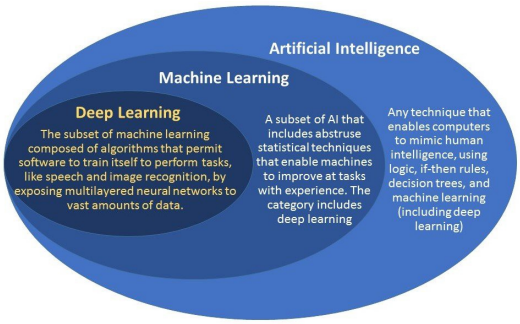
MA333 Introduction to Big Data Science
Deep Learning

Zhen Zhang

Southern University of Science and Technology

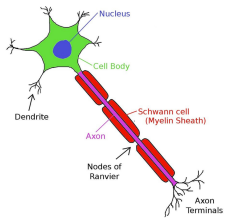
Outlines

What is Deep Learning?



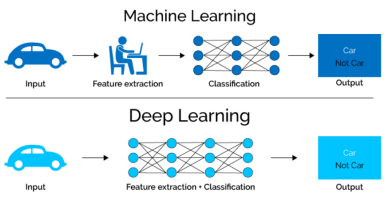
Deep Learning

- Deep learning is a sub field of Machine Learning that very closely tries to mimic human brain's working using neurons.
- These techniques focus on building Artificial Neural Networks (ANN) using several hidden layers.
- There are a variety of deep learning networks such as Multilayer Perceptron (MLP), Autoencoders (AE), Convolution Neural Network (CNN), Recurrent Neural Network (RNN).



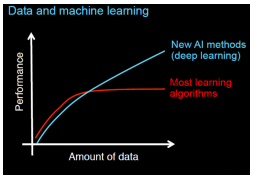
ML vs DL

- In Machine Learning, the features need to be identified by an domain expert.
- In Deep Learning, the features are learned by the neural network.

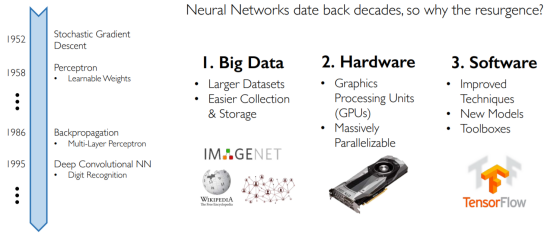


Why Deep Learning is Growing?

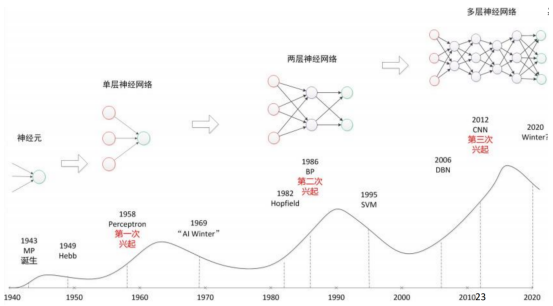
- Processing power needed for Deep learning is readily becoming available using GPUs, Distributed Computing and powerful CPUs.
- Moreover, as the data amount grows, Deep Learning models seem to outperform Machine Learning models.
- Explosion of features and datasets.
- Focus on customization and real time decisioning.



Why Now ?



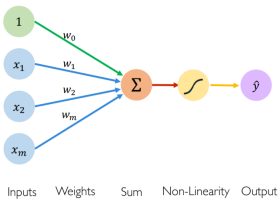
History



Big Guys

Outlines

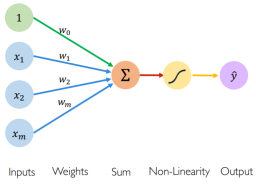
The Perceptron : Forward Propagation



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right),$$

- $\hat{y}$ is the Output,
- g is a Non-linear activation function,
- w_0 is the Bias.

