

# Deep Learning and Deep Neural Networks

Man-Wai MAK

Dept. of Electronic and Information Engineering,  
The Hong Kong Polytechnic University

[enmwmak@polyu.edu.hk](mailto:enmwmak@polyu.edu.hk)

<http://www.eie.polyu.edu.hk/~mwmak>

## References:

- I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.
- <http://www.eie.polyu.edu.hk/~mwmak/papers/bp.pdf>
- S.Y. Kung, M.W. Mak and S.H. Lin, *Biometric Authentication: A Machine Learning Approach*, Prentice Hall, 2005, Chapter 5.
- M.W. Mak and J.T. Chien, *Machine Learning for Speaker Recognition*, Cambridge University Press, 2020, Chapter 4.

November 8, 2020

## 1 Motivations

## 2 Deep Learning and DNN

- Neural Networks
- NN as Nonlinear Classifier
- Deep Neural Networks

## 3 Gradient Descent

- Training of DNN
- Simple Gradient Descent
- Stochastic Gradient Descent

## 4 Backpropagation

- Loss Function: Cross Entropy

## 5 Convolutional Neural Networks

# Motivations

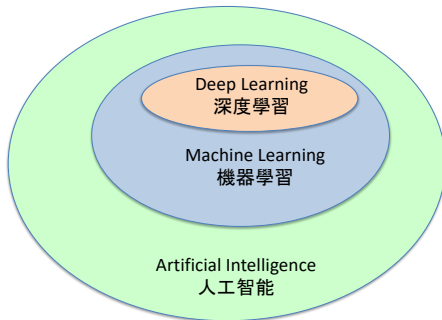
- Deep learning is currently the most exciting area in AI, perhaps because of the success of AlphaGO and Master.
- Deep learning finds its way in many disciplines, e.g.,
  - **Power Electronics:** Neural Network Applications in Power Electronics and Motor Drives: An Introduction and Perspective,  
<http://www.nvidia.com/object/drive-px.html>
  - **Renewable Energy:**  
<https://www.youtube.com/watch?v=pghjLyAmc5g>
  - **Autonomous Vehicle:**  
<https://www.youtube.com/watch?v=DjAJnQoNdMA>
  - **Multimedia:** DNNs win almost all competitions in computer vision, face recognition, speech recognition, natural language processing, etc.
  - **Big-Data Analytics:** Recommendation systems, customer relationship management.

- Why does deep learning work so well? <http://fortune.com/ai-artificial-intelligence-deep-machine-learning/>
- Why you need to learn deep learning?  
<http://fortune.com/2016/10/05/ai-artificial-intelligence-deep-learning-employers/>
- Nvidia GTC Keynote Speech:  
<https://www.nvidia.com/en-us/gtc/keynote/>



# Deep Learning, Machine Learning and AI

- Deep learning is a branch of machine learning algorithms for training deep neural networks.



- Conventionally, we have shallow networks such as multi-layer perceptrons (discussed later), SVMs, and GMMs.
- Deep neural networks have narrow structure but many layers of non-linear processing elements.

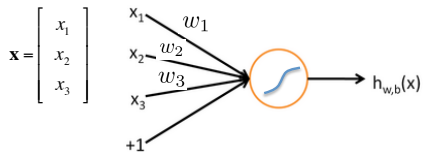
# What Makes DNN Successful?

- More complex models can be build, provided that we have more data
- In the big data era, we have a large amount of data
- With GPUs and the cloud, we also have a large amount of computational resources
- Open source efforts (e.g., Theano, TensorFlow, Keras, Torch, CNTK, and Kaldi) also facilitate the development of complex models.

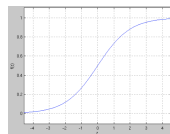
# Neural Networks

# Neural Networks

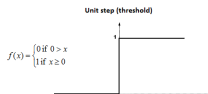
- Artificial neural networks (ANNs) are a family of statistical learning models inspired by biological neural networks (the brain).
- They are used to approximate some unknown functions that depend on a large number of inputs
- The simplest type of neural networks, called **perceptron**, has one neuron only:



$$h_{\mathbf{w},b}(\mathbf{x}) = f(\mathbf{w}^T \mathbf{x} + b) = f\left(\sum_{i=1}^3 w_i x_i + b\right)$$

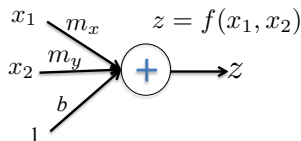
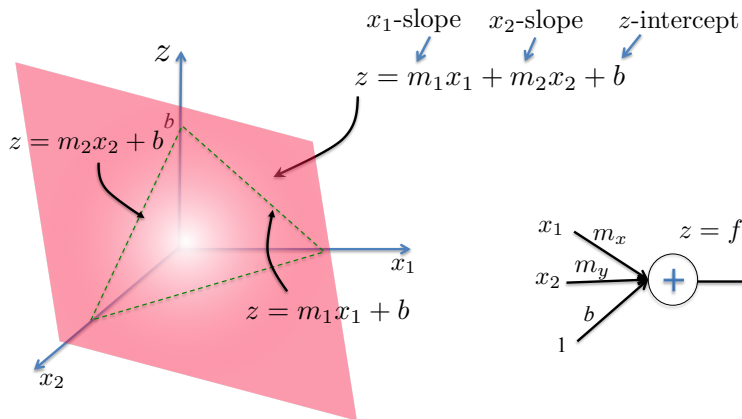


$$f(z) = \frac{1}{1 + e^{-z}}$$



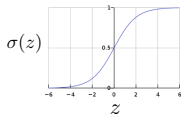
# Geometric Interpretation of Perceptrons

- For 2-D inputs, we may use coordinate geometry to understand how a perceptron works.

