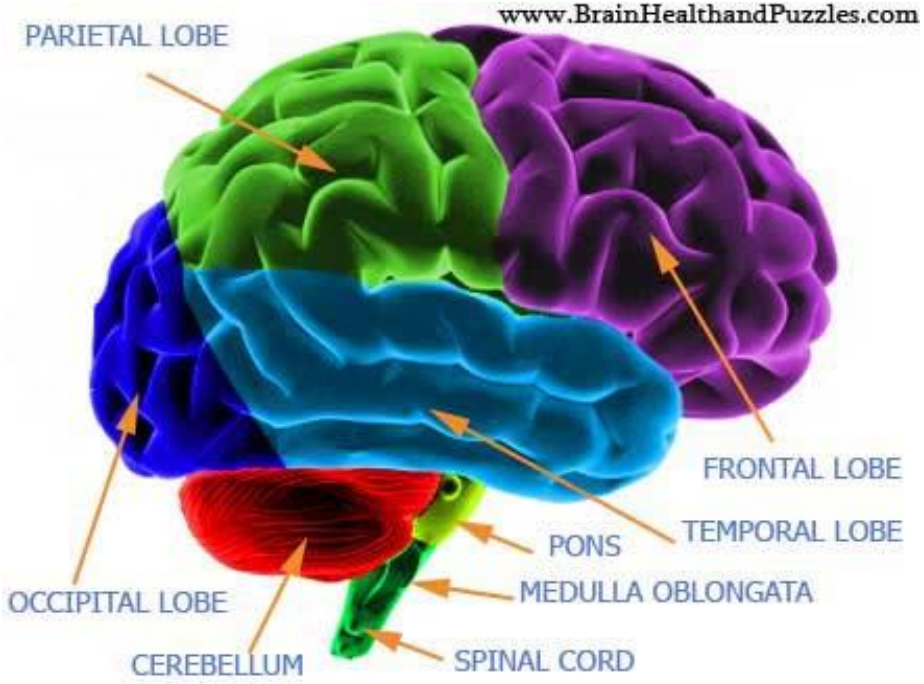
Lecture 4: Neural Networks

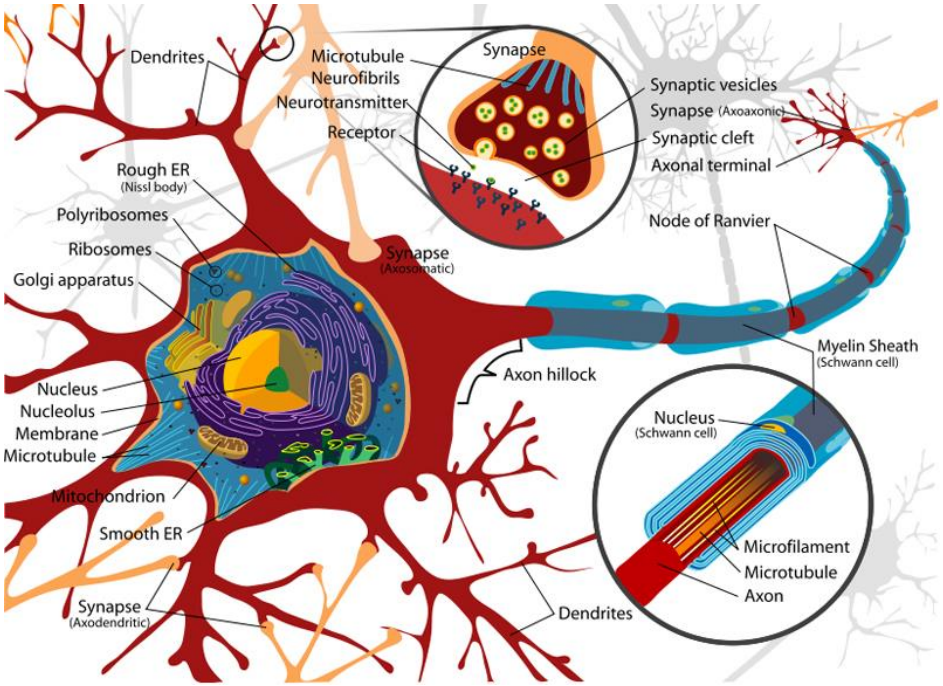
Pilsung Kang
School of Industrial Management Engineering
Korea University

AGENDA

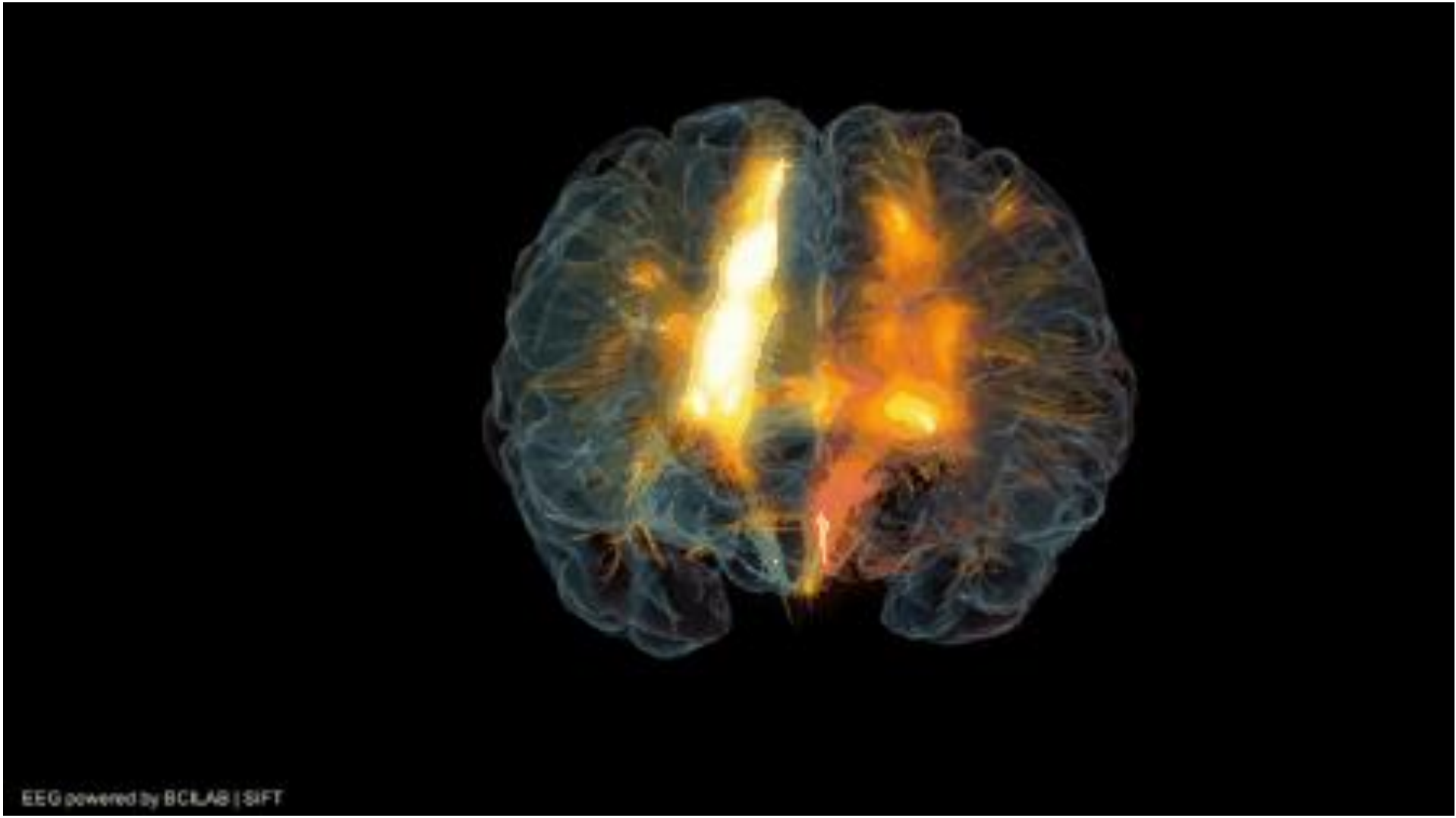
Brain Structure

- How our brain works…
  - ✓ Neurons transmit and analyze communication within the brain and other parts of the nervous system
  - ✓ A message within the brain is converted to electronic signs





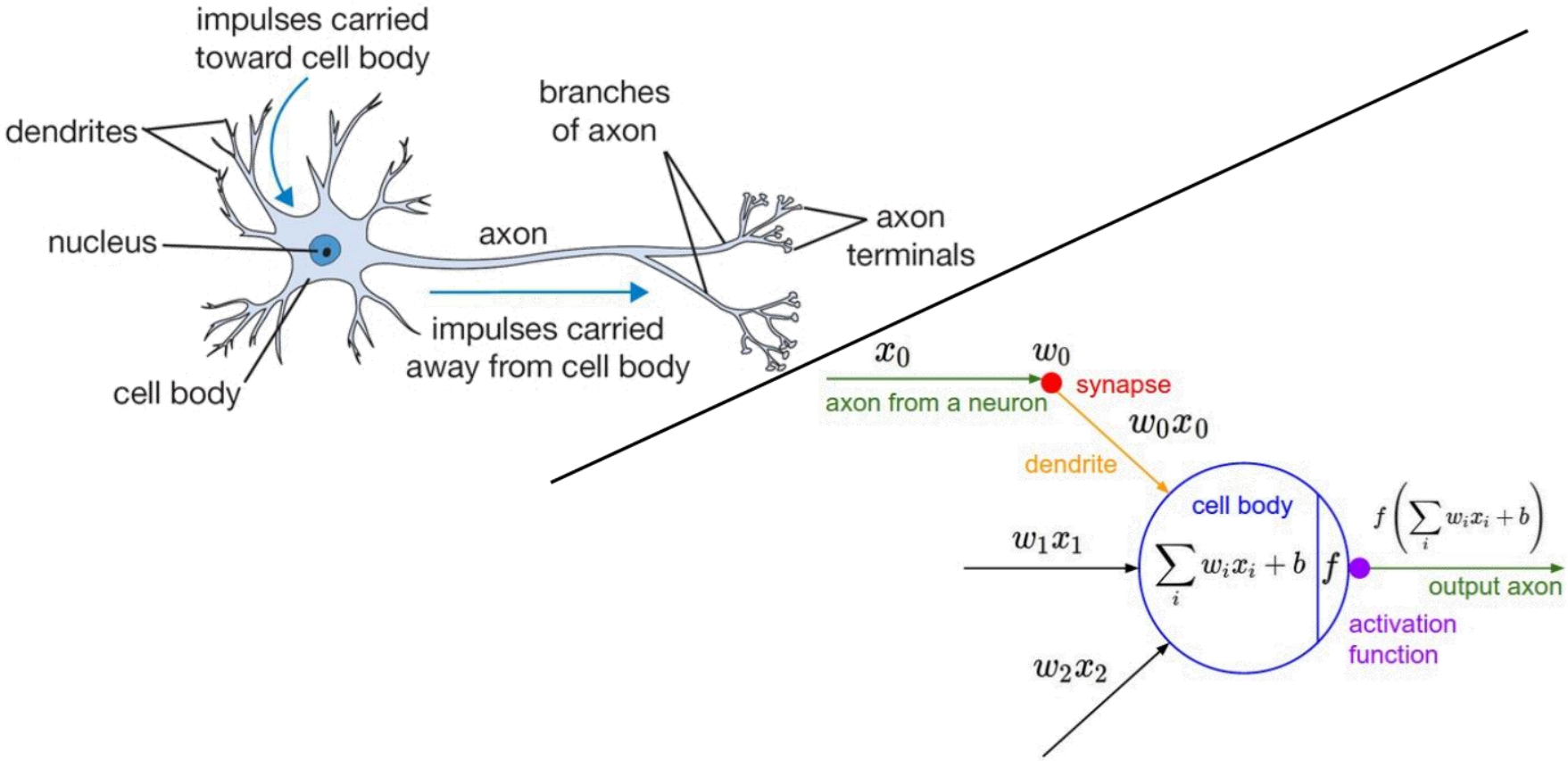
Neuron Firing Off in Real-Time



http://www.dailymail.co.uk/sciencetech/article-2581184/The-dynamic-mind-Stunning-3D-glass-brain-shows-neurons-firing-real-time.html

Perceptron

- Imitate a single neuron

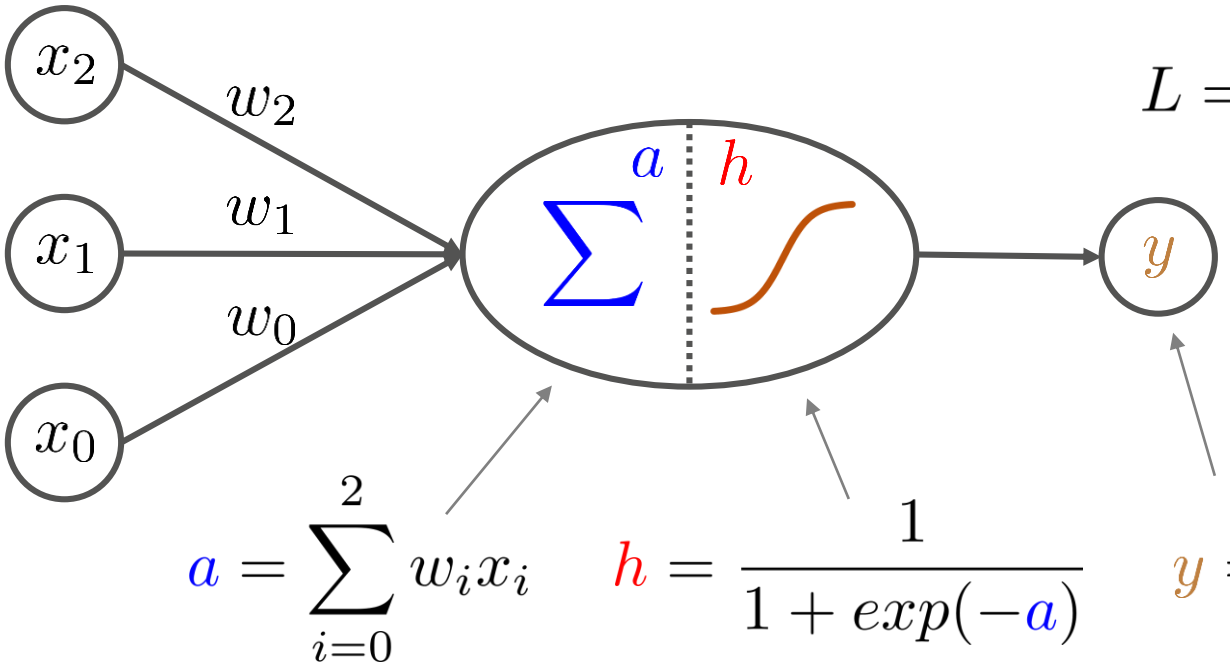


Perceptron

- Perceptron
  - ✓ An organism with only 1 neuron

Define the loss function as the squared difference between the desired value (t) and the predicted value (y)

$$L = \frac{1}{2}(t - y)^2$$



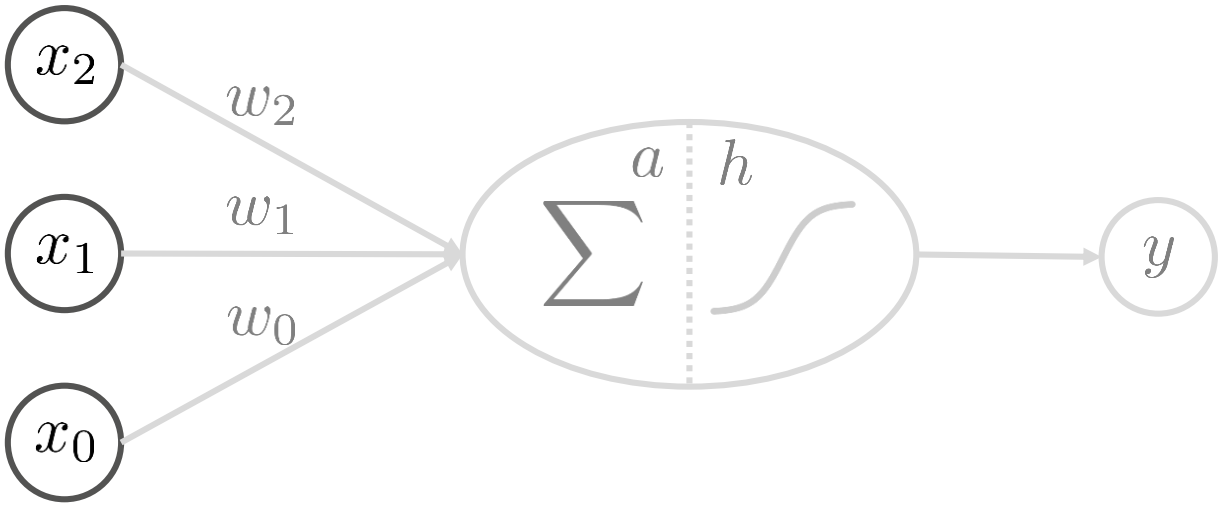
$$a = \sum_{i=0}^{2} w_i x_i \qquad h = \frac{1}{1 + exp(-a)} \qquad y = h$$

Take information from different sources

Decide the amount of information to be delivered (activation)

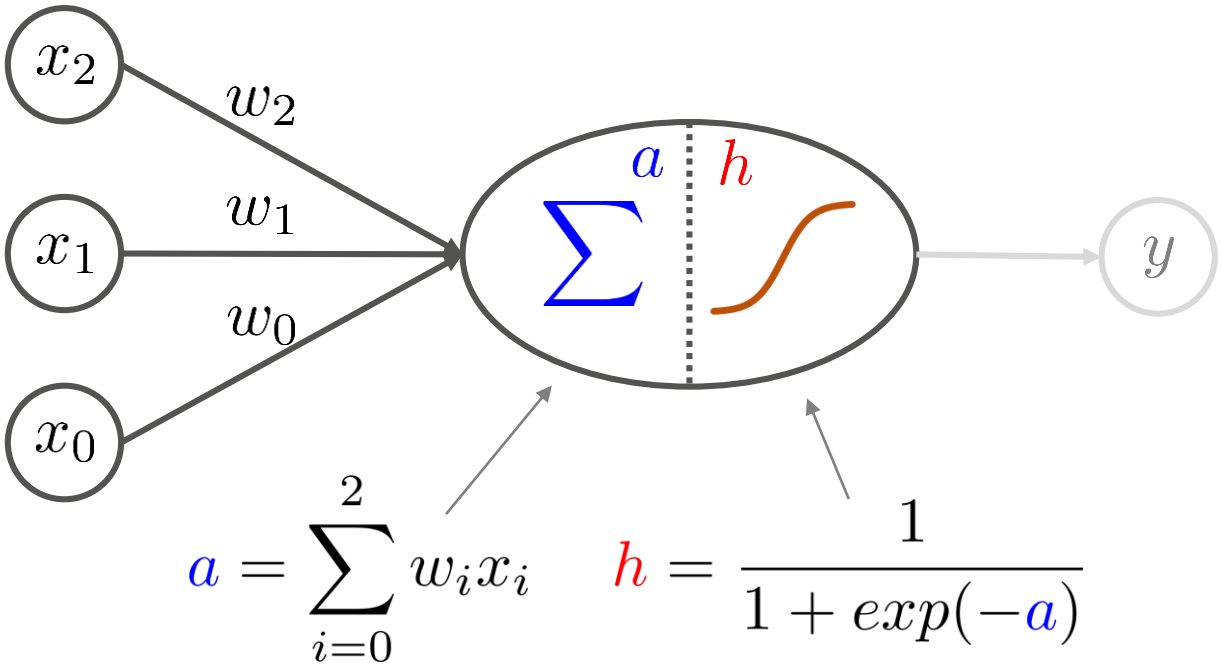
Perceptron

- Input node
  - ✓ Input (predictor, explanatory) variables



Perceptron

- Hidden node
  - ✓ Take the weighted sum of input values and perform a non-linear activation



Perceptron

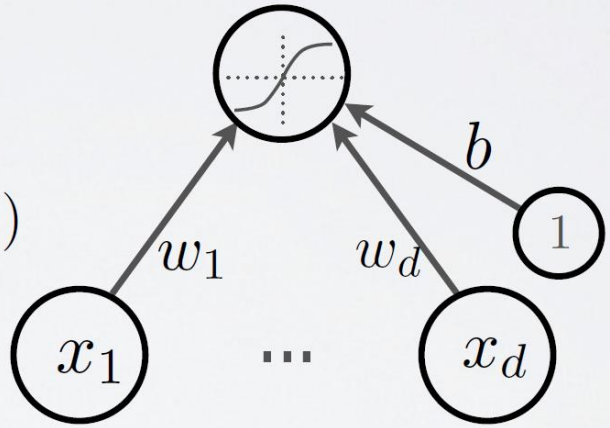
- Role of activation
  - ✓ Determine how much information from the previous layer is forward to the next layer

- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (output) activation

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$



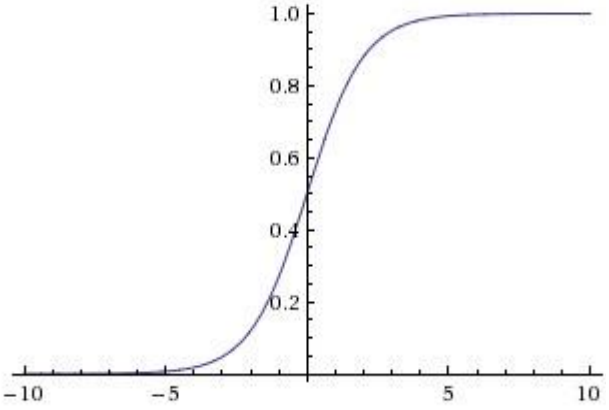
- $\mathbf{w}$ are the connection weights
- b is the neuron bias
- $g(\cdot)$ is called the activation function

http://info.usherbrooke.ca/hlarochelle/cours/ift725_A2013/contenu.html

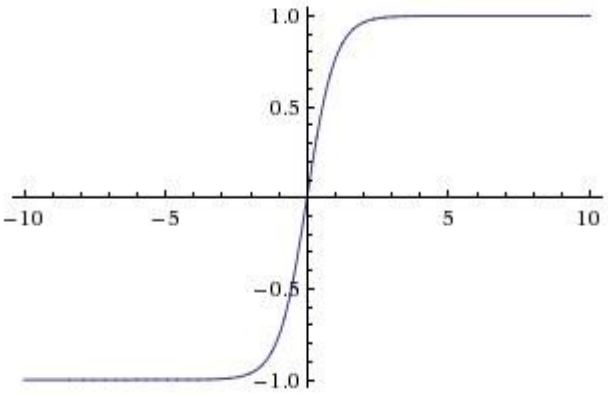
Perceptron

- Representative activation functions
  - ✓ Sigmoid: the most commonly used activation, [0, 1] range, learning speed is relatively slow
  - ✓ Tanh: Similar to sigmoid but [-1, 1] range, learning speed is relatively fast
  - ✓ ReLU (Rectified linear unit): very fast learning speed, easy to compute (without exponential function)

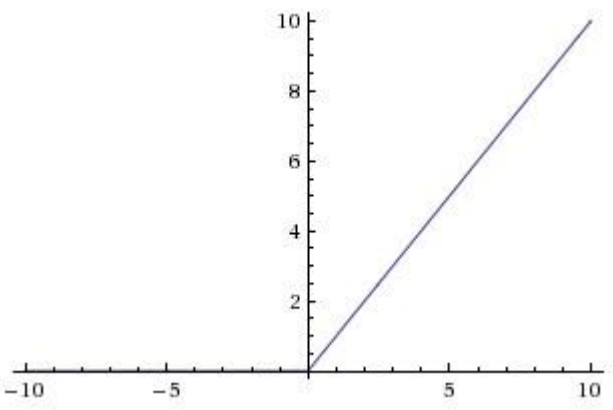
[Sigmoid]



[Tanh]



[ReLU]



$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

$$g(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1}$$

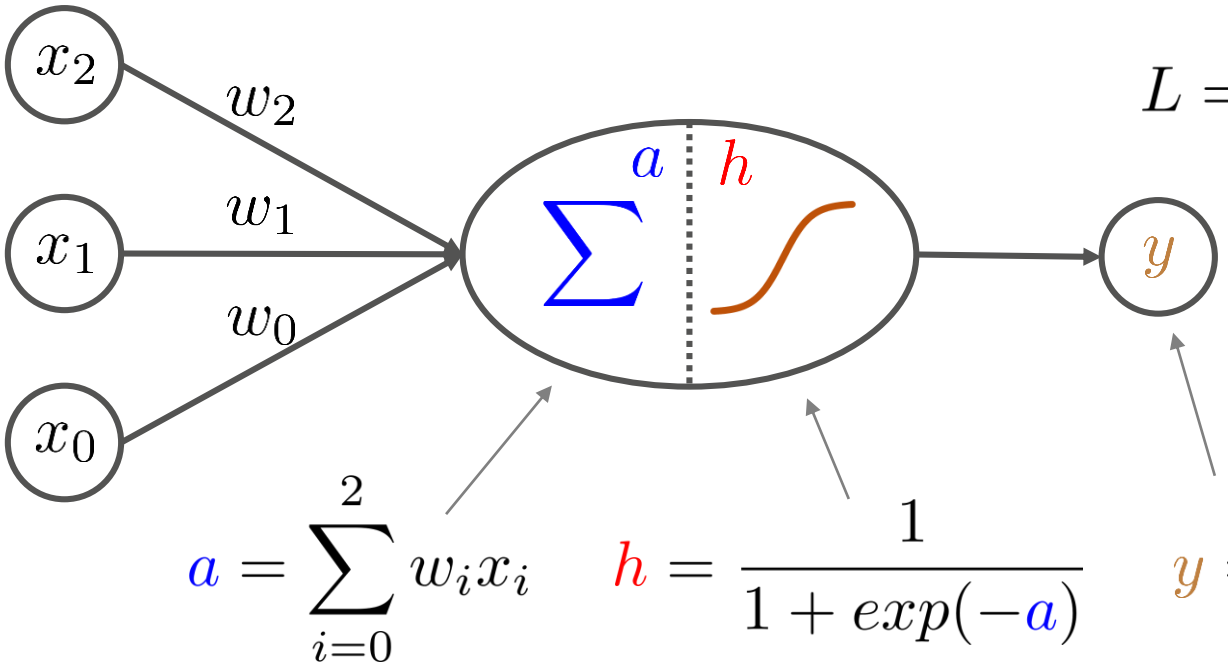
$$g(a) = \text{reclin}(a) = \max(0, a)$$

Perceptron

- Output node
  - ✓ Take the value from the hidden node (Perceptron has only one hidden node)
  - ✓ It takes a weighted sum of hidden nodes in a multi-layer perceptron

원하는 값(t)과 예측값(y)의 차이를 손실함수로 정의

$$L = \frac{1}{2}(t - y)^2$$



$$a = \sum_{i=0}^{2} w_i x_i$$

여러 변수들의 정보를 나름대로 취합해서

$$h = \frac{1}{1 + exp(-a)}$$

얼마만큼 다음 단계로 전달할지 결정한다 (활성화)

$$y = h$$

Perceptron

- Purpose of perceptron
  - ✓ Find the weight w that can best match the input (x) and the target (t)
- How do we know that the relationship is accurately found?
  - ✓ Use a loss function (how the output y is close to the target t)
    - Regression: squared loss is commonly used

$$L = \frac{1}{2}(t - y)^2$$

    - Classification: cross-entropy is usually used (only binary classification is possible with perceptron)

$$L = -\sum_i t_i \log p_i$$

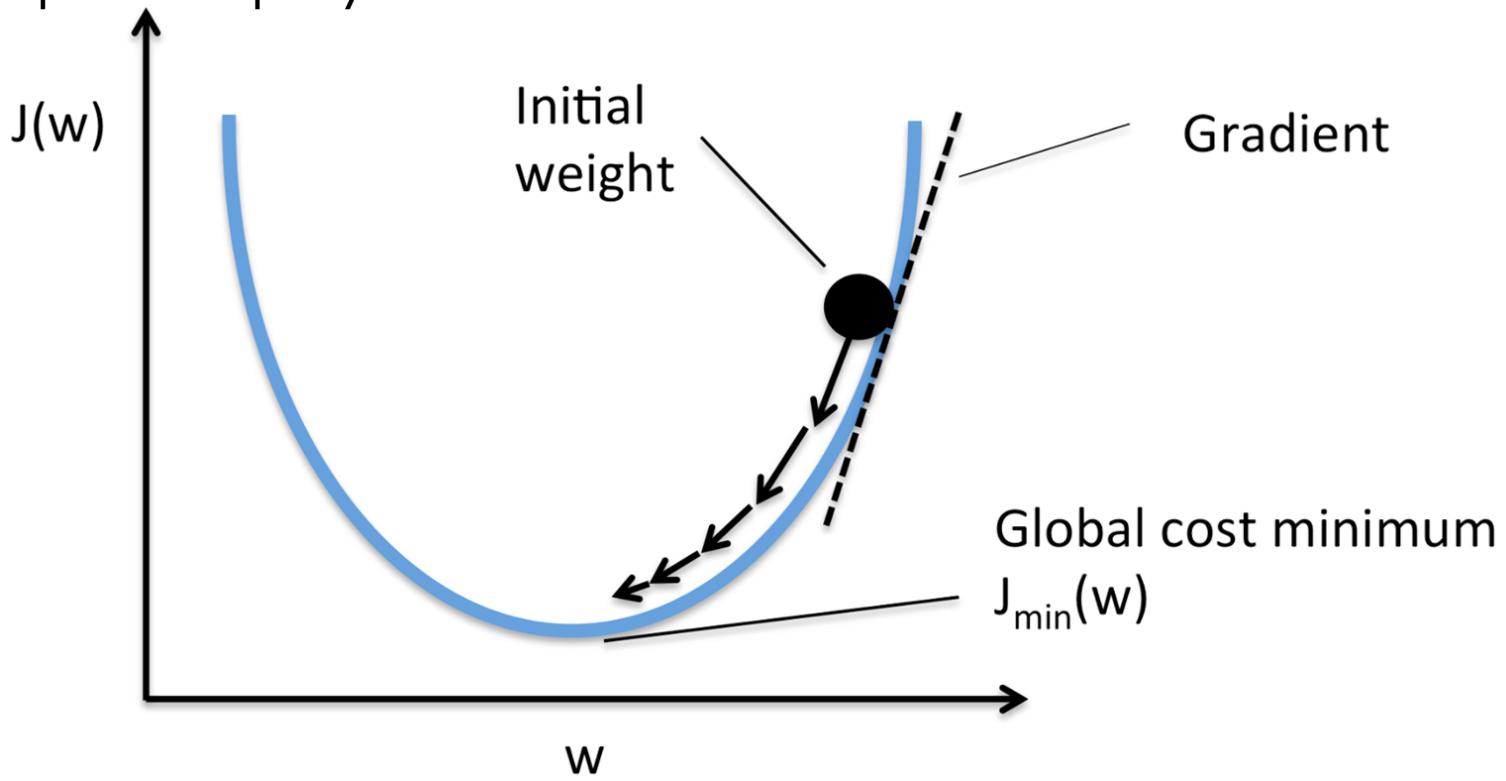
  - ✓ Cost function
    - Computes how inaccurate the current model is (the average of loss function values is commonly used)

Learning: Gradient Descent

- Gradient Descent

Learning: Gradient Descent

- Gradient Descent Algorithm
  - ✓ Blue line: the objective function to be minimized
  - ✓ Black circle: the current solution
  - ✓ Direction of the arrows: the direction that the current solution should move to improve the quality of solution



Learning: Gradient Descent

- Take the first derivative of the cost function w.r.t the current weight w
  - ✓ Is the gradient 0?
    - Yes: Current weights are the optimum! → end of learning
    - No: Current weights can be improved → learn more
  - ✓ How can we improve the current weights if the gradient is not 0?
    - Move the current weight toward to the opposite direction of the gradient
  - ✓ How much should the weights be moved?
    - Not sure
    - Move them a little and compute the gradient again
    - It will converge

Learning: Gradient Descent

- Theoretical Background
  - ✓ Taylor expansion

$$f(w+\Delta w) = f(w) + \frac{f'(w)}{1!}\Delta w + \frac{f''(w)}{2!}(\Delta w)^2 + \cdots$$

  - ✓ If the first derivative is not zero, we can decrease the function value by moving x toward the opposite direction of its first derivative

어느 방향으로 갈 것인가?

$$w_{new} = w_{old} - \alpha f'(w), \quad \text{where } 0 < \alpha < 1.$$

얼마만큼 갈 것인가?