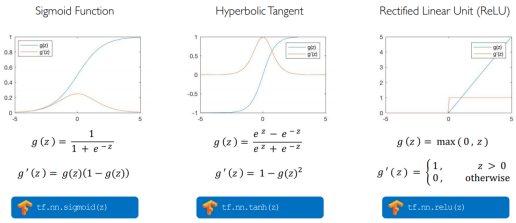
The Perceptron : Forward Propagation



$$\hat{y} = g\left(w_0 + \mathbf{X}^T\mathbf{W}\right), \textit{where}$$

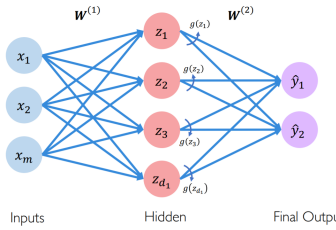
$$\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{ and } \mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}.$$

Common Activation Functions



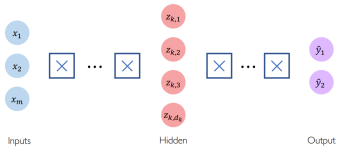
Note all activation functions are nonlinear.

Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)}, \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)}\right).$$

Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g\left(z_{k-1,j}\right) w_{j,i}^{(k)}.$$

Theorem (Universal approximation theorem (Cybenko 1980, 1989))

1. *Any function can be approximated by a three-layer neural network within sufficiently high accuracy.*
2. *Any bounded continuous function can be approximated by a two-layer neural network within sufficiently high accuracy.*

Loss Optimization

We want to find the network weights that achieve the lowest loss

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \mathbf{W}\right), y^{(i)}\right),$$

where $\mathcal{L}\left(f\left(x^{(i)}; \mathbf{W}\right), y^{(i)}\right)$ is the loss function we defined according to the specific problem to measure the differences between output state $f\left(x^{(i)}; \mathbf{W}\right)$ and reference state $y^{(i)}$. It also can be written as

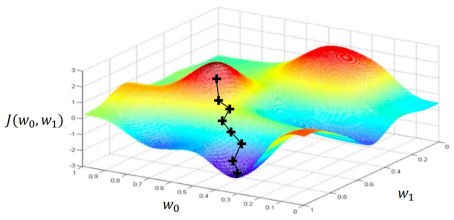
$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} C(\mathbf{W}).$$

Remember

$$\mathbf{W} = \left\{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \cdots\right\}.$$

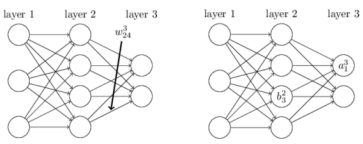
Gradient Decent

We can use Gradient Decent algorithm to find the optimal parameter $\mathbf{W}$.



Note that we should calculate $\frac{\partial C}{\partial \mathbf{W}}$ to update $\mathbf{W}$.

Notations



- w_{jk}^l is the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.
- for brevity $b_j^l = w_{j0}^l$ is the bias of the j^{th} neuron in the l^{th} layer.
- a_j^l for the activation of the j^{th} neuron in the l^{th} layer z_j^l.

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

Four fundamental equations

We first define the error δ_j^l of neuron j in layer by

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l},$$

and we give the four fundamental equations of back propagation :

$$\delta^L = \nabla_a C \odot \sigma'\left(z^L\right) \qquad (BP1)$$

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right) \qquad (BP2)$$

$$\frac{\partial C}{\partial b_j^L} = \delta_j^l \qquad (BP3)$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \qquad (BP4)$$

An equation for the error in the output layer (BP1)

The components of δ^L are given by

$$\delta^L = \nabla_a C \odot \sigma'\left(z^L\right) \qquad (BP1)$$

Démonstration.

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\ &= \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial \sigma\left(z_j^L\right)}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right)\end{aligned}$$

□

An equation for the error in the hidden layer (BP2)

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right) \qquad (BP2)$$

Démonstration.

$$\begin{cases} \delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l} \\ z_k^{l+1} = \left(\sum_i w_{ki}^{l+1}a_i^l\right) + b_k^{l+1} = \left(\sum_i w_{ki}^{l+1}\sigma\left(z_i^l\right)\right) + b_k^{l+1} \end{cases}$$

$$\Rightarrow \begin{cases} \delta_j^l = \sum_k \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l} \\ \frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1}\sigma'\left(z_j^l\right) \end{cases} \Rightarrow \delta_j^l = \sum_k \delta_k^{l+1}w_{kj}^{l+1}\sigma'\left(z_j^l\right)$$

□

The change of the cost with respect to any bias (BP3)

$$\frac{\partial C}{\partial b_j^L} = \delta_j^l \qquad (BP3)$$

Démonstration.

$$\begin{cases} \frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l}\frac{\partial z_k^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} \\ z_j^l = \left(\sum_k w_{jk}^l d_k^{l-1}\right) + b_j^l \Rightarrow \frac{\partial z_k^l}{\partial b_j^l} = 1 \end{cases} \Rightarrow \frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\cdot 1 = \delta_j^l.$$

□

The change of the cost with respect to any weight (BP4)

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \qquad (BP4)$$

Démonstration.

$$\begin{cases} \frac{\partial C}{\partial w_{jk}^l} = \sum_i \frac{\partial C}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} \\ z_j^l = \left(\sum_m w_{jm}^l a_m^{l-1}\right) + b_j^l \Rightarrow \frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \end{cases}$$

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

.

