2019년 겨울 한동대학교 전산전자공학부 머신러닝 캠프 – 김인중 교수

Machine Learning Camp

School of Computer Science and Electrical Engineering
Handong Global University

Prof. Injung Kim

2019.01.07 ~ 2019.01.09

주관 | 한동대학교 전산전자공학부

주최 | 스마트카-머신러닝 센터

후원 | SW중심대학지원사업

일정표

<table>
<tr><th></th><th></th><th>1/7 (Mon)</th><th>1/8 (Tue)</th><th>1/9 (Wed)</th></tr>
<tr><td>Morning</td><td>9:00~11:30</td><td rowspan="2"></td><td>Convolutional Neural Networks 1</td><td>Recurrent Neural Networks</td></tr>
<tr><td>Lunch</td><td>11:30~13:00</td><td>Lunch</td><td>Lunch</td></tr>
<tr><td rowspan="5">Afternoon</td><td rowspan="2">13:00~15:00</td><td>Registration (15 min)</td><td>Convolutional Neural Networks 2</td><td rowspan="2">Deep Generative Models</td></tr>
<tr><td>Introduction to Machine Learning</td><td>Emerging Topics 1</td></tr>
<tr><td>15:00~16:00</td><td rowspan="2">Neural Network Basics</td><td rowspan="3">Practice (pytorch)</td><td>Emerging Topics 2</td></tr>
<tr><td>16:00~17:00</td><td rowspan="2">Special Lecture</td></tr>
<tr><td>17:00~18:00</td><td>Deep Learning Overview</td></tr>
</table>

Emerging topics 1:

One-shot learning, Meta learning, Continual learning

Emerging topics 2

Network architecture search, Lightweight models, Non-local Nets

목차

목차

DAY 1

Introduction to Machine Learning

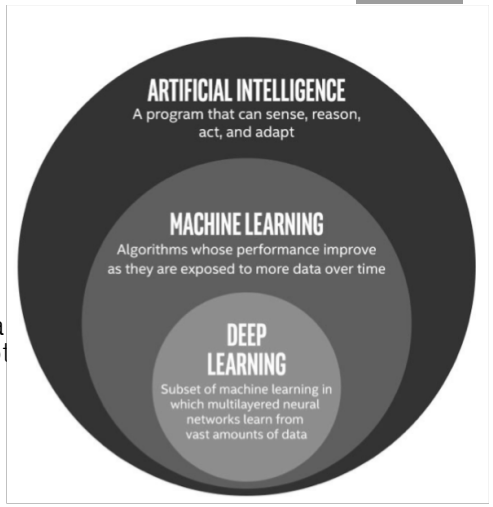
Injung Kim
Handong Global University
2019. 1. 7.

Agenda

- Introduction
- Recognition System
- Bayes' Decision Theory
- Q&A

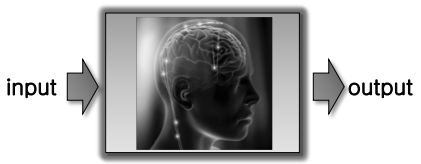
AI, Machine Learning, Deep Learning

- **Artificial intelligence**: the **intelligence** exhibited by machines or software
- **Machine learning**: a field of study that gives computers the ability to **learn from data** without being explicitly programmed.
- **Deep learning**: a branch of machine learning based on a set of algorithms that attempt to model **high-level abstractions in data**, mostly, based on **deep neural networks**.



Artificial Intelligence

- Artificial intelligence (AI) is **the intelligence** exhibited by machines or software
- **Automation of tasks** that require intelligence
  - Recognition, prediction, learning, natural language processing, planning, reasoning, etc.



Strong AI, AGI

Week AI, Narrow AI

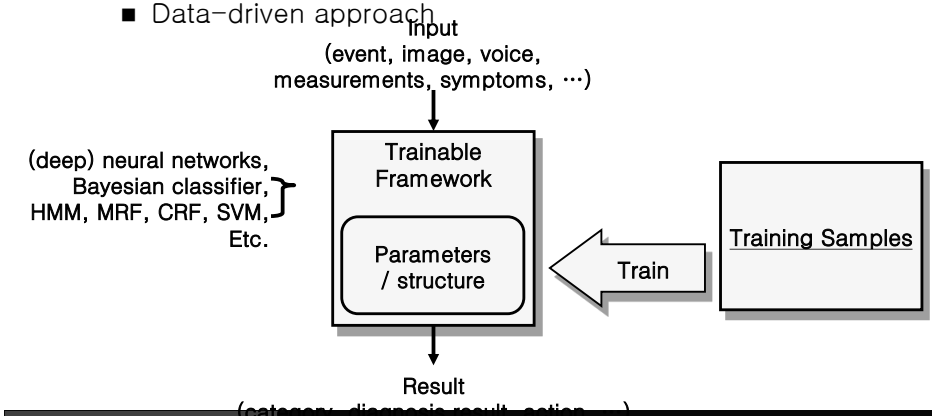
AI vs. Conventional SW

- Conventional SW
  - Perform tasks by following **predefined algorithm**
- Artificial intelligence targets
  - **Complex tasks** hard to solve by a fixed procedure
  - Tasks under **changing environment**
  - Decisions under **uncertainty** or **ambiguity**

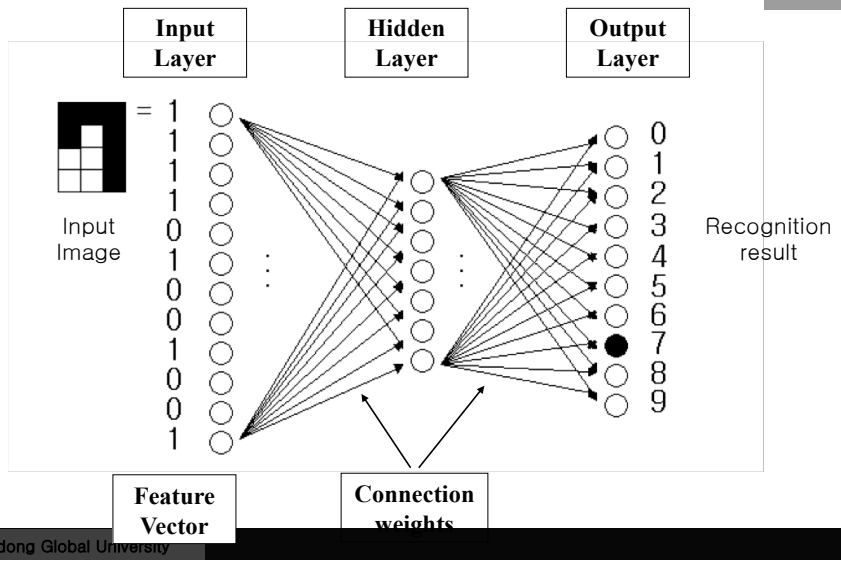
vs.

Machine Learning

- Field of study that gives computers the ability to learn without being explicitly programmed.
  - Data-driven approach

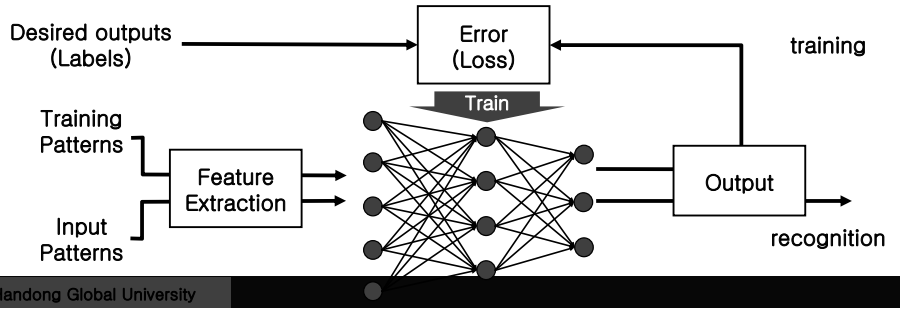


Ex) Digit Recognition using Neural Network



Ex) Building Neural Network Recognizer

1. Design network structure
2. Collect or acquire training samples (with labels)
3. Train connection weights
   - Given **training samples** and **desired outputs**, find **weights** that minimizes error.
4. Apply the trained neural network to target data

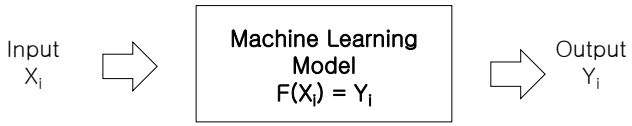


Categories of Machine Learning

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

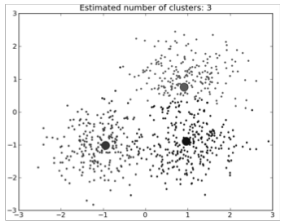
Supervised Learning

- **Supervised learning** is the machine learning task of inferring a function from labeled training data.
  - Learn to produce desired outputs for each training samples
  - Given a set of labeled training samples { (X_1, Y_1), (X_2, Y_2), (X_3, Y_3) ⋯ }, learn $F(X_i) = Y_i$

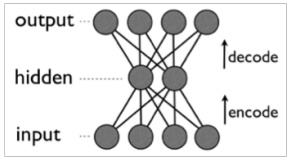


Unsupervised Learning

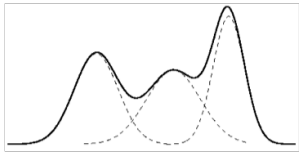
- **Unsupervised learning** is the machine learning task of inferring a function to describe hidden structure from unlabeled data.
  - Clustering, reproduction, latent variable models (hidden factor analysis)



clustering



auto-encoder

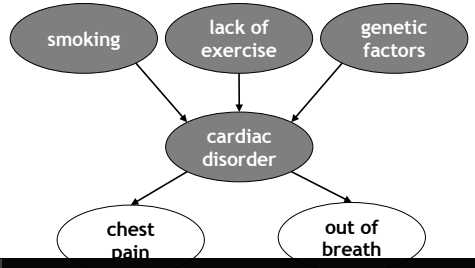


latent variable models

Latent Variables

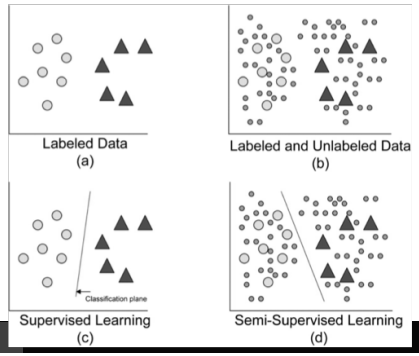
- **Latent variables** (hidden variables) are variables that are <u>not directly observed</u> but are rather inferred (through a mathematical model) from other variables that are observed.

  Ex) intention (latent variable) vs. behavior (visible variable)
  state (latent variable) vs. observation (visible variable)

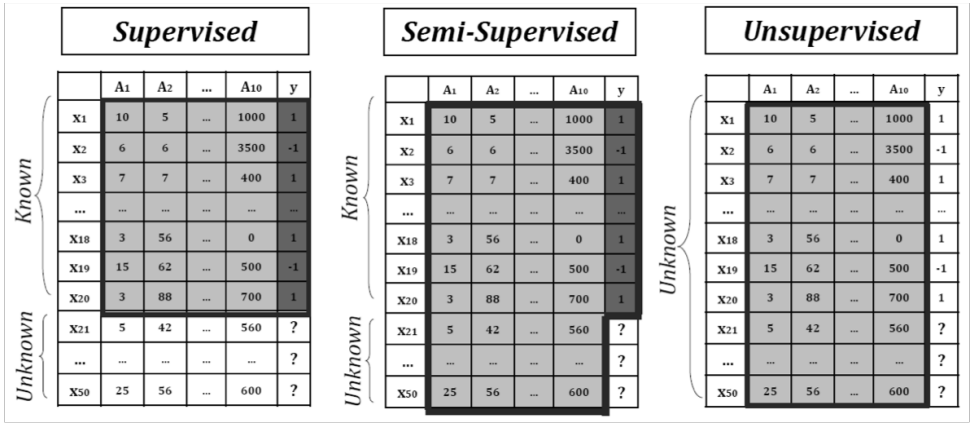


Semi-supervised Learning

- **Semi-supervised learning** is a class of supervised learning tasks and techniques that also make use of unlabeled data for training.
  - Small amount of labeled data + **large amount of unlabeled data.**



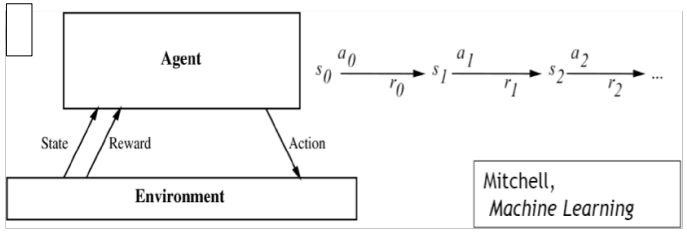
Semi-supervised Learning



	A1	A2	...	A10	y
X1	10	5	...	1000	1
X2	6	6	...	3500	-1
X3	7	7	...	400	1
...	...	...	...	...	...
X18	3	56	...	0	1
X19	15	62	...	500	-1
X20	3	88	...	700	1
X21	5	42	...	560	?
...	...	...	...	...	?
X50	25	56	...	600	?

Reinforcement Learning

- **Reinforcement learning** is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.
  - Learn **actions** to maximize **reward** signal
  - Feedback is scarce and comes late

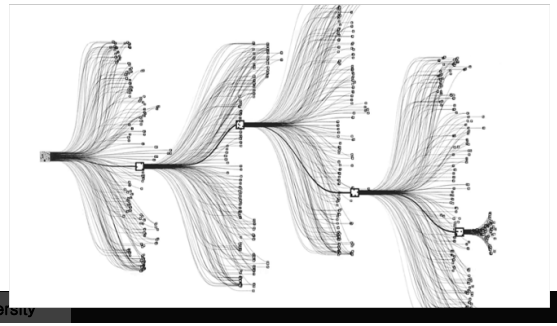


Reinforcement Learning

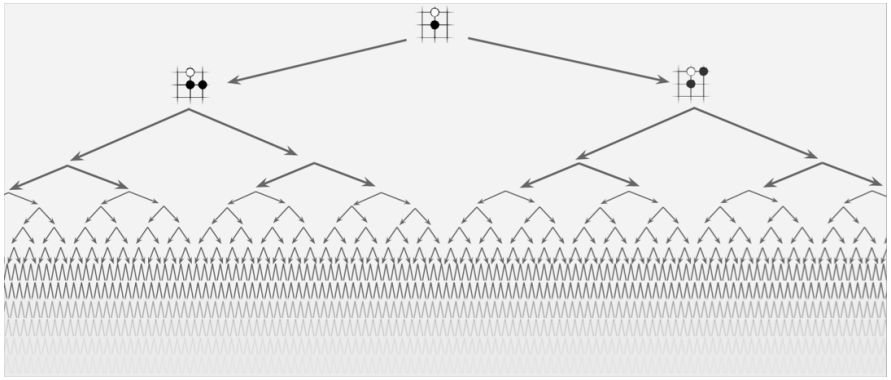
- **Policy**: recommended action for each state
  - $P(a|s)$
- **Value of state**: expected return starting from a state
  - $V(s)$
  - Depends on the agent's policy
- **Value of action (Q-function)**: expected return starting from that state, taking that action, and thereafter following policy function
  - $Q(s,a)$
  - "Quality of action"

AlphaGo

- Game play by tree search
  - Infeasible for huge search space
- Monte Carlo Tree Search (MCTS)
  - Random sampling
  - Policy network, value networks guide random sampling

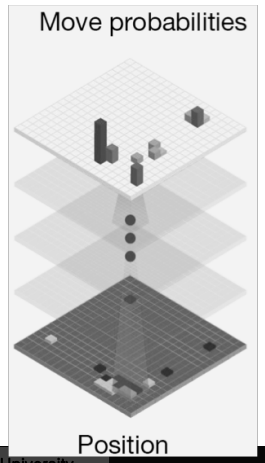


Exhaustive Search

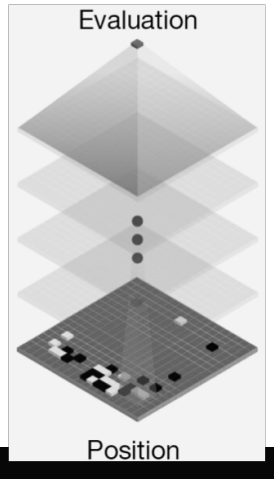


CNNs in AlphaGo

- Policy network

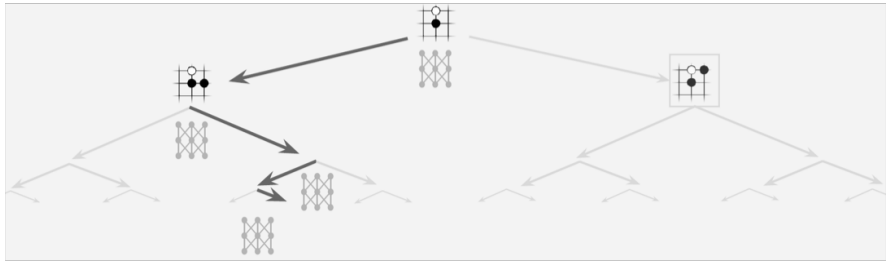


- Value network



Reducing Search Space

- Policy network reduces breath
- Value network reduces depth



Agenda

- Introduction
- Recognition System
- Bayes' Decision Theory
- Q&A

Classification

- The act of taking in raw data and taking an action based on the **category** of the data.
  - Input: a data (event, object, observation, …) that belongs to one of predefined categories
  - Output: category of the input event

Regression

- Regression: estimating the relationships among variables.
  - Techniques for modeling and analyzing several variables focusing on the relationship between a dependent variable and independent variables (or 'predictors').

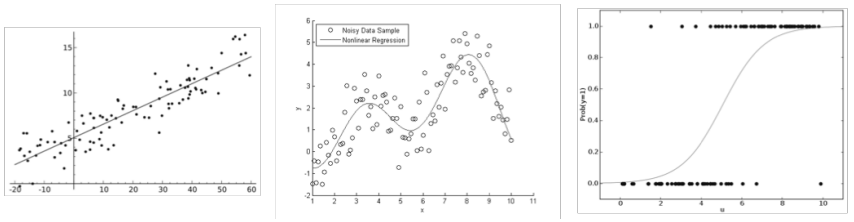
Ex) Predicting a **variable** from other variables

(bankrupt prediction, sales prediction, object coordinate estimation, …)

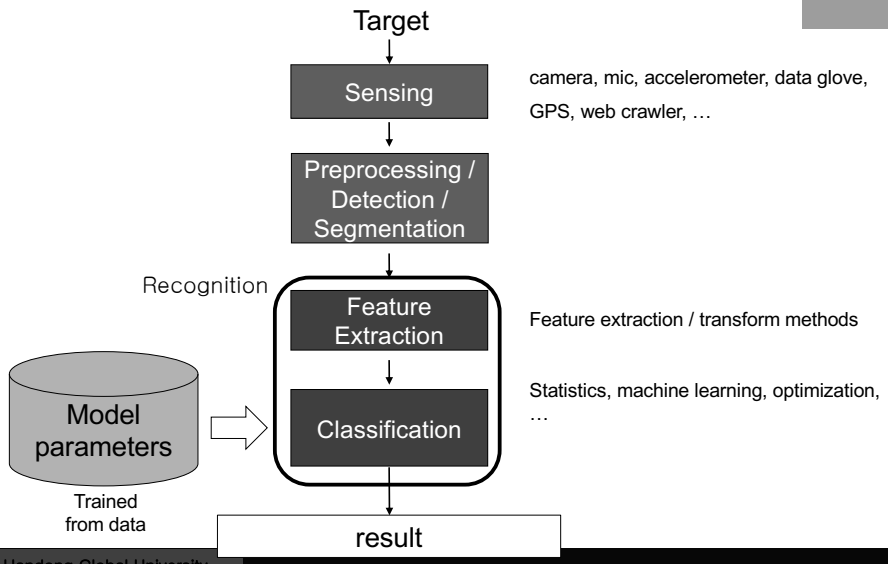
Regression

- Regression model: $y \approx F(X; \Theta)$
  - The form of function F is specified.
    - Θ: set of model parameters

linear regression | nonlinear regression | logistic regression



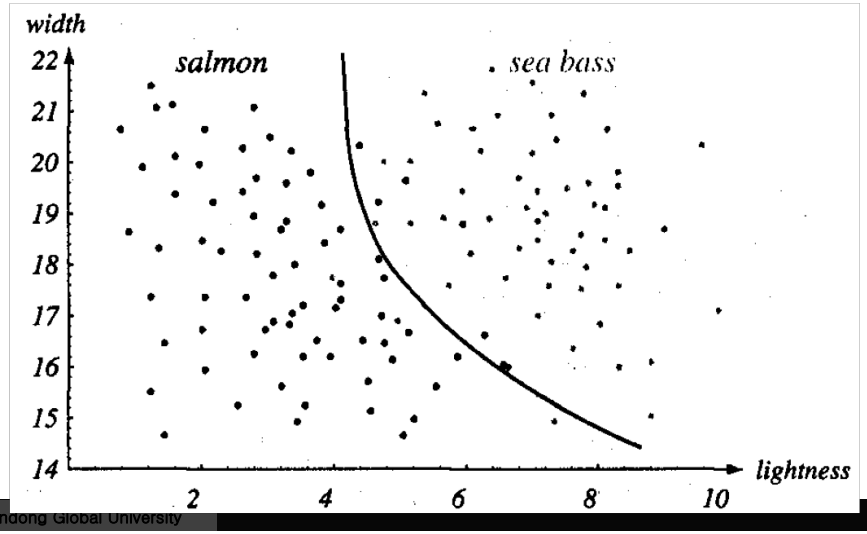
Recognition System



Why Feature Vector?

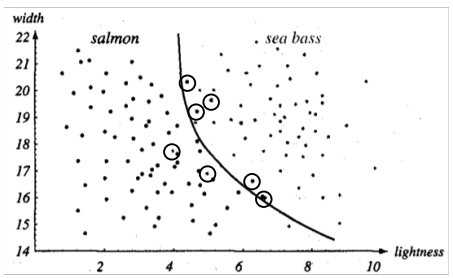
- Easy to handle
  - Most machine learning architectures receive input in vectors
- Focus on relevant information
  - Ignore irrelevant information
- Efficiency

Pattern Boundary

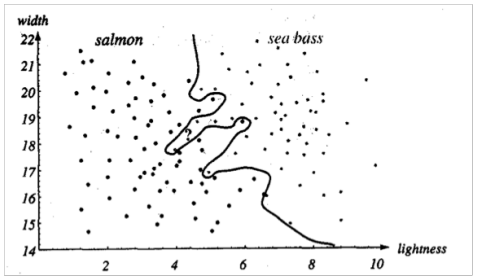


Complexity and Generalization Ability

- Complex classifier requires more training data
  - Otherwise, suffer from overfitting
- Simple classifier has advantage on unknown patterns



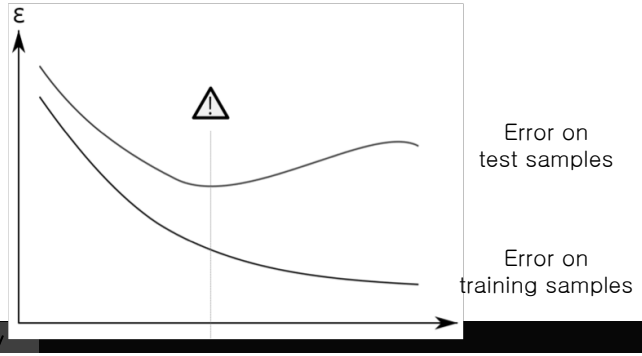
Simple model



Too complex model

Overfitting

- Overfitting occurs when a model is excessively complex relative to the number of observations.
  - Training error decreases
  - Test error increases



Agenda

- Introduction
- Recognition System
- Bayes' Decision Theory
- Q&A

Bayesian Theorem

- How to find the class that is likely to have generated the input pattern?
  - Select ω_i that maximizes $P(\omega_i|X)$
    - ω_i : class or hidden var, x: observation
- However, it's difficult to know $P(\omega_i|X)$ for each class

➔ Bayes' theorem

posterior prob. → $P(\omega \mid X) = \frac{P(\omega \cap X)}{P(X)}$

likelihood → $= \frac{P(X \mid \omega)P(\omega)}{P(X)}$ ← prior prob.

Bayesian Classifier

- A classifiers that selects ω^* such that …

$$\omega^* = \arg\max_\omega P(\omega \mid X)$$ — *Posteriori prob.*

$$= \arg\max_\omega \frac{P(X \mid \omega)P(\omega)}{P(X)}$$ ➔ P(X) is independent from ω

$$= \arg\max_\omega P(X \mid \omega)P(\omega)$$

likelihood — *prior prob.*

Bayesian Classifier

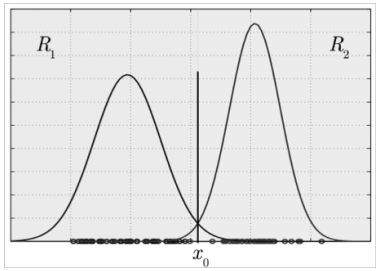
- Prior probability P(ω)
  - Measure the frequencies of classes
    - Estimate the distributions of classes by models
  - Assume P(ω)'s are same for all ω's. ➔ ignore
  - Heuristically designed constraints
    Ex) Smoothness assumption
- Likelihood P(X|ω)
  - Estimated by a generative model

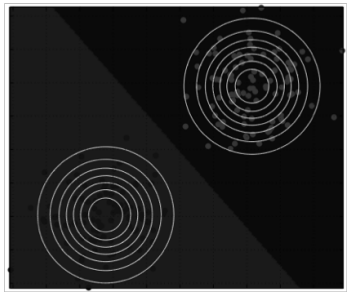
Cf. Discriminative models estimate P(ω|X) directly

Ex) Gaussian Classifier

- Assume each class has Gaussian distribution

$$P(X \mid \omega) = \frac{1}{(2\pi)^{d/2}\left|\Sigma_\omega\right|^{1/2}} \exp\left[-\frac{(X-\mu_\omega)^T \Sigma_\omega^{-1}(X-\mu_\omega)}{2}\right]$$





2. Neural Networks Basics

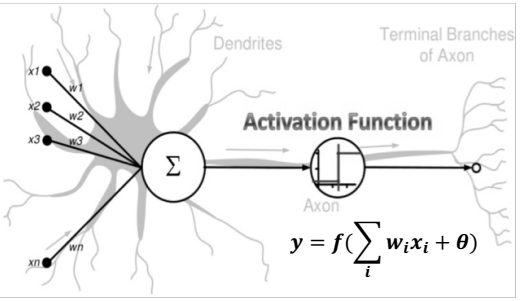
Injung Kim
Handong Global University
2019. 1. 7.

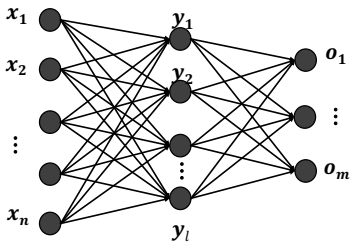
Agenda

- Single-layer Perceptron
- Multi-layer Perceptron
- Activation Functions
- Back-propagation

Neural Networks

- An artificial neural network is a mathematical model inspired by biological neural networks.
  - Intelligence comes from their connection weights
  - Connection weights are decided by learning or adaptation



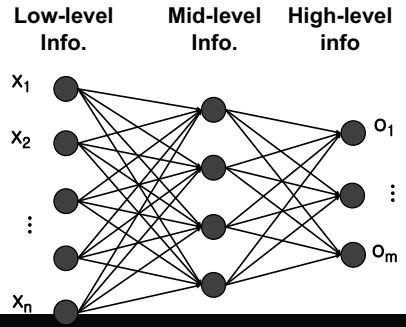


Neural Networks

- Each layer combines the input information to produce higher-level information
  - Weight of each input information is learned by training algorithms

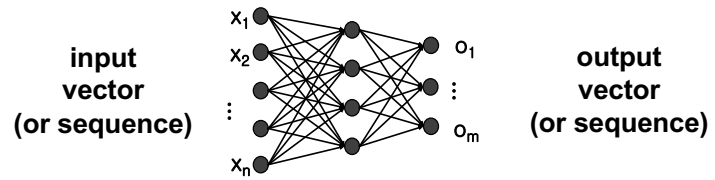
$$y = f(\sum_i w_i x_i + \theta)$$

Optimized for target task and data



Neural Networks

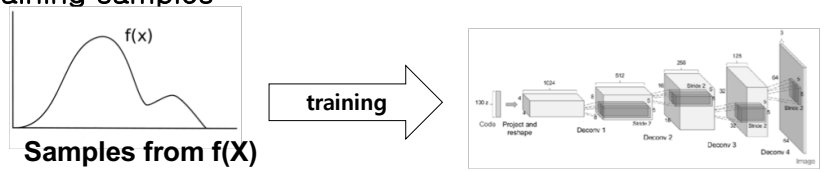
- Neural networks is a mathematical model to learn mappings
  - Mapping from a vector to another vector (or a scalar value)



Examples)
- Pattern → class (classification)
- Independent variables → dependent variables (regression)
- Symptoms → diseases (diagnosis)
- Sentence → another sentence (translation, dialogue)
- Text → Speech (TTS)
- State → action (control, game play)

Neural Networks

- Neural networks can learn probability distribution from training samples

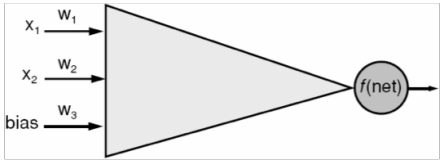


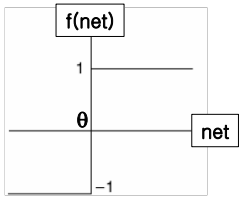
Examples)
- Estimates joint or conditional probability P(x, y), P(x|y)
  - likelihood P(x|ω), a posteriori probability P(ω|x)
- Sample generation
- Restoration $argmax_x P_\theta(x)$
- Transform $argmax_{x_{lost}} P_\theta(x_{lost}, x_{preserved})$

$argmax_{x_{transformed}} P_\theta(x_{transformed} \mid x_{source})$

Perceptron Neuron

- Perceptron [Rosenblatt 1958,62]
  - Input signals x_i
  - A set of real valued **weights**, w_i
  - **Activation level,** net = $\Sigma w_i x_i$
  - Activation function
    - Mapping from real value to a binary value (decision)
    - If net ≥ θ, output = 1, otherwise, −1





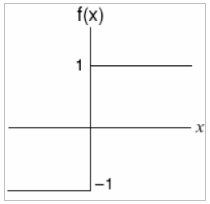
Hard-limiting function

Activation Functions

- Why activation functions?
  - Non-linearity
  - Restrict outputs in a specific range
  - Measurement → probability or decision

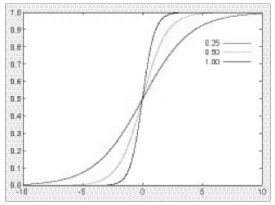
Hard-limit

$$f(x) = \begin{cases} +1 & if\ x \geq 0 \\ -1 & otherwise \end{cases}$$



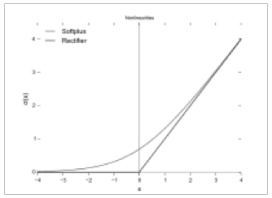
Sigmoid

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



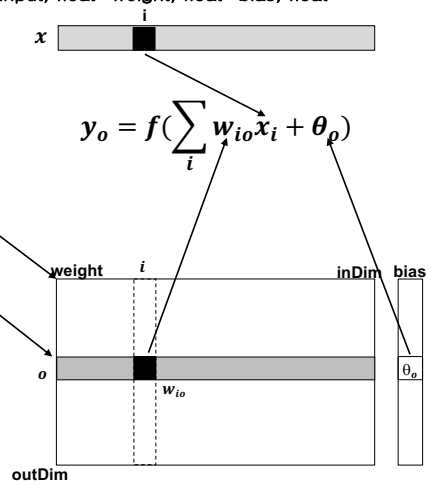
ReLU

$$f(x) = max(x, 0)$$



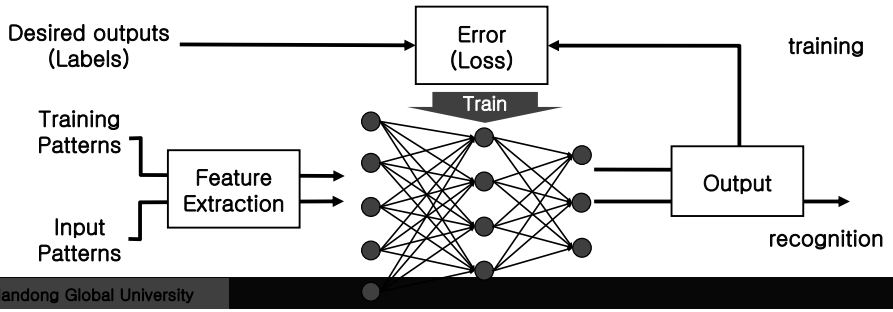
Forward Propagation

```
int ForwardPropagate(int inDim, int outDim, float *input, float *weight, float *bias, float
*output)
{
    for(int o = 0; o < outDim; o++){
        float *inWeight = weight + o * inDim;

        // compute net_o = Σ_i w_io x_i + θ_o
        float net = 0.F;
        for(int i = 0; i < inDim; i++)
            net += input[i] * inWeight[i];
        net += bias[o];              // bias

        // y_o = f(net_o)
        output[o] = max(net, 0);  // ReLU
    }

    return TRUE;
}
```



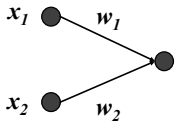
Ex) Building Neural Network Recognizer

1. Design network structure
2. Collect or acquire training samples (with labels)
3. Train connection weights
   - Given **training samples** and **desired outputs**, find **weights** that minimizes error.
4. Apply the trained neural network to target data



An Example of Perceptron Classifier

- Single layer perceptron classifier



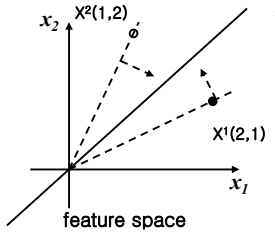
$$o = f\left(\sum w_i x_i\right) = f(w_1 x_1 + w_2 x_2)$$

  - Decision rule
    - If o(X) < 0, X is assigned with class 1
    - If o(X) > 0, X is assigned with class 2
  - Assume we have two training samples
    - X^1 = (2, 1), class 1 ➔ $o(X^1) < 0$
    - X^2 = (1, 2), class 2 ➔ $o(X^2) > 0$
  - What's the meaning of equation "o(X) = 0"?

Decision Boundary of Perceptron

- Equation o(X) = 0 forms a decision boundary
  - $o = f\left(\sum w_i x_i\right) = w_1 x_1 + w_2 x_2 = 0$ ➔ *equation of a line*

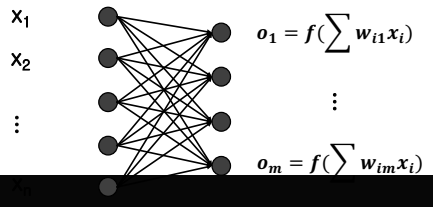
∴ The decision boundary generated by a perceptron is a line.



$$w_1 x_1 + w_2 x_2 = 0$$

Training of Single Layer Perceptron

- Training of perceptron: adjusting weights to approximate the target mapping
  - Given
    - Network structure and initial weights (random values)
    - A set of training samples
    - Labels (desired output) for each training sample
  - Goal
    - **Proper weights of perceptron**

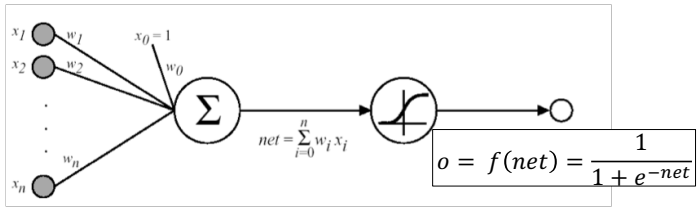


Perceptron Training Algorithms

- Single layer perceptron
  - Discrete perceptron training
  - Continuous perceptron training
    - Gradient descent
- Multi layer perceptron
  - Back-propagation algorithm

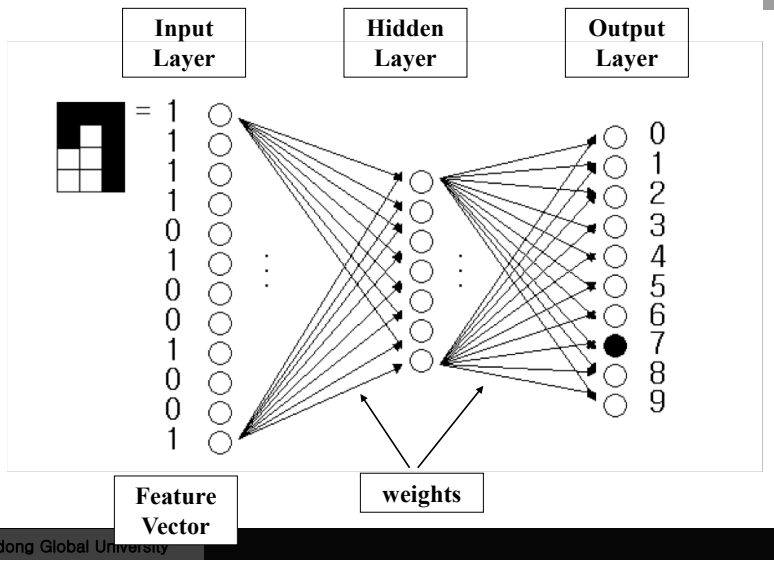
Gradient Descent

- Gradient descent: an iterative error minimization technique using gradient vector
  - Given a training data ($\boldsymbol{X}$) and desired outputs ($\boldsymbol{d}$)
    - Network output $o = f(net) = f(\Sigma_i w_i x_i)$
    - $\boldsymbol{E} = \frac{(o-d)^2}{2} = \frac{(f(\Sigma_i w_i x_i)-d)^2}{2}$



  - If d and X are fixed, error depends only on weights
    - Move weight vector to the direction that decreases error

Neural Networks Classifier

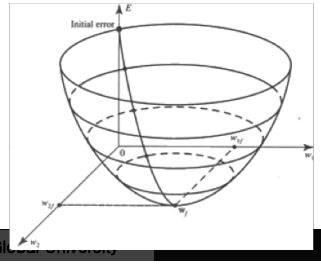


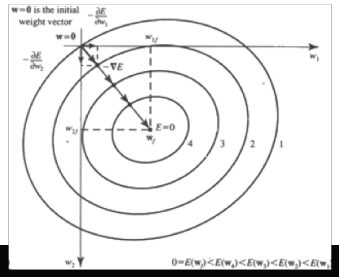
Gradient Descent

- Gradient descent for single layer perceptron
  - Error surface: error as a function of network weights

    Ex) single output: $\boldsymbol{E} = \frac{1}{2}(\boldsymbol{o} - \boldsymbol{d})^2$, multiple output: $\boldsymbol{E} = \frac{1}{2K}\sum_k(\boldsymbol{o_k} - \boldsymbol{d_k})^2$
  - Given current weights W^t, learning algorithm finds a direction in which error is reduced most rapidly
    - Gradient vector

$$\nabla \boldsymbol{E}(\boldsymbol{W}) = \frac{\boldsymbol{\partial E(W)}}{\boldsymbol{\partial W}} = (\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial w_1}}, \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial w_2}}, \dots, \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial w_n}})$$



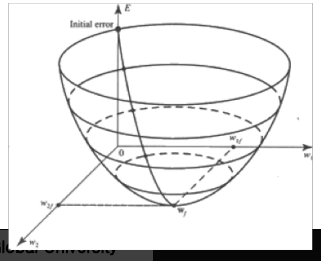


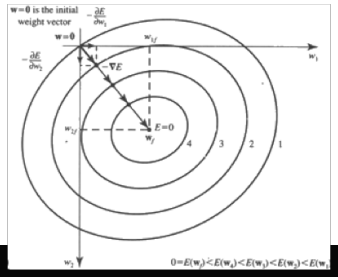
Gradient Descent

- Gradient descent for single layer perceptron
  - Update rule

$$W \leftarrow W - \eta \nabla E(W) = W - \eta \frac{\partial E(W)}{\partial W}$$

$$\nabla E(W) = \frac{\partial E(W)}{\partial W} = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_n})$$





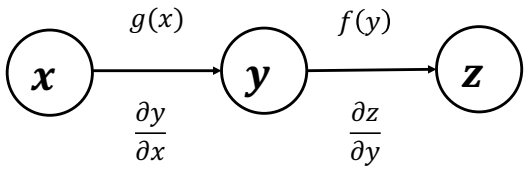
Chain Rule

- For real numbers x, y, and z

$$y = g(x),\ z = f(y) = f\big(g(x)\big)$$

  - Chain rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$



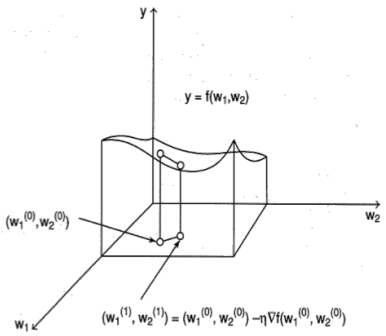
Gradient Descent

- Gradient descent learning
  - $W^{t+1} = W^t - \eta\Delta E(W^t)$
    - η: learning rate

$$\Delta \boldsymbol{E}(\boldsymbol{W}) = \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial W}} = \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial o}}\frac{\boldsymbol{\partial o}}{\boldsymbol{\partial net}}\frac{\boldsymbol{\partial net}}{\boldsymbol{\partial W}} = (\boldsymbol{o} - \boldsymbol{d})\boldsymbol{f'(net)}\boldsymbol{X}$$

    - $o = f(net) = f(\sum_i w_{ij}x_i + b_j)$
  - f'(net) of sigmoid function

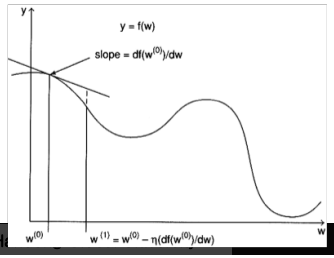
$$f'(net) = \frac{e^{-net}}{(1+e^{-net})^2} = o(1-o)$$

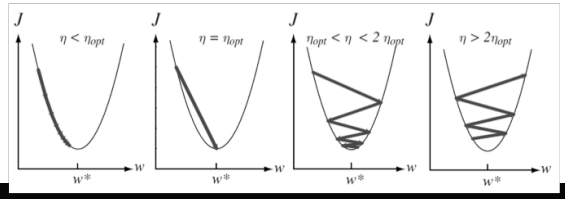


- Update rule
  - $W^{t+1} = W^t - \eta*(o-d)o(1-o)X$

Learning Rate

- Learning rate decides amount of movement in learning
  - Too small: slow, easy to fall in a local minima
  - Too large: fail to converge
- General strategy
  - Start training with a large η and decrease η as the training progresses





Training Algorithm

1. Select parameters
   - η: learning rate
   - E_{max}: acceptable error
2. Initialize W, with random weights
3. Repeat until ($E < E_{max}$) or (# of iteration < predefined limit)
   - Using current W, compute the network output o and error E

     $o = f(\Sigma_i w_i x_i)$

     $E = \frac{1}{2}(o - d)^2$
   - Update weight
     - $W^{t+1} = W^t - \eta*(o-d)o(1-o)X$

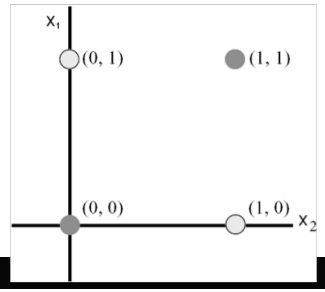
Agenda

- Single-layer Perceptron
- Multi-layer Perceptron
- Activation Functions
- Back-propagation

Limitation of Single-Layer Perception

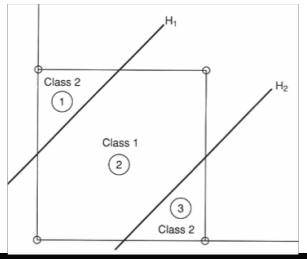
- Perceptron cannot solve even simple problems such as XOR [Minsky and Paper 1969]
  - A set of perceptron weight corresponds to a line in feature space
  - A perceptron can separate only **linearly separable patterns**
    - XOR is linearly non-separable problem

x_1	x_2	Output
1	1	0
1	0	1
0	1	1
0	0	0



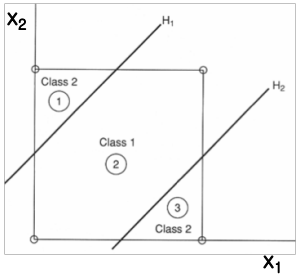
Multi-Layer Perceptron

- Limitation of perceptron
  - A perceptron node corresponds to a line
    - Cannot solve linearly non-separable patterns (ex: XOR)
- Idea: classify patterns with multiple lines
  - Classifier composed of two lines H_1 and H_2
    - Line H_1
      - Weights: W_1,
      - Output: $y_1 = f(W_1X)$
    - Line H_2
      - Weights: W_2
      - Output: $y_2 = f(W_2X)$
  - Class 1 vs. class 2
    - Class1: $y_1 < 0$ AND $y_2 > 0$
    - Class2: otherwise

