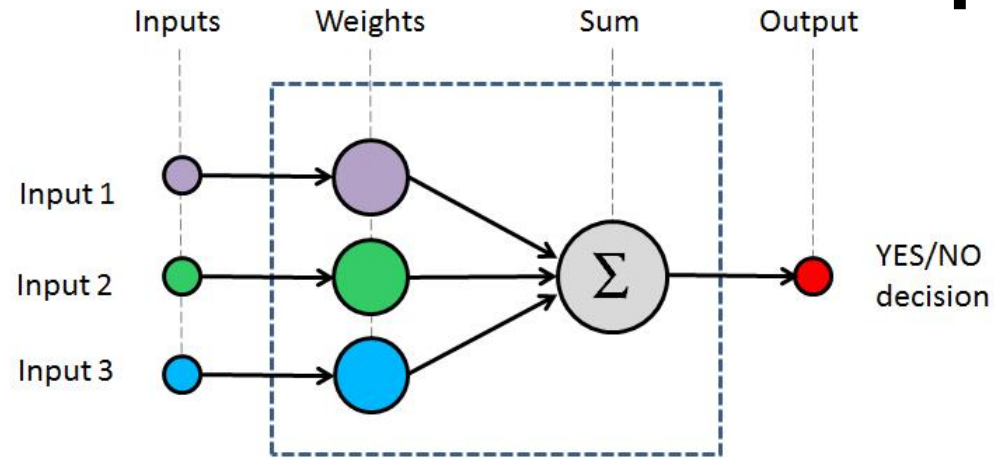
Reinforcement learning

- Rewards from sequence of actions



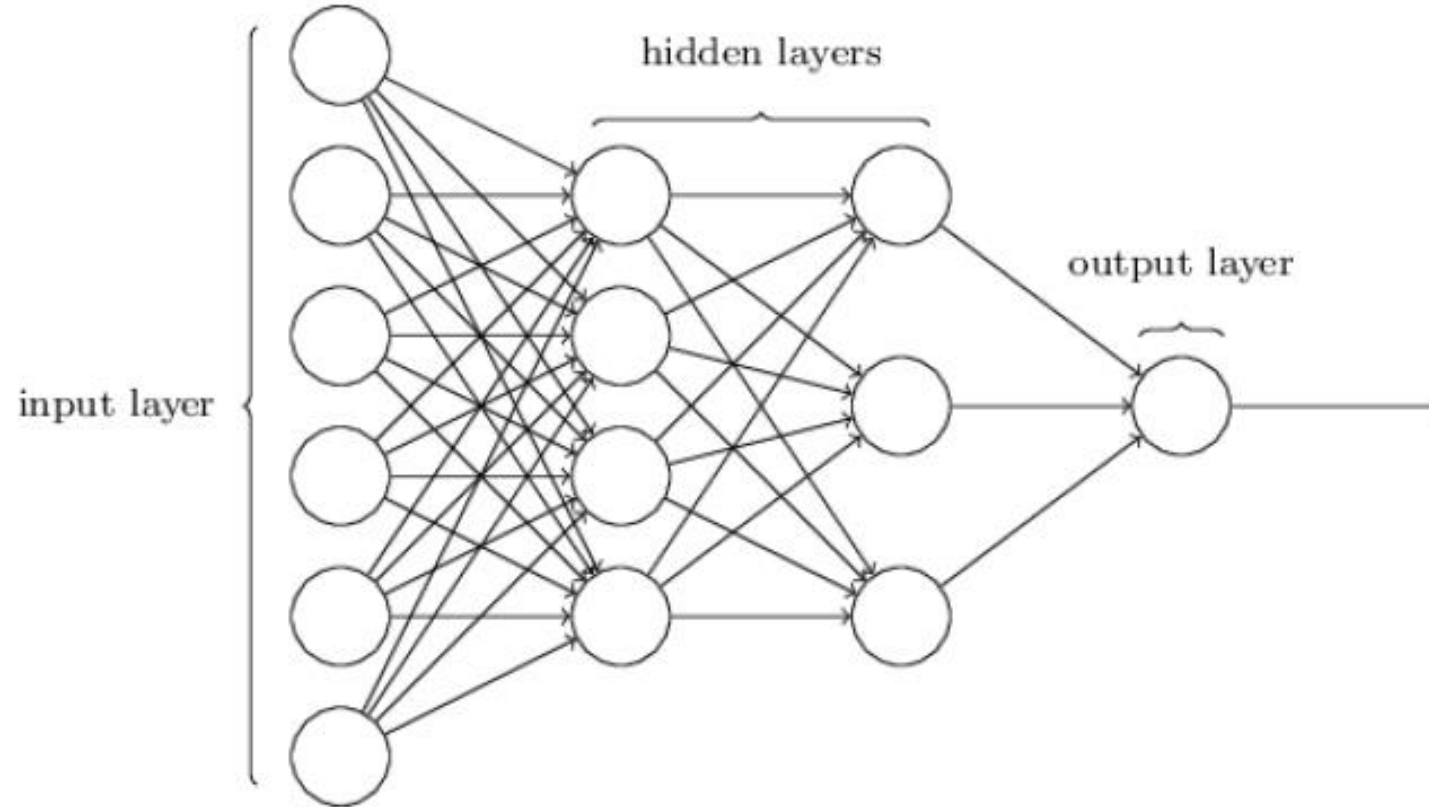
Frank Rosenblatt
(1927-1971)

Perceptron

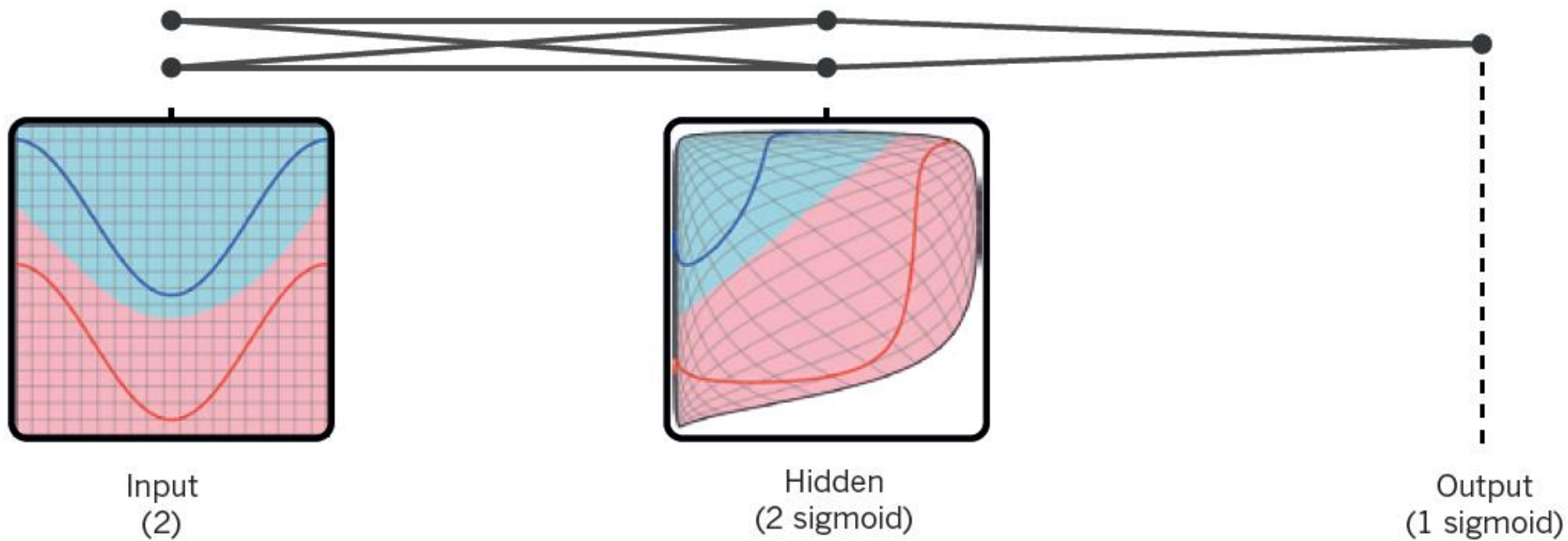


$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

The architecture of neural network



Role of hidden layer

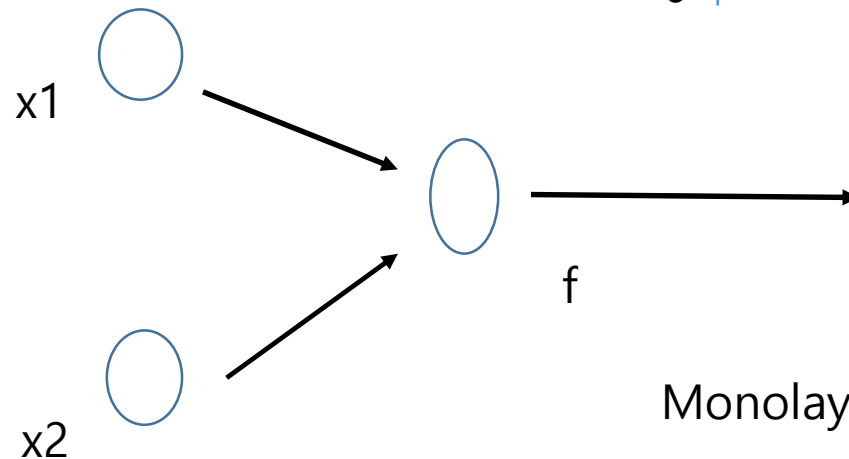
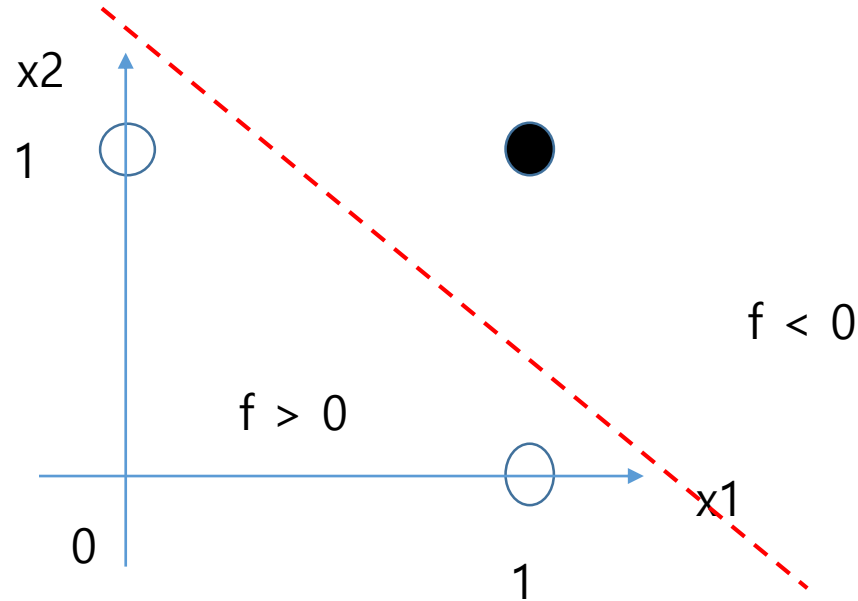


Example: AND operation

x1	x2	f
1	0	0
0	1	0
1	1	1

Monolayer net is enough for AND operation.
One can find proper weight and bias.

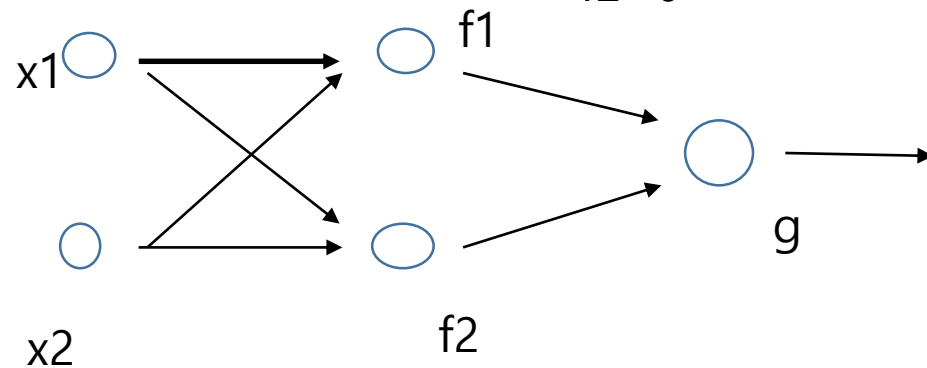
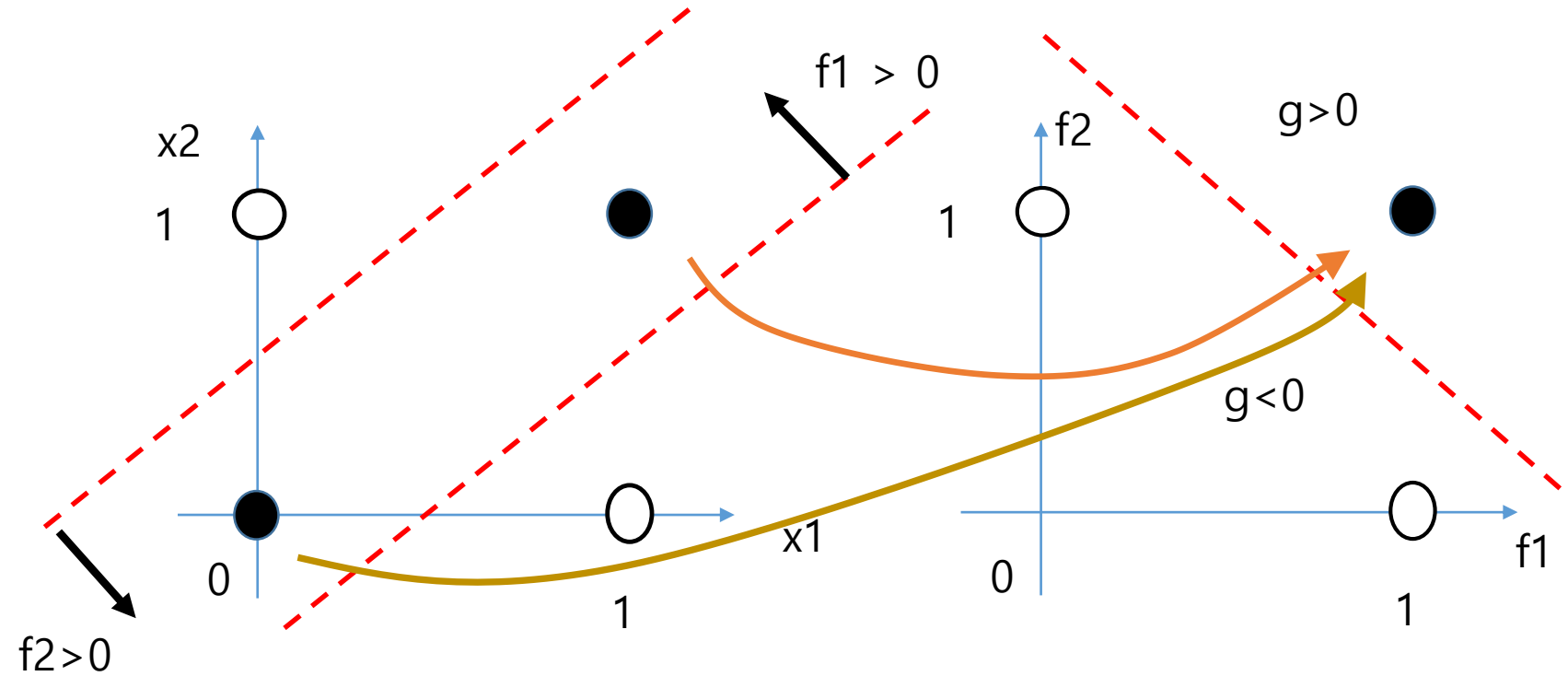
Desired output



$$f = w_{11}x_1 + w_{12}x_2 + b$$

XOR problem : Monolayer is not enough. Hidden layer is required.