  - ✓ Then the function value of the new x is always smaller than that of the old x

$$f(w_{new}) = f(w_{old} - \alpha f'(w_{old})) \cong f(w_{old}) - \alpha \left| f'(w) \right|^2 < f(w_{old})$$

https://www.youtube.com/watch?v=3d6DsjIBzJ4&t=322s

Learning: Gradient Descent

- Use chain rule

$$\frac{\partial L}{\partial y} = y - t \qquad \frac{\partial y}{\partial h} = 1$$

$$\frac{\partial h}{\partial a} = \frac{exp(-a)}{(1 + exp(-a))^2} = \frac{1}{1 + exp(-a)} \cdot \frac{exp(-a)}{1 + exp(-a)} = h(1 - h)$$

$$\frac{\partial a}{\partial w_i} = x_i$$

- Gradients for w and x

$$\frac{L}{\partial w_i} = \frac{L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial a} \cdot \frac{\partial a}{\partial w_i} = (y - t) \cdot 1 \cdot h(1 - h) \cdot x_i$$

$$w_i^{new} = w_i^{old} - \alpha \times \frac{L}{\partial w_i} = w_i^{old} - \alpha \times (y - t) \cdot 1 \cdot h(1 - h) \cdot x_i$$

Learning: Gradient Descent

- Weight update by Gradient Descent

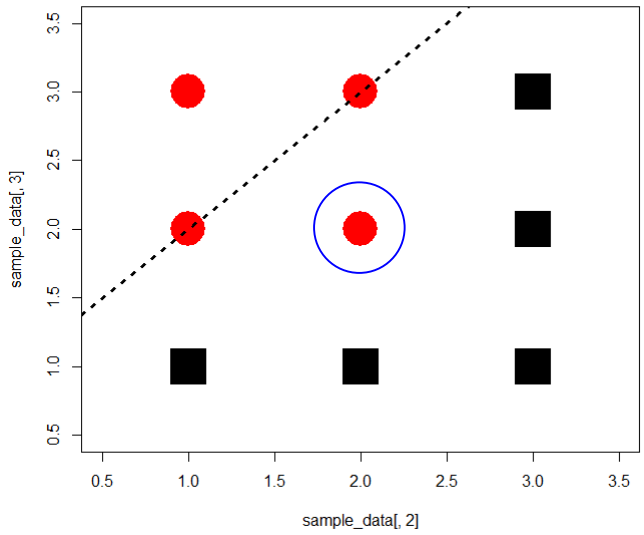
$$w_i^{new} = w_i^{old} - \alpha \times (y - t) \cdot 1 \cdot h(1 - h) \cdot x_i$$

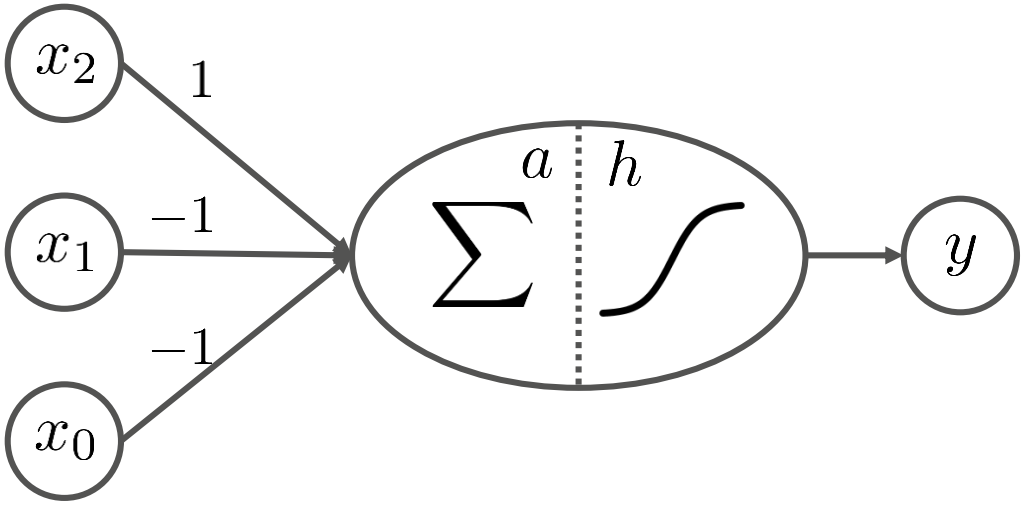
현재의 출력값(y)과 정답(t)이
차이가 많이 날 수록
가중치를 많이 업데이트 하라

대상 가중치와 연결된 입력
변수의 값이 클 수록
가중치를 많이 업데이트 하라

Training Perceptron: Example 1 (alpha = 1)

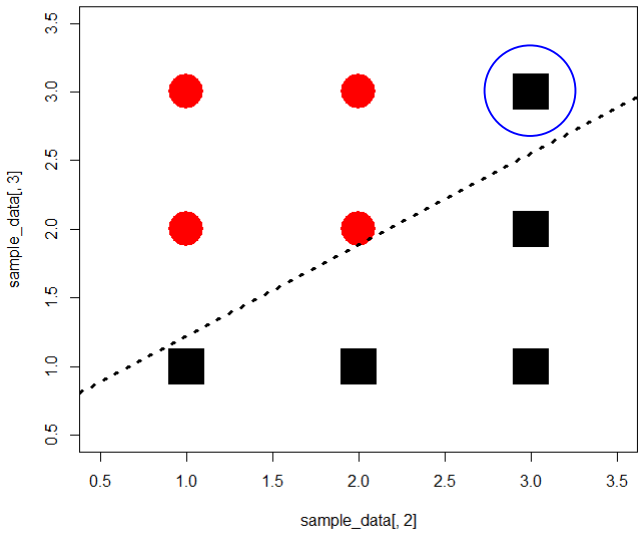
- Initialize and select the first training example (x_1=2, x_2=2, t=1)

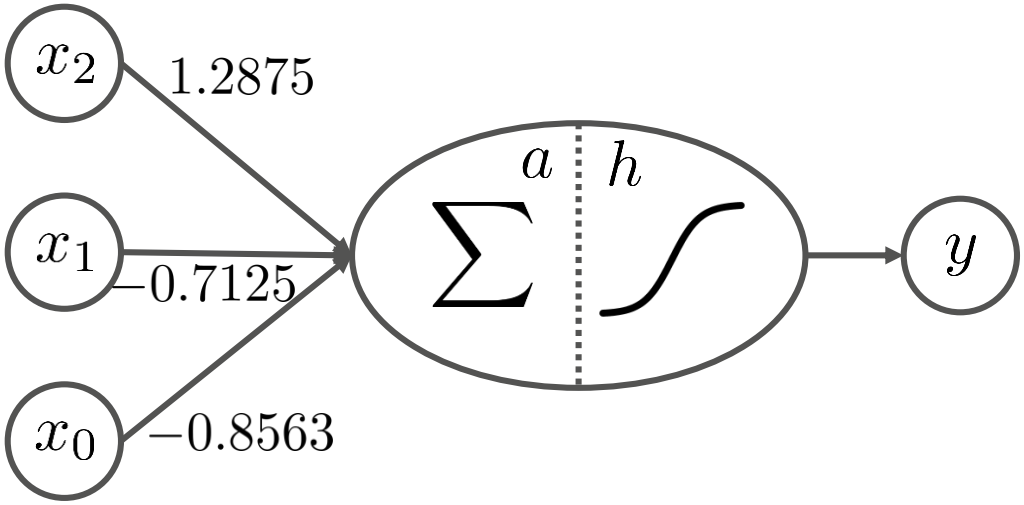




Training Perceptron: Example 1 (alpha = 1)

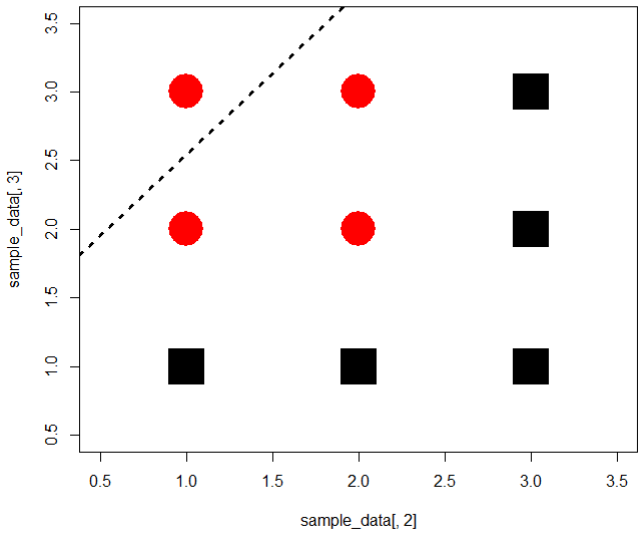
- Training result and selection of the second training example (x_1=3, x_2=3, t=0)

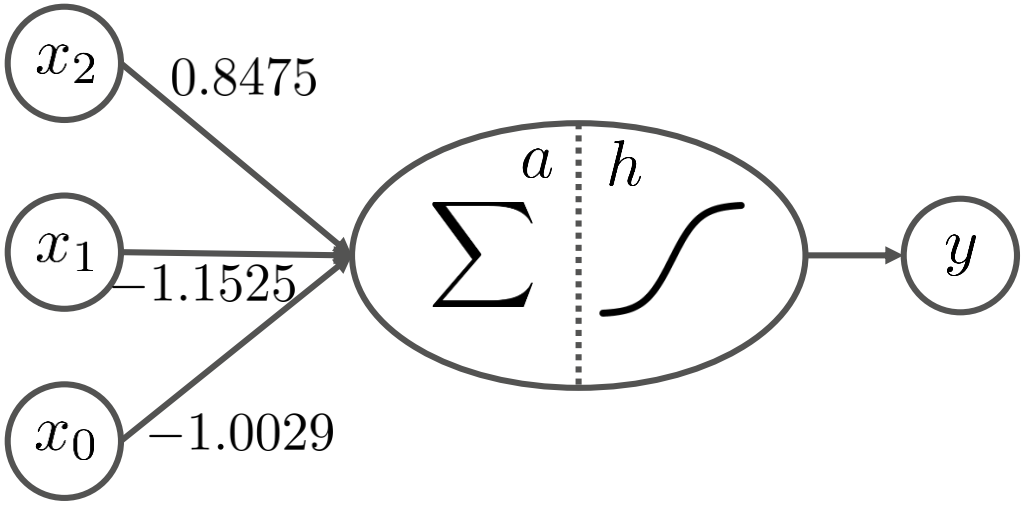




Training Perceptron: Example 1 (alpha = 1)

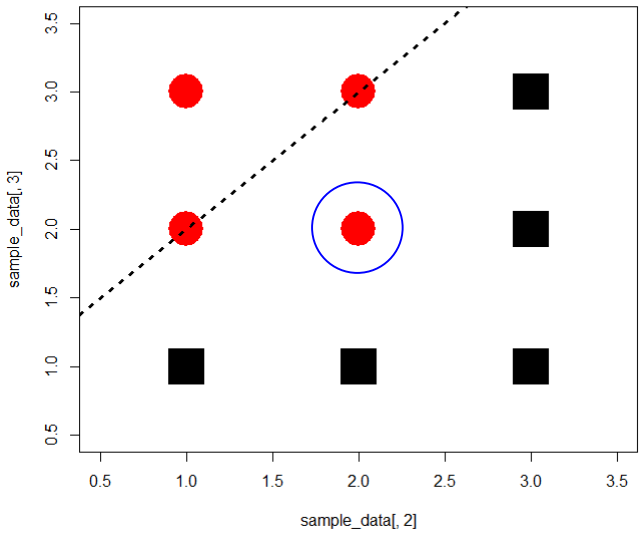
- Training result

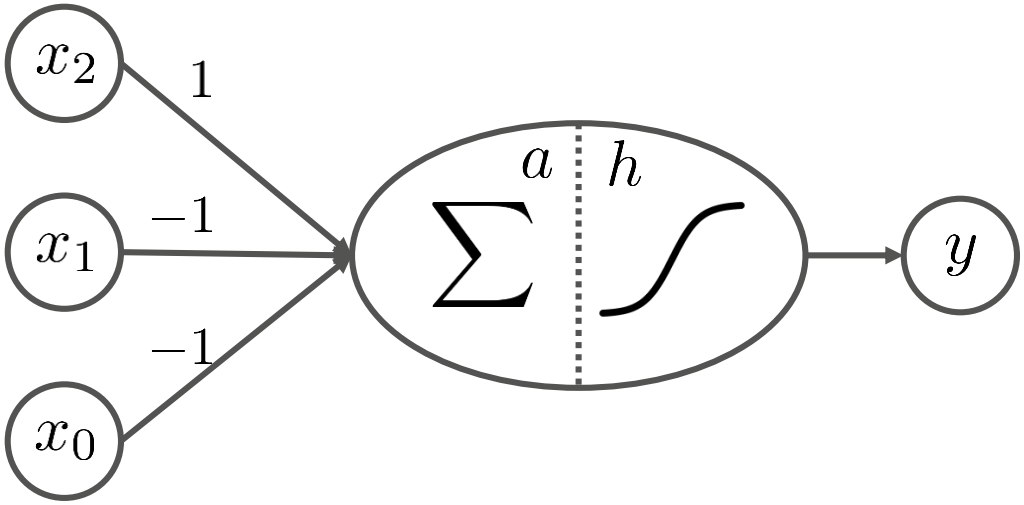




Training Perceptron: Example 1 (alpha = 0.5)

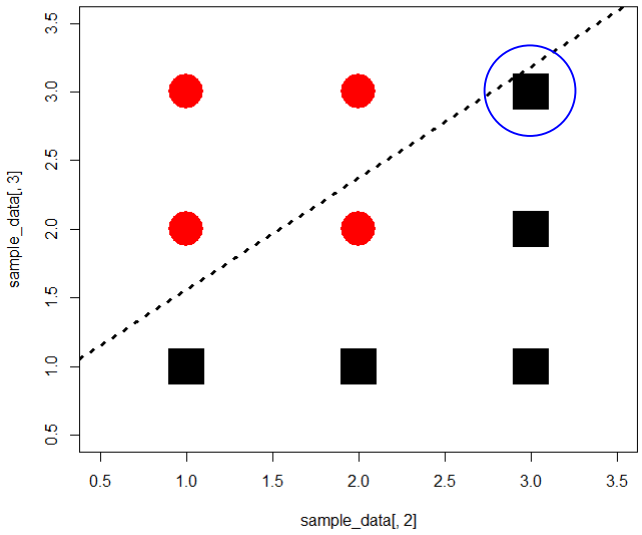
- Initialize and select the first training example (x_1=2, x_2=2, t=1)

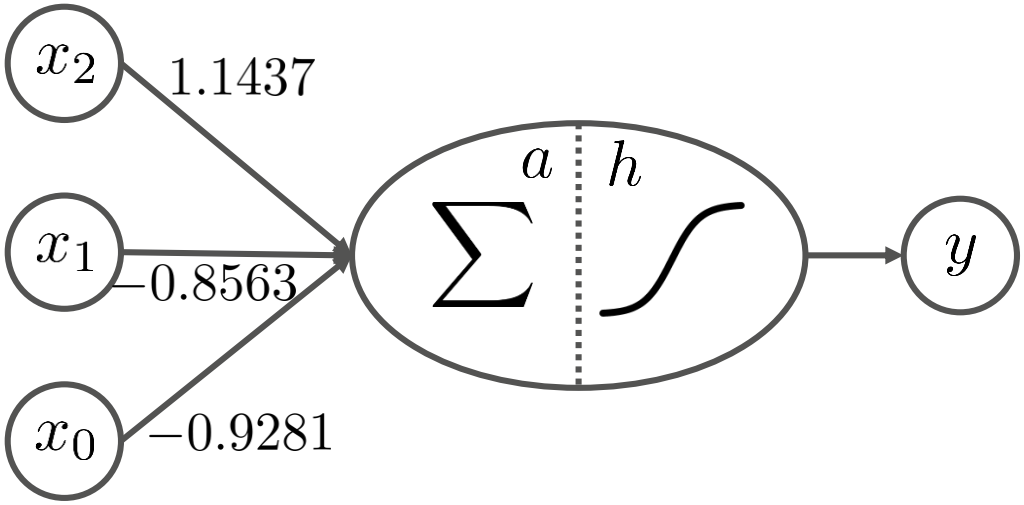




Training Perceptron: Example 1 (alpha = 0.5)

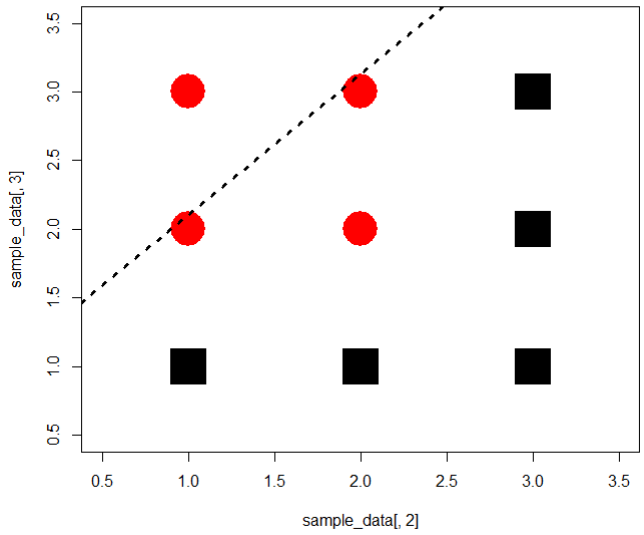
- Training result and selection of the second training example (x_1=3, x_2=3, t=0)

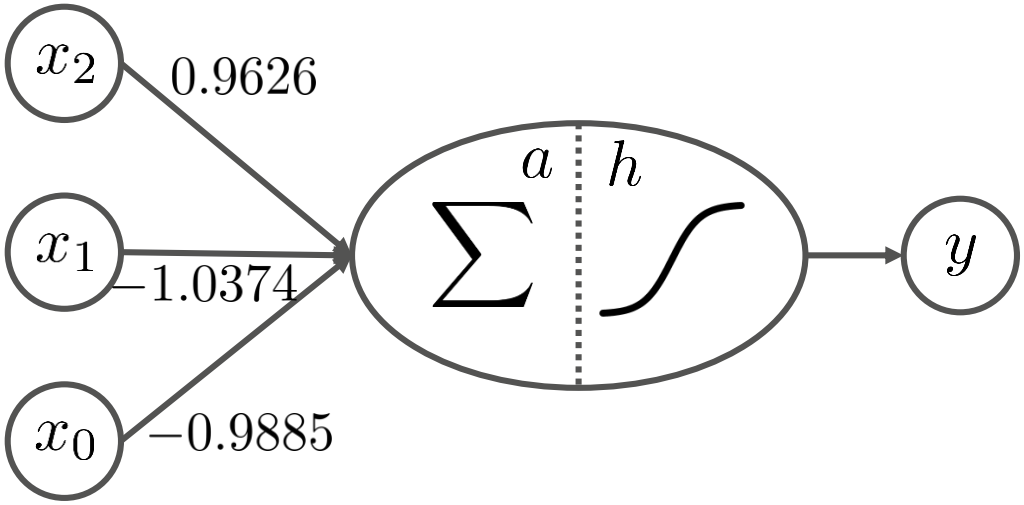




Training Perceptron: Example 1 (alpha = 0.5)

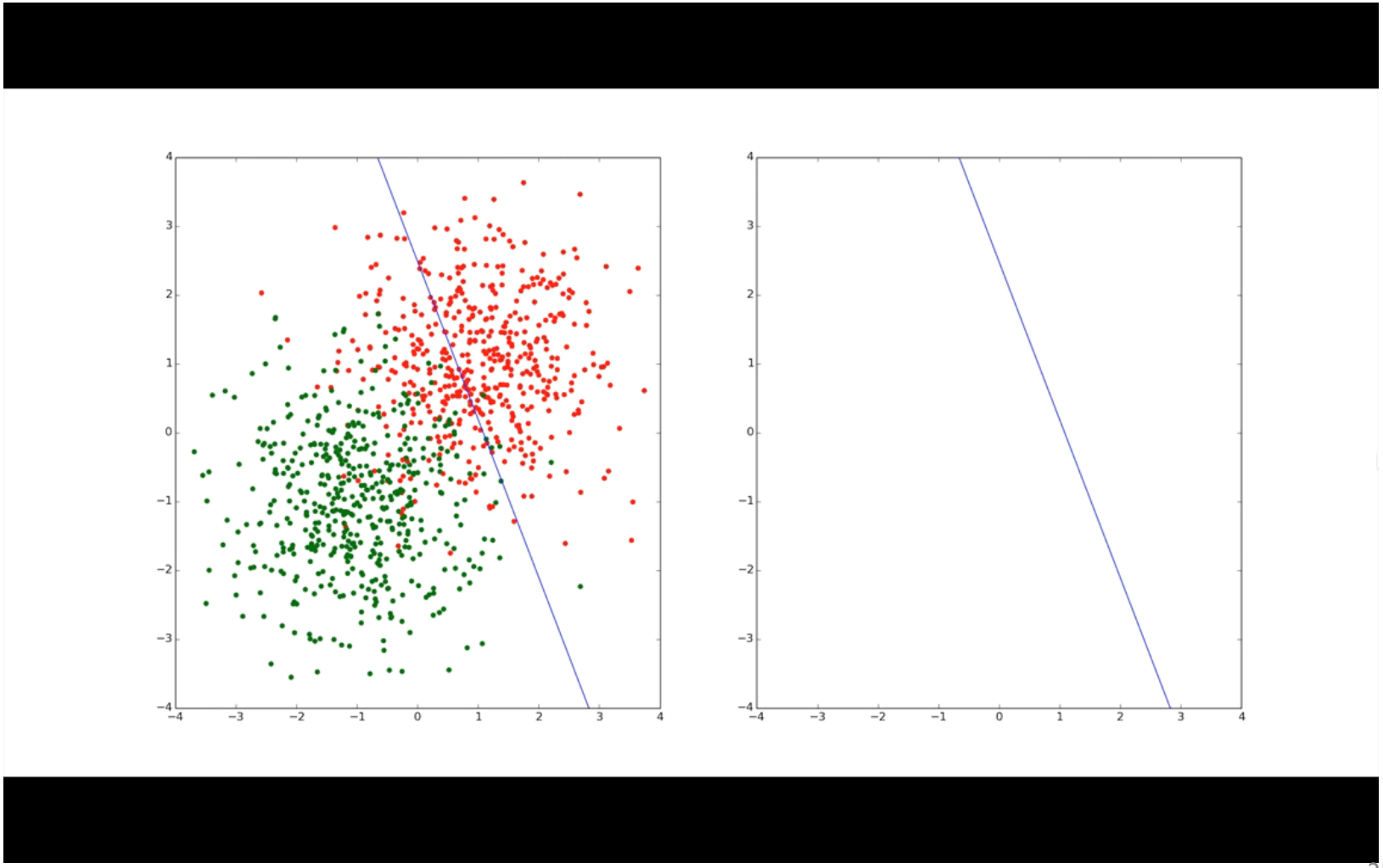
- Training result





Perceptron

- Training Perceptron

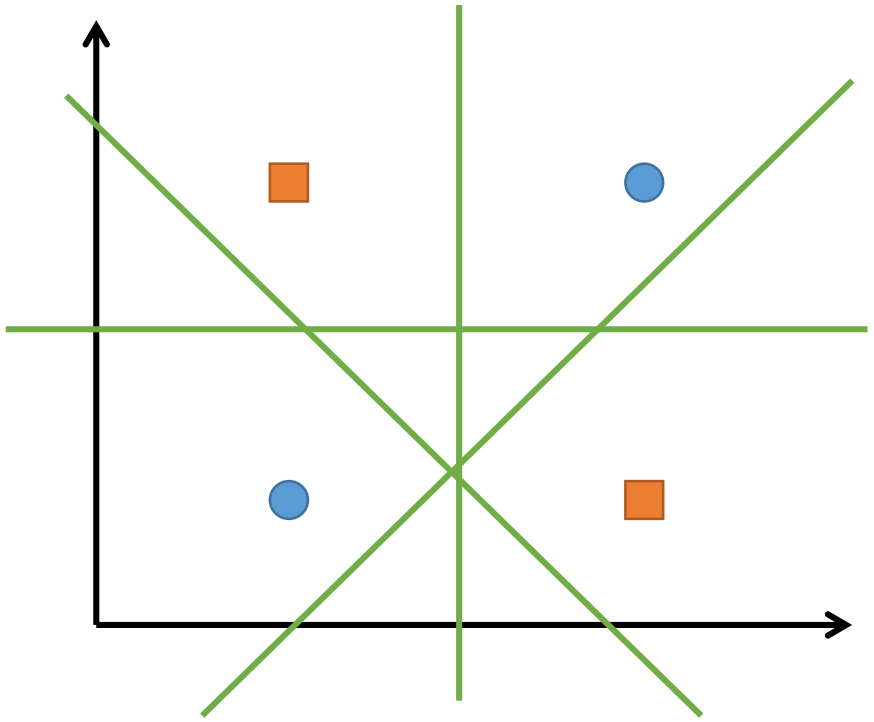


Perceptron: Limitation

- The Limitation of Linear Models
  - ✓ **Classification:**
    - Linear (Fisher) discriminant analysis, logistic regression, etc.
    - Can only produce a linear class boundary
  - ✓ **Regression:**
    - Multiple linear regression
    - Can only capture the linear relationship between the predictors and the outcome
  - ✓ Cannot results in good prediction performance ***when the classification boundary or the predictor/outcome relationship is not linear***

Perceptron: Limitation

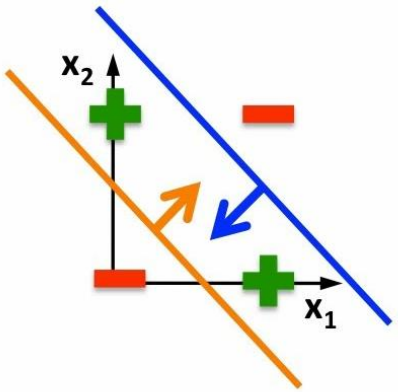
- The Limitation of Linear Models
  - ✓ Draw a straight line that perfectly separates the circles and crosses (XOR)

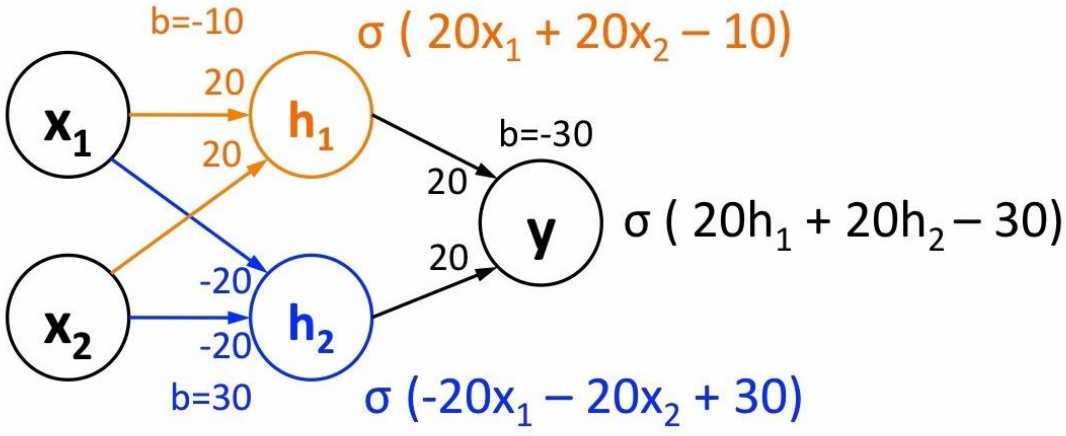


Multi-Layer Perceptron (MLP)

- Combine multiple perceptrons!
  - ✓ If we cannot solve a complex problem directly, then it is better to decompose it into some small and simple problems that can be solved!

Linear classifiers cannot solve this





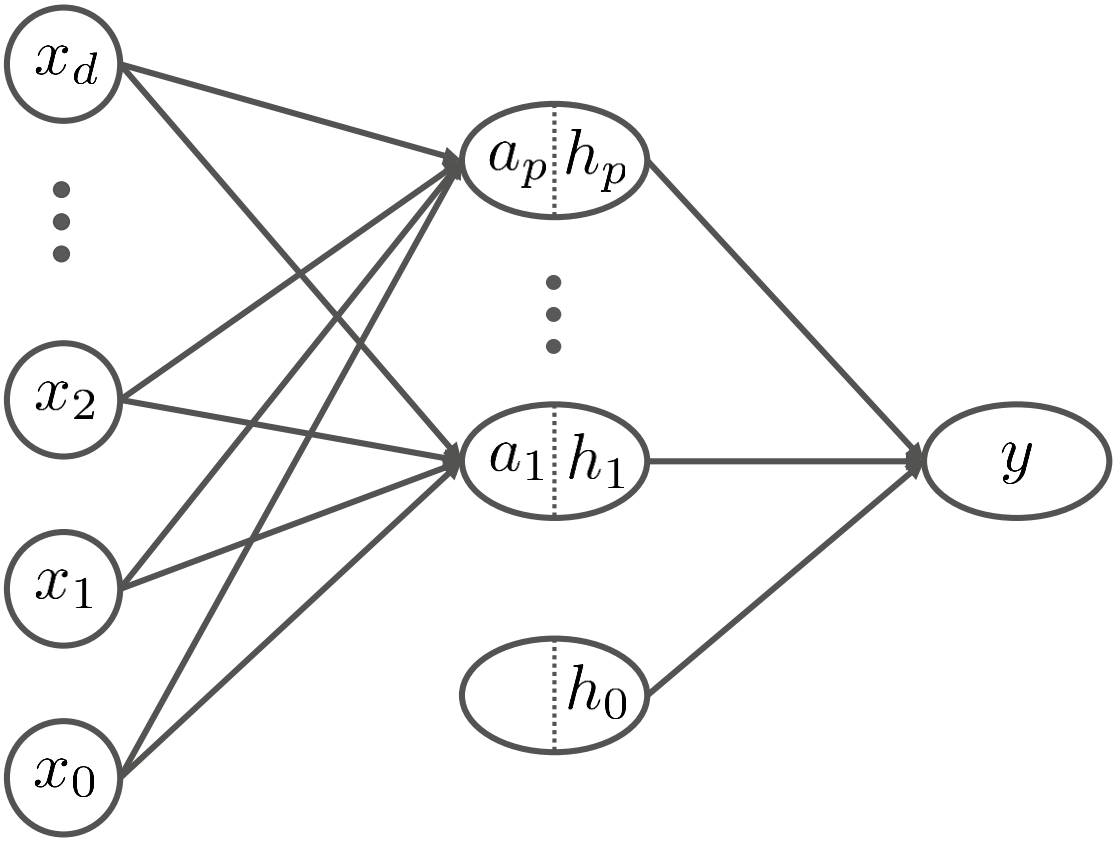
σ(20*0 + 20*0 – 10) ≈ 0
σ(20*1 + 20*1 – 10) ≈ 1
σ(20*0 + 20*1 – 10) ≈ 1
σ(20*1 + 20*0 – 10) ≈ 1

σ (-20*0 – 20*0 + 30) ≈ 1
σ (-20*1 – 20*1 + 30) ≈ 0
σ (-20*0 – 20*1 + 30) ≈ 1
σ (-20*1 – 20*0 + 30) ≈ 1

σ (20*0 + 20*1 – 30) ≈ 0
σ (20*1 + 20*0 – 30) ≈ 0
σ (20*1 + 20*1 – 30) ≈ 1
σ (20*1 + 20*1 – 30) ≈ 1

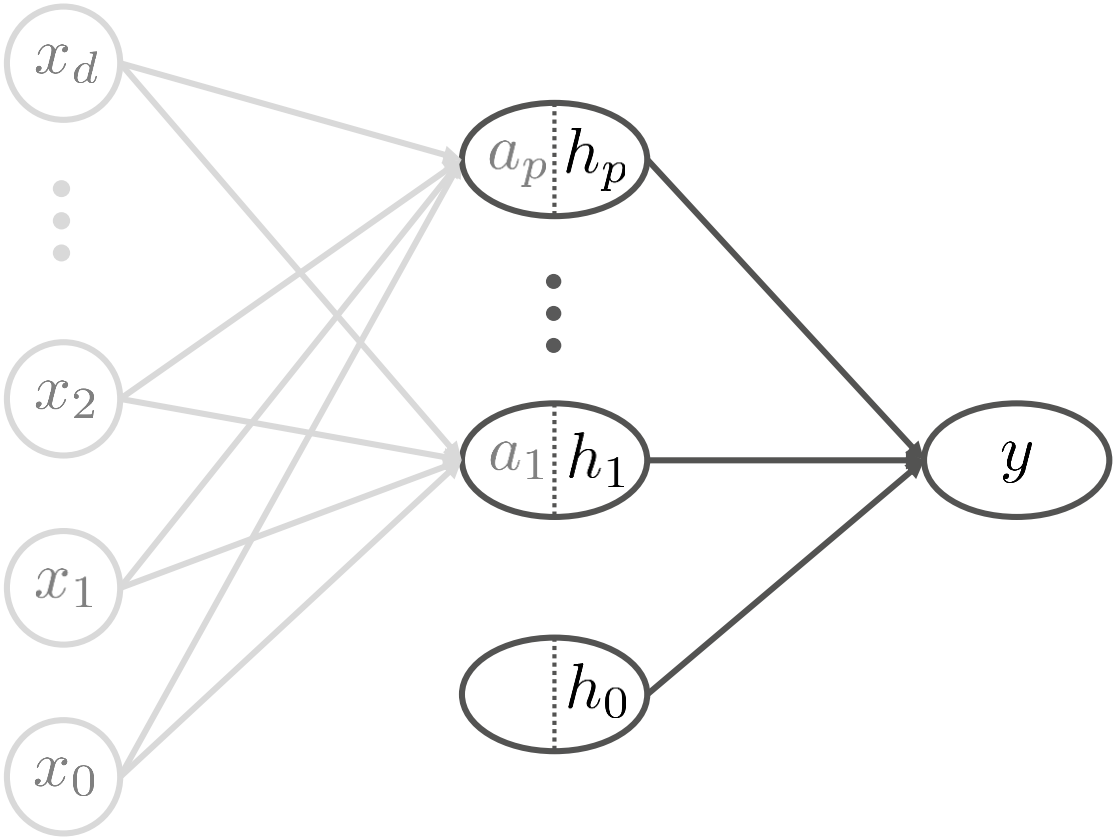
Multi-Layer Perceptron (MLP)

- Basic Structure: Feed-forward Neural Network with One Hidden Layer
  - ✓ Each hidden node can be considered as an independent perceptron
  - ✓ The output node is a combination of all perceptrons



Multi-Layer Perceptron (MLP)

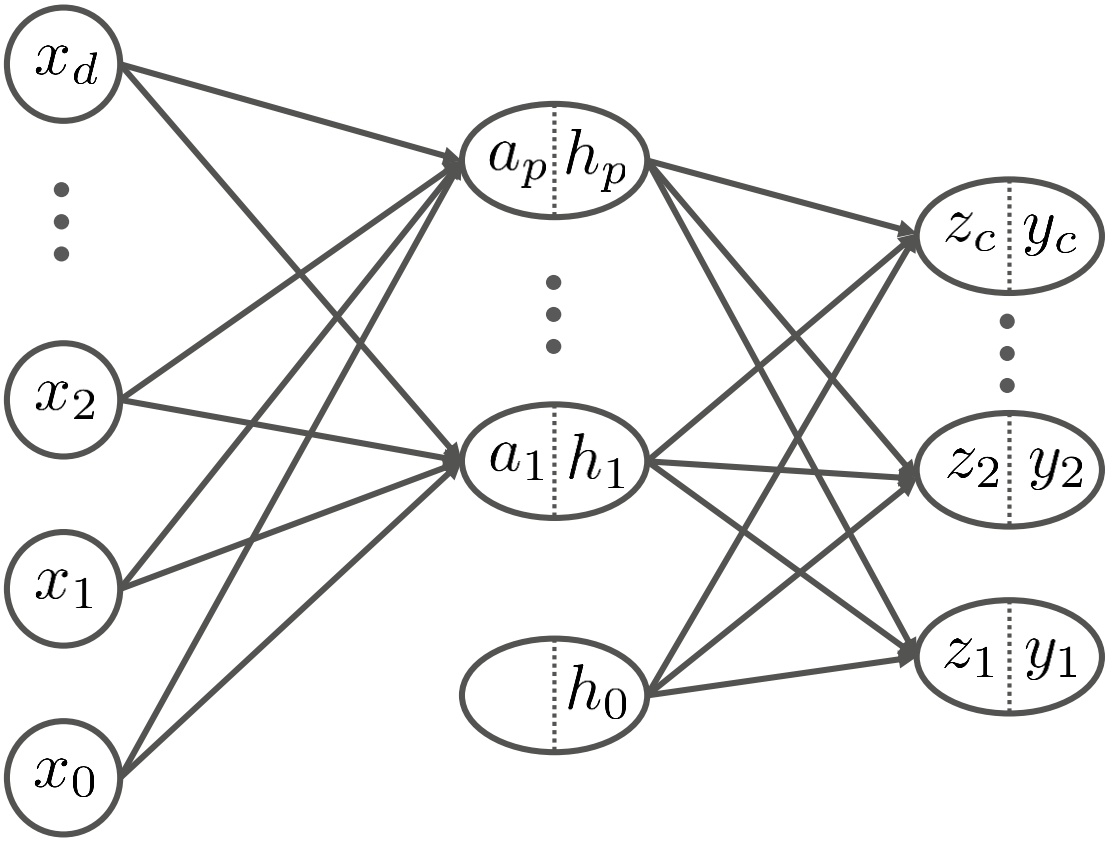
- Basic Structure: Feed-forward Neural Network with One Hidden Layer
  - ✓ Each hidden node can be considered as an independent perceptron
  - ✓ The output node is a linear combination of all perceptrons (regression)



$$y = \sum_{i=0}^{p} w_p^{(2)} h_p$$

Multi-Layer Perceptron (MLP)