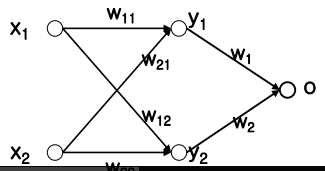


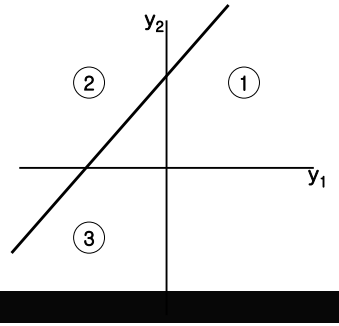
Multi-Layer Perceptron

- Classification using multiple lines



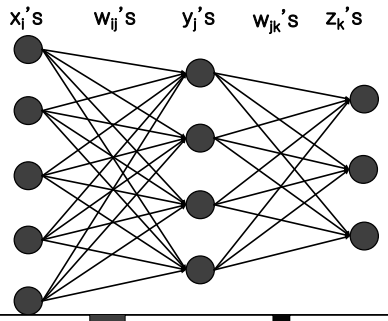
sample	y_1	y_2	class
1	+	+	class 2
2	−	+	class 1
3	−	−	class 2

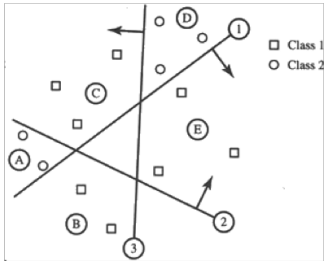




Multi-Layer Perceptron

- 2-layer perceptron (MLP) – not including input layer
  - Nodes of output layer integrates the outputs of hidden layer
    - $z_k = f(\Sigma_j w_{jk} y_j)$
    - # of nodes in hidden layer = # of lines in feature space

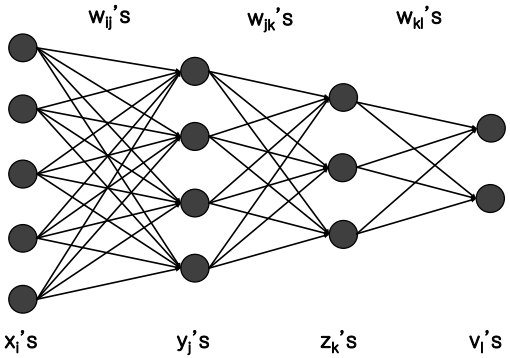




Feature space separated by 3 lines

Multi-Layer Perceptron

- Multi-layer perceptron
  - N^{th} layer integrates the output of $(N-1)^{th}$ layer

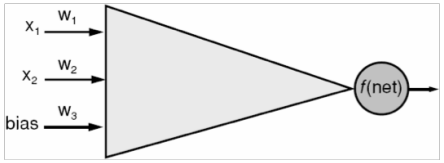


Agenda

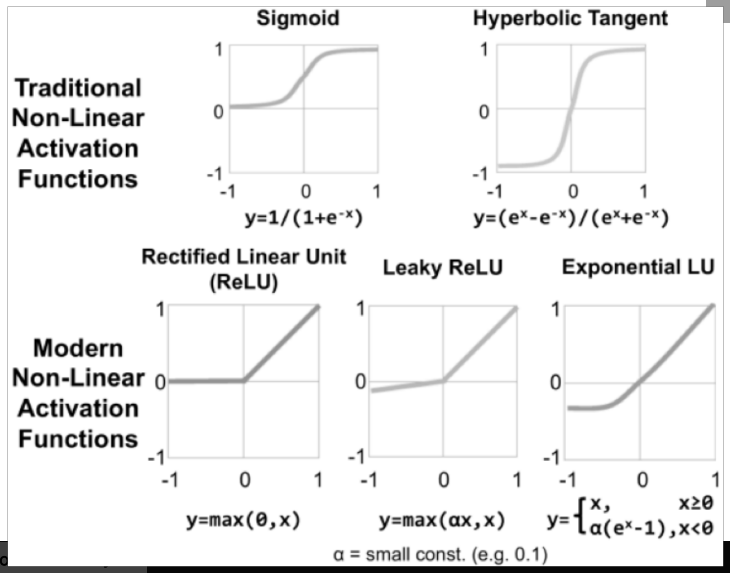
- Single-layer Perceptron
- Multi-layer Perceptron
- Activation Functions
- Back-propagation

Activation Functions

- A.k.a. “Non-linearity functions”
- Why activation functions?
  - Non-linearity
  - Restrict outputs in a specific range
  - Measurement → probability or decision



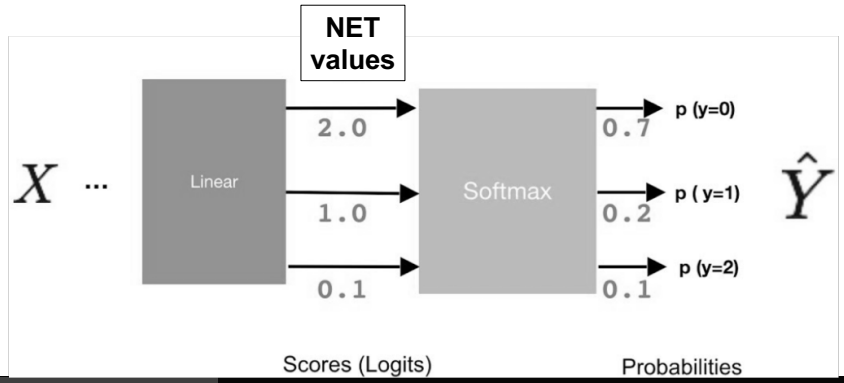
Activation Functions



Activation Functions

- Softmax

$$f(net) = \frac{\exp(net_i)}{\sum_j \exp(net_j)}$$



Activation Functions

- Hidden units
  - Sigmoid, Tanh
  - ReLU, LReLU, PReLU, RReLU, ELU, max-out
  - Linear, gated linear unit (GLU)
- Output units
  - Identity (no activation function): regression
  - Sigmoid: [0, +1] (eg. probability of independent event)
  - Tanh: [−1, +1]
  - Softmax: [0, +1] (eg. probabilities of competing classes)

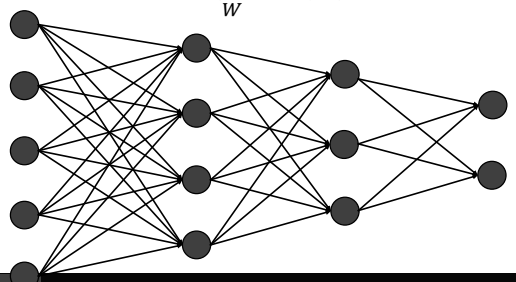
Agenda

- Single-layer Perceptron
- Multi-layer Perceptron
- Activation Functions
- Back-propagation

MLP Learning

- Given a training sample X^0 and its label c,
  - Desired output $D = \{ d_i \ 's \}$, $d_i = 1$ if $i = c$, otherwise, $d_i = 0$
- Define a loss function $E(W)$
- Find connection weights W^* usually by gradient-based algorithm

$$W^* = \arg\min_W E(W)$$



Loss Function (Error Criteria)

- Given
  - X_c^N: the output of the top level layer for the c^{th} class
  - $D = (d_1, d_2, \dots, d_C)$: desired output
- Mean square error

$$E_{MSE} = \frac{1}{2}\frac{\sum_c (X_c^N - d_c)^2}{C}$$

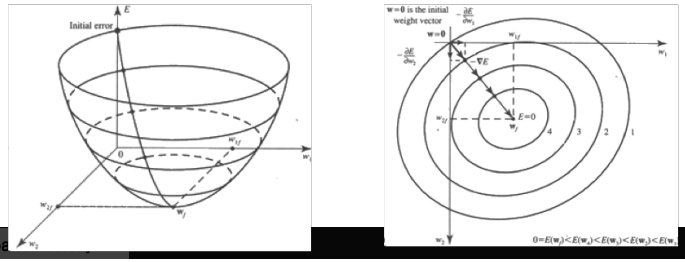
- Cross entropy (with softmax activation)
  - Softmax activation: $X_c^N = \frac{\exp(net_c^N)}{\sum_c \exp(net_c^N)}$

$$E_{CE} = -\sum_c d_c \log(X_c^N)$$

Gradient-based Learning

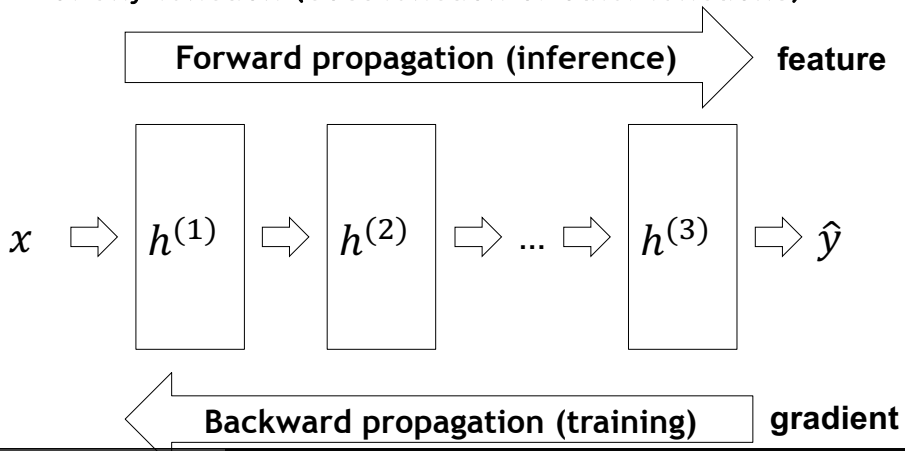
- Given current weights *W*, the gradient gives a direction in which increases the error most rapidly
  - Gradient of E(W) with respect to weight

$$\frac{\nabla E(W)}{\nabla W} = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_i}, \ldots, \frac{\partial E}{\partial w_M}\right)$$



Back-Propagation

- Back-propagation: a method for computing gradient of any function (cost function or other functions)



Matrix Notation

- Propagation
  - $y_j = f\left(\sum_i w_{ij} x_i + b_j\right) = f(a_j)$
  - $a_j = \sum_i w_{ij} x_i + b_j$
- Vector/matrix notation
  - $X = (x_1, x_2, \ldots, x_N)^T, Y = (y_1, y_2, \ldots, y_M)^T, A = (a_1, a_2, \ldots, a_M)^T$
  - $W = \begin{pmatrix} w_{11} & \cdots & w_{N1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{NM} \end{pmatrix}, B = (b, b_2, \ldots, b_M)^T$
  - $A = WX + B = \begin{pmatrix} w_{11} & \cdots & w_{N1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{NM} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_N \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_M \end{pmatrix} = \begin{pmatrix} \sum_i w_{i1} x_i + b_1 \\ \sum_i w_{i2} x_i + b_2 \\ \ldots \\ \sum_i w_{iM} x_i + b_M \end{pmatrix}$
  - $Y = F(A) = F(WX + B)$

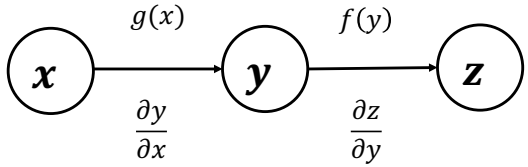
Chain Rule

- For real numbers x, y, and z

$$y = g(x),\ z = f(y) = f\left(g(x)\right)$$

- Chain rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$



Chain Rule

- Example [Stanford cs224n]

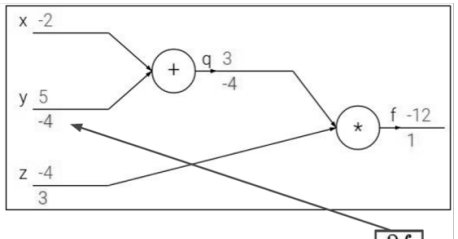
$f(x, y, z) = (x + y)z$

e.g. x = -2, y = 5, z = -4

$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

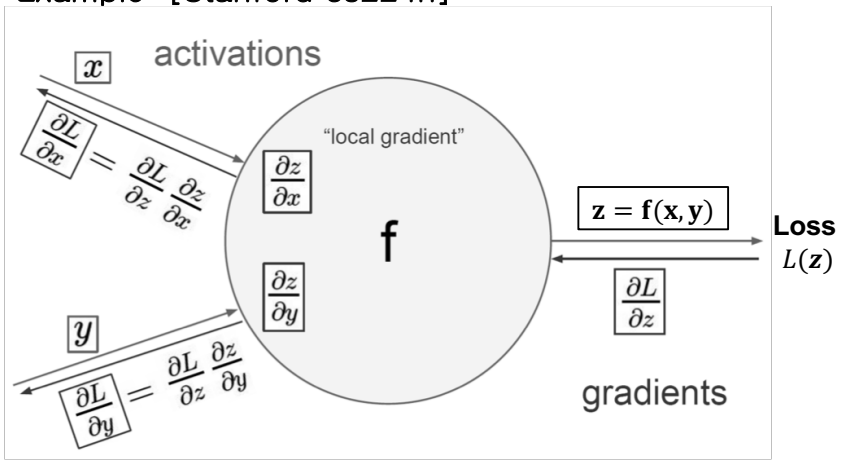


Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

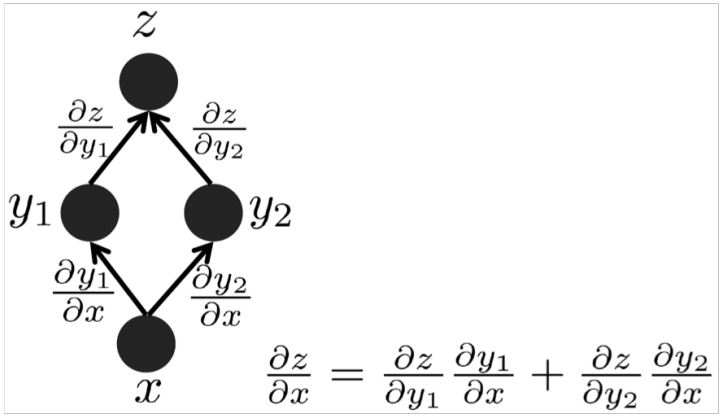
Chain Rule

- Example [Stanford cs224n]



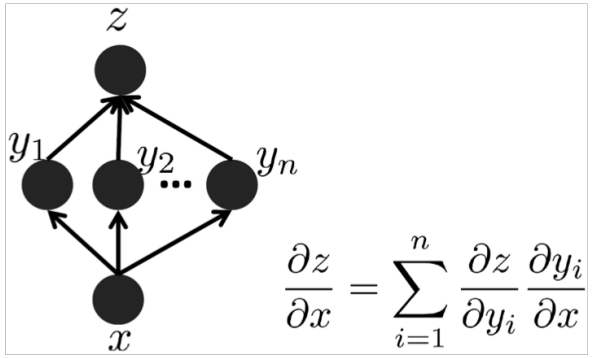
Chain Rule

- Multiple Paths Chain Rule [Stanford cs224n]



Chain Rule

- Multiple Paths Chain Rule [Stanford cs224n]



Chain Rule

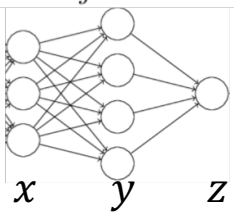
- For vectors $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$
  - Chain rule

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

gradient vectors

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z$$

n x m **Jacobian matrix**



- For tensors of arbitrary dim ?
  - Flatten the tensor into a vector

Gradient and Jacobian

- Gradient vector: a multi-variable generalization of the derivative. (f is a scalar-valued function)

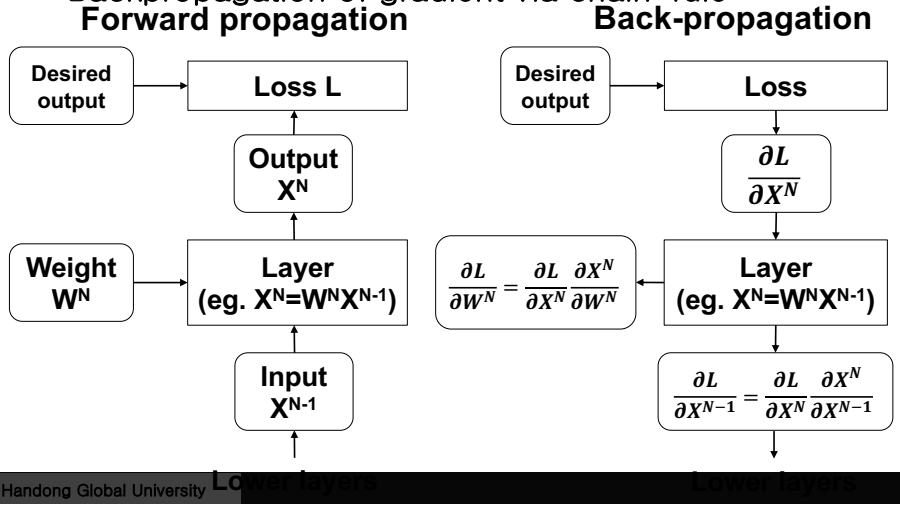
$$\frac{\partial f}{\partial x} = \nabla_x f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

- Jacobian matrix: matrix of all 1st order partial derivatives of a vector-valued function

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Back-Propagation on Neural Nets

- Backpropagation of gradient via chain-rule



Back-Propagation on MLP

- Mean square error (MSE)

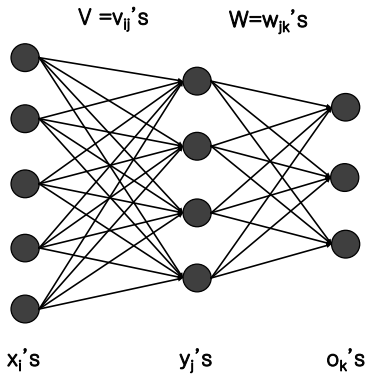
$$E_{MSE} = \frac{1}{2}\frac{\sum_k (\boldsymbol{o}_k - d_k)^2}{K}$$

- Training algorithm
  - 2nd layer
    - $W^{t+1} = W^t + \Delta W$

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}}$$

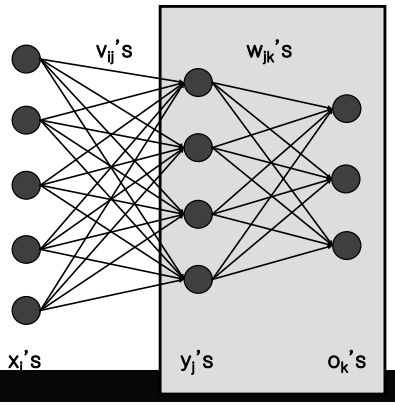
  - 1st layer
    - $V^{t+1} = V^t + \Delta V$

$$\Delta v_{ij} = -\eta \frac{\partial E}{\partial v_{ij}}$$



Training of 2nd Layer

- Update formula for 2nd layer
  - $W^{t+1} = W^t + \Delta W$
    - $w_{jk}^{t+1} = w_{jk}^t + \Delta w_{jk}$, where $\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}}$



Training of 2nd Layer

Actually, $\frac{\partial NET}{\partial W}$ is a $|NET| * |W|$ matrix, because W is regarded as a vector

- Gradient for weight update

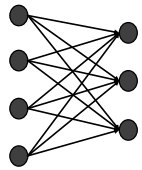
$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial O}\frac{\partial O}{\partial NET}\frac{\partial NET}{\partial W} = \left(\frac{\partial E}{\partial o_1}, \frac{\partial E}{\partial o_2}, \dots, \frac{\partial E}{\partial o_K}\right)\begin{pmatrix} \frac{\partial o_1}{\partial net_1} & 0 & \dots \\ 0 & \frac{\partial o_2}{\partial net_2} & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}\begin{pmatrix} \frac{\partial net_1}{\partial w_{11}} & \frac{\partial net_1}{\partial w_{21}} & \dots \\ \frac{\partial net_2}{\partial w_{12}} & \frac{\partial net_2}{\partial w_{22}} & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

$$\frac{\partial E}{\partial o_k} = \frac{\partial}{\partial o_k}\frac{1}{2}\frac{\sum_k (o_k - d_k)^2}{K} = \frac{1}{K}(o_k - d_k)$$

$$\frac{\partial o_k}{\partial net_k} = \frac{\partial f(net_k)}{\partial net_k} = f'(net_k) \qquad \frac{\partial net_k}{\partial w_{jk}} = \frac{\partial \sum_j w_{jk} y_j}{\partial w_{jk}} = y_j$$

- Gradient with respect to w_{jk}

$$\boxed{\frac{\partial E}{\partial w_{jk}} = \frac{1}{K}(o_k - d_k) f'(net_k) y_j}$$



Training of 2nd Layer

- Gradient for back-propagation

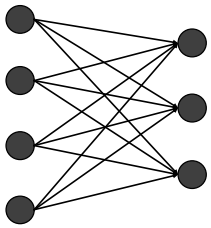
$$\frac{\partial E}{\partial Y} = \frac{\partial E}{\partial O}\frac{\partial O}{\partial NET}\frac{\partial NET}{\partial Y} = \left(\frac{\partial E}{\partial o_1}, \frac{\partial E}{\partial o_2}, \dots, \frac{\partial E}{\partial o_K}\right)\begin{pmatrix} \frac{\partial o_1}{\partial net_1} & 0 & \dots \\ 0 & \frac{\partial o_2}{\partial net_2} & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}\begin{pmatrix} \frac{\partial net_1}{\partial y_1} & \frac{\partial net_1}{\partial y_2} & \dots \\ \frac{\partial net_2}{\partial y_1} & \frac{\partial net_2}{\partial y_2} & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

$$\frac{\partial E}{\partial o_k} = \frac{\partial}{\partial o_k}\frac{1}{2}\frac{\sum_k (o_k - d_k)^2}{K} = \frac{1}{K}(o_k - d_k)$$

$$\frac{\partial o_k}{\partial net_k} = \frac{\partial f(net_k)}{\partial net_k} = f'(net_k) \qquad \frac{\partial net_k}{\partial y_j} = \frac{\partial \sum_j w_{jk} y_j}{\partial y_j} = w_{jk}$$

- Gradient with respect to y_j

$$\boxed{\frac{\partial E}{\partial y_j} = \sum_k \frac{1}{K}(o_k - d_k) f'(net_k) w_{jk}}$$



Training of 2nd Layer

- Gradients

$$\frac{\partial E}{\partial w_{jk}} = \frac{1}{K}(o_k - d_k) f'(net_k) y_j$$

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{1}{K}(o_k - d_k) f'(net_k) w_{jk}$$

- When using Sigmoid nonlinearity

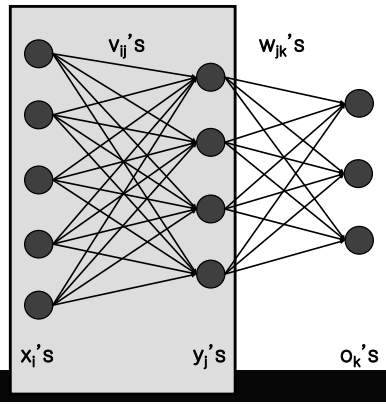
$$f(net) = \frac{1}{1+e^{-net}},\ f'(net) = \frac{e^{-net}}{(1+e^{-net})^2} = o(1-o)$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{1}{K}(o_k - d_k) o_k (1 - o_k) y_j$$

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{1}{K}(o_k - d_k) o_k (1 - o_k) w_{jk}$$

Training of 1st Layer

- Training of hidden layers
  - $V^{t+1} = V^t + \Delta V$
    - $v_{ij}^{t+1} = v_{ij}^t + \Delta v_{ij}$, where $\Delta v_{ij} = -\eta \frac{\partial E}{\partial v_{ij}}$



Training of 1st Layer

- Gradient for weight update

Actually, $\frac{\partial NET}{\partial V}$ is a $|NET| * |V|$ matrix, because V is regarded as a vector

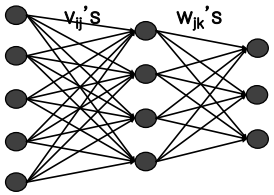
$$\frac{\partial E}{\partial V} = \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial NET}\frac{\partial NET}{\partial V} = \left(\frac{\partial E}{\partial y_1}, \frac{\partial E}{\partial y_2}, \cdots, \frac{\partial E}{\partial y_J}\right)\begin{pmatrix} \frac{\partial y_1}{\partial net_1} & 0 & \cdots \\ 0 & \frac{\partial y_2}{\partial net_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}\begin{pmatrix} \frac{\partial net_1}{\partial v_{11}} & \frac{\partial net_1}{\partial v_{21}} & \cdots \\ \frac{\partial net_2}{\partial v_{12}} & \frac{\partial net_2}{\partial v_{22}} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

$$\frac{\partial E}{\partial y_j} = \frac{1}{K}\sum_k (o_k - d_k) f'(net_k) w_{jk}$$

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f(net_j)}{\partial net_j} = f'(net_j) \qquad \frac{\partial net_j}{\partial v_{ij}} = \frac{\partial \sum_i v_{ij} x_i}{\partial v_{ij}} = x_i$$

- General formula for delta learning

$$\frac{\partial E}{\partial v_{ij}} = \sum_k \left[\frac{1}{K}(o_k - d_k) f'(net_k) w_{jk}\right] f'(net_j) x_i$$



3. Deep Learning Overview

Injung Kim
Handong Global University
2019. 1. 7.

Agenda

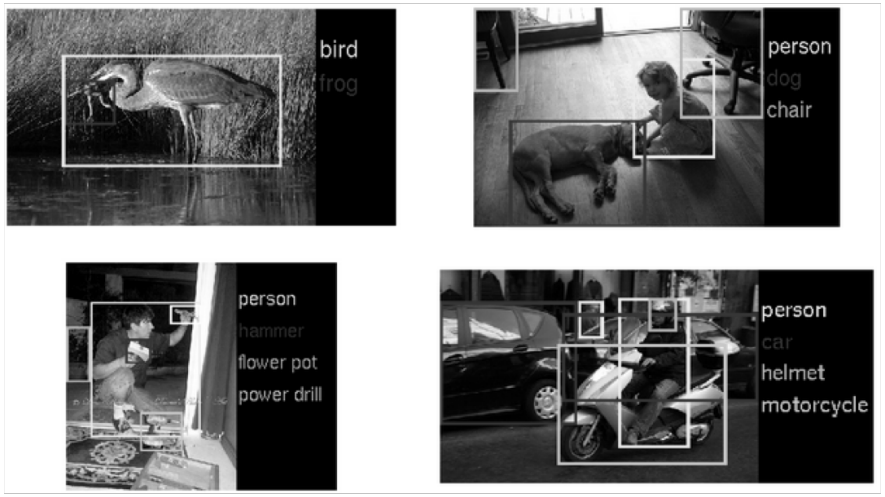
- Success Stories
- Introduction to Deep Learning
- Deep Learning Architectures
- Practical Problems and Solutions
- Q&A

ImageNet Dataset

- ImageNet Large Scale Visual Recognition Challenge (http://www.image-net.org)
  - 1000 object categories
  - Training set: 1,281,167 images
  - Validation set: 50,000 images
  - Test set: 100,000 images

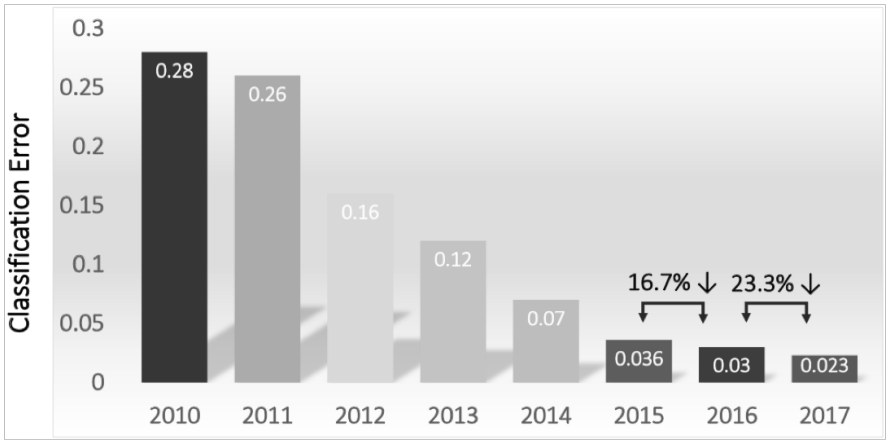
ImageNet Dataset

1.3M training + 50k validation + 100k test images



ILSVRC (ImageNet) Results

- 2017 SOTA error rate: 2.3%



Instance Segmentation

- He, et.al., "Mask R-CNN," 2017

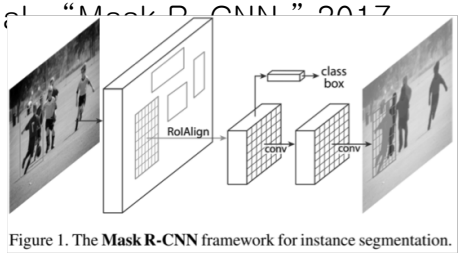


Figure 1. The **Mask R-CNN** framework for instance segmentation.

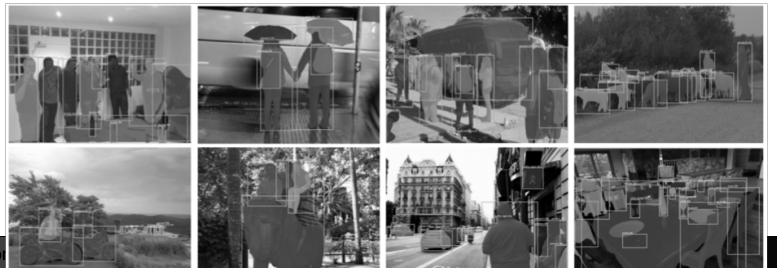
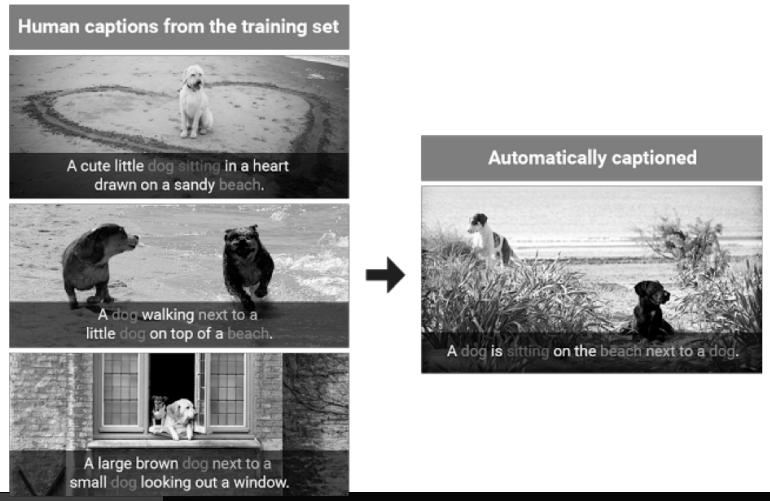


Image Captioning



Dense Captioning

- Johnson, et.al., "DenseCap: Fully Convolutional Localization Networks for Dense Captioning," 2017.5
  - Object detection + captioning

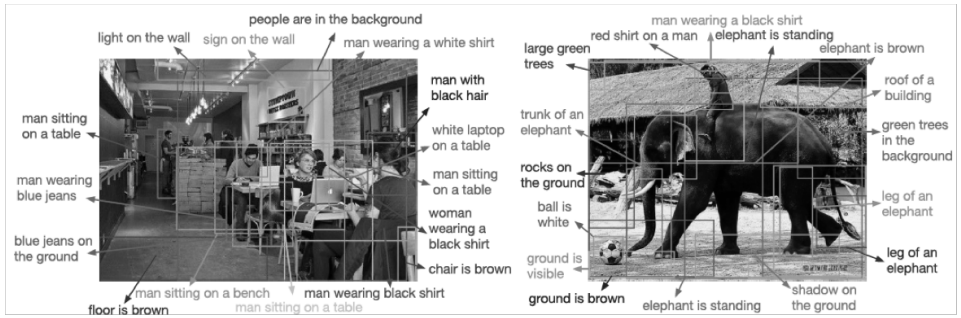


Image Synthesis: ProGAN

- Karras, et.al., Progressive Growing of GANs for Improved Quality, Stability, and Variation, Nov., 2017.

Image Synthesis: BigGAN

- A. Brock, et al., "LARGE SCALE GAN TRAINING FOR HIGH FIDELITY NATURAL IMAGE SYNTHESIS" 2018.
  - Large-scale GAN training using large batch
  - Truncation trick for random noise generation
    - Trade-off between variety and fidelity)
  - Orthogonal regularization to the generator

Images synthesized by BigGAN

Image-to-Image Translation

- Image-to-Image translation using conditional GAN [Isola16]

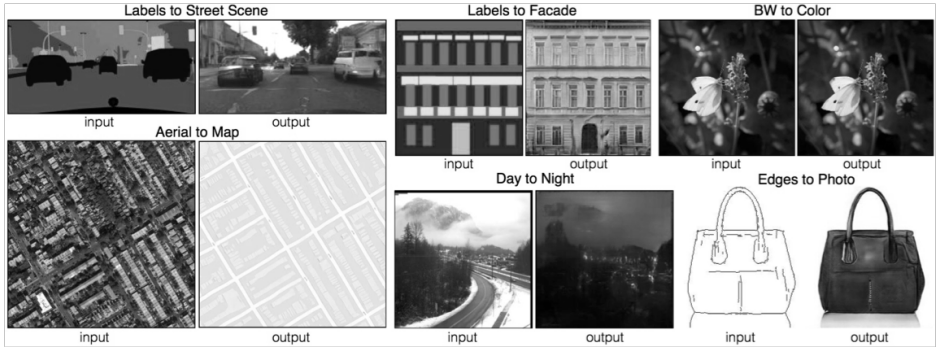
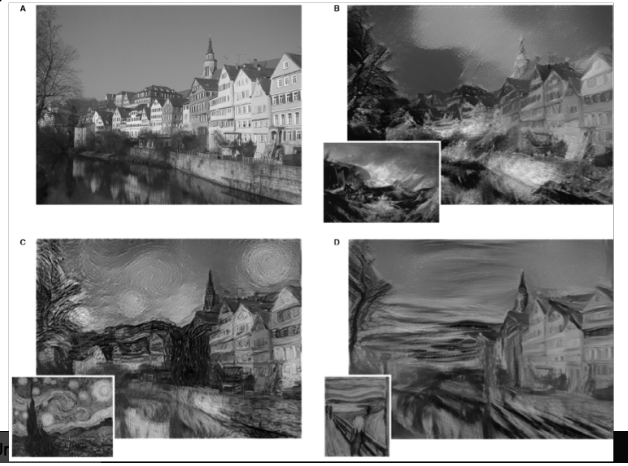
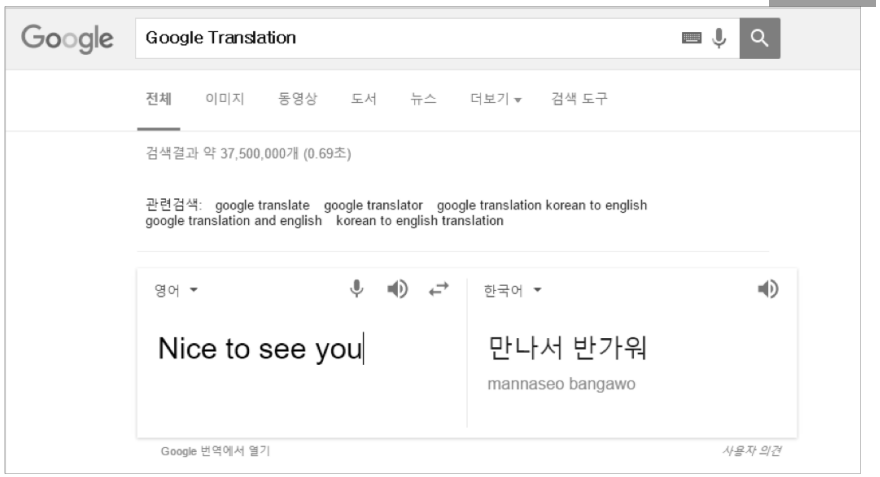


Image Style Transfer

- L. Gatys, et al., "A Neural Algorithm of Artistic Style," 2015

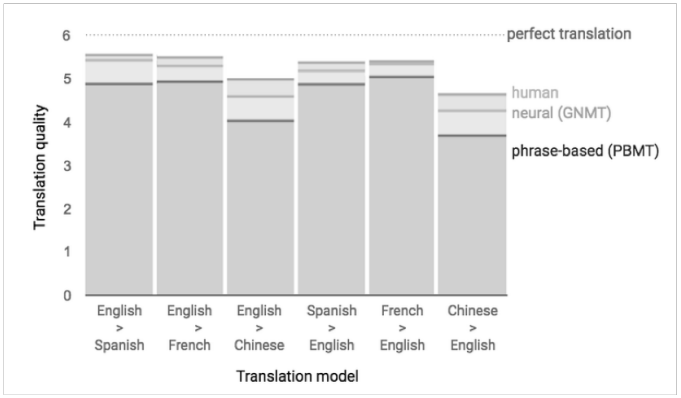


Google Neural Machine Translation



Machine Translation

- Google Neural Machine Translation (GNMT)
  - performance



Agenda

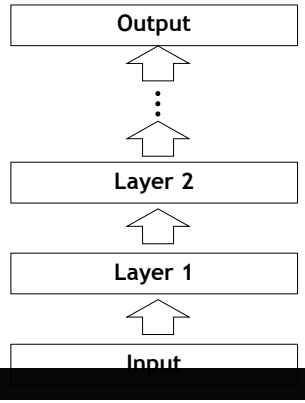
- Success Stories
- Introduction to Deep Learning
- Deep Learning Architectures
- Practical Problems and Solutions
- Q&A

Deep Learning

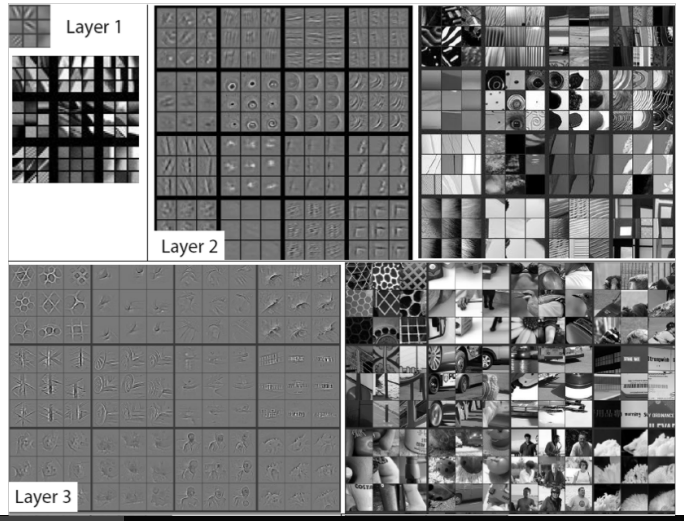
- A branch of machine learning to model **high-level abstractions in data**, mostly, based on **deep networks**.
  - Each layer combines input features to produce high-level features.

$$o = f(\sum_{i=1}^{n} w_i x_i + \theta)$$

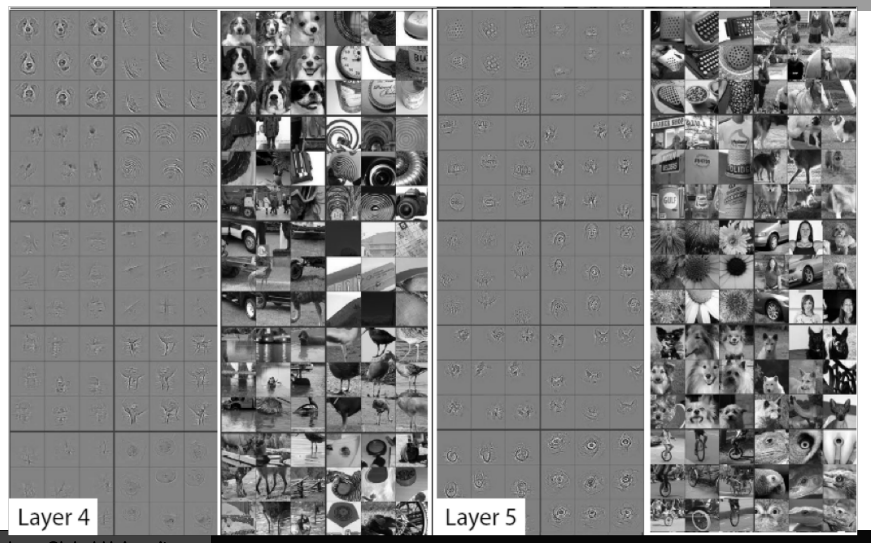
  - A neural network with many layers can learn high-level features.



Visualization of Low-level Layers [Zeiler2013]

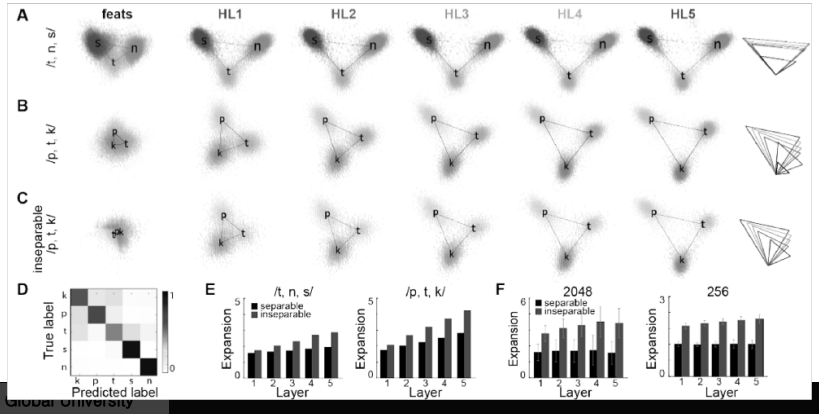


Visualization of High-level Layers [Zeiler2013]



Feature of MLP Layers

- Nagamine and Mesgarani, “Understanding the Representation and Computation of Multilayer Perceptrons: A Case Study in Speech Recognition,” 2017.



Why Deep Networks?

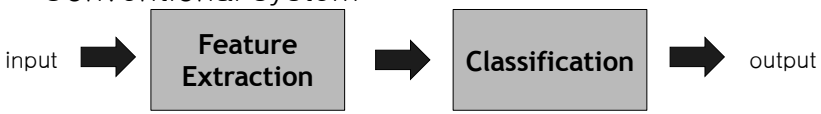
- Efficient in learning high-level feature representation
- Integrated learning
  - DNN integrates feature extractor and classifier in a single network
- Efficient in modeling of **highly varying functions**
- Large capacity
  - DNN can learn very well from a huge volume of samples
- Framework that embraces various methodologies and techniques

High-level Feature Representation

- Good representation of information is crucial in AI
- Improvement of handcrafting feature is stagnant.
  - Performance bottleneck (discriminative ability vs robustness)
- High-level feature representation learning from data
  - Optimized for target task
  - More informative and robust than lower-level features

High-Level Feature Learning

- Deep networks optimize both feature extractor and classifier in a unified framework.
- Conventional system



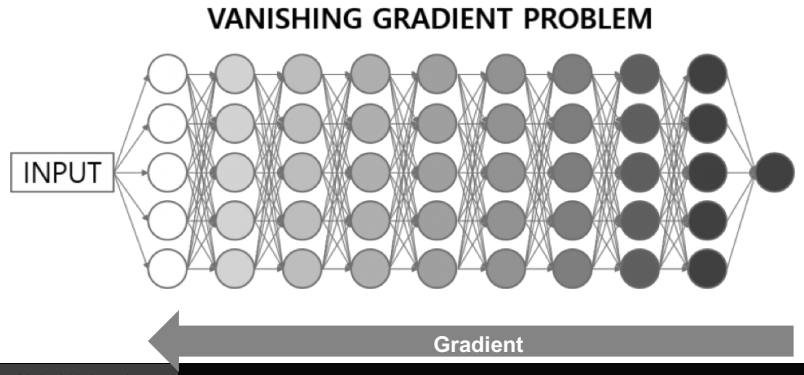
- Deep Network

Challenges in Deep Learning

- Difficulties in deep learning
  - Backpropagation algorithm does not work or slow
  - Not better than shallow networks
- Why?
  - The vanishing/exploding gradient problem
  - Local minima, saddle points, plateaus
  - Overfitting
  - Internal covariate shift [Ioffe15]
  - Scattered gradient problem [Balduzzi17]
  - Many unknown reasons

Vanishing Gradient Problem

- Conventional back-propagation algorithm does not work for deep networks.