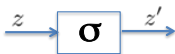
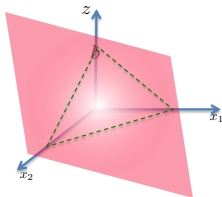
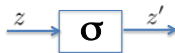


# Geometric Interpretation of Perceptrons

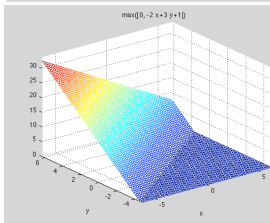
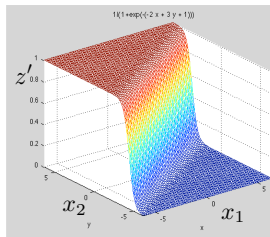
- Apply non-linear activation function on  $z$ :



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

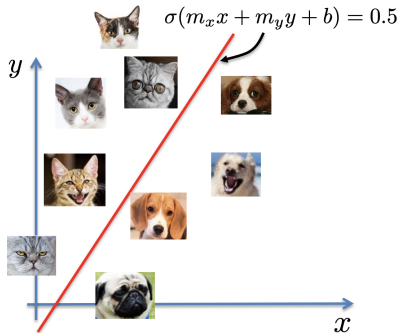
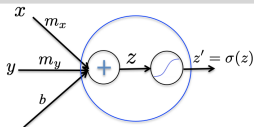
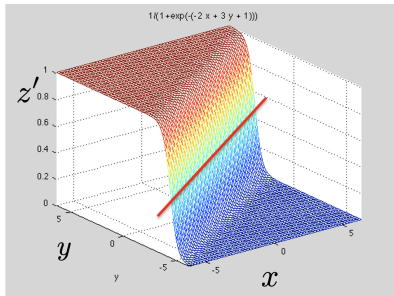


$$\sigma(z) = \max\{z, 0\}$$



# Capability of Perceptrons

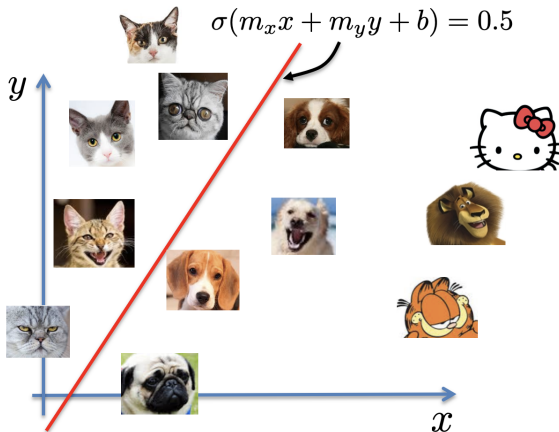
- We may apply a perceptron for binary classification



$$\begin{aligned}\sigma(m_x x + m_y y + b) &\geq 0.5 \Rightarrow \text{cat} \\ \sigma(m_x x + m_y y + b) &< 0.5 \Rightarrow \text{dog}\end{aligned}$$

# Capability of Perceptrons

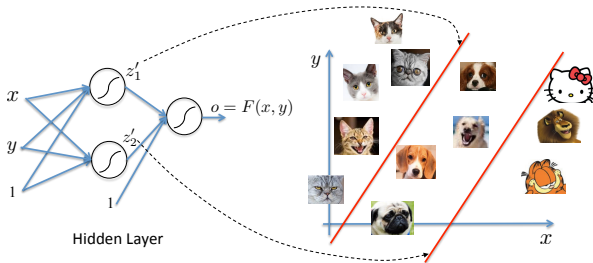
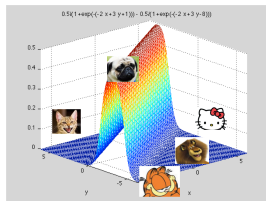
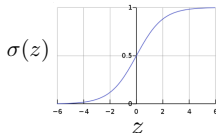
- A single-layer neural network (like a perceptron) cannot do much. It fails in the following simple problem.





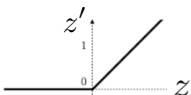
# Neural Networks with 1 Hidden Layer

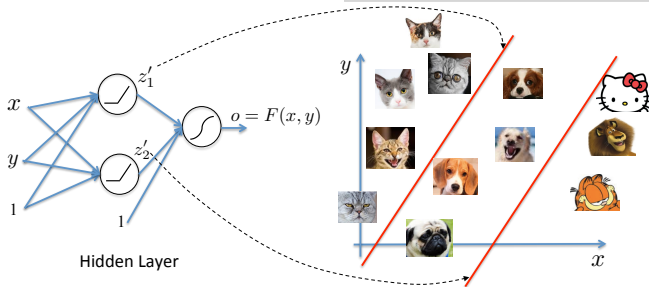
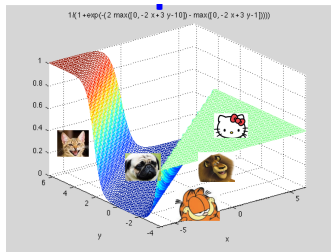
- But magic happens if we add one more layer.



- Now each hidden neuron creates one decision plane.

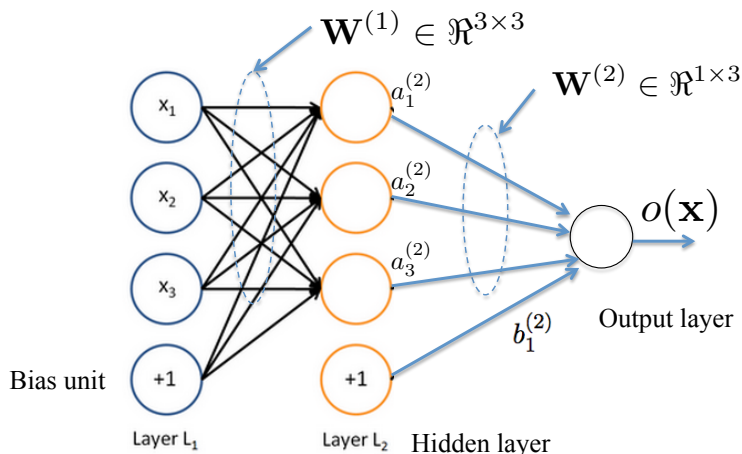
# Neural Networks with 1 Hidden Layer


$$\sigma(z) = \max\{z, 0\}$$



# Neural Networks with 1 Hidden Layer

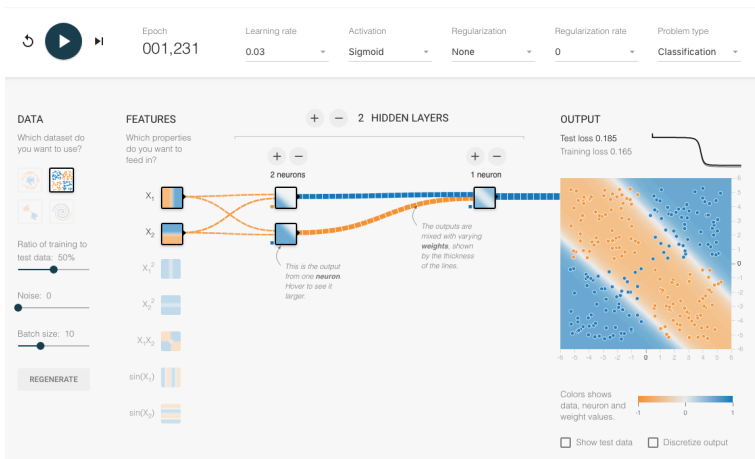
- A neural network is constructed by hooking many neurons together.