□

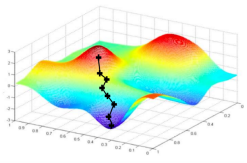
Back Propagation procedure

- 1. Input x : Set the corresponding activation a^1 for the input layer.
- 2. Feedforward : For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma\left(z^l\right)$.
- 3. Output error δ^L : Compute the vector $\delta^L = \nabla_a C \odot \sigma'\left(z^L\right)$.
- 4. Backpropagate the error : For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right)$.
- 5. Output : The gradient of the cost function is given by $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

Gradient Descent

Algorithm

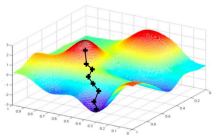
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights



Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



- Mini-batches lead to fast training !
- Can parallelize computation + achieve significant speed increases on GPUs.

Outlines

A sequence modeling problem : predict the next word

- **France** is where I grew up, but I now live in Boston. I speak fluent ???.
- The food was good, not bad at all.
  vs. The food was bad, not good at all.

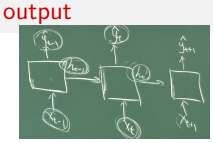
Sequence modeling : design criteria

To model sequences, we need to :

- 1. Handle variable-length sequences
- 2. Track long-term dependencies
- 3. Maintain information about order
- 4. Share parameters across the sequence

Today : Recurrent Neural Networks (RNNs) as an approach to sequence modeling problems

RNN state update and output

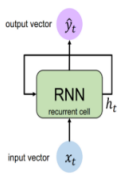


Apply a recurrence relation at every time step to process a sequence :

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

(new state; function parameterized by W; old state; input vector at time step t)

Note : the same function and set of parameters are used at every time step



Output Vector

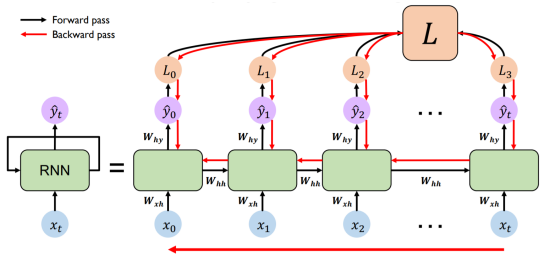
$$\hat{y}_t = \boldsymbol{W}_{hy} h_t$$

Update Hidden State

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

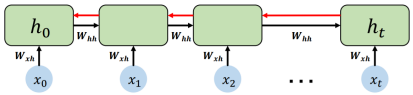
Input Vector

RNNs : backpropagation through time



Note that it reuse the same weight matrices at every time step

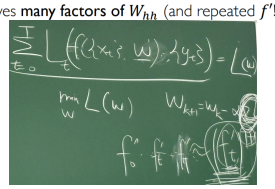
Standard RNN gradient flow : exploding gradients



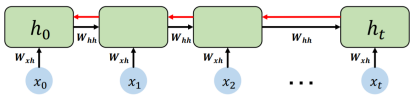
Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients



Standard RNN gradient flow : vanishing gradients



Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

Largest singular value > 1:
exploding gradients

Gradient clipping to scale big gradients

Largest singular value < 1:
vanishing gradients

1. Activation function
2. Weight initialization
3. Network architecture

The problem of long-term dependencies

Why are vanishing gradients a problem?