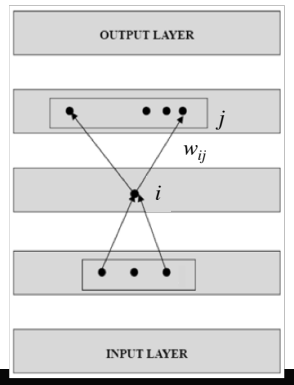


Vanishing Gradient Problem

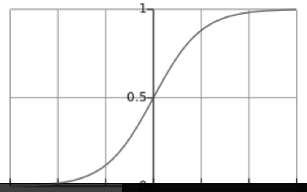
- Conventional back-propagation algorithm does not work well for deep networks.
  - Backpropagation formula

$$\delta_i = f'(net_i) \sum_j w_{ij}\delta_j$$

Blended error signal

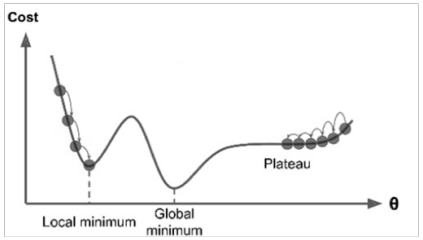


Saturated regime of Activation functions

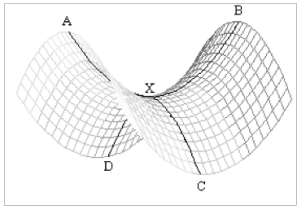


Local Minima, Saddle Point, Plateau

- Learning sometimes stops at local minima, saddle points or plateaus
  - Gradient is near zero.



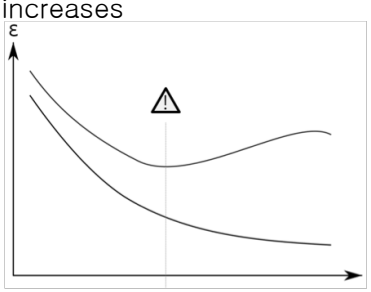
Local minima / Plateau



Saddle point

Overfitting

- Overfitting occurs when a model is excessively complex relative to the number of observations.
  - Training error decreases
  - Test error increases



Error on test samples

Error on training samples

- Remedies
  - More data or simpler model
  - Regularization, transfer learning, batch norm, dropout, etc.

Deep Learning Approaches

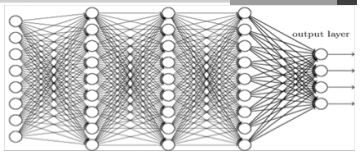
- Layer-wise unsupervised pre-training
  - DBN, stacked auto-encoders
- Architectures to avoid vanishing gradient problem
  - **Convolutional neural networks (CNN)**
    - Sparse connection, shared weights
  - **Gated units (LSTM, GRU, GLU)**
- Improved structures and learning algorithms
  - **Piece-wise linear activation functions**
    - max-out, **ReLU**, **LReLU**, **PReLU**, EReLU, etc.…
  - **Skip connection (ResNet, DenseNet, DPN)**
  - **Batch normalization**
  - **Xavier initialization, He initialization**, LL-initialization
  - Transfer learning, multi-task learning
  - Auxiliary networks, deeply supervised network

Agenda

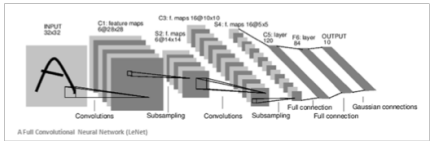
- Success Stories
- Introduction to Deep Learning
- Deep Learning Architectures
- Practical Problems and Solutions
- Q&A

Deep Learning Architectures

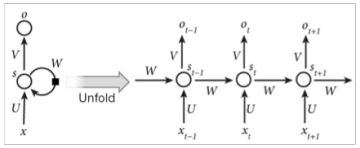
- Deep Neural Networks (DNN)
  - MLP, SOM, RBF, …
  - RBM, DBN, DBM, …



- Convolutional neural networks (CNN)
  - Recognition/processing/generation of **images**
  - Combines heterogeneous layers (convolution, pooling, etc.)
  - Learns position independent local features



- Recurrent neural networks (RNN)
  - Recognition/processing/generation of **time-series data**
  - Recurrent connection (memory)
  - Input + context

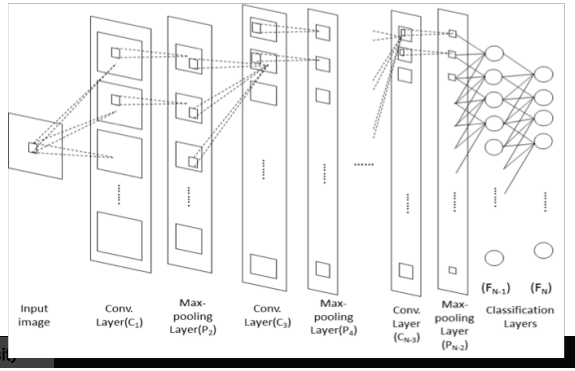


Deep Learning Architectures

- Hybrid models / combined models
  - Convolutional RBM, Recurrent CNN, Long-term recurrent convolutional network (LSTM + CNN), CBHG
- Deep generative models
  - GAN, VAE, pixel RNN/CNN
- Detection / segmentation models
  - R-CNN, Fast/Faster R-CNN, Mask R-CNN, YOLO, SSD
  - FCN, SegNet, RefineNet, DeepLab, JPP net, etc.
- Attention models
  - Machine translation, ASR/TTS, visual recognition,
  - Non-local nets, self-attention
- Explicit memory models
  - Memory networks, neural Turing machines (NTM)

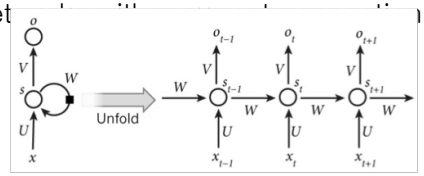
Convolutional Neural Networks

- Composed of heterogeneous layers
  - Conv, pooling, fully-connected, dropout, batch-norm, …
- Learns **position-independent local feature**
  - Sparse connection, parameter tying



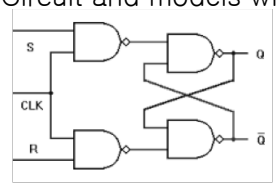
Recurrent Neural Networks

- Recurrent neural networks
  - Neural net

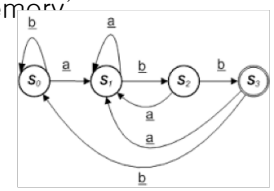


Recurrent neural networks

cf. Circuit and models with 'memory'



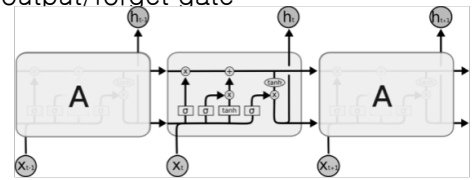
Flip-flop



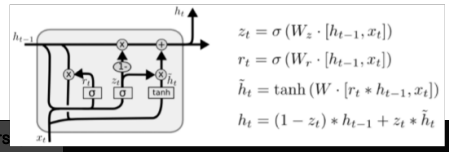
Finite state automata

LSTM and GRU

- LSTM (Long Short-Term Memory) network
  - Input/output/forget gate



- GRU (Gated Recurrent Unit)
  - From LSTM, combines forget and input gates into 'update' gate



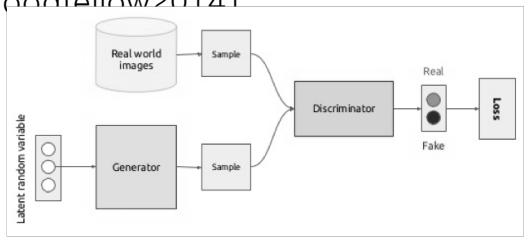
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Emerging Generative Models

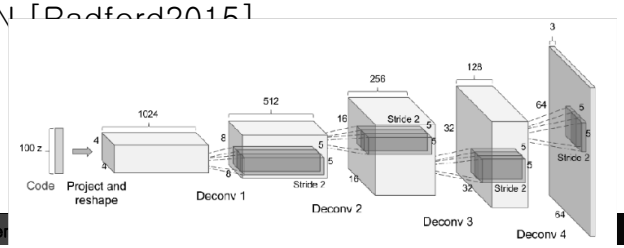
- Generative Adversarial Networks [Goodfellow14]
  - Goodfellow, et.al., "Generative Adversarial Nets," 2014
  - DCGAN, Conditional GAN, InfoGAN, EBGAN, BEGAN, CycleGAN, DiscoGAN, StarGAN, Unrolled GAN, WGAN, WGAN-GP, Cramer GAN, LS-GAN, Progressive GAN, SN-GAN, SA-GAN, BigGAN …
- Variational Auto-Encoders (VAE)
  - Kingma, et.al, "Auto-Encoding Variational Bayes," 2013
- Pixel Networks
  - Oord, et.al, "Pixel Recurrent Neural Networks," 2016
  - Pixel RNN, Pixel CNN, Pixel CNN++

Deep Convolutional GAN (DCGAN)

- GAN [Goodfellow2014]



- DCGAN [Radford2015]



Agenda

- Success Stories
- Introduction to Deep Learning
- Deep Learning Architectures
- Practical Problems and Solutions
- Q&A

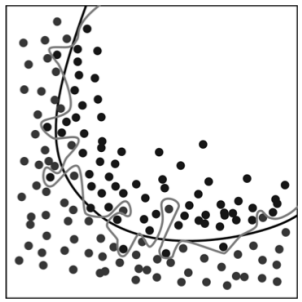
Practical Issues

- Requires a large volume of training data
  - Large scale public/private data collected from SNS
  - Regularization techniques
  - Unsupervised/semi-supervised/reinforcement learning
  - Transfer learning, one-shot learning, meta-learning
- Requires heavy computation
  - GPU, Cloud, TPU, FPGA, Neuromorphic Computing, etc.
- Requires much effort to implement and debug
  - Open source frameworks
    - TensorFlow, PyThorch, **WICWIU**, etc.

Regularization

- Introduce additional information to solve ill-posed problems or prevent over-fitting
- Weight shrinking
  - Loss function with regularization
  - $E(W) + \lambda \|W\|$
    - E(W): error
    - λ: regularization factor
    - $\|W\|$: norm of W
      - L_1 or L_2 regularization

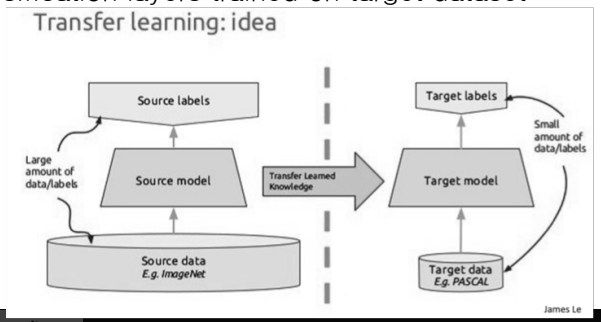
[Wikipedia]



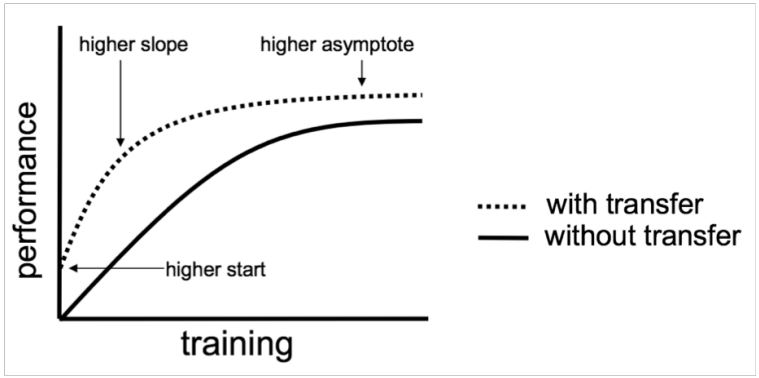
Transfer Learning

- Transfer learning applying knowledge gained while solving one problem to a different but related problem.

  Ex) Feature extraction layers pretrained on ImageNet dataset + classification layers trained on target dataset

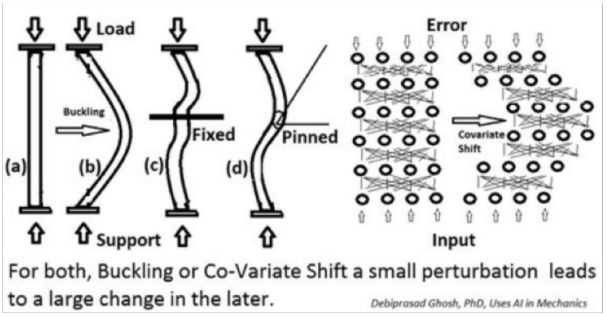


Transfer Learning



Internal Covariate Shift

- Internal covariate shift [Ioffe15]
  - Change in the output distribution of the preceding layer makes learning the following layers difficult.



Batch Normalization

- Normalization to minimize internal covariate shift

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

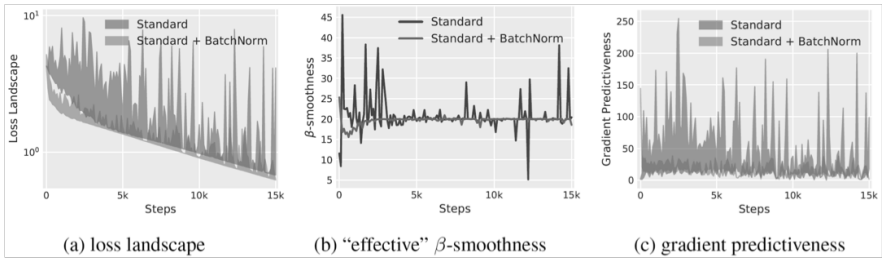
- Scale and shift to relax constraint

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

  - $\gamma^{(k)}$ and $\beta^{(k)}$ are trainable parameters

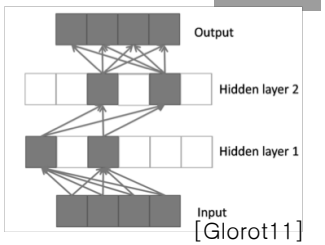
Batch Normalization

- Santurkar, et.al., “How Does Batch Normalization Help Optimization, ” 2018
  - Controls internal covariate shift (?)
  - Makes the optimization landscape significantly smoother

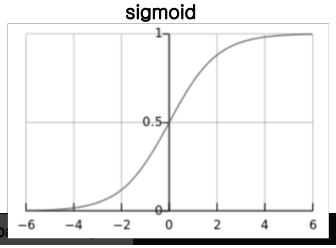


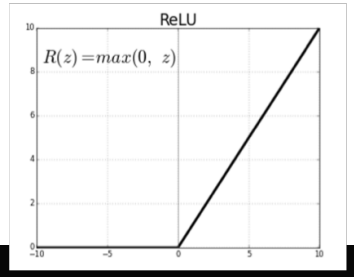
ReLU Activation Function [Hinton10]

- ReLU (Rectified Linear Unit)
  - Faster than Sigmoid or Tanh
  - Makes network activation sparse
  - No 'saturated regime'
- Problems
  - No gradient for negative input
  - Unbounded in positive direction

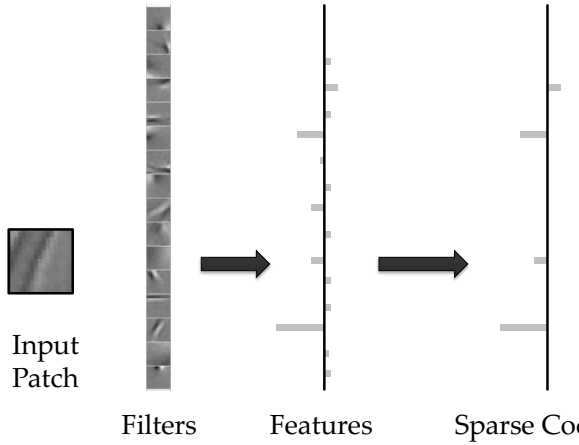


[Glorot11]





Sparse Coding

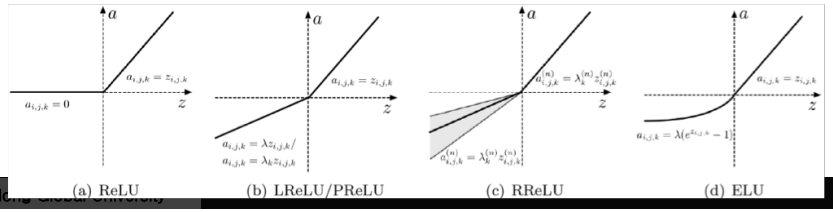


Variations of ReLU

- Leaky ReLU (LReLU)
  - λ is fixed
- Parametric ReLU (PReLU)
  - λ is learned
- Randomized ReLU (RReLU)
  - λ is randomly sampled
- Exponential LU (ELU)

$$a_{i,j,k} = \max(z_{i,j,k}, 0) + \lambda_k \min(z_{i,j,k}, 0)$$

$$a_{i,j,k} = \max(z_{i,j,k}, 0) + \min(\lambda(e^{z_{i,j,k}} - 1), 0)$$



Practical Issues

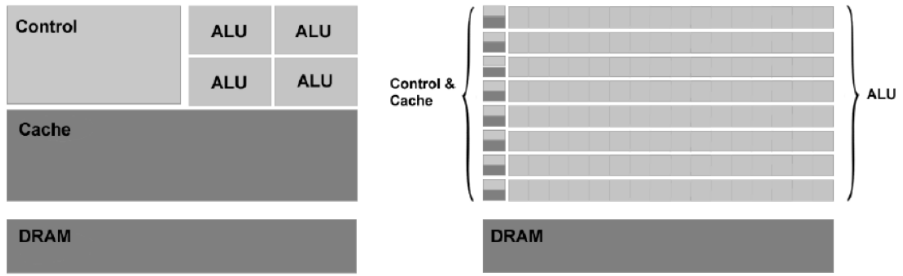
- Requires huge volume of training data
  - Large scale public/private datasets collected from SNS
  - Regularization techniques
  - Unsupervised/semi-supervised/reinforcement learning
- Requires heavy computation
  - GPU, Cloud, TPU, FPGA, Neuromorphic Computing, etc.
- Requires much effort to implement and debug
  - Open source frameworks
    - TensorFlow, Theano, Caffe, Torch, WICWIU, etc.

Computing on GPU

- GPU-based massive parallel processing
  - 20~100 times faster than CPU-based implementation
  - Training single epoch for HCCR (3755 classes): 3.4 hours

CPU is optimized for sequential processing

GPU is optimized for massive number of floating-point calculation



Practical Issues

- Requires huge volume of training data
  - Large scale public/private datasets collected from SNS
  - Regularization techniques
  - Unsupervised/semi-supervised/reinforcement learning
- Requires heavy computation
  - GPU, Cloud, TPU, FPGA, Neuromorphic Computing, etc.
- Requires much effort to implement and debug
  - Open source frameworks
    - TensorFlow, Theano, Caffe, Torch, WICWIU, etc.

Popular Deep Learning SW

- Caffe (caffe.berkeleyvision.org), Caffe2(caff2.ai)
  - BVLC (Berkeley Vision and Learning Center)
  - CNN, Vision, C++
- TensorFlow (www.tensorflow.org)
  - Google
  - CNN, RNN, Multi-GPU, Cloud, Tensor-board
- Torch (torch.ch), PyTorch (pytorch.org)
  - NYU, Facebook, DeepMind, Twitter
  - Fast
- Theano (deeplearning.net/software/theano)
  - The Theano Development Team
  - Fast
- WICWIU (github.com/WICWIU/WICWIU) – "What I Create, What I Understand"
  - Handong Global University
  - C++, readable, easy for hands-on

DAY 2

Convolutional Neural Networks

Injung Kim
Handong Global University
2019. 1. 8.

Agenda

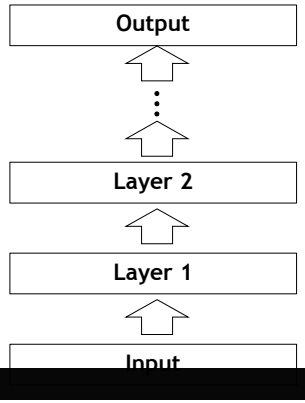
- Introduction
- Fundamentals of CNNs
- Training of CNN
- Optimization Algorithms
- Advanced CNN Models
- Detection Models
- Semantic Segmentation

Deep Learning

- A branch of machine learning to model **high-level abstractions in data**, mostly, based on **deep networks**.
  - Each layer combines input features to produce high-level features.

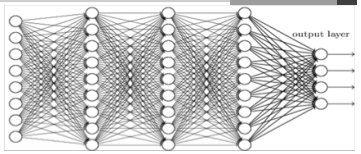
$$o = f(\sum_{i=1}^{n} w_i x_i + \theta)$$

  - A neural network with many layers can learn high-level features.

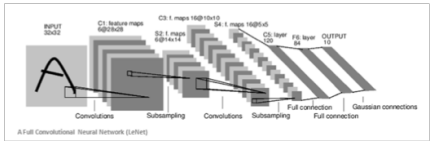


Deep Learning Architectures

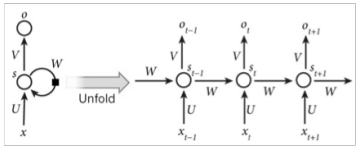
- Deep Neural Networks (DNN)
  - MLP, SOM, RBF, …
  - RBM, DBN, DBM, ⋯



- **Convolutional neural networks (CNN)**
  - Recognition/processing/generation of **images**
  - Combines heterogeneous layers (convolution, pooling, etc.)
  - Learns position independent local features



- **Recurrent neural networks (RNN)**
  - Recognition/processing/generation of **time-series data**
  - Recurrent connection (memory)
  - Input + context



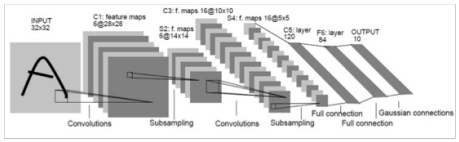
Fundamentals of CNNs

Biological Discoveries [Hubel&Wiesel 1962]

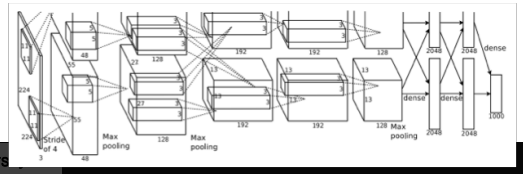
- Visual area – neurons responds selectively to **local feature** of a visual patterns
  - Ex) lines and edges.
- Higher area – neurons responds selective to **higher level features**
  - Ex) circles, triangles, squares, human face
- Neural networks in brain are not complete at birth.
  - They gradually develop, adapting flexibly to circumstances after birth.

Convolutional Neural Networks

- **Convolutional Neural networks** (CNN): a class of deep feed-forward network designed to mimic human/animal visual systems
  - LeNet 1998, MNIST, CPU

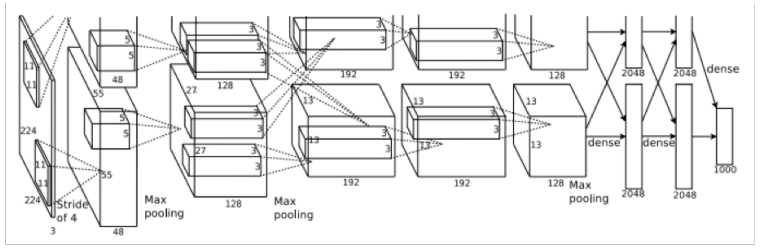


  - AlexNet 2012, ImageNet, GTX 580 x 2



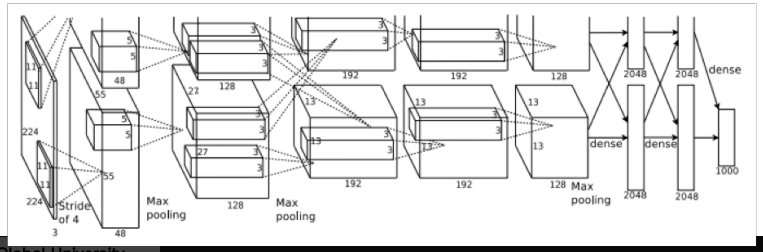
Convolutional Neural Networks (CNN)

- Most CNN layers input/output 3-5D tensors
  - 3D: channels, rows, columns
  - 4D: channels, rows, columns **+ batch**
  - 5D: channels, rows, columns **+ batch + time**
    - Recurrent convolutional layers



Convolutional Neural Networks (CNN)

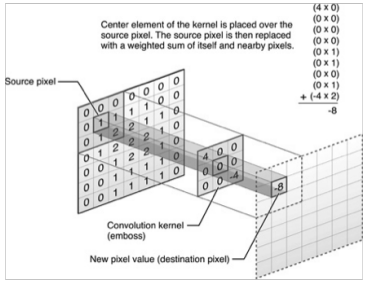
- Composed of heterogeneous layers
  - **Convolution** (standard, transposed, dilated, separable)
  - **Pooling** (max, GAP, average, SPP, …)
  - **Fully-connected**
  - Batch/instance/layer/group normalization, dropout
  - Skip connections (Highway, ResNet, DenseNet, DPN)
  - ROI pooling, RPN, etc.

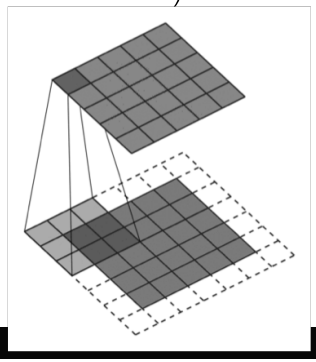


Convolution Layers

- Convolution layers extract **position-invariant local features** by convolution operation

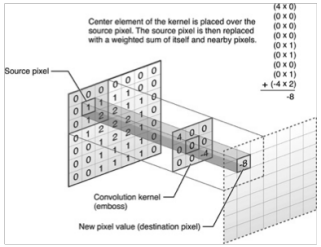
$$X^n_{(p,i,j)} = f\left(\sum_{q \in C^n_p} \sum_{0 \le u,v \le M_n - 1} w^n_{(q,p,u,v)} X^{n-1}_{(q,iS_n+u,jS_n+v)} + \theta^n_p\right)$$

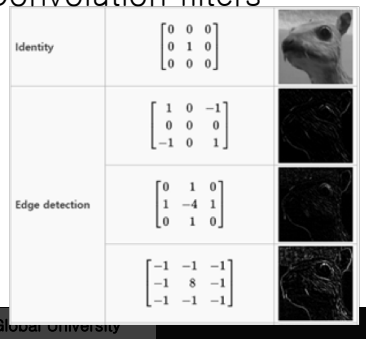




Convolution Operation

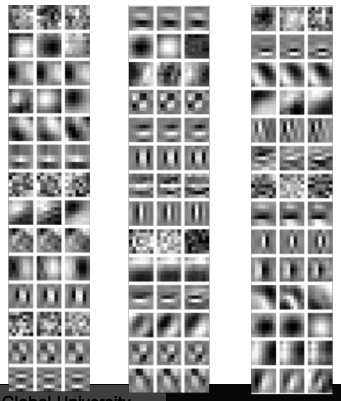
- Convolution operation
  - Weighed sum of local region and filters
- Convolution filters



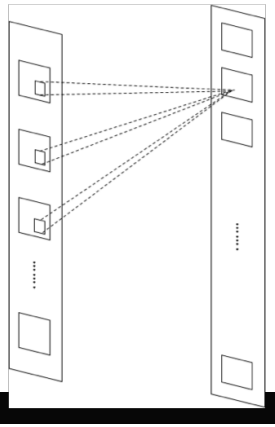


Convolution Layers

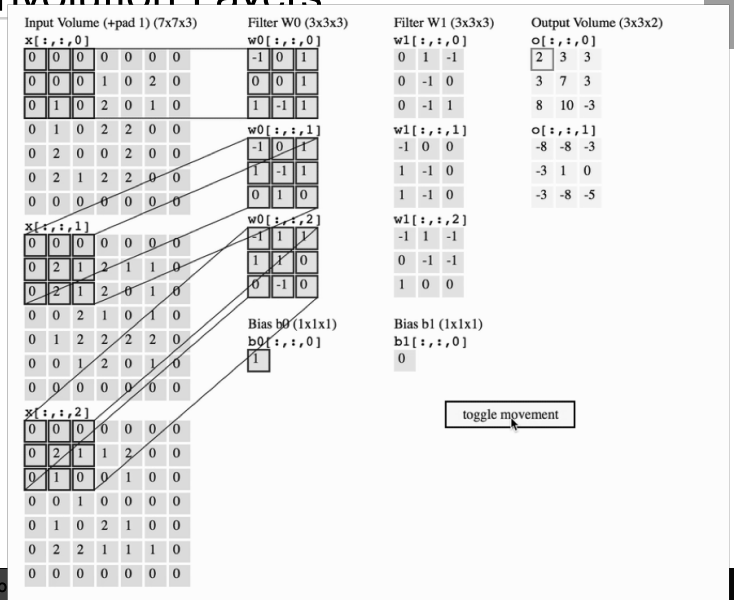
- Learning filters



- Multi-channel convolution



Convolution Layers



Convolution Layers

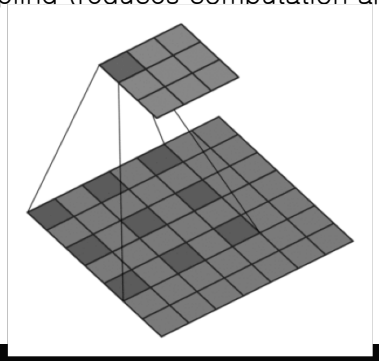
- Propagation

$$X^n_{(p,i,j)} = f\left(\sum_{q\in C^n_p}\sum_{0\le u,v\le M_n-1} w^n_{(q,p,u,v)} X^{n-1}_{(q,iS_n+u,jS_n+v)} + \theta^n_p\right)$$

  - q: input plane, p: output plane, M_n: mask width/height
  - C^n_p: # of input planes connected to p^{th} output plane
  - $w^n_{(q,p,u,v)}$: weight at (u,v) on the mask from q^{th} plane to p^{th} plane
  - $X^n_{(p,i,j)}$: feature at (i,j) on p^{th} plane of layer n
  - S_n: stride (horizontal/vertical distance between adjacent windows)
  - θ^n_p: bias

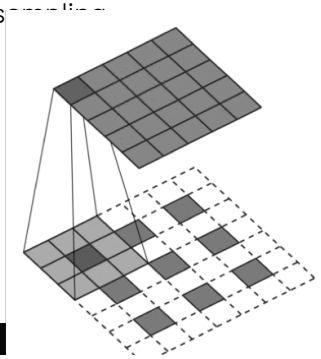
Dilated Convolution

- A.k.a. "astrous convolution"
- Dilated convolutions "inflate" the kernel by inserting spaces between the kernel elements
  - Filter upsampling (reduces computation and parameters)



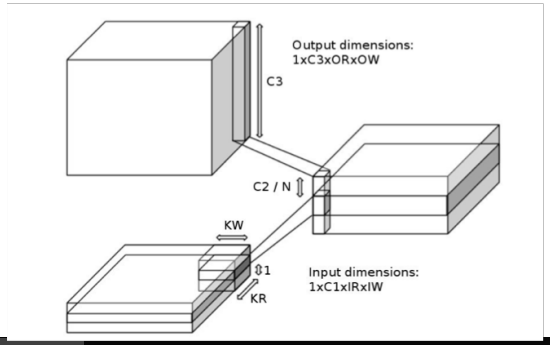
Transposed Convolution

- A.k.a. "fractionally-strided convolution" or "deconvolution(?)"
- Transposed convolutions work by swapping the forward and backward passes of convolution.
  - Feature ups



Separable Convolutions

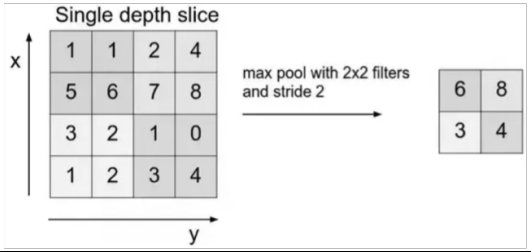
- Separable convolutions consists of two consecutive convolution operations.
  - Depth-wise convolution + point-wise convolution
  - Reduce computation and parameters

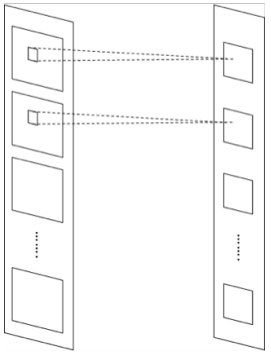


Max-Pooling Layers

- Max-pooling layers
  - Reduces feature dim.
  - Reduces positional variation

$$X_{(p,i,j)}^{n} = f\left(\max_{0 \le u,v \le M_n - 1} X_{(p,iS_n+u,jS_n+v)}^{n-1}\right)$$

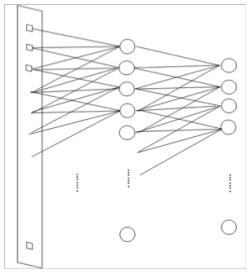




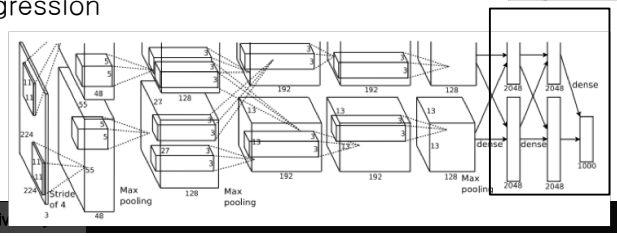
Fully-connected Layers

- Propagation

$$X_p^n = f\left(\sum_q w_{(q,p)}^n X_q^{n-1} + \theta_p^n\right)$$



➔ Perform target task with the feature extracted by previous layers
  - Classification
  - Regression

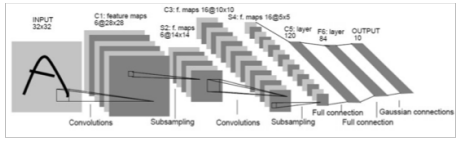


CNN in PyTorch

- Defining a CNN

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

Names	Layer Types	Hyper-parameters
conv1	convolution	3 -> 6, mask = 5x5
pool	max-pooling	window = 2x2, stride = 2
conv2	convolution	6 -> 16, mask = 5x5
pool	max-pooling	window = 2x2, stride = 2
fc1	fully-connected	400 (16 * 5 * 5) -> 120
fc2	fully-connected	120 -> 84
fc3	fully-connected	84 -> 10



Why CNN Works Well?

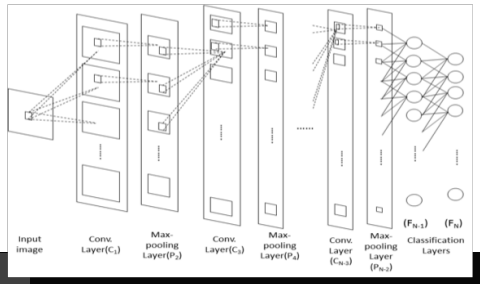
- Effective in learning high-level representation
- Good at catching 2D structures
  - Convolution layers are effective in learning 2D features
  - Tolerates shape variation by pooling and abstraction
- Network structure that less suffers from vanishing gradient problem and overfitting
  - Parameter sharing, sparse connection
- Flexible structure
- Easy to parallelize
- Recent issues: limited context compared to RNN
  - Remedy: non-local neural nets

Training of CNN

CNN Learning

- Given a training sample X^0 and its label c,
  - Desired output $D = \{d_i\ 's\}$, $d_i = 1$ if $i = c$, otherwise, $d_i = 0$
- Define an error as a function of connection weights W.
- Find connection weights W^* usually by gradient-based algorithm

$$W^* = \underset{W}{arg\min}\, E(W)$$



Loss Functions

- Given
  - X_c^N: the output of the top level layer for the c^{th} class
  - $D = (d_1, d_2, \cdots, d_C)$: desired output
- Mean square error

$$E_{MSE} = \frac{1}{2}\frac{\sum_c (X_c^N - d_c)^2}{C}$$

- Cross entropy (with softmax activation)
  - Softmax activation: $X_c^N = \frac{\exp(net_c^N)}{\sum_c \exp(net_c^N)}$

$$E_{CE} = -\sum_c d_c \log(X_c^N)$$

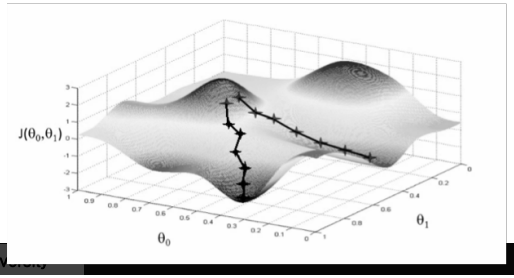
Gradient-based Learning

- Given current weights W, the gradient gives a direction in which increases the error most rapidly

$$\frac{\partial E}{\partial W} = \left(\frac{\partial E}{\partial W_1}, \frac{\partial E}{\partial W_2}, \cdots, \frac{\partial E}{\partial W_M}\right)$$

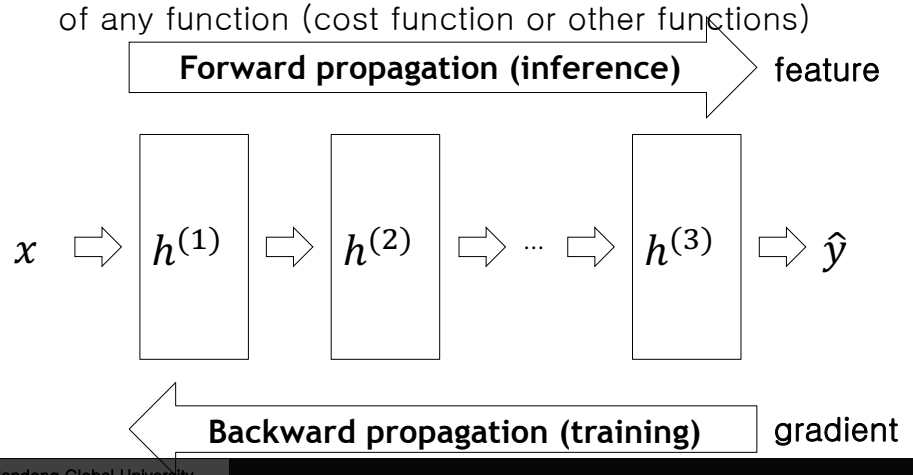
  - Weight update in gradient descent

$$W^{t+1} = W^t - \eta \frac{\partial E}{\partial W^t}$$



Back-Propagation

- Back-propagation: a method for computing gradient of any function (cost function or other functions)



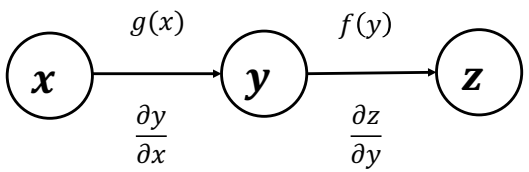
Chain Rule

- For real numbers x, y, and z

$$y = g(x),\ z = f(y) = f\big(g(x)\big)$$

  - Chain rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$



Chain Rule

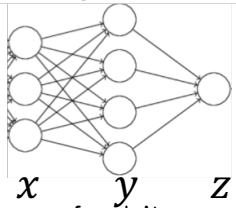
- For vectors $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$
  - Chain rule

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z$$

gradient vectors

nxm Jacobian matrix



- For tensors of arbitrary dim ?