- Both input and output layers can have multiple nodes.

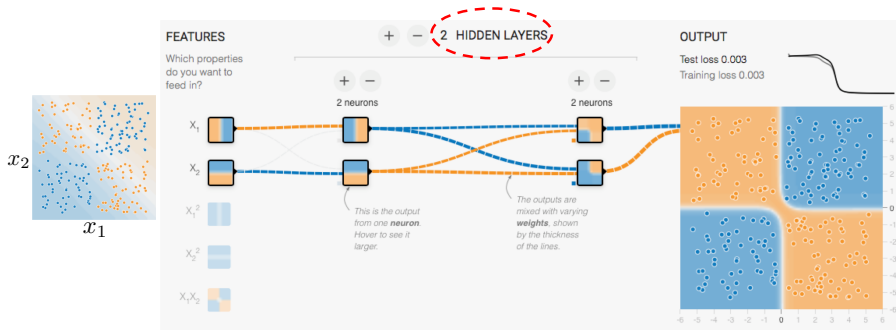
# XOR Problem

- Shadow Network (<http://playground.tensorflow.org>):



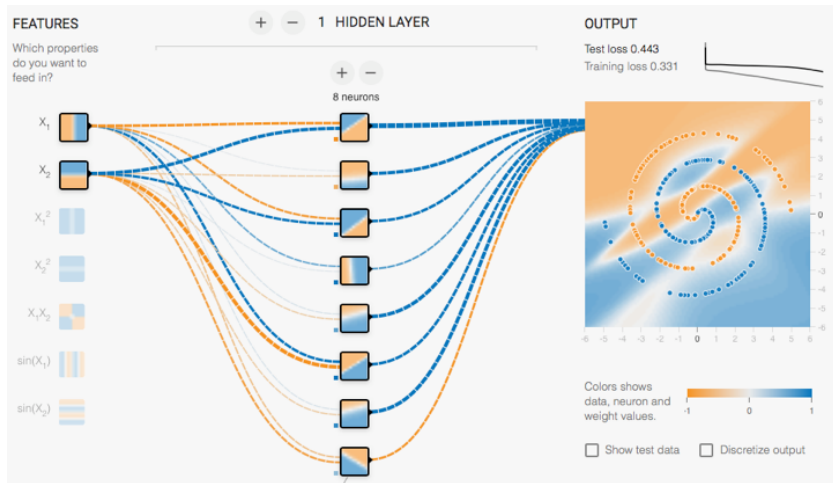
# XOR Problem

- Going Deeper:



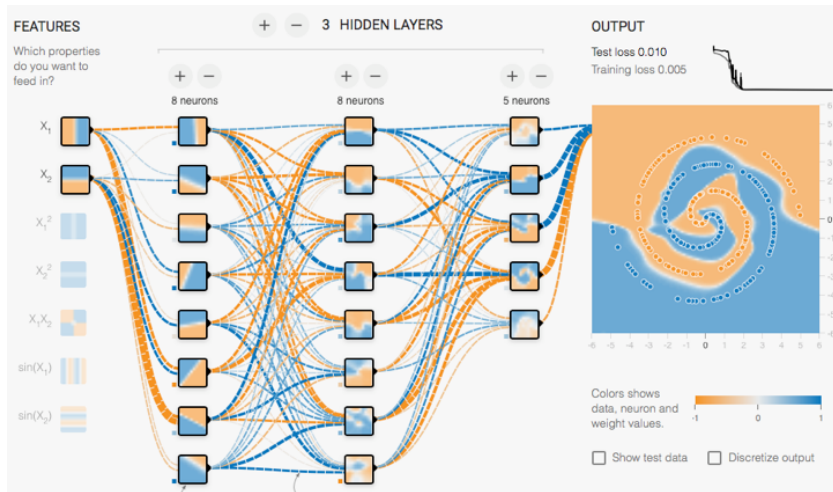
# 2-D Spiral Problem

- Shadow Network:



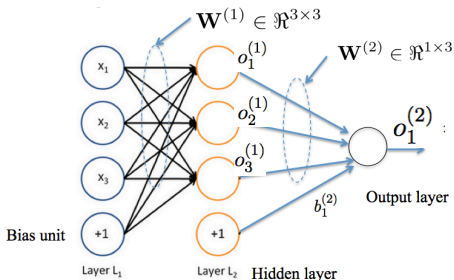
# 2-D Spiral Problem

- Going Deeper:



# Forward Propagation in Neural Networks

- Forward propagation:



$$\begin{aligned} o_1^{(1)} &= f\left(\sum_{i=1}^3 w_{1i}^{(1)} x_i + b_1^{(1)}\right) \\ o_2^{(1)} &= f\left(\sum_{i=1}^3 w_{2i}^{(1)} x_i + b_2^{(1)}\right) \\ o_3^{(1)} &= f\left(\sum_{i=1}^3 w_{3i}^{(1)} x_i + b_3^{(1)}\right) \\ o_1^{(2)} &= f\left(\sum_{j=1}^3 w_{1j}^{(2)} o_j^{(1)} + b_1^{(2)}\right) \\ &\equiv y(\mathbf{x}) \end{aligned}$$