- Basic Structure: Feed-forward Neural Network with One Hidden Layer
  - ✓ Each hidden node can be considered as an independent perceptron
  - ✓ The output node is a linear combination of all perceptrons (regression)



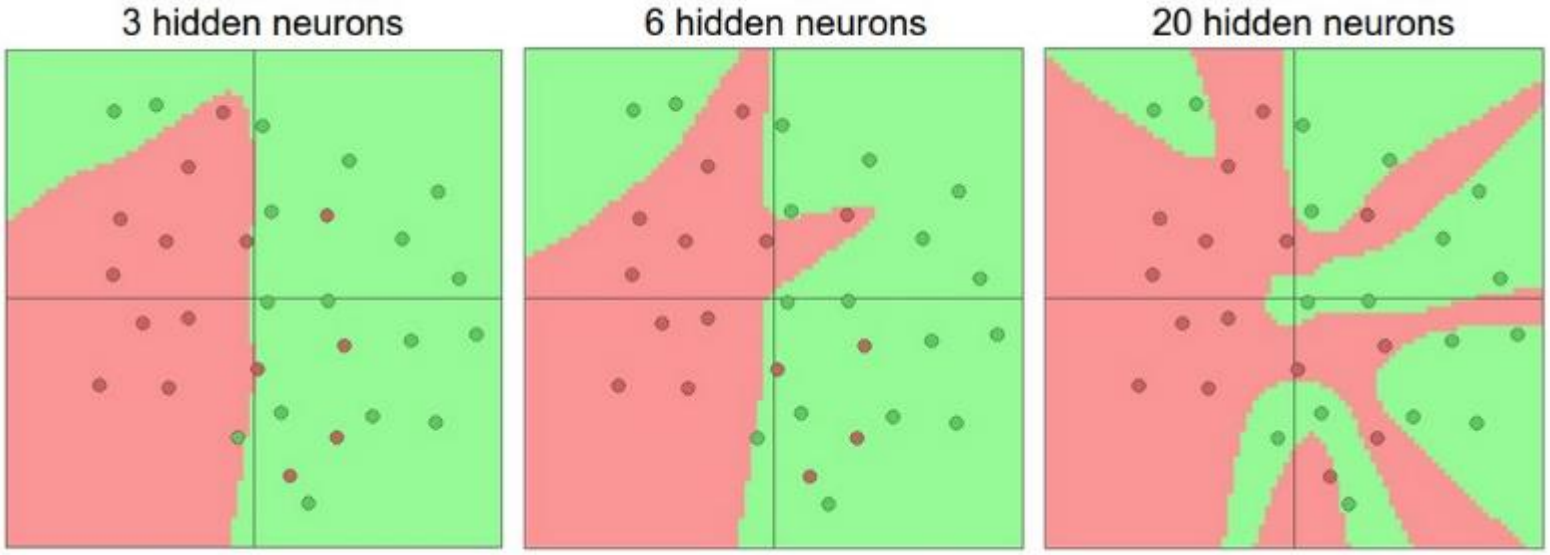
$$z_j = \sum_{i=0}^{p} w_{jp}^{(2)} h_p$$

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{c} e^{z_k}}$$

각 출력 노드의 출력값을 해당 범주에 속하는 확률로 해석할 수 있음

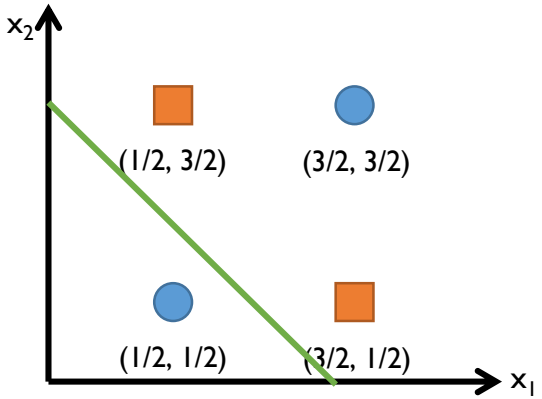
Multi-Layer Perceptron (MLP)

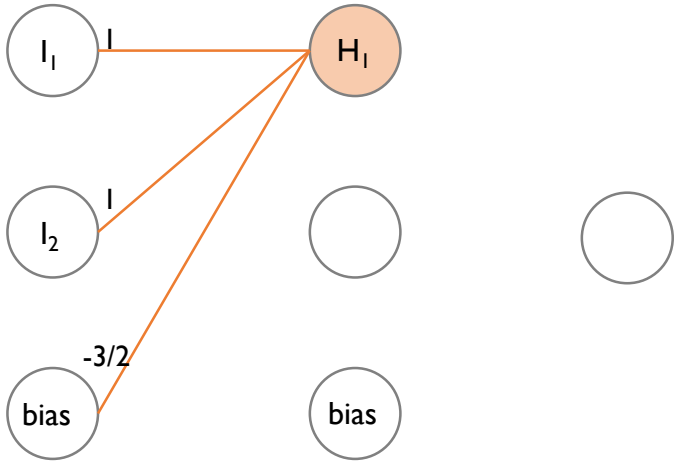
- The role of hidden nodes
  - ✓ Determines the complexity of ANN
  - ✓ If we use more number of hidden nodes, we can find a more sophisticated decision boundary (classification) or an arbitrary shape of function (regression)



Multi-Layer Perceptron (MLP)

- XOR problem revisited





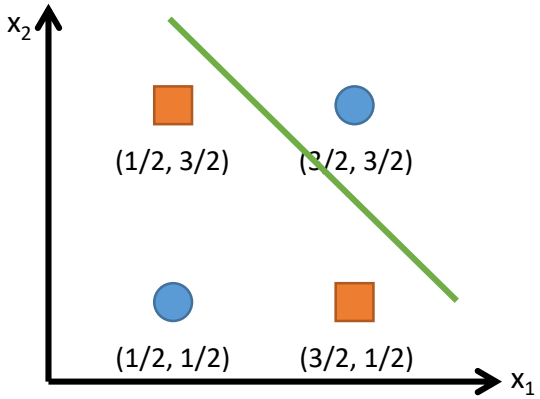
$$a_1 = x_1 + x_2 - \frac{3}{2}$$

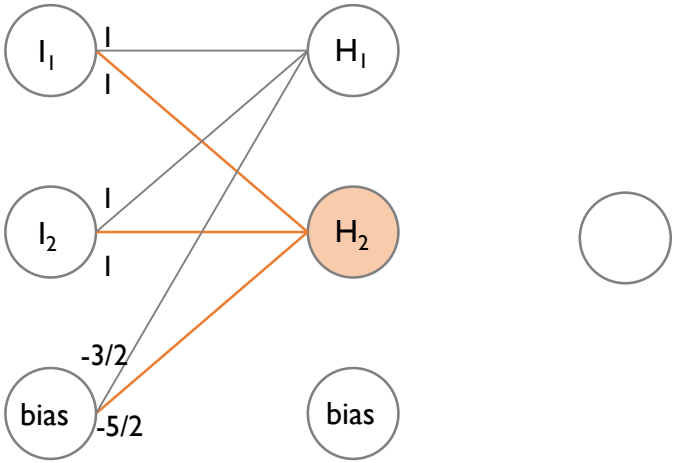
$$h_1 = g(a_1) = \begin{cases} 1 & \text{if } a_1 \geq 0 \\ -1 & \text{if } a_1 < 0 \end{cases}$$

x_1	x_2	a_1	h_1
1/2	1/2	-1/2	-1
3/2	1/2	1/2	1
1/2	3/2	1/2	1
3/2	3/2	3/2	1

Multi-Layer Perceptron (MLP)

- XOR problem revisited (cont')





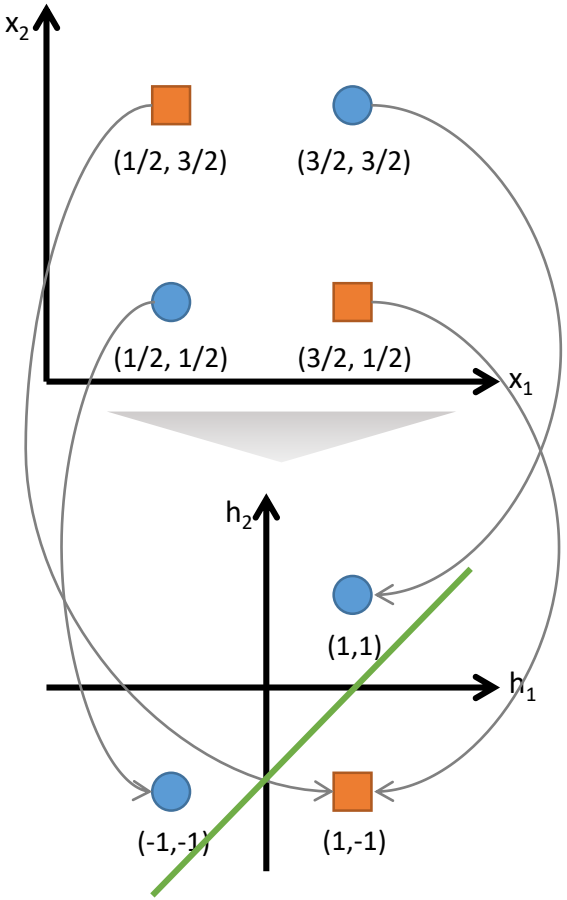
$$a_2 = x_1 + x_2 - \frac{5}{2}$$

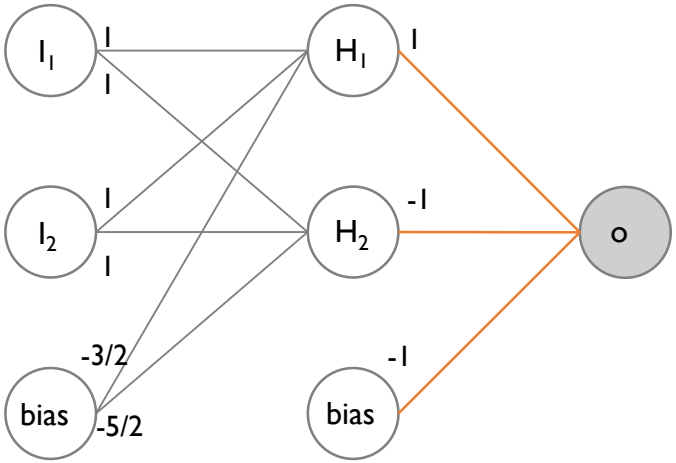
$$h_2 = g(a_2) = \begin{cases} 1 & \text{if } a_2 \geq 0 \\ -1 & \text{if } a_2 < 0 \end{cases}$$

x_1	x_2	a_2	h_2
1/2	1/2	-3/2	-1
3/2	1/2	-1/2	-1
1/2	3/2	-1/2	-1
3/2	3/2	1/2	1

Multi-Layer Perceptron (MLP)

- XOR problem revisited (cont')





	x_1	x_2	a_1	h_1	a_2	h_2	o	y
●	1/2	1/2	-1/2	-1	-3/2	-1	-1	-1
■	3/2	1/2	1/2	1	-1/2	-1	1	1
■	1/2	3/2	1/2	1	-1/2	-1	1	1
●	3/2	3/2	3/2	1	1/2	1	-1	-1

$$o = h_1 + h_2 - 1 \qquad y = g(o) = \begin{cases} 1 & \text{if } o \geq 0 \\ -1 & \text{if } o < 0 \end{cases}$$

Multi-Layer Perceptron (MLP)

- General formulation
  - ✓ The output of the hidden node j (when the activation function is sigmoid):

$$a_j = \sum_{i=0}^{d} w_{ji}^{(1)} x_i, \quad h_j = g(a_j) = \frac{1}{1 + \exp(-a_j)}$$

  - ✓ The output of the output node (when the activation function is linear):

$$y = \sum_{j=0}^{p} w_j^{(2)} h_j$$

  - ✓ The final outcome of the neural network:

$$y = \sum_{j=0}^{p} w_j^{(2)} \cdot g(\sum_{i=0}^{d} w_{ji}^{(1)} x_i)$$

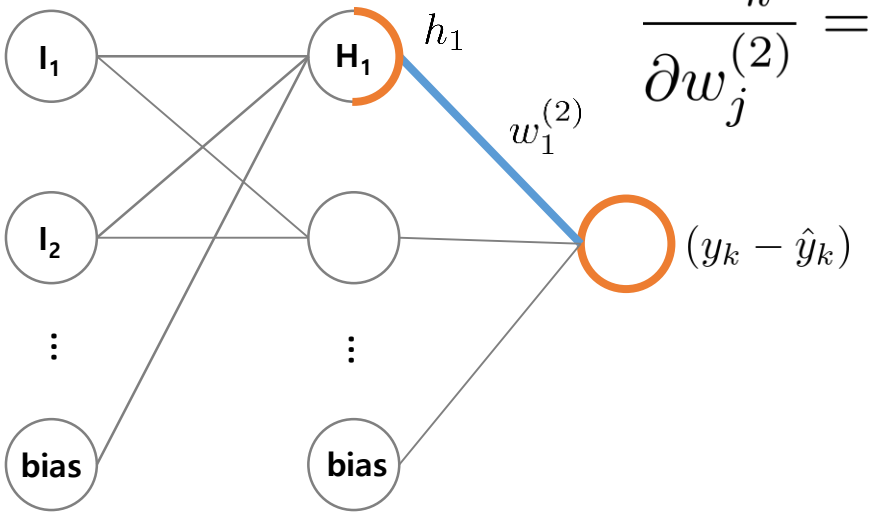
Multi-Layer Perceptron (MLP)

- Error Back-Propagation
  - ✓ The loss of k[th] observation

$$L_k = \frac{1}{2}(y_k - \hat{y}_k)^2 \quad , \quad y_k = \sum_{j=0}^{p} w_j^{(2)} \cdot g(\sum_{i=0}^{d} w_{ji}^{(1)} \mathbf{x}_{ki})$$

  - ✓ The weight $w_j^{(2)}$ which connects the jth hidden node and the output node will be updated by

$$\frac{\partial L_k}{\partial w_j^{(2)}} = \frac{\partial L_k}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_j^{(2)}} = (y_k - \hat{y}_k) \cdot h_j$$

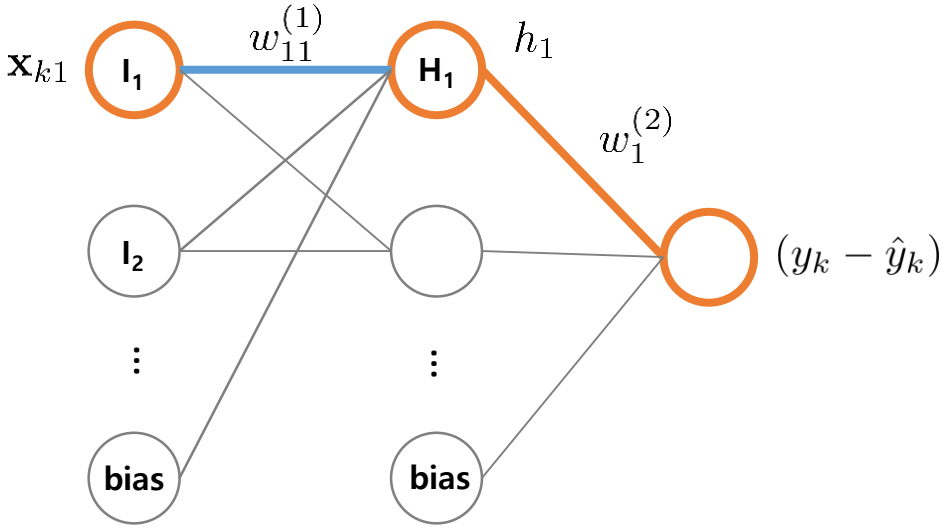


Multi-Layer Perceptron (MLP)

- Error Back-Propagation
  - ✓ The weight $w_{ji}^{(l)}$ which connects the i^{th} input node and j^{th} hidden node

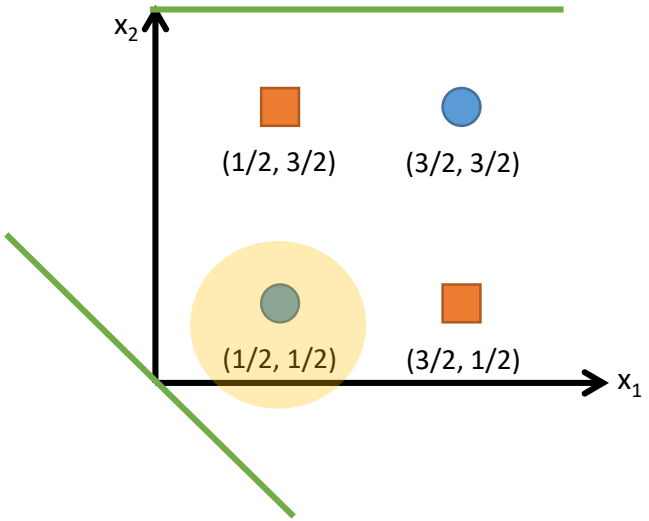
$$\frac{\partial L_k}{\partial w_j i^{(1)}} = \frac{\partial L_k}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_j^{(2)}} \cdot \frac{\partial h_j}{\partial a_j} \cdot \frac{\partial a_j}{\partial w_{ji}^{(1)}}$$

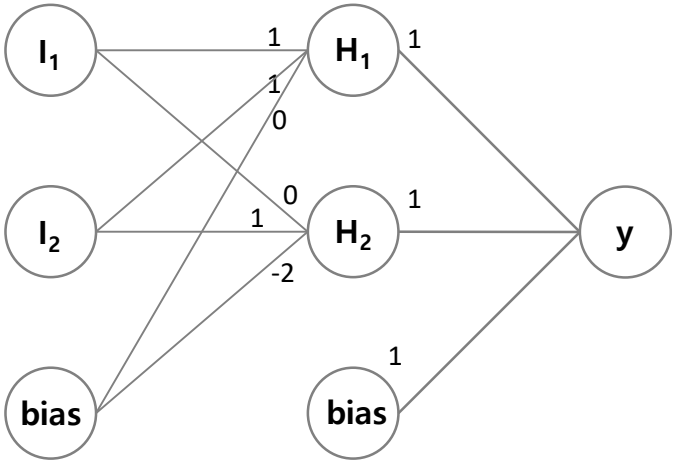
$$= (y_k - \hat{y}_k) \cdot w_j^{(2)} \cdot a_j \cdot (1 - a_j) \cdot \mathbf{x}_{ki}$$



MLP: Training

- Error Back-Propagation: Example
  - ✓ Initial weight: Random generation





$$a_1 = \sum w_{1i}^{(1)} x_i = 1 \times 0.5 + 1 \times 0.5 + 0 \times 1 = 1 \qquad h_1 = \frac{1}{1 + \exp(1)} = 0.269$$

$$a_2 = \sum w_{2i}^{(1)} x_i = 0 \times 0.5 + 1 \times 0.5 + (-2) \times 1 = -1.5 \qquad h_2 = \frac{1}{1 + \exp(-1.5)} = 0.818$$

$$\hat{y} = \sum w_j^{(2)} h_j = 1 \times 0.269 + 1 \times 0.818 + 1 \times 1 = 2.087$$

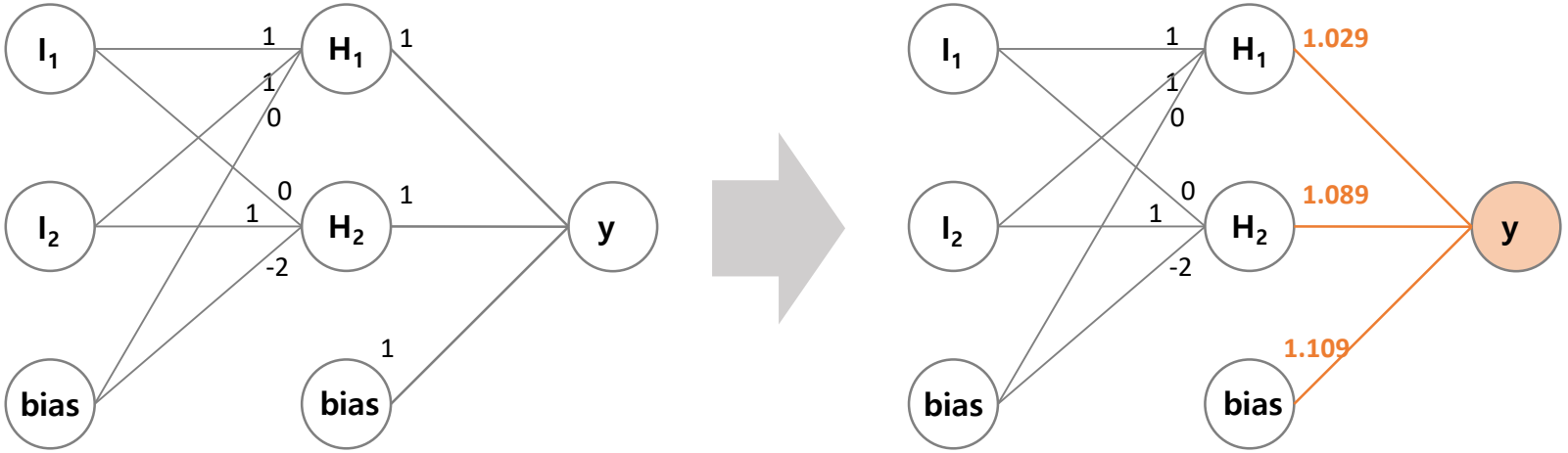
MLP: Training

- Error Back-Propagation: Example
  - ✓ Update the weights between the output and the hidden nodes

$$w_1^{(2)}(new) = w_1^{(2)}(old) - \eta \times (y - \hat{y}) \times h_1 = 1 - 0.1 \times (1 - 2.087) \times 0.269 = 1.029$$

$$w_2^{(2)}(new) = w_2^{(2)}(old) - \eta \times (y - \hat{y}) \times h_2 = 1 - 0.1 \times (1 - 2.087) \times 0.818 = 1.089$$

$$w_0^{(2)}(new) = w_0^{(2)}(old) - \eta \times (y - \hat{y}) \times b^{(2)} = 1 - 0.1 \times (1 - 2.087) \times 1 = 1.109$$



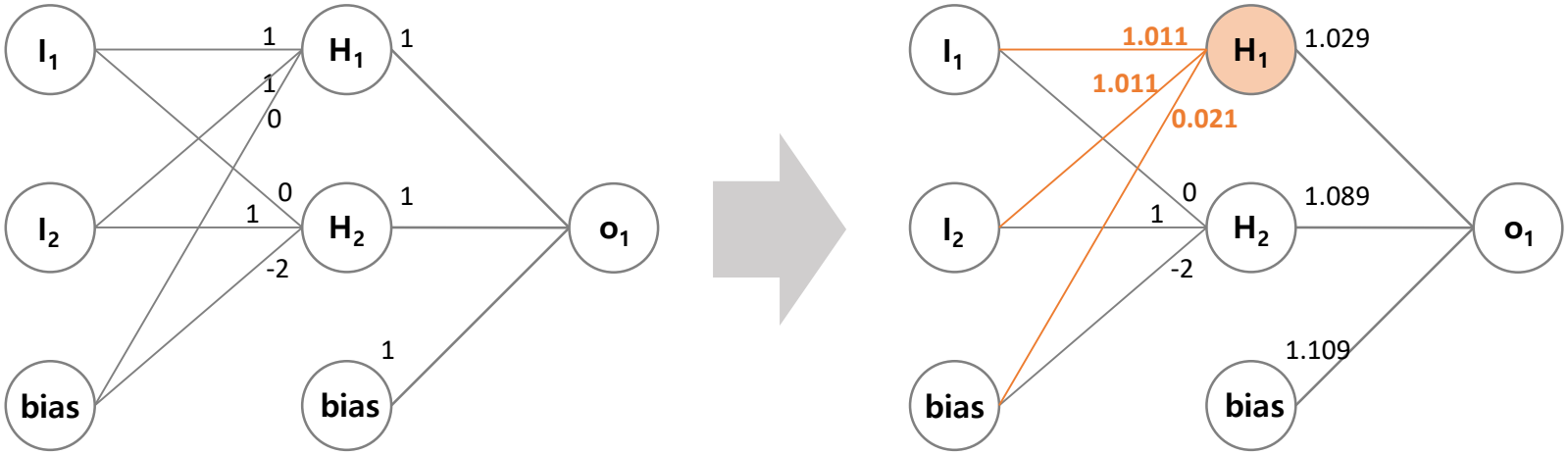
MLP: Training

- Error Back-Propagation: Example
  - ✓ Update the weights between the H_1 and the input nodes

$$w_{11}^{(1)}(new) = w_{11}^{(1)}(old) - \eta \times (y - \hat{y}) \times w_1^{(2)} \times h_1 \times (1 - h_1) \times x_1 = 1.011$$

$$w_{12}^{(1)}(new) = w_{12}^{(1)}(old) - \eta \times (y - \hat{y}) \times w_1^{(2)} \times h_1 \times (1 - h_1) \times x_2 = 1.011$$

$$w_{10}^{(1)}(new) = w_{10}^{(1)}(old) - \eta \times (y - \hat{y}) \times w_1^{(2)} \times h_1 \times (1 - h_1) \times b^{(1)} = 0.021$$



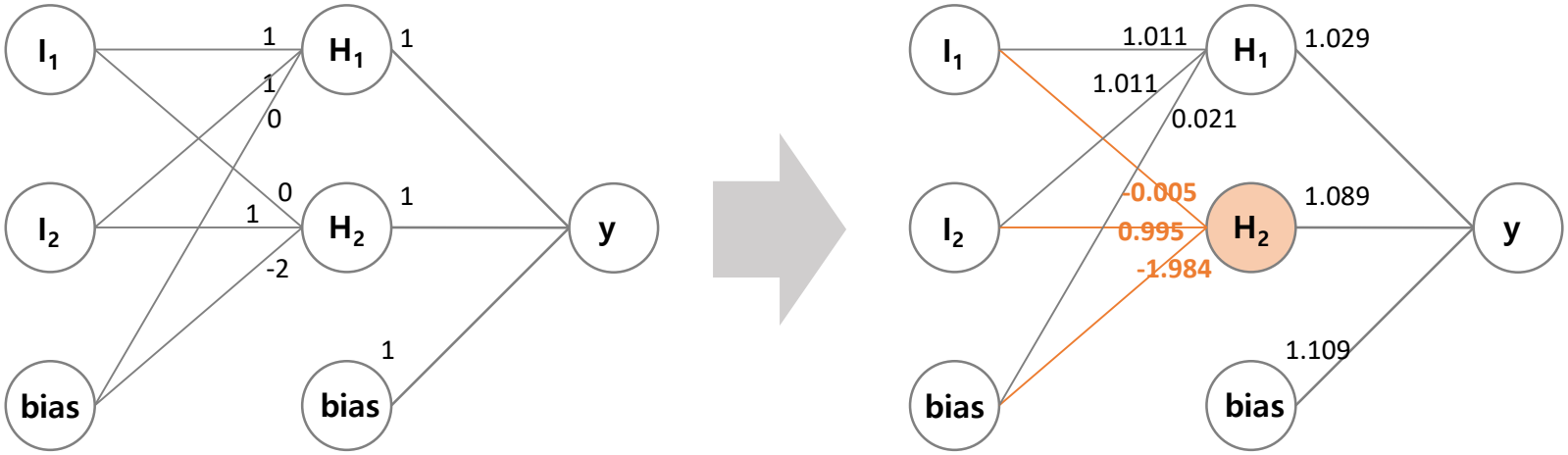
MLP: Training

- Error Back-Propagation: Example
  - ✓ Update the weights between the H_1 and the input nodes

$$w_{21}^{(1)}(new) = w_{21}^{(1)}(old) - \eta \times (y - \hat{y}) \times w_2^{(2)} \times h_2 \times (1 - h_2) \times x_1 = -0.005$$

$$w_{22}^{(1)}(new) = w_{22}^{(1)}(old) - \eta \times (y - \hat{y}) \times w_2^{(2)} \times h_2 \times (1 - h_2) \times x_2 = 0.995$$

$$w_{20}^{(1)}(new) = w_{20}^{(1)}(old) - \eta \times (y - \hat{y}) \times w_2^{(2)} \times h_2 \times (1 - h_2) \times b^{(1)} = -1.984$$



MLP: Training

- Goal
  - ✓ Find the weights that yield best predictions
- Features
  - ✓ The process described before is repeated for all records
  - ✓ At each record, compare the prediction to the actual target
  - ✓ Difference is the error for the output node
  - ✓ Error is propagated back and distributed to all the hidden nodes and used to update their weights

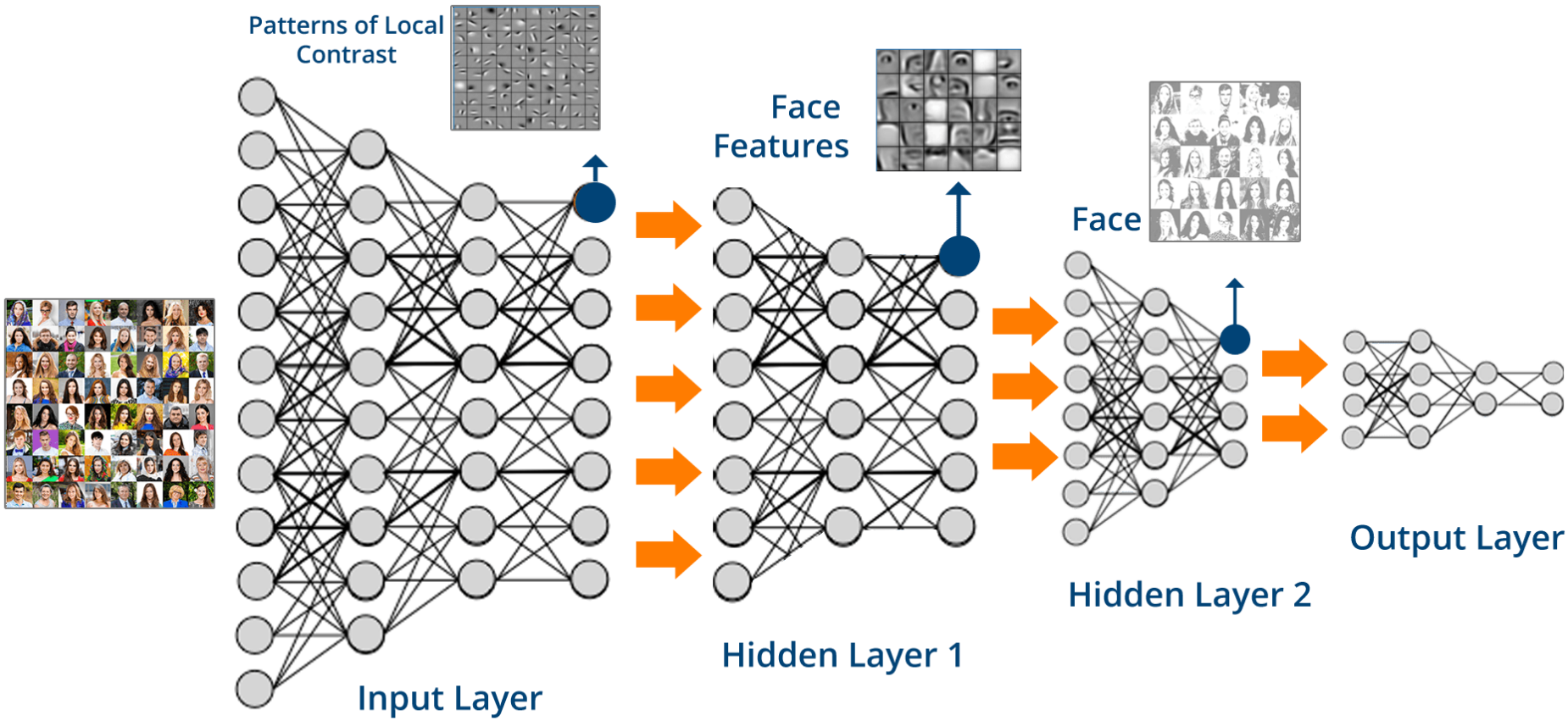
MLP: Training

- Why it works
  - ✓ Big errors lead to big changes in weights
  - ✓ Small errors leave weights relatively unchanged
  - ✓ Over thousand of updates, a given weight keeps changing until the error associated with it is negligible

- Common criteria to stop updating
  - ✓ When weights change very little from one epoch to the next
  - ✓ When the misclassification rate reaches a required threshold
  - ✓ When a limit on runs is reached

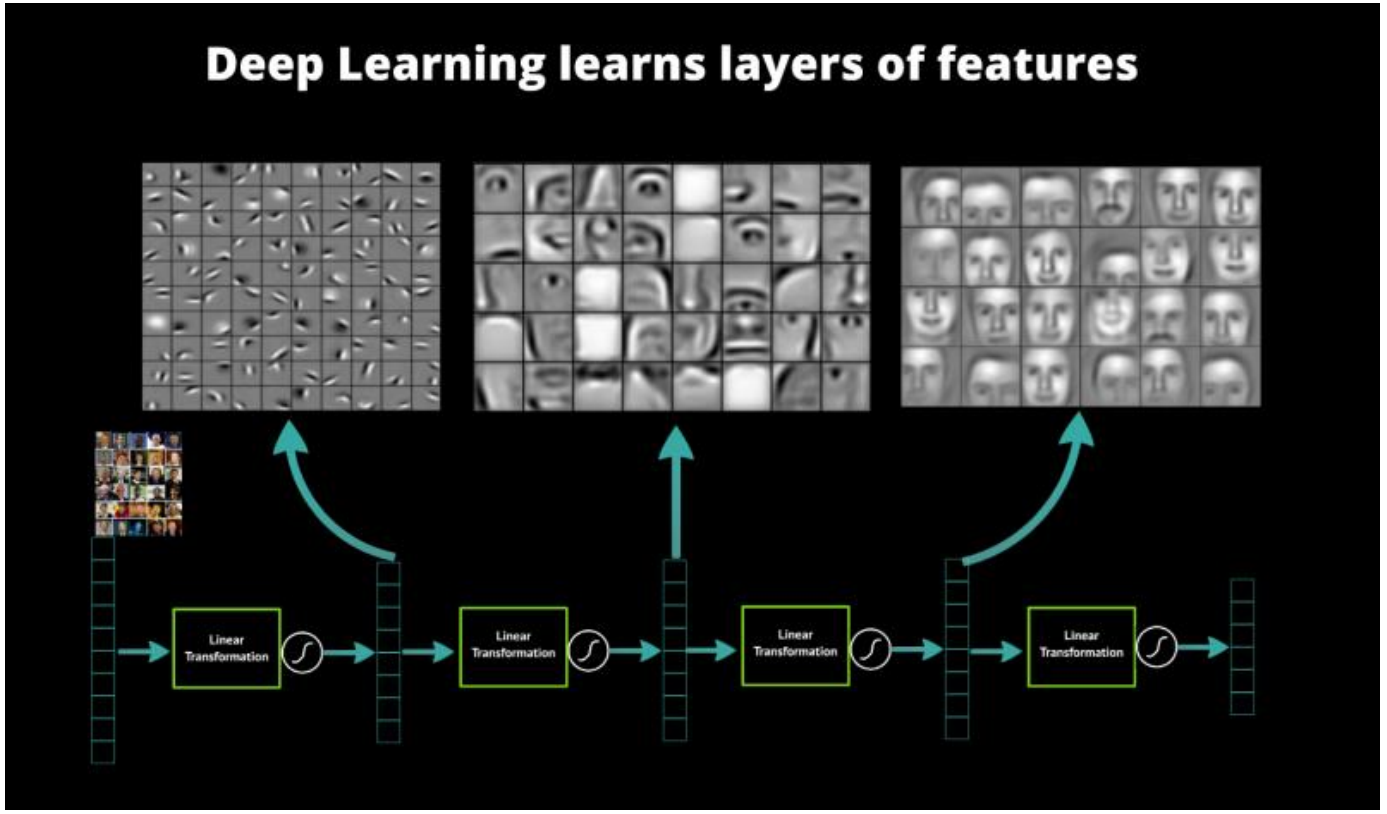
Deep Neural Network (DNN)

- Deep Neural Network (DNN)
  - ✓ Neural Network with many layers



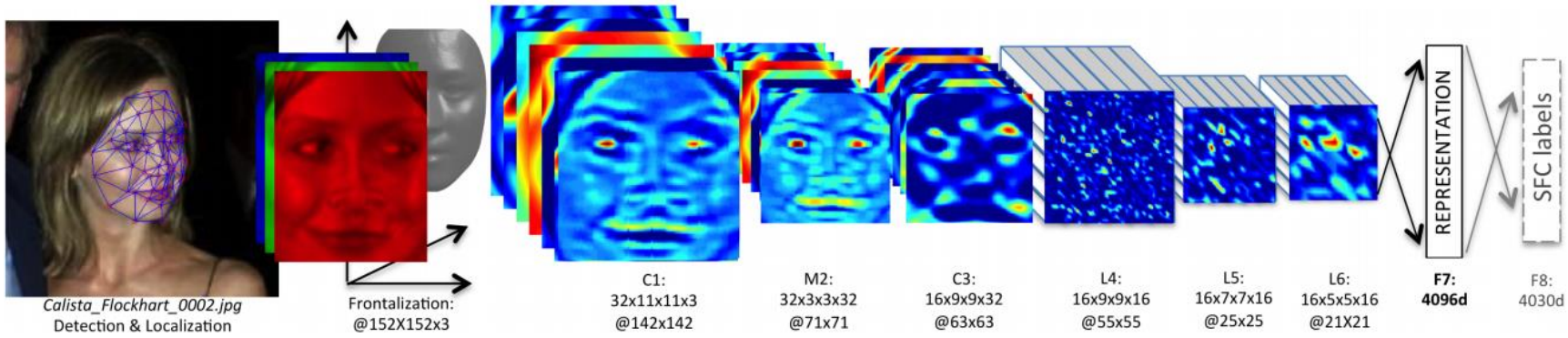
Deep Neural Network (DNN)

- Why DNN?
  - ✓ Be able to find significant features from raw data without hand-crafted engineering from high dimensional data (images, speeches, texts, etc.)