Gradient and Jacobian

- **Gradient vector**: a multi-variable generalization of the derivative. (f is a scalar-valued function)

$$\frac{\partial f}{\partial x} = \nabla_x f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

- **Jacobian matrix**: matrix of all 1st order partial derivatives of a vector-valued function

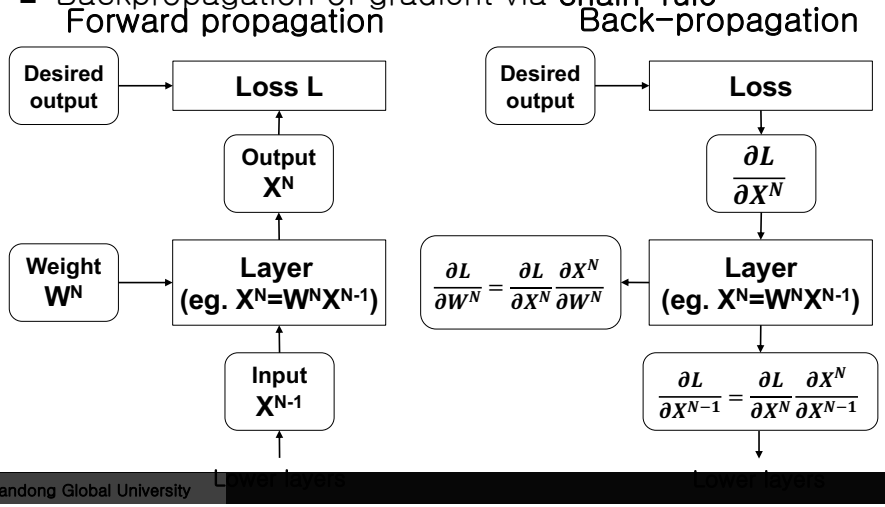
$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Matrix Form

- Propagation
  - $y_j = f(\sum_i w_{ij} x_i + \theta_j) = f(net_j)$
  - $net_j = \sum_i w_{ij} x_i + \theta_j$
- Matrix form
  - $X = (x_1, x_2, \ldots, x_N)^T, Y = (y_1, y_2, \ldots, y_M)^T, NET = (net_1, net_2, \ldots, net_M)^T$
  - $W = \begin{pmatrix} w_{11} & \cdots & w_{N1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{NM} \end{pmatrix}, \Theta = (\theta_1, \theta_2, \ldots, \theta_M)^T$
  - $NET = WX + \Theta = \begin{pmatrix} w_{11} & \cdots & w_{N1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{NM} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_N \end{pmatrix} + \begin{pmatrix} \theta_1 \\ \theta_2 \\ \cdots \\ \theta_M \end{pmatrix} = \begin{pmatrix} \sum_i w_{i1} x_i + \theta_1 \\ \sum_i w_{i2} x_i + \theta_2 \\ \cdots \\ \sum_i w_{iM} x_i + \theta_M \end{pmatrix}$
  - $Y = F(WX + \Theta)$

Back-Propagation

- Backpropagation of gradient via **chain-rule**

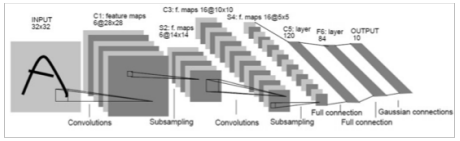


CNN in PyTorch

- Defining a CNN

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

Names	Layer Types	Hyper-parameters
conv1	convolution	3 -> 6, mask = 5x5
pool	max-pooling	window = 2x2, stride = 2
conv2	convolution	6 -> 16, mask = 5x5
pool	max-pooling	window = 2x2, stride = 2
fc1	fully-connected	400 (16 * 5 * 5) -> 120
fc2	fully-connected	120 -> 84
fc3	fully-connected	84 -> 10



CNN in PyTorch

- Defining Loss function and Optimizer

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

  - Cross entropy (with softmax activation)
    - Softmax activation: $X_c^N = \frac{\exp(net_c^N)}{\sum_c \exp(net_c^N)}$

$$E_{CE} = -\sum_c d_c \log(X_c^N)$$

  - Stochastic gradient descent

$$W^{t+1} = W^t + \Delta W^t$$

$$\Delta W^t = -\eta \frac{\partial Loss}{\partial W^t} + m\Delta W^{t-1}$$

CNN in PyTorch

- Training

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)               # feed input to network
        loss = criterion(outputs, labels)   # compute loss function
        loss.backward()                     # compute gradients
        optimizer.step()                    # update weights

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:                # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
```


Optimization Algorithms

Optimization Algorithms

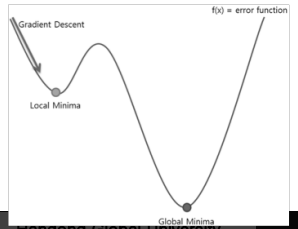
- Stochastic gradient descent (without momentum)

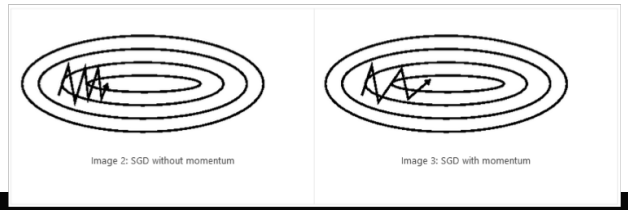
$$\theta_{t+1} = \theta_t + -\eta \nabla_\theta J(\theta_t)$$

- Stochastic gradient descent (with momentum)

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

$$\Delta\theta_t = -\eta \nabla_\theta J(\theta_t) + \gamma \Delta\theta_{t-1}$$





Optimization Algorithms

- AdaGrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

- AdaDelta

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

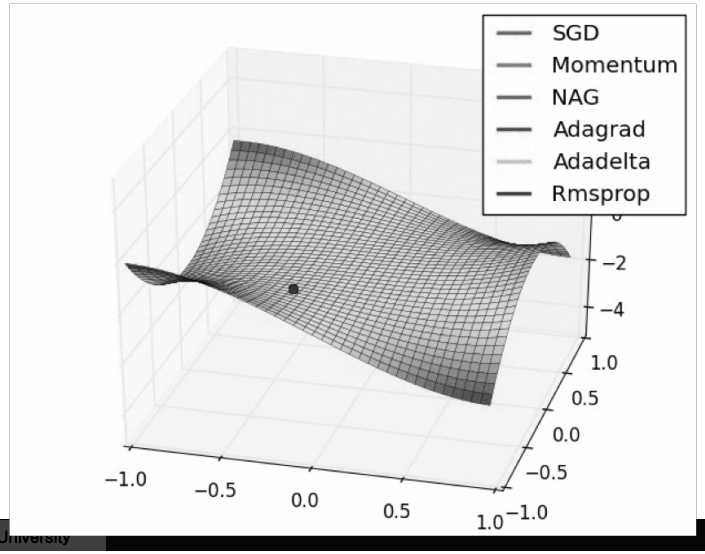
$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

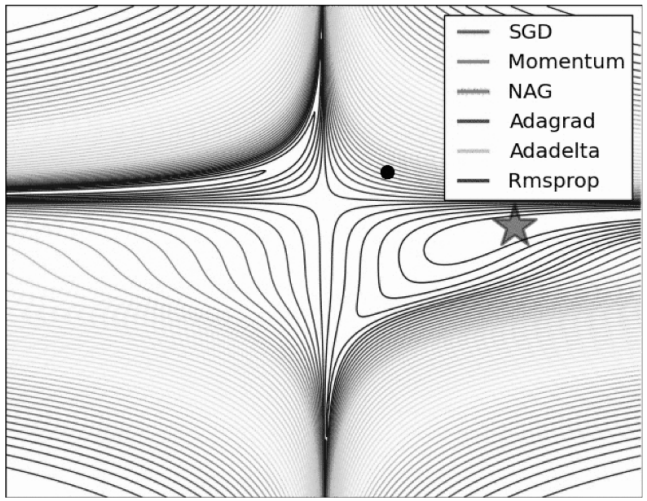
$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Optimization Algorithms



Optimization Algorithms



Optimization Algorithms

- RMSProp

$$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

  - Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

- ADAM

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

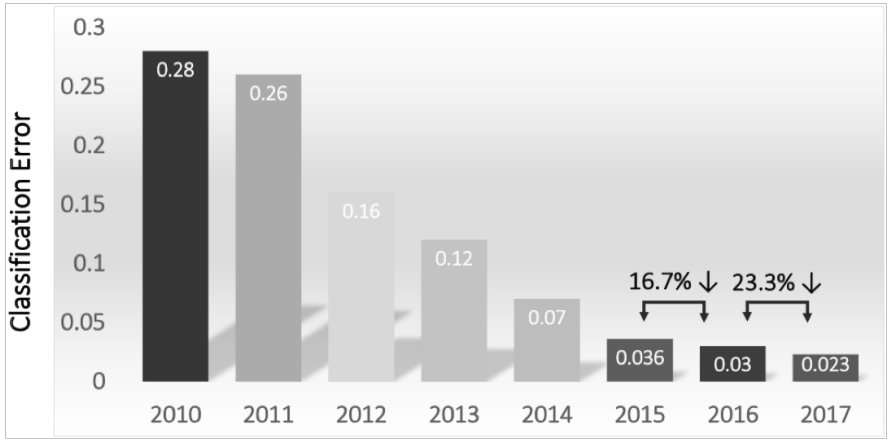
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

The authors propose default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ.

Advanced CNN Models

ILSVRC (ImageNet) Results

- 2017 SOTA error rate: 2.3%



Advances of CNN

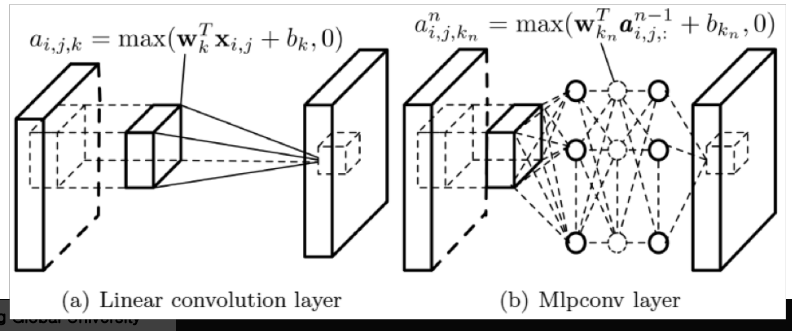
- Improved structures
  - Max-out network[Goodfellow13]
  - Network-in-network[Lin13]
  - Spatial pyramid pooling CNN [He14]
  - Very deep CNN [Simonyan15]
  - GoogLeNet[Szegedy14], Inception v2,v3,v4, Inception-ResNet
  - Residual learning networks [He15]
  - Dense convolutional networks [Huang16]
  - **SENet[Hu17], Dual-Path Net[Chen17]**
- Improved learning algorithms
  - **Batch normalization [Ioffe15]**, layer/weight/group normalization
  - Xavier init[Xavier10], **He init[He15]**, LL-init [Balduzzi17]
- Detection & classification
  - R-CNN[Girshik14], Fast R-CNN[Girshik14], **Faster R-CNN[Ren15]**
  - **Mask R-CNN[He17]**
  - Visual attention models [Mnih14, Ba15, Sermanet15]
  - **YOLO[Redmon16], SSD[Liu16]**
  - **FCN[Long16], DeepLab1-3[Chen15-17], PSP-Net[Zhao17]**

Advances of CNN

- Visualization and understanding
  - Visualization using deconvolution layers [Zeiler13]
  - Class saliency maps [Simonyan13]
  - Inverting CNN [Mahendran14]
- Building lightweight networks
  - Network compression [Bucilu06]
  - Knowledge distillation [Hinton14]
  - FitNet [Romero14]
  - SqueezNet [Iandola16]
  - ShuffleNet [Zhang17]
  - MobileNet, MobileNet.v2

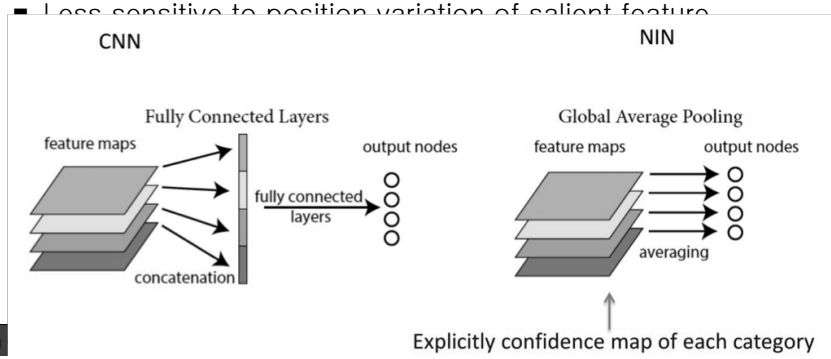
MLPconv Layer

- Lin, et al. “Network In Network”, 2014
- Layer to learn non-linear filters
  - Equivalent to (conv. + CCCP + CCCP)
    - CCCP: 1x1 conv.
    - CCCP is also used to reduce # of feature maps



Global Average Pooling

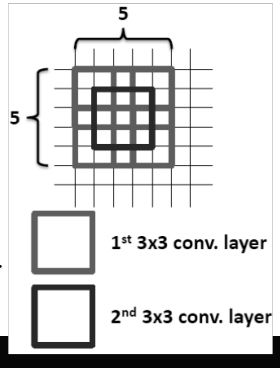
- Lin, et al. “Network In Network”, 2014
- Less suffers from overfitting than fully-connected layer
  - Usually, preceded by CCCP
  - Less sensitive to position variation of salient feature



VGG Net

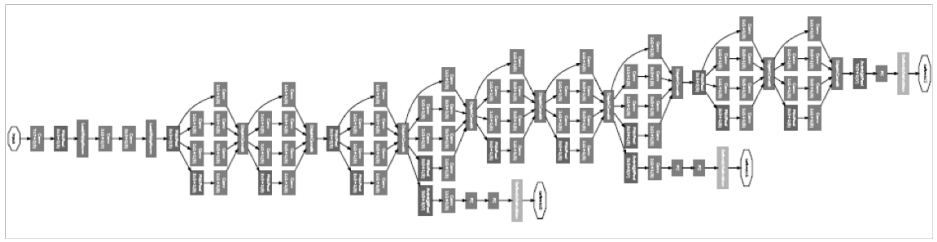
- Simonyan and Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition” 2015
  - Stack convolution layers have large receptive field
    - Two 3x3 layers – 5x5 receptive field
    - Three 3x3 layers – 7x7 receptive field
  - More nonlinearity
  - Less parameters

➔ Multiple 3x3 conv layers are better than single conv layer with a large filter



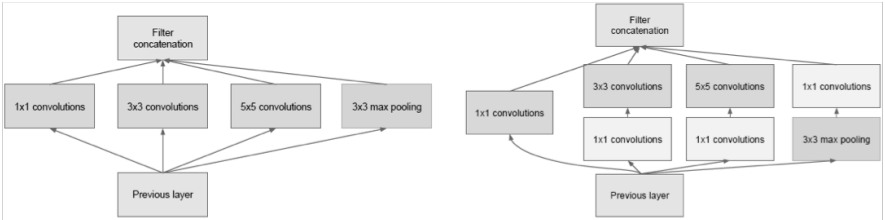
GoogLeNet

- Szegedy, et al. “Going deeper with convolutions”, 2015
  - Very deep network with 22 layers
  - Inception module



GoogLeNet

- Inception module
  - Multi-scale convolution
  - Dimensionality reduction by 1x1 convolution



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

Residual Learning

- He, et.al, "Deep Residual Learning for Image Recognition", 2015
  - Target function $H(x) = F(x) + x$
  - Residual Function $F(x) = H(x) - x$
  - Deep residual network contains 152 layers

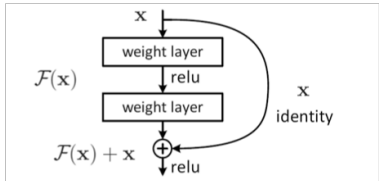
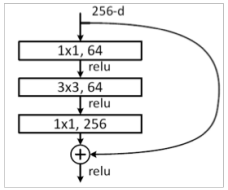


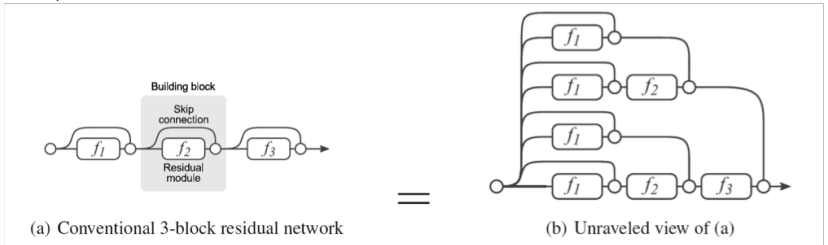
Figure 2. Residual learning: a building block.



Bottleneck building block

Residual Learning

- Veit, et.al., "Residual Networks Behave Like Ensembles of Relatively Shallow Networks", 2016
  - During training, gradients are mainly from relatively shallow paths

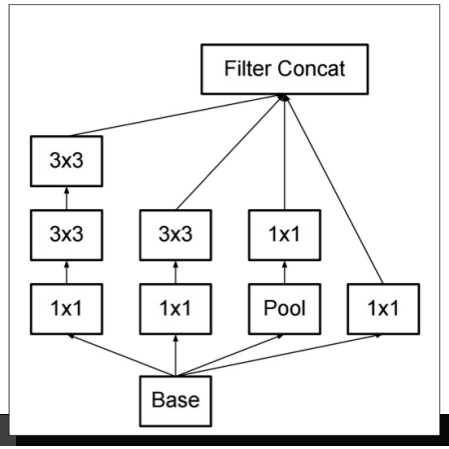


(a) Conventional 3-block residual network (b) Unraveled view of (a)

Figure 1: Residual Networks are conventionally shown as (a), which is a natural representation of Equation (1). When we expand this formulation to Equation (6), we obtain an *unraveled view* of a 3-block residual network (b). Circular nodes represent additions. From this view, it is apparent that residual networks have $O(2^n)$ implicit paths connecting input and output and that adding a block doubles the number of paths.

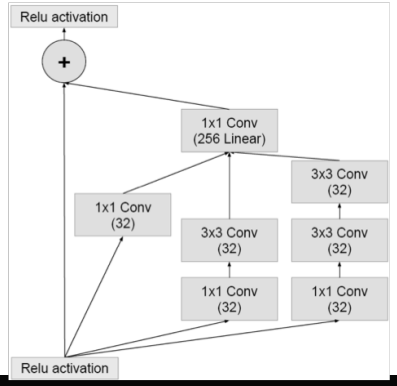
Inception v3

- Szegedy, et.al., "Rethinking the Inception Architecture for Computer Vision", 2015



Inception-ResNet

- Szegedy, et.al., "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning", 2016

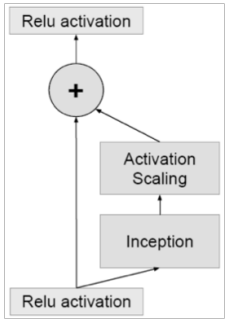


Inception-ResNet: Activation Scaling

- If # of filters exceeded 1000,
  - Residual variants started to exhibit instabilities and the network has just "died" early in the training

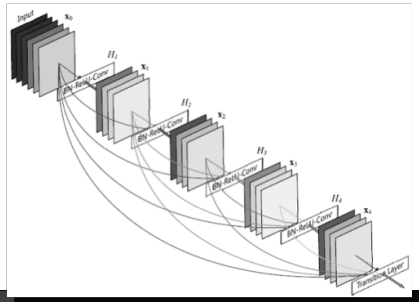
  ➔ Last layer before the average pooling started to produce only zeros after a few tens of thousands of iterations.

- Scaling down the residuals (0.1~0.3) before adding them to the previous layer activation seemed to stabilize the training



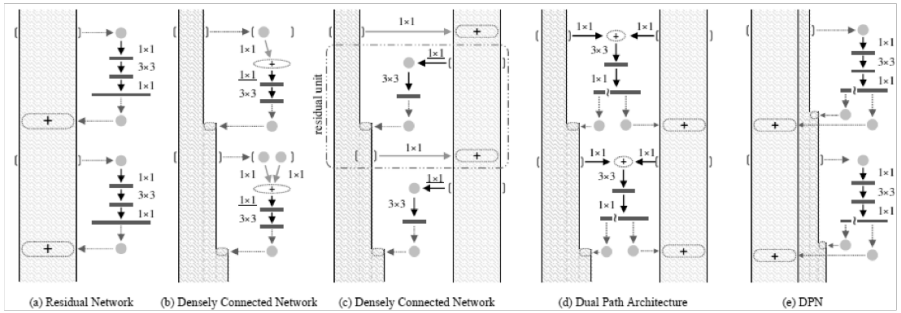
Dense Net

- G.Huang, et.al., "Densely Connected Convolutional Networks," 2016.
- All layers has connection from **all preceding layers**
  - **Short gradient path**
  - **Does not learn redundant features**



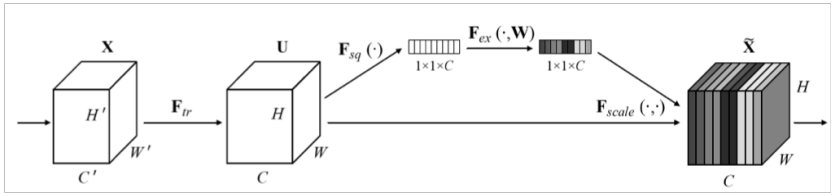
Dual-Path Networks

- Y. Chen, et al., "Dual-Path Networks," Aug. 2017
  - ResNet + DenseNet
    - ResNet: feature re-use
    - DenseNet: explore new feature



SENet

- Hu, et al., "Squeeze-and-Excitation Networks," 2018
  - Recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels
  - "Attention among channels"



Weight Initialization

- Weight initialization to prevent vanishing/exploding gradient
  - Xavier initialization
    - Sample from Gaussian with $\mu = 0, \sigma^2 = \frac{2}{N_{in}+N_{out}}$
  - He initialization
    - Sample from Gaussian with $\mu = 0, \sigma^2 = \frac{2}{N_{in}}$ (or $\sigma^2 = \frac{2}{N_{out}}$)
  - Recurrent weights (with ReLU)
    - Identity matrix or its scaled version
- For shallow networks, initialization by small random numbers (eg. [-0.01,0.01]) is sufficient

Bias Initialization

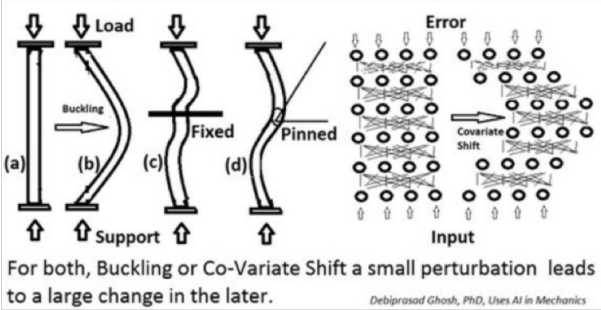
- Bias
  - Zero (the most common)
  - Small positive values (eg. 0.01)
    - To make ReLU units have positive values (?)
  - Large value (eg. 6)
    - Forget gate of LSTM
  - According to data distribution
    - Output layer

Batch Normalization

- Ioffe and Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", 2015
- One reason of slow learning: internal covariate shift
  - Change in the output distribution of the preceding layer makes learning the following layers difficult.
- Remedy: normalize output distribution of each layer on each batch

Internal Covariate Shift

- Internal covariate shift [Ioffe15]
  - Change in the output distribution of the preceding layer makes learning the following layers difficult.



Batch Normalization

- Normalization to remove internal covariate shift

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

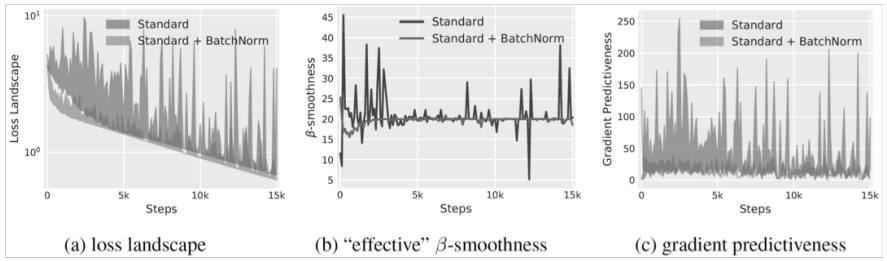
- Scale and shift to relax constraint

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

  - $\gamma^{(k)}$ and $\beta^{(k)}$ are trainable parameters

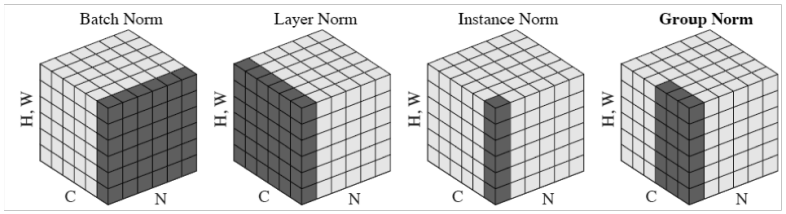
Batch Normalization

- Santurkar, et.al., "How Does Batch Normalization Help Optimization, " 2018
  - Controls internal covariate shift (?)
  - Makes the optimization landscape significantly smoother



Layer/Instance/Group Normalization

- Layer Normalization[Ba16]
- Instance Normalization[Ulyanov17]
- Group Normalization[Wu18]



Object Detection

Spatial Pyramid Pooling

- He, et al. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”, 2014

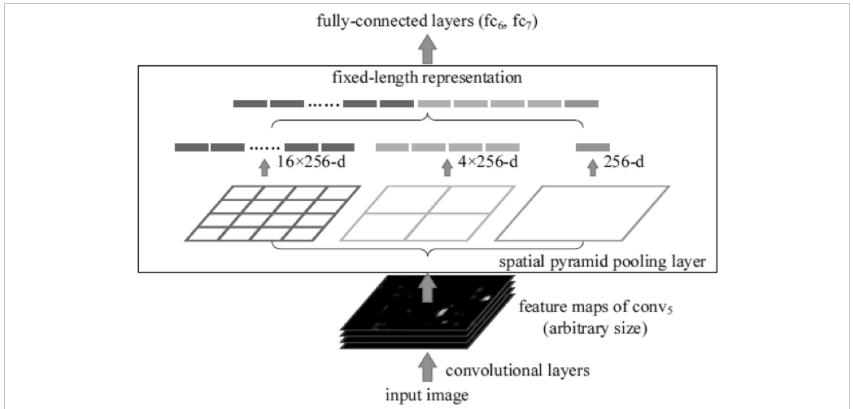


Figure 3. The network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the $conv_5$ layer.

Spatial Pyramid Pooling [He2014]

- Scale-invariant CNN
  - Apply convolution layers to the whole image.
  - Then, apply SPP layer to generate a fixed-length feature vector to feed into classification layers.

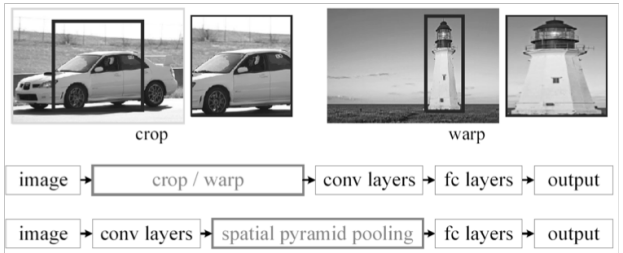


Figure 1. Top: cropping or warping to fit a fixed size. Middle: a conventional deep convolutional network structure. Bottom: our spatial pyramid pooling network structure.

R-CNN

- Girshick, et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, 2014
  - Classical vision technique + CNN

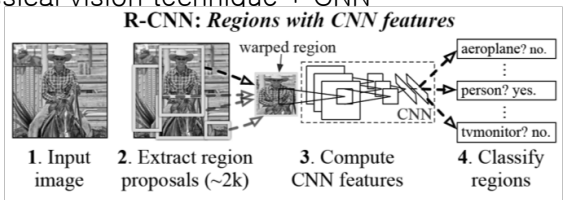


Figure 1: Object detection system overview. Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs. R-CNN achieves a mean average precision (mAP) of **53.7% on PASCAL VOC 2010**. For comparison, [34] reports 35.1% mAP using the same region proposals, but with a spatial pyramid and bag-of-visual-words approach. The popular deformable part models perform at 33.4%.

Fast R-CNN

- Girshick, “Fast R-CNN”, 2015
- Problems of R-CNN
  - Training is a multi-stage pipeline
  - Training is expensive in space and time
  - Object detection is slow
- Remedy
  - Share CNN feature
  - RoI pooling layer

Fast R-CNN

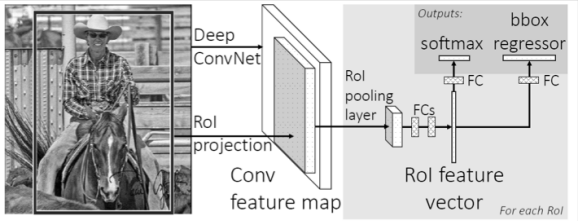


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

Faster R-CNN

- Ren, et.al., “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, 2015
  - RPN (region proposal network)

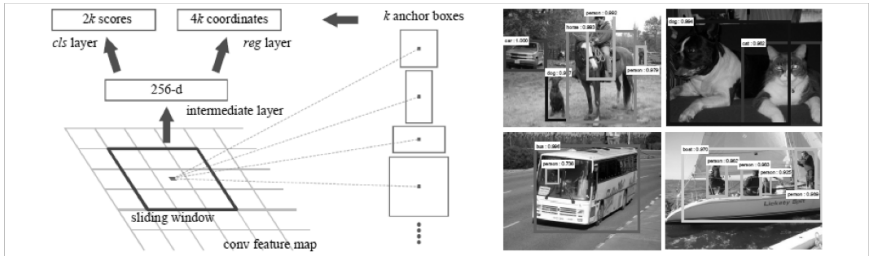
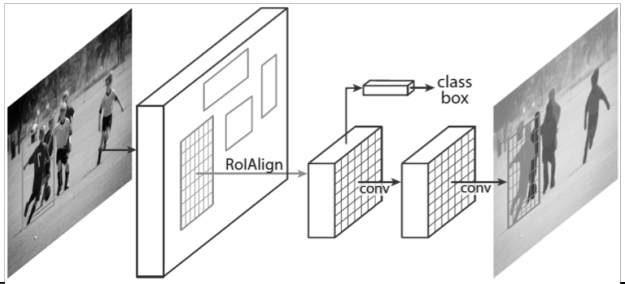


Figure 1: **Left**: Region Proposal Network (RPN). **Right**: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

Mask R-CNN

- He, et.al., “Mask R-CNN,” 2018
  - Faster R-CNN + object mask detection
    - RoI-Pooling (max pooling) → RoI-Align (linear interpolation)
    - 1st branch: bounding box + classification
    - 2nd branch: object mask

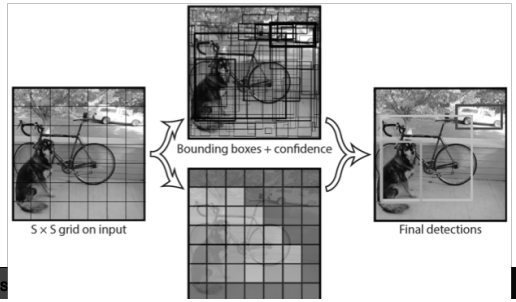


Mask R-CNN

- Instance segmentation

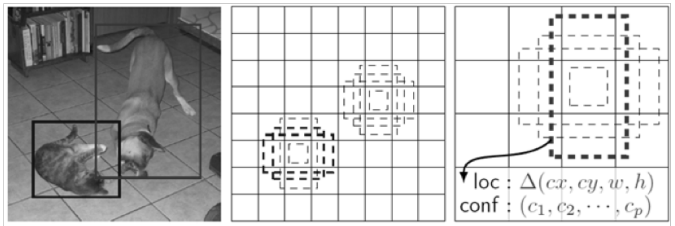
YOLO

- Redmon, et. al., "You Only Look Once: Unified, Real-Time Object Detection", 2016
  - Divide image into SxS grid
  - Extract B bounding boxes from each grid cell
  - (5 + C)-dim regression from each bounding boxes
    - x, y, w, h, Pr(object), Pr(class$_c$|object)



SSD

- Liu, et. al., "SSD: Single Shot MultiBox Detector", 2016.
  - Multi-scale feature maps for detection
  - Convolutional predictors for detection



SSD

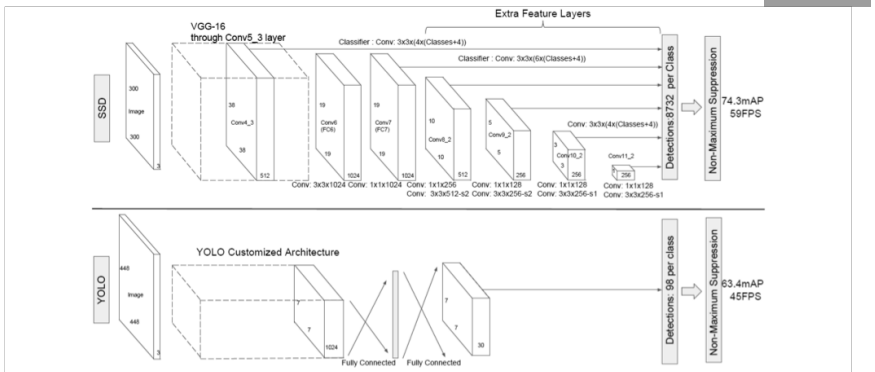
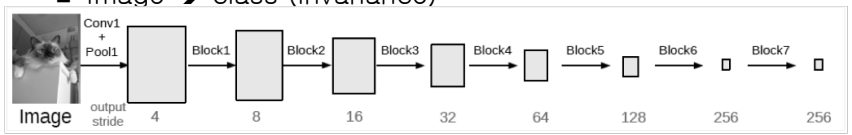


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300 × 300 input size significantly outperforms its 448 × 448 YOLO counterpart in accuracy on VOC2007 `test` while also improving the speed.

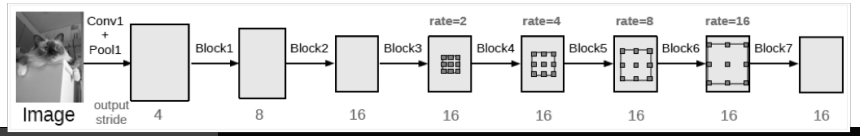
Semantic Segmentation

Semantic Segmentation

- Classification
  - Image ➔ class (invariance)

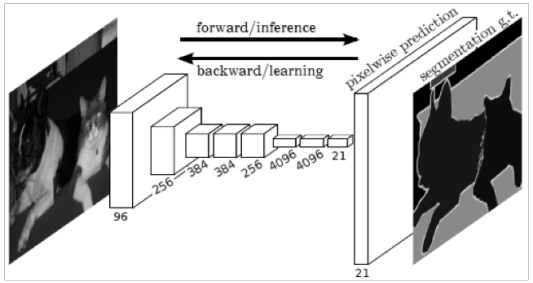


- Semantic segmentation
  - Image ➔ map (localization)
  - Pixel-wise dense labeling/prediction



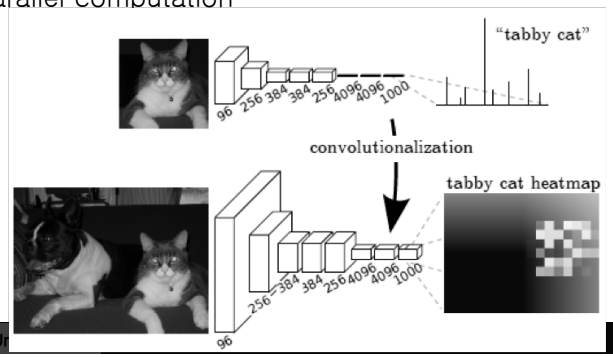
Fully Convolutional Networks [Long16]

- J. Long, et al., "Fully Convolutional Networks for Semantic Segmentation," May 2016.
  - Repurpose CNN for semantic segmentation



Fully Convolutional Networks [Long16]

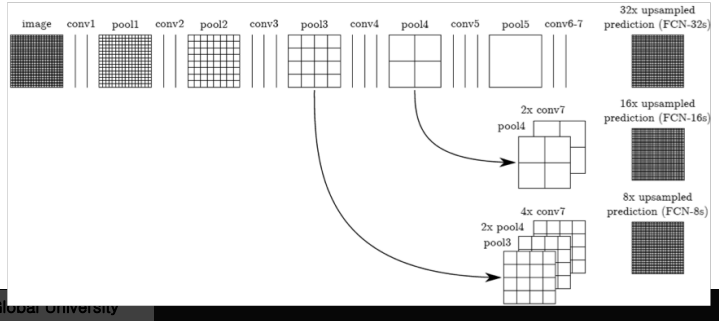
- Transforming fully connected layers into convolution layers
  - Use fully connected layers as convolutions with kernels that cover their entire input regions
  - Parallel computation



Fully Convolutional Networks [Long16]

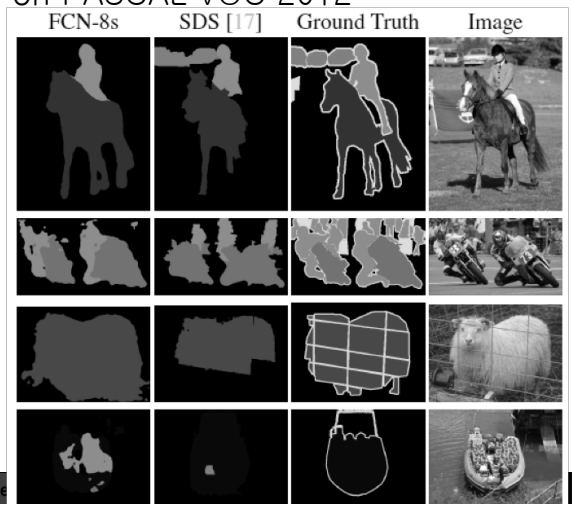
- Up-sampling techniques
  - Atrous algorithm (shift and stitch)
  - Backwards convolution (a.k.a. deconvolution)

- Combines low-level and high-level results



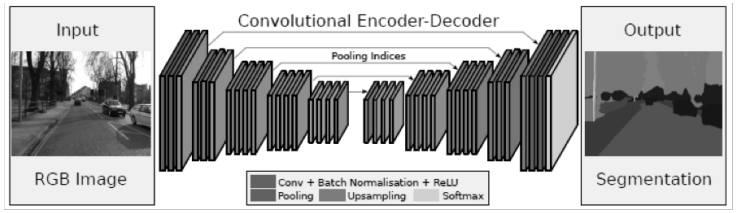
Fully Convolutional Networks [Long16]

- 62.2% on PASCAL VOC 2012



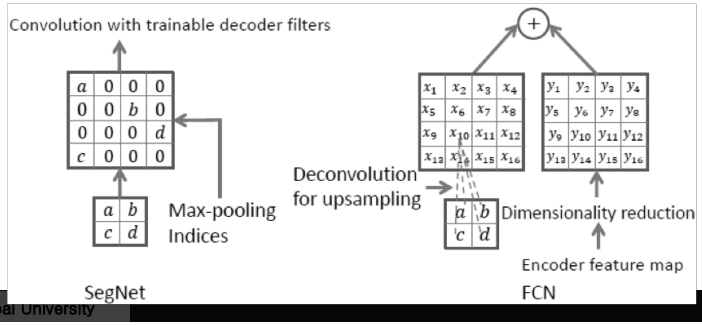
SegNet

- V. Badrinarayanan, et al., "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," Oct. 2016.
  - Convolutional encoder-decoder model
  - Upsampling using pooling indices



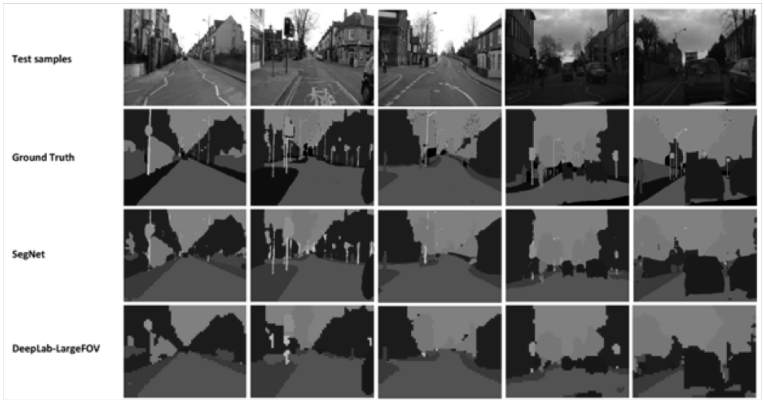
SegNet

- V. Badrinarayanan, et al., "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," 2016.
  - Convolutional encoder-decoder model
  - Upsampling using pooling indices



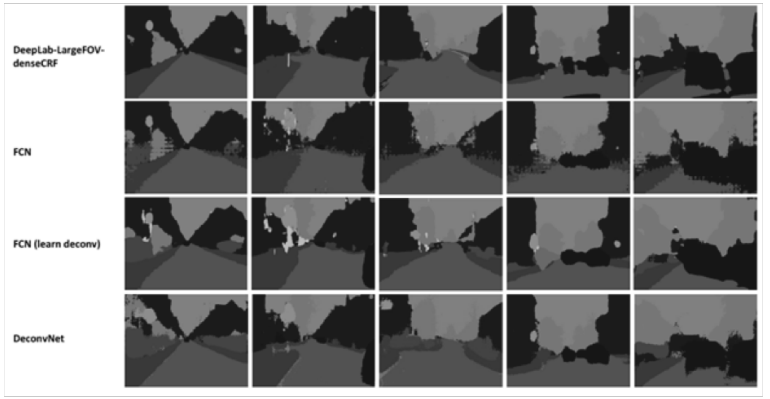
SegNet

- Applied to road images



SegNet

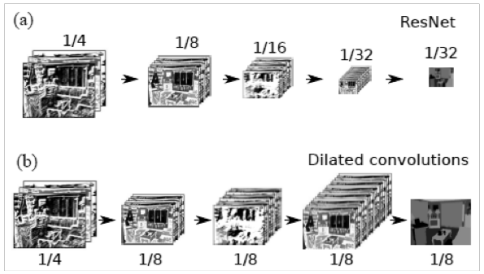
- Applied to road images

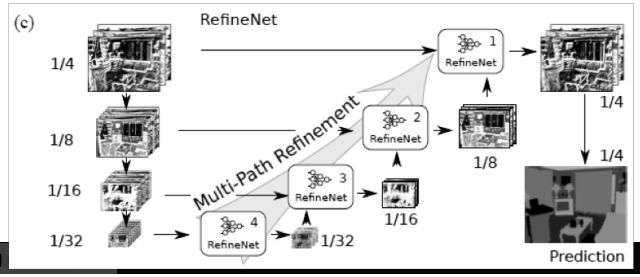


RefineNet

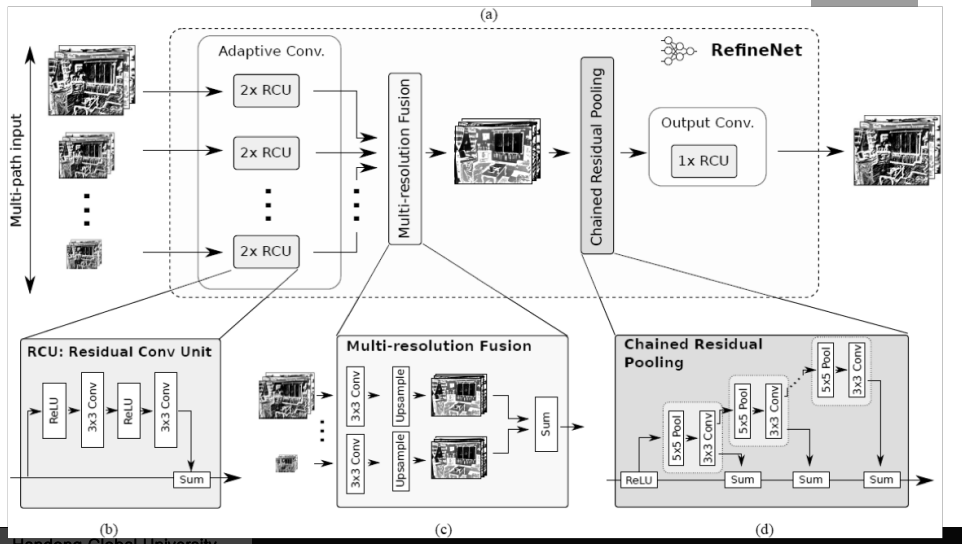
- Lin, et al., "RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation," Nov. 2016
  - Mult-level feature for high-resolution prediction
    - Refines low-resolution semantic features with fine-grained lo-level features by long-range residual connections
  - Chained residual pooling
    - Skip connection
  - 83.4% on PASCAL VOC 2012 dataset

RefineNet



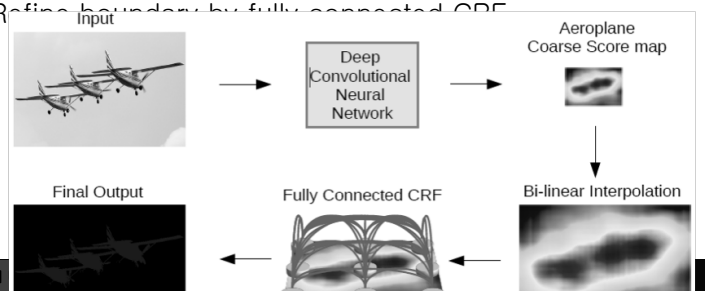


RefineNet



DeepLab

- Chen, et al., "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs," 2016.
  - Astrous convolution on last few layers
    - **Filter upsampling** instead of down-sampling
  - Upsampling by bilinear interpolation
  - Refine boundary by fully connected CRF



DeepLab

- Astrous convolution

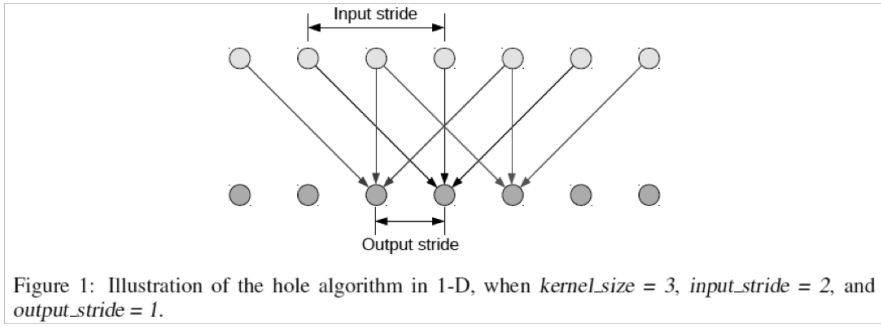
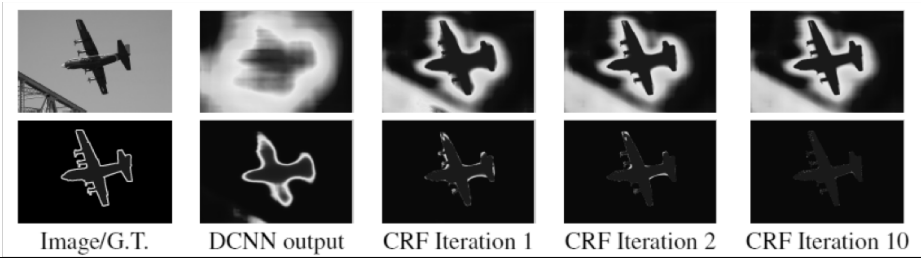


Figure 1: Illustration of the hole algorithm in 1-D, when *kernel_size* = 3, *input_stride* = 2, and *output_stride* = 1.

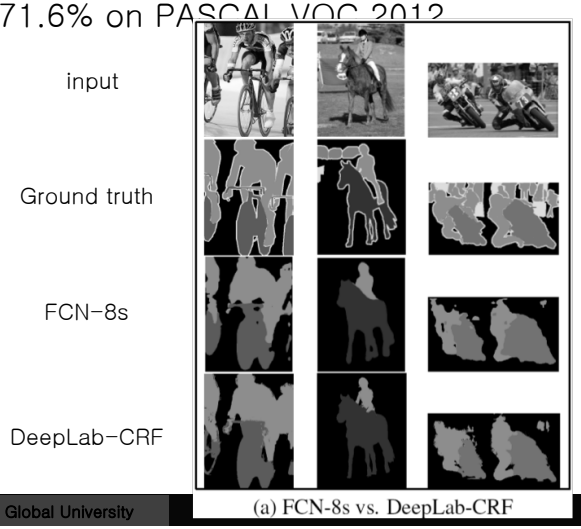
DeepLab

- Chen, et al., "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs," 2016.
  - Upsampling by astrous convolution (hole algorithm)
  - Refine boundary by fully connected CRF



DeepLab

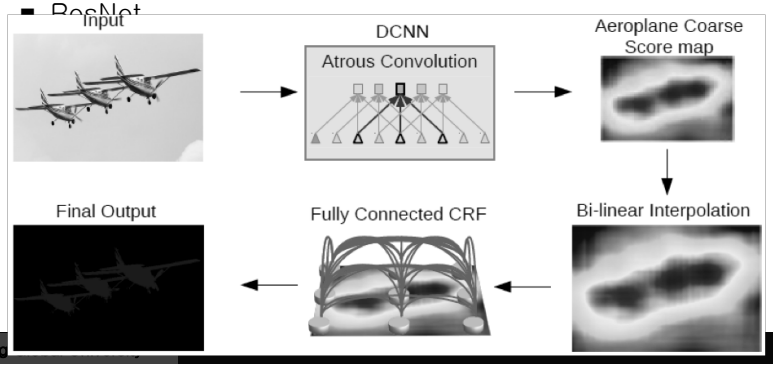
- 71.6% on PASCAL VOC 2012



(a) FCN-8s vs. DeepLab-CRF

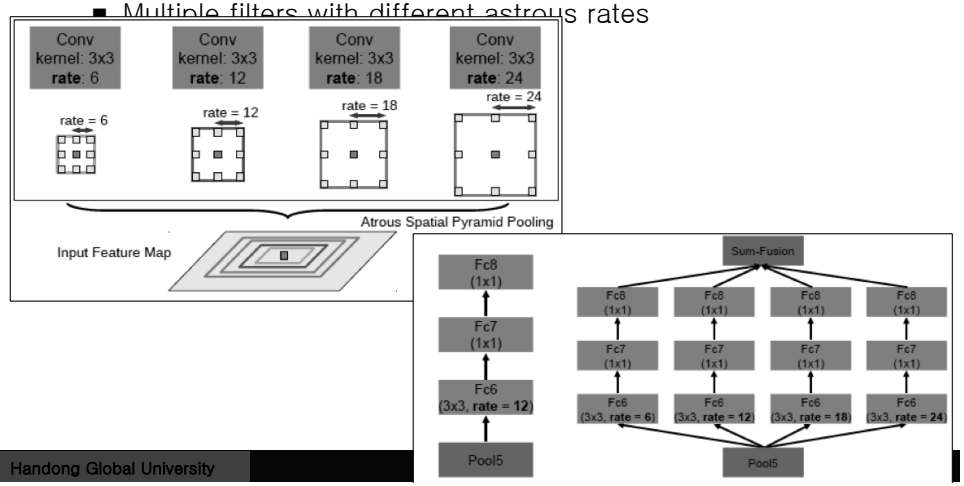
DeepLab2

- Chen, et al., "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," 2017
  - Astrous spatial pyramid pooling (ASPP)
  - ResNet



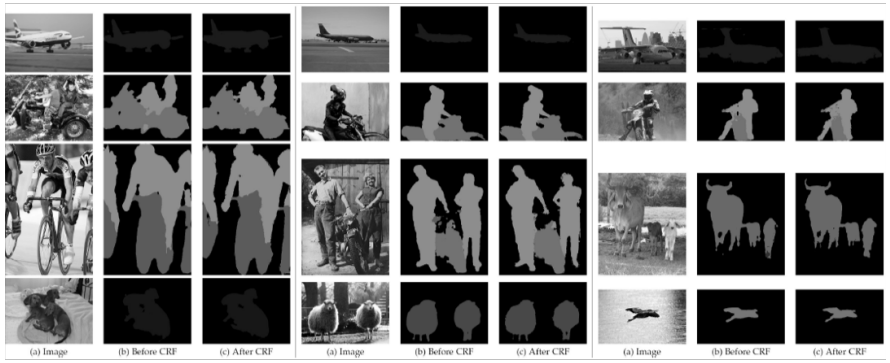
DeepLab2

- Astrous spatial pyramid pooling (ASPP)
  - Multiple filters with different astrous rates



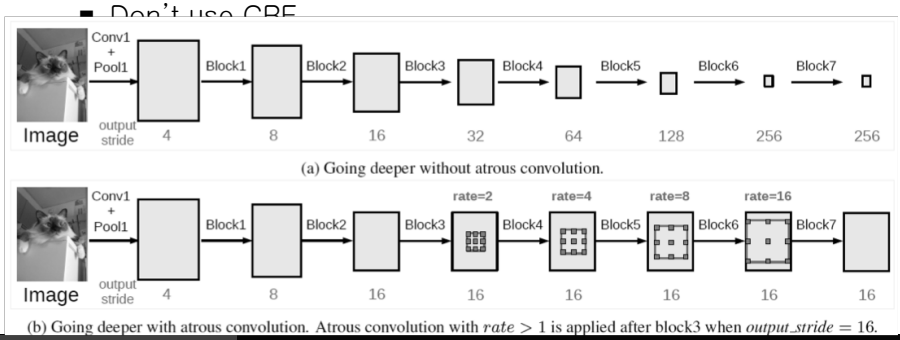
DeepLab2

- 79.7% on PASCAL VOC 2012



DeepLab3

- Chen, et al., “Rethinking Atrous Convolution for Semantic Image Segmentation,” 2017.
  - Revisiting astrous conv in cascade or in parallel
  - Based on ResNet, apply astrous conv. In block 4-7
  - Don’t use CRF



(a) Going deeper without atrous convolution.