x1	x2	f
1	0	0
0	1	0
1	1	1
0	0	1



$$f1 = w_{11} x1 + w_{12} x2 + b1$$

$$f2 = w_{21} x1 + w_{22} x2 + b2$$

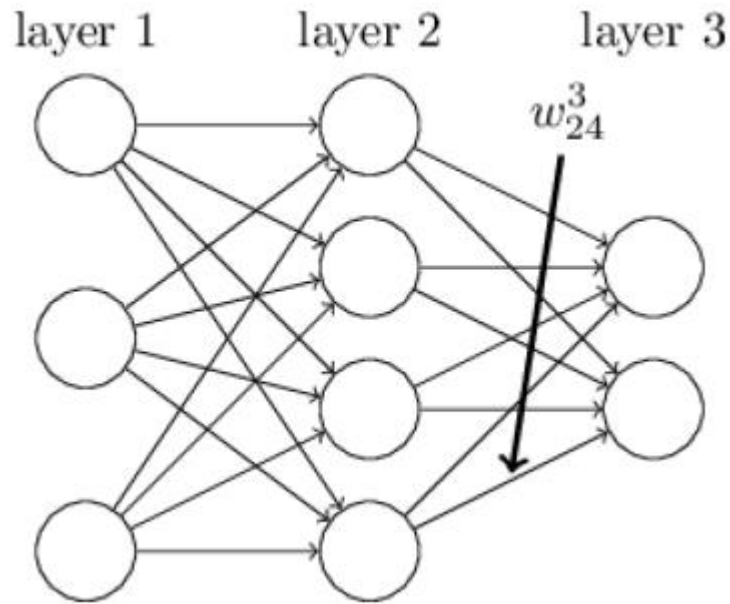
$$g = w'_{11} f1 + w'_{12} f2 + b'$$

Training : Minimizing an object function (error)

$$C(w, b) \equiv \sum_x \|y(x) - a\|^2$$

Desired output

Calculated output



$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

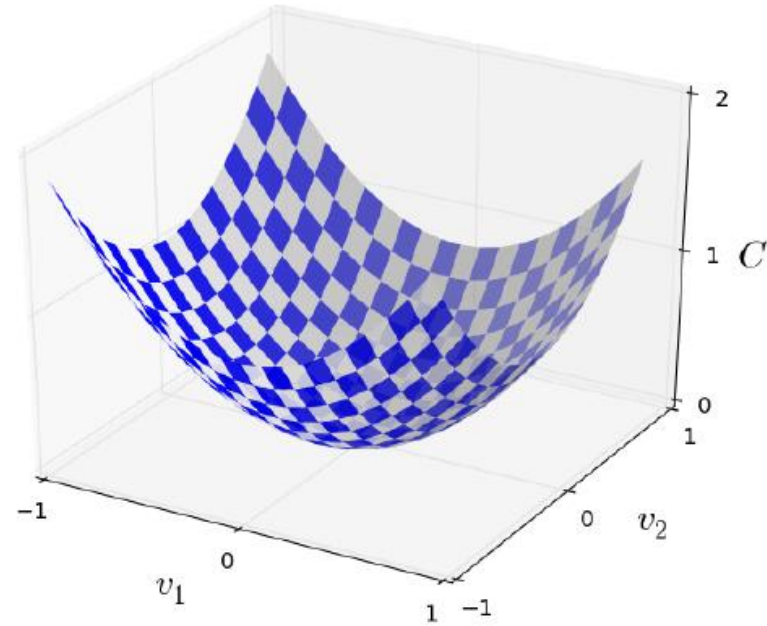
activation

Weight matrix

bias

Gradient descent method

$$\Delta C \approx \nabla C \cdot \Delta v,$$



$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \dots, \frac{\partial C}{\partial v_m} \right)^T$$

$$\Delta v = (\Delta v_1, \dots, \Delta v_m)^T$$

$$\Delta v = -\eta \nabla C,$$

Back propagation

A simple linear equation for the error in terms of the error of next layer

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

Proof)

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1},\end{aligned}$$

where

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}.$$

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

— The equations of backpropagation —

$$1. \delta^L = \nabla_a C \odot \sigma'(z^L)$$

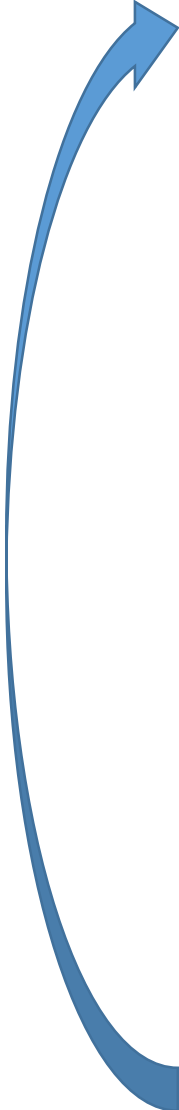
$$2. \delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$3. \frac{\partial C}{\partial b_j^l} = \delta_j^l$$

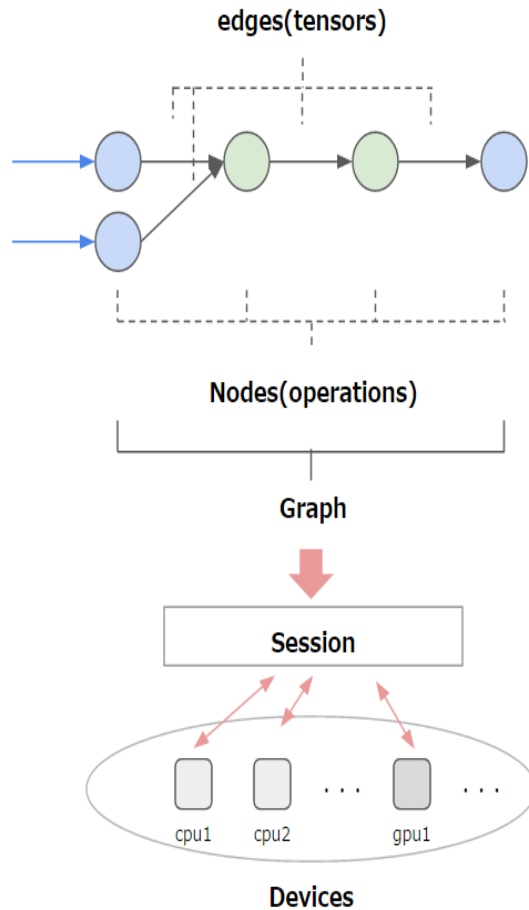
$$4. \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Hardamard product



- 
1. **Input x :** Set the corresponding activation a^1 for the input layer.
 2. **Feedforward:** For each $l = 2, 3, \dots, L$ compute
$$z^l = w^l a^{l-1} + b^l \text{ and } a^l = \sigma(z^l).$$
 3. **Output error δ^L :** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
 4. **Backpropagate the error:** For each $l = L - 1, L - 2, \dots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
 5. **Output:** The gradient of the cost function is given by
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \text{ and } \frac{\partial C}{\partial b_j^l} = \delta_j^l.$$
 6. **Gradient descent:** For each $l = L, L - 1, \dots, 2$ update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

Tensorflow



Tensorflow is an open source software developed by Google Brain, research organization of Google. This software is designed for configuring AI programs, so it is suitable for making neural networks. Neural networks can be represented as a graph as shown in the figure. The circles are called neurons, and passing data from one to another can be represented by arrows. In this graph, the neurons form a node with arrows, and the circle itself contains some sort of operation, including what we will introduce later, Sigmoid or ReLU.

In tensorflow, the neural networks to be calculated are first constructed as graphs. These graphs are just a representation of a plan to perform some calculations, that is, a kind of code generation. When you perform something called "Session", data is input and the actual calculation is performed. In this process, the resources of the computer can be used in parallel. The arrows indicate the name tensorflow because it is represented by a tensor (I am not 100% sure).

Let's create a simple tensorflow program that multiplies two numbers.

```
import tensorflow as tf  
a=tf.placeholder("float")  
b=tf.placeholder("float")  
y=tf.multiply(a,b)
```

This code constitutes a graph of the tensorflow. ***a*** has no value in this situation, instead, ***a*** will be ordered "to stay in place." It is a "**placeholder.**" ***b*** is the same. Multiplication with two placeholders is performed with the multiply command of tf. You don't need to separately specify the placeholder of ***y***. The following steps are preparing to execute the calculation and running session.

```
sess=tf.Session()  
print(sess.run(y, feed_dict={a:3,b:3}))
```

If we consider a graph as an architectural design, creating a session means that you prepare construction workers and construction equipment. In the above, '**sess**' is the name of the session, which is prepared to do so. You need a "**sess.run**" to start this. The last line gives inputting values and printing outputs at the same time. Note that sess is not executed until sess.run appears. And at the moment sess is run, all variables on the graph are assigned proper values.

Ex) Single Neural Layer Network

```
import tensorflow as tf  
import input_data  
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
x= tf.placeholder("float",[None,784])  
W=tf.Variable(tf.zeros([784,10]))  
b=tf.Variable(tf.zeros([10]))
```

```
y=tf.nn.softmax(tf.matmul(x,W)+b)  
y_=tf.placeholder("float",[None,10])
```

```
cross_entropy = - tf.reduce_sum(y_*tf.log(y))
train_step=tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
sess=tf.Session()
sess.run(tf.global_variables_initializer())
```

```
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

This way, 100 pieces of data will be randomly sampled and used for training.
Here, “**_xs**” means images and “**_ys**” means their labels. Now run the following code to check the test results

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print(sess.run(accuracy, feed_dict={x: mnist.test.images,
y_:mnist.test.labels})))
```

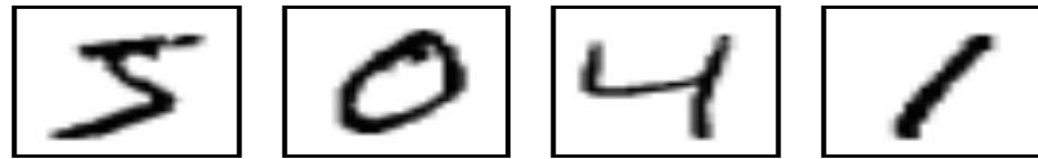
In the first line, **tf.argmax(y, 1)** finds the largest value of **y** along axis = 1. The **y** matrix is **None** by 10 (= (None by 784) * (784 by 10)), where axis = 0 refers to the **None** side, row, while axis = 1 refers to the 10 side, column. "**None**" commonly means that any number is possible, and here, it means the number of input image data. Depending whether the maximum value of **y** and **y_** in the first line is the same or not, it returns TRUE or FALSE as a shape of array. **tf.cast** will do this for [0,1,1,1,1,0,1,1 ... 1], by **TRUE** is 1 and **FALSE** is 0. Then, the accuracy percentage is obtained by **tf.reduce_mean**, which calculates the mean of the input array.

Ref) MNIST data

The MNIST data-set is composed by a set of black and white images containing hand-written digits, containing more than 60.000 examples for training a model, and 10.000 for testing it. The MNIST data-set can be found at the *MNIST database*.

This data-set is ideal for most of the people who begin with pattern recognition on real examples without having to spend time on data pre-processing or formatting, two very important steps when dealing with images but expensive in time.

The images are centered in 28×28 pixel frames by computing the mass center and moving it into the center of the frame. The images are like the ones shown here:



Also, the kind of learning required for this example is *supervised learning*, the images are labeled with the digit they represent. This is the most common form of Machine Learning.