Deep Neural Network (DNN)

- Why DNN?
  - ✓ Much better performance than classic machine learning algorithms
    - Even outperform humans in some tasks (ex: image classification)



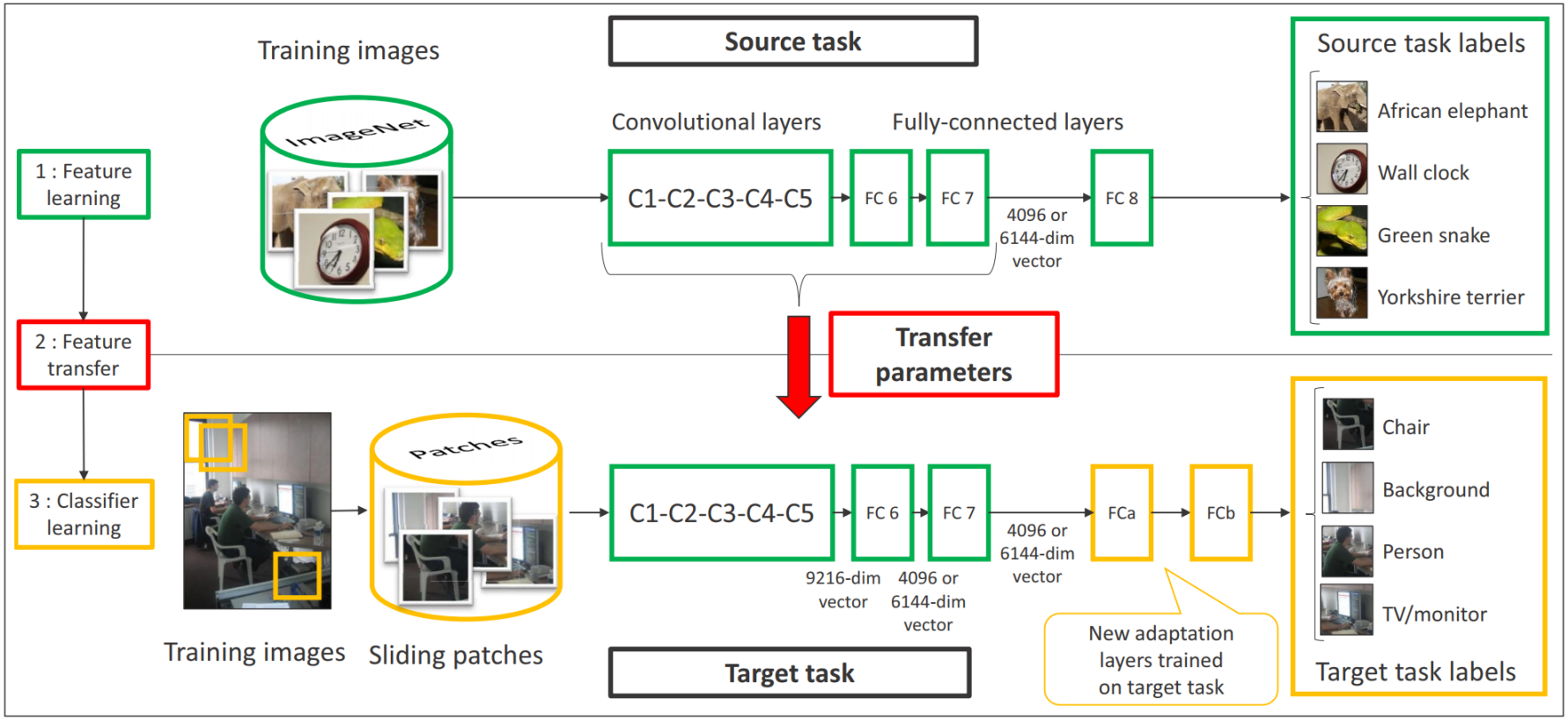
Taigman, Y. et al. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification, CVPR'14.

Recognition Accuracy for Labeled Faces in the Wild (LFW) dataset (13,233 images, 5,749 people)

- Human: 95% vs. DeepFace in Facebook: 97.35%

Deep Neural Network (DNN)

- Why DNN?
  - ✓ Be able to utilize learned knowledge in one domain to learn another domain (Transfer Learning)



Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1717-1724).

Deep Neural Network (DNN)

- Why DNN?
  - ✓ Work well with those tasks that people can do well inherently but machines have difficulty (visual/language recognition)

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

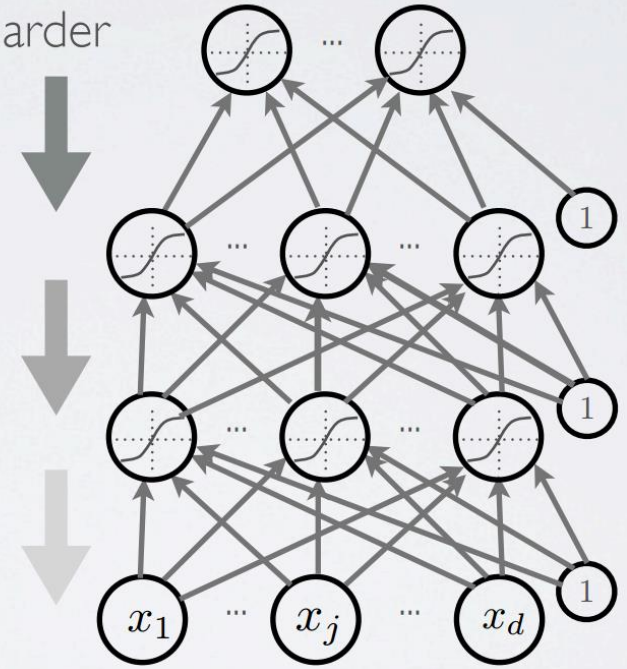
Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (pp. 2048-2057).

Deep Neural Network (DNN)

- What were the problems? (1) Training is difficult
  - ✓ Vanishing gradient problem: Gradient becomes close to 0 if many hidden layers are used → Errors cannot be propagated

Topics: why training is hard

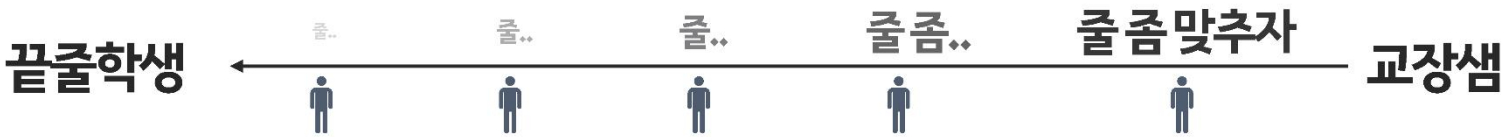
- First hypothesis: optimization is harder (underfitting)
  - vanishing gradient problem
  - saturated units block gradient propagation
- This is a well known problem in recurrent neural networks

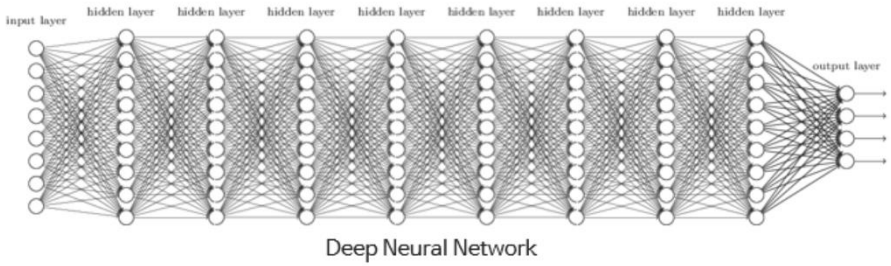


https://dl.dropboxusercontent.com/u/19557502/7_02_difficulty_of_training.pdf

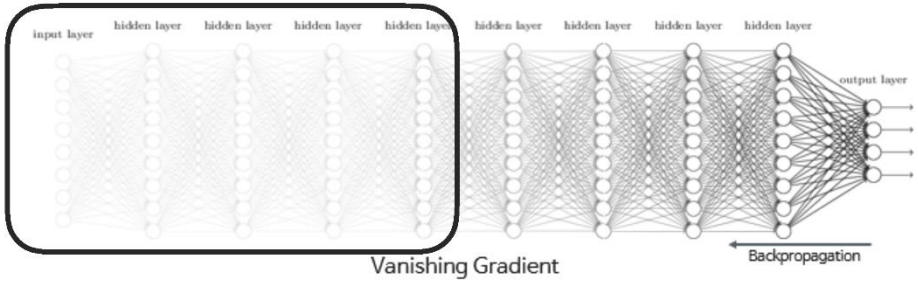
Deep Neural Network (DNN)

Vanishing gradient 현상 : 레이어가 깊을 수록 업데이트가 사라져간다.
그래서 fitting이 잘 안됨(underfitting)





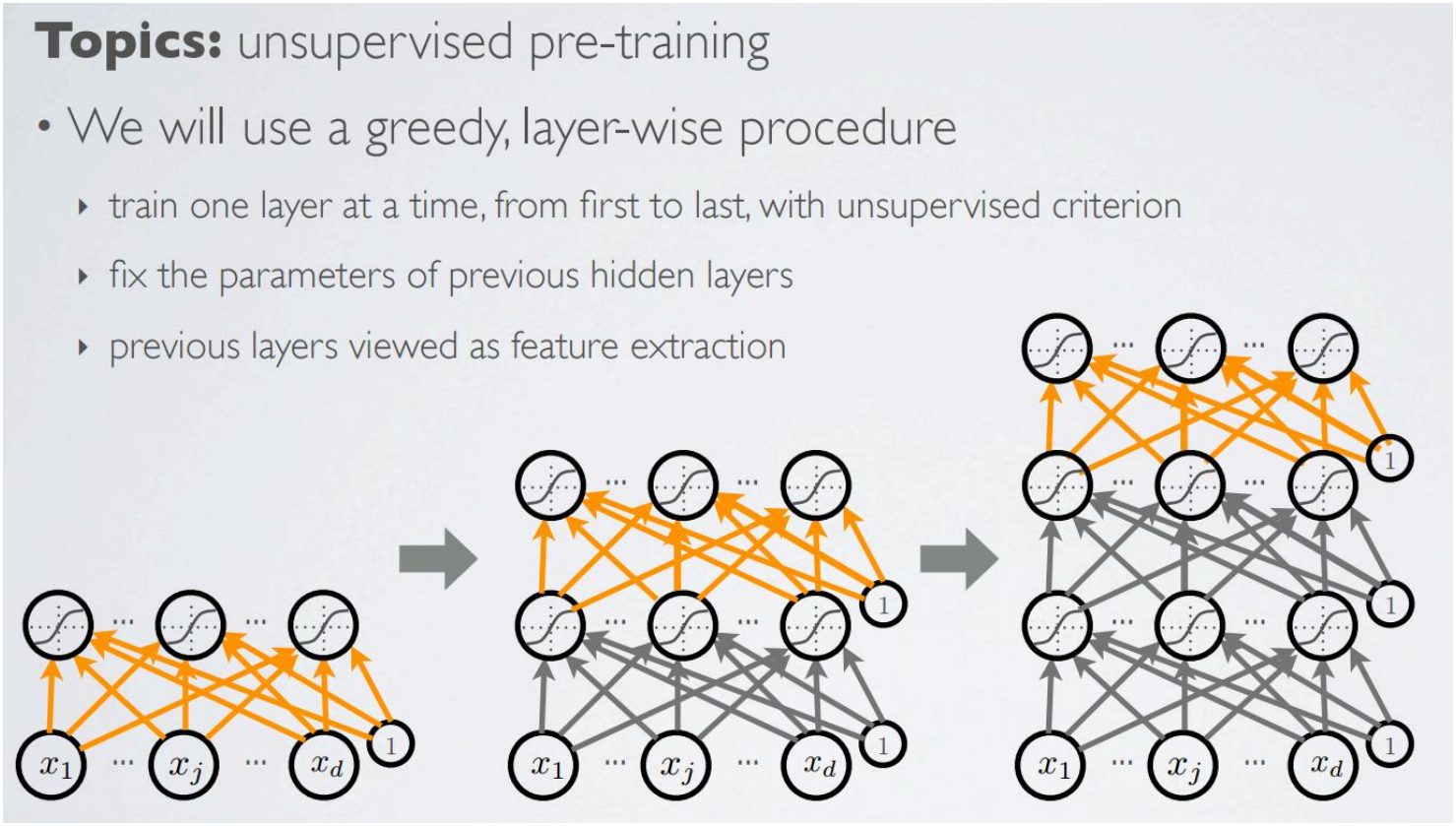
학습이 잘 안됨



하용호, 백날 자습해도 이해 안가던 딥러닝, 머리속에 인스톨 시켜드립니다

Deep Neural Network (DNN)

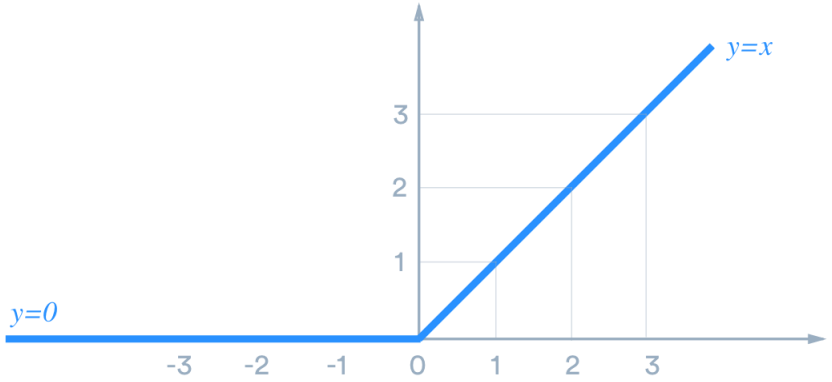
- Solution 1-1: Layer-wise training
  - ✓ User an energy function to stabilize the network between two adjacent layers
  - ✓ Prof. Geoffery Hinton's team first succeeded it: Not commonly used nowadays

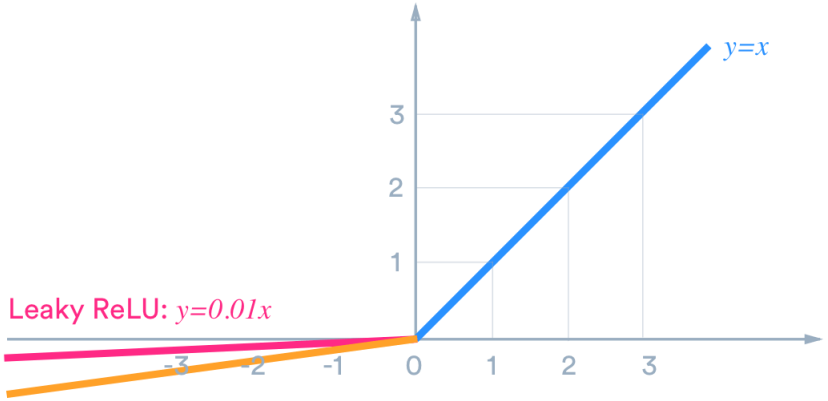


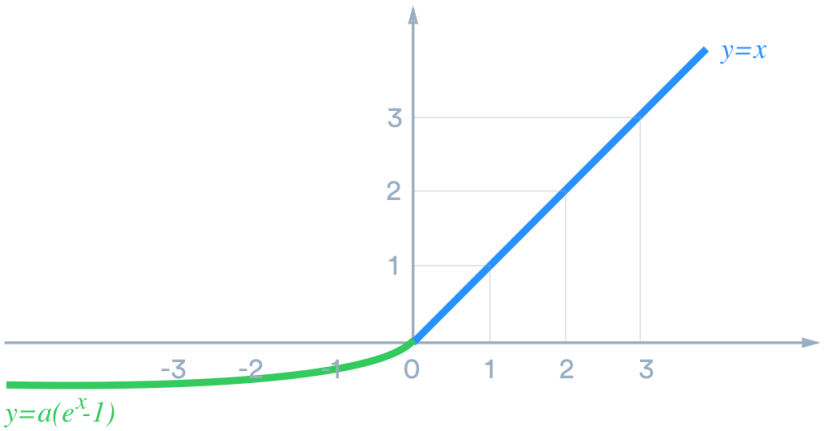
https://dl.dropboxusercontent.com/u/19557502/7_03_unsupervised_pretraining.pdf

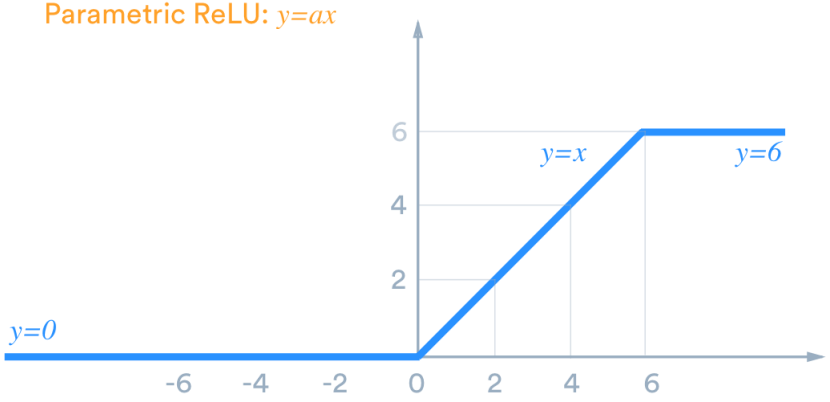
Deep Neural Network (DNN)

- Solution 1-2: Use another activation function that can send the gradient farther
  - ✓ Lectified Linear Unit (ReLU): de facto activation function nowadays





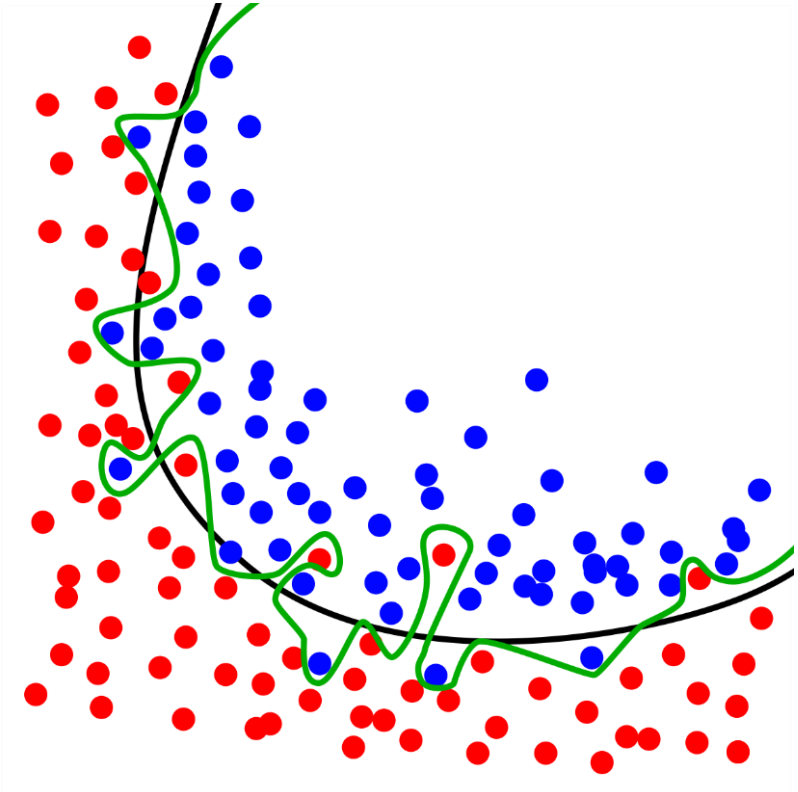




https://www.tinymind.com/learn/terms/relu

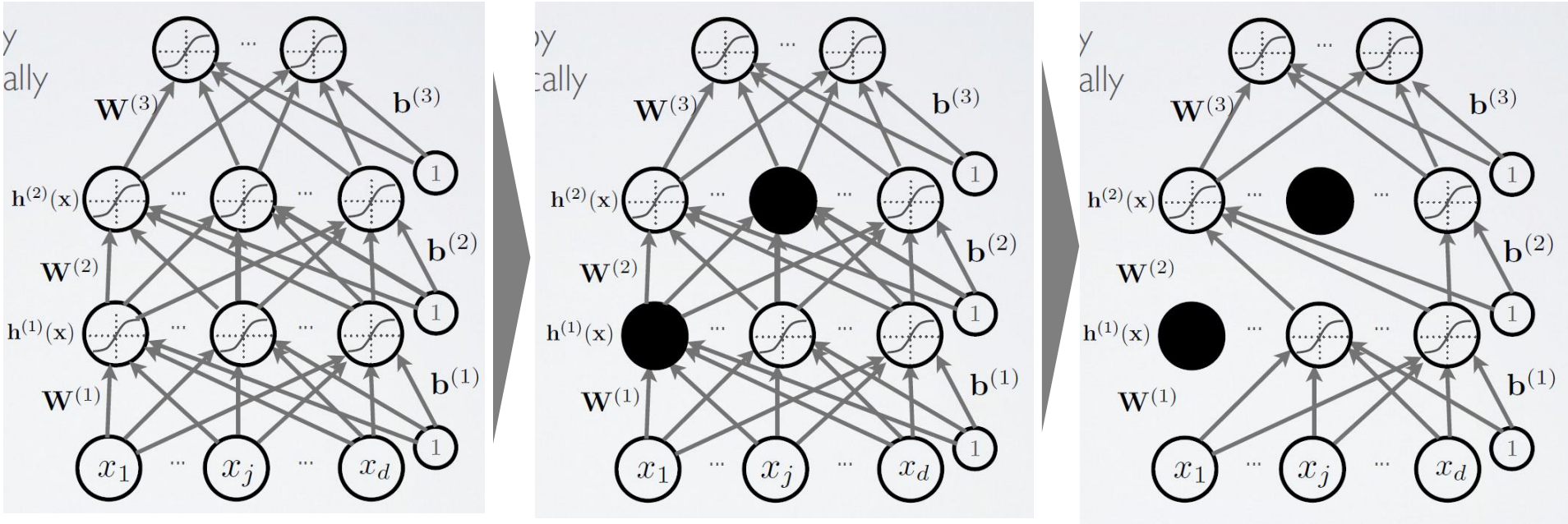
Deep Neural Network (DNN)

- What were the problems? (2) Overfitting
  - ✓ If you increase the number of hidden layers/nodes, you can find an arbitrary function (regression) or decision boundary (classification)
  - ✓ Not a desired property especially when the number of training examples are not sufficient
    - Need the black decision boundary
    - DNN finds the green one



Deep Neural Network (DNN)

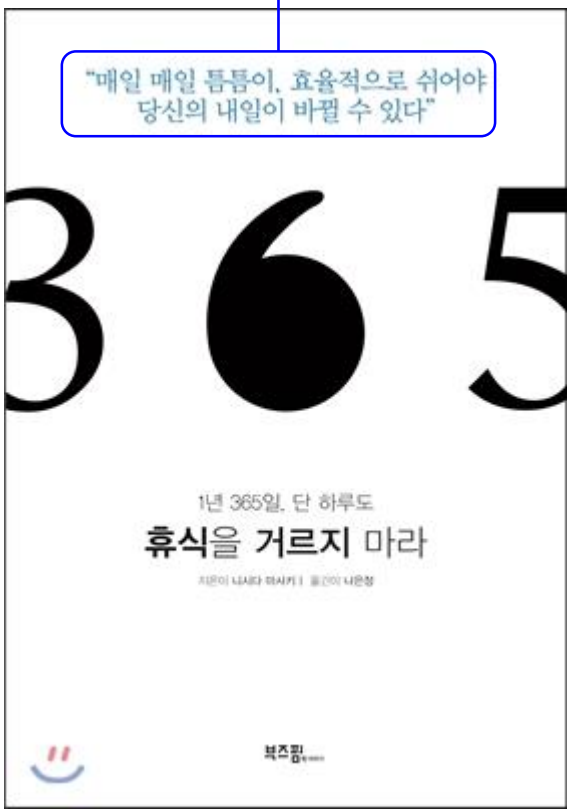
- Solution 2: Drop-Out
  - ✓ Sum of partial optimums ≠ Global optimum
  - ✓ Do not update the weights connected to certain nodes
  - ✓ Randomly change those nodes during training



Deep Neural Network (DNN)

- Solution 2: Drop-Out
  - ✓ Sum of partial optimums ≠ Global optimum

사람도 쉬고, hidden node도 쉬고…

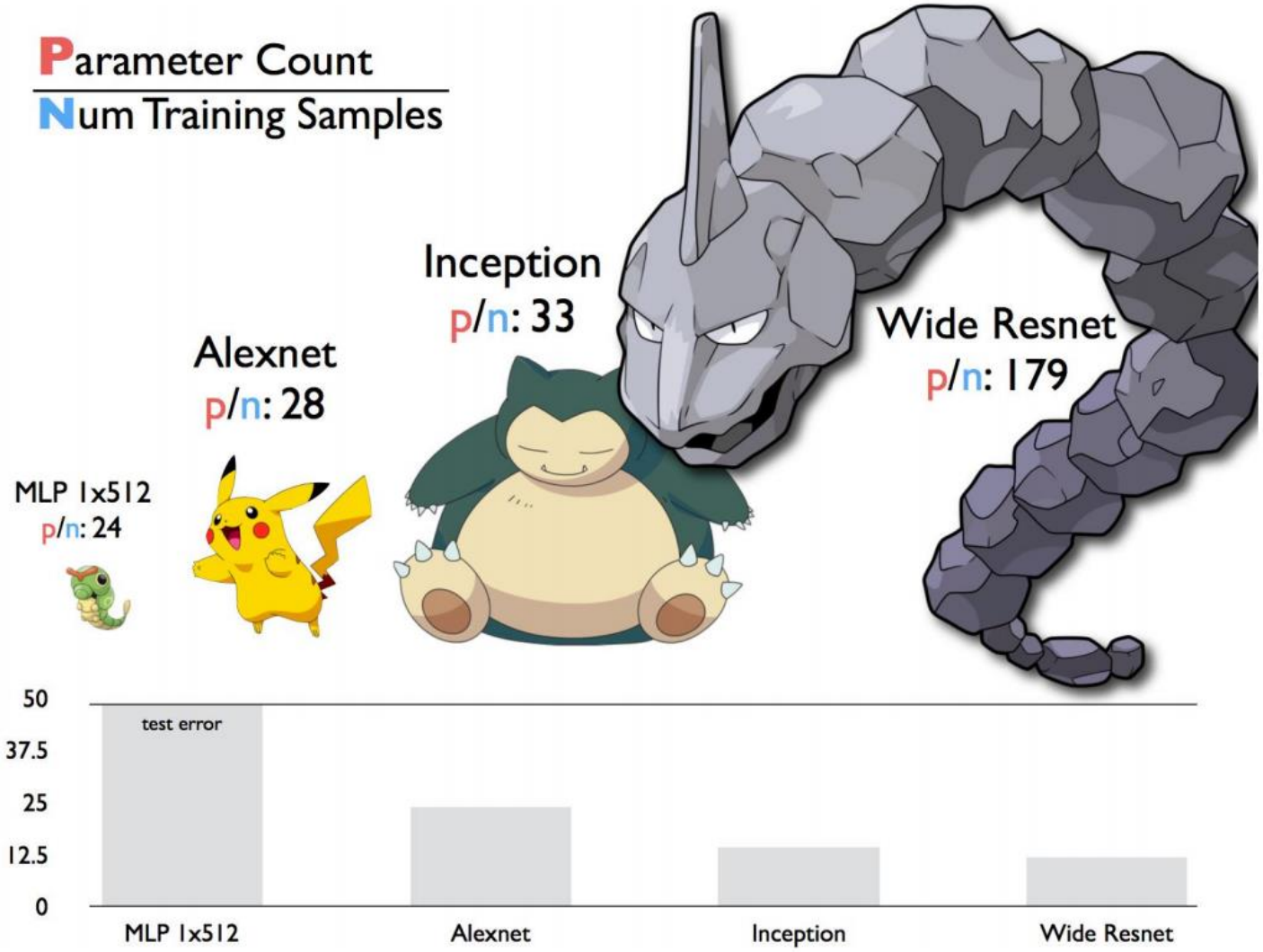


https://www.superookie.com/contents/58d3d99f8b129f44aa761811

http://www.yes24.com/24/goods/13144172

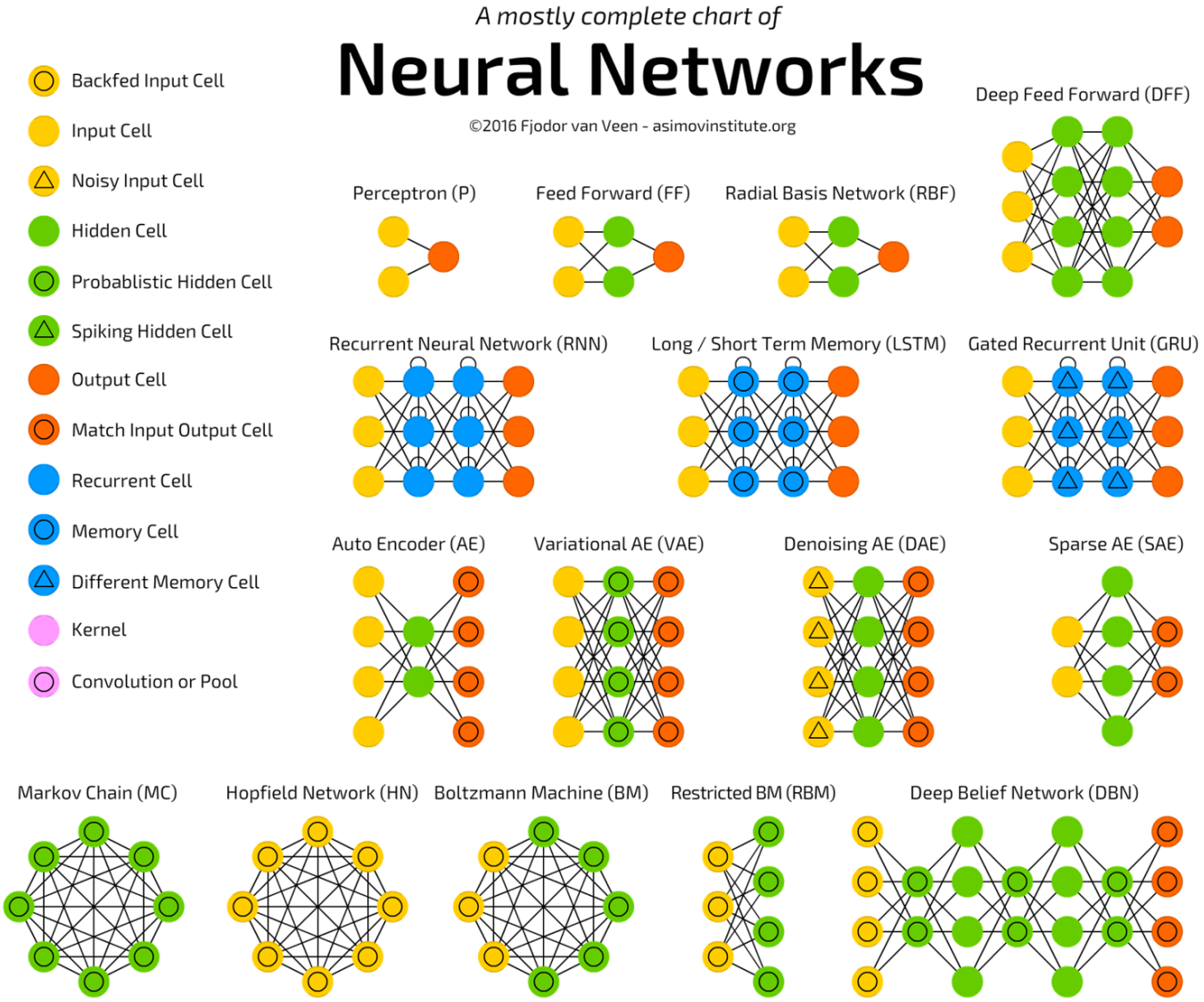
Deep Neural Network (DNN)