(b) Going deeper with atrous convolution. Atrous convolution with $rate > 1$ is applied after block3 when $output\_stride = 16$.

DeepLab3

- Chen, et al., “Rethinking Atrous Convolution for Semantic Image Segmentation,” 2017.
  - Revisiting astrous convolution in cascade or in parallel
  - Don’t use CRF

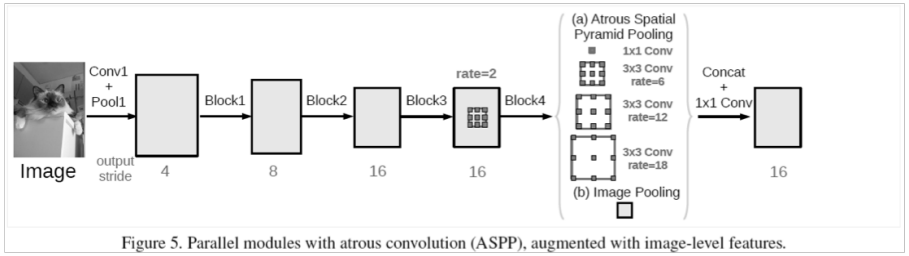


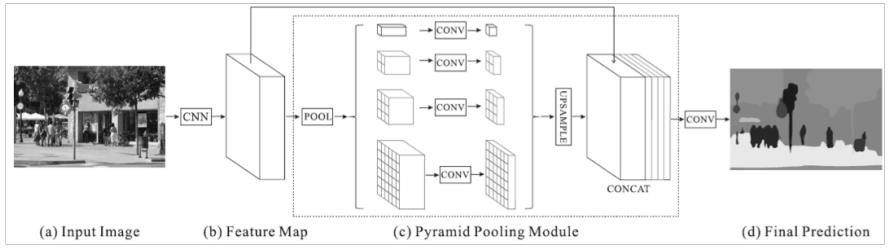
Figure 5. Parallel modules with atrous convolution (ASPP), augmented with image-level features.

DeepLab3

- 85.7% on PASCAL VOC 2012
  - 86.9% when trained by ImageNet + JFT-300M dataset

Pyramid Scene Parsing Network

- H. Zhao, et al., “Pyramid Scene Parsing Network,” 2017.
  - Context is important to resolve ambiguity
  - Pyramid pooling to catch larger context



(a) Input Image (b) Feature Map (c) Pyramid Pooling Module (d) Final Prediction

Pyramid Scene Parsing Network

- 85.4% on PASCAL VOC 2012

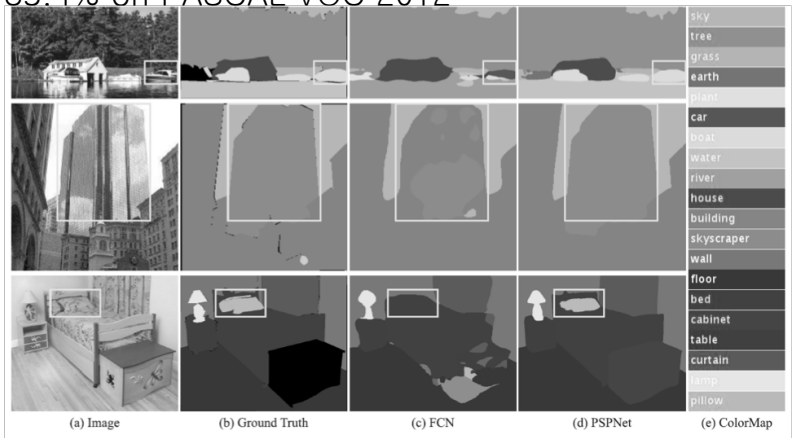
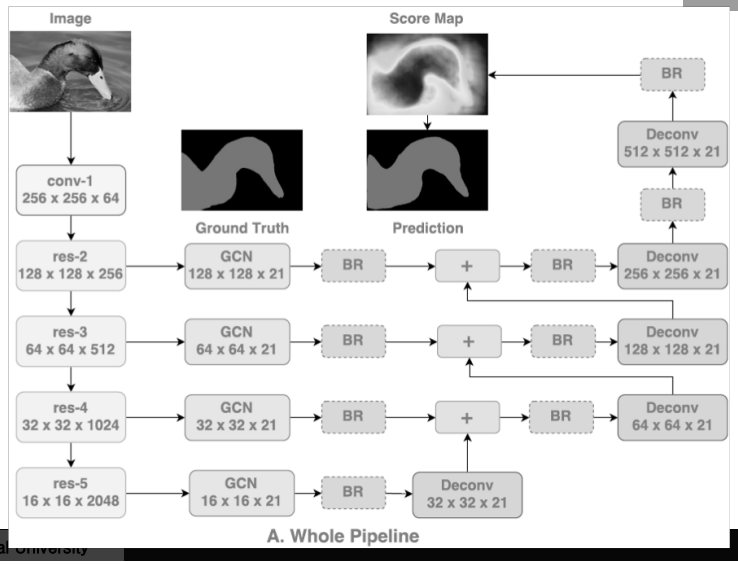


Figure 2. Scene parsing issues we observe on ADE20K [43] dataset. The first row shows the issue of mismatched relationship – cars are seldom over water than boats. The second row shows confusion categories where class "building" is easily confused as "skyscraper". The third row illustrates inconspicuous classes. In this example, the pillow is very similar to the bed sheet in terms of color and texture. These

Global Convolutional Network

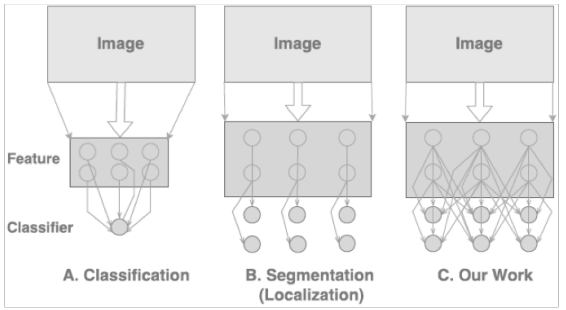
- C. Peng, et al., "Large Kernel Matters – Improve Semantic Segmentation by Global Convolutional Network," Mar. 2017.
  - Global convolutional network
    - Classification: stacking small filters (1x1 or 3x3)
    - Semantic segmentation: large kernels
  - Boundary refinement

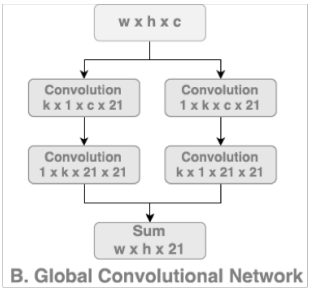
Global Convolutional Network



Global Convolutional Network

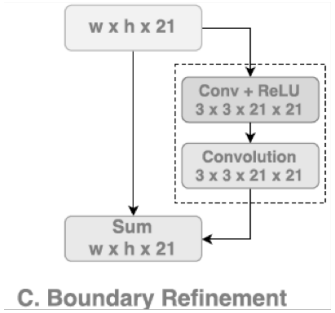
- Global convolutional network
  - Kernel size is the same as size of feature map
  - Symmetric, separable large filters





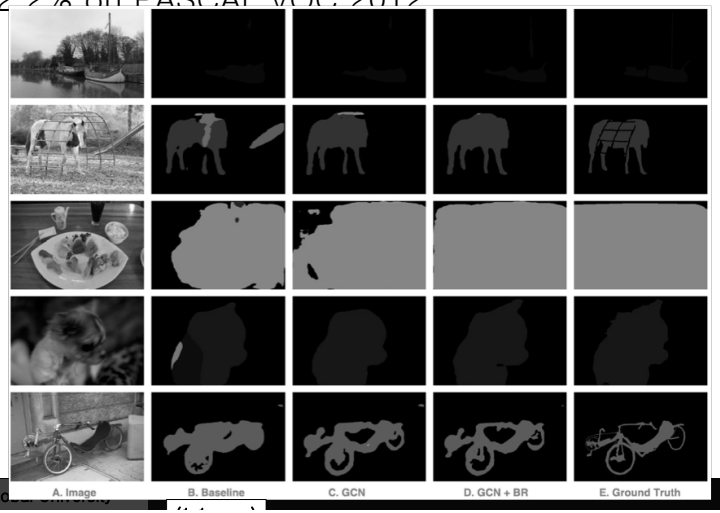
Global Convolutional Network

- Boundary refinement block
  - Residual structure



Global Convolutional Network

- 82.2% on PASCAL VOC 2012



Emerging Topics in Deep Learning Part 1

Injung Kim
Handong Global University
2019. 1. 8.

Agenda

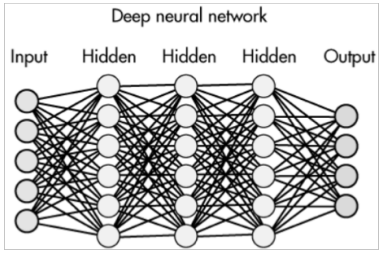
- One-shot Learning
- Meta Learning
- Continual Learning

One-Shot Learning

- One-shot learning aims to learn information about object categories from one, or only a few, training samples/images.
- Popular approach
  - Representation learning
    - Transfer learning (+ fine-tuning)
    - Metric learning (Siamese Net, triplet net)
  - Simple classifier
    - k-NN
    - Memory network

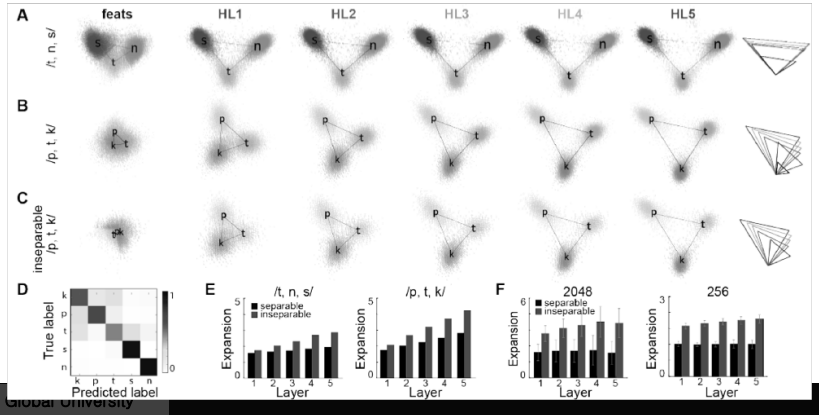
Representation Learning

- Hidden layers of neural networks learn **representation** for the task
  - Each layer learns representation that makes classification task easier



Representation Learning in MLP Layers

- Nagamine and Mesgarani, “Understanding the Representation and Computation of Multilayer Perceptrons: A Case Study in Speech Recognition,” 2017.



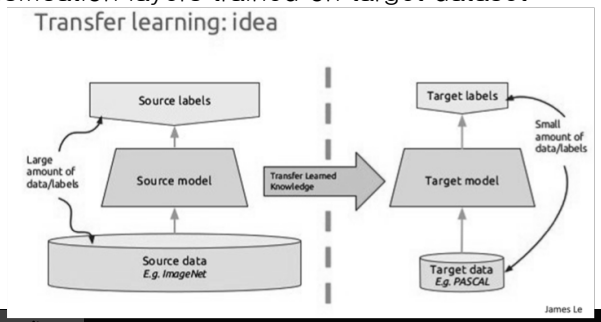
Representation Learning

- Learning shared representations
  - Useful to handle multiple domains or modalities
  - Useful to transfer learned knowledge to tasks for which few or no examples are given but a task representation exists
- Representation learning provides one way to perform unsupervised and semi-supervised learning
  - Very large amounts of unlabeled training data
    + relatively little labeled training data
- Most representation learning problems face a trade-off between
  - Preserving as much information about the input as possible
  - Attaining nice properties (eg. independence)

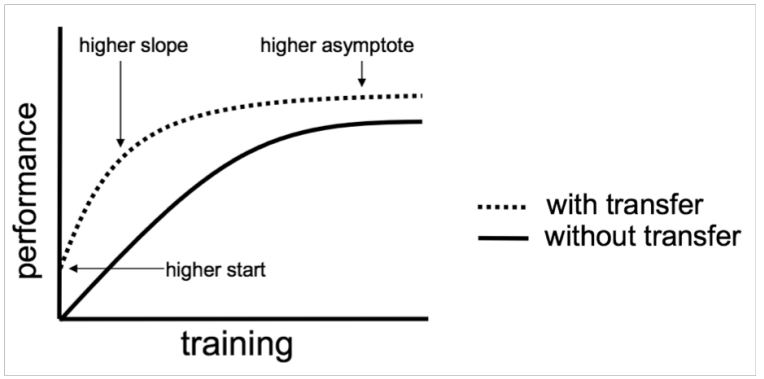
Transfer Learning

- Transfer learning applying knowledge gained while solving one problem to a different but related problem.

  Ex) Feature extraction layers pretrained on ImageNet dataset
  + classification layers trained on target dataset



Transfer Learning



Transfer Learning in NLP

- Natural language processing
  - ELMo (Embeddings from Language Models)
    - Peters, et al., "Deep contextualized word representations," Mar. 2018
  - GPT (Generative Pre-Training)
    - Radford, et al., "Improving Language Understanding by Generative Pre-Training," 2018.
  - BERT (Bidirectional Encoder Representations from Transformers)
    - Devlin, et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Oct. 2018

Transfer Learning in Speech Processing

- Jia, et al., "Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis," Oct. 2018

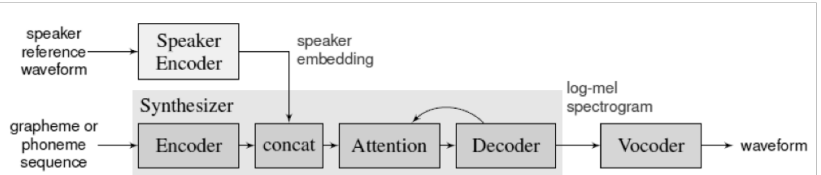
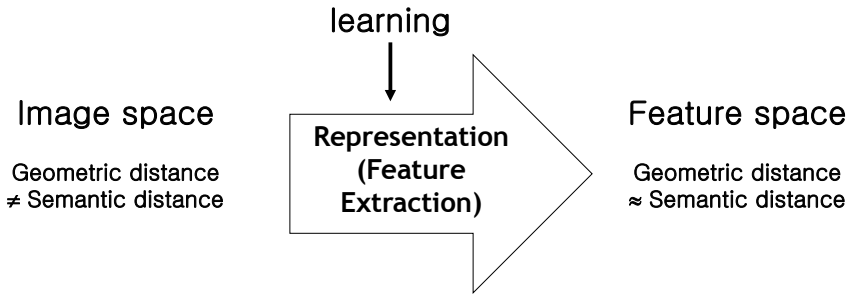


Figure 1: Model overview. Each of the three components are trained independently.

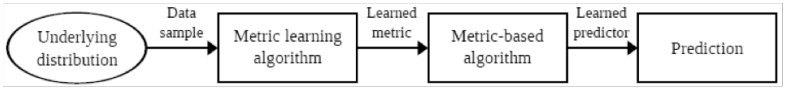
Metric Learning

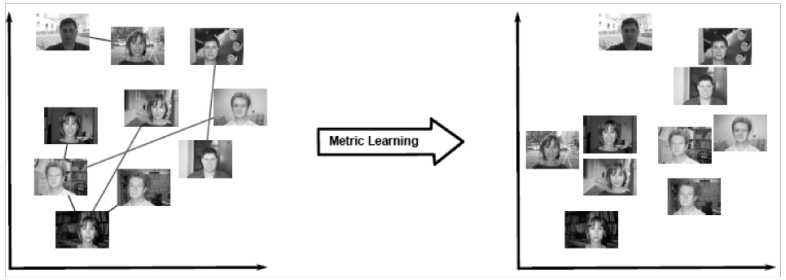
- Learn representation such that geometric distance approximate semantic distance in the resulting feature space



Metric Learning

- Bullet, et al., "A Survey on Metric Learning for Feature Vectors and Structured Data," 2014.



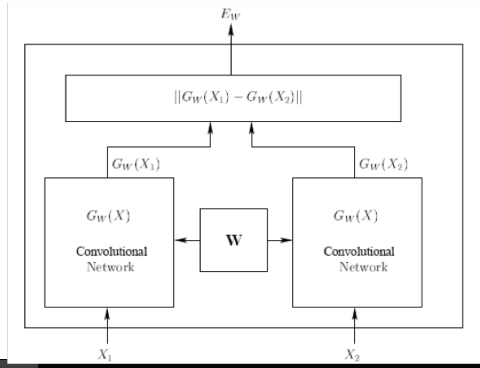


Siamese Network [Chopra2005]

- Developed for application where …
  - Large number of categories
  - Small number of samples per category
  - Only a subset of the categories is known at training time.
  - ➔ Convention methods are NOT appropriate
- Common approaches
  - Metric learning
    - Learn similarity/distance metric between the patterns
    - Recognize with a library of stored prototypes
  - Generative probabilistic methods
    - One category can be trained w/o examples from others

Siamese Network [Chopra2005]

- Learn similarity metric from data
  - Feature extraction by a CNN $G_W(X)$
  - Similarity metric $E_W(X_1, X_2) = \|G_W(X_1) - G_W(X_2)\|$



Siamese Network [Chopra2005]

- Training of Siamese Network
  - If X_1 and X_2 belong to the same class, adjust W to decrease $E_W(X_1, X_2)$
  - If X_1 and X_2 belong to different classes, adjust W to increase $E_W(X_1, X_2)$

- Training of Siamese Network
  - If X_1 and X_2 belong to the same class, adjust W to decrease $E_W(X_1, X_2)$
  - If X_1 and X_2 belong to different classes, adjust W to increase $E_W(X_1, X_2)$

Triplet Network

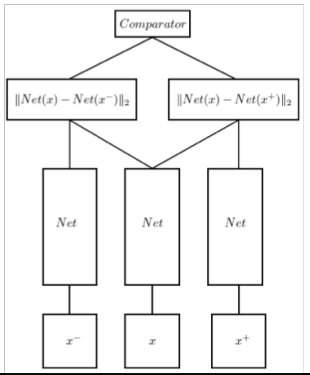
- Hoffer and Ailon, "Deep Metric Learning using Triplet Network," ICLR2015
  - Comprised of 3 instances of the same FFN

$$TripletNet(x, x^-, x^+) = \begin{bmatrix} \|Net(x) - Net(x^-)\|_2 \\ \|Net(x) - Net(x^+)\|_2 \end{bmatrix} \in \mathbb{R}^2_+$$

$$Loss(d_+, d_-) = \|(d_+, d_- - 1)\|_2^2 = const \cdot d_+^2$$

$$d_+ = \frac{e^{\|Net(x) - Net(x^+)\|_2}}{e^{\|Net(x) - Net(x^+)\|_2} + e^{\|Net(x) - Net(x^-)\|_2}}$$

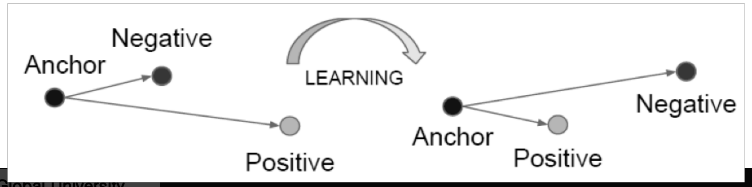
$$d_- = \frac{e^{\|Net(x) - Net(x^-)\|_2}}{e^{\|Net(x) - Net(x^+)\|_2} + e^{\|Net(x) - Net(x^-)\|_2}}$$



Triplet Loss (Face Recognition)

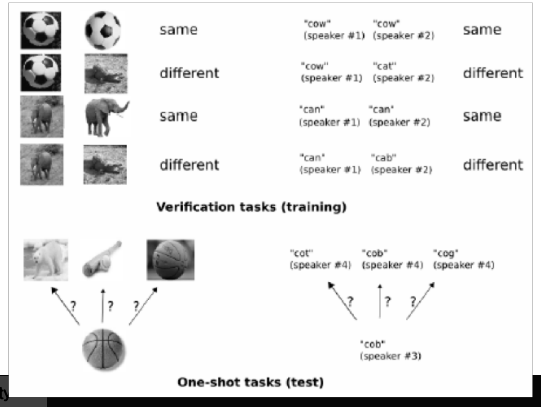
- Schroff, et al., "FaceNet: A Unified Embedding for Face Recognition and Clustering," Jun. 2015.

- Loss = $\sum_{i}^{N}\left[\|f(x_i^a)-f(x_i^p)\|_2^2-\|f(x_i^a)-f(x_i^n)\|_2^2+\alpha\right]_+$



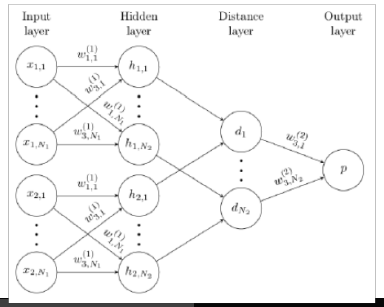
One-Shot Learning by Siamese Network

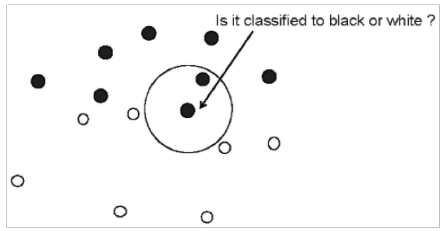
- Koch, et al., "Siamese Neural Networks for One-shot Image Recognition," 2015.
  - Make predictions given only a single example of each new class



One-Shot Learning by Siamese Network

- Koch, et al., "Siamese Neural Networks for One-shot Image Recognition," 2015.
  - Metric learning by Siamese network
  - Classification by nearest neighbor or hierarchical Bayesian program learning

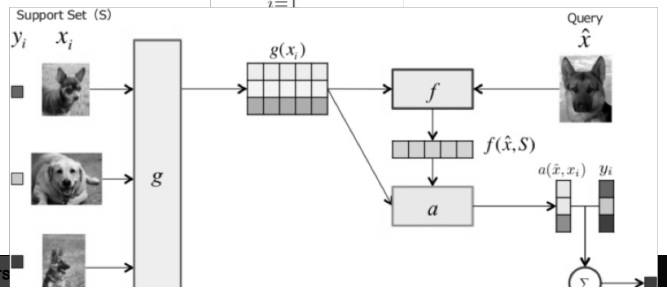




Matching Network

- Vinyals, et al., "Matching Networks for One Shot Learning, Jun. 2016.
  - Embedding ($g(.)$, $f(.)$)
  - Neural net with explicit memory
    - Attention using cosine distance
    - Adding new class

$$\hat{y}=\sum_{i=1}^{k}a(\hat{x},x_i)y_i$$



Memory Networks

- Weston, et al., "Memory Networks," Oct. 2014
- Memory network
  - A memory m: an array of objects indexed by m
  - Four (potentially learned) components I, G, O and R:
    - **I (input feature map)**: converts the incoming input to the internal feature representation.
    - **G (generalization)**: updates old memories given the new input.
    - **O (output feature map)**: produces a new output, given the new
    - input and the current memory state.
    - **R (response)**: converts the output into the response format desired.

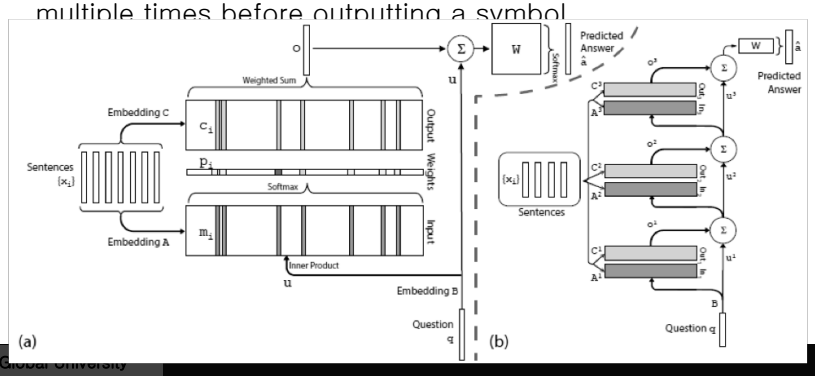
Memory Networks

- Weston, et al., "Memory Networks," Oct. 2014
- Operation given an input x,

1. Convert x to an internal feature representation $I(x)$.
2. Update memories $\mathbf{m}_i$ given the new input: $\mathbf{m}_i = G(\mathbf{m}_i, I(x), \mathbf{m}),\ \forall i.$
3. Compute output features o given the new input and the memory: $o = O(I(x), \mathbf{m}).$
4. Finally, decode output features o to give the final response: $r = R(o).$

End-to-End Memory Networks

- Sukhbaatar, et al., "End-to-End Memory Networks," Nov. 2015.
  - Recurrent neural network (RNN) architecture where the recurrence reads from a possibly large external memory multiple times before outputting a symbol



Question Answering using MemNN

- Weston, et al., "TOWARDS AI-COMPLETE QUESTION ANSWERING: A SET OF PREREQUISITE TOY TASKS," Dec. 2015

Task 1: Single Supporting Fact
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

Task 2: Two Supporting Facts
John is in the playground.
John picked up the football.
Bob went to the kitchen.
Where is the football? A:playground

Task 3: Three Supporting Facts
John picked up the apple.
John went to the office.
John went to the kitchen.
John dropped the apple.
Where was the apple before the kitchen? A:office

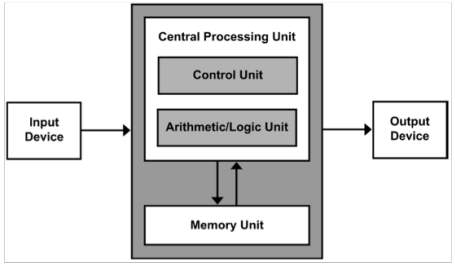
Task 4: Two Argument Relations
The office is north of the bedroom.
The bedroom is north of the bathroom.
The kitchen is west of the garden.
What is north of the bedroom? A: office
What is the bedroom north of? A: bathroom

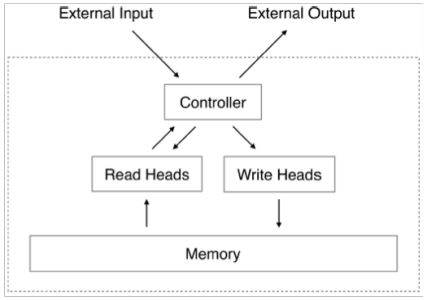
Task 5: Three Argument Relations
Mary gave the cake to Fred.
Fred gave the cake to Bill.
Jeff was given the milk by Bill.
Who gave the cake to Fred? A: Mary
Who did Fred give the cake to? A: Bill

Task 6: Yes/No Questions
John moved to the playground.
Daniel went to the bathroom.
John went back to the hallway.
Is John in the playground? A:no
Is Daniel in the bathroom? A:yes

Neural Turing Machines

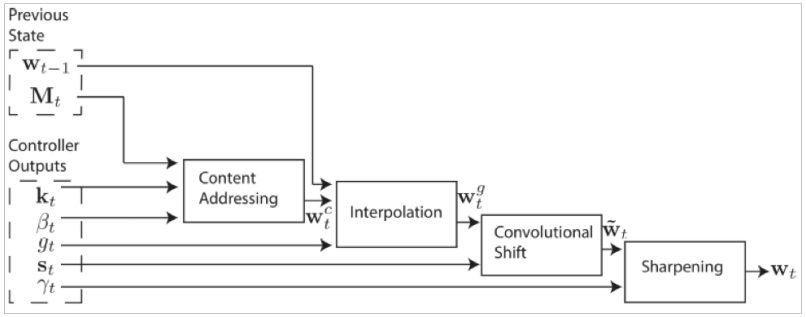
- Graves, et al., “Neural Turing Machines,” Dec. 2014.





Neural Turing Machines

- Graves, et al., “Neural Turing Machines,” Dec. 2014.



Agenda

- One-shot Learning
- Meta Learning
- Continual Learning

Meta Learning

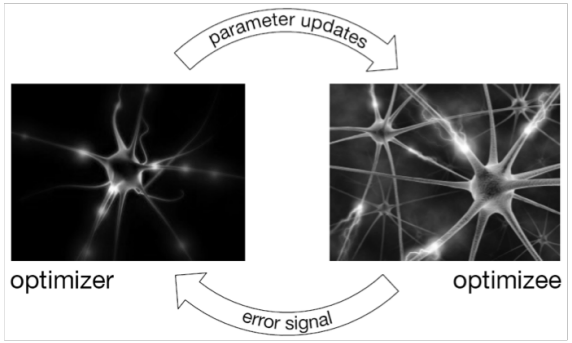
- Meta learning = “Learning to Learn”
  - A subfield of machine learning where automatic learning algorithms are applied on metadata about machine learning experiments.
- Motivation
  - Machine learning usually requires a large number of training iteration on a large volume of training data.
  - Human learning
    - Human can learn instantly from one or few examples

Meta Learning

- Meta-learning suggests framing the learning problem at two levels.
  - Slower extraction of information learned across all the tasks
  - Quick acquisition of knowledge within each separate task presented.

Learning Optimizer

- Andrychowicz, et al., "Learning to learn by gradient descent by gradient descent," Nov. 2016.



Learning Optimizer

- Andrychowicz, et al., "Learning to learn by gradient descent by gradient descent," Nov. 2016.
- Gradient descent

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t)$$

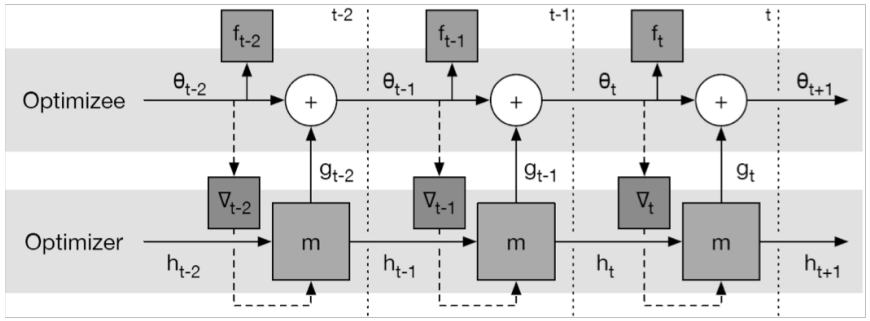
- Learning optimizer

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi)$$

$$\mathcal{L}(\phi) = \mathbb{E}_f\left[f\left(\theta^*(f,\phi)\right)\right] \quad \text{or} \quad \mathcal{L}(\phi) = \mathbb{E}_f\left[\sum_{t=1}^{T} w_t f(\theta_t)\right]$$

Learning Optimizer

- Andrychowicz, et al., "Learning to learn by gradient descent by gradient descent," Nov. 2016.



Learning Optimizer

- Ravi, et al., "Optimization as a Model for Few-Shot Learning," ICLR2017.
  - Meta-learner (LSTM)
  - Learner (neural network classifier)
- Gradient-based learning

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

- LSTM step

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$f_t = 1, c_{t-1} = \theta_{t-1}, i_t = \alpha_t, \text{ and } \tilde{c}_t = -\nabla_{\theta_{t-1}} \mathcal{L}_t$$

Learning Optimizer

- Gradient-based learning

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

- LSTM step

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

  - Input gate

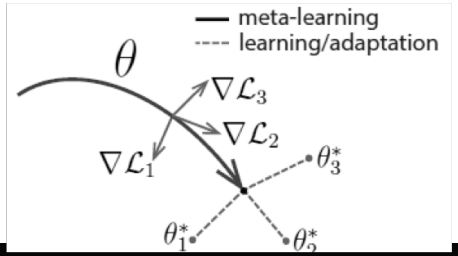
$$i_t = \sigma\left(\mathbf{W}_I \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}\right] + \mathbf{b}_I\right)$$

  - Forget gate

$$f_t = \sigma\left(\mathbf{W}_F \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1}\right] + \mathbf{b}_F\right)$$

MAML

- Finn, et al., "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," Jul. 2017.
  - Explicitly trained such that a small number of gradient steps with a small amount of training data from a new task will produce good generalization performance on that task.
  - Maximizing the **sensitivity** of the loss functions of new tasks with respect to the parameters



MAML

- Finn, et al., "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," Jul. 2017.
  - Maximizing the sensitivity of the loss functions of new tasks with respect to the parameters

Algorithm 1 Model-Agnostic Meta-Learning

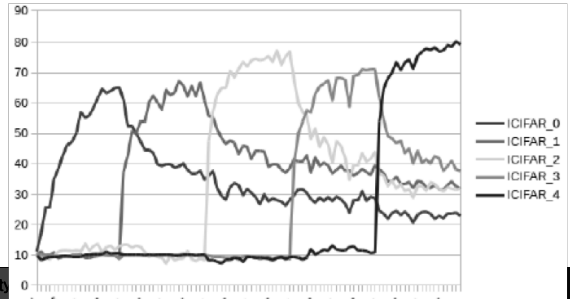
Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

Agenda

- One-shot Learning
- Meta Learning
- Continual Learning

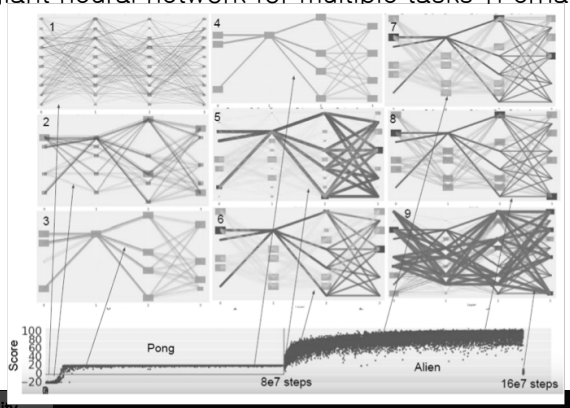
Continual Learning

- Continual learning (lifelong learning): frameworks where **knowledge acquired from past tasks is accumulated for use on future tasks**, i.e. where learning continually proceeds in an online fashion.
  - Major hurdle: **catastrophic forgetting**



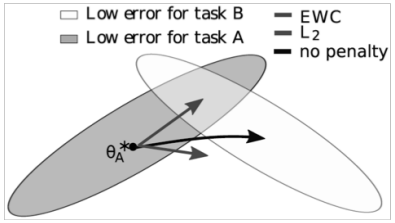
PathNet

- Fernando, et al., "PathNet: Evolution Channels Gradient Descent in Super Neural Networks, 2017.
  - One giant neural network for multiple tasks [Fernando17]



Elastic Weight Consolidation

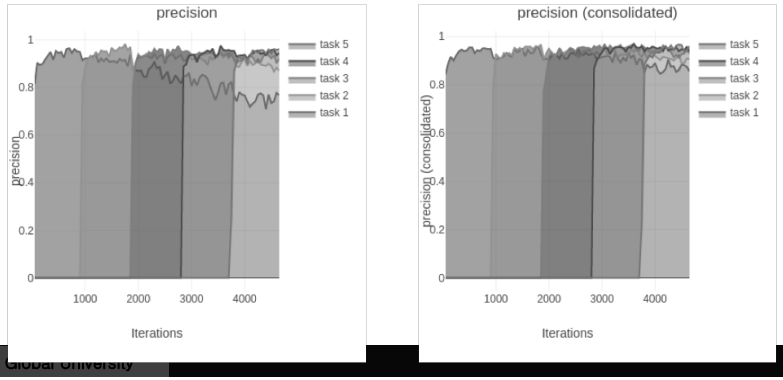
- Kirkpatrick, et al., "Overcoming catastrophic forgetting in neural networks," Jan. 2017.
  - Selectively slow down learning on the weights important for previous tasks



$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$$

Elastic Weight Consolidation

- Kirkpatrick, et al., “Overcoming catastrophic forgetting in neural networks,” Jan. 2017.
  - Selectively slow down learning on the weights important for previous tasks



Continual Learning

- Following works
  - J. Yoon, et al., "LIFELONG LEARNING WITH DYNAMICALLY EXPANDABLE NETWORKS," ICLR2018, Aug. 2017
  - H. Shin, et al., "Continual Learning with Deep Generative Replay," Dec. 2017
  - S. Lee, et al., “Overcoming Catastrophic Forgetting by Incremental Moment Matching,” Jan 2018
  - S. Wen, “Overcoming catastrophic forgetting problem by weight consolidation and long-term memory,” May 2018.
  - J. Serra, et al., “Overcoming catastrophic forgetting with hard attention to the task,” May 2018
  - A. Zacarias, et al., “SeNA-CNN – Overcoming Catastrophic Forgetting in Convolutional Neural Networks by Selective Network Augmentation,” May 2018

PyTorch Tutorial

Jinhyeok Yang
Handong Global University
2018. 01. 08.

- What is Deeplearning Framework?
- Basic Usage
- MLP, CNN, GAN Code (Additional Resources - Github)

한동대학교 HANDONG GLOBAL UNIVERSITY

What is Deeplearning Framework?

- 개발과 엔지니어링에 최소한의 시간만 투자 가능
- 연구 위주로 집중 가능
- 빠르고 안정적인 딥러닝 어플리케이션 개발 가능
- Python으로 쉽고 명료하게 모델 실험 가능

한동대학교 HANDONG GLOBAL UNIVERSITY

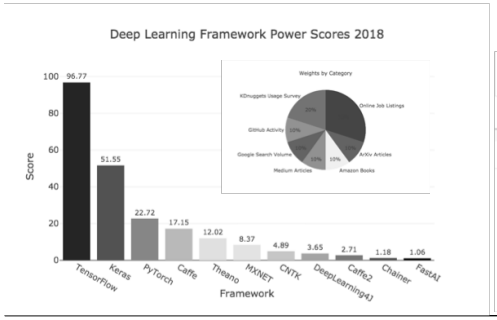
What is Deeplearning Framework?

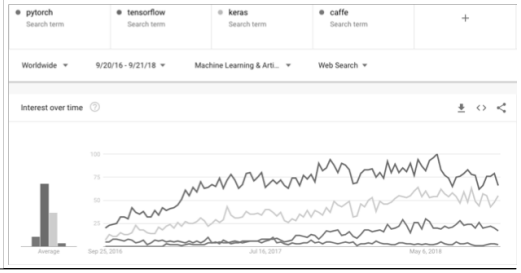
- 우후죽순 등장하는 딥러닝 프레임워크들
- 가장 많이 사용하는 프레임워크는 TensorFlow
- But, 우리가 실습할 프레임워크는 PyTorch
  - 모델을 만들기 쉽다.
  - 모델을 다양한 방식으로 변경해서 실험하기 쉽다.
  - 편리하다.
  - 연구자들이 많이 사용하기 때문에 오픈소스 모델도 많다.

한동대학교 HANDONG GLOBAL UNIVERSITY

What is Deeplearning Framework?

- **Power and Trends**



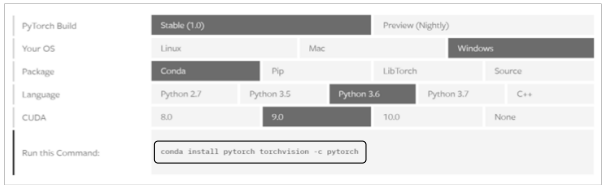


Basic Usage

- **설치 방법 1 (Local machine, if you have a GPU)**
  - **Easy easy (follow the command and enter "yes")**
  - **Needed**
    - **Anaconda (https://www.anaconda.com/download/)**
      - **Recommended python 3.6 version**
      - **After installation, run the command**

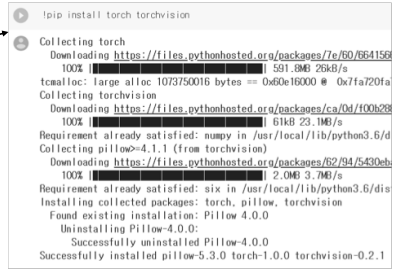
```
C:\Users\YANGYANG\>conda install python=3.6.7
```

    - **Pytorch (https://pytorch.org/get-started/locally/)**



Basic Usage

- **설치 방법 2 (Google Colab, GPU: Tesla K80, expire every 12h)**
  - **Recommended for the camp**
  - **Easier than the previous**
  - **Go to (https://colab.research.google.com)**
  - **Sign in with your Google ID**
  - **Make a new python3 note**
  - **Run the command**



Basic Usage

- **모델 학습의 순서**
  1) **모델 및 하이퍼파라미터 정의**
  2) **데이터 불러오기**
  3) **데이터 전처리**
  4) **전처리 된 데이터를 모델에 넣고 forward propagation**
  5) **Loss 계산**
  6) **Backward propagation (gradient 계산)**
  7) **Optimizer로 update**
  8) **원하는 step만큼 2-7 반복**

Basic Usage

- 모델 학습의 순서
  1) 모델 및 하이퍼파라미터 정의
  2) 데이터 불러오기
  3) 데이터 전처리
  4) 전처리 된 데이터를 모델에 넣고 forward propagation
  5) Loss 계산
  6) Backward propagation (gradient 계산)
  7) Optimizer로 update
  8) 원하는 step만큼 2-7 반복