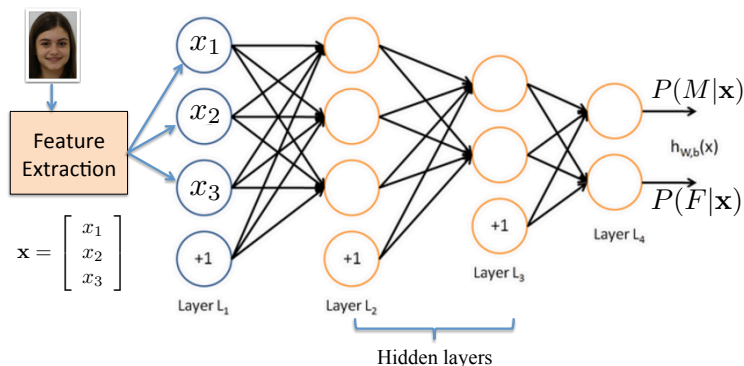
- General equations for forward propagation:

$$\begin{aligned} a_k^{(l)} &= (\mathbf{w}_k^{(l)})^\top \mathbf{o}^{(l-1)} + b_k^{(l)} \\ o_k^{(l)} &= f(a_k^{(l)}), \quad l = 1, 2 \\ \mathbf{o}^{(0)} &\equiv \mathbf{x} \end{aligned}$$



# NN as Nonlinear Classifier

- Networks can have multiple layers and multiple outputs:



- To train this network, we need training samples:

$$\{\mathbf{x}_n, \mathbf{t}_n\}_{n=1}^N, \quad \text{where } \mathbf{x}_n \in \mathbb{R}^3 \quad \text{and} \quad \mathbf{t}_n \in \mathbb{R}^2$$

- To ensure that the output nodes produce posterior probabilities given the input vectors, we typically apply a softmax function to the output nodes:

$$P(\text{Class} = k|\mathbf{x}) \equiv y_k = o_k^{(L)} = \frac{\exp(a_k^{(L)})}{\sum_j \exp(a_j^{(L)})}$$

where  $a_k^{(L)}$  is the activation of the  $k$ -th output node (at layer  $L$ ).

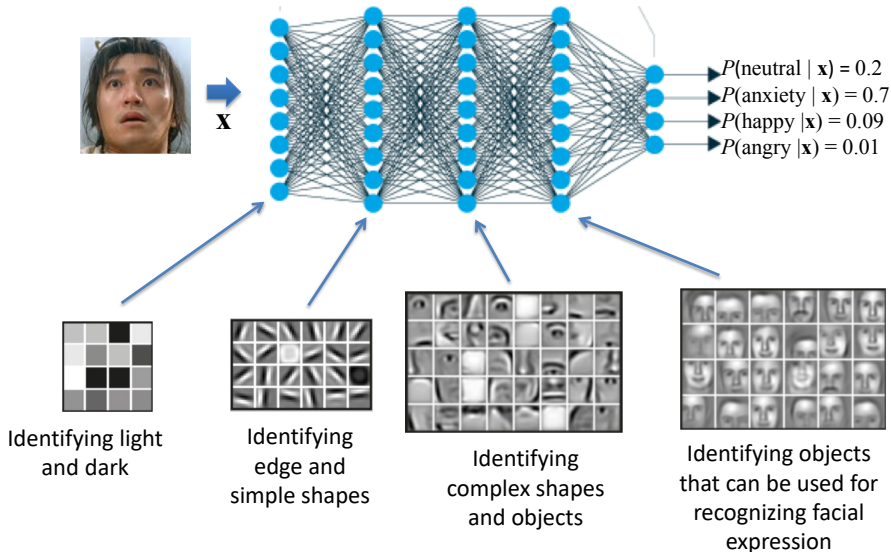
# Deep Neural Networks

# Deep Neural Networks

- Both SVM and GMM can be considered as networks with one hidden layer
- Deep neural networks (DNNs) are networks with many hidden layers (22 layers in GoogLeNet)
- Since the 80's, researchers knew that DNNs with many hidden layers are very powerful as long as they can be trained.
- From the 80's to 2005, NNs were trained by the backpropagation (BP) algorithm.
- Unfortunately, BP fails to train any networks with more than one hidden layer. People found that shallow networks such as SVM and GMM perform much better than NN with multiple layers.

- This situation has changed in 2006 when G. Hinton discovered a new approach to training DNNs.
- The method uses unsupervised learning to initialize the DNN and use BP to fine-tune the network weights.
- During the unsupervised learning stage, the network is built one layer at a time (from bottom to top) by stacking a number of Restricted Boltzmann Machines (RBM)
- The RBM is trained by an algorithm called contrastive divergence.
- For details of RBM and contrastive divergence, see <http://www.eie.polyu.edu.hk/~mwmak/papers/RBM.pdf>

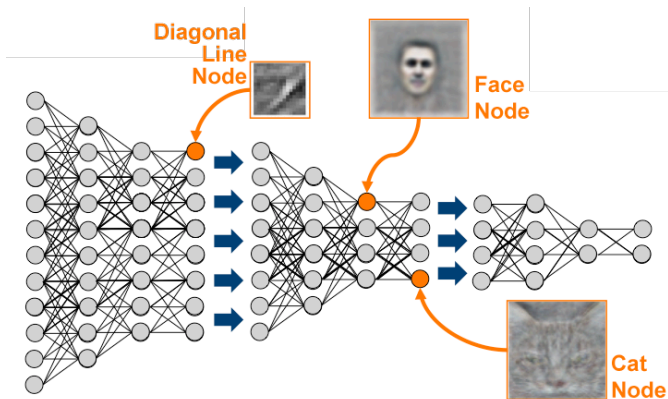
# Internal Representation in DNNs



<http://www.nature.com/news/computer-science-the-learning-machines-1.14481>

10

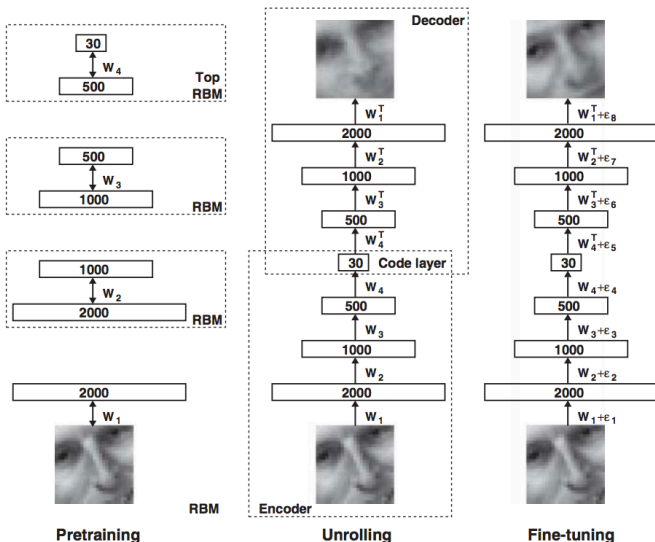
# Internal Representation in DNNs



- Data presented to the network are raw pixel values. But, internally the network generates much higher level features. For example, there might be a node in the network that responds to the presence of diagonal lines in an image or, at an even higher level, to the presence of faces.