- Effects of high capacity models without overfitting



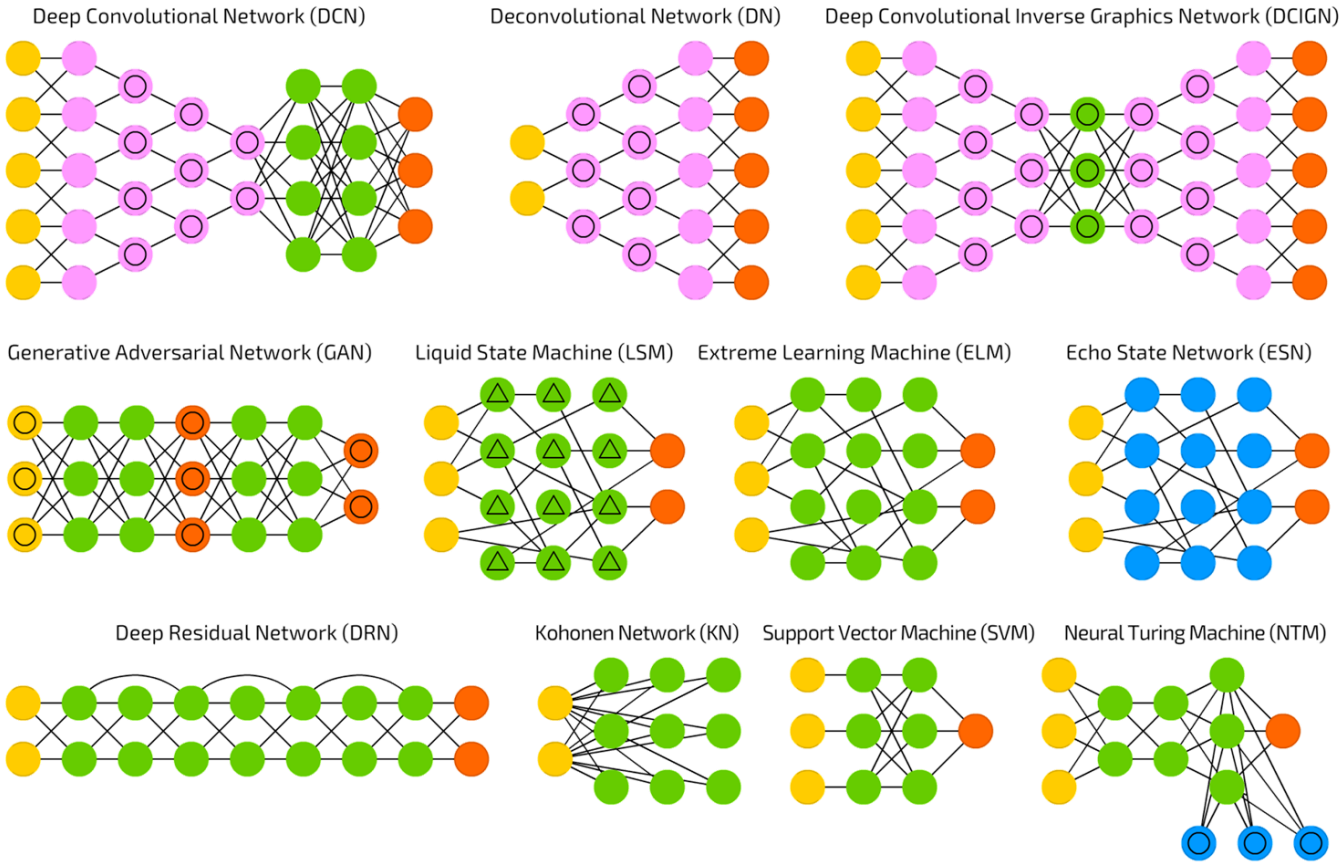
Neural Network Zoo

- Various Structure of Artificial Neural Networks



Neural Network Zoo

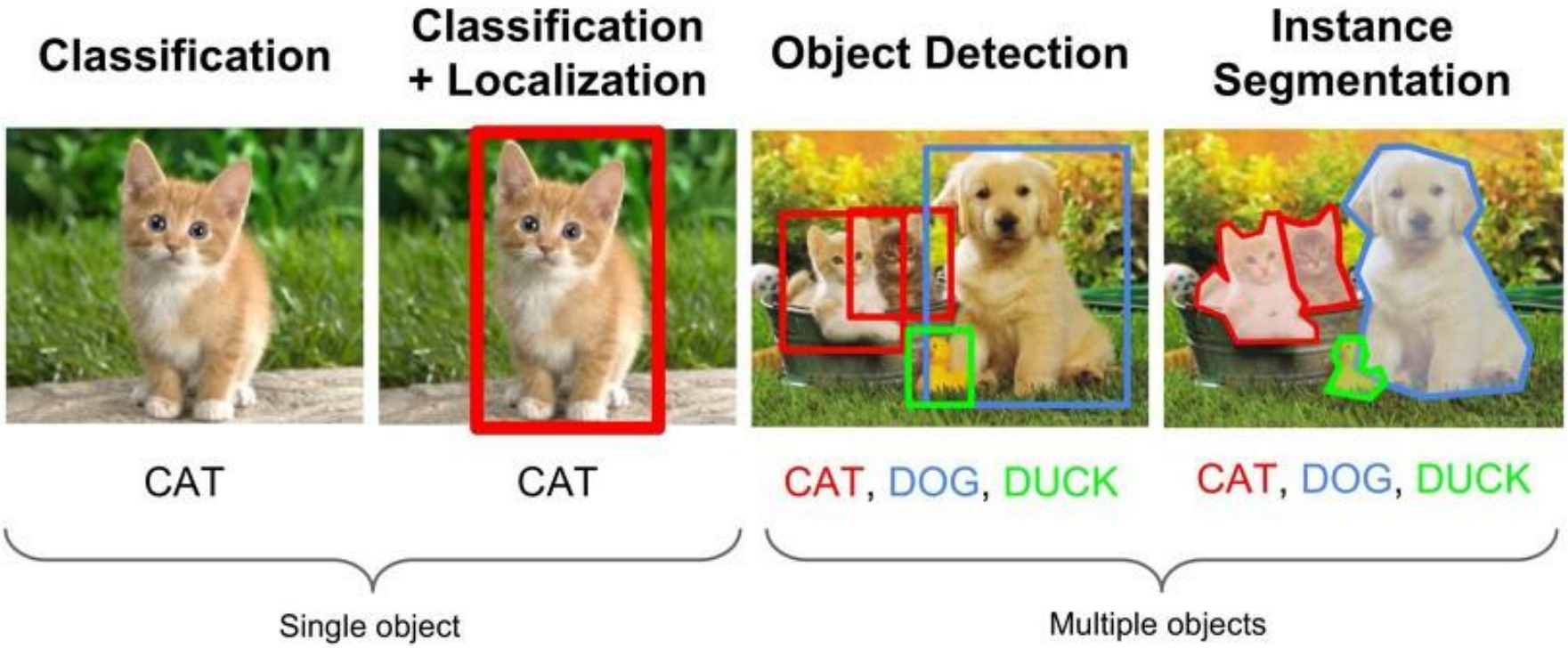
- Various Structure of Artificial Neural Networks



AGENDA

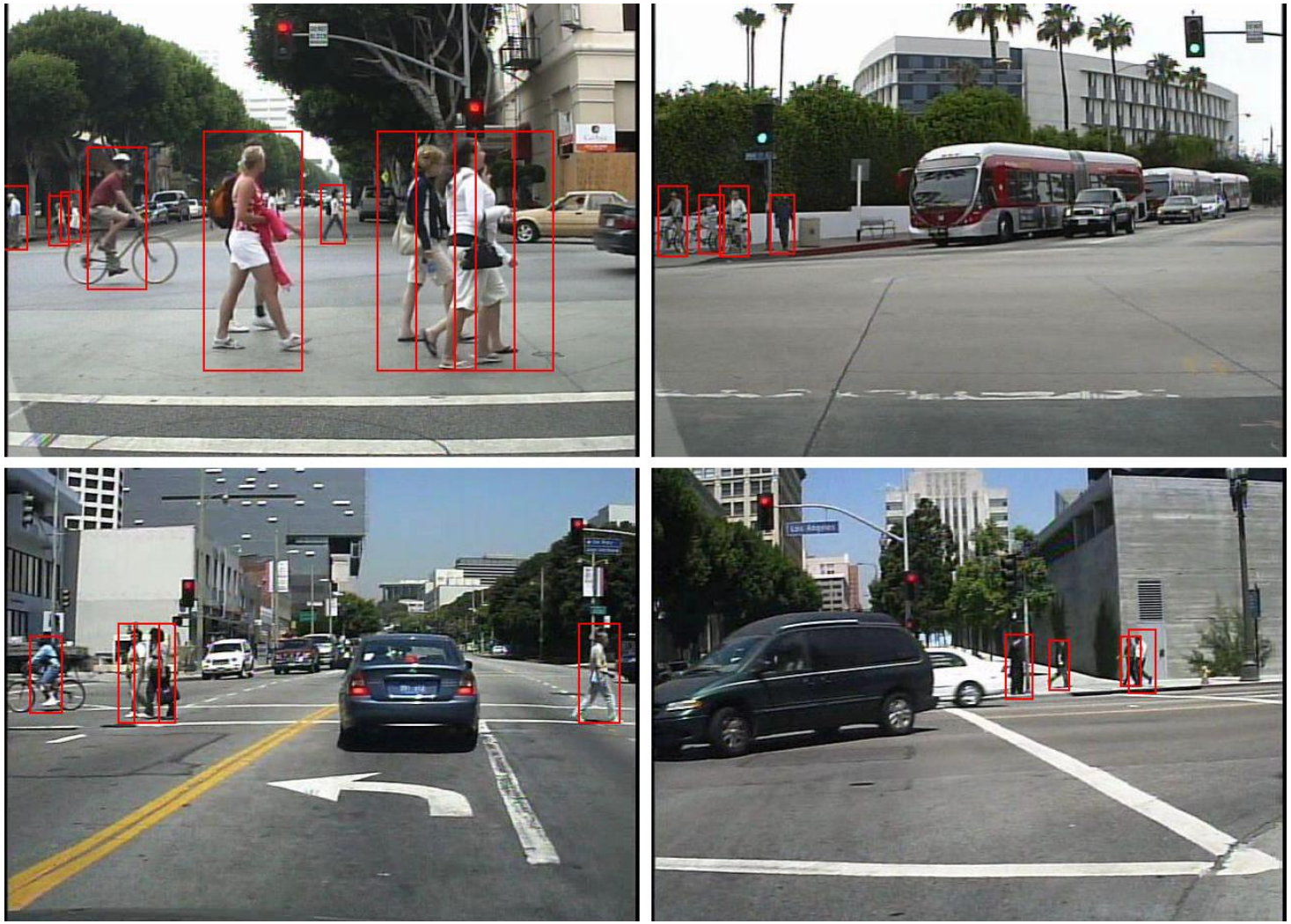
Convolutional Neural Network (CNN)

- Convolutional Neural Network
  - ✓ Able to learn useful features from images with convolution operations
- Category of Image Recognitions



Convolutional Neural Network (CNN)

- CNN Example: Pedestrian recognition (for autonomous driving car)



Convolutional Neural Network (CNN)

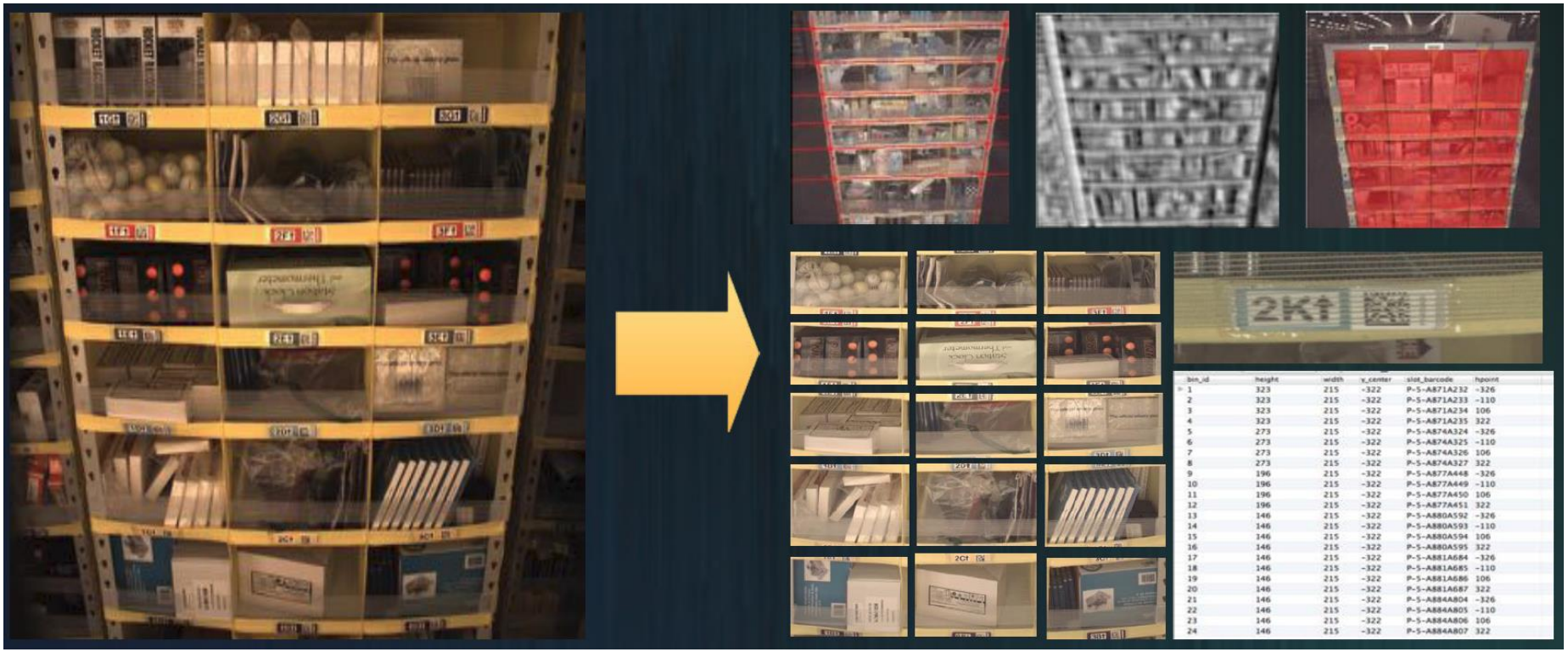
- CNN Example: Face and emotion detection

Who are absent or late?

How boring is the class?

Convolutional Neural Network (CNN)

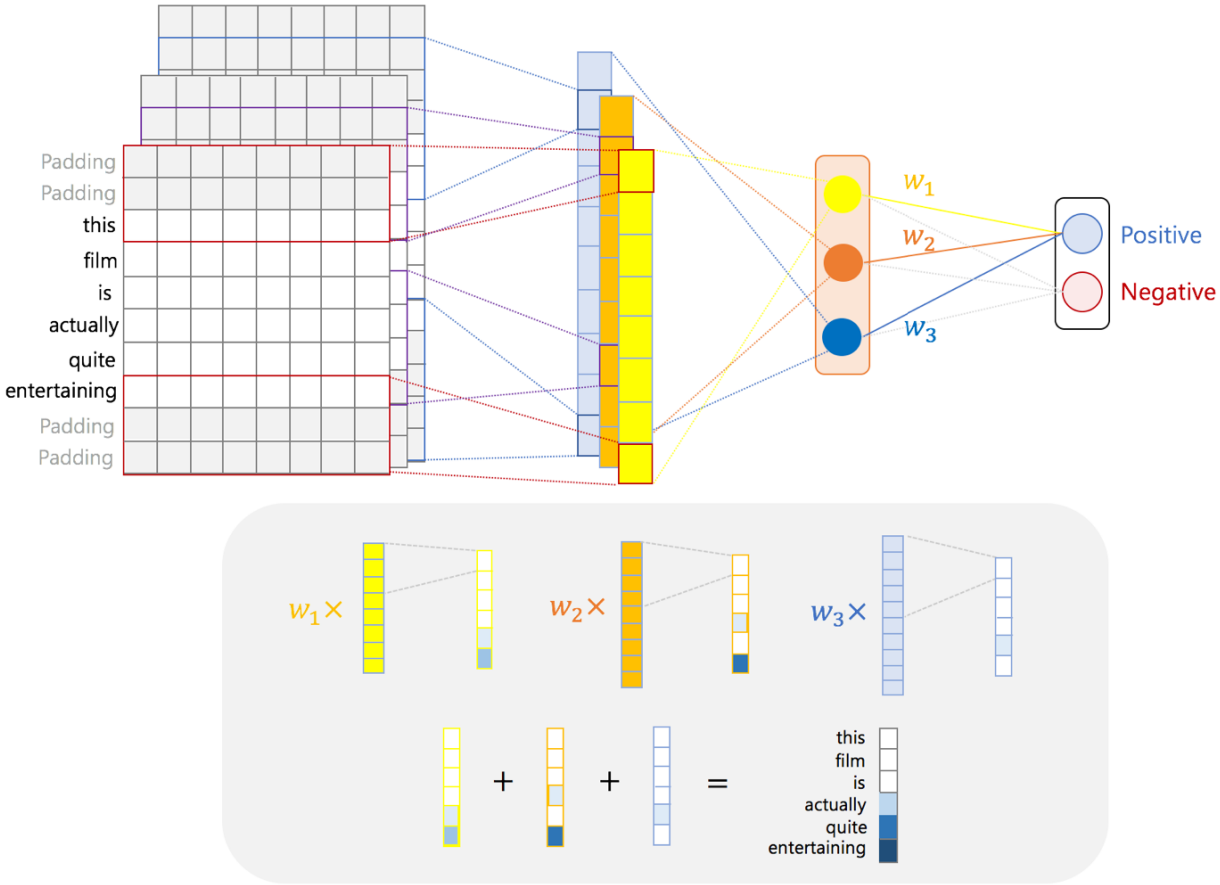
- CNN Example: Inventory



https://www.slideshare.net/awskorea/amazon-deeplearningcasesandmlonaws

Convolutional Neural Network (CNN)

- CNN for Text Analytics: Sentiment Classification and Word Localization



Lee, G., Jeong, J., Seo, S., Kim, C, & Kang, P. (2018). Sentiment classification with word localization based on weakly supervised learning with a convolutional neural network. Knowledge-Based Systems, 152, 70-82.

Convolutional Neural Network (CNN)

- CNN for Text Analytics: Sentiment Classification and Word Localization

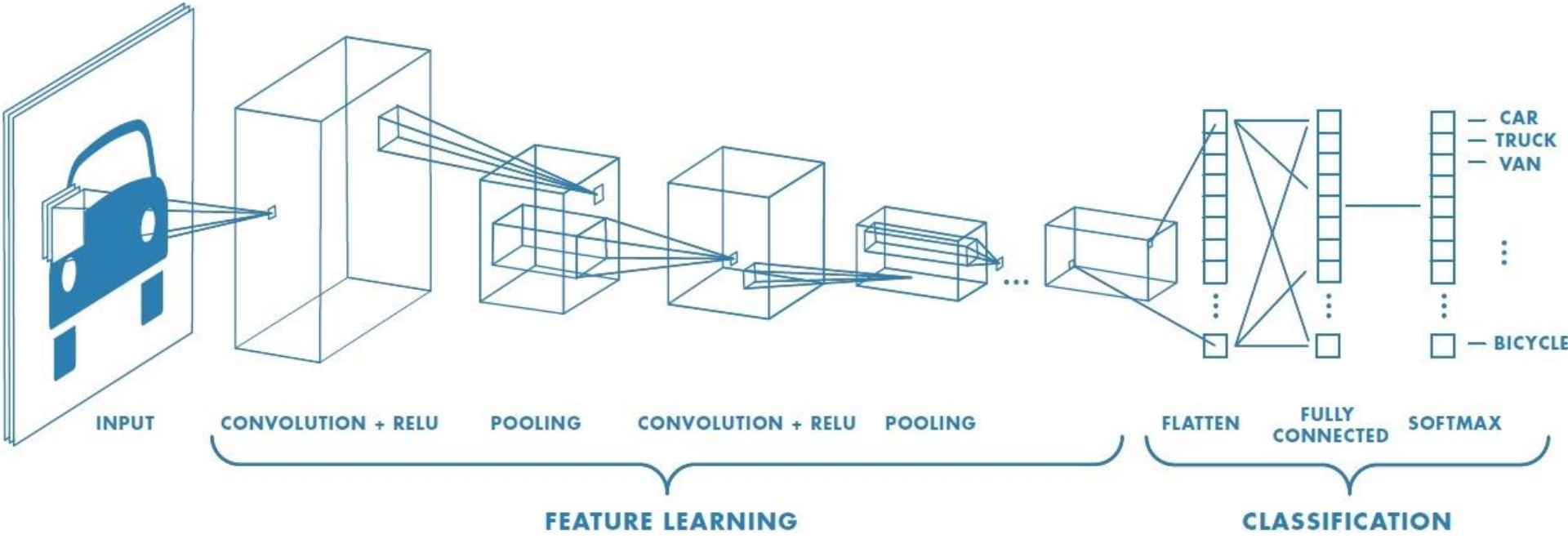
CAM²-4channel Seeing as the vote average was pretty low and the fact that the clerk in the video store thoug ht was just OK I didn t t have much expectations when renting this film But contrary to the a bove I **enjoyed it** a lot This is acharming movie It didn t need to grow on me **I enjoyed it** fro m the beginning Mel Brooks gives a great performance as thelead character I think somewha t different from his usual persona in his movies There s not a lot of knockout jokes or someth inglike that but there are some rather hilarious scenes and overall this is a **very enjoyable an d very easy** to watch film **Very recommended** **Positive**

CAM²-4channel I hate this movie It is a horrid movie Sean Young s character is completely unsympathetic He r performance is wooden at best The storyline is completely **predictable and completely uni nteresting** I would never recommend this film to anyone It is one of the **worst** movies I have ever had the misfortune to see **Negative**

Lee, G., Jeong, J., Seo, S., Kim, C, & Kang, P. (2018). Sentiment classification with word localization based on weakly supervised learning with a convolutional neural network. Knowledge-Based Systems, 152, 70-82.

Convolutional Neural Network (CNN)

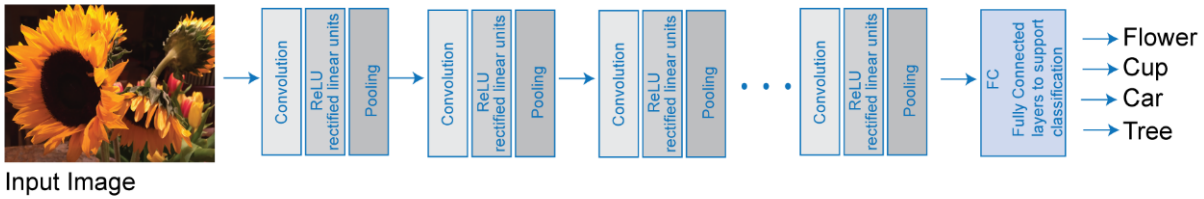
- Information processing by CNN
  - ✓ Phase 1 (Feature Learning): A set of convolution operation, activation operation (mostly ReLU), pooling operation
  - ✓ Phase 2: Flatten the learned feature (tensor) to 1-dimentional vector to do the required task

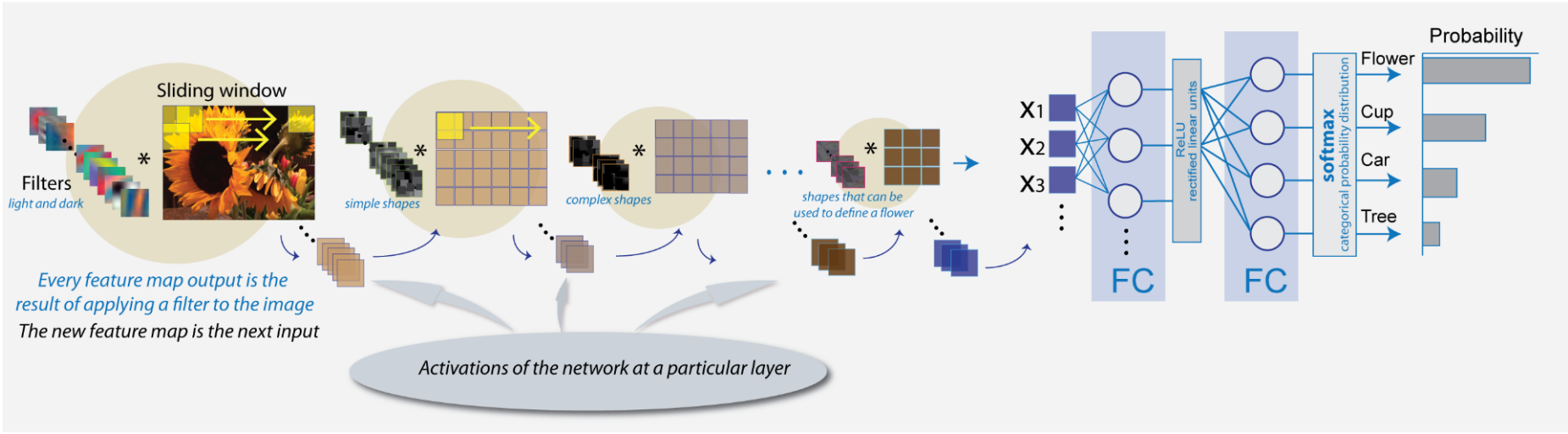


https://kr.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Convolutional Neural Network (CNN)

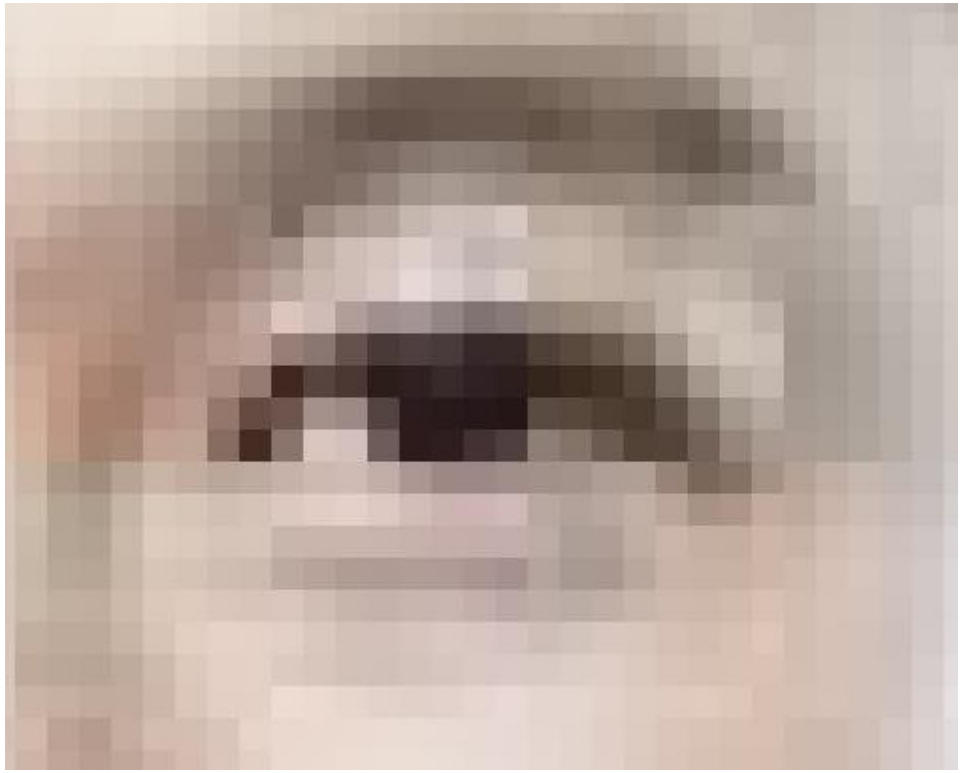
- Information processing by CNN
  - ✓ Phase 1 (Feature Learning): A set of convolution operation, activation operation (mostly ReLU), pooling operation
  - ✓ Phase 2: Flatten the learned feature (tensor) to 1-dimentional vector to do the required task





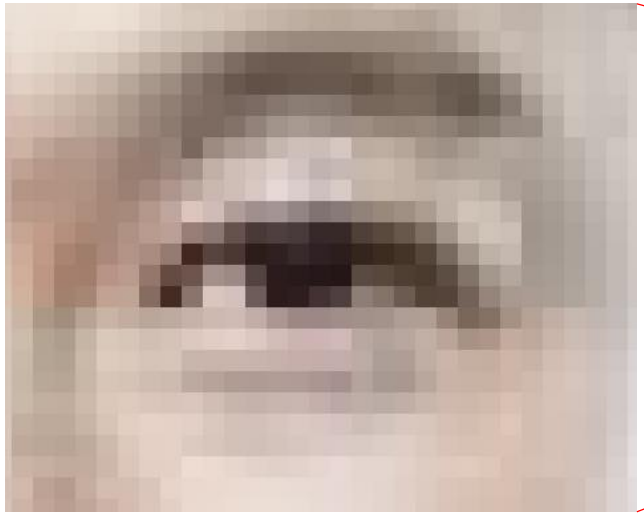
CNN Basics: Image Representation

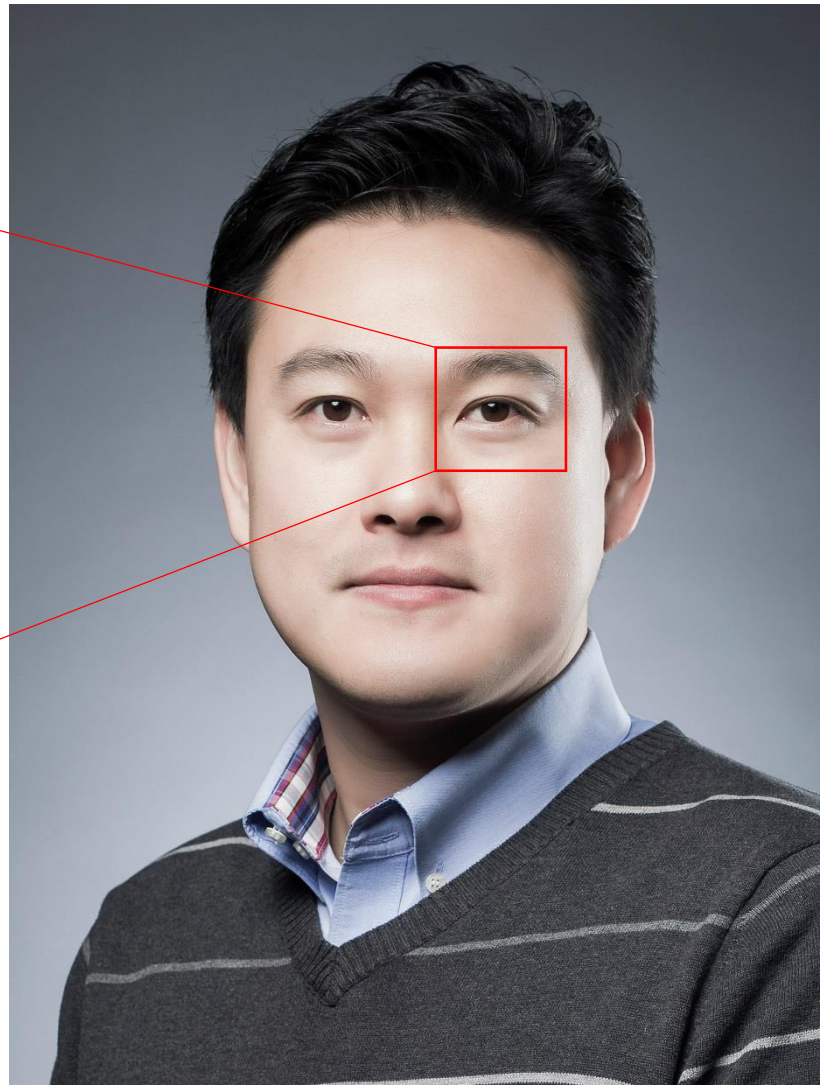
- Whose eye is it?



CNN Basics: Image Representation

- Whose eye is it?