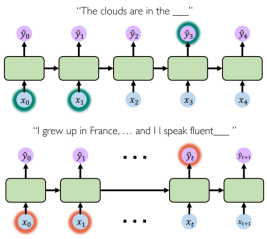
Multiply many **small numbers** together

↓

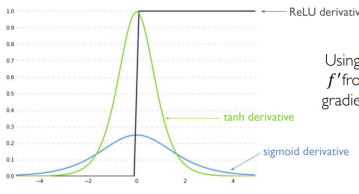
Errors due to further back time steps have smaller and smaller gradients

↓

Bias parameters to capture short-term dependencies



Trick 1 : activation functions



Using ReLU prevents f' from shrinking the gradients when $x > 0$

Trick 2 : parameter initialization

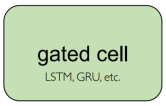
- Initialize weights to identity matrix .
- Initialize biases to zero.

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

Trick 3 : gated cells

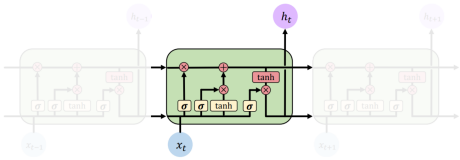
Idea : use a more complex recurrent unit with gates to control what information is passed through



Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

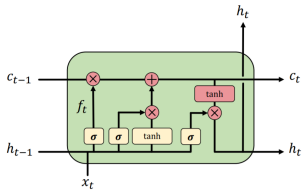
Long Short Term Memory (LSTMs)

LSTM repeating modules contain interacting layers that control information flow.



LSTM cells are able to track information throughout many time steps.

Long Short Term Memory (LSTMs)

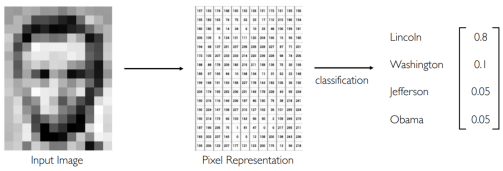


- What information to throw away ?
- What to store ?
- What to output ?

Recurrent neural networks (RNNs)

- 1. RNNs are well suited for sequence modeling tasks.
- 2. Model sequences via a recurrence relation.
- 3. Training RNNs with backpropagation through time.
- 4. Gated cells like LSTMs let us model long-term dependencies.
- 5. Models for music generation, classification, machine translation.

Tasks in Computer Vision

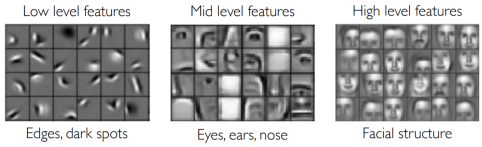


- Regression : output variable takes continuous value.
- Classification : output variable takes class label. Can produce probability of belonging to a particular class.

Problems in Manual Feature Extraction

Learning Feature Representations

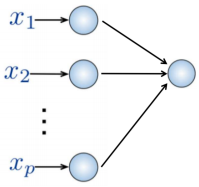
Can we learn a hierarchy of features directly from the data instead of hand engineering ?



Fully Connected Neural Network

Input :

- 2D image.
- Vector of pixel values.

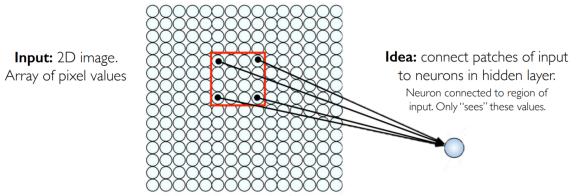


Fully Connected :

- Connect neuron in hidden layer to all neurons in input layer.
- No spatial information !
- And many, many parameters !

How can we use spatial structure in the input to inform the architecture of the network ?

Using Spatial Structure

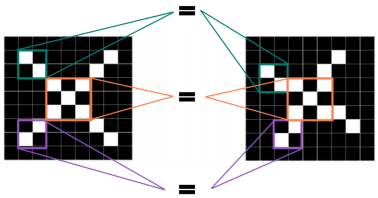


- Connect patch in input layer to a single neuron in subsequent layer.
- Use a sliding window to define connections.
- How can we weight the patch to detect particular features ?

Applying Filters to Extract Features

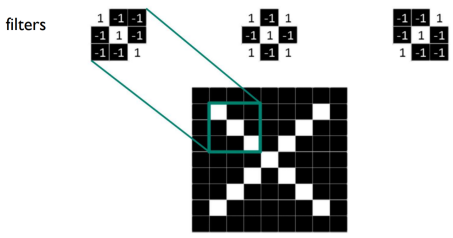
- 1 Apply a set of weights (a filter) to extract local features.
- 2 Use multiple filters to extract different features.
- 3 Spatially share parameters of each filter. (features that matter in one part of the input should matter elsewhere.)