# DNN for Dimension Reduction

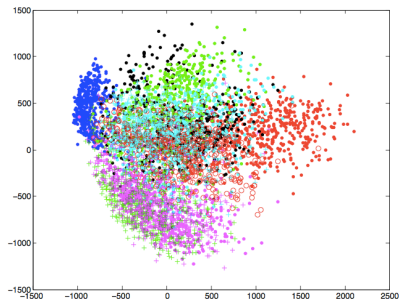
- Reduce high-dim facial images to 30-dim vectors by autoencoder:



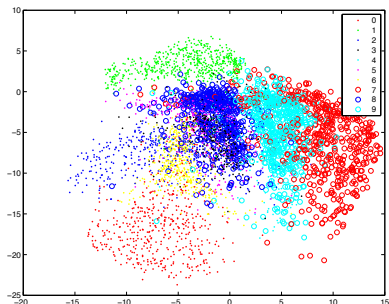


# DNN for Dimension Reduction

- Reduce 784 dimension in handwritten digit images to 2 dimension:  
784-1000-500-250-**2**-250-500-1000-784:



PCA



DNN

39

# Training of DNN

- A DNN could have millions of weights
- How to find these weights has been a problem for computer scientists and engineers for many years.
- The basic idea is to start from randomized weights.
- Then, we tell the network what went wrong and adjust the weights to reduce the number of mistakes.
- By repeating this process hundreds of times, we obtain a DNN with a proper set of weights that can solve our problem.
- This is called supervised learning, which is similar to how we learn things when a teacher tells us what is right and what is wrong.
- Instead of write computer programs to define the rules, we let the DNN to learn the rules from the data.

# Simple Gradient Descent

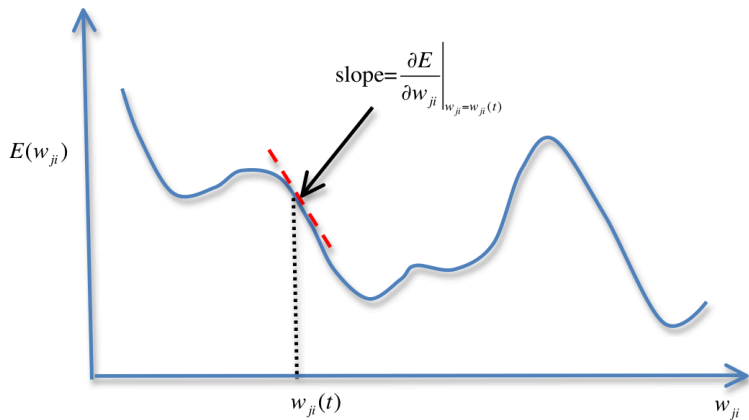
- Assume that we have a set of weights in the form of a weight vector  $\mathbf{w}$  and an error function  $E(\mathbf{w})$
- Our objective is to minimize  $E(\mathbf{w})$  with respect to  $\mathbf{w}$  based on the gradient of  $E$ :

$$\mathbf{w}^{(t+1)} = \mathbf{w}^t - \eta \nabla E(\mathbf{w}^{(t)})$$

where  $t$  is the iteration index (epoch) and  $\eta$  is a small positive learning rate.

- Techniques that use the whole data set at once are called batch methods. At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function.

# Simple Gradient Descent



Gradient Descent for 1D

# Stochastic Gradient Descent

- There is an on-line version of gradient descent that has proved useful in practice:

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w})$$

- The update formula becomes:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^t - \eta \nabla E_n(\mathbf{w}^{(t)})$$

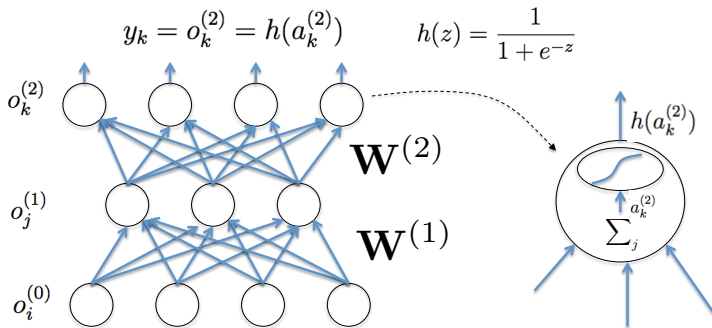
- This update is repeated by cycling through the data either in sequence or by selecting points at random with replacement.
- There are intermediate scenarios in which the updates are based on mini-batches of data points:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^t - \frac{\eta}{B} \sum_{n \in \mathcal{B}} \nabla E_n(\mathbf{w}^{(t)}),$$

where  $\mathcal{B}$  indexes to a small batch of training data and  $B$  is the batch size (typically 100).

# Backpropagation for Training Multilayer Networks

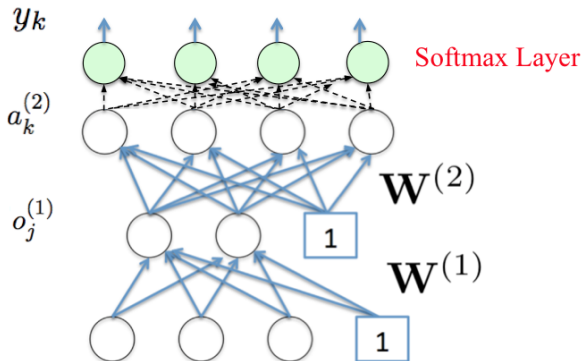
- The network weights  $\mathbf{W}^{(1)}$  and  $\mathbf{W}^{(2)}$  in multiple-layer neural network below can be trained by the Back-propagation (BP) algorithm.