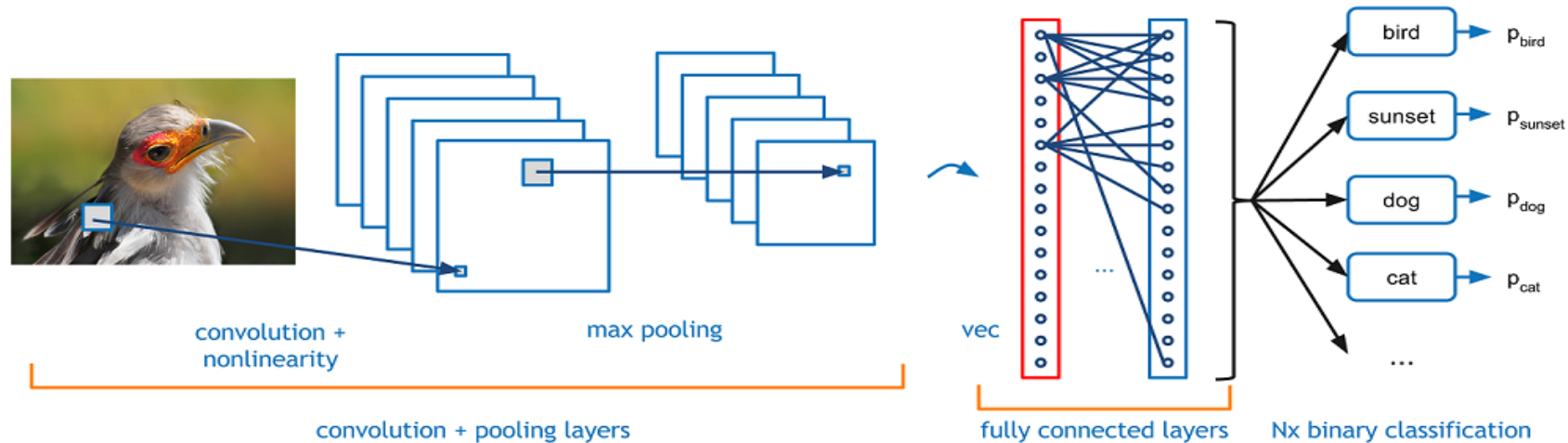
To download easily the data, you can use the script *input_data.py*, obtained from Google's site but uploaded to the book's *github* for your comodity. Simply download the code *input_data.py* in the same work directory where you are programming the neural network with TensorFlow. From your application you only need to import and use in the following way:

```
import input_data  
mnist = input_data.read_data_sets("MNIST_data", one_hot=True)
```

After executing these two instructions you will have the full training data-set in ***mnist.train*** and the test data-set in ***mnist.test***. Each element is composed by an image, referenced as "***xs***", and its corresponding label "***ys***", to make easier to express the processing code. Remember that all data-sets, training and testing, contain "***xs***" and "***ys***"; also, the training images are referenced in ***mnist.train.images*** and the training labels in ***mnist.train.labels***.

Convolutional Neural Network (CNN)

<https://www.youtube.com/watch?v=dGkDEHPSMq4>

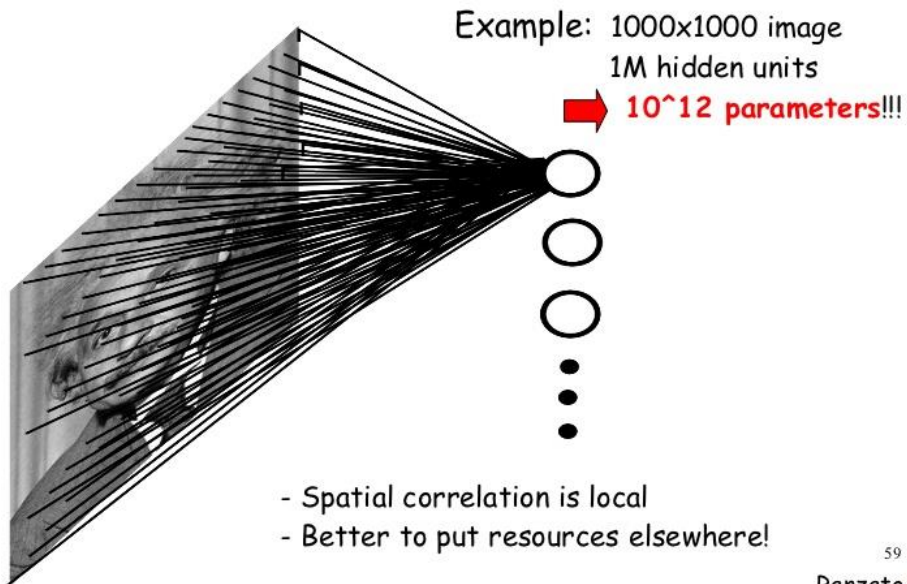


The convolutional Neural Network(CNN) is an innovative neural network introduced in 1998 by Yan LeCunn et al. This has led to dramatic improvements in automatic image processing and now, it is widely used in advanced machine learning models. Let's look at how to implement CNN through a simple example code.

◦ ◦ Convolutional neural network

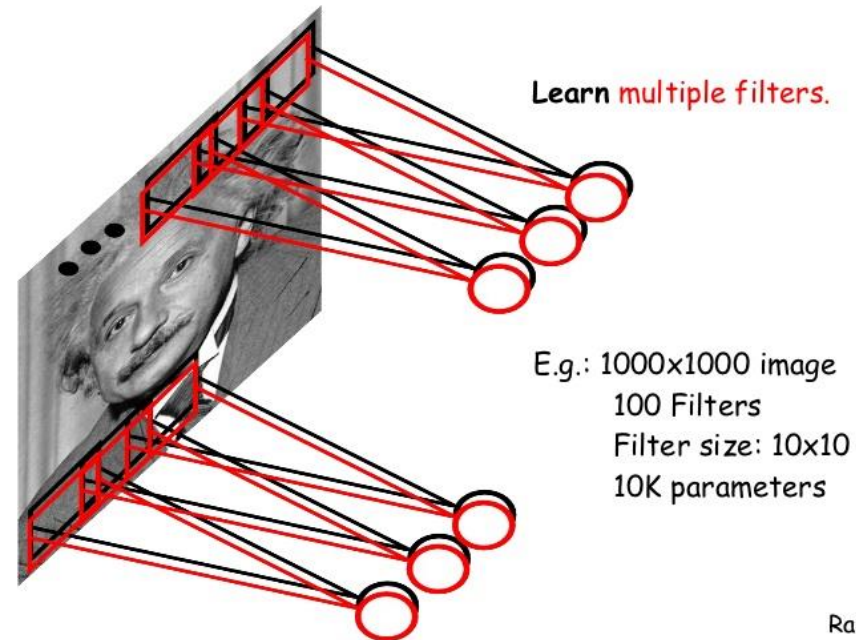
- Reduce parameters to optimize
- Avoid overfitting

FULLY CONNECTED NEURAL NET



59
Ranzato

CONVOLUTIONAL NET



64
Ranzato

<https://www.slideshare.net/zukun/p03-neural-networks-cvpr2012-deep-learning-methods-for-vision>




```
import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
import tensorflow as tf

x = tf.placeholder("float", shape=[None, 784])
y_ = tf.placeholder("float", shape=[None, 10])

x_image = tf.reshape(x, [-1,28,28,1])
```

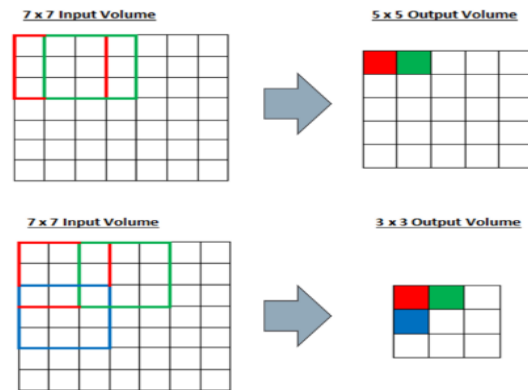
In the first two lines, we load the MNIST data through tensorflow. Then the "placeholder" literally holds a "place" to make room for tensors. A "reshape" changes the shape of x tensor into the shape in square brackets, where the first -1 means that you did not specify what number to be input, like "*NONE*". The second and the third numbers indicate the size(28x28) of the image data. And the last number is number of input data channel; here it has to be 1 because MNIST data are gray scale images.

```
def weight_variable(shape):  
    initial = tf.truncated_normal(shape, stddev=0.1)  
    return tf.Variable(initial)  
  
def bias_variable(shape):  
    initial = tf.constant(0.1, shape=shape)  
    return tf.Variable(initial)  
  
def conv2d(x, W):  
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')  
  
def max_pool_2x2(x):  
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],  
        padding='SAME')
```

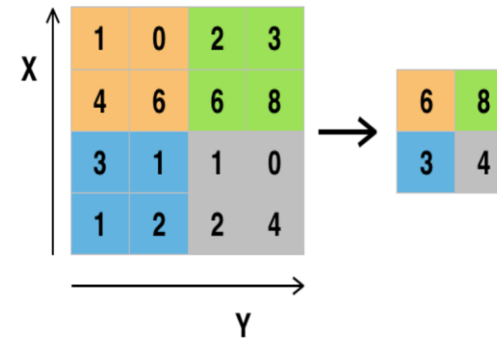
Above codes are making functions for CNN. First two functions, `weight_variable` and `bias_variable`, are make random matrix with given shape. `Conv2d` is function for a convolution layer and `max_pool_2x2` is function for a pooling layer. Convolution layers perform matrix multiplication among input data and weight variables. Then, in pooling layer, features are extracted by taking the largest value in each filter region. Basically, these processes are implemented by sharing all weight variables and bias of each filter through all hidden perceptrons.

When the filters, convolution filters or pooling filters, pass through the image, the degree that the filter moves each time is called "stride". Padding is attaching *zero* on boundary of input data to consider evenly about all input values. In above codes, convolution layer stride is (1x1), pooling layer stride is (2x2) and pooling filter size is (2x2). Both layers are performed with zero padding. See the below figures to understand how filtering processes work.

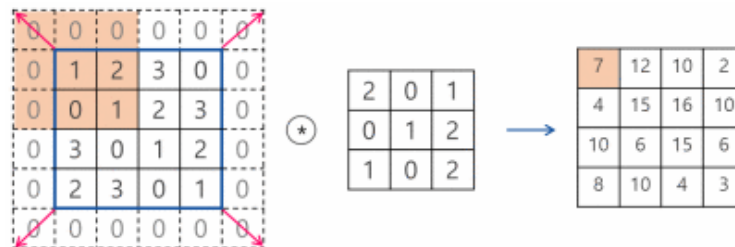
Convolution layer



Pooling layer (Max pooling)



Zero padding



```

W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

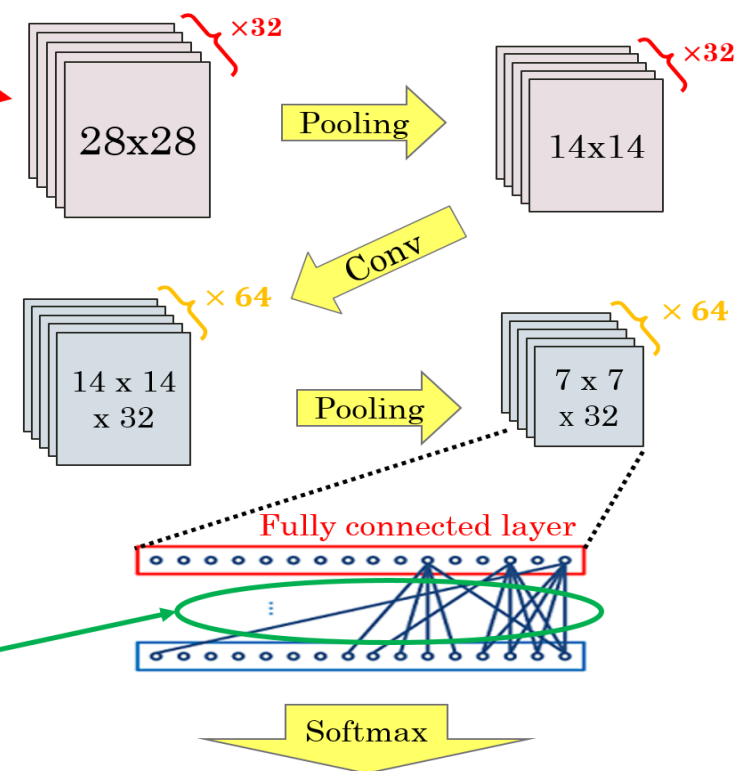
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

```



Classification !

Above figure shows whole process that how data shapes change as data passes through each layer. After whole process, finally, input data was classified through a fully connected layer.

Full code

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
x = tf.placeholder("float", shape=[None, 784])
y_ = tf.placeholder("float", shape=[None, 10])
```

```
x_image = tf.reshape(x, [-1,28,28,1])
print("x_image=", x_image)
```

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
```

```
def bias_variable(shape):
```

```
initial = tf.constant(0.1, shape=shape)  
return tf.Variable(initial)
```

```
def conv2d(x, W):  
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
```

맨 앞의 1 은 한놈씩 다룬다는 뜻, 가운데 둘은 1x1 stride 즉 위나 아래나 한칸씩 움직인다. 이렇게 하면 convolution 해도 결과이미지의 크기는 바뀌지 않겠지. 마지막은 1 은 흑백 을 의미하는 것임.

```
def max_pool_2x2(x):  
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],  
padding='SAME')
```

#커널 사이즈 2x2 스트라이드 2x2 즉 위아래 두칸씩. 이러면 결과 이미지 크기가 반으로 1/2 x 1/2 만큼 줄겠지.

```
W_conv1 = weight_variable([5,5, 1, 32])  
b_conv1 = bias_variable([32])
```

32개의 5x5 필터를 이용해서 인풋 개수 1개의 인풋 이미지로 32 개의 아웃풋 이미지를 만드는 필터.

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)  
h_pool1 = max_pool_2x2(h_conv1)
```

#convolution 하고 나서는 여전히 이미지 크기가 28x28 이다. 풀링을 하고 나서 14x14 로 이미지 크기가 준다. 이러한 이미지는 32개가 존재한다.

```
print(x_image.get_shape())
```

#이걸 하면 (?,28,28,1) 이라고 나오겠지. 28x28 이미지 하나.

```
print(h_conv1.get_shape())
```

#이걸 하면 (?,14,14,32) 라고 나오겠지. 32개의 14x14 이미지라는 뜻.

```
W_conv2 = weight_variable([5,5, 32, 64])  
b_conv2 = bias_variable([64])
```

#32개의 이미지를 훑는 필터 64개. 그필터의 사이즈는 5x5.

여기서 32개의 이미지를 훑기 때문에 이미지가 사실은 3차원 구조14x14x32.

```
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
```

#이걸 거치고 나면 64개의 14x14 이미지가 생긴다.

```
h_pool2 = max_pool_2x2(h_conv2)
```

#2x2 풀링을 통해 7x7 이미지로 변환. 64개.