CNN Basics: Image Representation

- A color image is a tensor with the size of (Width X Height X 3 (RGB))

1,685

2,247

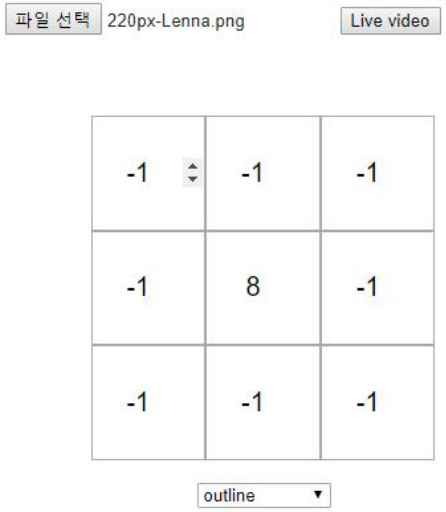
```
> pskang[1:10, 1:10, 1]
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
 [1,] 0.2784314 0.2784314 0.2745098 0.2745098 0.2745098 0.2745098 0.2705882 0.2705882 0.2862745 0.2901961
 [2,] 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2666667 0.2705882
 [3,] 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2705882 0.2745098
 [4,] 0.2705882 0.2705882 0.2745098 0.2745098 0.2745098 0.2745098 0.2784314 0.2784314 0.2823529 0.2823529
 [5,] 0.2705882 0.2705882 0.2705882 0.2745098 0.2745098 0.2784314 0.2784314 0.2784314 0.2784314 0.2784314
 [6,] 0.2705882 0.2705882 0.2745098 0.2745098 0.2745098 0.2745098 0.2784314 0.2784314 0.2823529 0.2784314
 [7,] 0.2705882 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2784314 0.2941176 0.2862745
 [8,] 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2901961 0.2823529
 [9,] 0.2745098 0.2705882 0.2745098 0.2784314 0.2823529 0.2823529 0.2745098 0.2666667 0.2784314 0.2784314
[10,] 0.2784314 0.2784314 0.2784314 0.2823529 0.2862745 0.2862745 0.2784314 0.2745098 0.2745098 0.2745098
```

```
> pskang[1:10, 1:10, 2]
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
 [1,] 0.2980392 0.2980392 0.2941176 0.2941176 0.2941176 0.2941176 0.2901961 0.2901961 0.2980392 0.3019608
 [2,] 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2784314 0.2823529
 [3,] 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2823529 0.2862745
 [4,] 0.2901961 0.2901961 0.2941176 0.2941176 0.2941176 0.2941176 0.2980392 0.2980392 0.2941176 0.2941176
 [5,] 0.2901961 0.2901961 0.2901961 0.2941176 0.2941176 0.2980392 0.2980392 0.2980392 0.2901961 0.2901961
 [6,] 0.2901961 0.2901961 0.2941176 0.2941176 0.2941176 0.2941176 0.2980392 0.2980392 0.2941176 0.2901961
 [7,] 0.2901961 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2980392 0.3058824 0.2980392
 [8,] 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.3019608 0.2941176
 [9,] 0.2862745 0.2823529 0.2862745 0.2901961 0.2941176 0.2941176 0.2862745 0.2784314 0.2901961 0.2901961
[10,] 0.2901961 0.2901961 0.2901961 0.2941176 0.2980392 0.2980392 0.2901961 0.2862745 0.2862745 0.2862745
```

```
> pskang[1:10, 1:10, 3]
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
 [1,] 0.3215686 0.3215686 0.3176471 0.3176471 0.3176471 0.3176471 0.3137255 0.3137255 0.3254902 0.3294118
 [2,] 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3058824 0.3098039
 [3,] 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3098039 0.3137255
 [4,] 0.3137255 0.3137255 0.3176471 0.3176471 0.3176471 0.3176471 0.3215686 0.3215686 0.3215686 0.3215686
 [5,] 0.3137255 0.3137255 0.3137255 0.3176471 0.3176471 0.3215686 0.3215686 0.3215686 0.3176471 0.3176471
 [6,] 0.3137255 0.3137255 0.3176471 0.3176471 0.3176471 0.3176471 0.3215686 0.3215686 0.3215686 0.3176471
 [7,] 0.3137255 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3215686 0.3333333 0.3254902
 [8,] 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3294118 0.3215686
 [9,] 0.3058824 0.3019608 0.3058824 0.3098039 0.3137255 0.3137255 0.3058824 0.2980392 0.3098039 0.3098039
[10,] 0.3098039 0.3098039 0.3098039 0.3137255 0.3176471 0.3176471 0.3098039 0.3058824 0.3058824 0.3058824
```

CNN Basics: Convolution

- Problem
  - ✓ If all pixels are connected to the first hidden layer, we need to optimize too many network weights
- Image Convolution (Filter, Kernel)
  - ✓ A matrix that capture a certain type of image feature (ex: edge detection)



She appeared on the cover of the journal *Optical Engineering* in 1991, and the model herself was invited to a conference of the Image Science and Technology society in 1997.

https://www.software.ac.uk/blog/2016-10-03-how-photo-playboy-became-part-scientific-culture

CNN Basics: Convolution

- Image Convolution (Filter, Kernel)
  - ✓ A matrix that can detect a certain type of feature

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

CNN Basics: Convolution

- Image Convolution (Filter, Kernel)
  - ✓ A matrix that can detect a certain type of feature

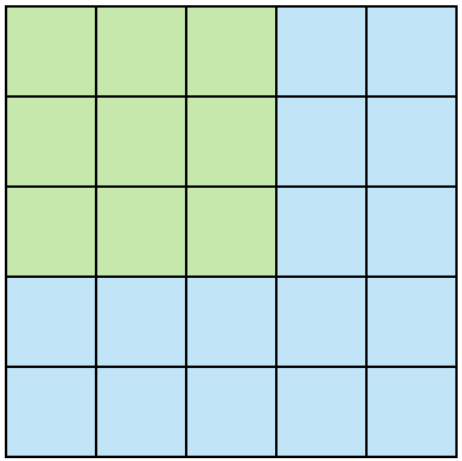
1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

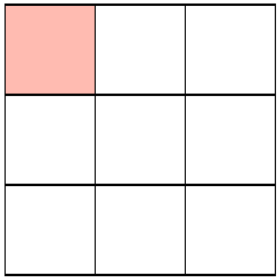
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

CNN Basics: Convolution

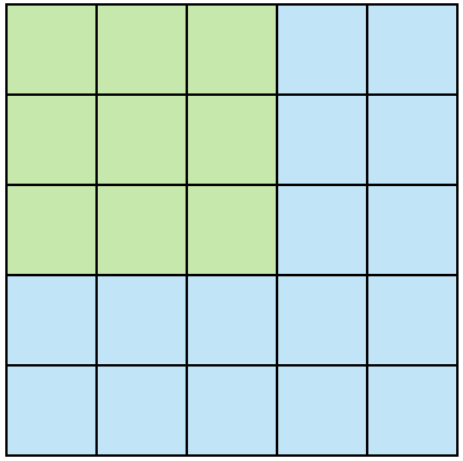
- Image Convolution (Filter, Kernel)
  - ✓ Issue: Takes too long time to move by one pixel
  - ✓ Solution: Allow filters to move by more than one pixel (stride)



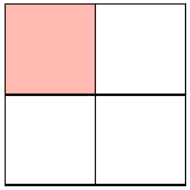
Stride 1



Feature Map



Stride 2



Feature Map

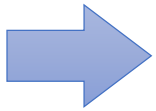
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

CNN Basics: Convolution

- Image Convolution (Filter, Kernel)
  - ✓ Issue: Pixels in the edge have less opportunities for convolution operation
  - ✓ Solution: Padding (add a pad with zero values)

3	1	1	2	8	4
1	0	7	3	2	6
2	3	5	1	1	3
1	4	1	2	6	5
3	2	1	3	7	2
9	2	6	2	5	1

Padding with size 1

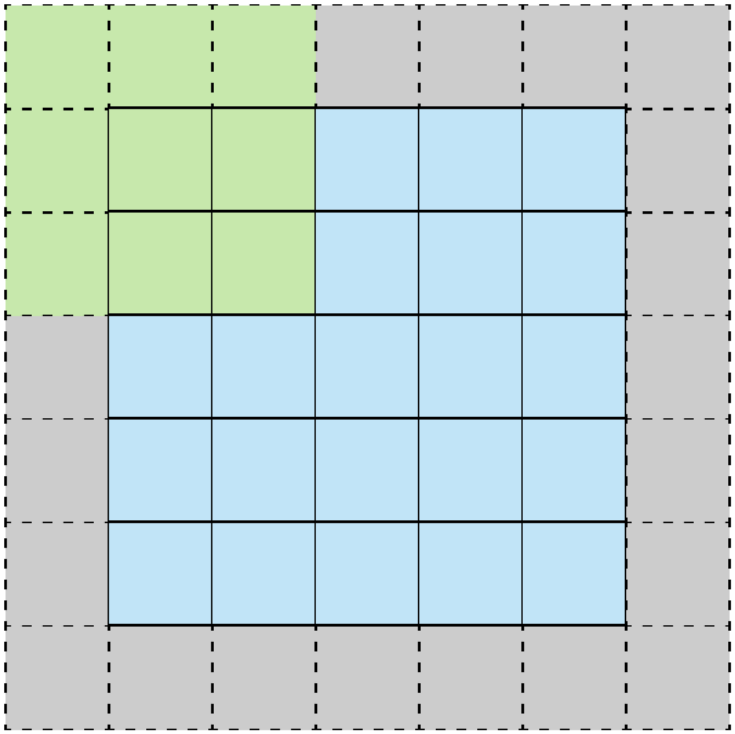


0	0	0	0	0	0	0	0
0	3	1	1	2	8	4	0
0	1	0	7	3	2	6	0
0	2	3	5	1	1	3	0
0	1	4	1	2	6	5	0
0	3	2	1	3	7	2	0
0	9	2	6	2	5	1	0
0	0	0	0	0	0	0	0

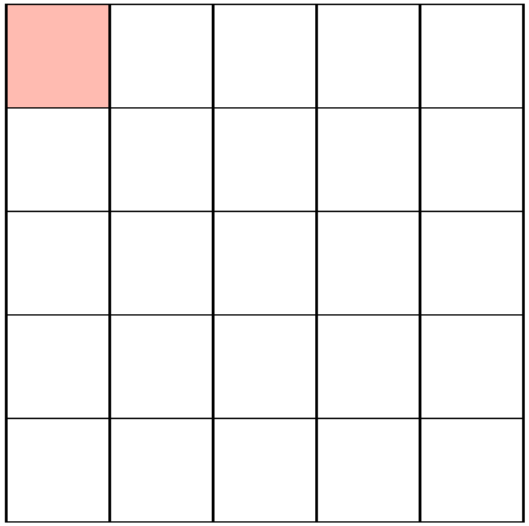
https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

CNN Basics: Convolution

- Image Convolution (Filter, Kernel)
  - ✓ Convolution with padding



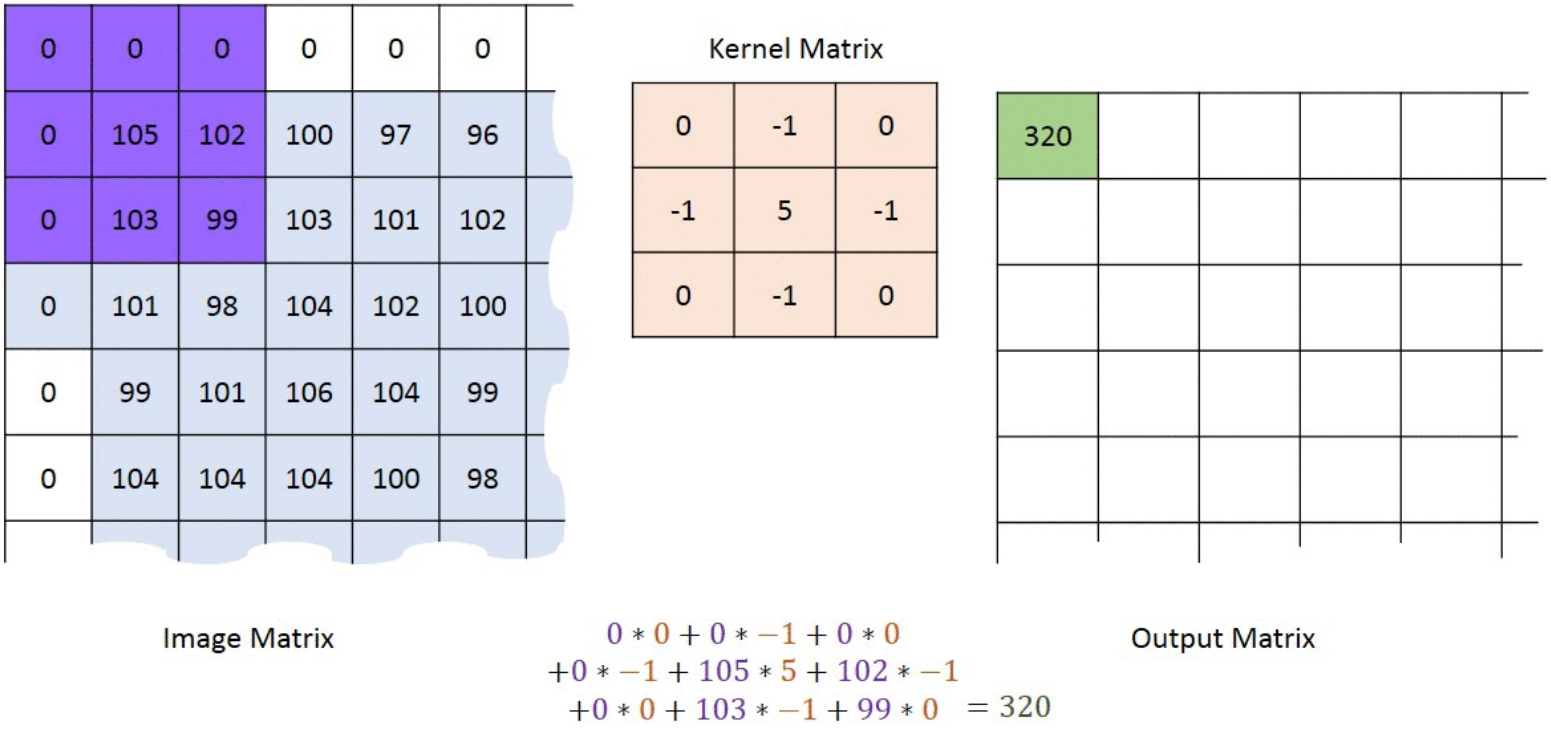
Stride 1 with Padding



Feature Map

CNN Basics: Convolution

- Image Convolution (Filter, Kernel)
  - ✓ Convolution with padding

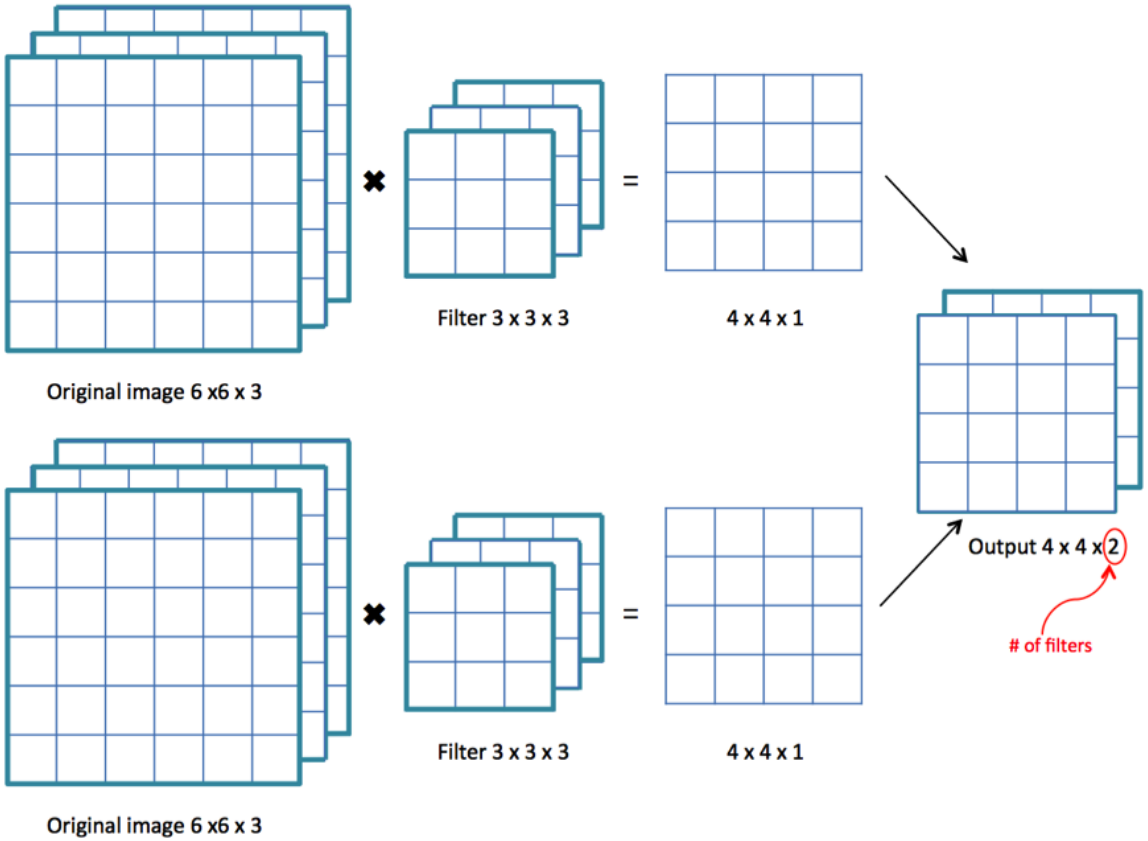


Convolution with horizontal and vertical strides = 1

CNN Basics: Convolution

- Output size according to the size of padding and stride

$$\text{Output size} = \left(\frac{H + 2P - F}{S} + 1\right) \times \left(\frac{W + 2P - F}{S} + 1\right)$$

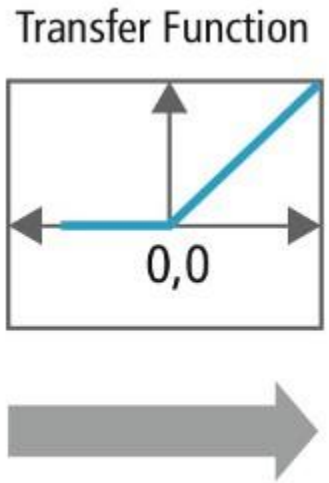


https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

CNN Basics: Activation

- Activation
  - ✓ Non-linear transformation after convolution
  - ✓ Rectified Linear Unit (ReLU) is commonly used

15	20	-10	35
18	-110	25	100
20	-15	25	-10
101	75	18	23

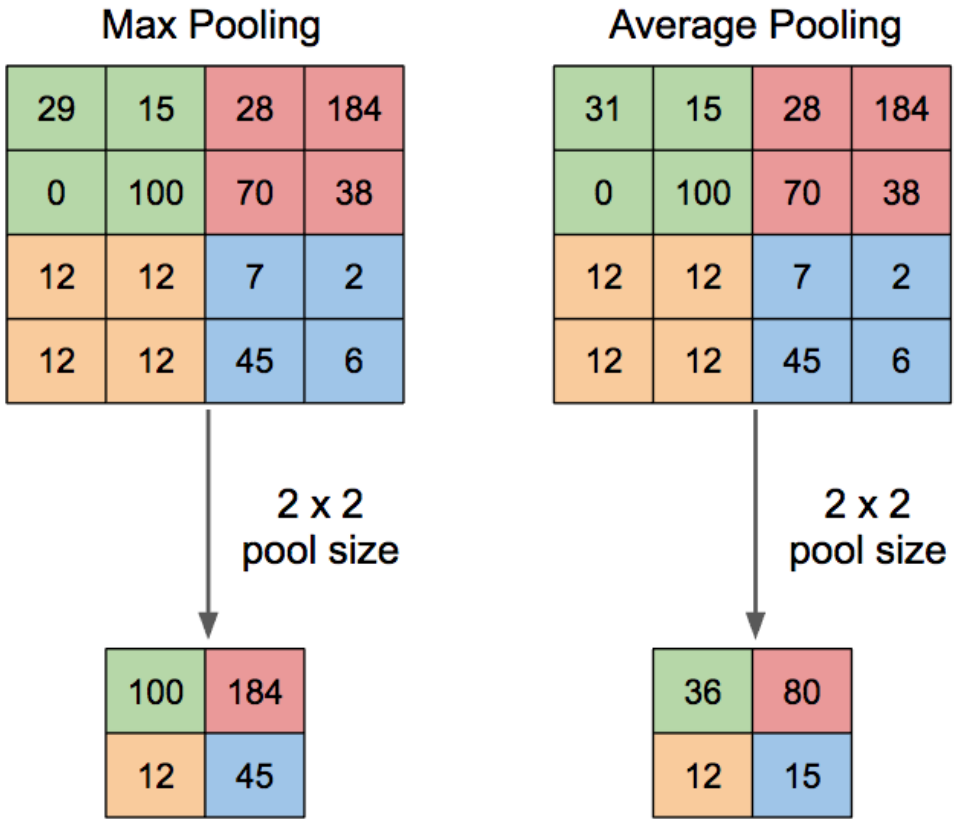


15	20	0	35
18	0	25	100
20	0	25	0
101	75	18	23

https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

CNN Basics: Pooling

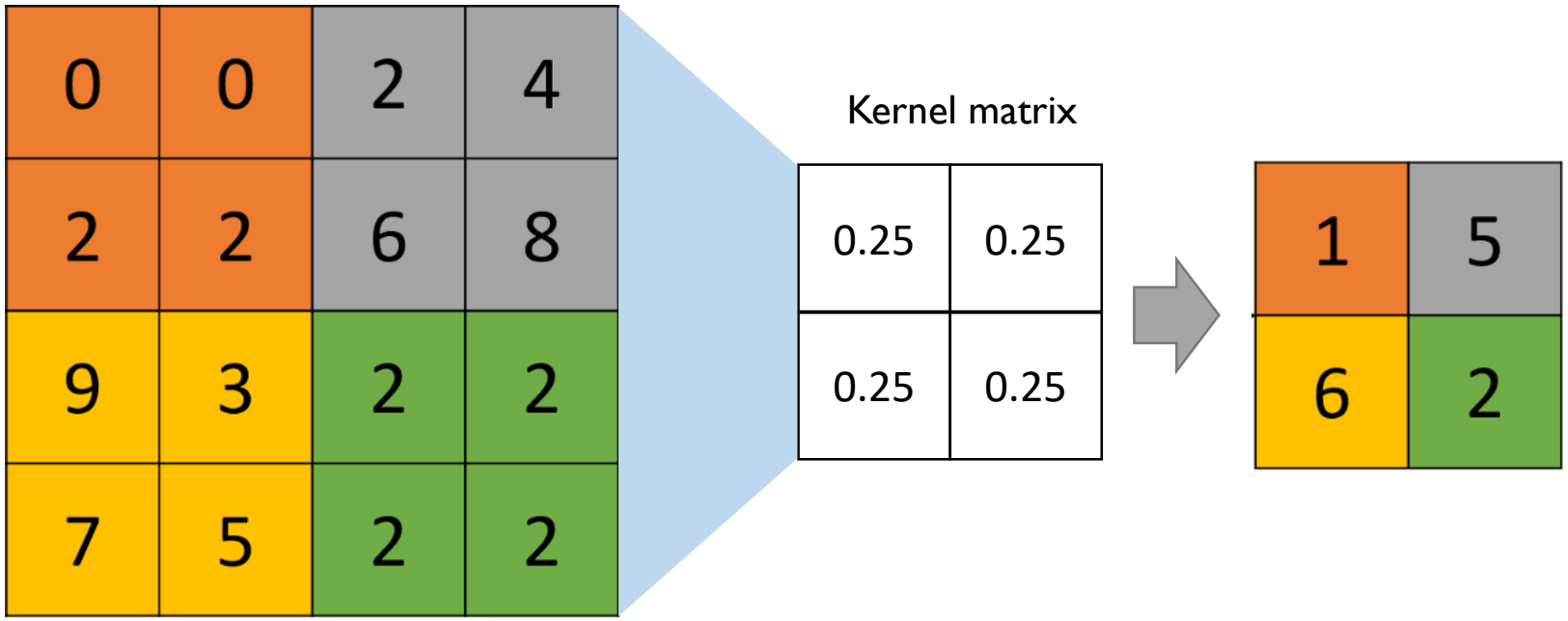
- Pooling
  - ✓ Issue: Need to compress the learned features
  - ✓ Solution: pooling (take a representative value from a certain area)



https://pythonmachinelearning.pro/introduction-to-convolutional-neural-networks-for-vision-tasks/

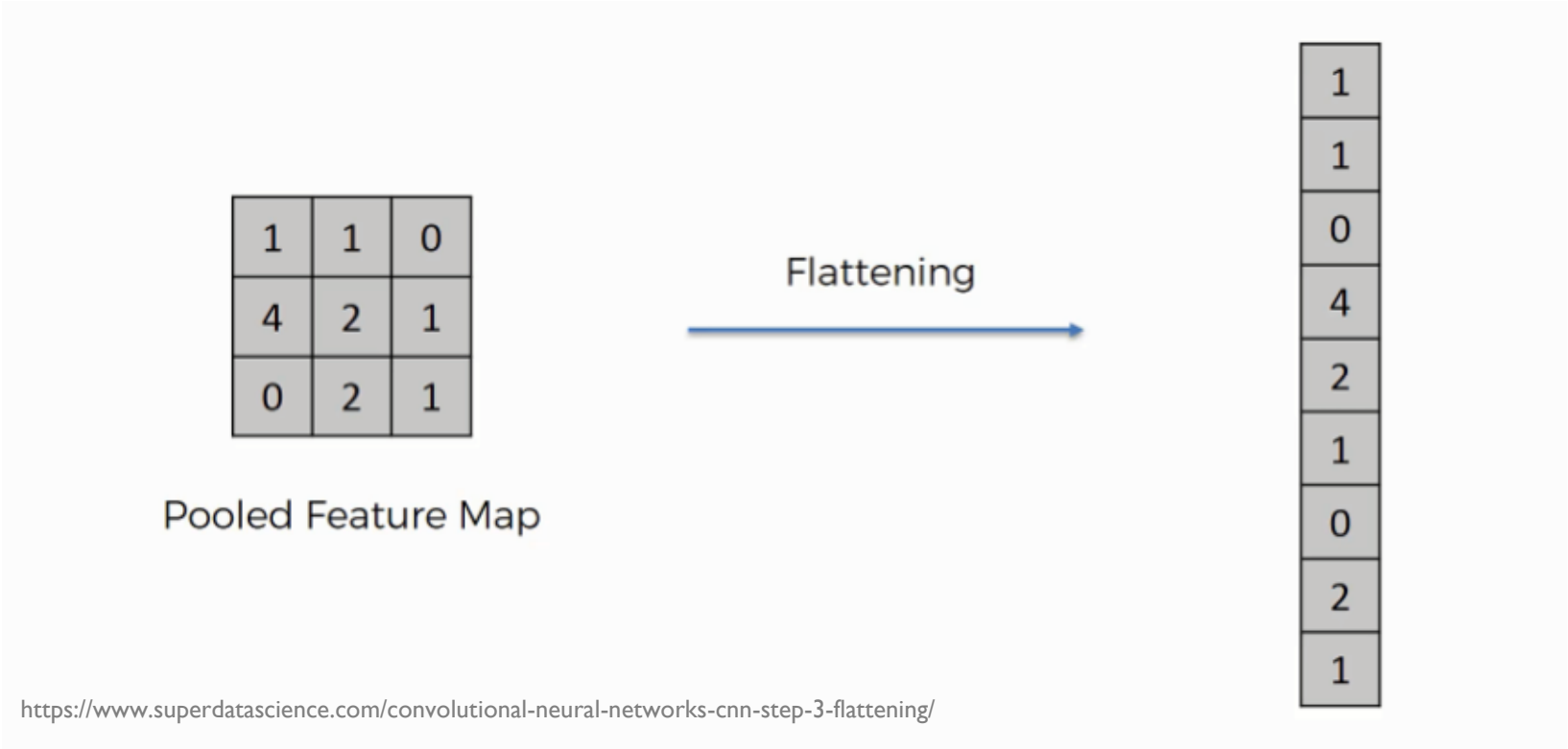
CNN Basics: Strided Convolution

- Strided Convolution
  - ✓ Average pooling is a special case of strided convolution: all weights are 1/n

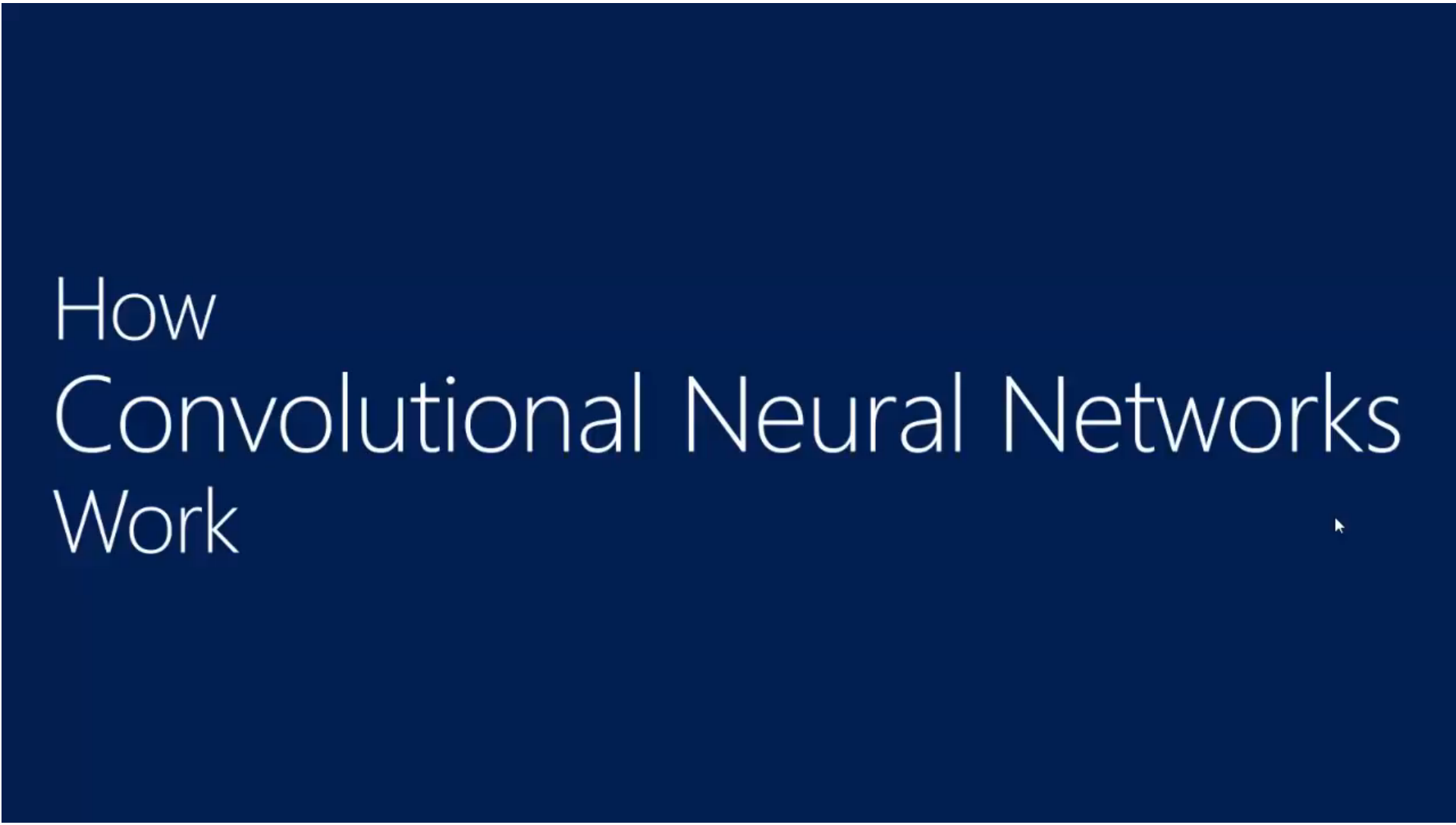


CNN Basics: Strided Convolution

- Flatten
  - ✓ Transform a 2D matrix or 3D tensor to 1D vector



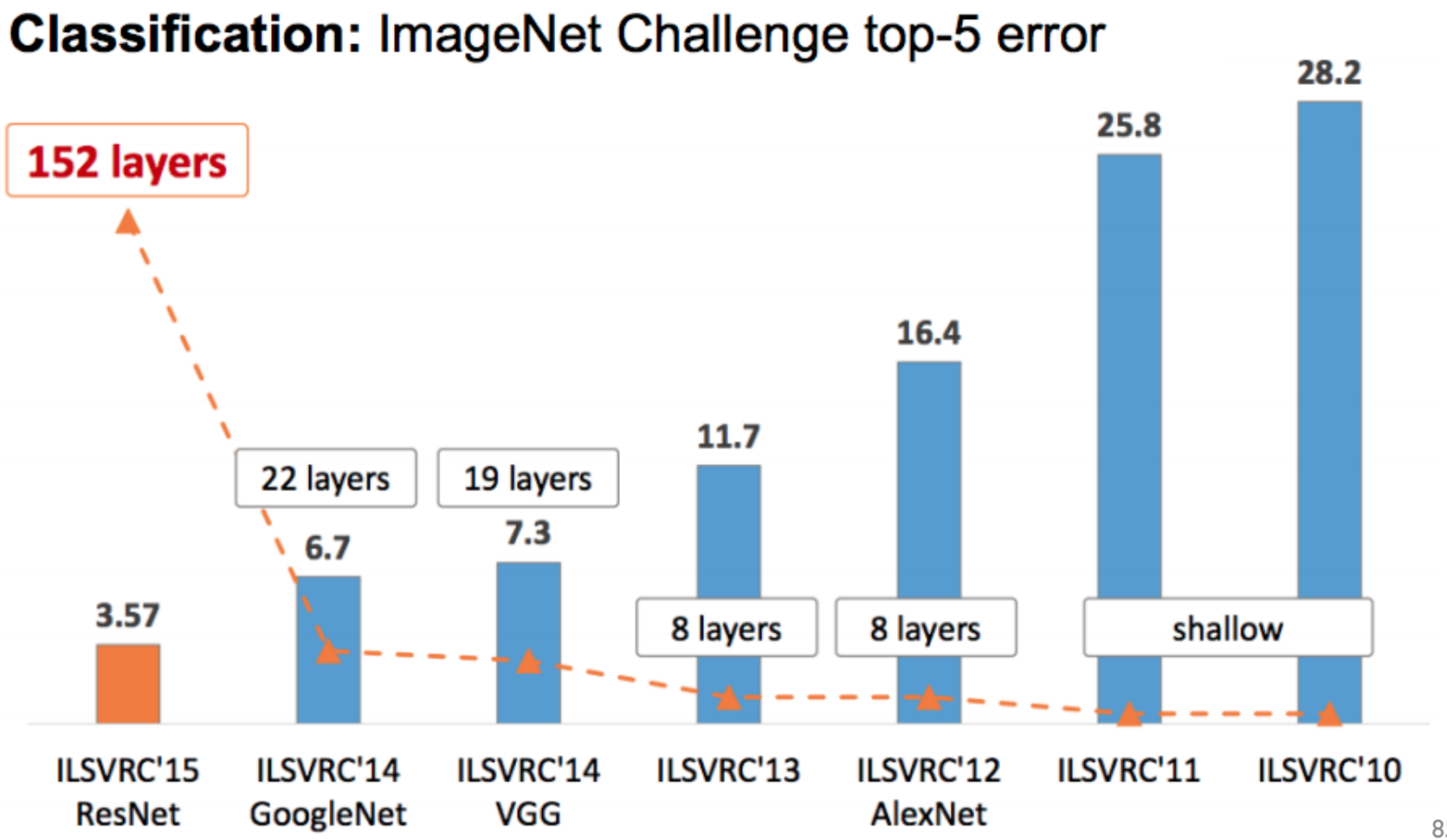
CNN Procedure



https://www.youtube.com/watch?time_continue=797&v=FmpDIaiMIeA

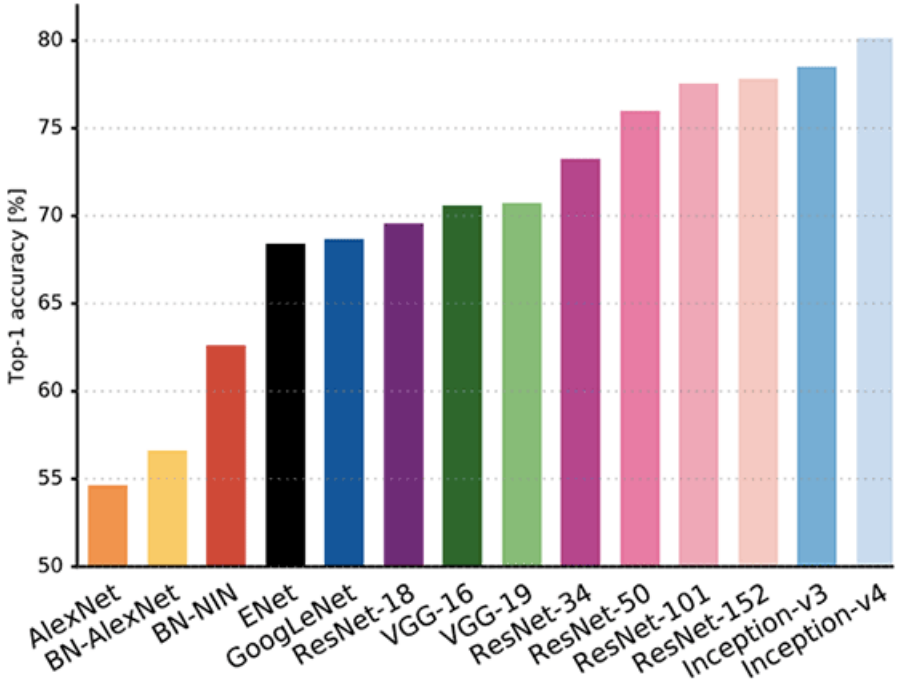
Convolutional Neural Network (CNN)