# Loss Function: Categorical Cross-Entropy

- For multi-class classification, we apply the **softmax** function to the output layer's activations:

$$y_k = \frac{\exp(a_k^{(2)})}{\sum_{r=1}^K \exp(a_r^{(2)})} = P(\text{Class} = k | \mathbf{x})$$



# Loss Function: Categorical Cross-Entropy

- Then, we minimize the cross-entropy between the target output and the actual output:

$$E(\mathbf{w}) = -\frac{1}{2} \sum_{n \in \mathcal{B}} \sum_{k=1}^K t_{n,k} \log y_{n,k},$$

where  $K$  is the number of classes,  $y_{n,k}$  is the softmax output, and  $t_{n,k}$  is the target output.

- $t_{n,k}$  uses one-hot or 1-of- $K$  encoding:

$$t_{n,k} = \begin{cases} 1 & \mathbf{x}_n \in \text{Class } k \\ 0 & \text{otherwise} \end{cases}$$

- For example, if  $\mathbf{x}_n \in \text{Class } 2$  then,

$$\mathbf{t}_n = [0, 1, 0, \dots, 0]^T$$



# Loss Function: Binary Cross Entropy

- For binary classification, we may use a single output node with the binary cross-entropy as the loss function:

$$E(\mathbf{w}) = - \sum_{n \in \mathcal{B}} \{t_n \log y_n + (1 - t_n) \log(1 - y_n)\}$$

where  $n$  indexes the samples in a mini-batch and  $t_n \in \{0, 1\}$  is the target output of  $\mathbf{x}_n$ .

- To simplify notation, we drop the subscript  $n$  and ignore the batch:

$$E(\mathbf{w}) = -t \log y - (1 - t) \log(1 - y),$$

where  $y = h(a_1^{(2)})$  and

$$a_1^{(2)} = \sum_j w_{1j}^{(2)} o_j^{(1)} + b_1^{(2)}$$

# Gradient of Cross-Entropy Loss

- Instantaneous error gradient:

$$\frac{\partial E}{\partial w_{1j}^{(2)}} = \frac{\partial E}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial w_{1j}^{(2)}} = \delta_1^{(2)} o_j^{(1)}$$

where

$$\begin{aligned}\delta_1^{(2)} &= \frac{\partial E}{\partial a_1^{(2)}} = \frac{\partial E}{\partial o_1^{(2)}} \frac{\partial o_1^{(2)}}{\partial a_1^{(2)}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial a_1^{(2)}} \\ &= \left[ -\frac{t}{y} + \frac{1-t}{1-y} \right] h'(a_1^{(2)}) & h(z) &= \frac{1}{1+e^{-z}} \\ &= \frac{-t(1-y) + (1-t)y}{y(1-y)} h(a_1^{(2)}) \left[ 1 - h(a_1^{(2)}) \right] \\ &= y - t & \because y &= h(a_1^{(2)}) \\ &\Rightarrow \frac{\partial E}{\partial w_{1j}^{(2)}} = (y - t) o_j^{(1)}\end{aligned}$$

# Gradient of Cross-Entropy Loss

- Similarly, the gradient w.r.t. the first-layer weights is

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \frac{\partial E}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i$$

where

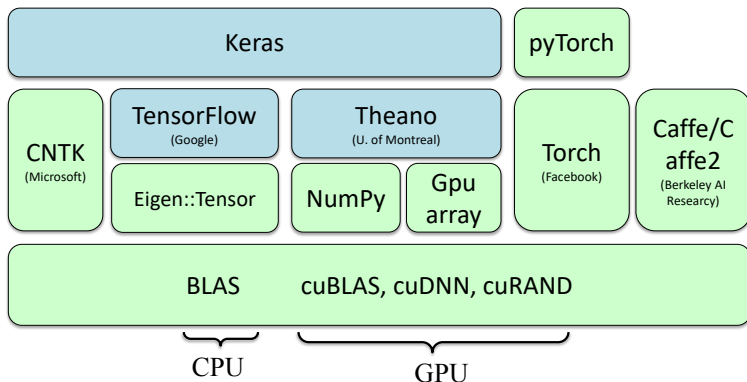
$$\begin{aligned}\delta_j^{(1)} &= \frac{\partial E}{\partial a_j^{(1)}} = \frac{\partial E}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial a_j^{(1)}} = \delta_1^{(2)} \frac{\partial a_1^{(2)}}{\partial o_j^{(1)}} \frac{\partial o_j^{(1)}}{\partial a_j^{(1)}} \\ &= (y - t) w_{1j}^{(2)} o_j^{(1)} (1 - o_j^{(1)})\end{aligned}$$

- Update Formulae:

$$\begin{aligned}w_{1j}^{(2)} &\leftarrow w_{1j}^{(2)} - \eta (y - t) o_j^{(1)} \\ w_{ji}^{(1)} &\leftarrow w_{ji}^{(1)} - \eta \left[ (y - t) w_{1j}^{(2)} o_j^{(1)} (1 - o_j^{(1)}) \right] x_i\end{aligned}$$

# Deep Learning Software Stack

- We do not have to derive the gradient. Instead, we only need to define the forward propagation.
- The following software stack will perform the gradient computation and BP for us.

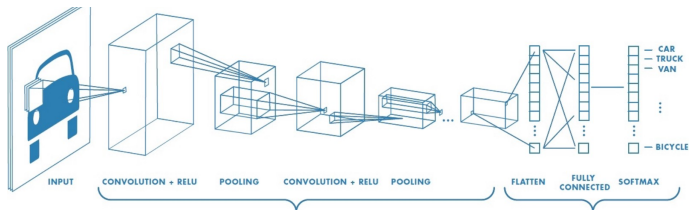


They are all free!

# Convolutional Neural Networks

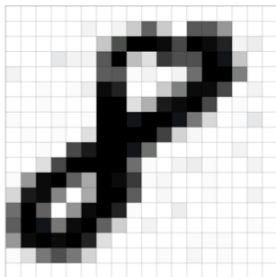
# Convolutional Neural Networks

- Convolutional neural networks (CNNs) are a type of machine learning programs that are well-suited for image classification tasks, like spotting the early signs of cervical cancer and recognition of objects from images.
- A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.