```
print(h_conv2.get_shape()) #(? ,14,14,64)
print(h_pool2.get_shape()) #(? ,7,7,64)
```

```
W_fc1 = weight_variable([7*7*64, 1024])
b_fc1 = bias_variable([1024])
```

Fully connected network 을 위한 준비

```
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64]) #none 을 위한 축을 넣어서
계산 모양을 맞추어줌
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

```
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
```

```
y_conv = tf.nn.softmax(tf.matmul(h_fc1,W_fc2)+b_fc2)
```

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
train_step = tf.train.AdamOptimizer(0.0003).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```



```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

```
Acc_train = []
```

```
Acc_test = []
```

```
acc_te = 0
```

```
for i in range(3001):
```

```
    batch = mnist.train.next_batch(50)
```

```
    sess.run(train_step, feed_dict={x: batch[0], y_: batch[1]})
```

```
# batch 는 [[데이터],[라벨]] 이렇게 돼있다. 그리고 데이터는 50x784 라벨은 50x10.
```

```
    if i % 10 == 0:
```

```
        acc_tr = sess.run(accuracy, feed_dict = {x:batch[0], y_:batch[1]})
```

```
        acc_te = sess.run(accuracy, feed_dict={x: mnist.test.images, y_:
mnist.test.labels})
```

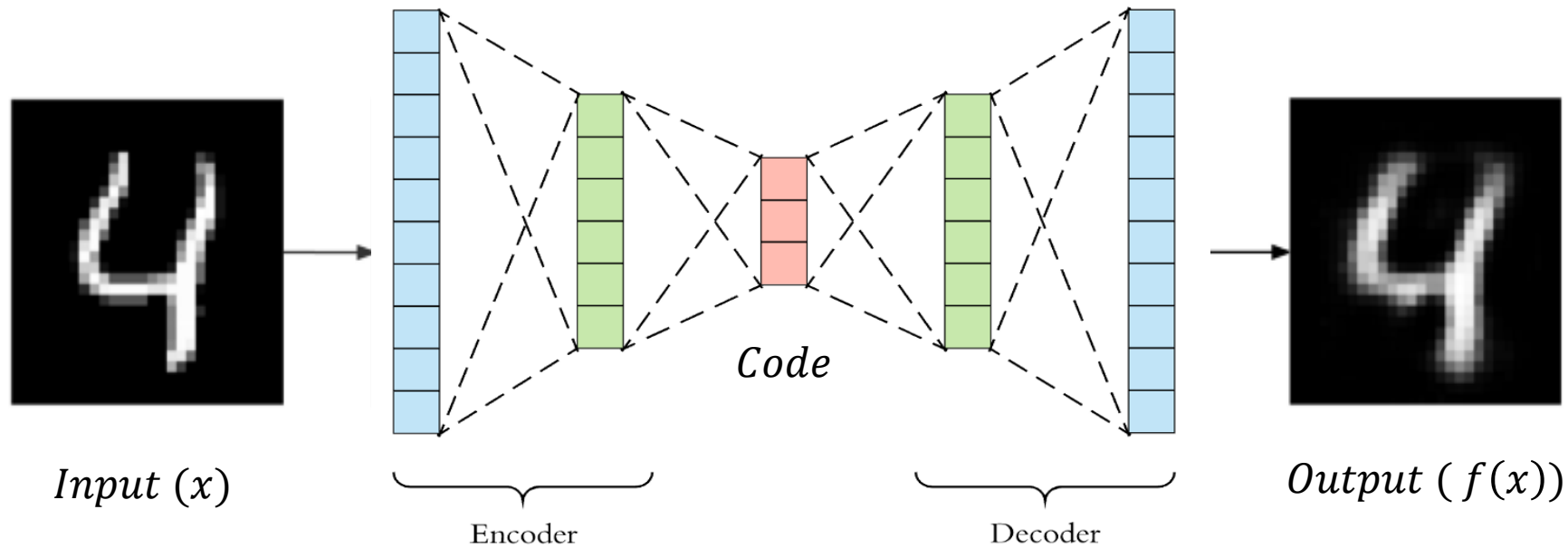
```
        print("step %d, training accuracy %g"%(i, acc_tr),"test
accuracy %g"%acc_te)
```

```
        Acc_train.append(acc_tr)
```

```
        Acc_test.append(acc_te)
```

Auto Encoder

CONCEPT



Input size = Output size

$$\text{minimize } C(x) = \frac{1}{n} \sum_i^n (x_i - f(x_i))^2$$

Implementation

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

##### graph
learning_rate = 0.01
training_epoch = 20
batch_size = 100

n_hidden = 256
n_input = 28*28

X = tf.placeholder(tf.float32, [None, n_input])

W_encode = tf.Variable(tf.random_normal([n_input, n_hidden]))
b_encode = tf.Variable(tf.random_normal([n_hidden]))
encoder = tf.nn.sigmoid(
    tf.add(tf.matmul(X, W_encode), b_encode))

W_decode = tf.Variable(tf.random_normal([n_hidden, n_input]))
b_decode = tf.Variable(tf.random_normal([n_input]))

decoder = tf.nn.sigmoid(
    tf.add(tf.matmul(encoder, W_decode), b_decode))
cost = tf.reduce_mean(tf.square(X - decoder))
optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(cost)
```

```
##### Session
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

total_batch = int(mnist.train.num_examples/batch_size)

for epoch in range(training_epoch):
    total_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        _, cost_val = sess.run([optimizer, cost],
                                feed_dict={X: batch_xs})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.4f}'.format(total_cost / total_batch))

print('training complete')

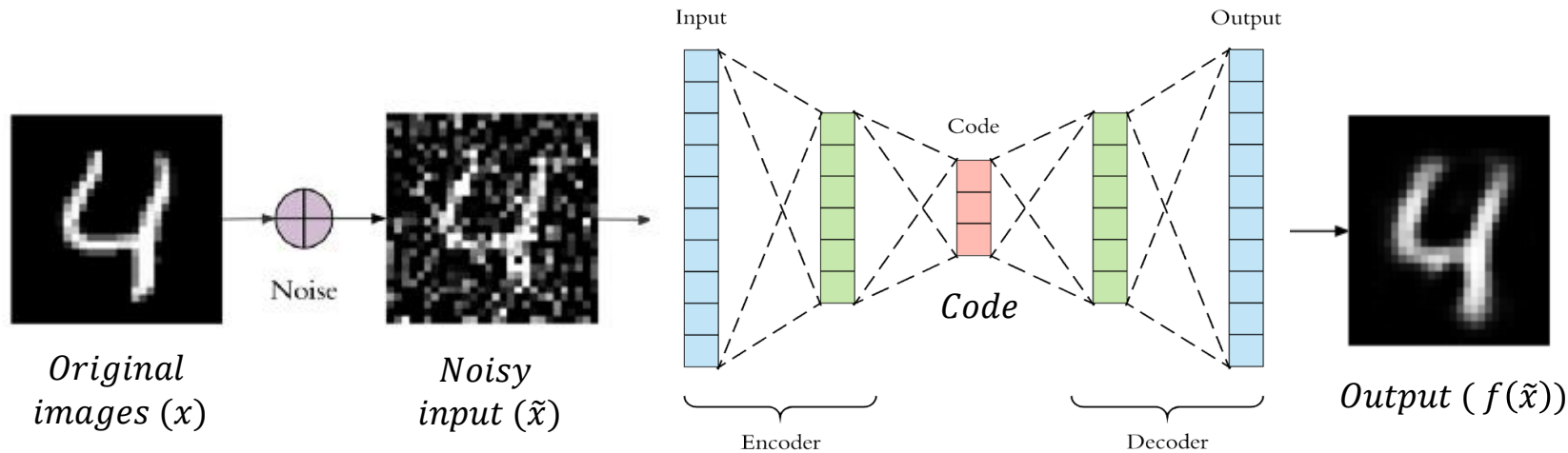
##### test
sample_size = 10

samples = sess.run(decoder,
                    feed_dict={X: mnist.test.images[:sample_size]})

fig, ax = plt.subplots(2, sample_size, figsize=(sample_size, 2))
```

Application

- Denoising Auto Encoder (DAE)



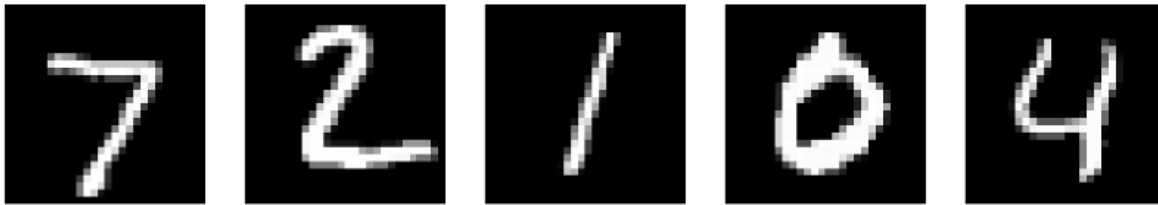
$$\text{Training} - \text{minimize } C(x, \tilde{x}) = \frac{1}{n} \sum_i^n (x_i - f(\tilde{x}_i))^2$$

Make the output of noisy data similar to the original data

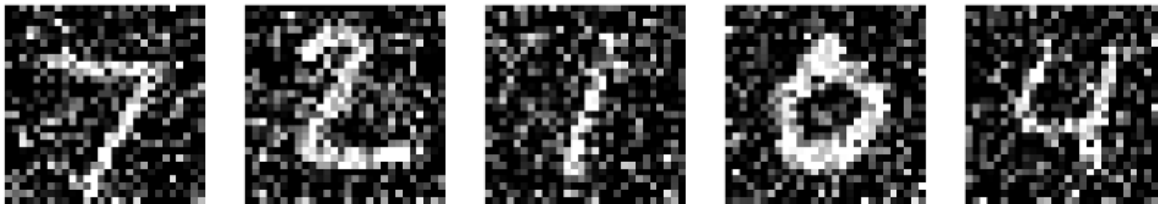
Application

- Denoising Auto Encoder (DAE)

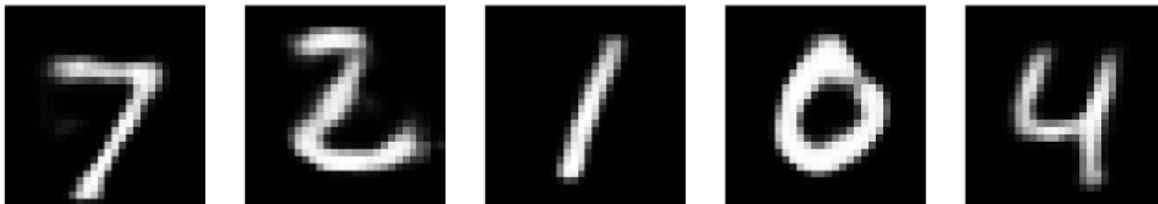
Original Images



Noisy Input

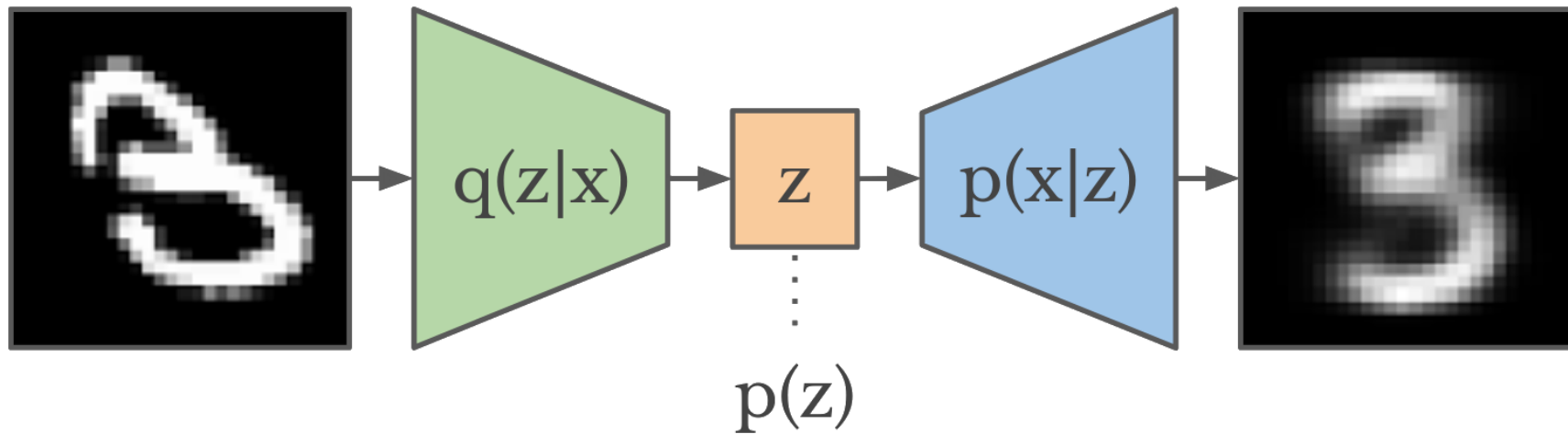


Autoencoder Output



Application

- Variational Auto Encoder (VAE)

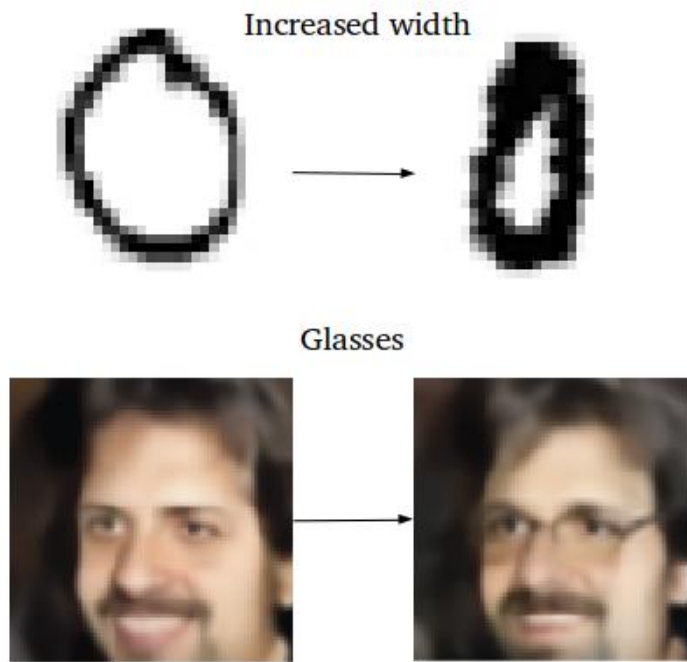


$$\text{minimize } \text{Loss} = -E_{q(z|x)} [\ln p(x|z)] + D_{KL}[q(z|x)||p(z)]$$

z : latent variable D_{KL} : Kullback – Leibler divergence

Application

- Variational Auto Encoder (VAE)