Features of X



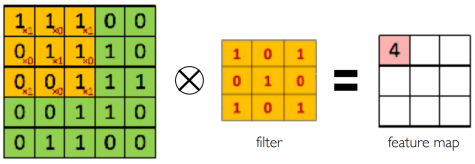
- Image is represented as matrix of pixel values and computers are literal !
- We want to be able to classify an X as an X even if it is shifted, shrunk, rotated, deformed.

Filters to Detect X Features



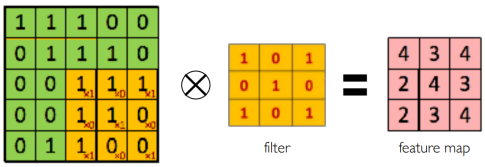
The Convolution Operation

- Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter.
- We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs :



The Convolution Operation

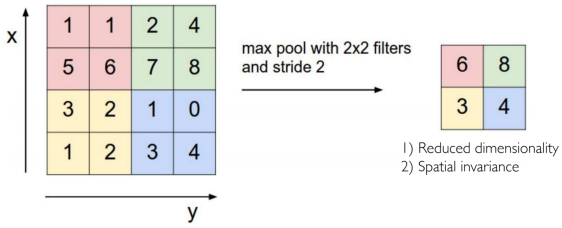
- Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter.
- We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs :



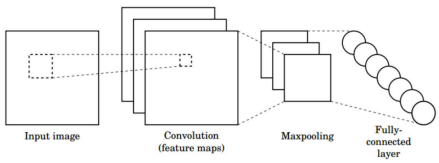
Producing Feature Maps

Different filters have different effects !

Pooling



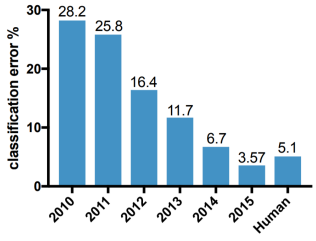
CNNs for Classification



- 1. Convolution : Apply filters with learned weights to generate feature maps.
- 2. Non-linearity : Often ReLU.
- 3. Pooling : Downsampling operation on each feature map.

Train model with image data.
Learn weights of filters in convolutional layers.

ImageNet Challenge : Classification Task



2012: AlexNet. First CNN to win.
- 8 layers, 61 million parameters

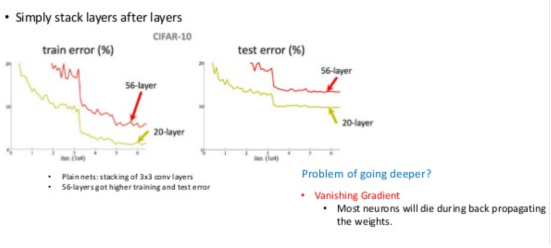
2013: ZFNet
- 8 layers, more filters

2014: VGG
- 19 layers

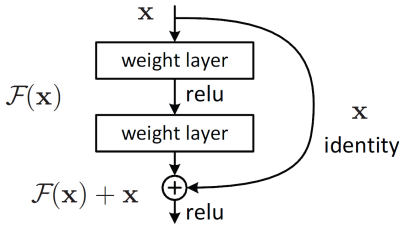
2014: GoogLeNet
- "Inception" modules
- 22 layers, 5million parameters

2015: ResNet
- 152 layers

The problem when go deeper



Better network structure for learning : ResNet

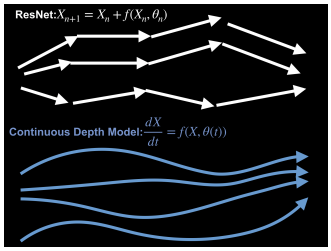


- Introduce shortcut connections (exists in prior literature in various forms).
- Key invention is to skip 2 layers. Skipping single layer do not give much improvement for some reason.

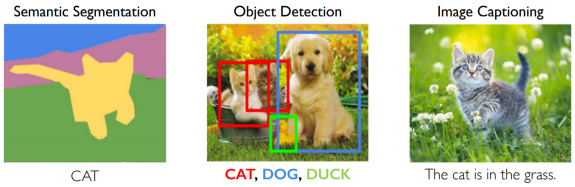
前面可能丢失了信息
如果不加上x，这些信息就永久丢失

ResNet vs ODENet

- **ResNet :** $\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t)$,
- **ODENet :** $\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta)$.