```
class ConvNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
```

```
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
```

```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize(mean=(0.5, 0.5, 0.5),
                                std=(0.5, 0.5, 0.5))])
```

```
outputs = model(images)
```

```
loss = criterion(outputs, labels)
```

```
loss.backward()
```

```
optimizer.step()
```

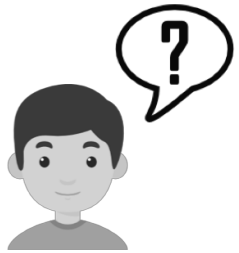
- 완료하면 학습된 모델 테스트 !

Practice

- Go to GitHub Code
  - https://github.com/Yangyangii/pytorch-basic
- Copy and paste the above link into Colab
- Or download it to your machine

Reference

- Google Colab Usage (Really good reference)
  - https://zzsza.github.io/data/2018/08/30/google-colab/
- PyTorch Tutorial (yunjey's repo, pytorch 0.4)
  - https://github.com/yunjey/pytorch-tutorial
- More GAN codes (Colab, pytorch 1.0)
  - https://github.com/Yangyangii/GAN-Tutorial



DAY 3

Recurrent Neural Networks

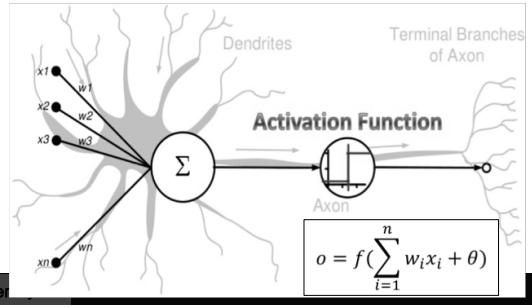
Injung Kim
Handong Global University
2019. 1. 9.

Agenda

- Introduction to RNN
- Training RNN
- Long Short-Term Memory Network
- Successful Applications
- Q&A

Artificial Neural Networks

- **Artificial neural network**: a mathematical model inspired by biological neural networks.
  - Learns mapping between vectors or sequences
    - Classification, regression, prediction, diagnosis, etc.
  - Learns probabilistic density
    - Sample generation, transform, restoration, etc.

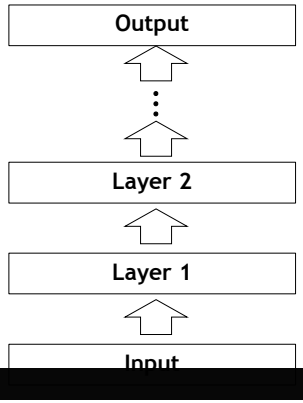


Deep Learning

- A branch of machine learning based on a set of algorithms that attempt to model **high-level abstractions in data**, mostly, based on **deep networks**.
  - Each layer combines input features to produce high-level features

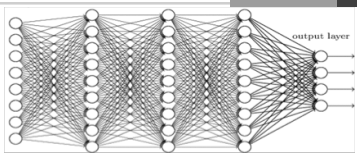
$$o = f(\sum_{i=1}^{n} w_i x_i + \theta)$$

  - A neural network with many layers can extract high-level features

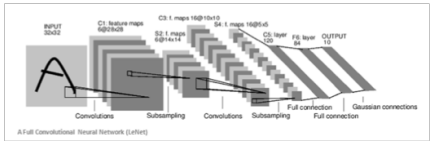


Deep Learning Architectures

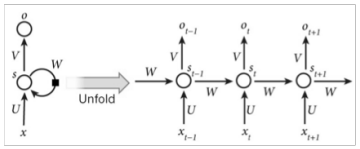
- Deep Neural Networks (DNN)
  - MLP, SOM, RBF, …
  - RBM, DBN, DBM, ⋯



- Convolutional neural networks (CNN)
  - Recognition/processing/generation of images
  - Combines heterogeneous layers (convolution, pooling, etc.)
  - Learns position independent local features

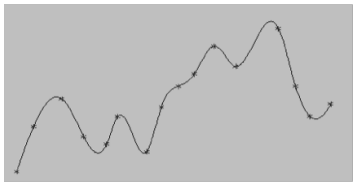


- Recurrent neural networks (RNN)
  - Recognition/processing/generation of time-series data
  - Recurrent connection (memory)
  - Input + context

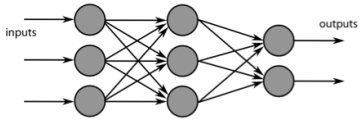


Recurrent Neural Networks

- Motivation: analyzing time series data
  - Many real world data are dependent on the previous or next data.

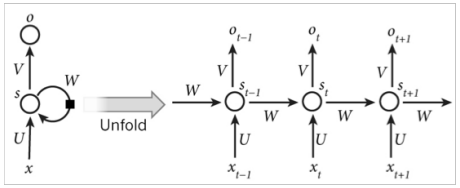


  - Feed forward networks assumes all inputs are independent from each other



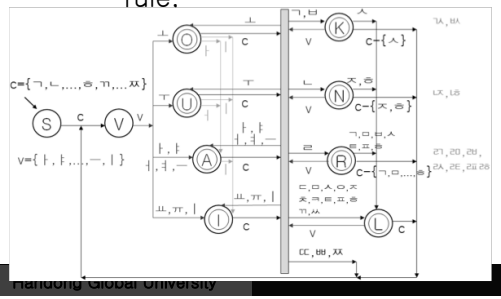
Recurrent Neural Networks

- Recurrent neural network is a neural network specialized for processing a sequence of values $x^{(1)}, x^{(2)}, \ldots, x^{(\tau)}$.
  - Neural networks with recurrent connection
  - State of nodes affect the output and the next state
  - Model for dynamic process
  - Temporarily shared connections

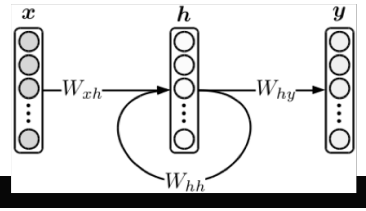


RNN State vs. Automata

- Automata (finite state machine)
  - Discrete states
    - 1st order Markov assumption
  - Discrete transition/output rule.

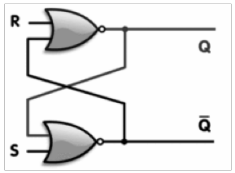


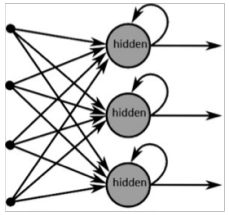
- RNN state (memory)
  - Vector representation of states
    - Longer history
  - Transition/output rule: mapping by neural network
  - State and rules are trained from data



Flip-Flop vs. RNN

- **Flip-flop** or **latch** is a circuit that has two stable states and can be used to store state information.
- RNN decides new state from input and previous state through feedback connection





State Transition Without Input

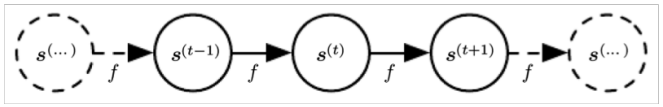
- A recurrent dynamic system

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta})$$

  - Contains a feed back loop

$$\begin{aligned}\boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) \\ &= f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta})\end{aligned}$$

  - Unfolded computational graph sharing parameters across a deep network structure



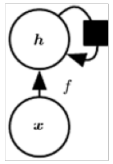
State Transition With Input

- A recurrent dynamic system with external input $x^{(t)}$
  - Input sequence: $x^{(1)}, x^{(2)}, \dots, x^{(t)}, \dots, x^{(\tau)}$

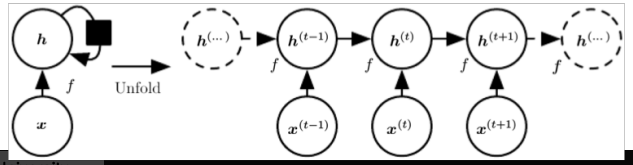
$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

  - When the state is hidden

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$



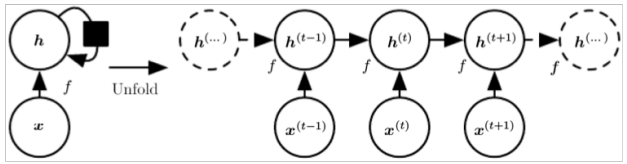
    - $h^{(t)}$: lossy summary of task-relevant aspects of the past sequence



State Transition With Input

- State $g^{(t)}$ after accepting t inputs:

$$\begin{aligned}\boldsymbol{h}^{(t)} &= g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \dots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) \\ &= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})\end{aligned}$$

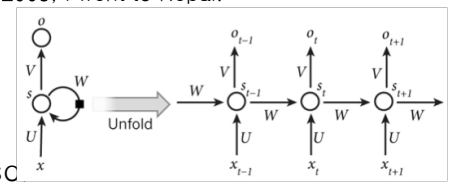


State Transition With Input

- $g^{(t)}$ is factorized into repeated application of a function f.
  - $g^{(t)}$: separate model for all possible time steps
  - f: a single model that operates on all time steps and all sequence lengths
  - RNN learns f rather than $g^{(t)}$.
- Advantages
  1. Regardless of the sequence length, the learned model always has the same input size.
     - Specified in terms of transition from one state to another state, rather than specified in terms of a variable-length history of states.
  2. The same transition function f with the same parameters at every time step.

Parameter Sharing in RNN

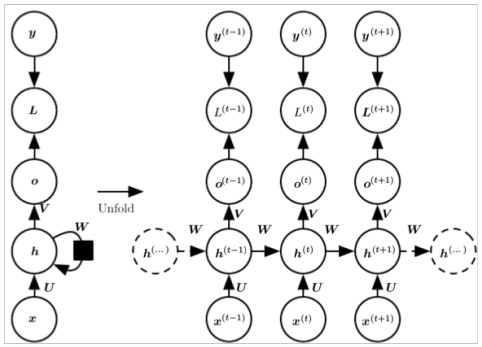
- Parameter sharing in RNN
  - A specific piece of information can occur at multiple positions within the sequence.
    Ex) "I went to Nepal **in 2009**"
    "**In 2009**, I went to Nepal."



- Comparisc
  - Traditional FFN: all rules **separately at each position**
  - CNN: shares the same weights **across position**
  - RNN: shares the same weights **across time steps**

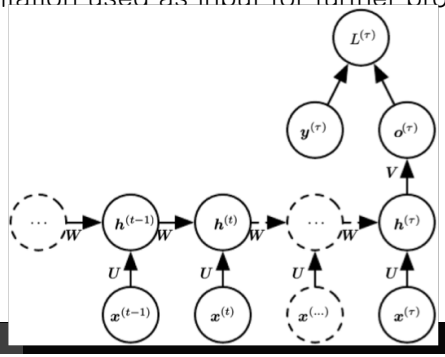
Recurrent Networks (Elman net)

- RNN with <u>feedback connections between hidden units</u>



Recurrent Networks with Single Output

- Recurrent networks with recurrent connections between hidden units
  - sequence input ➔ single output
  - Summarize a sequence and produce a fixed-size representation used as input for further processing



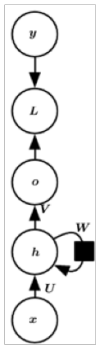
Matrix Notation

- Propagation
  - $y_j = f(\sum_i w_{ij} x_i + b_j) = f(a_j)$
  - $a_j = \sum_i w_{ij} x_i + b_j$
- Matrix form
  - $X = (x_1, x_2, \dots, x_N)^T, Y = (y_1, y_2, \dots, y_M)^T, A = (a_1, a_2, \dots, a_M)^T$
  - $W = \begin{pmatrix} w_{11} & \cdots & w_{N1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{NM} \end{pmatrix}, B = (b, b_2, \dots, b_M)^T$
  - $A = WX + B = \begin{pmatrix} w_{11} & \cdots & w_{N1} \\ \vdots & \ddots & \vdots \\ w_{1M} & \cdots & w_{NM} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_M \end{pmatrix} = \begin{pmatrix} \sum_i w_{i1} x_i + b_1 \\ \sum_i w_{i2} x_i + b_2 \\ \dots \\ \sum_i w_{iM} x_i + b_M \end{pmatrix}$
  - $Y = f(A) = f(WX + B)$

Recurrent Neural Networks

- RNN that produce an output at each time step and have recurrent connections.

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

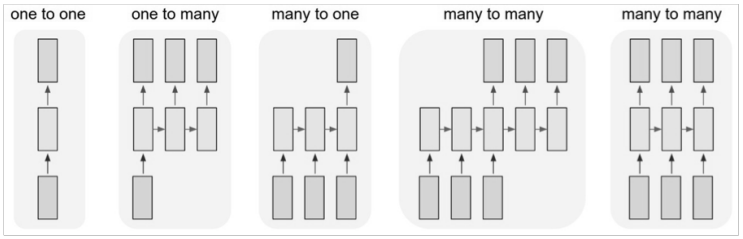


- V: hidden-to-output
- W: hidden-to-hidden
- U: input-to-hidden
- b: bias for hidden
- c: bias for output

Any function computable by a Turing machine can be computed by such a recurrent network of a finite size. (RNN is Turing-complete.)

Recurrent Neural Networks

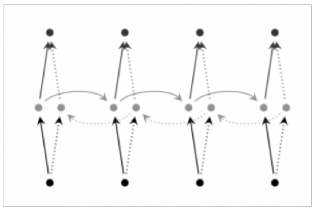
- RNN can learning various types of mappings



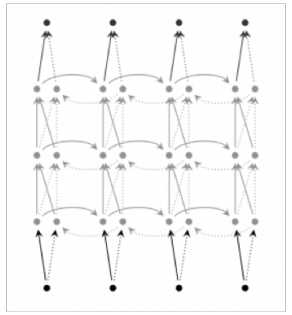
- Widely applied to practical tasks
  - Speech recognition/synthesis, handwriting recognition, machine translation, automatic image captioning, etc.

Extensions of RNNs

- Bidirectional RNNs
  - Learns both forward and backward dependency



- Deep RNNs
  - Higher learning capacity



CBHG Module

- Wang, et.al., “TACOTRON: TOWARDS END-TO-END SPEECH SYNTHESIS,” 2017
  - Convolution Bank
  - Highway network
  - Bidirectional GRU

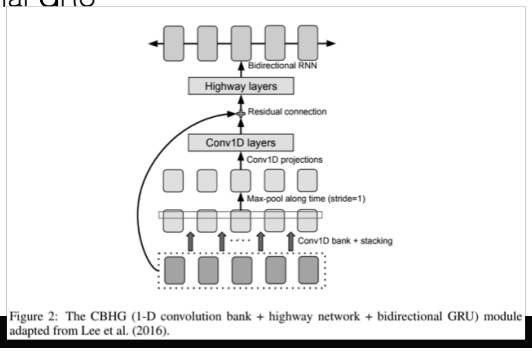
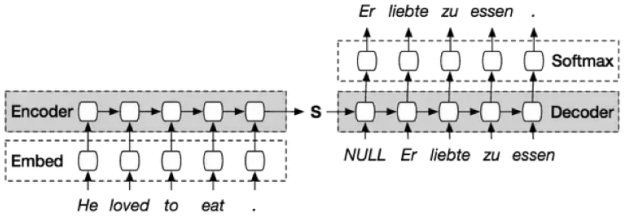


Figure 2: The CBHG (1-D convolution bank + highway network + bidirectional GRU) module adapted from Lee et al. (2016).

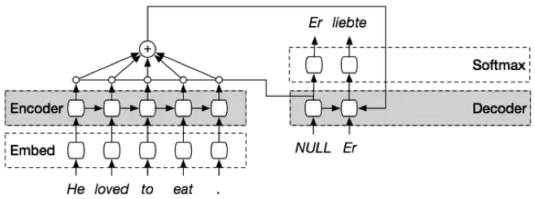
Encoder-Decoder Models

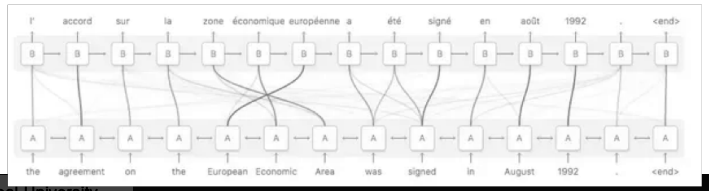
- Architecture for sequence-to-sequence mapping
  - Encoder: input sequence ➔ context vector
  - Decoder: context vector (+ previous output) ➔ new output
- The two modules are trained jointly to maximize the average of $\log P(y^1, y^2, \ldots, y^m | x^1, x^2, \ldots, x^n)$



Attention Models

- Decoder accesses context composed of weighted average of input states





Attention Models [Bahdanau16]

- Output

$$p(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$

- Hidden state

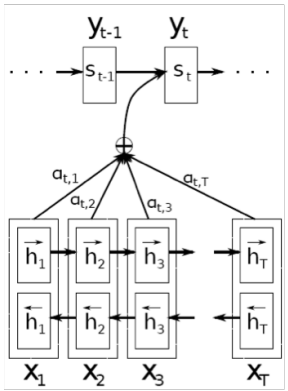
$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

- Context vector

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

- Attention

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \qquad e_{ij} = a(s_{i-1}, h_j)$$



Agenda

- Introduction to RNN
- Training RNN
- Long Short-Term Memory Network
- Successful Applications
- Q&A

Recurrent Neural Networks

- Loss of RNN
  - $L^{(t)}$: loss at time t

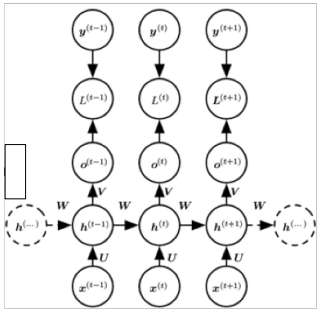
    Ex) negative log-likelihood of true target $y^{(t)}$

    $\hat{y}^{(t)}$ (activation of true-class output node)

    $$-\log p_{model}(y^{(t)}|\{x^{(1)}, \ldots, x^{(t)}\})$$

  - Total loss for a sequence

$$L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right)$$
$$= \sum_t L^{(t)}$$
$$= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right)$$



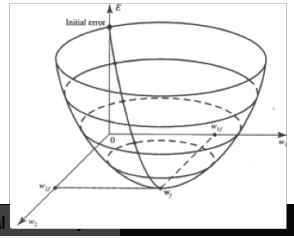
Gradient Descent Algorithm

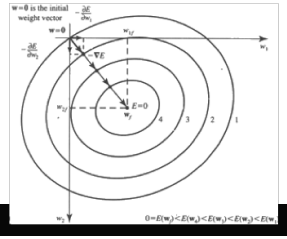
- Finding a parameter set W that minimizes loss L
  - Gradient of L w.r.t. W

$$\frac{\partial L}{\partial W^t} = \left(\frac{\partial L}{\partial w_1^t}, \frac{\partial L}{\partial w_2^t}, \ldots, \frac{\partial L}{\partial w_N^t}\right)$$

  - Update rule

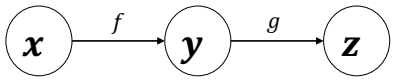
$$W^{t+1} = W^t - \eta \frac{\partial L}{\partial W^t}$$





Computing Gradients in RNNs

- Chain rule
  - If three variables x, y, and z have the following relation
    - $y = f(x)$
    - $z = g(y)$



  - Then,
    - For row vectors

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

    - For column vectors

$$\frac{\partial z}{\partial x} = \left(\frac{\partial y}{\partial x}\right)^T \frac{\partial z}{\partial y}$$

Chain Rule

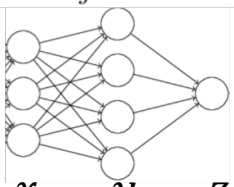
- For vectors $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$
  - Chain rule

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

gradient vectors

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z$$

n x m Jacobian matrix



x y z

- For tensors of arbitrary dim ?

Gradient and Jacobian

- **Gradient vector**: a multi-variable generalization of the derivative. (f is a scalar-valued function)

$$\frac{\partial f}{\partial x} = \nabla_x f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

- **Jacobian matrix**: matrix of all 1st order partial derivatives of a vector-valued function

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Training RNNs

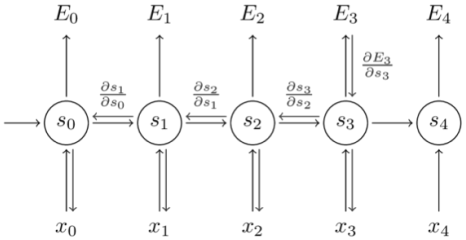
- BPTT (back-propagation through time)
  - Back-propagation algorithm applied to unfolded RNN
  - For each training sample, sum up the gradient at each time step

y_t: one-hot vector (only true class element is 1)

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t$$

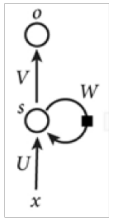
$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

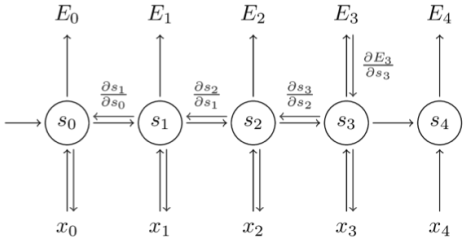


Training RNNs

- Example

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} = \boxed{\frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial o_3}\frac{\partial o_3}{\partial V}} = (\hat{y}_3 - y_3) \otimes s_3$$





$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}}\right)\frac{\partial s_k}{\partial W}$$

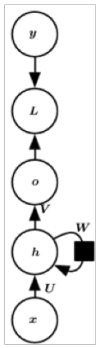
$$\boxed{\frac{\partial E_3}{\partial U}} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}}\right)\boxed{\frac{\partial s_i}{\partial U}}$$

Complexity: $O(\cdot)$, cannot be parallelized

Recurrent Neural Networks

- RNN that produce an output at each time step and have recurrent connections.

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$



- V: hidden-to-output
- W: hidden-to-hidden
- U: input-to-hidden
- b: bias for hidden
- c: bias for output

Any function computable by a Turing machine can be computed by such a recurrent network of a finite size. (RNN is Turing-complete.)

Derivative of Nonlinearities

- Hyperbolic tangent

$$y = \tanh(o) = \frac{\sinh(o)}{\cosh(o)} = \frac{e^{o} + e^{-o}}{e^{o} - e^{-o}}$$

  - Derivative of hyperbolic tangent

$$\frac{\partial y}{\partial o} = \frac{\partial \tanh(o)}{\partial o} = 1 - \tanh^2(o) = 1 - y^2$$

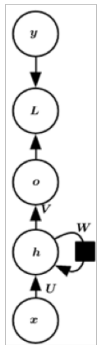
- Softmax

$$y_j = \frac{e^{o_j}}{\sum_k e^{o_k}}$$

  - Derivative of Softmax

$$\boldsymbol{\frac{\partial y_j}{\partial o_i} = y_j(1_{i,j} - y_i)}$$

Computing Gradients in RNNs



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

$$
\begin{aligned}
& L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) \\
&= \sum_t L^{(t)} \\
&= -\sum_t \log p_{\mathrm{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right)
\end{aligned}
$$

- Gradient w.r.t. output

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}$$

Derivative of Softmax

- Softmax

$$y_j = \frac{e^{o_j}}{\sum_k e^{o_k}}$$

- Derivative of fraction function

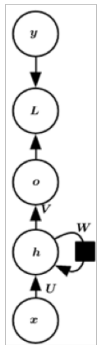
$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)\,g(x) - f(x)g'(x)}{g(x)^2}$$

- Derivative of Softmax

$$\boldsymbol{\frac{\partial y_j}{\partial o_i} = y_j(1_{i,j} - y_i)}$$

  - If $i = j$, $\frac{\partial y_j}{\partial o_i} = \frac{e^{o_j}(\sum_k e^{o_k}) - e^{o_j} e^{o_i}}{(\sum_k e^{o_k})^2} = \frac{e^{o_j}(\sum_k e^{o_k})}{(\sum_k e^{o_k})^2} - \frac{e^{o_j} e^{o_i}}{(\sum_k e^{o_k})^2}$
  $= \frac{e^{o_j}}{\sum_k e^{o_k}} - \frac{e^{o_j}}{\sum_k e^{o_k}} \frac{e^{o_i}}{\sum_k e^{o_k}} = y_j - y_j y_i = \boldsymbol{y_j(1 - y_i)}$
  - If $i \neq j$, $\frac{\partial y_j}{\partial o_i} = \frac{-e^{o_j} e^{o_i}}{(\sum_k e^{o_k})^2} = -\frac{e^{o_j}}{\sum_k e^{o_k}} \frac{e^{o_j}}{\sum_k e^{o_k}} = \boldsymbol{-y_i y_i = y_i(-y_i)}$

Computing Gradients in RNNs



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

$$
\begin{aligned}
&L\left(\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(\tau)}\},\{\boldsymbol{y}^{(1)},\ldots,\boldsymbol{y}^{(\tau)}\}\right) \\
&= \sum_t L^{(t)} \\
&= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(t)}\}\right)
\end{aligned}
$$

- Gradient w.r.t. output

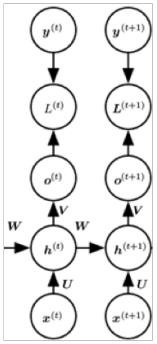
$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}}\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}$$

Let $j = y^{(t)}$, then

$$\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \frac{\partial(-\log \hat{y}_j^{(t)})}{\partial o_i^{(t)}} = \frac{\partial(-\log \hat{y}_j^{(t)})}{\partial \hat{y}_j^{(t)}}\frac{\partial \hat{y}_j^{(t)}}{\partial o_i^{(t)}}$$

$$= -\frac{1}{\hat{y}_j^{(t)}}\hat{y}_j^{(t)}(1_{i,j} - \hat{y}_j^{(t)}) = \hat{\boldsymbol{y}}_{\boldsymbol{j}}^{(\boldsymbol{t})} - \mathbf{1}_{\boldsymbol{i},\boldsymbol{j}}$$

Computing Gradients in RNNs



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

$$
\begin{aligned}
&L\left(\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(\tau)}\},\{\boldsymbol{y}^{(1)},\ldots,\boldsymbol{y}^{(\tau)}\}\right) \\
&= \sum_t L^{(t)} \\
&= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(t)}\}\right)
\end{aligned}
$$

- Gradient w.r.t. hidden ($t < \tau$)

$$\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} = \boldsymbol{V}$$

$$
\begin{aligned}
\nabla_{\boldsymbol{h}^{(t)}} L &= \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^{\top}(\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^{\top}(\nabla_{\boldsymbol{o}^{(t)}} L) \\
&= \boldsymbol{W}^{\top}(\nabla_{\boldsymbol{h}^{(t+1)}} L)\operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) + \boldsymbol{V}^{\top}(\nabla_{\boldsymbol{o}^{(t)}} L)
\end{aligned}
$$

$$\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}} = \frac{\partial \boldsymbol{a}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{a}^{(t+1)}}$$

Derivative of Hyperbolic Tangent

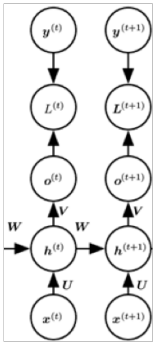
- Hyperbolic tangent

$$y = \tanh(o) = \frac{\sinh(o)}{\cosh(o)} = \frac{e^o + e^{-o}}{e^o - e^{-o}}$$

- Derivative of hyperbolic tangent

$$\frac{\partial \tanh(o)}{\partial o} = \frac{\partial \tanh(o)}{\partial o} = 1 - \tanh^2(o) = \mathbf{1} - \boldsymbol{y}^{\boldsymbol{2}}$$

Computing Gradients in RNNs



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

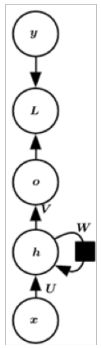
$$
\begin{aligned}
&L\left(\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(\tau)}\},\{\boldsymbol{y}^{(1)},\ldots,\boldsymbol{y}^{(\tau)}\}\right) \\
&= \sum_t L^{(t)} \\
&= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(t)}\}\right)
\end{aligned}
$$

- Gradient w.r.t. hidden (last time step, $\boldsymbol{t} = \boldsymbol{\tau}$)

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^{\top}\nabla_{\boldsymbol{o}^{(\tau)}} L$$

$$\frac{\partial L}{\partial h^{(\tau)}} = (\frac{\partial o^{(\tau)}}{\partial h^{(\tau)}})^T \frac{\partial L}{\partial o^{(\tau)}} = V^T \nabla_{o^{(\tau)}} L$$

Computing Gradients in RNNs



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

$$
\begin{aligned}
& L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) \\
&= \sum_t L^{(t)} \\
&= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right)
\end{aligned}
$$

- Gradient w.r.t. bias

$$
\begin{aligned}
\nabla_{\boldsymbol{c}} L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^{\top} \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \\
\nabla_{\boldsymbol{b}} L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^{\top} \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L
\end{aligned}
$$

$$
\frac{\partial L}{\partial b} = \left(\frac{\partial h^{(t)}}{\partial b}\right)^T \frac{\partial L}{\partial h^{(t)}} = \left(\frac{\partial a^{(t)}}{\partial b}\right)^T \left(\frac{\partial h^{(t)}}{\partial a^{(t)}}\right)^T \frac{\partial L}{\partial h^{(t)}}
$$

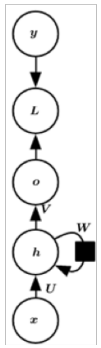
Jacobian Matrix

- A vector $\boldsymbol{x} \in \mathbb{R}^n$
- A vector function $\boldsymbol{f}: \mathbb{R}^n \Rightarrow \mathbb{R}^m$,
- Jacobian matrix $\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$

$$
\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}
$$

$$
\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}
$$

Computing Gradients in RNNs



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

$$
\begin{aligned}
& L\left(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\right) \\
&= \sum_t L^{(t)} \\
&= -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right)
\end{aligned}
$$

- Gradient w.r.t. connection weights

$$
\begin{aligned}
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L)\, \boldsymbol{h}^{(t)\top} \\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)\, \boldsymbol{h}^{(t-1)\top} \\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L)\, \boldsymbol{x}^{(t)\top}
\end{aligned}
$$

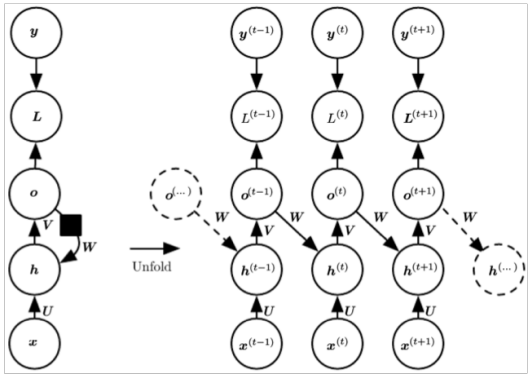
$$
\frac{\partial L}{\partial V} = \frac{\partial L}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial V}
$$

$$
\frac{\partial L}{\partial W} = \frac{\partial L}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial W}
$$

$$
\frac{\partial L}{\partial U} = \frac{\partial L}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial U}
$$

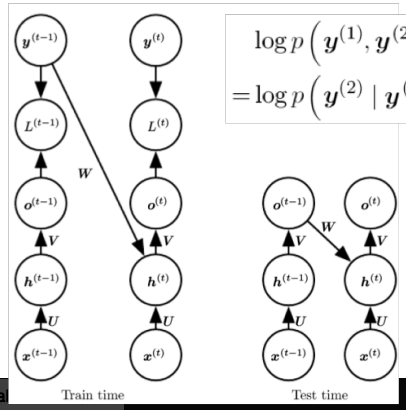
Recurrent Networks (Jordan net)

- RNN with only <u>feedback connections from the output to the hidden layer</u>.



Training with Teacher Forcing

- Model receives the ground truth output $y^{(t)}$ as input at time (t+1).
  - Examine a sequence with two time steps.



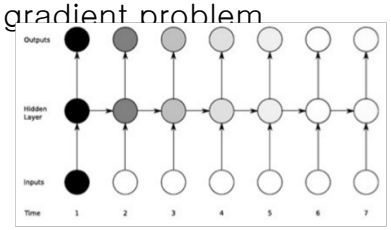
$$\log p\left(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right)$$
$$= \log p\left(\boldsymbol{y}^{(2)} \mid \boldsymbol{y}^{(1)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right) + \log p\left(\boldsymbol{y}^{(1)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right)$$

Agenda

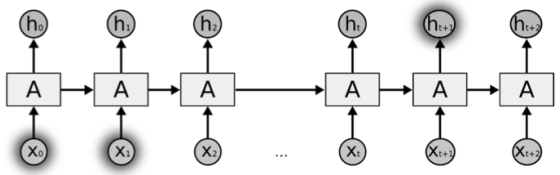
- Introduction to RNN
- Training RNN
- Long Short-Term Memory Network
- Successful Applications
- Q&A

Problems in RNN

- Vanishing gradient problem



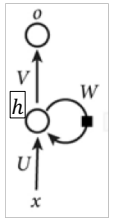
- Learning long-term dependency



Challenge of Long-Term Dependencies

- Recurrent networks involve the composition of the same function multiple times, once per time step.

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^{\top}\boldsymbol{h}^{(t-1)}$$
$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^{t}\right)^{\top}\boldsymbol{h}^{(0)}$$



- Eigen decomposition of W

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top}$$
$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^{\top}\boldsymbol{\Lambda}^{t}\boldsymbol{Q}\boldsymbol{h}^{(0)}$$

Challenge of Long-Term Dependencies

$$W = Q\Lambda Q^{\top}$$
$$h^{(t)} = Q^{\top}\Lambda^{t} Q h^{(0)}$$

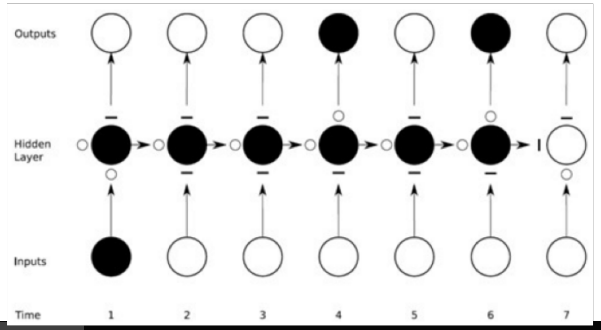
- Long-term dependency problem
  - Gradients propagated over many stages tend to either **vanish** ($|\lambda_i|<1$) or **explode** ($|\lambda_i|>1$)
  - Any component of $h(0)$ that is not aligned with the largest eigenvector will eventually be discarded.
  - Long-term dependencies are tend to be hidden by short-term dependencies

Input/Output Weight Conflict

- What if keep $|\lambda_i| \approx 1$?
  - Some networks (eg. ESN) are based on this idea.
- Remaining problems
  - Input weight conflict