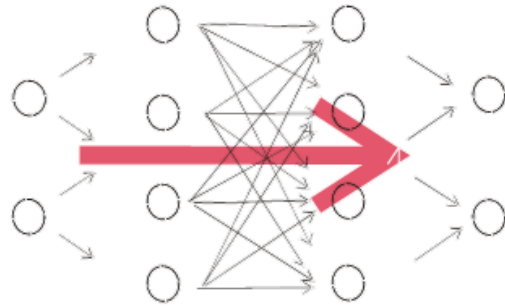


With variation



Generated celebrity-lookalike images

Feedforward network



Feedfowrd networks

Recurrent Neural Network

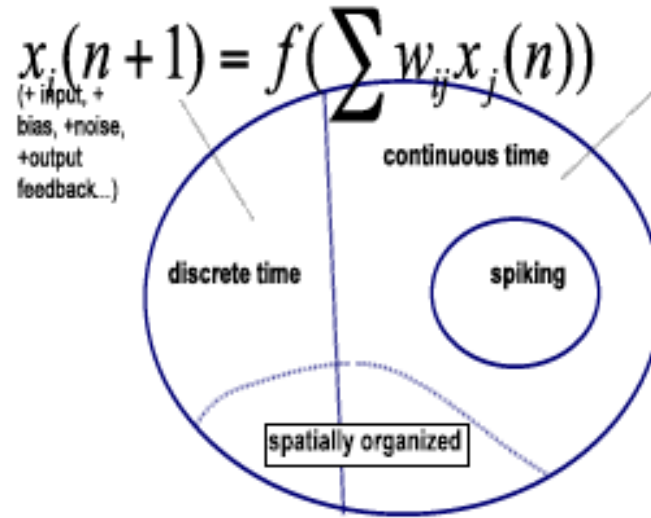


- mathematically they implement static input-output mapping
- Multi-layer perceptron(MLP) can approximate arbitrary nonlinear functions with arbitrary precision
- Most popular supervised training algorithm: backpropagation algorithm
 - Most (95%?) of neural network publication concern feedforward net
 - have proven useful in many practical applications as pattern classifications

Recurrent network

- all biological neural networks are recurrent
- mathematically, RNNs implement dynamical systems
- basic theoretical result: RNNs can approximate arbitrary (term needs some qualification) dynamical systems with arbitrary precision ("universal approximation property")
- several types of training algorithms are known, no clear winner
- theoretical and practical difficulties by and large have prevented practical applications so far

Formal description of RNN



$$\tau \dot{x}_i = -x_i + \sum w_{ij} f(x_j)$$

$n=1,2,3,\dots$ denotes the time

K input units

$$\mathbf{u}(n) = (u_1(n), \dots, u_K(n))^t$$

N internal units

$$\mathbf{x}(n) = (x_1(n), \dots, x_N(n))^t$$

L output units

$$\mathbf{y}(n) = (y_1(n), \dots, y_L(n))^t$$

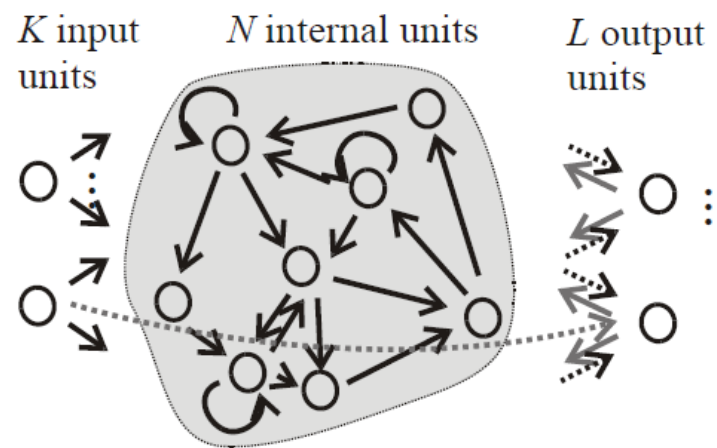
$$\mathbf{W}^{in} = (w_{ij}^{in}), \quad \mathbf{W} = (w_{ij}), \quad \mathbf{W}^{out} = (w_{ij}^{out}), \quad \mathbf{W}^{back} = (w_{ij}^{back}).$$

$N \times K$

$N \times N$

$L \times (N+K)$

$N \times L$



Updates of internal units

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{in} \mathbf{u}(n+1) + \mathbf{W} \mathbf{x}(n) + \mathbf{W}^{back} \mathbf{y}(n))$$

Input

internal

output

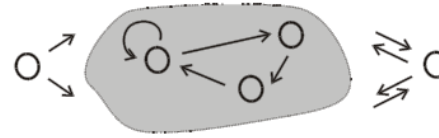
The output

$$\mathbf{y}(n+1) = \mathbf{f}^{out}(\mathbf{W}^{out}(\mathbf{u}(n+1), \mathbf{x}(n+1))),$$

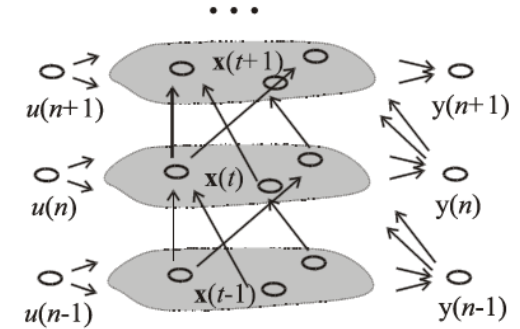
$\mathbf{f} = \tanh$ or 1

Training of recurrent network

Backpropagation through time (BPTT) method



**Unfold the recurrent network in time,
by stacking identical copies of RNN !**



Teacher data $\mathbf{u}(n) = (u_1(n), \dots, u_K(n))'$, $\mathbf{d}(n) = (d_1(n), \dots, d_L(n))'$ $n = 1, \dots, T$

The error to be minimized $E = \sum_{n=1, \dots, T} \|\mathbf{d}(n) - \mathbf{y}(n)\|^2 = \sum_{n=1, \dots, T} E(n)$

where

$$\mathbf{y}(n+1) = \mathbf{f}^{out}(\mathbf{W}^{out}(\mathbf{u}(n+1), \mathbf{x}(n+1))),$$

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{in} \mathbf{u}(n+1) + \mathbf{W} \mathbf{x}(n) + \mathbf{W}^{back} \mathbf{y}(n)),$$

Then the algorithm is straightforward feedforward backpropagation algorithm.

$$\delta_j(T) = (d_j(T) - y_j(T)) \frac{\partial f(u)}{\partial u} \Big|_{u=z_j(T)} \quad \text{error for the output units}$$

$$\delta_i(T) = \left[\sum_{j=1}^L \delta_j(T) w_{ji}^{out} \right] \frac{\partial f(u)}{\partial u} \Big|_{u=z_i(n)} \quad \text{error for internal units } x_i(t)$$

$$\delta_j(n) = \left[(d_j(n) - y_j(n)) + \sum_{i=1}^N \delta_i(n+1) w_{ij}^{back} \right] \frac{\partial f(u)}{\partial u} \Big|_{u=z_j(n)} \quad \text{error for the output units of earlier layer}$$

$$\delta_i(n) = \left[\sum_{j=1}^N \delta_j(n+1) w_{ji} + \sum_{j=1}^L \delta_j(n) w_{ji}^{out} \right] \frac{\partial f(u)}{\partial u} \Big|_{u=z_i(n)} \quad \text{error for the internal units of earlier times}$$

Adjustment

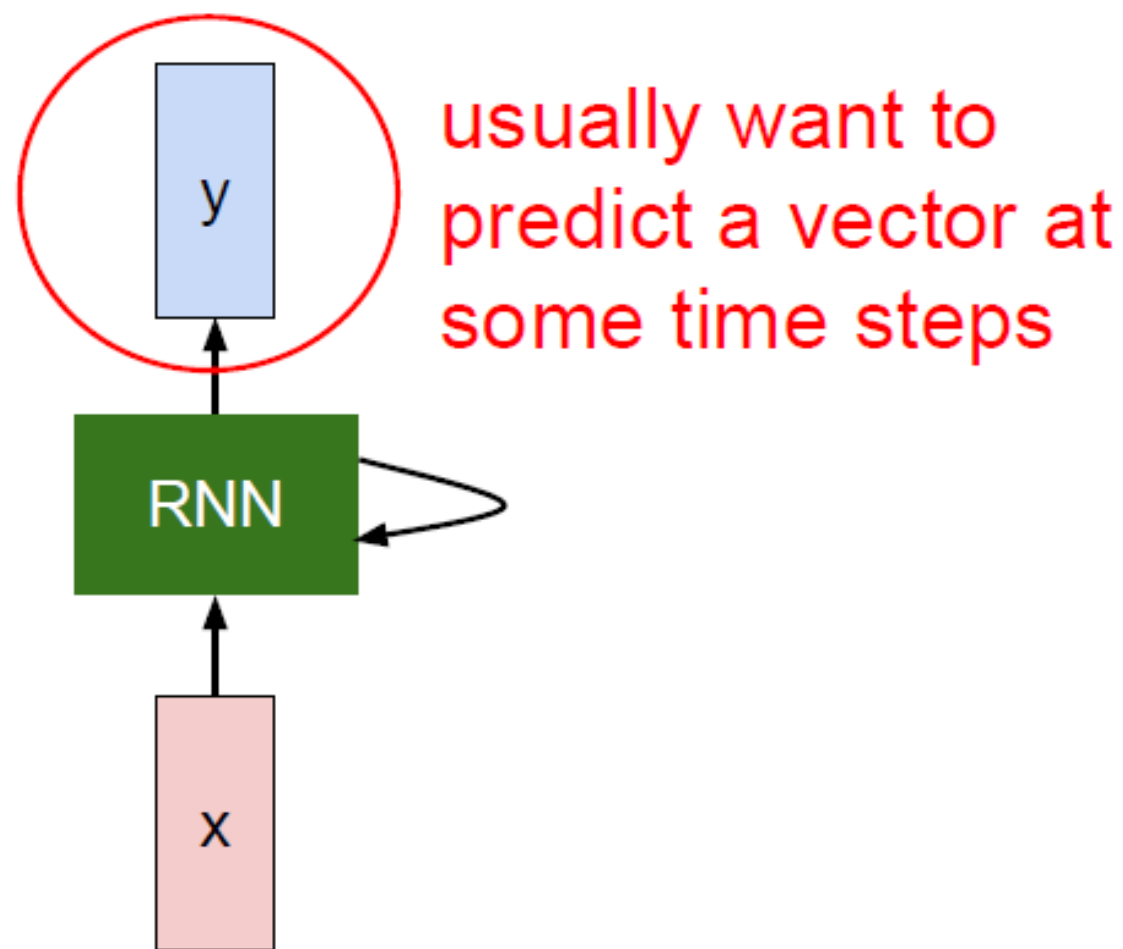
$$new\ w_{ij} = w_{ij} + \gamma \sum_{n=1}^T \delta_i(n) x_j(n-1) \quad [use\ x_j(n-1) = 0\ for\ n = 1]$$

$$new\ w_{ij}^{in} = w_{ij}^{in} + \gamma \sum_{n=1}^T \delta_i(n) u_j(n)$$

$$new\ w_{ij}^{out} = w_{ij}^{out} + \gamma \times \begin{cases} \sum_{n=1}^T \delta_i(n) u_j(n), & \text{if } j \text{ refers to input unit} \\ \sum_{n=1}^T \delta_i(n) x_j(n), & \text{if } j \text{ refers to hidden unit} \end{cases}$$

$$new\ w_{ij}^{back} = w_{ij}^{back} + \gamma \sum_{n=1}^T \delta_i(n) y_j(n-1) \quad [use\ y_j(n-1) = 0\ for\ n = 1]$$

Recurrent Neural Network



Recurrent Neural Network

We can process a sequence of vectors \mathbf{x} by applying a recurrence formula at every time step:

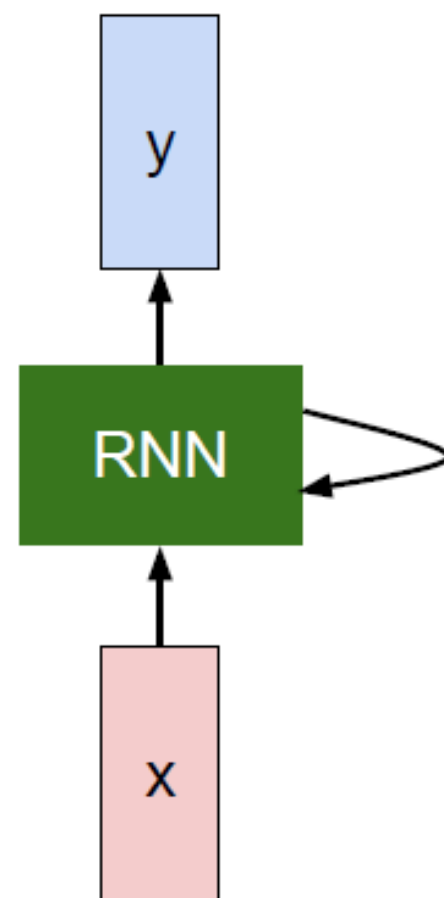
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