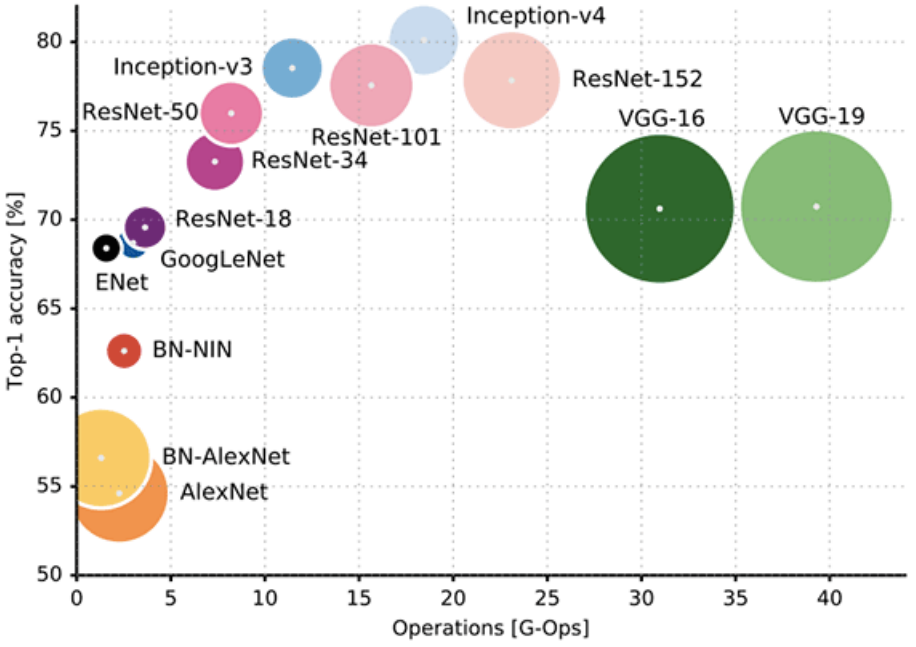
- Power of deep layers in CNN
  - ✓ Tends to improve the performance with regard to an increased layers



Convolutional Neural Network (CNN)

- Power of deep layers in CNN
  - ✓ More efficient (fewer parameters) but more effective models are developed

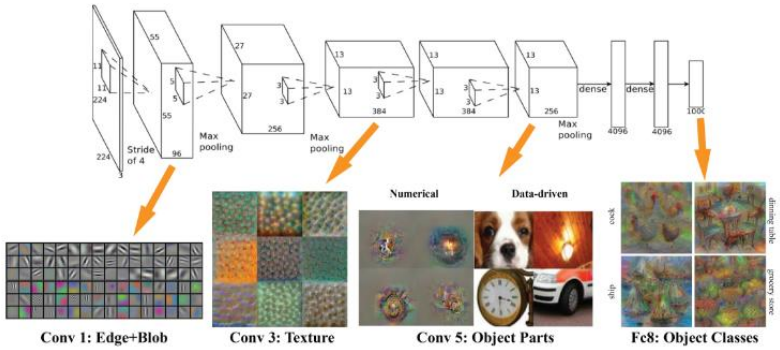




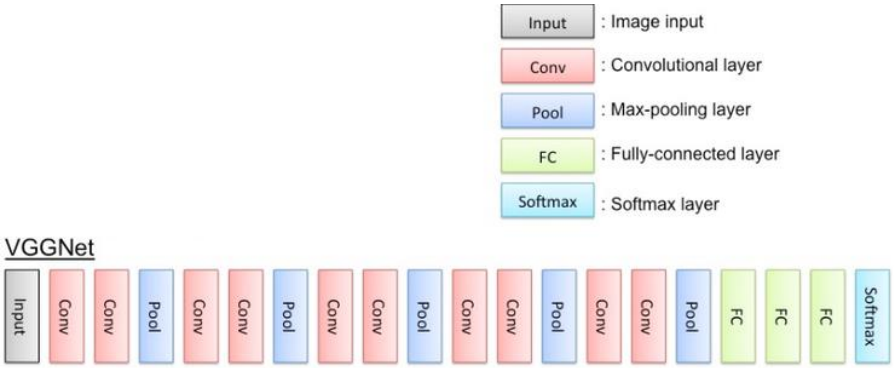
https://www.topbots.com/14-design-patterns-improve-convolutional-neural-network-cnn-architecture/

Convolutional Neural Network (CNN)

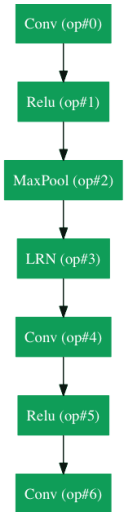
- Famous CNN Architectures



Alexnet

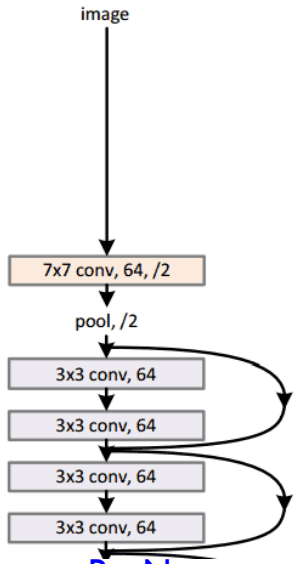


VGGNet



GoogLeNet

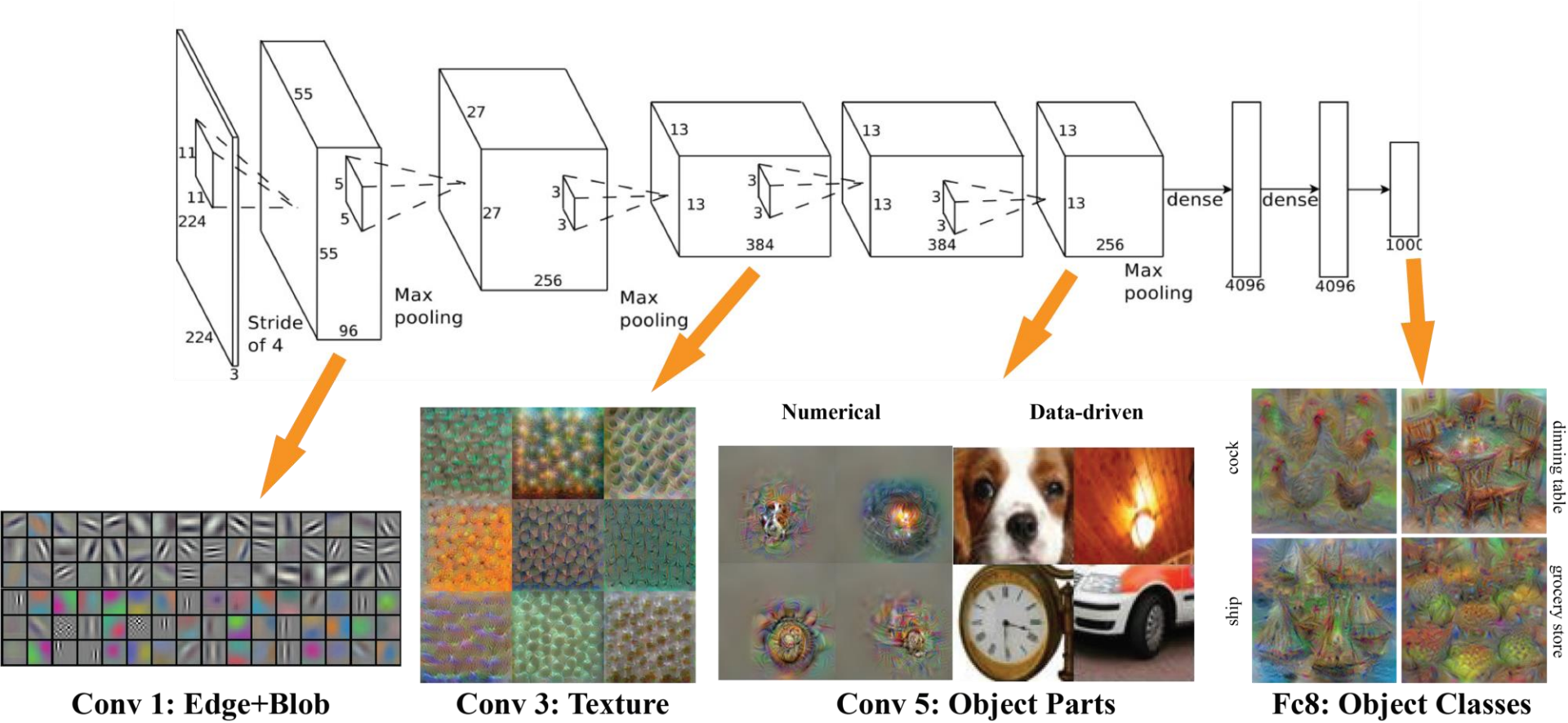
34-layer residual



ResNet

CNN Architecture 1: AlexNet

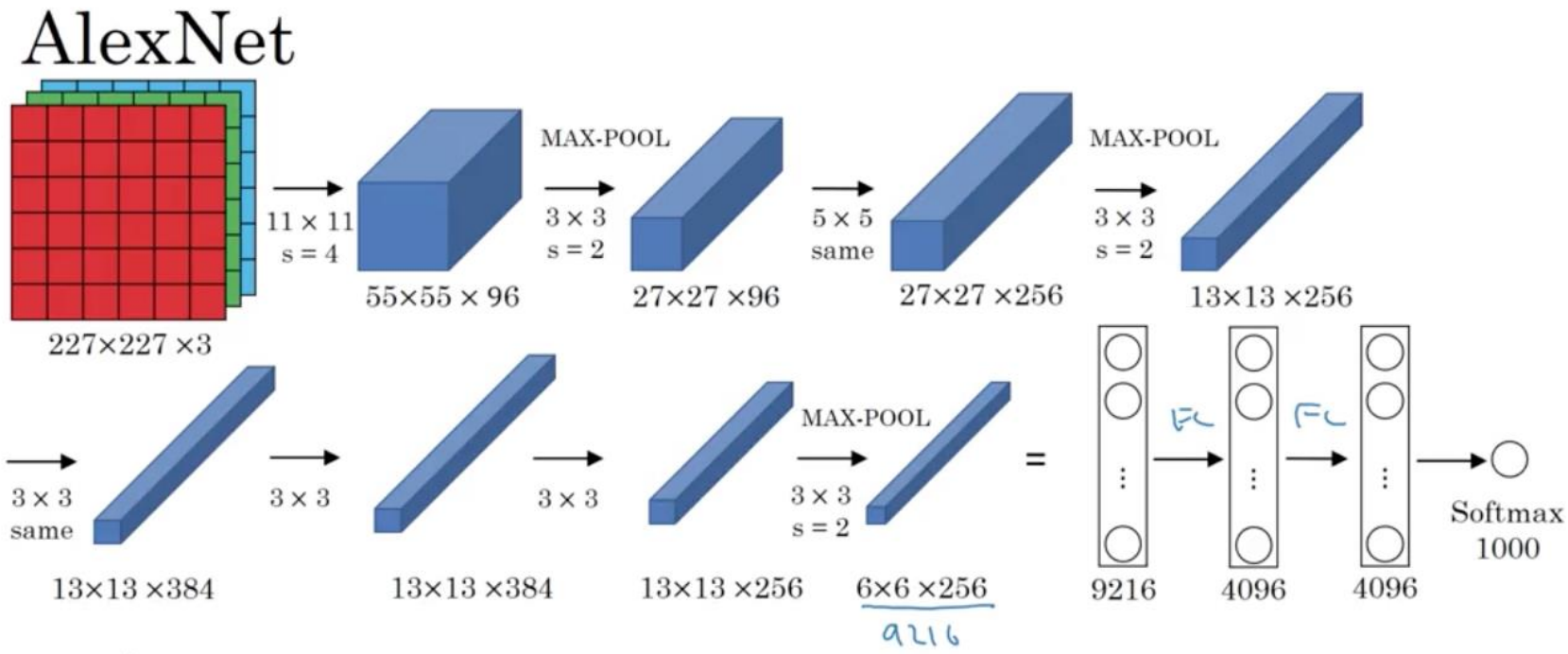
- AlexNet



Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

CNN Architecture 1: AlexNet

- AlexNet

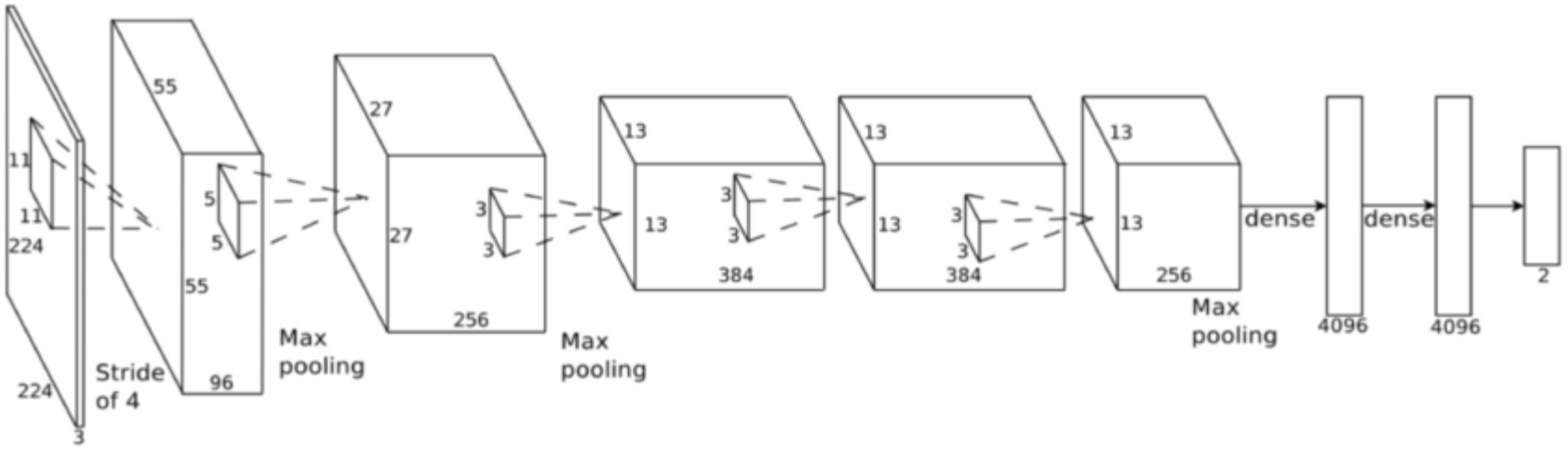


[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

Andrew Ng

CNN Architecture 1: AlexNet

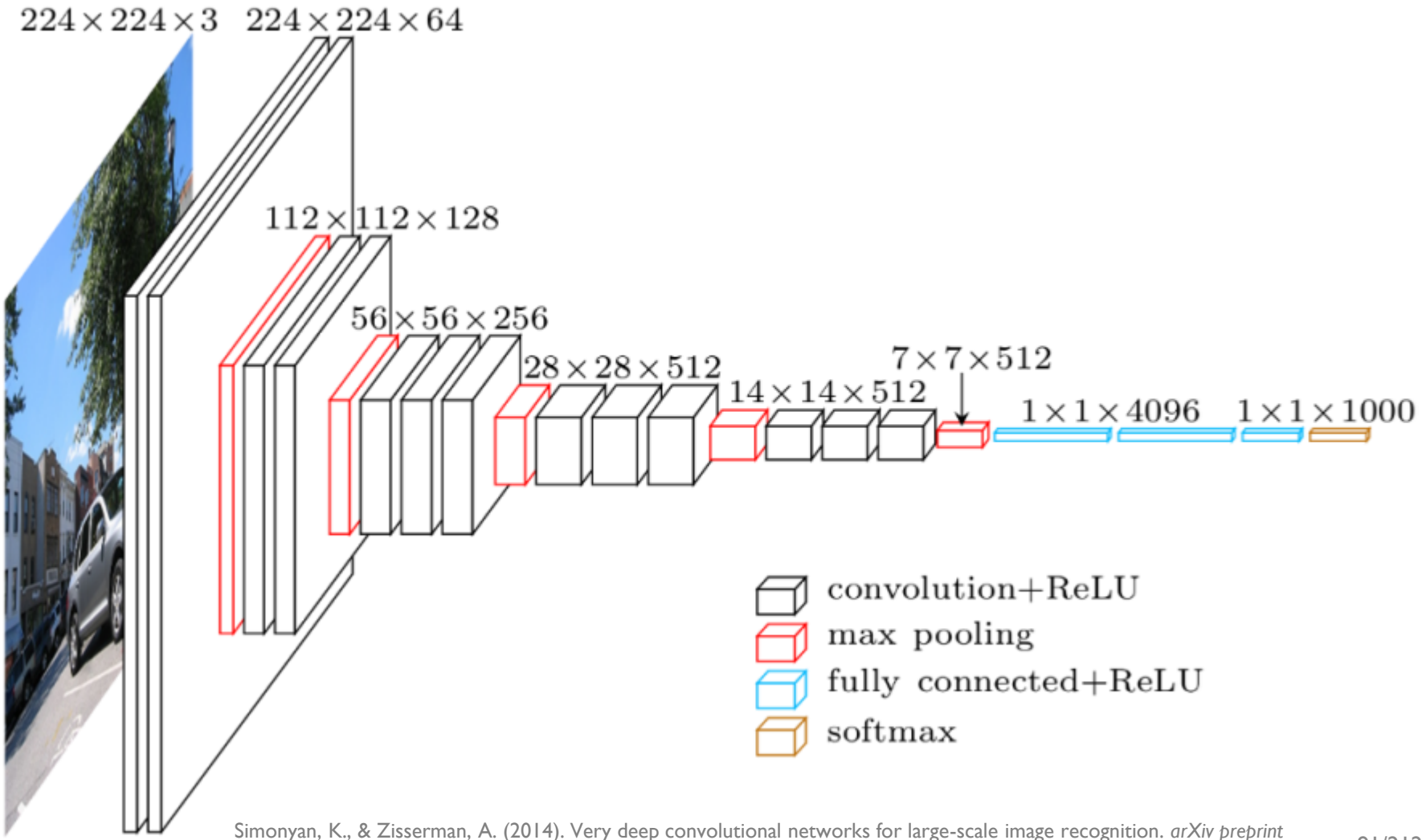
- AlexNet



- ✓ Input size: 224 by 224
- ✓ Large filter sizes with wide strides in the lower layers
- ✓ Small filter sizes with narrow strides in the upper layers
- ✓ 2 fully connected layers
- ✓ No. of parameters: 60 millions

CNN Architecture 2:VGGNet

- VGGNet



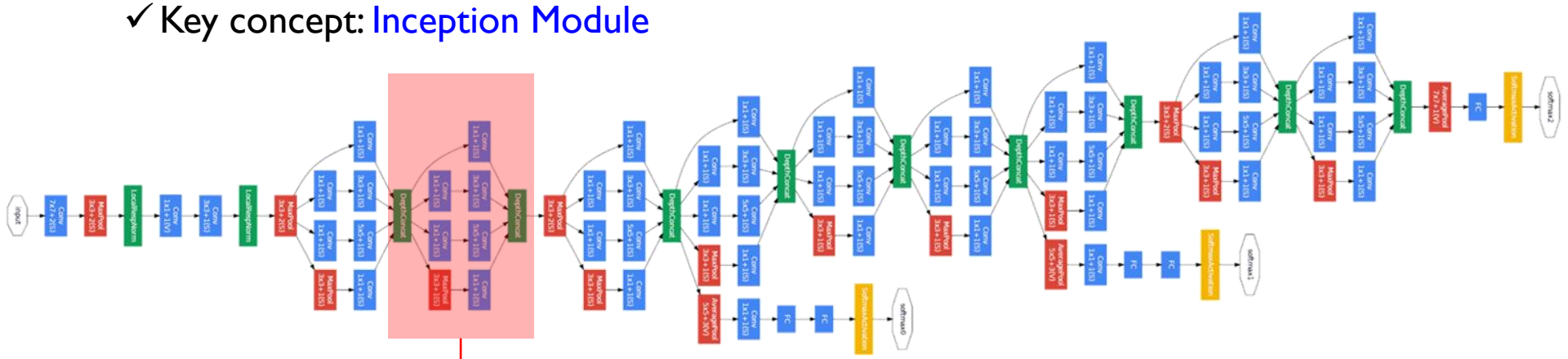
Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

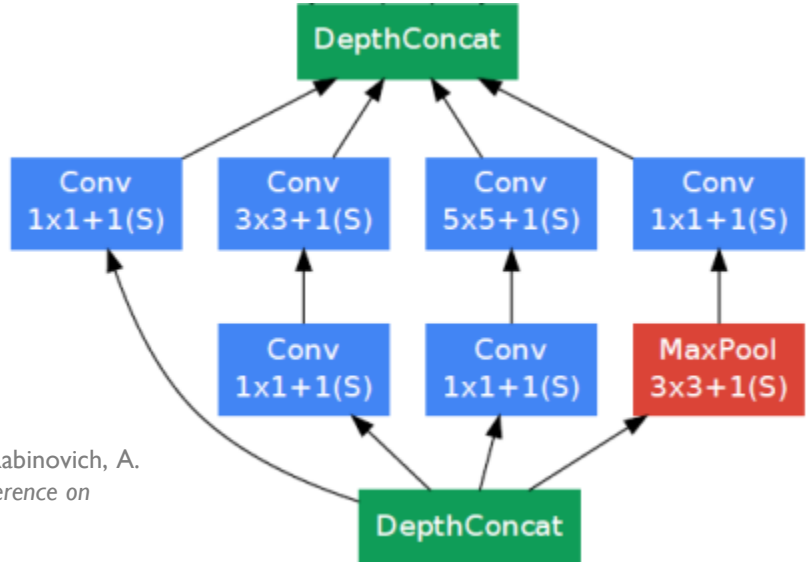
CNN Architecture 2:VGGNet

- VGGNet
  - ✓ Simple but Deep!
  - ✓ Basic operation: 3 by 3 convolution with stride 1
  - ✓ 2 by 2 max pooling is used
  - ✓ No. of parameters: 138 million

CNN Architecture 3: GoogLeNet

- GoogLeNet
  - ✓ Key concept: Inception Module



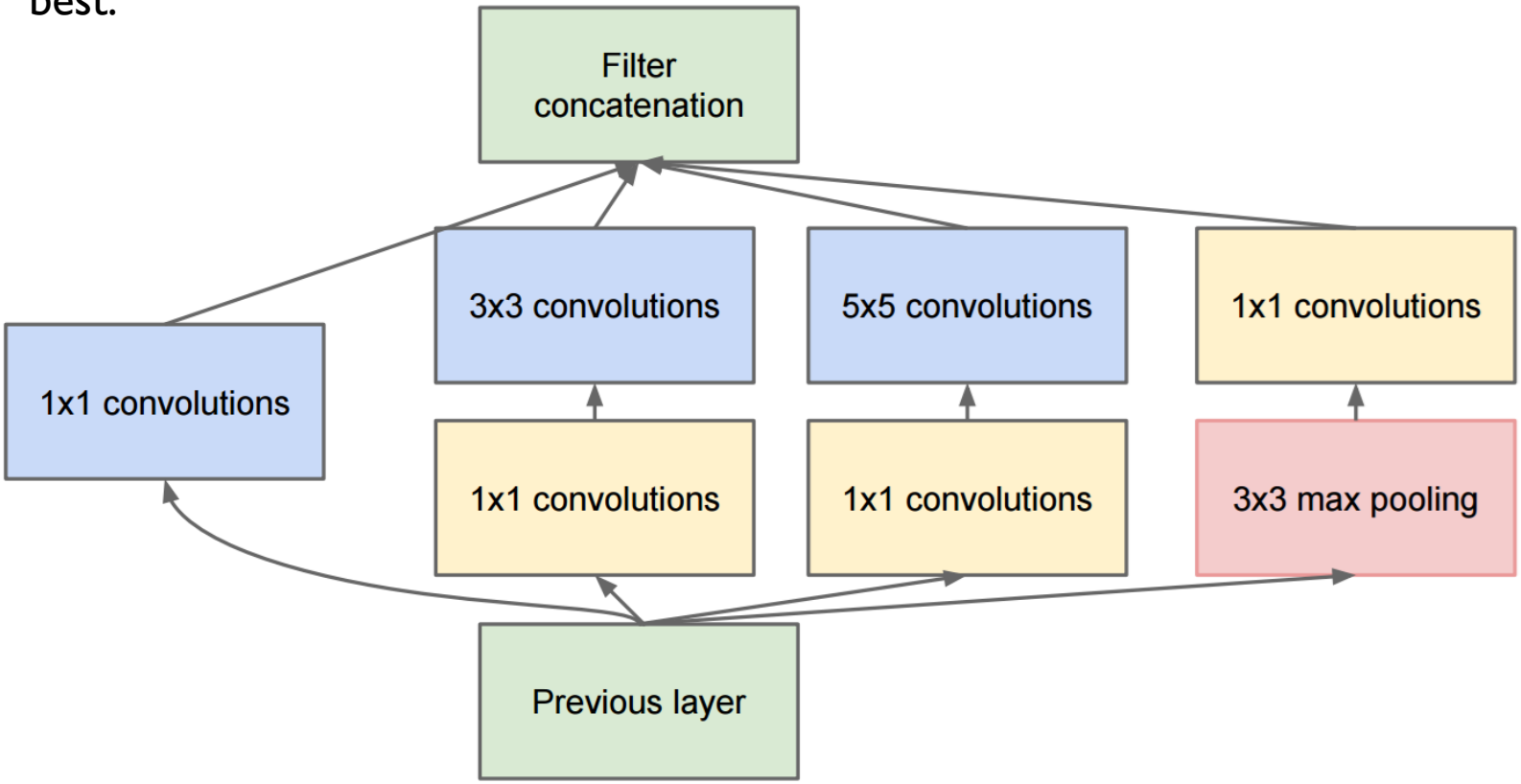


Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7

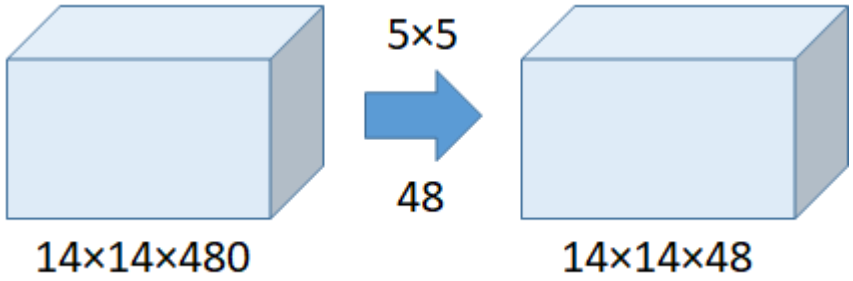
CNN Architecture 3: GoogLeNet

- GoogLeNet
  - ✓ Why should we use only one filter size?
  - ✓ Use 1 by 1, 3 by 3, 5 by 5, 3 by 3 max pooling since we do not know which one is the best.



CNN Architecture 3: GoogLeNet

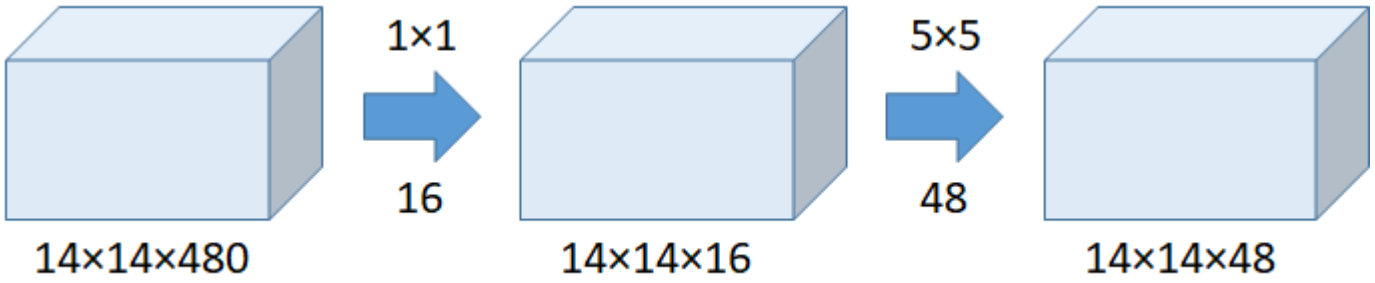
- GoogLeNet
  - ✓ The role of 1 by 1 convolution: decrease the depth of feature map → less parameters required
  - ✓ Ex) 5 by 5 convolution without 1 by 1 convolution



  - No. of parameters: $(14 \times 14 \times 48) \times (5 \times 5 \times 480) = 112.9\text{M}$

CNN Architecture 3: GoogLeNet

- GoogLeNet
  - ✓ The role of 1 by 1 convolution: decrease the depth of feature map → less parameters required
  - ✓ Ex) 5 by 5 convolution after 1 by 1 convolution



  - ✓ No. of parameters for 1 by 1 Conv.: $(14 \times 14 \times 16) \times (1 \times 1 \times 480) = 1.5\text{M}$
  - ✓ No. of parameters for 5 by 5 Conv.: $(14 \times 14 \times 48) \times (5 \times 5 \times 16) = 3.8\text{M}$
  - ✓ Total parameters: 1.5M + 3.8M = 5.3M << 112.9M

CNN Architecture 3: GoogLeNet

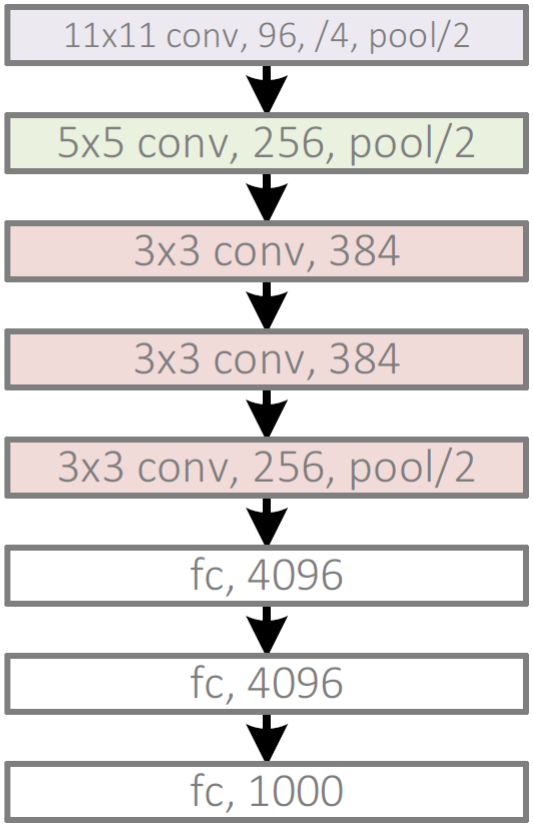
- GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

CNN Architecture 4: ResNet

- Deeper, deeper, deeper!!

AlexNet, 8 layers
(ILSVRC 2012)



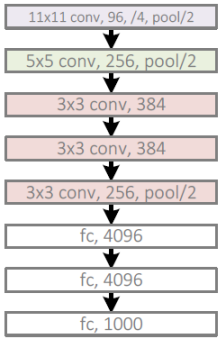
He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

https://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

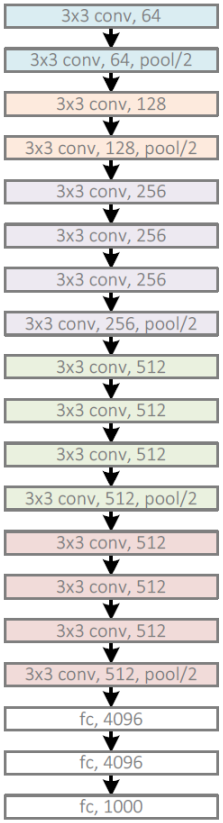
CNN Architecture 4: ResNet

- Deeper, deeper, deeper!!

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

https://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

CNN Architecture 4: ResNet

- Deeper, deeper, deeper!!