    "The same incoming weight has to be used for both storing certain inputs and ignoring others, w_{ji} will often receive conflicting weight update signals during this time."
  - Output weight conflict
    "Conflicting weight update signals generated during sequence processing: these signals will attempt to make w_{kj} participate in (1) accessing the information stored in j and – at different times – (2) protecting unit k from being perturbed by j."

LSTM Networks

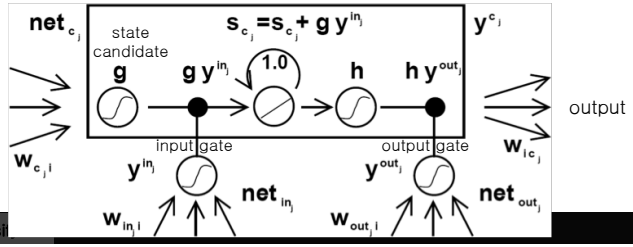
- LSTM: Long Short-Term Memory
  - Designed to learning long-term dependency
  - RNN with explicit gate to control data flow
    - Input/output/forget gates



LSTM Networks

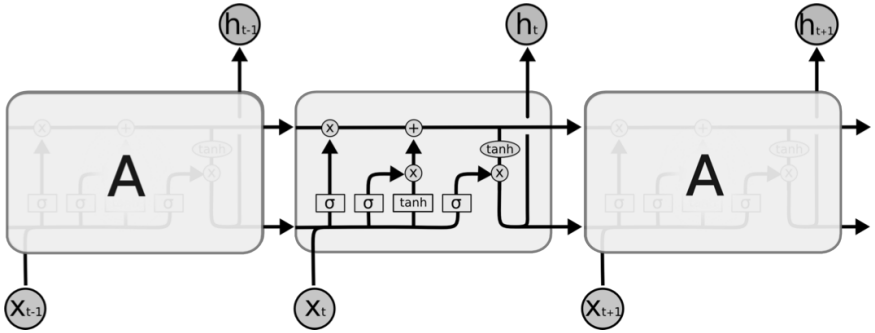
- Memory cell
  - Central linear unit with a fixed self-connection
- Input gate unit
  - Protects the memory contents stored in j from perturbation by irrelevant inputs.
- Output gate unit
  - Protects other units from perturbation by currently irrelevant memory contents stored in j.

cf. Forget gate was introduced in [Gers1999]



LSTM Networks

- Structure of LSTM networks
  - σ: gate networks



Leaky Units

- Accumulate a running average μ^t of target value v^t

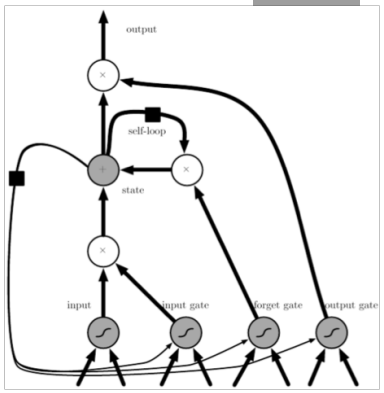
$$\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1-\alpha)v^{(t)}$$

  - α is near one (time constant)
  - μ^t moves smoothly transferring values from the past

- Leaky units: hidden units with linear self-connections

LSTM Networks

- State unit $s_i^{(t)}$
  - Linear self-loop
  - Weight of self-loop is controlled by **forget gate**



Forget gate

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

Input gate

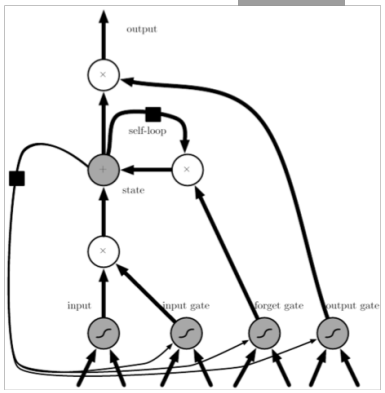
$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right)$$

State update

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right)$$

LSTM Networks

- Output $h_i^{(t)}$
  - Controlled by **output gate**



Output

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

Output gate

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right)$$

Gated Recurrent Units

- A single gating unit simultaneously controls the forgetting factor and the decision to update the state unit

Update of GRU state

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)} \right)$$

  - $u_i^{(t)}$: update gate, $r_i^{(t)}$: reset gate

cf. Update of LSTM state

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right)$$

Gated Recurrent Units

- Update gate

$$u_i^{(t)} = \sigma \left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)} \right)$$

  - Acts like conditional leaky integrators

- Reset gate

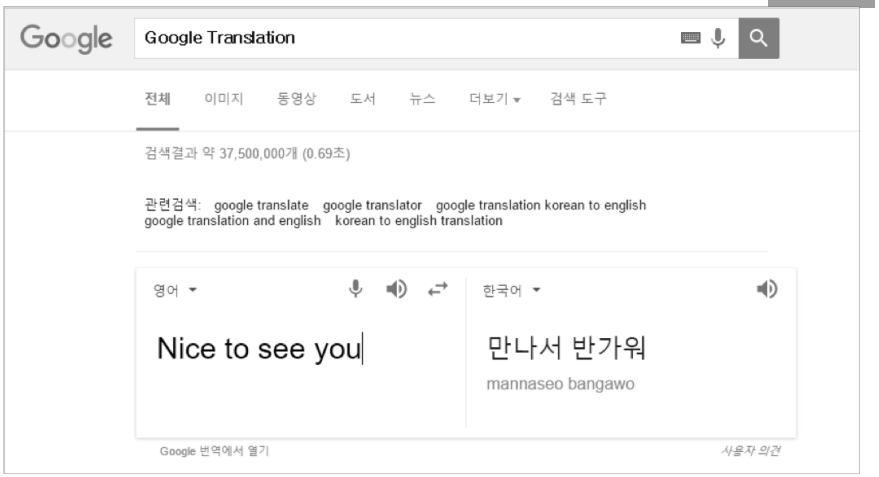
$$r_i^{(t)} = \sigma \left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)} \right)$$

  - Controls which parts of the state get used to compute the next target state,

Agenda

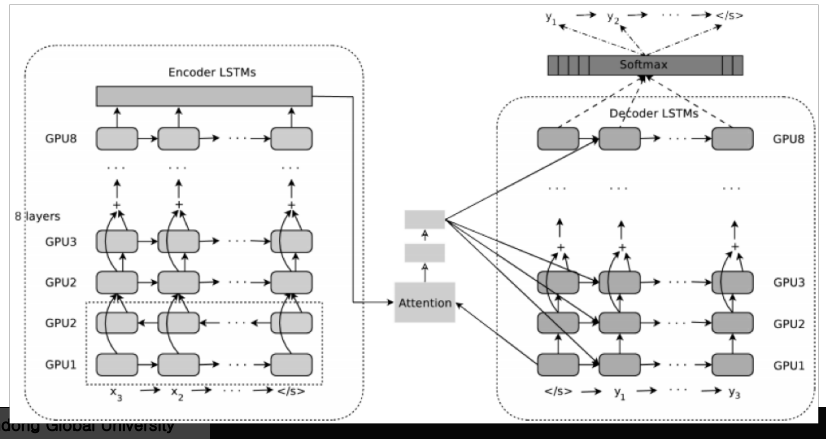
- Introduction to RNN
- Training RNN
- Long Short-Term Memory Network
- Successful Applications of RNN
- Q&A

Google Translation



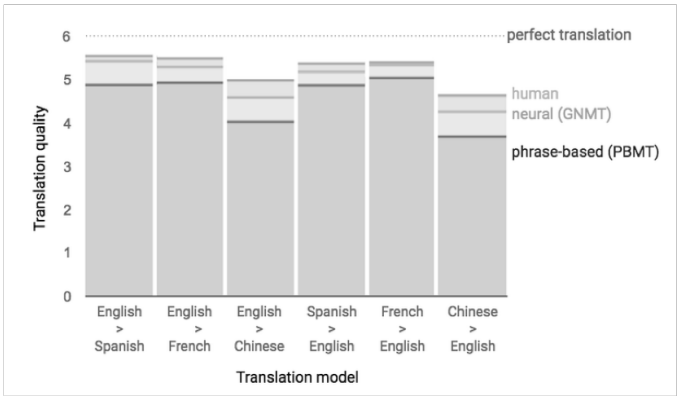
Machine Translation

- Google Neural Machine Translation (GNMT)
  - Word embedding + LSTM + attention model



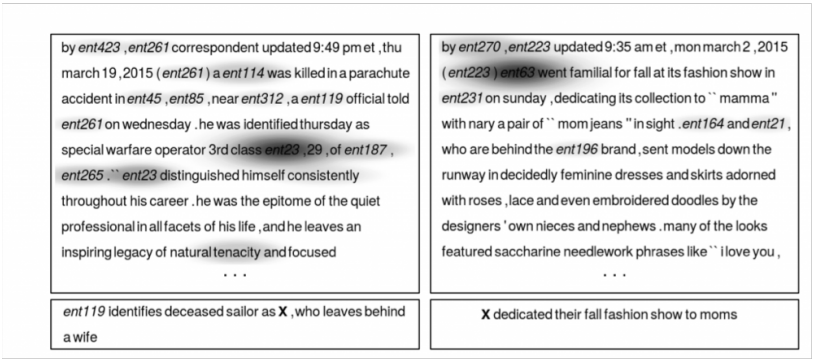
Machine Translation

- Google Neural Machine Translation (GNMT)
  - performance



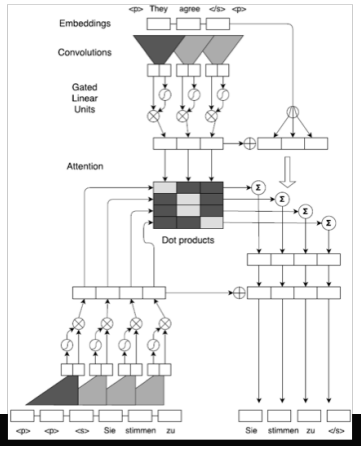
Question Answering

- Hermann, et.al, “Teaching Machines to Read and Comprehend,” 2015.



Machine Translation using CNN

- J. Gehring, et al., “Convolutional Sequence to Sequence Learning,” May 2017



Transformer Model

- A. Vaswan, “Attention Is All You Need” Jun. 2017
  - Attention mechanism instead of convolutional or recurrent units

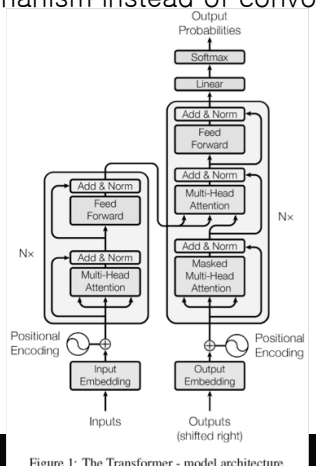
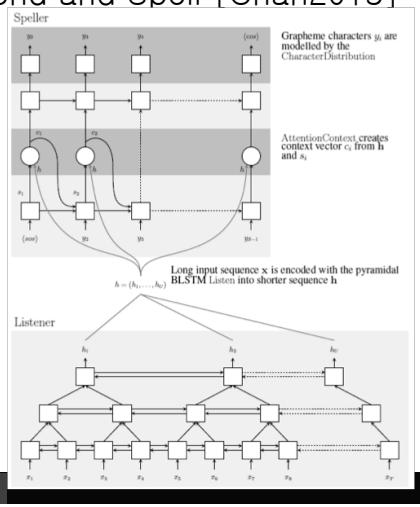


Figure 1: The Transformer - model architecture.

Neural Speech Recognition

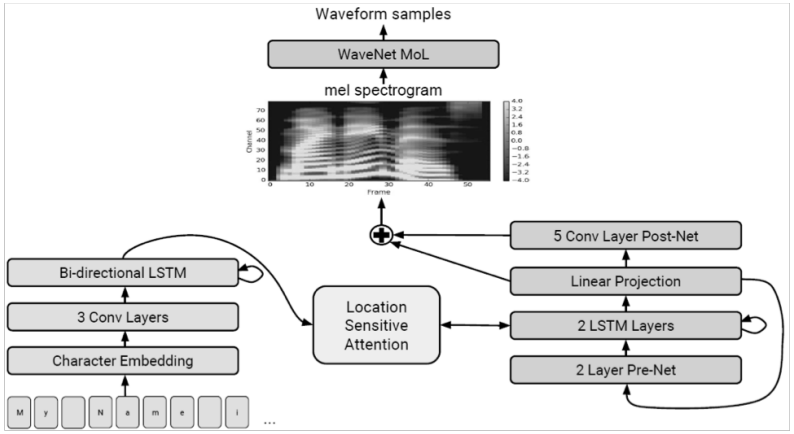
- Listen, Attend and Spell [Chan2015]



Pyramid RNN

Neural TTS (Speech Synthesis)

- Tacotron2 [Shen2017]



Visual Attention Model

- Mnih, et al, “Recurrent Models of Visual Attention”, 2014

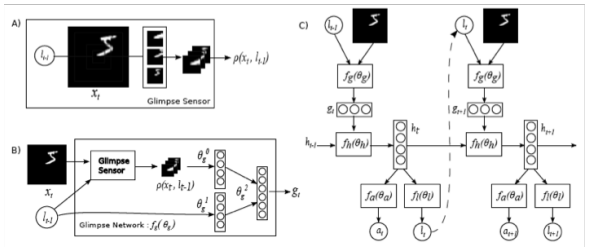
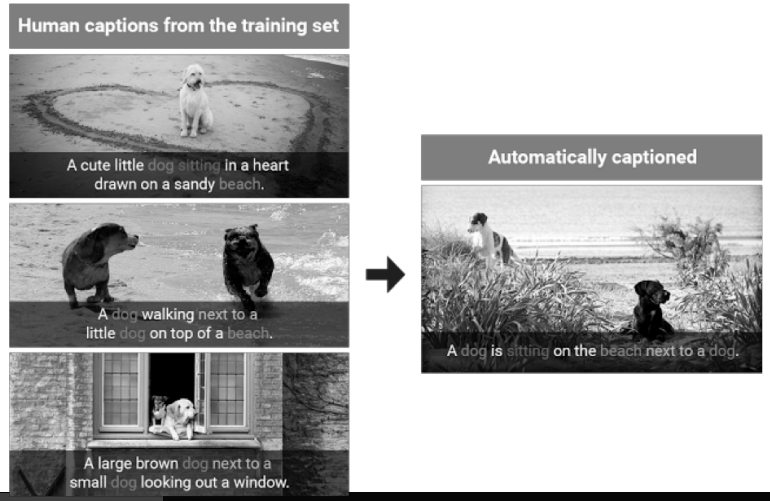


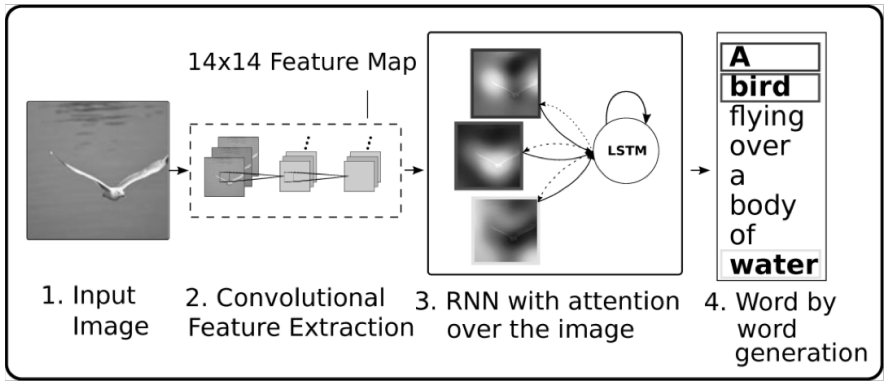
Figure 1: **A) Glimpse Sensor:** Given the coordinates of the glimpse and an input image, the sensor extracts a *retina-like* representation $\rho(x_t, l_{t-1})$ centered at l_{t-1} that contains multiple resolution patches. **B) Glimpse Network:** Given the location (l_{t-1}) and input image (x_t), uses the glimpse sensor to extract retina representation $\rho(x_t, l_{t-1})$. The retina representation and glimpse location is then mapped into a hidden space using independent linear layers parameterized by θ_g^0 and θ_g^1 respectively using rectified units followed by another linear layer θ_g^2 to combine the information from both components. The glimpse network $f_g(.; \{\theta_g^0, \theta_g^1, \theta_g^2\})$ defines a trainable bandwidth limited sensor for the attention network producing the glimpse representation g_t. **C) Model Architecture:** Overall, the model is an RNN. The core network of the model $f_h(.; \theta_h)$ takes the glimpse representation g_t as input and combining with the internal representation at previous time step h_{t-1}, produces the new internal state of the model h_t. The location network $f_l(.; \theta_l)$ and the action network $f_a(.; \theta_a)$ use the internal state h_t of the model to produce the next location to attend to l_t and the action/classification a_t respectively. This basic RNN iteration is repeated for a variable number of steps.

Automatic Image Captioning



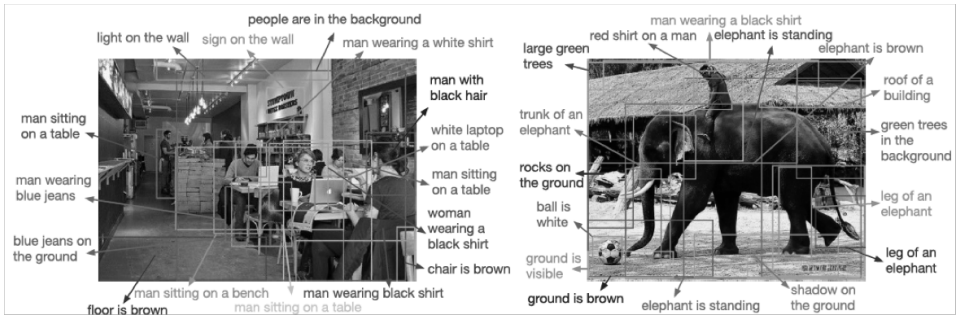
Automatic Image Captioning

- Xu, et.al, "Show, Attend, and Tell" 2016.
  - R-CNN + LSTM + attention model



Dense Captioning

- Johnson, et.al., "DenseCap: Fully Convolutional Localization Networks for Dense Captioning," 2017.5
  - Object detection + captioning



Deep Generative Models

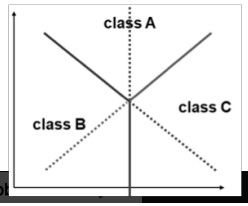
Injung Kim
Handong Global University
2019. 1. 9.

Agenda

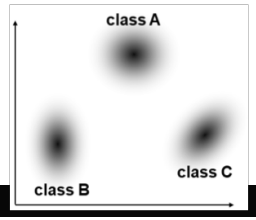
- Introduction
- Generative Adversarial Nets
- Variational Auto-Encoders
- Pixel RNN/CNN
- Q&A

Discriminative Models vs. Generative Models

- Discriminative models
  - How to categorize samples?
  - Decision boundary
    - "Eyes of solders"
    - Often models P(y|x)
  - Supervised learning
  - More discriminative

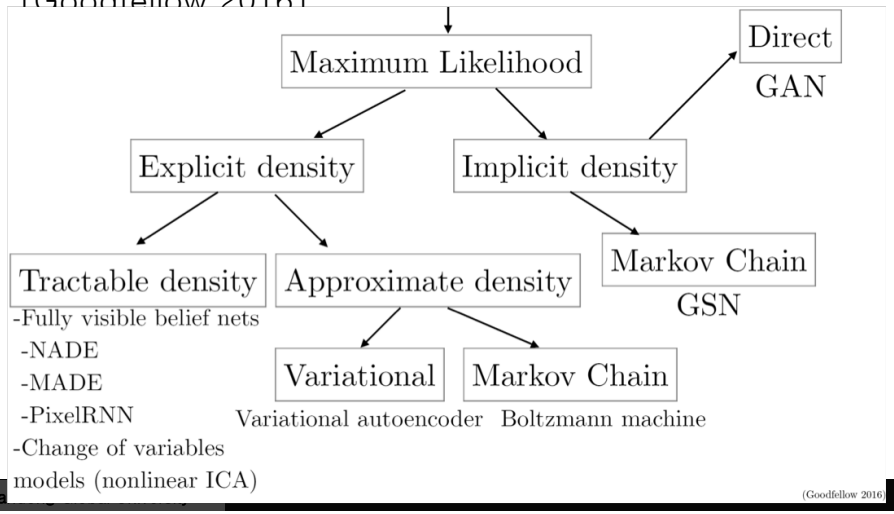


- Generative models
  - How the samples were generated?
  - Distribution
    - "Eyes of painters"
    - Often models P(x) or P(x|y)
  - Supervised/unsupervised learning
  - More stable



Texonomy of Generative Models

[Goodfellow 2016]



Emerging Generative Models

- Generative Adversarial Networks [Goodfellow14]
  - Goodfellow, et.al., "Generative Adversarial Nets," 2014
  - DCGAN, Conditional GAN, InfoGAN, EBGAN, BEGAN, CycleGAN, DiscoGAN, StarGAN, Unrolled GAN, WGAN, WGAN-GP, Cramer GAN, LS-GAN, Progressive GAN, SN-GAN, SA-GAN, BigGAN …
- Variational Auto-Encoders (VAE)
  - Kingma, et.al, "Auto-Encoding Variational Bayes," 2013
- Pixel Networks
  - Oord, et.al, "Pixel Recurrent Neural Networks," 2016
  - Pixel RNN, Pixel CNN, Pixel CNN++

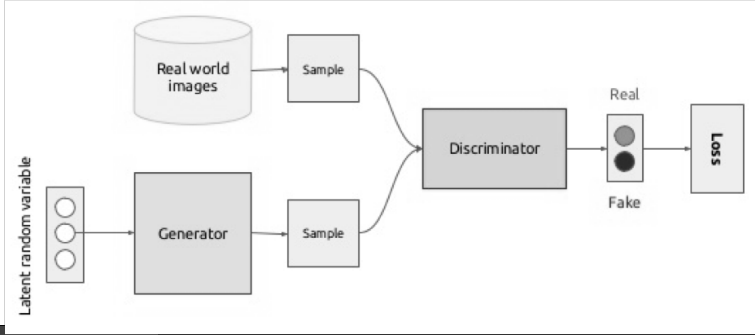
Agenda

- Introduction
- Generative Adversarial Nets
- Variational Auto-Encoders
- Pixel RNN/CNN
- Q&A

Generative Adversarial Networks (GANs)

- I. Goodfellow, et.al, "Generative Adversarial Nets," 2014. 6.

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

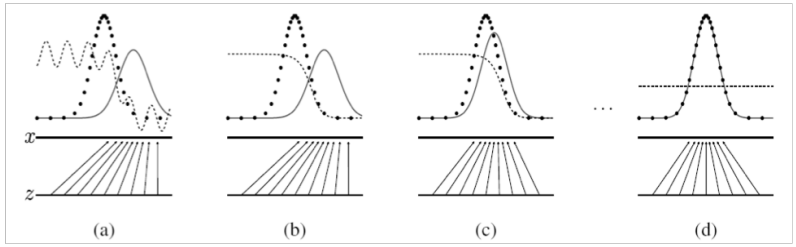


Training GAN

- Objective function

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

  - Minimax game between discriminator and generator



Training Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Divergence Between Distributions

- KL-divergence

$$KL\left(P_r(x) | P_g(x)\right) = \int P_r(x) \log \frac{P_r(x)}{P_g(x)} dx$$

  - Not symmetric

- Jenson-Shannon divergence (JSD)

$$JSD(\mathbb{P}_r \| \mathbb{P}_g) = \frac{1}{2} KL(\mathbb{P}_r \| \mathbb{P}_A) + \frac{1}{2} KL(\mathbb{P}_g \| \mathbb{P}_A)$$

where, $P_A = \frac{P_r + P_g}{2}$

Generative Adversarial Networks (GANs)

- GAN minimizes Jenson-Shannon divergence

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

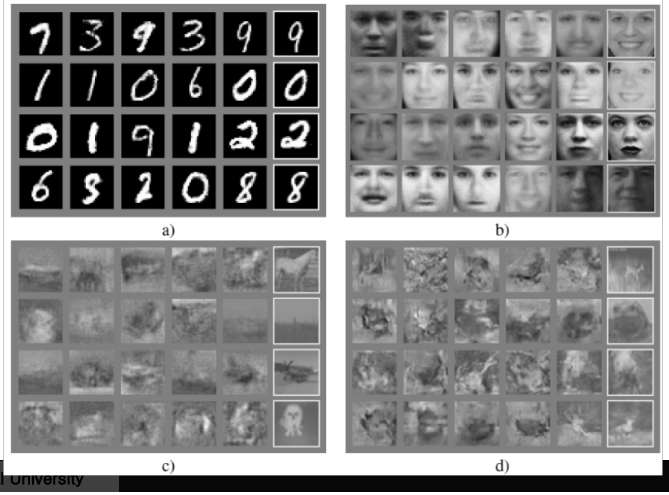
$$\begin{aligned} C(G) &= \max_D V(G, D) \\ &= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\ &= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] \\ &= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \end{aligned}$$

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right)$$

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right)$$

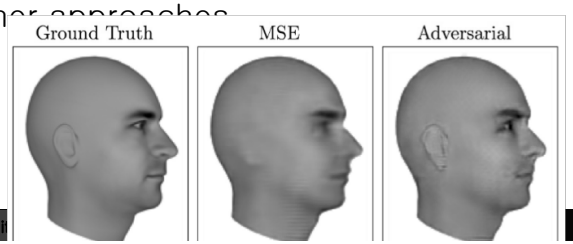
Generative Adversarial Networks (GANs)

- Generated samples and their nearest neighbors



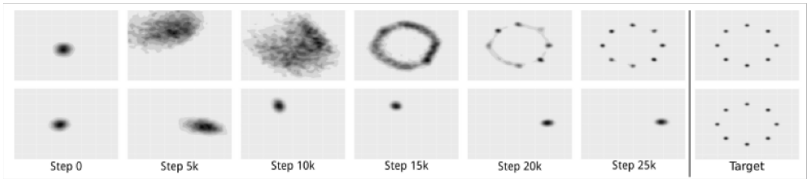
Advantage of GAN

- Generates samples only through FFN
  - Can be trained by only backprop., not requiring Markov Chain
  - Can utilize piece linear units (ex. ReLU)
- D and G can be implemented by a wide variety of functions
- Empirically, GAN produces more realistic samples than other approaches



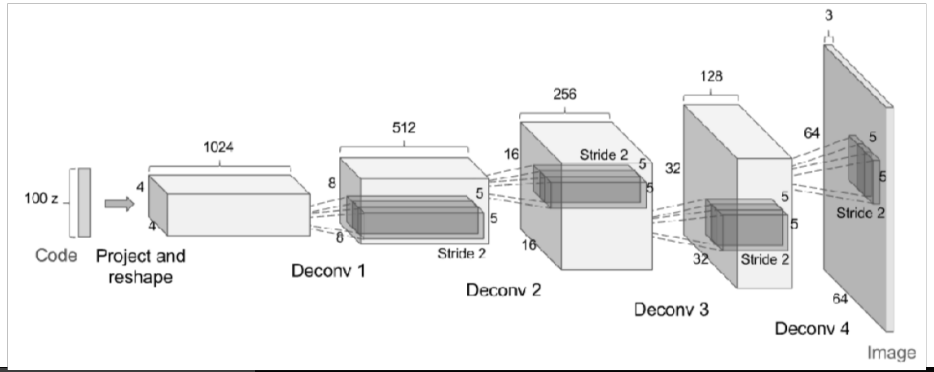
Disadvantages

- No explicit representation of probability density
- Training is unstable [Metz17]
  - Mode collapse
  - Generator and discriminator oscillate
  - Generator or discriminator completely wins the other



Deep Convolutional GAN (DCGAN)

- Radford, et.al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," 2015.11.

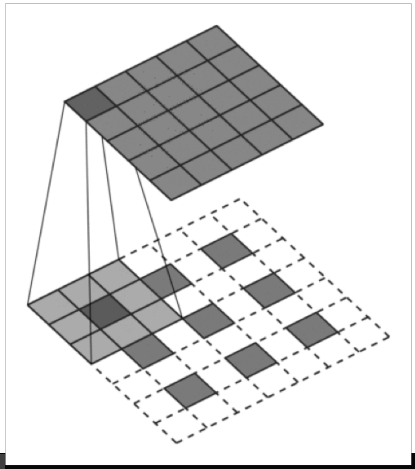


Deep Convolutional GAN (DCGAN)

- Architecture guidelines for stable Deep Convolutional GANs
  - Replace any pooling layers with **strided convolutions** (discriminator) and **fractionally-strided (transposed) 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.

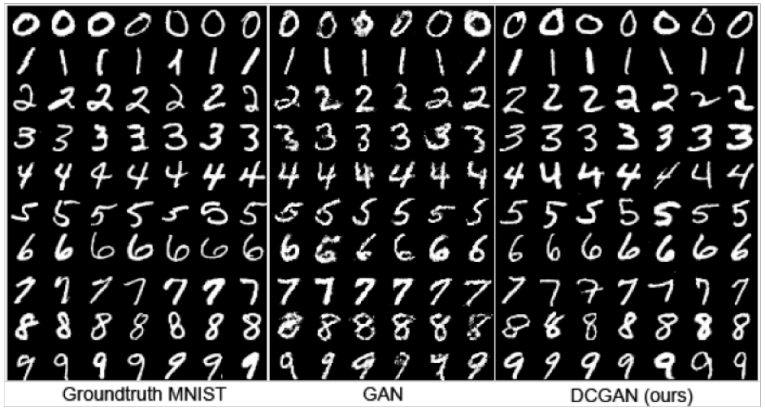
Transposed Convolution

- Convolution for upsampling



Deep Convolutional GAN (DCGAN)

- MNIST



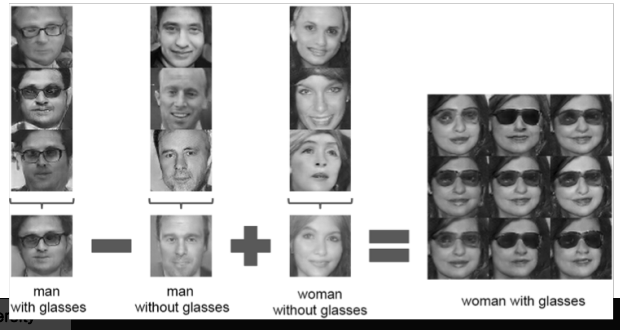
Deep Convolutional GAN (DCGAN)

Deep Convolutional GAN (DCGAN)

- LSUN Bedroom Images (5 epochs)

Deep Convolutional GAN (DCGAN)

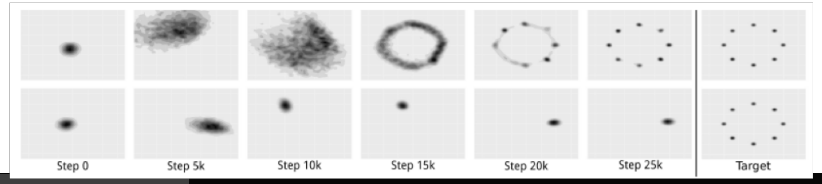
- Vector arithmetic for visual concepts



Unrolled GAN

- Metz, et.al., Unrolled Generative Adversarial Networks, 2017

- Instability of GAN
  - Generator collapse to produce only a single sample or a small family of very similar samples (mode collapse)
  - Generator and discriminator oscillate
  - Generator or discriminator completely wins the other



Unrolled GAN

- Optimal parameter θ_G^* for generator $G(z;\theta_G)$

$$\theta_G^* = \underset{\theta_G}{\operatorname{argmin}} \max_{\theta_D} f(\theta_G, \theta_D)$$
$$= \underset{\theta_G}{\operatorname{argmin}} f(\theta_G, \theta_D^*(\theta_G))$$
$$\theta_D^*(\theta_G) = \underset{\theta_D}{\operatorname{argmax}} f(\theta_G, \theta_D)$$

$$f(\theta_G, \theta_D) = \mathbb{E}_{x\sim p_{data}}[\log(D(x;\theta_D))] + \mathbb{E}_{z\sim\mathcal{N}(0,I)}[\log(1 - D(G(z;\theta_G);\theta_D))]$$

- Typically solved by alternating gradient descent on G and ascent on D.
  - For a fixed D, G would generated data points to which D assigns highest probability.

Unrolled GAN

- Update G considering future response of D

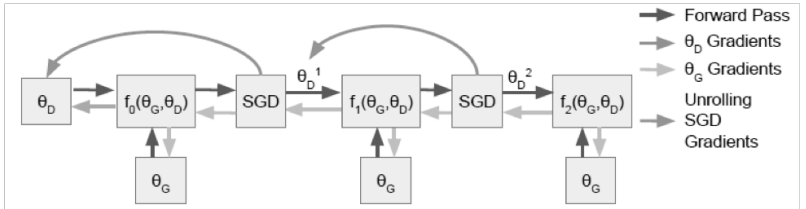


Figure 1: An illustration of the computation graph for an unrolled GAN with 3 unrolling steps. The generator update in Equation 10 involves backpropagating the generator gradient (blue arrows) through the unrolled optimization. Each step k in the unrolled optimization uses the gradients of f_k with respect to θ_D^k, as described in Equation 7 and indicated by the green arrows. The discriminator update in Equation 11 does not depend on the unrolled optimization (red arrow).

Unrolled GAN

- GAN

$$\begin{aligned}\theta_G^* &= \underset{\theta_G}{\operatorname{argmin}} \max_{\theta_D} f(\theta_G, \theta_D) \\ &= \underset{\theta_G}{\operatorname{argmin}} f(\theta_G, \theta_D^*(\theta_G)) \\ \theta_D^*(\theta_G) &= \underset{\theta_D}{\operatorname{argmax}} f(\theta_G, \theta_D),\end{aligned}$$

$$f(\theta_G, \theta_D) = \mathbb{E}_{x \sim p_{data}}[\log(D(x; \theta_D))] + \mathbb{E}_{z \sim \mathcal{N}(0,I)}[\log(1 - D(G(z; \theta_G); \theta_D))]$$

- Unrolled GAN
  - Surrogate loss

$$f_K(\theta_G, \theta_D) = f\left(\theta_G, \theta_D^K(\theta_G, \theta_D)\right)$$

$$\theta_D^0 = \theta_D$$

$$\theta_D^{k+1} = \theta_D^k + \eta^k \frac{\mathrm{d}f(\theta_G, \theta_D^k)}{\mathrm{d}\theta_D^k}$$

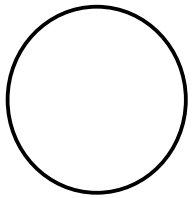
Source of Instability

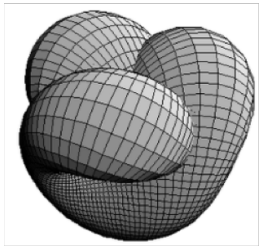
- Arjovsky and Bottou, Towards Principled Methods for Training Generative Adversarial Networks, 2017
- Perfect discriminator theorem
  - Low-dimensional manifolds in high-dimensional space are likely to be disjoint
  - In this case, discriminator can perfectly separate data distribution and model distribution

➔ Generator cannot learn

Manifold

- **Manifold**: a topological space that **locally resembles Euclidean space** near each point.
  - Each point of an n-dimensional manifold has a neighborhood that is homeomorphic to the Euclidean space of dimension n.





Source of Instability

- Original GAN loss $(\log D(X) + \log(1 - D(G(z))))$
  - $\|D - D^*\| < \epsilon$ and $\mathbb{E}_{z \sim p(z)}[\|J_\theta g_\theta(z)\|_2^2] \leq M^2$,

$$\|\nabla_\theta \mathbb{E}_{z \sim p(z)}[\log(1 - D(g_\theta(z)))]\|_2 < M\frac{\epsilon}{1 - \epsilon}$$

  - Suffers from vanishing gradient
- Alternative loss $(\log D(X) - \log D(G(z)))$

$$\mathbb{E}_{z \sim p(z)}[-\nabla_\theta \log D^*(g_\theta(z))|_{\theta=\theta_0}] = \nabla_\theta[KL(\mathbb{P}_{g_\theta} \| \mathbb{P}_r) - 2JSD(\mathbb{P}_{g_\theta} \| \mathbb{P}_r)]|_{\theta=\theta_0}$$

  - JSD has opposite sign
  - KL –divergence is flipped ➔ Causes mode dropping

$$KL(\mathbb{P}_r \| \mathbb{P}_g) = \int \log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x) d\mu(x)$$

Source of Instability

- Remedy: adding noise to P_r and P_g

$$P_{X+\epsilon}(x) = \mathbb{E}_{y\sim\mathbb{P}_X}\left[P_\epsilon(x-y)\right] = \int_{\mathcal{M}} P_\epsilon(x-y)\,\mathrm{d}\mathbb{P}_X(y)$$

  - When $\epsilon \sim N(0,\sigma^2 I)$

$$\mathbb{E}_{z\sim p(z)}\left[\nabla_\theta \log(1-D^*(g_\theta(z)))\right] = \mathbb{E}_{z\sim p(z)}\left[a(z)\int_{\mathcal{M}} P_\epsilon(g_\theta(z)-y)\nabla_\theta\|g_\theta(z)-y\|^2\,\mathrm{d}\mathbb{P}_r(y) - b(z)\int_{\mathcal{P}} P_\epsilon(g_\theta(z)-y)\nabla_\theta\|g_\theta(z)-y\|^2\,\mathrm{d}\mathbb{P}_g(y)\right]$$

- G learns to minimize the loss
  - First term: move G closer data distribution
  - Second term: move G away from model distribution

Wasserstein GAN

- Arjovsky, "Wasserstein Generative Adversarial Networks," 2016
- Conventional distance measure
  - Kullback-Leibler divergence (MLE)

$$KL(\mathbb{P}_r\|\mathbb{P}_g) = \int \log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x)d\mu(x)$$

  - Jensen-Shannon divergence (GAN)

$$JS(\mathbb{P}_r,\mathbb{P}_g) = KL(\mathbb{P}_r\|\mathbb{P}_m) + KL(\mathbb{P}_g\|\mathbb{P}_m)$$

Wasserstein GAN

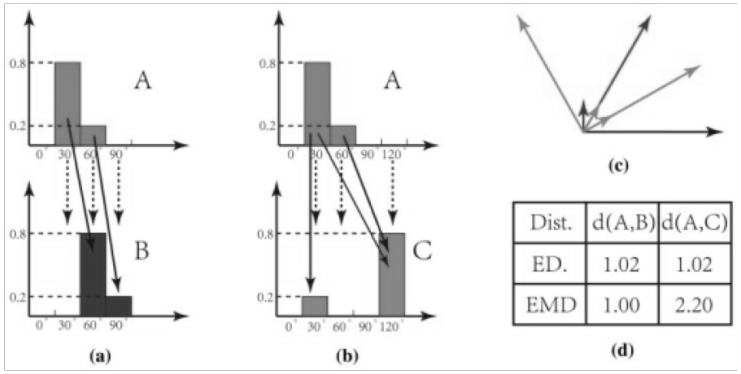
- Wasserstein-1 distance (Earth-Mover distance)

$$W(\mathbb{P}_r,\mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\,\|x-y\|\,\right]$$

where $\Pi(\mathbb{P}_r,\mathbb{P}_g)$ is the set of all joint distributions $\gamma(x,y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. Intuitively, $\gamma(x,y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ into the distribution $\mathbb{P}_g$. The EM distance then is the "cost" of the optimal transport plan.

Earth-Mover Distance

$$W(\mathbb{P}_r,\mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\,\|x-y\|\,\right]$$



Dist.	d(A,B)	d(A,C)
ED.	1.02	1.02
EMD	1.00	2.20

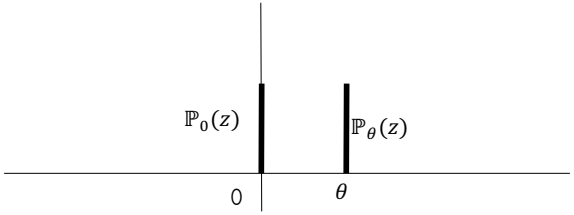
Figure source: https://arxiv.nlm.nih.gov/abs/1701.07875

Wasserstein GAN

- Wasserstein (EM) distance vs. JS divergence
  - Wasserstein metric is sensitive not only to change in probability but also to the geometry of possible outcomes.

Ex) $z \sim U[0,1], \mathbb{P}_0(z)$: distribution over $(0, z) = \mathbb{R}^2$

$\mathbb{P}_\theta(z)$: distribution over $(\theta, z) = \mathbb{R}^2$



Wasserstein GAN

- Wasserstein (EM) distance vs. JS divergence

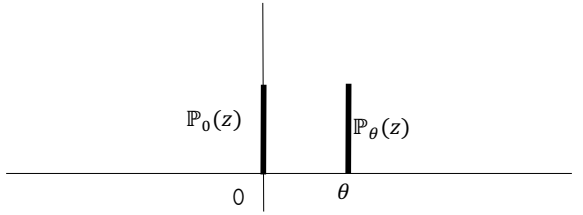


Figure 1: These plots show $\rho(\mathbb{P}_\theta, \mathbb{P}_0)$ as a function of θ when ρ is the EM distance (left plot) or the JS divergence (right plot). The EM plot is continuous and provides a usable gradient everywhere. The JS plot is not continuous and does not provide a usable gradient.

Wasserstein GAN

- Wasserstein distance (Earth-Mover distance)

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\, \|x - y\| \,\right]$$

- Kantorovich-Rubinstein duality

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

  - For a K-Lipschitz function $f_w: X \to \mathbb{R}$ (called 'critic')
    - f_w should be smoothly changing function

$$\max_{w\in\mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z))]$$

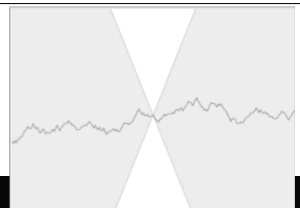
Lipschitz Continuity

- Intuitively, a Lipschitz continuous function is limited in how fast it can change [Wikipedia]

Given two metric spaces (X, d_X) and (Y, d_Y), where d_X denotes the metric on the set X and d_Y is the metric on set Y, a function $f: X \to Y$ is called **Lipschitz continuous** if there exists a real constant $K \geq 0$ such that, for all x_1 and x_2 in X,

$$d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2).$$[3]

Any such K is referred to as **a Lipschitz constant** for the function f.



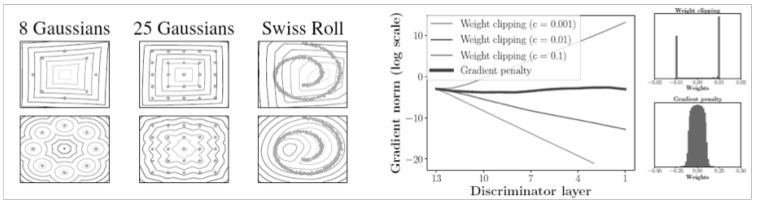
Wasserstein GAN

- WGAN learning (α=0.00005, c=0.01, m = 64, n_{critic}=5)

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of priors.
    $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

WGAN with Gradient Penalty

- WGAN applies weight clipping to ensure K-Lipschitz continuity of generator



- WGAN-GP

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]}_{\text{Our gradient penalty}}.$$

Cramer GAN

- Bellemare, et.al., The Cramer Distance as a Solution to Biased Wasserstein Gradients, 2017.

- Ideal divergence (S, I)
  - Scale sensitive (of order β)

$$\mathbf{d}(cX, cY) \leq |c|^{\beta} \mathbf{d}(X, Y)$$

  - Sum invariant

$$\mathbf{d}(A + X, A + Y) \leq \mathbf{d}(X, Y)$$

- Unbiased sample gradient (U)

$$\mathop{\mathbb{E}}_{\mathbf{X}_m \sim P} \nabla_\theta \mathbf{d}(\hat{P}_m, Q_\theta) = \nabla_\theta \mathbf{d}(P, Q_\theta)$$

Cramer GAN

- KL divergence has unbiased sample gradients, but is not scale sensitive.
- Wasserstein metric is ideal (I, S), but does not have unbiased sample gradients.
- Cramer distance satisfies all of the three properties

$$l_2^2(P, Q) := \int_{-\infty}^{\infty} (F_P(x) - F_Q(x))^2 \mathrm{d}x$$

  - Square root of Cramer distance

$$l_p(P, Q) := \left(\int_{-\infty}^{\infty} |F_P(x) - F_Q(x)|^p \mathrm{d}x \right)^{1/p}$$

    - Dual form

$$l_p(P, Q) = \sup_{f \in \mathbb{F}_q} \left| \mathop{\mathbb{E}}_{x \sim P} f(x) - \mathop{\mathbb{E}}_{x \sim Q} f(x) \right|$$

Cramer GAN

Algorithm 1: Cramér GAN Losses.

Defaults: The gradient penalty coefficient $\lambda = 10$.
Sample $x_r \sim P$ a real sample.
Sample $x_g, x'_g \sim Q$ two independent generator samples.
Sample $\epsilon \sim \text{Uniform}(0, 1)$ a random number.
Interpolate real and generated samples:
$\hat{x} = \epsilon x_r + (1 - \epsilon) x_g$
Define the critic:
$f(x) = \left\| h(x) - h(x'_g) \right\|_2 - \|h(x)\|_2$
Compute the generator loss:
$L_g = \|h(x_r) - h(x_g)\|_2 + \|h(x_r) - h(x'_g)\|_2 - \|h(x_g) - h(x'_g)\|_2$
Compute the surrogate generator loss:
$L_{surrogate} = f(x_r) - f(x_g)$
Compute the critic loss:
$L_{critic} = -L_{surrogate} + \lambda(\|\nabla_{\hat{x}} f(\hat{x})\|_2 - 1)^2$

EBGAN (Energy-based GAN)

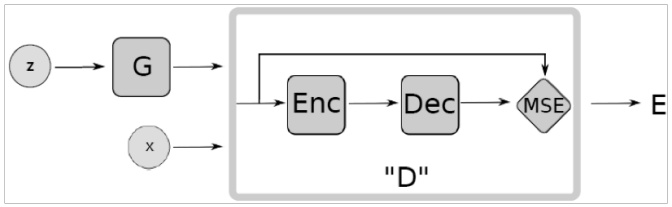
- Zhao, Energy-based Generative Adversarial Networks, Mar. 2017.
  - Energy-based models

$$P(X) = \frac{e^{-Energy(X)}}{Z} = \frac{e^{-Energy(X)}}{\sum_X e^{-Energy(X)}}$$

  - Discriminator as an energy function
    - The regions of high data density ➔ lower energy
    - Outside these regions ➔ higher energy
  - Advantages
    - Can convert energies into probabilities through a Gibbs distribution
    - Provides greater flexibility in the choice of architecture of the discriminator and the training procedure

EBGAN (Energy-based GAN)

- Auto-encoder discriminator



$$D(x) = ||Dec(Enc(x)) - x||$$

EBGAN

Figure 6: Generation from the CelebA dataset. Left(a): DCGAN generation. Right(b): EBGAN-PT generation.

BEGAN

- Berthelot, et.al., “BEGAN: Boundary Equilibrium Generative Adversarial Networks,” Apr. 2017.
  - Based on W-distance and auto-encoder based GAN
  - Balances generator and discriminator during training
  - Equilibrium between generator and discriminator

$$\gamma = \frac{\mathbb{E}\left[\mathcal{L}(G(z))\right]}{\mathbb{E}\left[\mathcal{L}(x)\right]}$$

  - Training objective

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - k_t.\mathcal{L}(G(z_D)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(G(z_G)) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k(\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))) & \text{for each training step } t \end{cases}$$

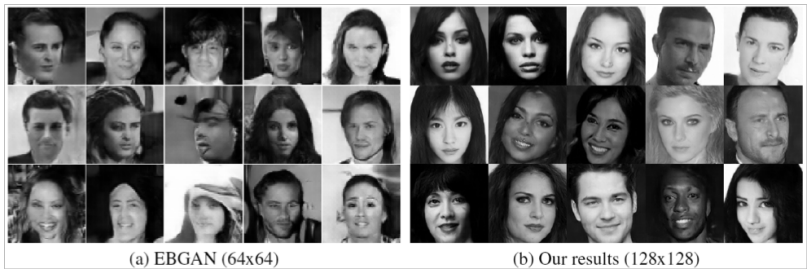
BEGAN

- Berthelot, et.al., BEGAN: Boundary Equilibrium Generative Adversarial Networks, Apr. 2017.

Figure 3: Random 64x64 samples at varying $\gamma \in \{0.3, 0.5, 0.7\}$

BEGAN

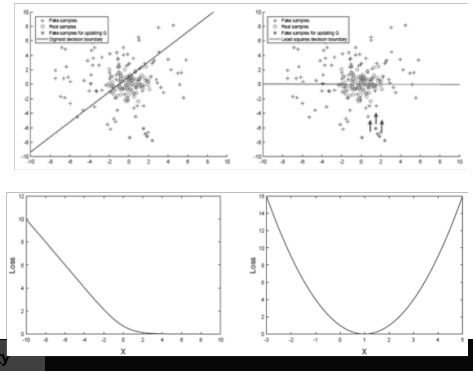
- Berthelot, et.al., “BEGAN: Boundary Equilibrium Generative Adversarial Networks,” Apr. 2017.



(a) EBGAN (64x64) (b) Our results (128x128)

LS-GAN (Least Square GAN)

- Mao, et.al., “Least Squares Generative Adversarial Networks,” Apr, 2017
  - In training generator of regular GAN, fake samples on correct side of decision boundary get small penalty



LS-GAN (Least Square GAN)

- LS-GAN

$$\min_D V_{\mathrm{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x})-b)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z}))-a)^2\right]$$
$$\min_G V_{\mathrm{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z}))-c)^2\right],$$

- Advantages
  - Better quality (closer to real samples)
  - Stable during learning process
    - Relieves from problem of vanishing gradients

LS-GAN (Least Square GAN)

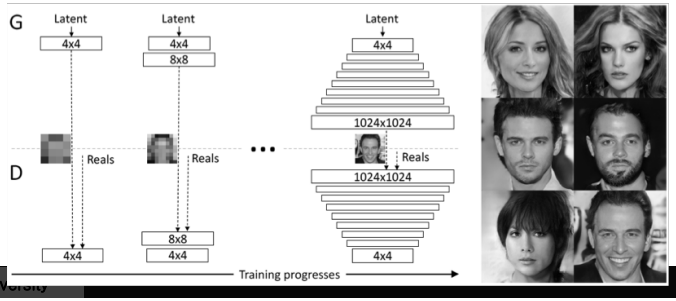