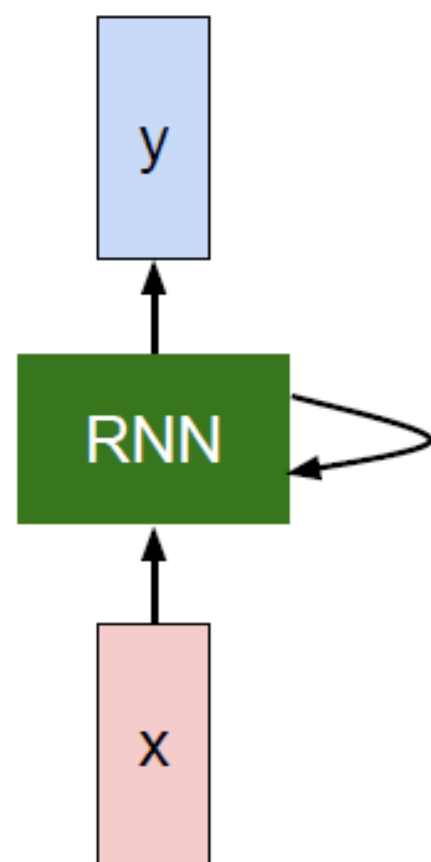
old state

input vector at some time step



(Vanilla) Recurrent Neural Network

The state consists of a single “*hidden*” vector \mathbf{h} :



$$h_t = f_W(h_{t-1}, x_t)$$



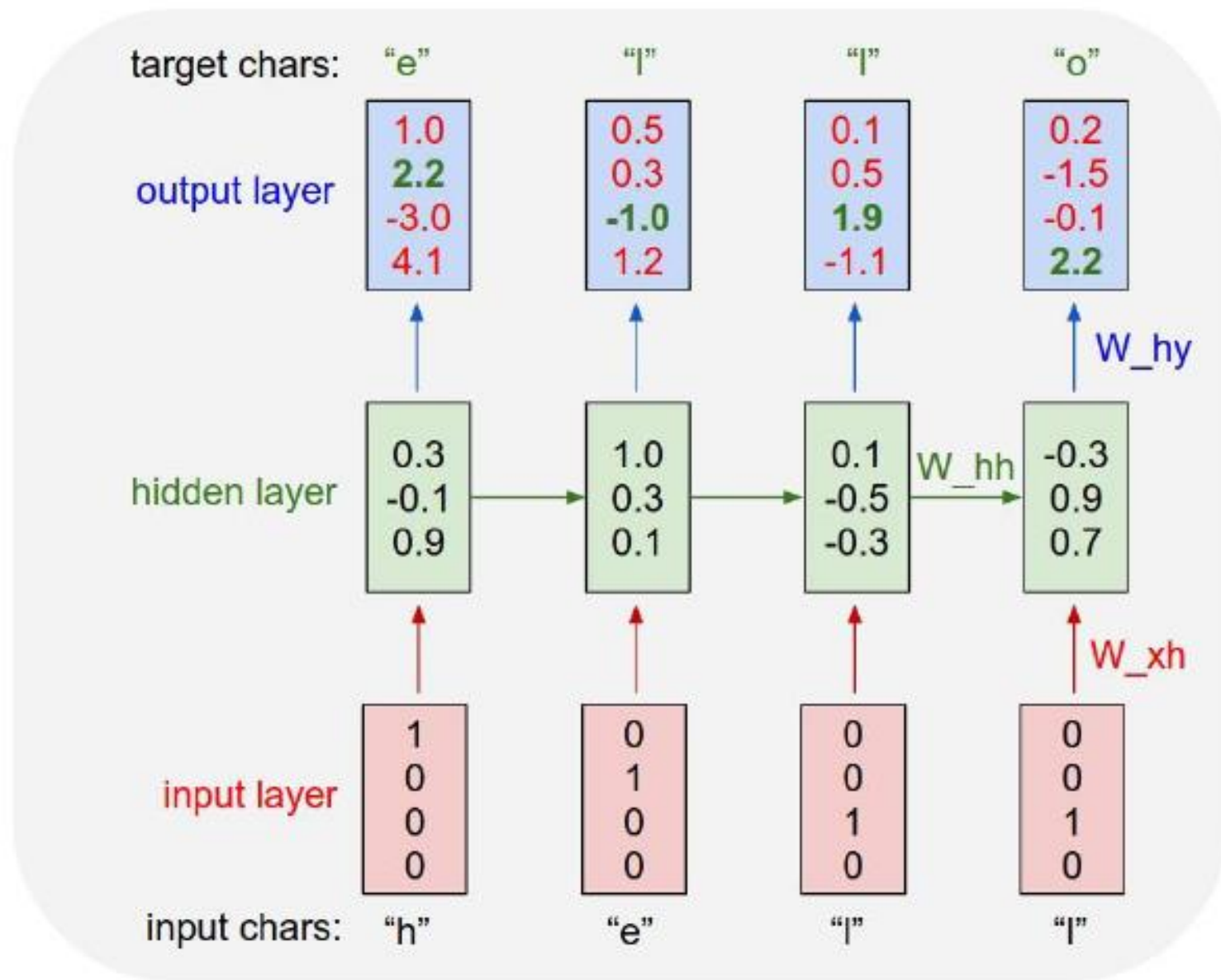
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

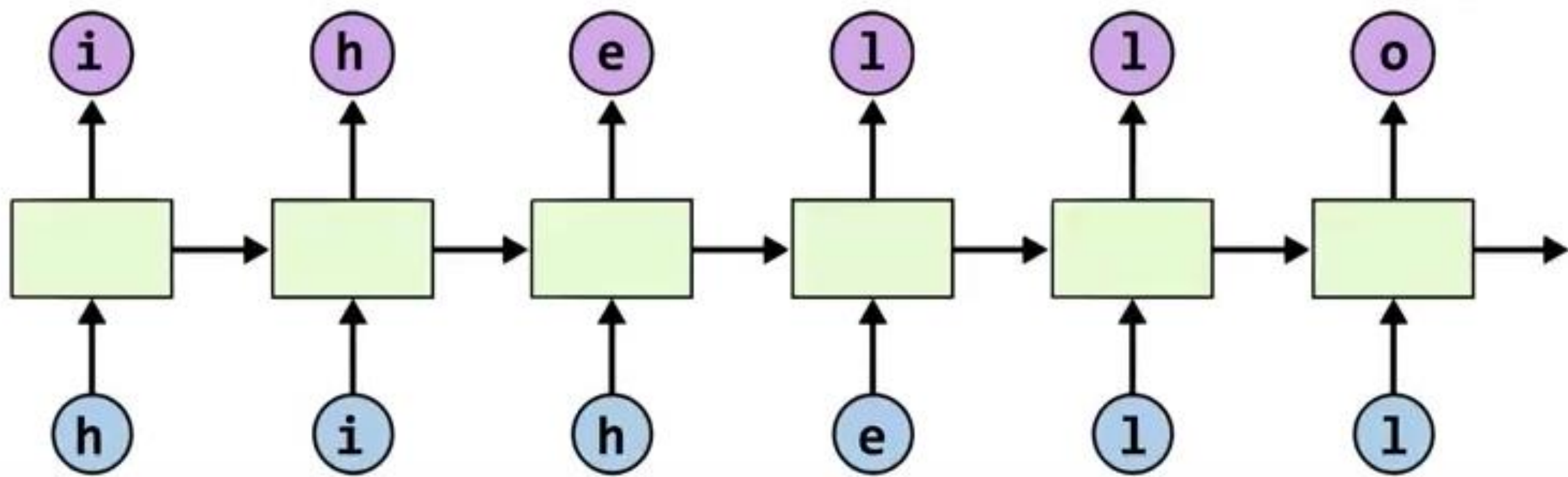
Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



Teach RNN 'hihello'



Manual data creation

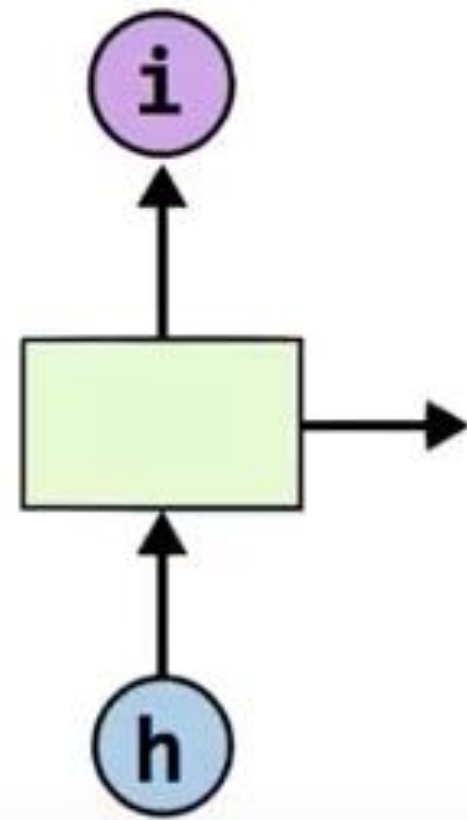
```
idx2char = ['h', 'i', 'e', 'l', 'o']  
x_data = [[0, 1, 0, 2, 3, 3]] # hiheLL  
① x_one_hot = [[ [1, 0, 0, 0, 0], # h 0  
                  [0, 1, 0, 0, 0], # i 1  
                  [1, 0, 0, 0, 0], # h 0  
                  [0, 0, 1, 0, 0], # e 2  
                  [0, 0, 0, 1, 0], # l 3  
                  [0, 0, 0, 1, 0]] # l 3  
  
y_data = [[1, 0, 2, 3, 3, 4]] # ihello
```

TensorFlow Coding

```
hidden_size = 5      # output from the LSTM
input_dim = 5         # one-hot size
batch_size = 1        # one sentence
sequence_length = 6   # |ihello| == 6
```

아무개수나

```
y_data = [[1, 0, 2, 3, 3, 4]]      # ihello
X = tf.placeholder(tf.float32,
                   [None, sequence_length, input_dim]) # X one-hot
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label
```



```
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size,  
state_is_tuple=True)  
initial_state = cell.zero_state(batch_size, tf.float32)  
outputs, _states = tf.nn.dynamic_rnn(  
cell, X, initial_state=initial_state, dtype=tf.float32)
```

5

RNN 을 작동시키기

Cell 과 X를 받아 output 으로


```
outputs, _states = tf.nn.dynamic_rnn(  
    cell, X, initial_state=initial_state, dtype=tf.float32)  
weights = tf.ones([batch_size, sequence_length])  
  
sequence_loss = tf.contrib.seq2seq.sequence_loss(  
    logits=outputs, targets=Y, weights=weights)  
loss = tf.reduce_mean(sequence_loss)  
train = tf.train.AdamOptimizer(learning_rate=0.1).minimize(loss)
```

Loss 를 계속 줄이도록 train 한다

RNN 에서 나온 결과를 one-hot sequence 형태로 만들고
그것의 target Y와의 차이를 loss 로 나타낸다

```

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(2000):
        l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
        result = sess.run(prediction, feed_dict={X: x_one_hot})
        print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

        # print char using dic
        result_str = [idx2char[c] for c in np.squeeze(result)]
        print("\tPrediction str: ", ''.join(result_str))

```

```

0 loss: 1.55474 prediction: [[3 3 3 3 4 4]] true Y: [[1, 0, 2, 3, 3, 4]] Prediction str: 1111oo
1 loss: 1.55081 prediction: [[3 3 3 3 4 4]] true Y: [[1, 0, 2, 3, 3, 4]] Prediction str: 1111oo
2 loss: 1.54704 prediction: [[3 3 3 3 4 4]] true Y: [[1, 0, 2, 3, 3, 4]] Prediction str: 1111oo
3 loss: 1.54342 prediction: [[3 3 3 3 4 4]] true Y: [[1, 0, 2, 3, 3, 4]] Prediction str: 1111oo
...
1998 loss: 0.75305 prediction: [[1 0 2 3 3 4]] true Y: [[1, 0, 2, 3, 3, 4]] Prediction str: ihello
1999 loss: 0.752973 prediction: [[1 0 2 3 3 4]] true Y: [[1, 0, 2, 3, 3, 4]] Prediction str: ihello

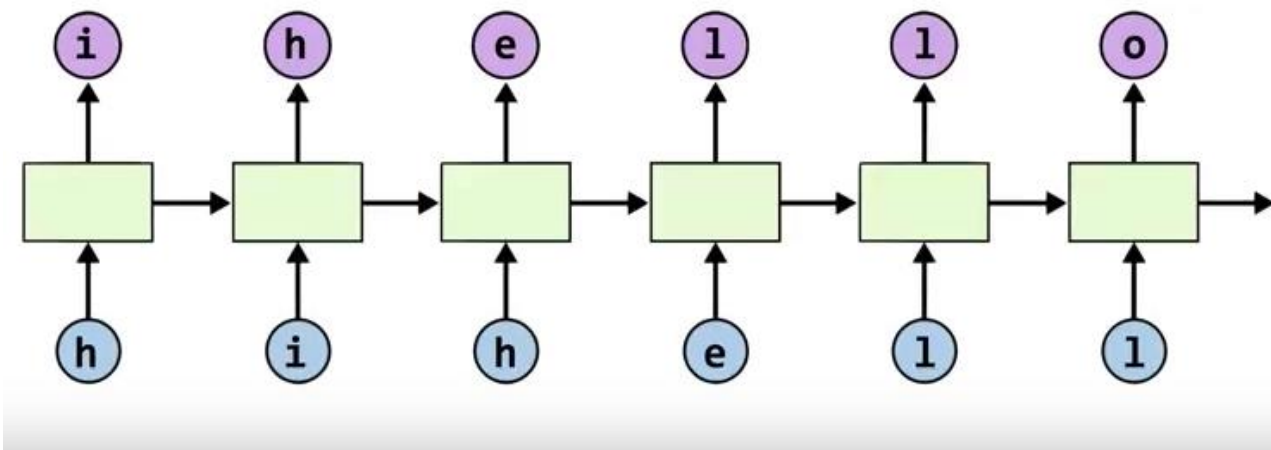
```


다음 중 괄호 안에 들어갈 말을 고르시오

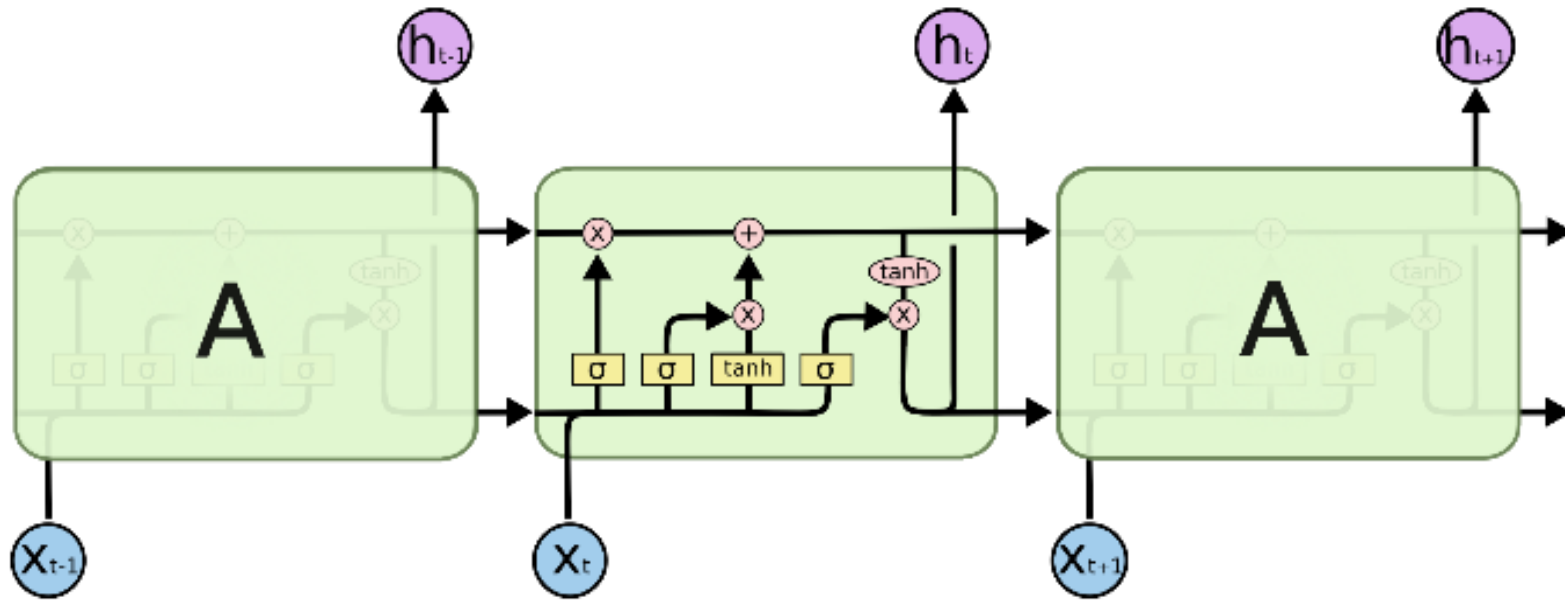
Hi hello I was born in Korea. That is why I can speak ().