# Why CNN?

- If we use a DNN for handwritten digit recognition, we need to stack (flatten) the rows or columns of an image to form a vector.

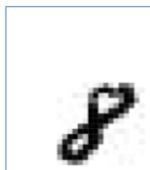
[illegible]

==

[illegible]

# Why CNN?

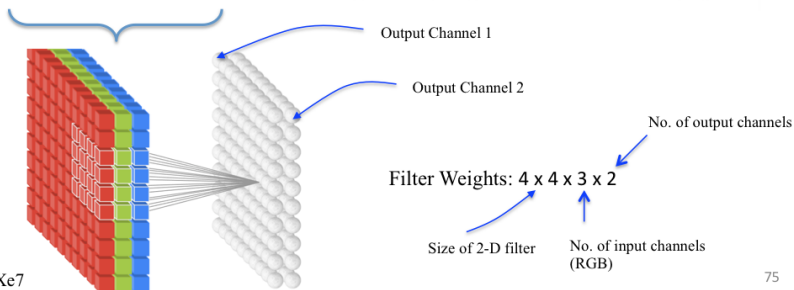
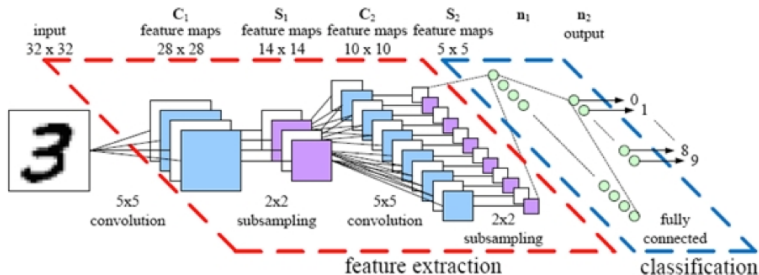
- This method works fine but we have problem when the digit is off-centered



- The flattened vector is now completely different from the training data.
- A better solution is to apply convolutional filters to patches of pixels, similar to the receptive field of human's visual system.
- Human do not look at an image at the pixel level. Instead, we look for the spatial structure of an image.
- This means that it does not matter where the "8" is as long as it is an "8".



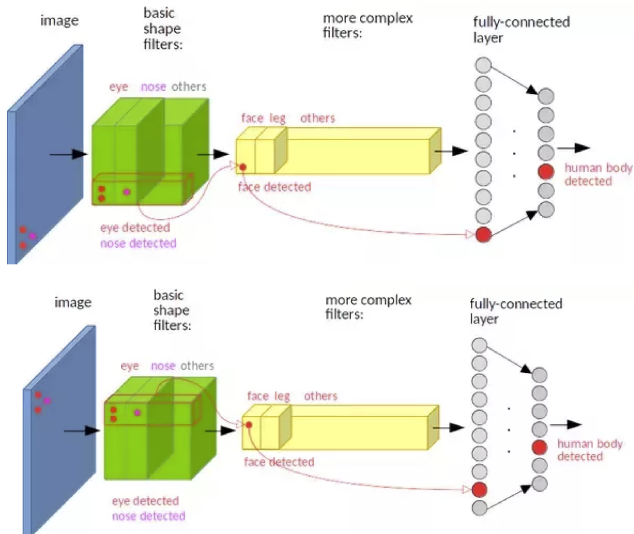
# CNN for Digit Recognition



[goo.gl/pHeXe7](http://goo.gl/pHeXe7)

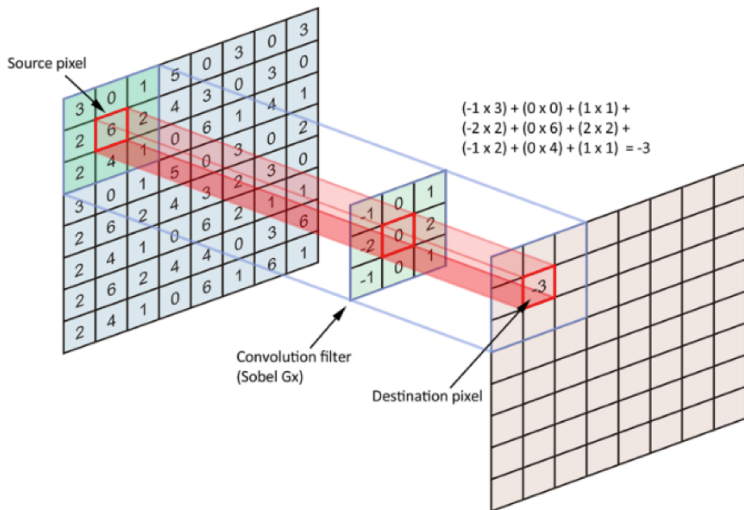
75

# How CNN Detect Patterns in Images



<https://www.quora.com/How-is-a-convolutional-neural-network-able-to-learn-invariant-features>