(a) Generated by LSGANs.

(b) Generated by DCGANs (Reported in [11]).

(c) Generated by EBGANs (Reported in [26]).

ProgressiveGAN

- Karras, et.al., Progressive Growing of GANs for Improved Quality, Stability, and Variation, Nov., 2017. (ICPR2018)
  - Grow both the generator and discriminator progressively
  - Starting from a low resolution, add new layers that model increasingly fine details as training progresses



SN-GAN

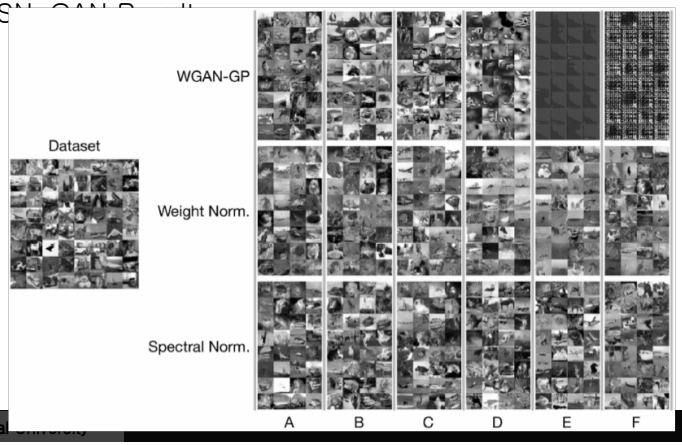
- T. Miyato, et al., “SPECTRAL NORMALIZATION FOR GENERATIVE ADVERSARIAL NETWORKS,” ICLR2018
  - Normalization by spectral norm to satisfy Lipschitz constraint

$$\bar{W}_{\mathrm{SN}}(W) := W/\sigma(W)$$

  - Fast approximation of spectral norm by power iteration method

SN-GAN

- T. Miyato, et al., “SPECTRAL NORMALIZATION FOR GENERATIVE ADVERSARIAL NETWORKS,” ICLR2018
  - SN-GAN Results



SA-GAN

- H. Zhang, et al., “Self-Attention Generative Adversarial Networks,” May. 2018

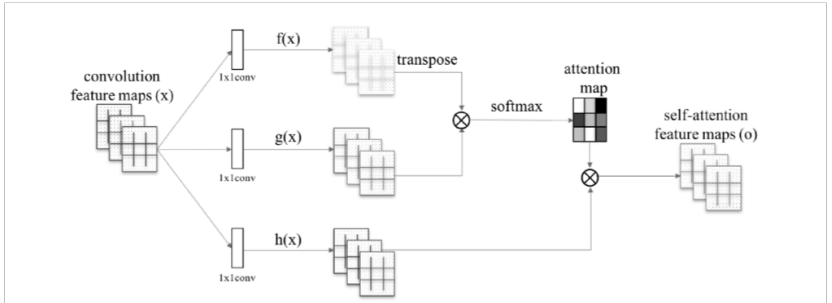


Figure 2: The proposed self-attention mechanism. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

SA-GAN

- H. Zhang, et al., “Self-Attention Generative Adversarial Networks,” May. 2018
  - Self-attention

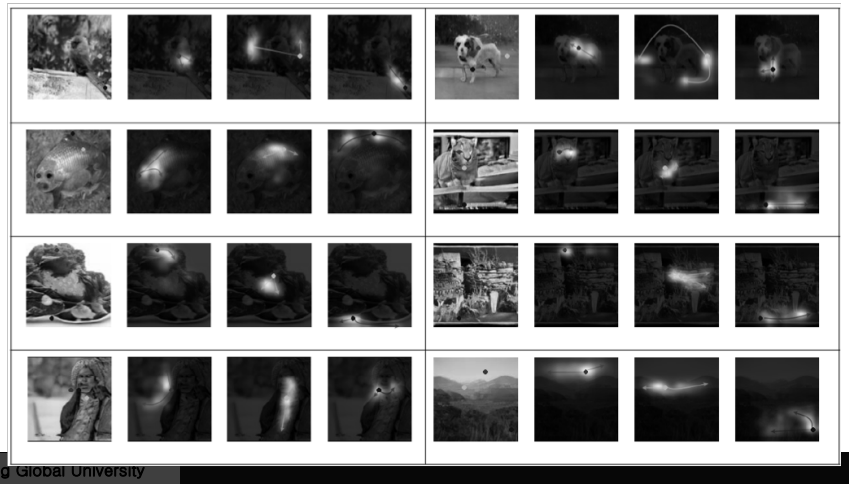
$$\beta_{j,i} = \frac{\exp(s_{ij})}{\sum_{i=1}^{N} \exp(s_{ij})}, \text{where } s_{ij} = \boldsymbol{f}(\boldsymbol{x_i})^T \boldsymbol{g}(\boldsymbol{x_j})$$

  - Output of attention layer

$$\boldsymbol{o_j} = \sum_{i=1}^{N} \beta_{j,i} \boldsymbol{h}(\boldsymbol{x_i}), \text{where } \boldsymbol{h}(\boldsymbol{x_i}) = \boldsymbol{W_h} \boldsymbol{x_i}$$

SA-GAN

- Visualization of attention maps



SA-GAN

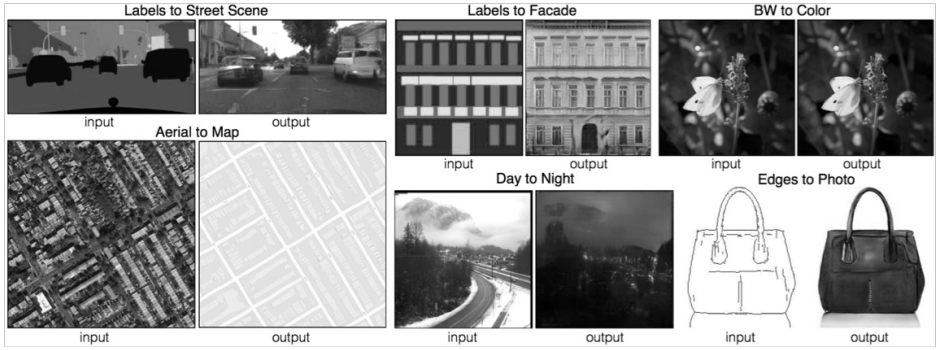
BigGAN

- A. Brock, et al., "LARGE SCALE GAN TRAINING FOR HIGH FIDELITY NATURAL IMAGE SYNTHESIS" 2018.
  - Large-scale GAN training using large batch
  - Truncation trick for random noise generation
    - Trade-off between variety and fidelity)
  - Orthogonal regularization to the generator

Figure 1: Class-conditional samples generated by our model.

Image-to-Image Translation

- Isola, et al., "Image-to-Image translation using conditional GAN," 2016

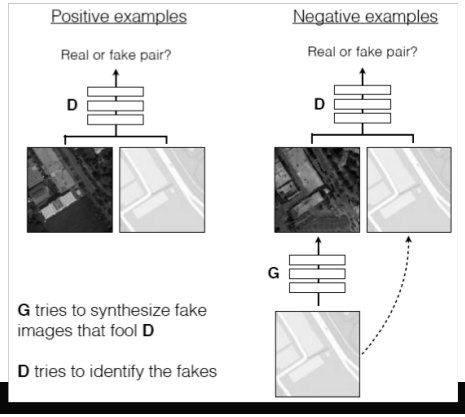


Conditional GAN

- Objective of conditional GAN

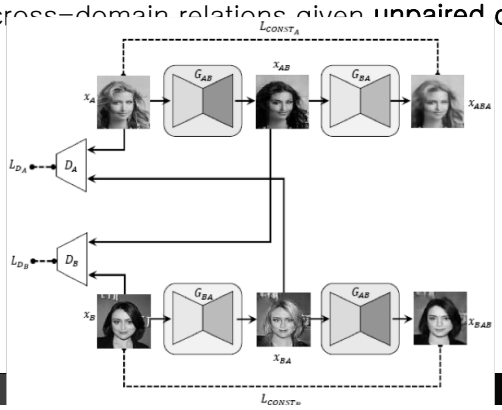
$$\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{x,y\sim p_{data}(x,y)}[\log D(x,y)] + \mathbb{E}_{x\sim p_{data}(x),z\sim p_z(z)}[\log(1-D(x,G(x,z)))]$$

  - z: noise
  - y: output image
  - x: observed image (condition image)



DiscoGAN

- T. Kim, et. al., Learning to Discover Cross-Domain Relations with Generative Adversarial Networks, May, 2017.
  - Learns cross-domain relations given unpaired data



StarGAN

- StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation [Choi2017]

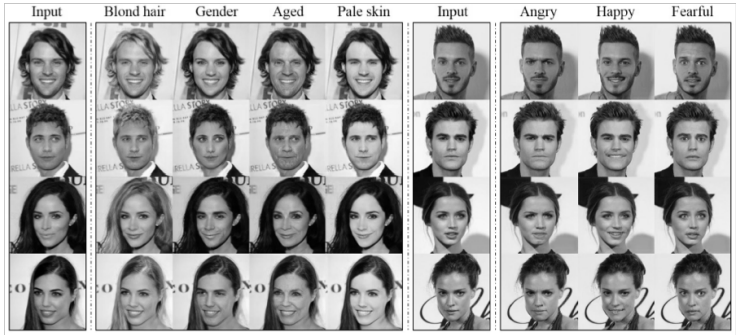


Figure 1. Multi-domain image-to-image translation results on the CelebA dataset via transferring knowledge learned from the RaFD dataset. The first and sixth columns show input images while the remaining columns are images generated by StarGAN. Note that the images are generated by a single generator network, and facial expression labels such as angry, happy, and fearful are from RaFD, not CelebA.

StarGAN

- Image-to-image translation for multiple domains using a single model

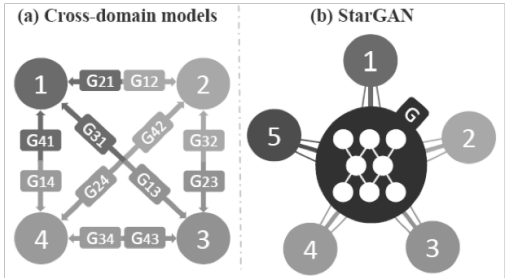


Figure 2. Comparison between cross-domain models and our proposed model, StarGAN. (a) To handle multiple domains, cross-domain models should be built for every pair of image domains. (b) StarGAN is capable of learning mappings among multiple domains using a single generator. The figure represents a star topology connecting multi-domains.

StarGAN

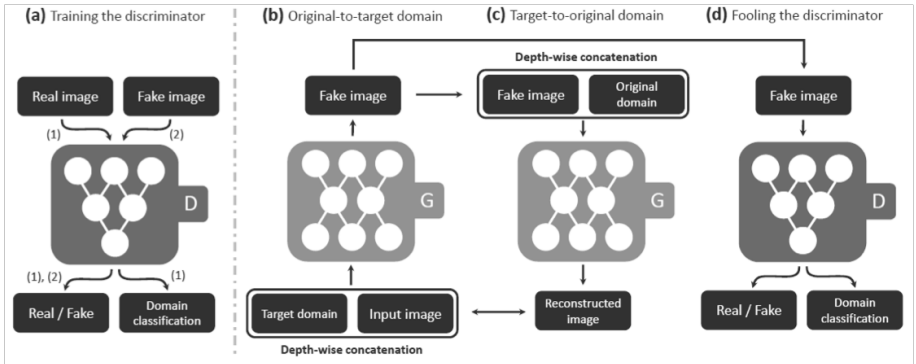
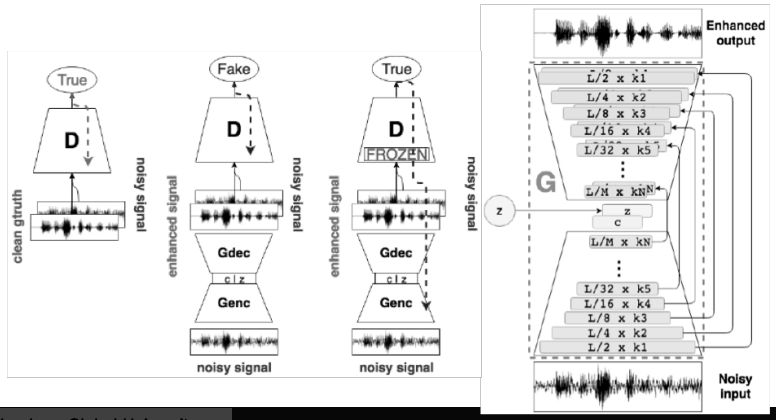


Figure 3. Overview of StarGAN, consisting of two modules, a discriminator D and a generator G. **(a)** D learns to distinguish between real and fake images and classify the real images to its corresponding domain. **(b)** G takes in as input both the image and target domain label and generates an fake image. The target domain label is spatially replicated and concatenated with the input image. **(c)** G tries to reconstruct the original image from the fake image given the original domain label. **(d)** G tries to generate images indistinguishable from real images and classifiable as target domain by D.

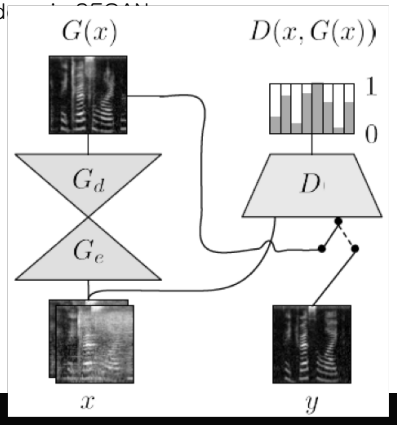
GAN for Speech: SEGAN

- S. Pascua, et.al., "SEGAN: Speech Enhancement Generative Adversarial Network," 2017



GAN for Speech: FSEGAN

- Donahue and Prabhavalkar, "EXPLORING SPEECH ENHANCEMENT WITH GENERATIVE ADVERSARIAL NETWORKS FOR ROBUST SPEECH RECOGNITION," 2017
  - Frequency-d

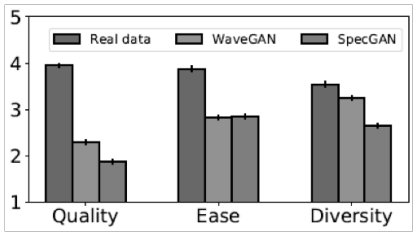


Time Domain vs. Frequency Domain

- Donahue, et.al., "Synthesizing Audio with Generative Adversarial Networks," 2018.
  - WaveGAN: time domain, SpecGAN: frequency domain

Experiment	Inception score	$\lvert D\rvert_{\text{self}}$	$\lvert D\rvert_{\text{train}}$	Noise
Real (train)	9.18 ± 0.04	1.1	0.0	1.0
Real (test)	8.01 ± 0.24	1.0	1.0	1.0
Parametric	5.02 ± 0.06	0.7	1.1	0.2
WaveGAN	4.12 ± 0.03	1.4	2.0	1.2
+ Phase shuffle $n = 2$	4.67 ± 0.01	0.8	2.3	1.8
+ Phase shuffle $n = 4$	4.54 ± 0.03	1.0	2.3	1.5
+ Nearest neighbor	3.77 ± 0.02	1.8	2.6	1.4
+ Linear interpolation	2.88 ± 0.02	1.7	2.7	1.2
+ Cubic interpolation	2.60 ± 0.01	1.3	3.6	1.1
+ Post-processing	3.92 ± 0.03	1.4	2.9	1.0
+ DCGAN/BN	2.01 ± 0.01	0.9	4.3	0.0
+ Dropout	3.93 ± 0.03	1.0	2.6	1.6
SpecGAN	6.03 ± 0.04	1.1	1.4	1.5
+ Phase shuffle $n = 1$	3.71 ± 0.03	0.8	1.6	3.1

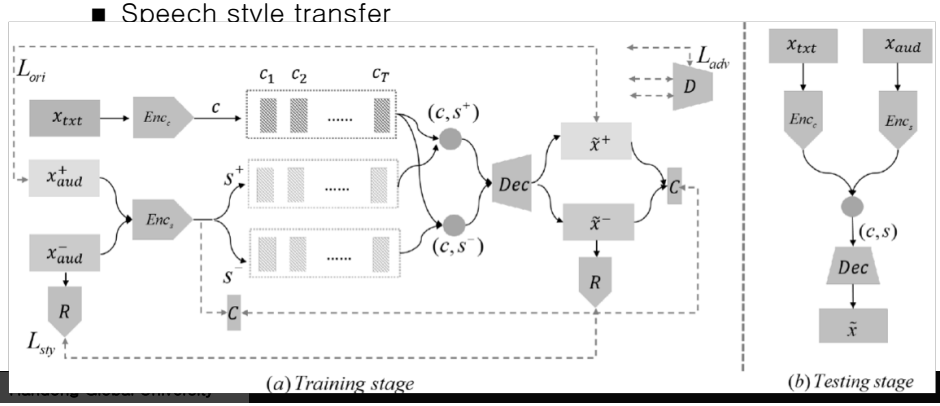
Discriminative task
(Inception score)



Generative task
(Mean opinion scores)

TTS-GAN

- Ma, et al, "TTS-GAN: A generative adversarial network for style modeling in a text-to-speech system," ICLR2019
  - Speech style transfer



Text GAN

- Zhang, et.al., "Generating Text via Adversarial Training", 2016.

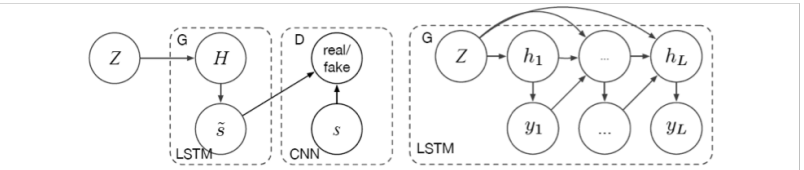
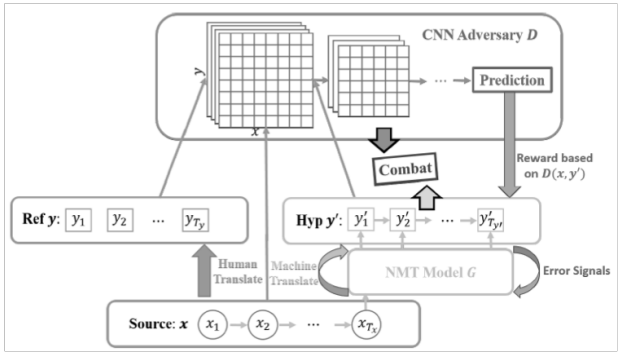


Figure 1: Left: Illustration of the textGAN model. The discriminator is a CNN, the sentence decoder is an LSTM. Right: the structure of LSTM model

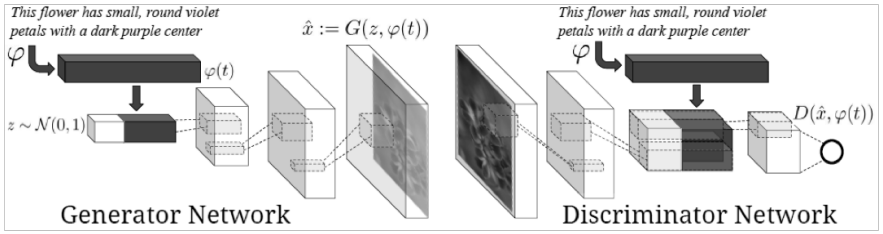
Adversarial Neural Machine Translation

- Wu, et.al., "Adversarial Neural Machine Translation", Apr. 2017.



Generative Adversarial Text to Image Synthesis

- Reed, et.al., "Generative Adversarial Text to Image Synthesis," 2016.

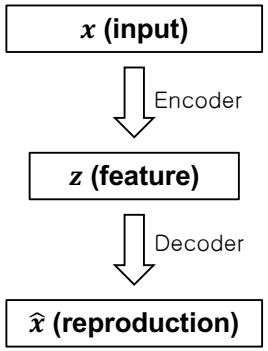


Agenda

- Introduction
- Generative Adversarial Nets
- Variational Auto-Encoders
- Pixel RNN/CNN
- Q&A

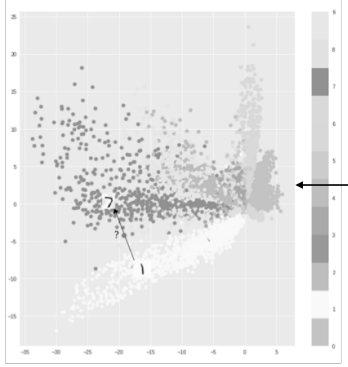
Auto-Encoders

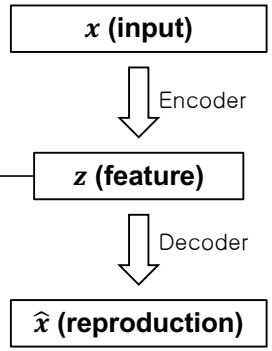
- Auto-Encoders
  - Composed of Encoder and Decoder
    - Usually, composed neural nets
  - Reproduces input
    - $Loss = \|\hat{x} - x\|^2$
- Useful for
  - Feature learning
  - Pre-training of deep neural nets



Latent Space of Auto-Encoders

- Latent space of vanilla auto-encoder is discontinuous ➔ not appropriate for sample generation



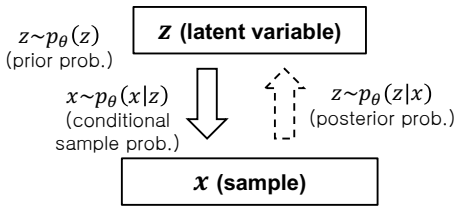


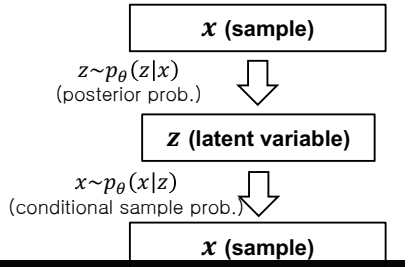
cf. However, VAE learns continuous latent space!

Figure source: Blog "Intuitively Understanding Variational Autoencoders" by Irhum Shafkat

Variational Auto-Encoders

- Extension of auto-encoder that can generate diverging samples
  - Learns **distribution** of samples
- A graphical model with a set of parameters θ to generate sample x from latent variable z.
  - In practice, z and θ are unknown

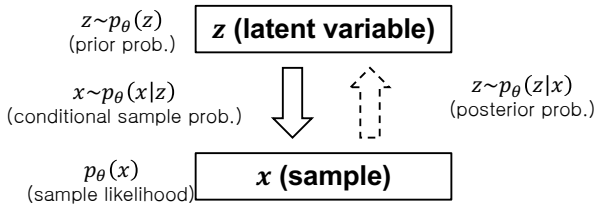




Variational Auto-Encoders

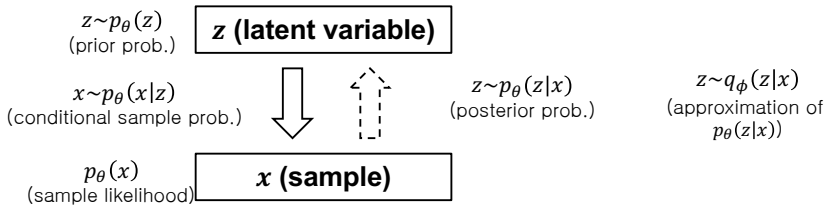
- $p_\theta(z)$: model by tractable dist.(eg. Gaussian)
- $p_\theta(x|z)$: model by a parametric function (eg. neural net)
- $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$: intractable
- $p_\theta(z|x)$: intractable

➔ Direct MLE is not possible ($\theta^* = \underset{\theta}{\operatorname{argmax}}\, p_\theta(x)$)



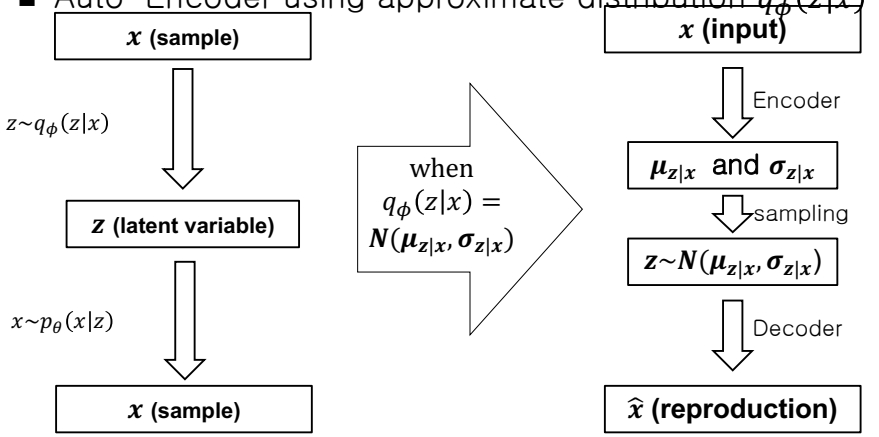
Variational Learning

- Approximate $p_\theta(z|x)$ by a tractable function. $q_\phi(z|x)$
- Jointly learn θ and ϕ to maximize **variational lower bound (ELBO)** of $\log p_\theta(x)$



Structure of VAE

- Auto-Encoder using approximate distribution $q_\phi(z|x)$



Variational Lower Bound

- Variational learning

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

Variational Lower Bound

- Variational learning

$$p_\theta(z|x^{(i)}) = \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(x^{(i)})}$$

$$p_\theta(x^{(i)}) = \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}$$

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

Variational Lower Bound

- Variational learning

$$p_\theta(z|x^{(i)}) = \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(x^{(i)})}$$

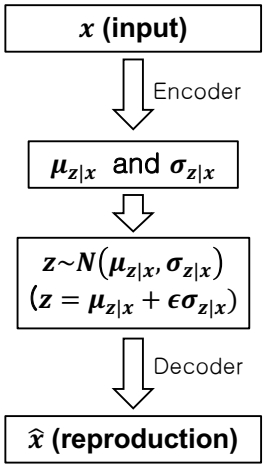
$$p_\theta(x^{(i)}) = \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}$$

$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}
\end{aligned}$$

[slide is from cs231n]

Re-parameterization Trick

- Sampling z from $q_\phi(z|x)$
  - Produce z by a deterministic function $z = g_\phi(\epsilon, x)$, where $\epsilon \sim p(\epsilon)$
- In VAE, $z \sim N(\mu, \sigma^2)$ and $\epsilon \sim N(0,1)$, then $z = \mu + \epsilon\sigma$
  - Encoder produces $\mu_{z|x}$ and $\sigma_{z|x}$
  - Sample z from $N(\mu_{z|x}, \sigma_{z|x})$
    $\boldsymbol{z = \mu_{z|x} + \epsilon\sigma_{z|x}}$
  - Decoder generates $\hat{\boldsymbol{x}}$ from z



Variational Lower Bound

- Variational learning

$$p_\theta(z|x^{(i)}) = \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(x^{(i)})}$$

$$p_\theta(x^{(i)}) = \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}$$

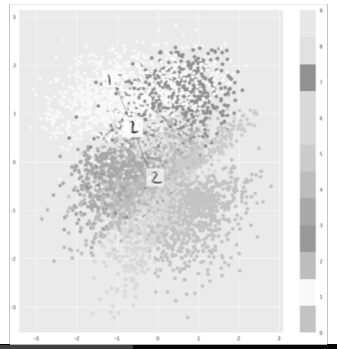
$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}
\end{aligned}$$

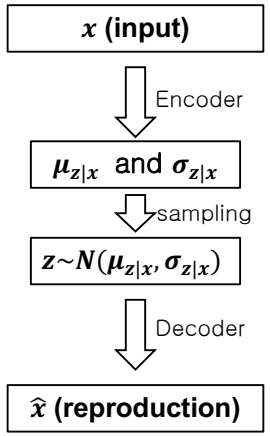
$$\mathcal{L}(\theta,\phi;x^{(i)}) = \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(x^{(i)}|z) + \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log\left(\left(\sigma_j^{(i)}\right)^2\right) - \left(\mu_j^{(i)}\right)^2 - \left(\sigma_j^{(i)}\right)^2\right)$$

[slide is from cs231n]

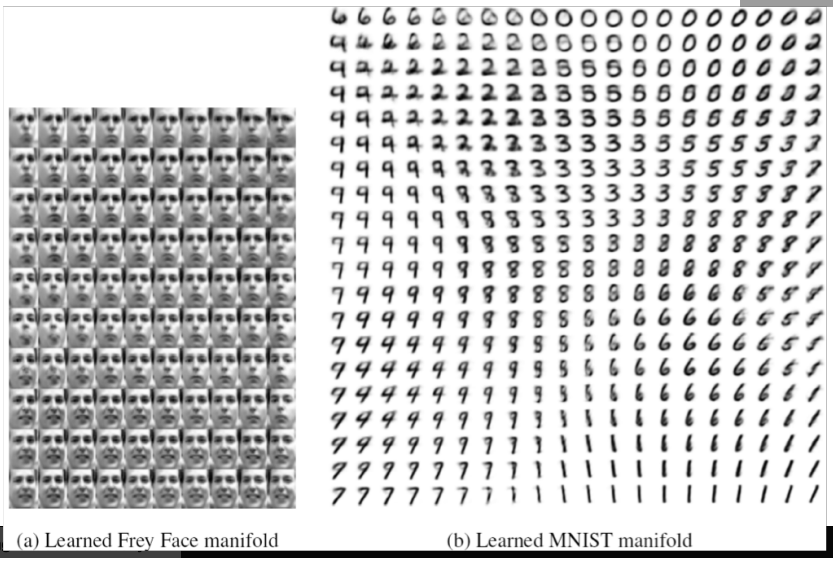
Latent Space of VAE

- VAE learns continuous (densely packed) latent space.
  - Reconstruction loss
    + KL divergence loss





Samples Generated By VAE



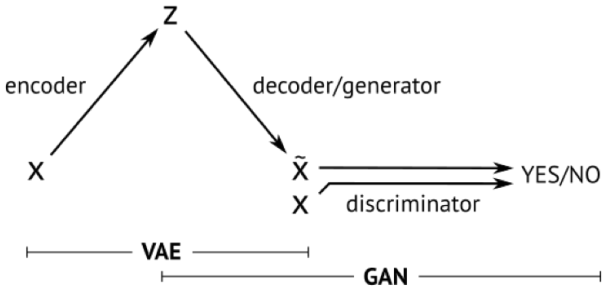
(a) Learned Frey Face manifold (b) Learned MNIST manifold

Samples Generated by VAE

Image source:
https://github.com/torch/tor

Adversarial VAE (VAE/GAN)

- A. Larsen, et.al., "Autoencoding beyond pixels using a learned similarity metric," 2016.



Adversarial VAE (VAE/GAN)

- A. Larsen, et.al., "Autoencoding beyond pixels using a learned similarity metric," 2016.

Figure 3. Samples from different generative models.

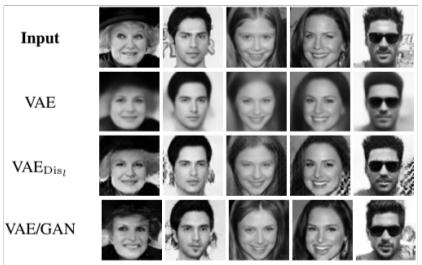


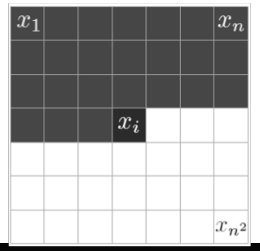
Figure 4. Reconstructions from different autoencoders.

Agenda

- Introduction
- Generative Adversarial Nets
- Variational Auto-Encoders
- Pixel RNN/CNN
- Q&A

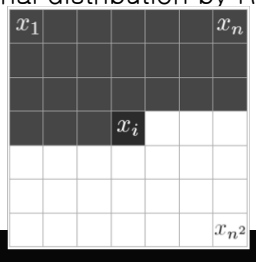
Generating Image Pixels

- Probability of an image $x = (x_1, x_2, \cdots, x_N)$, $N = n^2$
  - $p(x) = p(x_1, x_2, \cdots, x_N)$
    $= p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \dots p(x_N|x_1, \dots, x_{N-1})$
    $= \prod_{i=1}^{N} p(x_i|x_1, \dots, x_{i-1})$
    ➔ Each pixel follows a conditional distribution given previous pixels

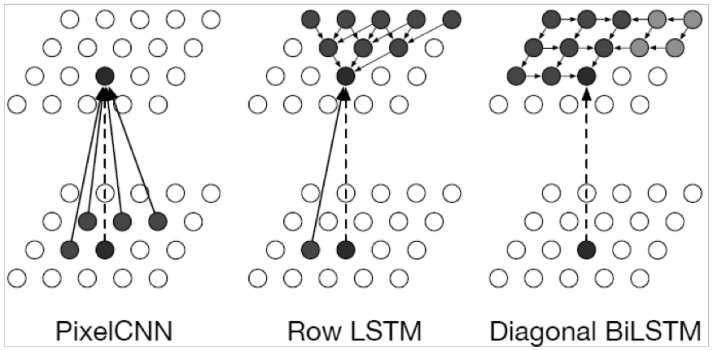


Generating Image Pixels

- Probability of an image $x = (x_1, x_2, \cdots, x_N)$, $N = n^2$
  - $p(x) = \prod_{i=1}^{N} p(x_i|x_1, \dots, x_{i-1})$
- Idea of Pixel CNN/RNN
  - Generate each pixel from $p(x_n|x_1, \dots, x_{n-1})$
    ➔ autoregressive model
  - Learn conditional distribution by RNN or CNN



Pixel CNN/RNN



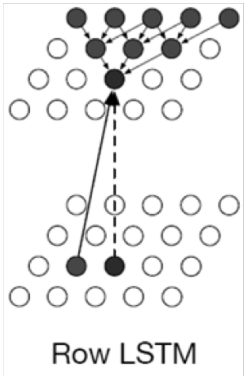
Pixel RNN (Row LSTM)

- Designed to parallelize computation

$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma(\mathbf{K}^{ss} \circledast \mathbf{h}_{i-1} + \mathbf{K}^{is} \circledast \mathbf{x}_i)$$
$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$
$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$

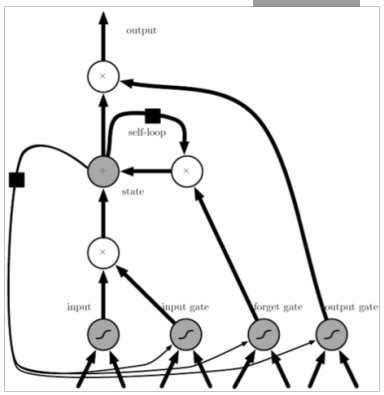
  - h_{i-1}, c_{i-1}: hidden/cell state
  - x_i: row i of input map
  - K^{ss}, K^{is}: s2s, i2s weight

- Takes a row as input vector
- Input-to-state / state-to-state connections are row-wise convolution



LSTM Networks

- State unit $s_i^{(t)}$
  - Weight of **self-loop** is controlled by **forget gate**
  - Weight of **state candidate** is controlled by **input gate**



Forget gate

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

Input gate

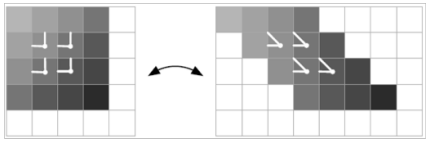
$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right)$$

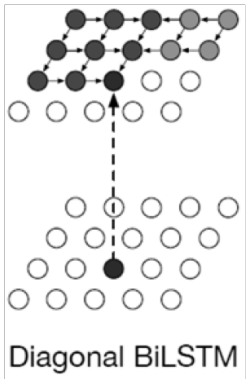
State update

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right)$$

Pixel RNN (Diagonal BiLSTM)

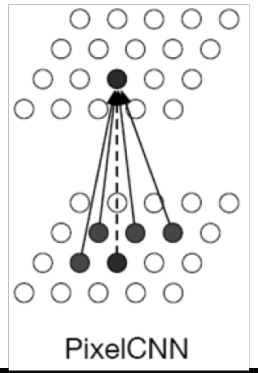
- Designed to parallelize computation and capture entire available context
  - Two directions of layer scans
    - Top-left corner to bottom-right corner
    - Top-right corner to bottom-left corner
- 2x1 column-wise convolution





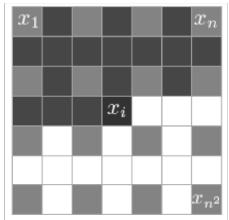
Pixel CNN

- Designed to faster training
  - Make receptive field large, but not unbounded
  - Masked convolution



Multi-Scale Pixel RNN

- Two steps
  1. Unconditional Pixel RNN generates s * s subsampled image
  2. Conditional Pixel RNN generates large n * n image conditioned on s * s image
     - Up-sampling by deconv. + biasing



Multi-scale context

Architectures of Pixel CNN/RNN

PixelCNN	Row LSTM	Diagonal BiLSTM
7×7 conv mask A		
Multiple residual blocks: (see fig 5)		
Conv 3×3 mask B	Row LSTM i-s: 3×1 mask B s-s: 3×1 no mask	Diagonal BiLSTM i-s: 1×1 mask B s-s: 1×2 no mask
ReLU followed by 1×1 conv, mask B (2 layers)		
256-way Softmax for each RGB color (Natural images) or Sigmoid (MNIST)		

Samples Generated by Pixel RNN

- 32x32 images generated by Pixel RNN

CIFAR-10 ImageNet

Emerging Topics in Deep Learning Part 2

Injung Kim
Handong Global University
2019. 1. 9.

Agenda

- Network Architecture Search
- Building Lightweight Models
- Non-local Neural Nets

Network Architecture Search

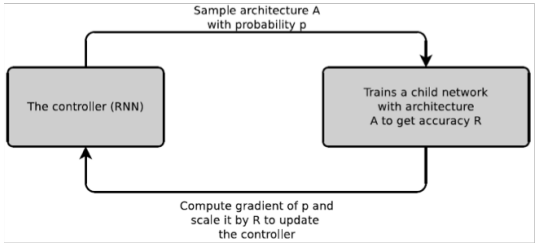
- Deep learning = Experts + Data + Computing
  - An important role of experts: architecture design
- NAS: automatic search of network architecture
  - Requires heavy computation
  - Outperforms manually designed models in many tasks

Network Architecture Search

- Define a target task
  - A training set + an evaluation set
- Design representation of network architectures (NAS space)
  - A sequence of hyper-parameters (usually, non-contiguous)
    Ex) layer = (layer type, input, filter size, stride, # of channels, etc.)
  - Recent trend: focus on "cell" rather than the whole network
- Repeat
  - Generate an architecture (hyper-parameters)
  - Evaluate the performance of the architecture
    - Time-consuming
- Major approaches
  - Reinforcement learning, evolutionary algorithm, heuristic search, etc.

NAS

- B. Zoph and Q. Le, "Neural Architecture Search with Reinforcement Learning," ICLR2017 (Nov. 2016)
  - Controller network (RNN) searches a child network that maximizes performance on evaluation data
  - NAS-net (error rate of 3.65% on CIFAR-10)



NAS

- Hyper-parameters of a network by a sequence of tokens
  Ex) filter height, filter width, stride height, stride width, and number of filters
- Controller RNN generates hyper-parameters as a sequence of tokens

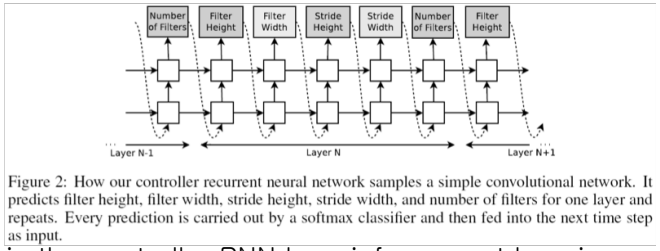
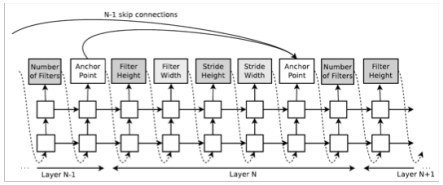


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step as input.

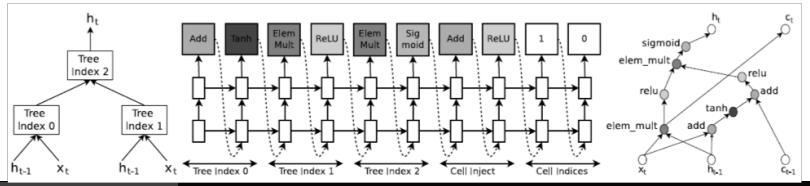
- Train the controller RNN by reinforcement learning

NAS

- Generating network containing skip connection

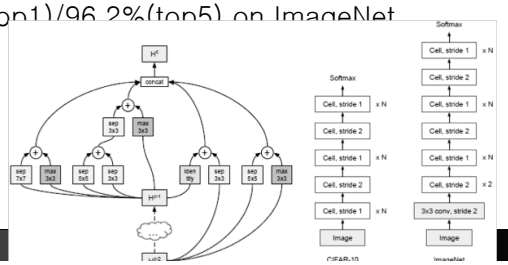


- Generation of recurrent cell



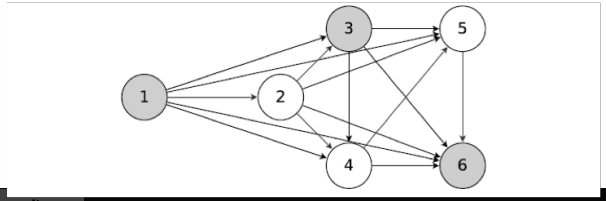
PNAS

- C. Liu, et al., "Progressive Neural Architecture Search," Mar. 2018.
  - **Heuristic search** to search the space of cell structures
    - Simple to complex models
  - Performance prediction with surrogate model
    - LSTM or MLP
  - Error rate of 3.41% on CIFAR-10
  - 82.9%(top1)/96.2%(top5) on ImageNet



ENAS

- H. Pham, et al., "Efficient Neural Architecture Search via Parameter Sharing," Feb. 2018
  - A controller discovers network architecture
    - Autoregressive LSTM with 100 hidden units
  - The controller is trained with policy gradient
  - Improve efficiency by **sharing parameters among child models** (1000x less expensive than NAS)
  - 2.89% test error on CIFAR-10



NASNet

- B. Zoph, et al., "Learning Transferable Architectures for Scalable Image Recognition," Apr. 2018
  - The design of a new search space which enables transferability.
  - A new regularization technique called ScheduledDropPath
  - 2.4% error on CIFAR-10
  - 82.7%(top1) / 96.2%(top5) on ImageNet

DART

- Anonymous authors, "DART: Differentiable Architecture Search," (under review at ICLR2019), June.2018.

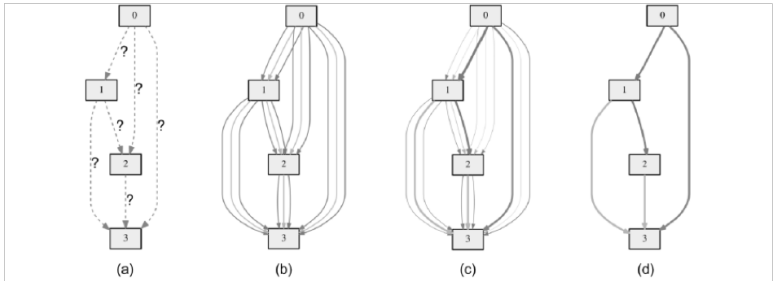


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

DART

- Performance of DART
  - DARTS achieved comparable results with the state of the art (Zoph et al., 2018; Real et al., 2018) while using three orders of magnitude less computation resources

Table 1: Comparison with state-of-the-art image classifiers on CIFAR-10 (lower error rate is better). Results marked with † were obtained by training the corresponding architectures using our setup.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	manual
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	2000	RL
NASNet-A + cutout (Zoph et al., 2018)†	2.83	3.1	2000	RL
AmoebaNet-A (Real et al., 2018)	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	evolution
Hierarchical evolution (Liu et al., 2018b)	3.75 ± 0.12	15.7	300	evolution
PNAS (Liu et al., 2018a)	3.41 ± 0.09	3.2	225	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	RL
Random search baseline‡ + cutout	3.29 ± 0.15	3.2	4	random
DARTS (first order) + cutout	3.00 ± 0.14	3.3	1.5	gradient-based
DARTS (second order) + cutout	2.76 ± 0.09	3.3	4	gradient-based

‡ Best architecture among 24 samples according to the validation error after 100 training epochs.

Agenda

- Network Architecture Search
- Building Lightweight Models
- Non-local Neural Nets

Building Efficient Deep Learning Models

- SqueezeNet (Nov. 2016)
- MobileNet (Apr. 2017)
- ShuffleNet (Dec. 2017)
- MobileNet.v2 (Apr. 2018)

SqueezeNet

- Iandola, et. al., “SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE”, 2016.
- Design strategies
  1. Replace 3x3 filters with 1x1 filters
  2. Decrease # of input channels to 3x3 filters
  3. Down-sample late in the network

  1, 2 are to reduce networks, 3 is to improve accuracy
- Fire module, residual connection

SqueezeNet

- Fire module

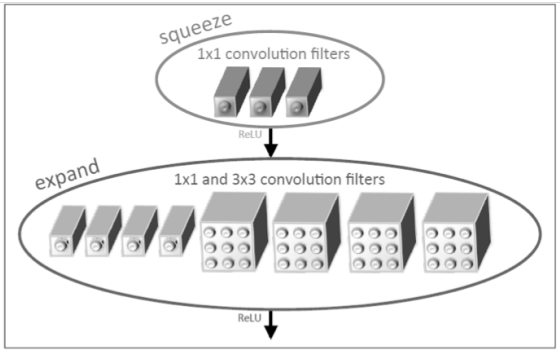


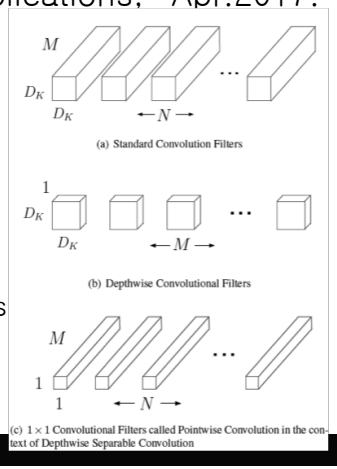
Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

MobileNet

- Howard, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr.2017.

- Depth-wise separable conv.
  - Standard convolution
    - $D_K D_K M N D_F D_F$
  - ➔ Depthwise convolution
    + Pointwise convolution
    - $D_K D_K M D_F D_F + M N D F D F$
  - Less computation and parameters

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$



ShuffleNet

- Zhang, et.al., "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices", Dec. 2017
  - Depth-wise separable convolution
  - Group convolution + channel shuffling

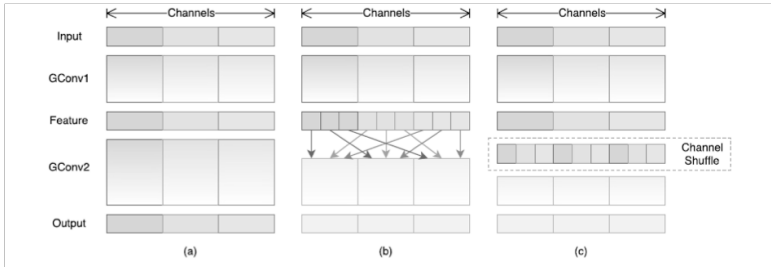
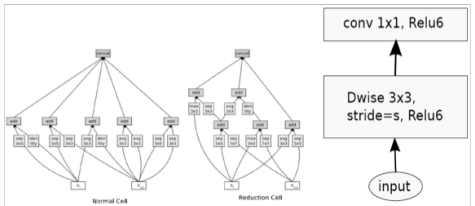


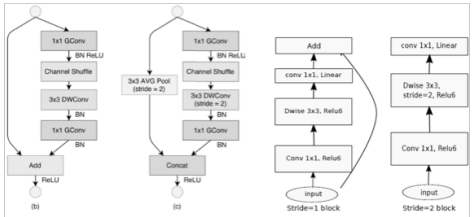
Figure 1. Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and

MobileNet v2

- Sandler, et.al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks", Apr. 2018
  - Depthwise separable convolution
  - Linear bottlenecks
  - Inverted residuals



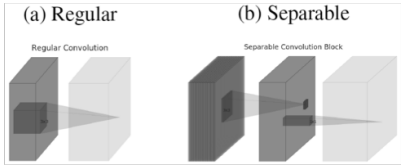
(a) NasNet[23] (b) MobileNet[27]



(c) ShuffleNet [20] (d) Mobilenet V2

MobileNet v2

- Linear bottlenecks
  - Hidden layers carry "manifold of interest"
  - ReLU requires more dimension than manifold dimension
  - Linear layer can carry similar amount of information with fewer feature maps



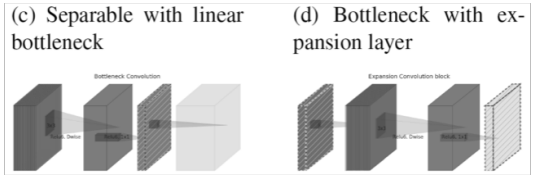


Figure 2: Evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities. The last (lightly colored) layer indicates the beginning of the next block. Note: 2d and 2c are equivalent blocks when stacked. Best viewed in color.

MobileNet v2

- Inverted residual block

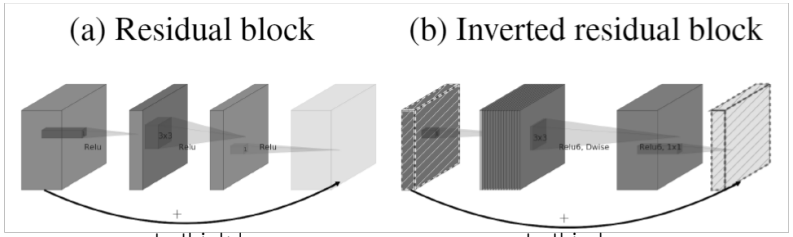


Figure 3: The difference between residual block [8, 30] and inverted residual. Diagonally hatched layers do not use non-linearities. We use thickness of each block to indicate its relative number of channels. Note how classical residuals connects the layers with high number of channels, whereas the inverted residuals connect the bottlenecks. Best viewed in color.

MobileNet v2

- Performance of MobileNet.v2

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Table 4: Performance on ImageNet, comparison for different networks. As is common practice for ops, we count the total number of Multiply-Adds. In the last column we report running time in milliseconds (ms) for a single large core of the Google Pixel 1 phone (using TF-Lite). We do not report ShuffleNet numbers as efficient group convolutions and shuffling are not yet supported.

	Params	MAdds
SSD[34]	14.8M	1.25B
SSDLite	**2.1M**	**0.35B**

Table 5: Comparison of the size and the computational cost between SSD and SSDLite configured with MobileNetV2 and making predictions for 80 classes.

Agenda

- Network Architecture Search
- Building Lightweight Models
- Non-local Neural Nets

Non-Local Nets

- Wang, et al., “Non-local Neural Networks,” Apr. 2018.
  - Learns relation among distant feature elements

Non-Local Nets

- Wang, et al., “Non-local Neural Networks,” Apr. 2018.

$$\mathbf{y}_i = \frac{1}{\mathcal{C}(\mathbf{x})} \sum_{\forall j} f(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_j)$$

$$f(\mathbf{x}_i, \mathbf{x}_j) = e^{\theta(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}$$