1) English 2) Korean 3) Italian 4) hujlnbhdtyrghijklknjmb

지금 까지 보여준 RNN 은 기초적인 RNN 또는 Vanilla RNN 이라고 불리며 단기적인 기억만 가지고 있음.



Long and Short Term Memory

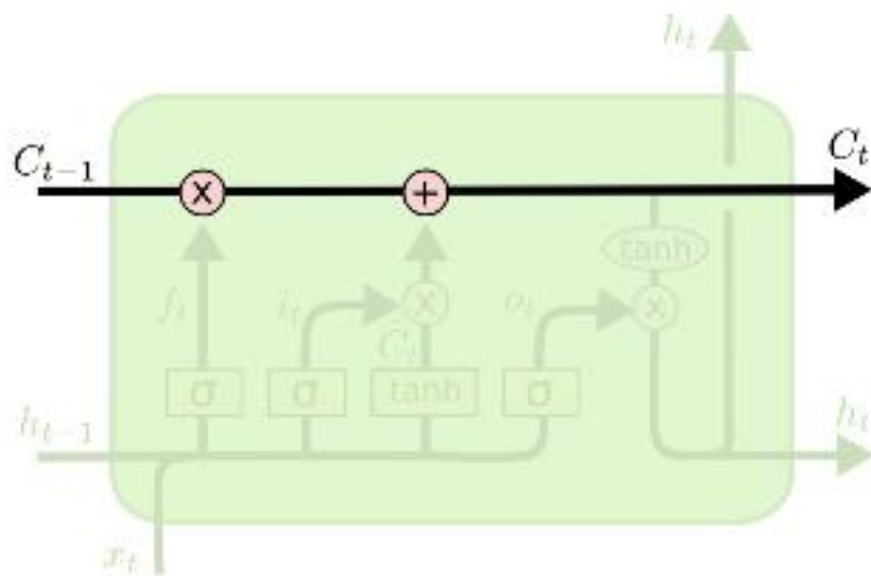


The repeating module in an LSTM contains four interacting layers.



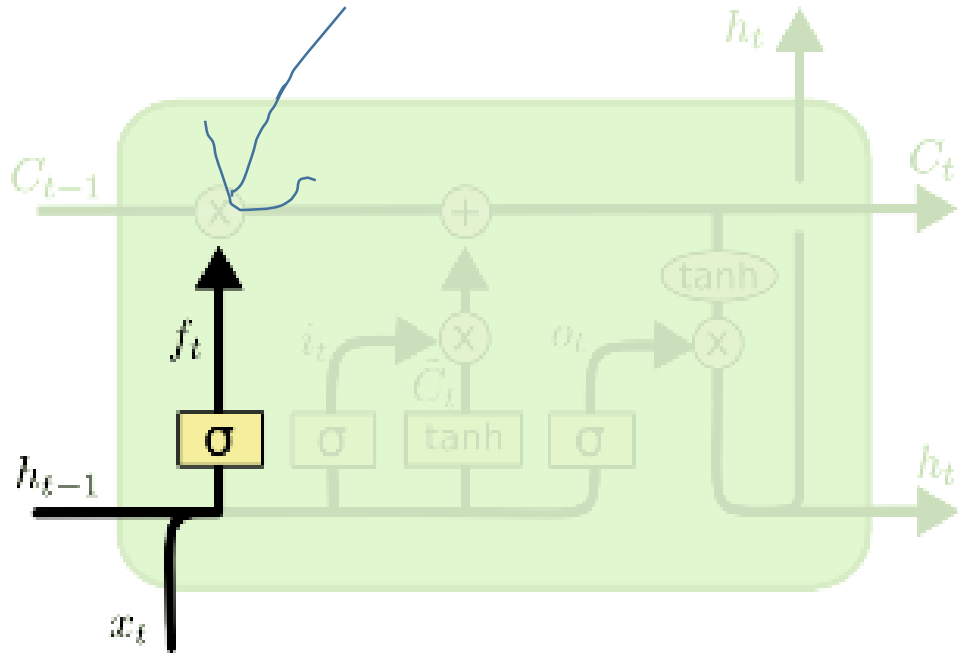
Juergen Schmidhuber

Cell 상태가 있어서 긴 시간의 기억을 가질 수 있게 된다.

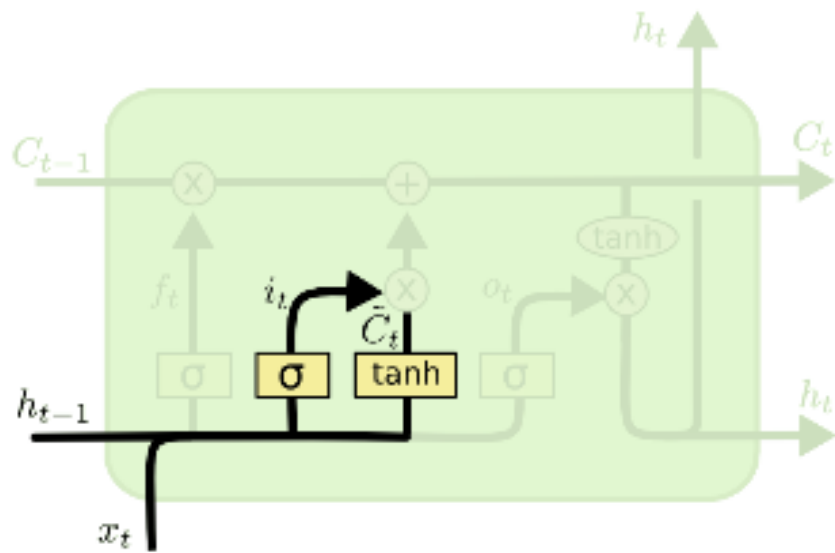


The LSTM does have the ability to remove or add information to the cell state

f 값이 0 이 되면 과거 셀의 정보가 완전히 없어진다. 과거의 정보를 잊는 forget gate.



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

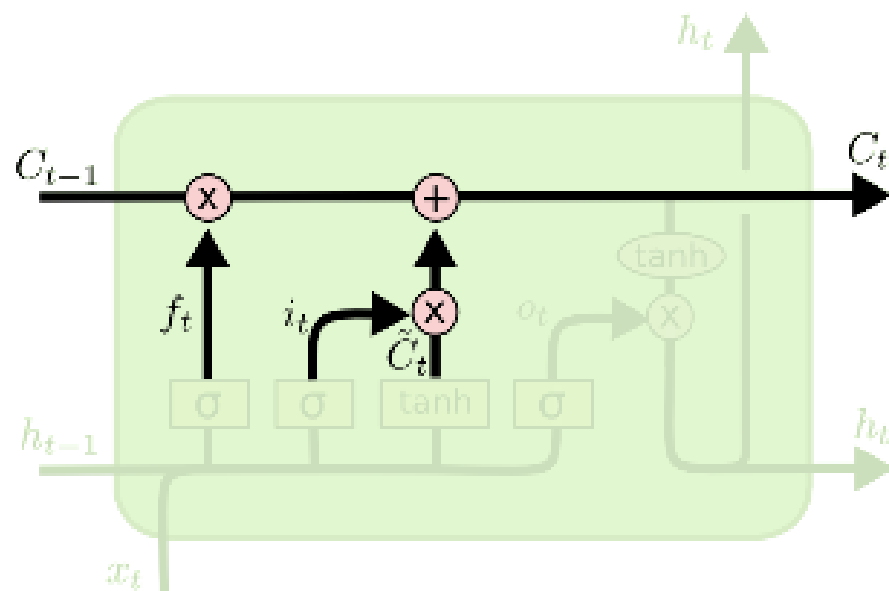


0 또는 1 이 되어서 정보를 살릴지 말지 결정

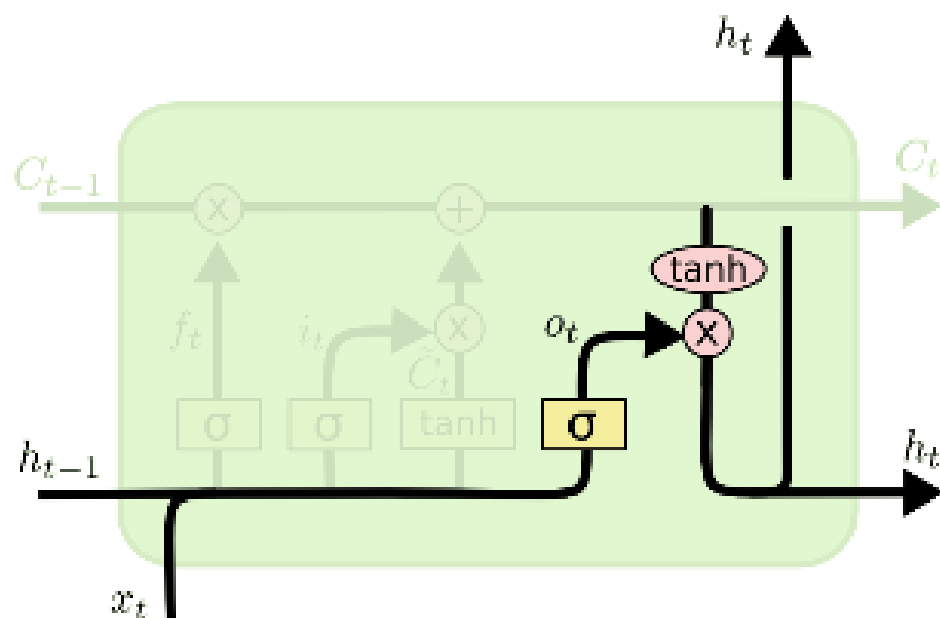
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

새로운 Cell 값 후보



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

This output will be based on our cell state, but will be a filtered version.
First, we run a sigmoid layer which decides what parts of the cell state we're going to output.

at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrge t o idoe ns,smtt h ne etie h,hregtrs nigtkie,aoaenns lng



train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuw y fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."



train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and offer.



train more

"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nudes begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

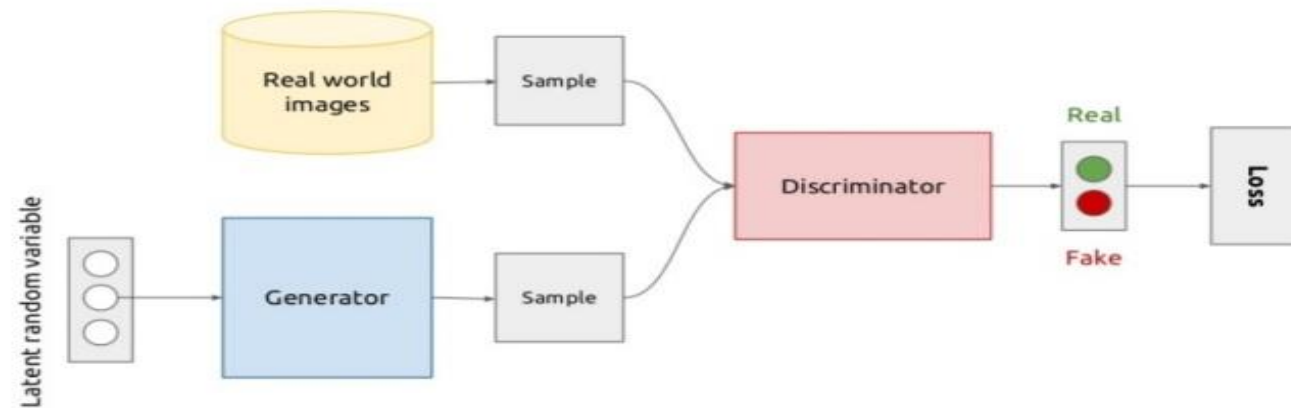
Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

이게 진짜 셰익스피어가 쓴 희곡일까요?
아닐까요? ㅋㅋㅋ

Generative Adversarial Network



<http://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016>

A Generative Adversarial Nets (GAN) consists of two models, a discriminator and generator. The discriminator and generator compete with each other to improve performance. A training sequence involves, the generator making fake data from real data, and the discriminator classifying the fake and real data. Therefore, we train the two networks by optimizing loss function,

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))],$$

Where D and G are the discriminator and generator, respectively, x is real data and z is latent variables, and D(x) is a probability that the input x is real data. G(z) has the same dimension as real data. First, we maximize V(D,G) by updating the parameters of the discriminator, and likewise minimize V(D,G) for the generator. It has been proven that solving the above equation produces equivalent fake and real data [1].