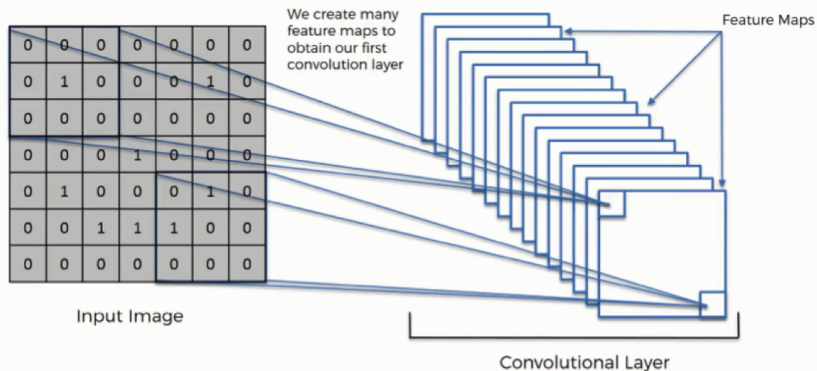
# Convolution Operation



<https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/>

# Feature Maps

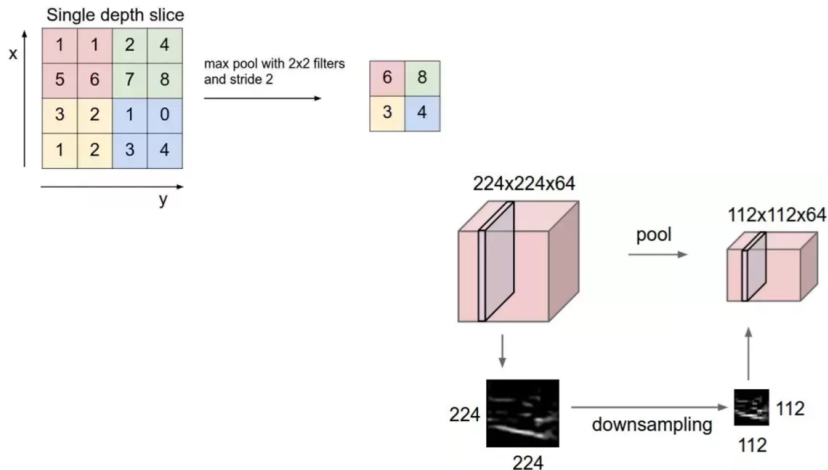
Each convolutional filter produces one feature maps in the next layer:



<https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/>

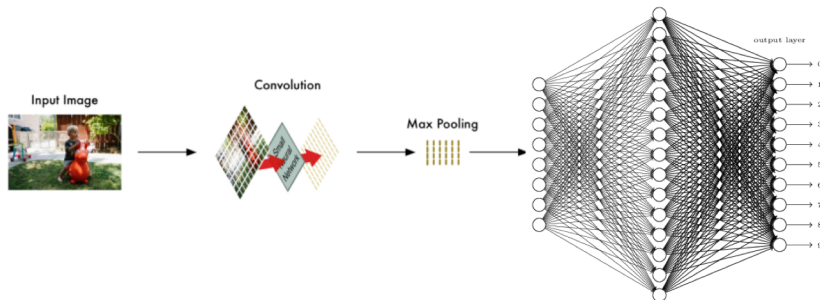
# Pooling Operation

- Apply down-sampling (also called max-pooling).
- Pooling can make the object to be recognized less dependent on its position in the image.

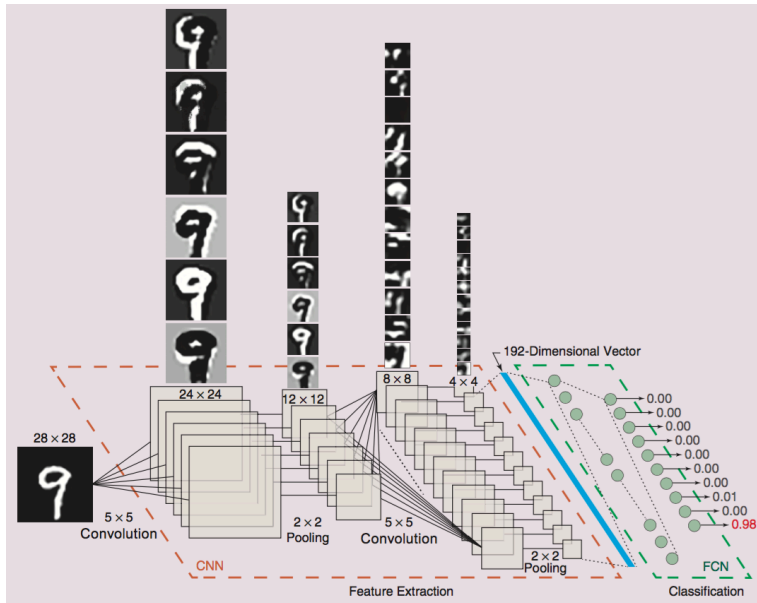


# Fully-Connected Layers

- We may repeat the convolution and max-pooling a number of times
- Feed the max-pooled output to a fully connected layer to make the final prediction.



# CNN for Handwritten Digit Recognition



# Recent Development in Deep Learning

- **Xavier Initialization** (Glorot and Bengio, 2010):
  - A simple but effective weight initialization method
  - With Xavier initialization, per-layer generative pre-training is not necessary
- **Rectified Linear Unit, ReLU** (Glorot and Bengio, 2011):
  - Rectifier units are closer to biological neurons
  - They can avoid the gradient vanishing problem. DNNs with ReLU as non-linear units do not require pre-training on supervised tasks with large labeled datasets.
- **Generative Adversarial Networks, GAN** (Goodfellow, 2014):
  - Use opposite objective functions during training.
  - The generator attempts to synthesize images that can fool the discriminator into believing that the synthetic images are real.



# Recent Development in Deep Learning

- **Batch Normalization** (Ioffe and Szegedy, 2015):

- During BP training, apply z-norm to the output of each layer, using the statistics within a mini-batch.
- Can achieve the same accuracy with 1/14 of the training steps

- **Residual Networks** (He et al. 2015):

- Aim to solve a common problem in DNNs: When depth increases, accuracy gets saturated and then degrades.
- Networks are trained to predict the residual  $H(\mathbf{x}) - \mathbf{x}$  instead of  $H(\mathbf{x})$ , where  $\mathbf{x}$  is the input and  $H(\mathbf{x})$  is the desired mapping of the application.
- Can train networks up to 1000 layers.

- **Variational Autoencoders** (Kingma and Welling, 2014):

- Combine DNNs with probabilistic models (bottleneck-layer nodes are stochastic)
- Work very well in generating complicated data.

# Recent Development in Deep Learning

- **Generative Adversarial Networks** (Goodfellow, 2014):
  - Based on two-player zero sum games
  - A generator aims to generate fake objects that look so real that they can fool a discriminator.
  - The discriminator aims to differentiate whether an input object is real or fake.
  - Many new types of GAN: CycleGAN, Domain adaptation network, VAEGAN, etc.
- **DenseNets** (Huang et al., 2016):
  - Connects each layer to every other layer in a feed-forward fashion.
  - Alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and reduce the number of parameters
- **Transformer Network** (Vaswani et al., 2017):
  - Based on the attention mechanism
  - Better than RNN and LSTM for NLP