Beyond Classification



Supervised vs unsupervised learning

Supervised Learning

- Data : (x, y)
  x is data, y is label.
- Goal : Learn function to map x ⇒ y.
- Examples : Classification, regression, object detection, semantic segmentation, etc.

Unsupervised Learning

- Data : x
  x is data, no labels!
- Goal : Learn some hidden or underlying structure of the data.
- Examples : Clustering, feature or dimensionality reduction, etc.

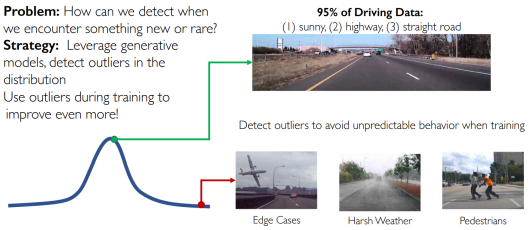
Generative modeling

Goal : Take as input training samples from some distribution and learn a model that represents that distribution

How can we learn $p_{model}(x)$ similar to $p_{data}(x)$?

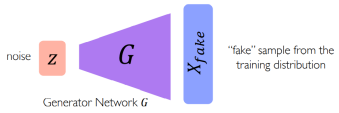
Why generative models ? Outlier detection

- **Problem:** How can we detect when we encounter something new or rare?
- **Strategy:** Leverage generative models, detect outliers in the distribution
- Use outliers during training to improve even more!



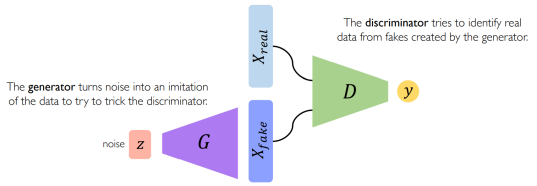
What if we just want to sample

- Idea : do not explicitly model density, and instead just sample to generate new instances.
- Problem : want to sample from complex distribution can not do this directly !
- Solution : sample from something simple (noise), learn a transformation to the training distribution.



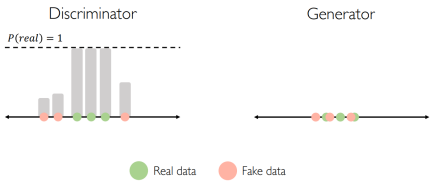
Generative Adversarial Networks (GANs)

- Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.



Training GANs

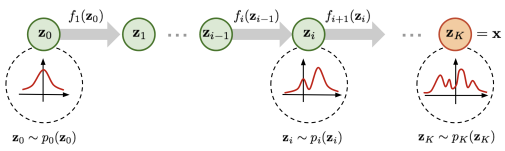
- Neural Network Discriminator tries to identify real data from fakes created by the generator.
- Neural Network Generator tries to create imitations of data to trick the discriminator.



- $$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right) \right]$$

What if we want the model density

Flow based model



- f is bijective and differentiable functions.
- We have the explicitly model density $p_{z_k}(z_k)$.

CVF

Theorem

(Change of Variable Theorem) *Given any $z_0 \sim p_{z_0}(z_0)$, if the transformation $f : \mathbb{R}^d \to \mathbb{R}^d$ is differentiable and bijective, then the distribution of $x = f(z_0)$ is $p_x(x) = \frac{p_{z_0}(z_0)}{\left|\det \frac{\partial f}{\partial z_0}\right|}$.*

- The density probability follows

$$\log p_{z_k}(z_k) = \log p_{z_0}(z_0) - \sum_{i=1}^{k} \left| \det \frac{\partial f_i}{\partial z_{i-1}} \right|.$$

- we want to find parameter λ of neural network f_i satisfies :

$$\arg\min_{\lambda} D_{KL}(p_{data} || p^{\lambda}_{z_k}) \approx \arg\max_{\lambda} \sum_{i=1}^{m} \log p^{\lambda}_{z_k}(x^{(i)}),$$

FUTURE : LONG WAY TO GO

- Theory : why deep learning works ? How ? Interpretability.
- Capacity : how deep is enough ?/Network structure selection.
- Manipulate network.
- Geometry of loss function.
- Deep learning with less data ?
- Interdisciplinary fields : statistics, physics, geometry, game theory...
- More exciting applications ! Healthcare, industrial process, finance, AI game play, animation...