Proof)

$$\begin{aligned} V(G, D) &= \int_x p_{data}(x) \log(D(x)) dx + \int_z p_z(z) \log(1 - D(g(z))) dz \\ &= \int_x p_{data}(x) \log(D(x)) + p_g(x) \log(1 - D(x)) dx \end{aligned}$$

Where $p_g(x)dx = p_z(z)dz$ 가 되도록 p_g 를 선택한다.

$$\max_D V(G, D) = \int_x p_{data}(x) \log(D_G^*(x)) + p_g(x) \log(1 - D_G^*(x)) dx,$$

$$\text{where } D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

이건 D로 미분해서 알게됨.

$$\begin{aligned}
\max_D V(G, D) &= \mathbb{E}_{x \sim p_{data}(x)} [\log D_G^*(x)] + \mathbb{E}_{x \sim p_g} [\log(1 - D_G^*(x))] \\
&= \mathbb{E}_{x \sim p_{data}(x)} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \right] + \mathbb{E}_{x \sim p_g} \left[\log \frac{p_g(x)}{p_{data}(x) + p_g(x)} \right] \\
&= -\log(2) + \mathbb{E}_{x \sim p_{data}(x)} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} + \log(2) \right] - \log(2) \\
&\quad + \mathbb{E}_{x \sim p_g} \left[\log \frac{p_g(x)}{p_{data}(x) + p_g(x)} + \log(2) \right] \\
&= -\log(4) + KL \left(p_{data} \parallel \frac{p_{data} + p_g}{2} \right) + KL \left(p_g \parallel \frac{p_{data} + p_g}{2} \right) \\
&= -\log(4) + 2 \cdot JSD(p_{data} \parallel p_g)
\end{aligned}$$

그래서 최적상태를 구한다면 만들어지는 데이터의 분포는 p_g 실재 존재하는 데이터의 분포 p_{data} 와 같아진다. $p_g = p_{data}$

$$JSD(p \parallel q) = \frac{1}{2} KL(p \parallel M) + \frac{1}{2} KL(q \parallel M), \quad KL(p \parallel q) = \sum_i p_i \log \left(\frac{p_i}{q_i} \right), \quad M = \frac{1}{2} (p + q)$$



However, a practical GAN training differs from this theoretical process, so fake data vary widely from real data. Therefore, before programming a GAN, let's take a look at successful application first.

1) Least Square GAN

A GAN uses cross-entropy for the loss function $V(D,G)$ with sigmoid function on the output of the discriminator. In this case, since the real data and the fake data are very easy to distinguish at the beginning of training, D has a value close to 0, and the gradients become very small. Like the above GAN proof, we can get coefficients a , b , and c through solving the following optimization problems [2],

$$\begin{aligned} \min_D V_{LSGAN}(D) &= \frac{1}{2} \mathbb{E}_{x \sim p_{data}(x)} [(D(x) - b)^2] + \frac{1}{2} \mathbb{E}_{z \sim p_z(z)} [(D(G(z)) - a)^2] \\ \min_G V_{LSGAN}(G) &= \frac{1}{2} \mathbb{E}_{x \sim p_{data}(x)} [(D(x) - c)^2] + \frac{1}{2} \mathbb{E}_{z \sim p_z(z)} [(D(G(z)) - c)^2]. \end{aligned}$$

Practice 1)

Find the V_{LSGAN} for both discriminator and generator.

$$\min_D V_{LSGAN}(D) = \frac{1}{2} \mathbb{E}_{x \sim p_{data}(x)} [(D(x) - b)^2] + \frac{1}{2} \mathbb{E}_{z \sim p_z(z)} [(D(G(z)) - a)^2]$$

$$p_g(x)dx = p_z(z)dz$$

$$\int p_d(x)[(D(x) - b)^2] + p_g(x)[(D(x) - b)^2] dx$$

$$D^*(x) = \frac{bp_{data}(x) + ap_g(x)}{p_{data}(x) + p_g(x)}$$

$$\min_G V_{LSGAN}(G) = \frac{1}{2} \mathbb{E}_{x \sim p_{data}(x)} [(D(x) - c)^2] + \frac{1}{2} \mathbb{E}_{z \sim p_z(z)} [(D(G(z)) - c)^2].$$

Define

$$\begin{aligned} 2C(G) &= \mathbb{E}_{\mathbf{x} \sim p_d} [(D^*(\mathbf{x}) - c)^2] + \mathbb{E}_{\mathbf{x} \sim p_g} [(D^*(\mathbf{x}) - c)^2] \\ &= \int_{\mathcal{X}} p_d(\mathbf{x}) \left(\frac{(b-c)p_d(\mathbf{x}) + (a-c)p_g(\mathbf{x})}{p_d(\mathbf{x}) + p_g(\mathbf{x})} \right)^2 d\mathbf{x} + \int_{\mathcal{X}} p_g(\mathbf{x}) \left(\frac{(b-c)p_d(\mathbf{x}) + (a-c)p_g(\mathbf{x})}{p_d(\mathbf{x}) + p_g(\mathbf{x})} \right)^2 d\mathbf{x} \\ &= \int_{\mathcal{X}} \frac{((b-c)(p_d(\mathbf{x}) + p_g(\mathbf{x})) - (b-a)p_g(\mathbf{x}))^2}{p_d(\mathbf{x}) + p_g(\mathbf{x})} d\mathbf{x} \end{aligned}$$

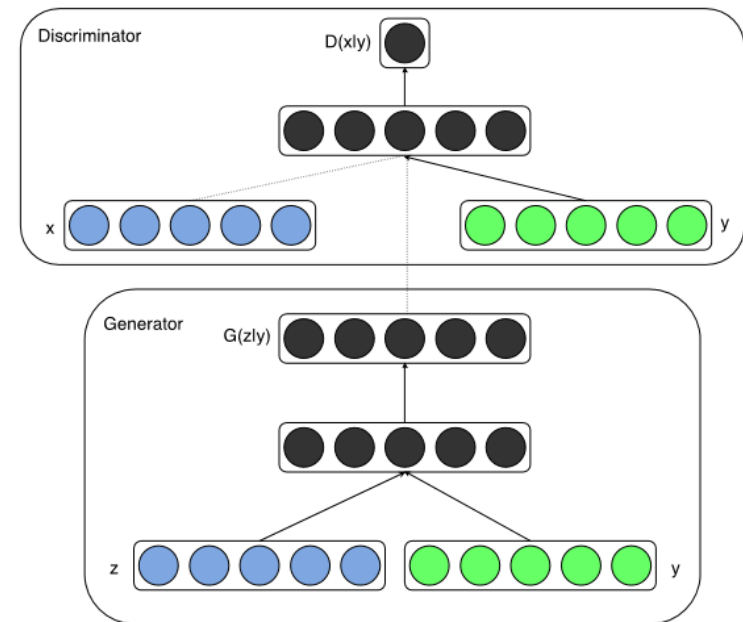
이것을 최소화 하는 것이 $p_g = p_d$ 가 되게 하려면 위의 적분이 다음과 같으면 된다.

$$= \int \frac{((p_d(x) + p_g(x)) - 2p_g(x))^2}{p_d(x) + p_g(x)} dx$$

그러므로 $b-c = 1$ $b-a=2$

2) Conditional GAN [3]

Let's consider that we successfully trained a GAN with the MNIST data set. A trained generator makes perfect hand written digits, but we cannot choose any particular digit. We can train the GAN with label information by conditioning the input of the discriminator and generator. In the case of MNIST, the discriminator gets images of digit and labels, and the generator gets latent variables and labels for the images.



3) Deep Convolutional GAN

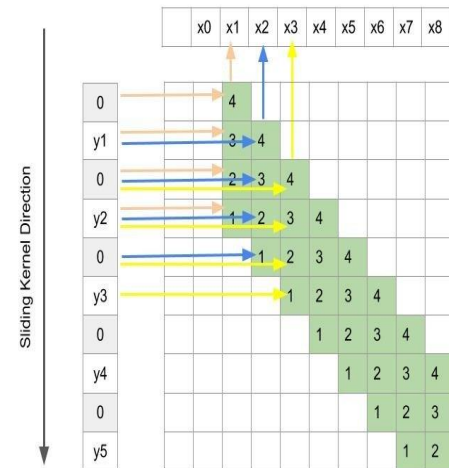
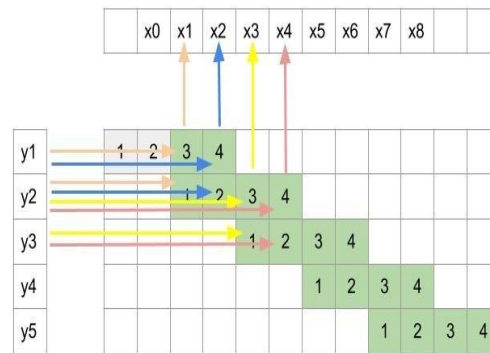
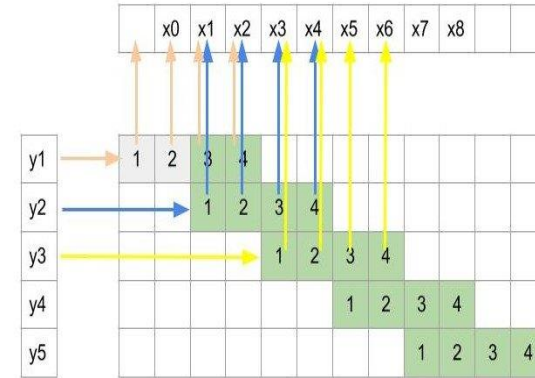
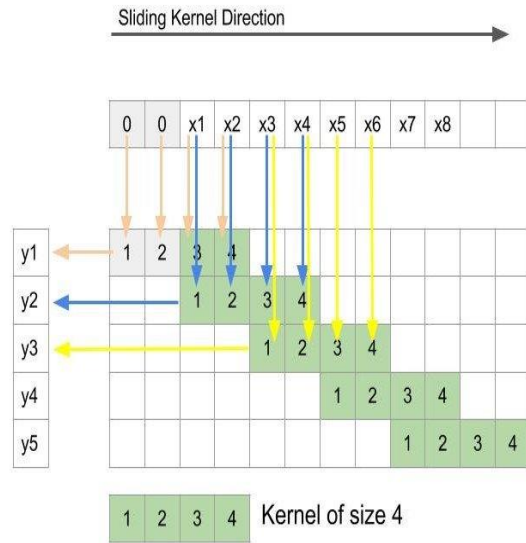
A Deep Convolutional GAN (DCGAN) [4], which has successfully trained many data sets (especially image data), has the following structure.

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Here, strided convolution means any convolution with a stride larger than 2. Strided convolution downsizes input into output, and fractional-strided convolution extends input into output. For a generator with a convolutional net, fractional-strided convolution can be used to match the output size with real data.

4) Fractional-strided convolution



We can use fractional-strided convolution as a built-in function in tensorflow.
`tf.nn.conv2d_transpose()`

arXiv:1712.06340

LANGUAGE AND NOISE TRANSFER IN SPEECH ENHANCEMENT GENERATIVE ADVERSARIAL NETWORK

Santiago Pascual¹, Maruchan Park², Joan Serrà³, Antonio Bonafonte¹, Kang-Hun Ahn²

¹ Universitat Politècnica de Catalunya, Barcelona, Spain

² Chungnam National University, Daejeon, Republic of Korea

³ Telefónica Research, Barcelona, Spain

ABSTRACT

Speech enhancement deep learning systems usually require large amounts of training data to operate in broad conditions or real applications. This makes the adaptability of those systems into new, low resource environments an important topic. In this work, we present the results of adapting a speech enhancement generative adversarial network by fine-

In previous work, we proposed an end-to-end speech enhancement system [6] based on a generative adversarial network [7] (GAN), namely speech enhancement generative adversarial network (SEGAN). SEGAN was proposed in the pursuit of end-to-end speech processing, where signal is enhanced at the raw waveform level, with a one-shot, non-recursive structure. It showed the applicability of latest deep

Training English speech & Result

Training data

Clean english speech : 9h

Noisy english speech : 9h

86 epochs

Test data

Unseen noisy english speech
(Different noise, speaker, sentence)

Result

noisy



enhanced



Data source :

C. Valentini-Botinhao, X. Wang, S. Takaki, and J. Yamagishi, "Investigating rnn-based speech enhancement methods for noiserobust text-to-speech," in 9th ISCA Speech Synthesis Workshop, pp. 146–152

C. Veaux, J. Yamagishi, and S. King, "The voice bank corpus: Design, collection and data analysis of a large regional accent speech database," in Int. Conf. Oriental COCODA, held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCODA/CASLRE). IEEE, 2013, pp. 1–4.

Subjective evaluation

19 listeners scored

Metric	Noisy	Wiener	SEGAN
MOS	2.41	2.89	3.16

Mean opinion score

1(bad quality) ~ 5(excellent quality)

Pascual, Santiago, Antonio Bonafonte, and Joan Serrà. "SEGAN: Speech Enhancement Generative Adversarial Network." *arXiv preprint arXiv:1703.09452* (2017).



Training Korean speech & Result

Training data

Pre-trained with English (86epochs)
Clean korean speech : 200m
Noisy korean speech : 200m
30 epochs

Test data

Unseen noisy korean speech
(Different noise, speaker, sentence)

Result

noisy



enhanced



noisy



enhanced



Erasing artillery sound



Erasing artillery sound

Data set

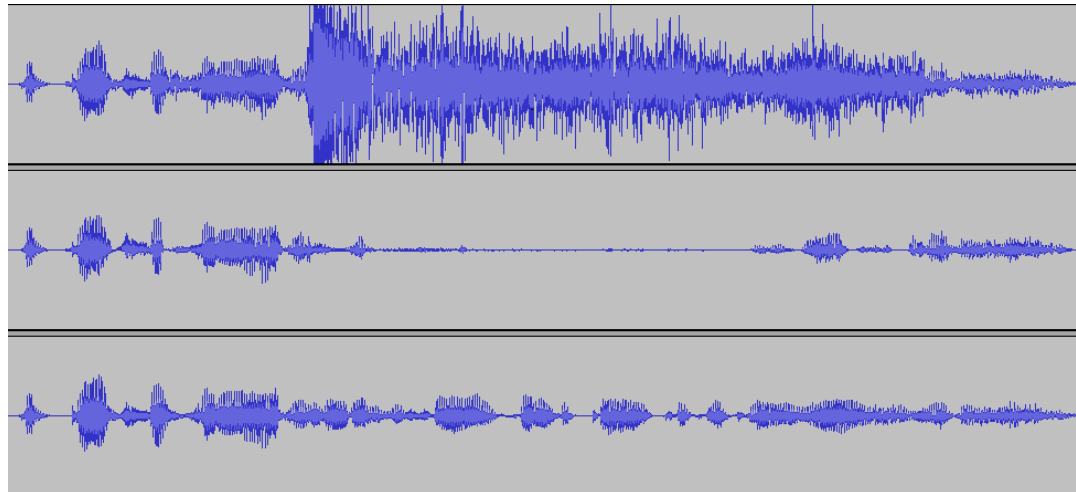
speech + artillery sound

54 Training data

4 Test data


noisy


enhanced





Other example

Enhanced : 

Noisy : 

Enhanced : 



Thank you