AlexNet, 8 layers
(ILSVRC 2012)

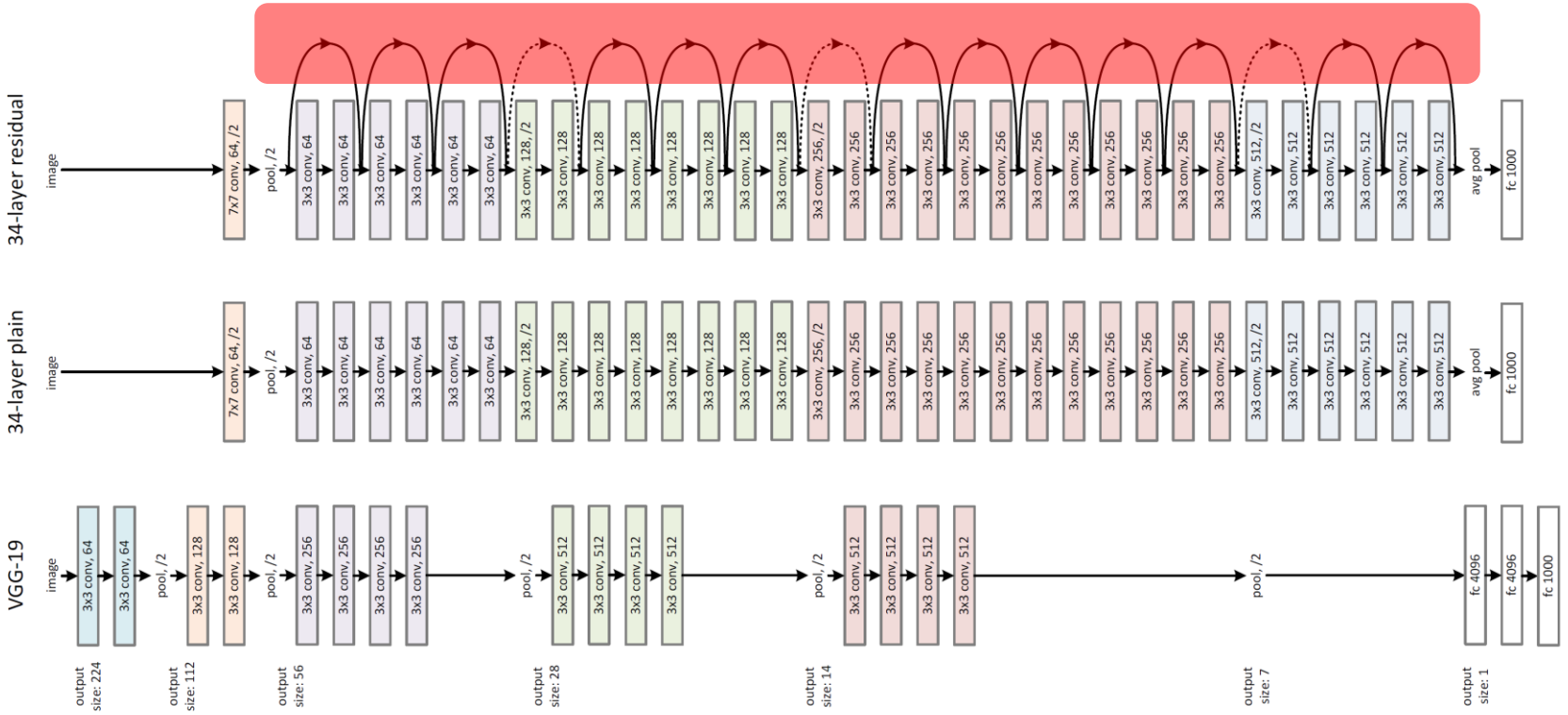
VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

https://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

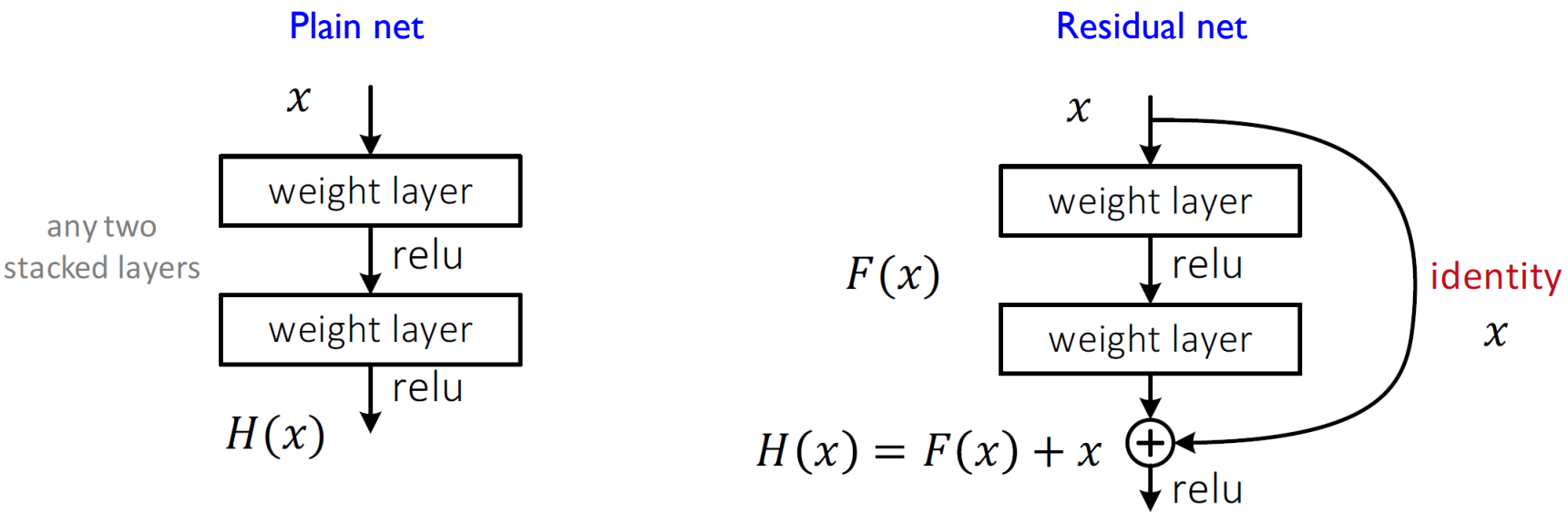
CNN Architecture 4: ResNet

- Key concept: Residual Learning



CNN Architecture 4: ResNet

- Residual learning: skip connection

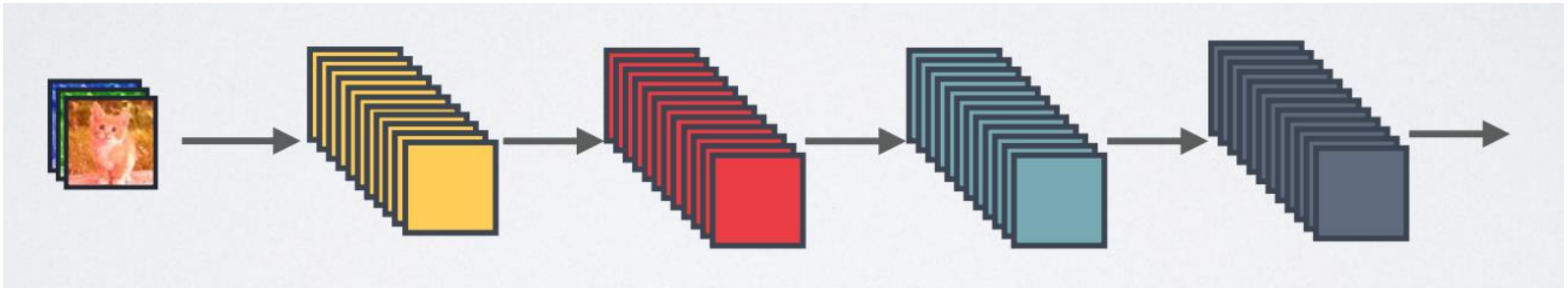


✓ Connect the input x to the feature map by skip connection

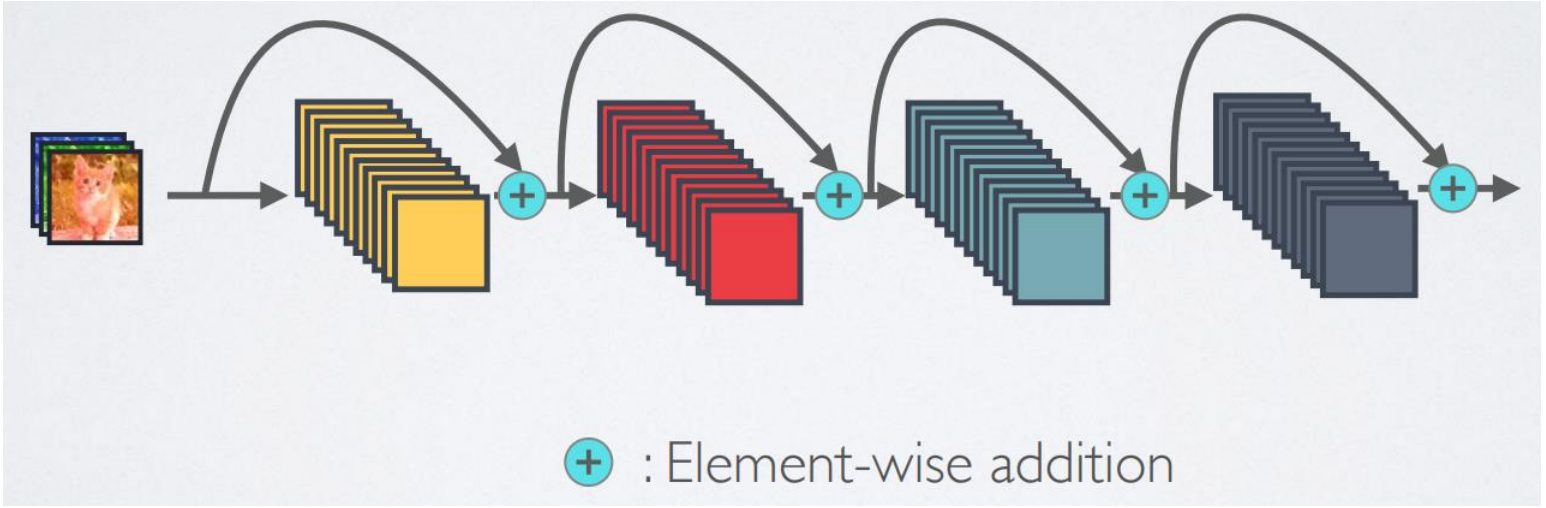
- Better gradient flow (we can make a really deep network)
- Remove fully connected layer (less parameters are required)

CNN Architecture 5: DenseNet

- Standard Connectivity



- Resnet Connectivity

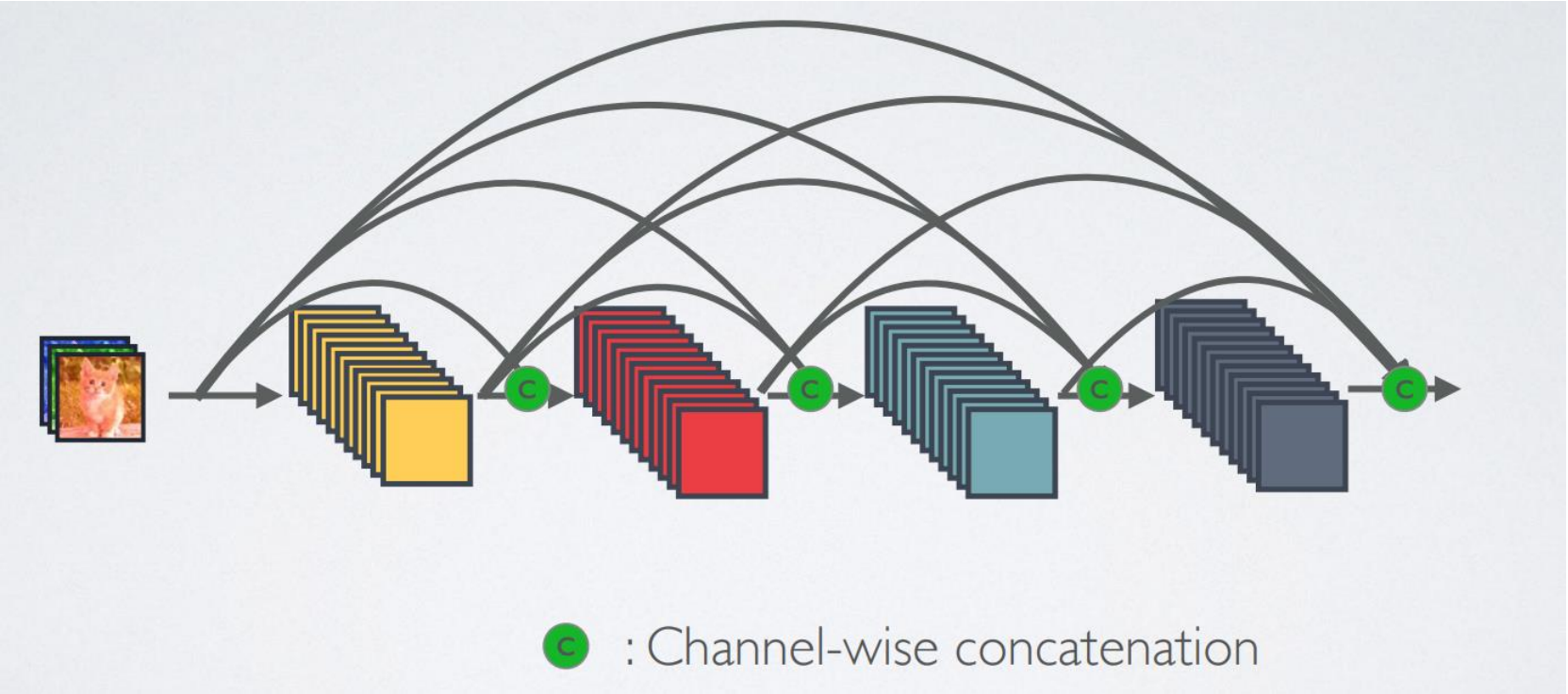


Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).

https://pdfs.semanticscholar.org/c3d9/26a85d85a83126f405ad40ff453611148c15.pdf

CNN Architecture 5: DenseNet

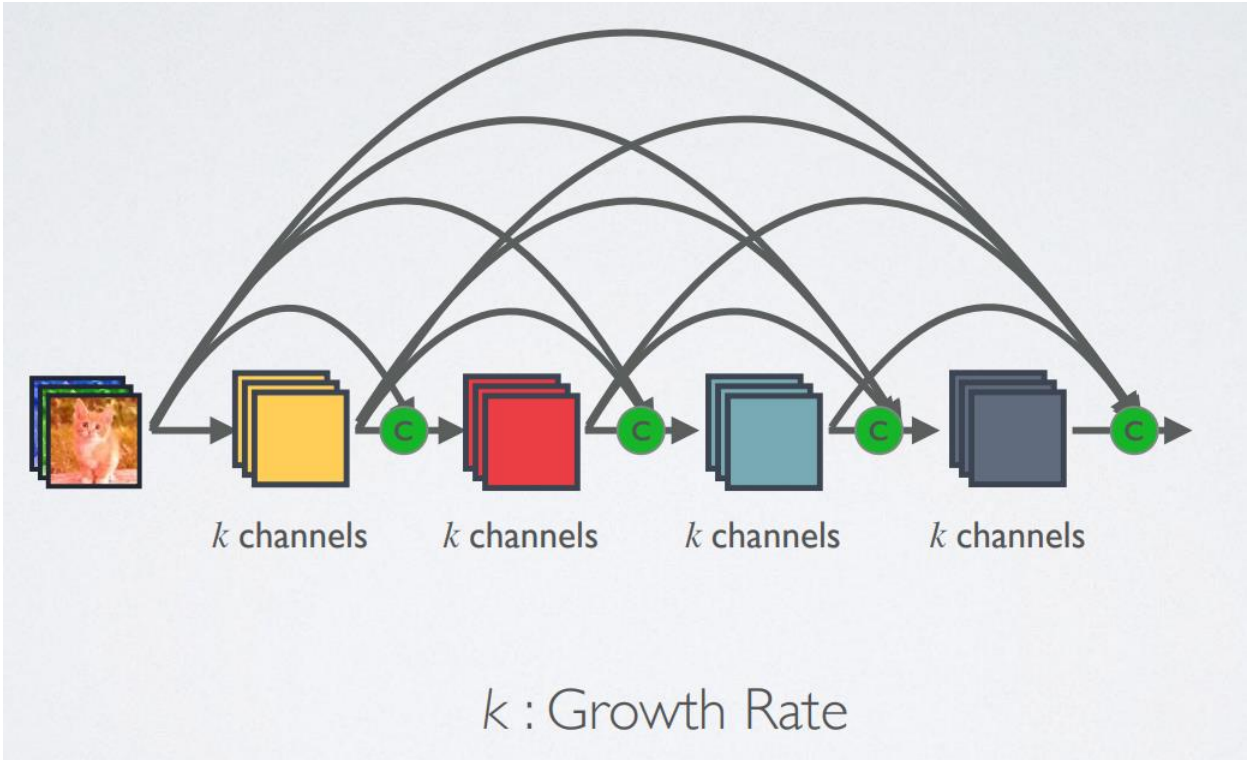
- Dense Connectivity



https://pdfs.semanticscholar.org/c3d9/26a85d85a83126f405ad40ff45361114 8c15.pdf

CNN Architecture 5: DenseNet

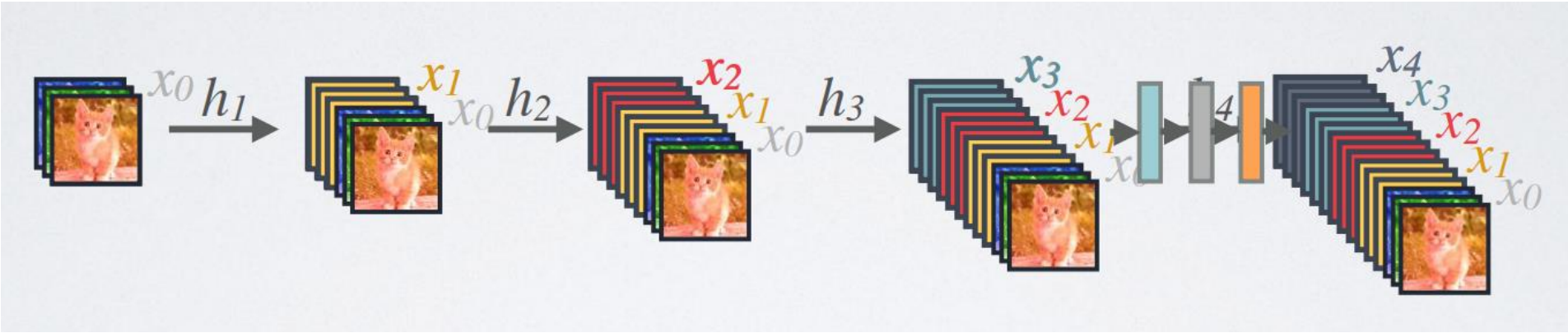
- Dense & Slim



https://pdfs.semanticscholar.org/c3d9/26a85d85a83126f405ad40ff45361114 8c15.pdf

CNN Architecture 5: DenseNet

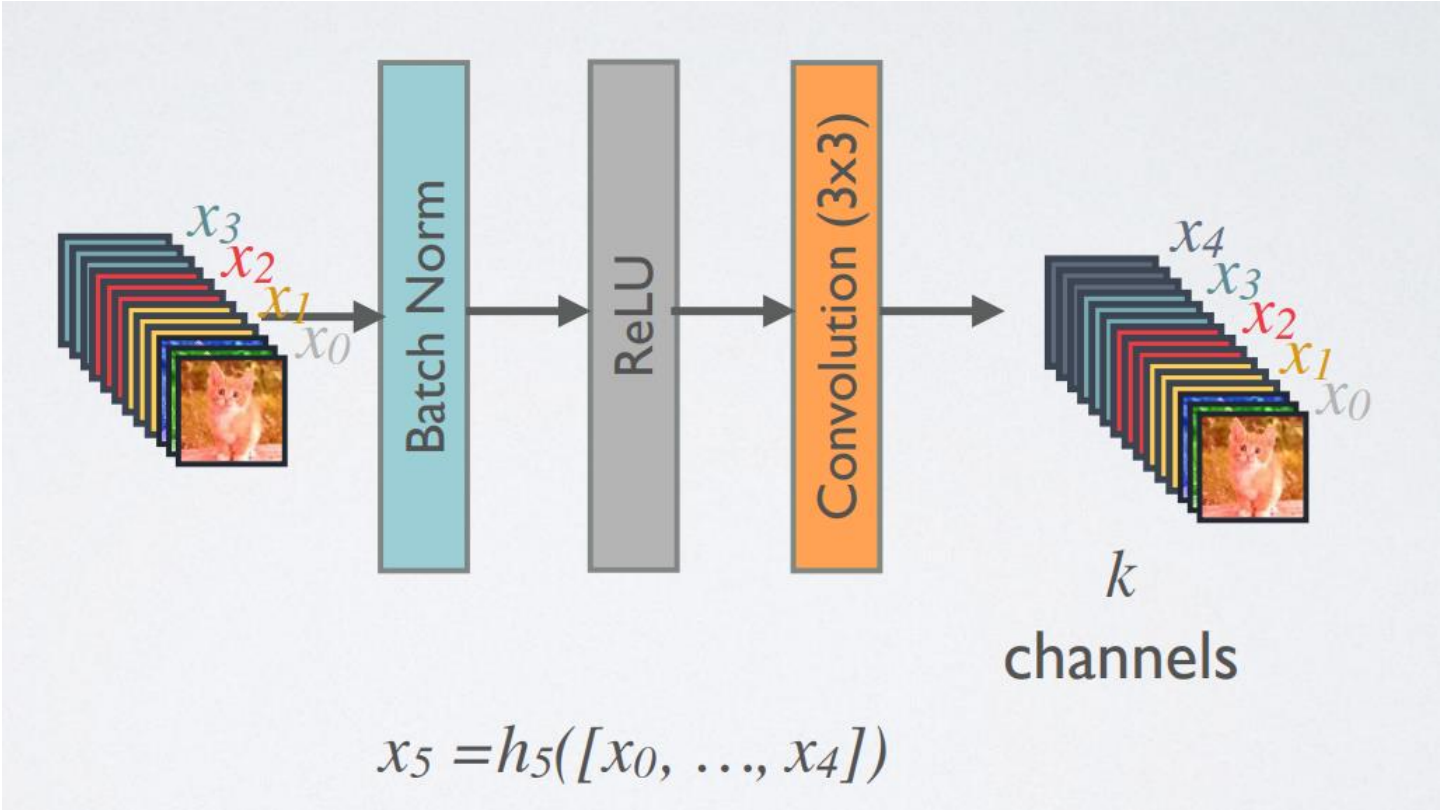
- Forward Propagation



https://pdfs.semanticscholar.org/c3d9/26a85d85a83126f405ad40ff453611148c15.pdf

CNN Architecture 5: DenseNet

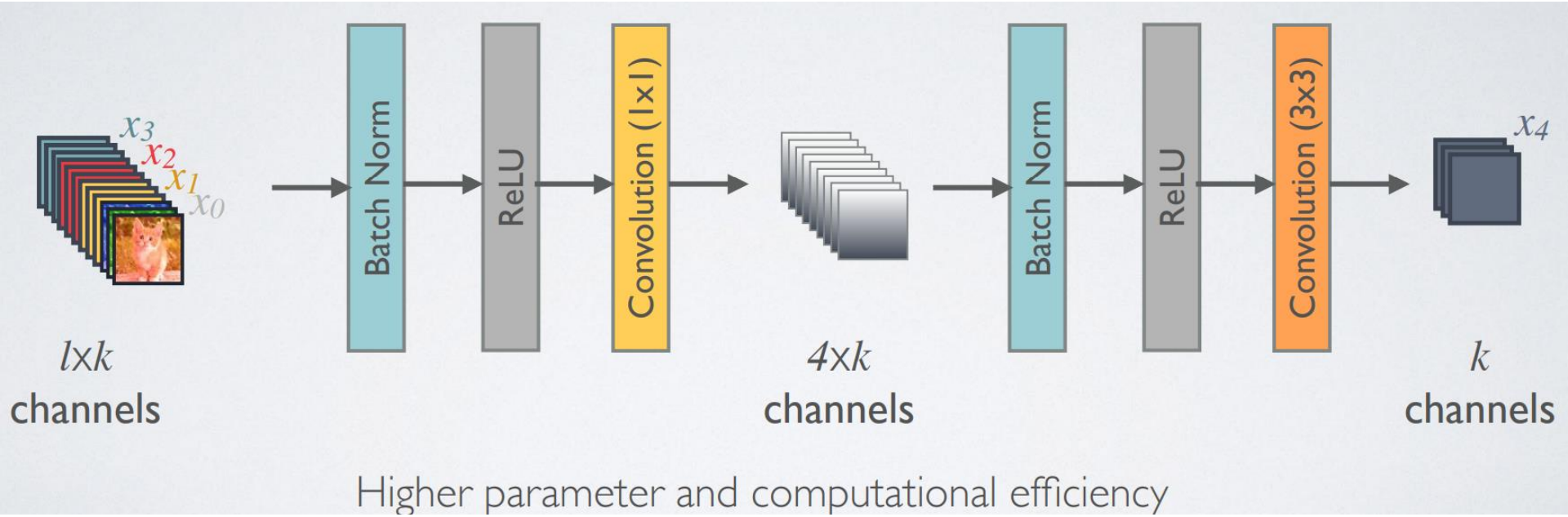
- Composite layer in Densenet



https://pdfs.semanticscholar.org/c3d9/26a85d85a83126f405ad40ff453611148c15.pdf

CNN Architecture 5: DenseNet

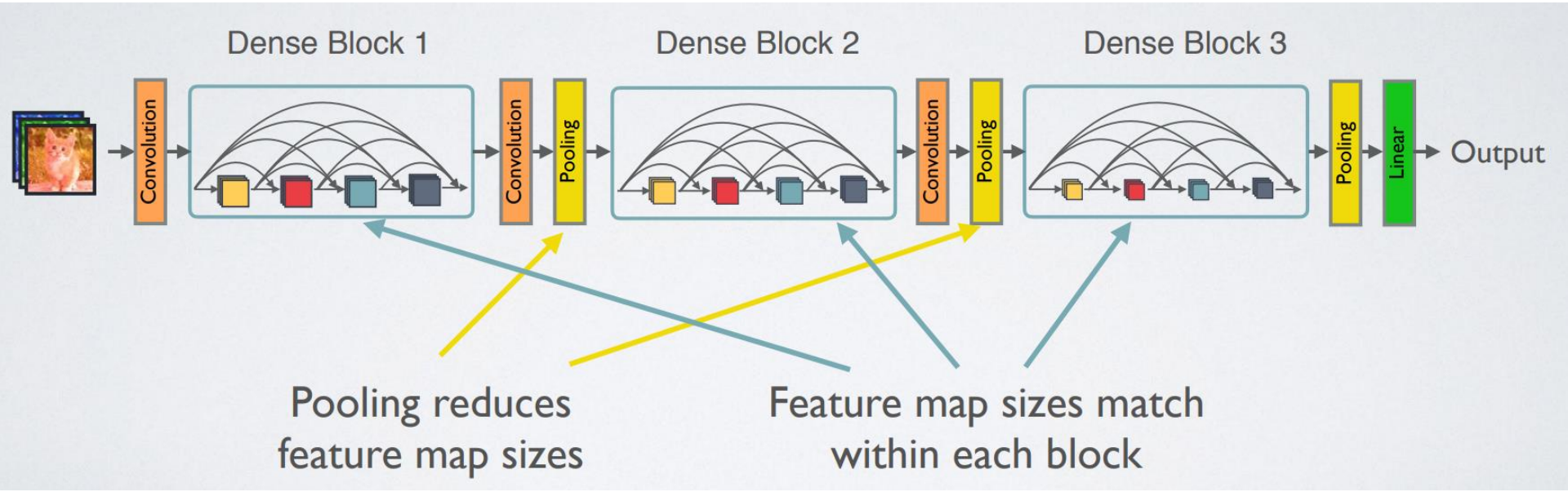
- Composite layer in Densenet with bottleneck layer



https://pdfs.semanticscholar.org/c3d9/26a85d85a83126f405ad40ff45361114 8c15.pdf

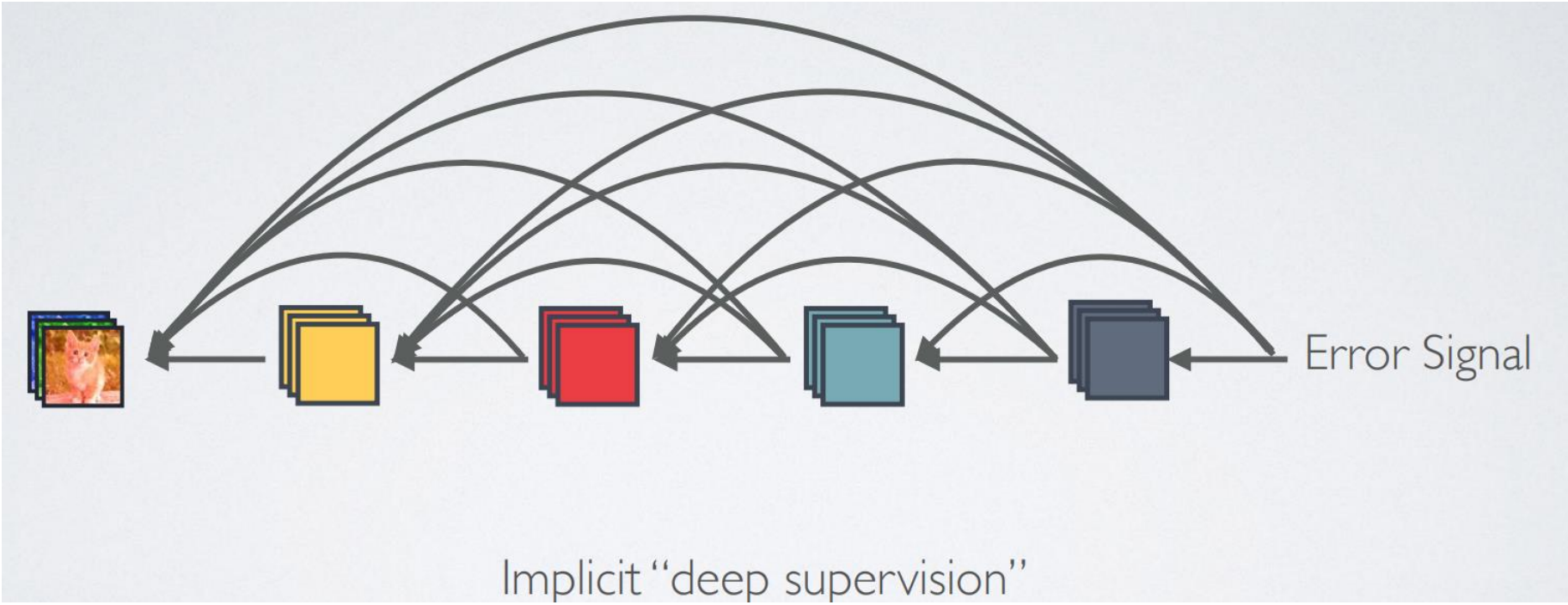
CNN Architecture 5: DenseNet

- Densenet



CNN Architecture 5: DenseNet

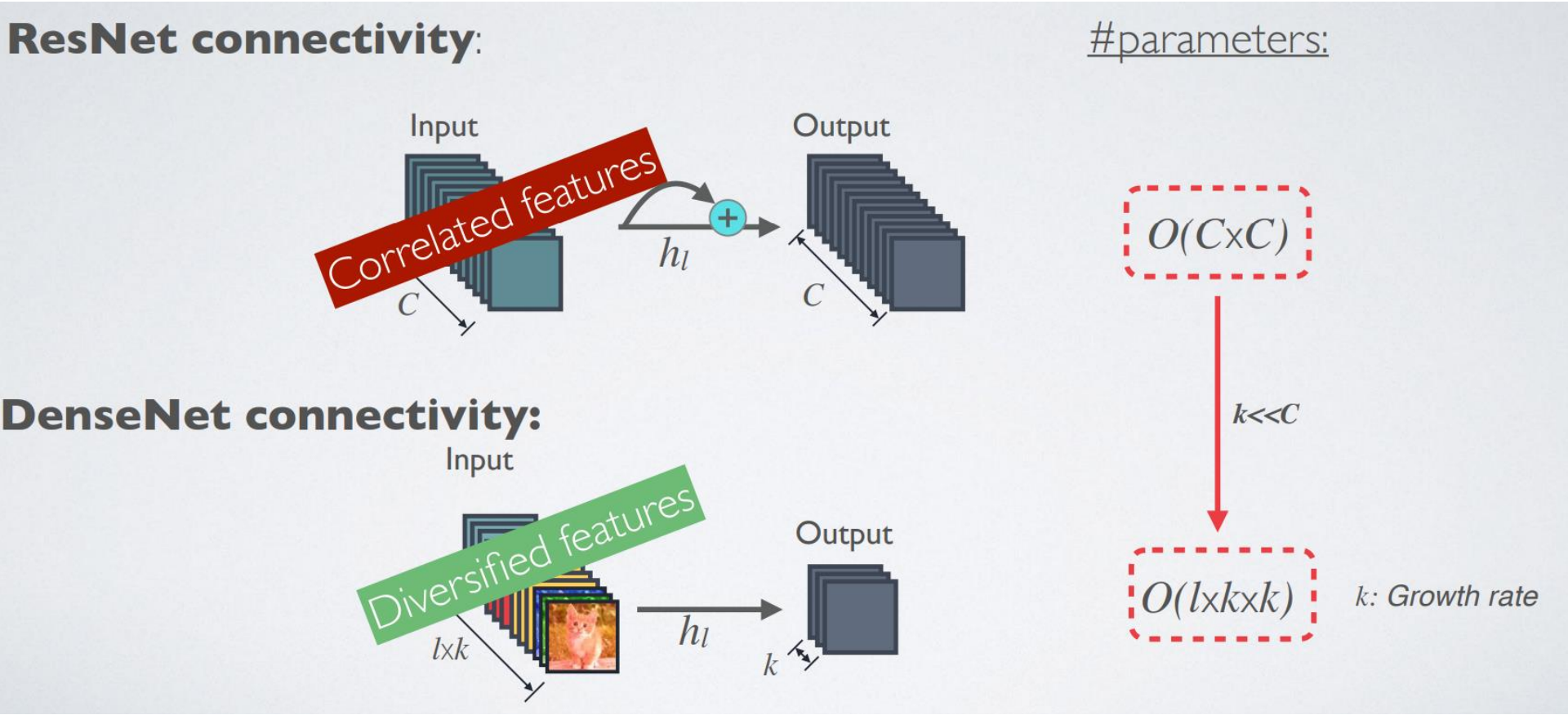
- Advantage of DenseNet 1: Smooth gradient flow



https://pdfs.semanticscholar.org/c3d9/26a85d85a83126f405ad40ff4536111148c15.pdf

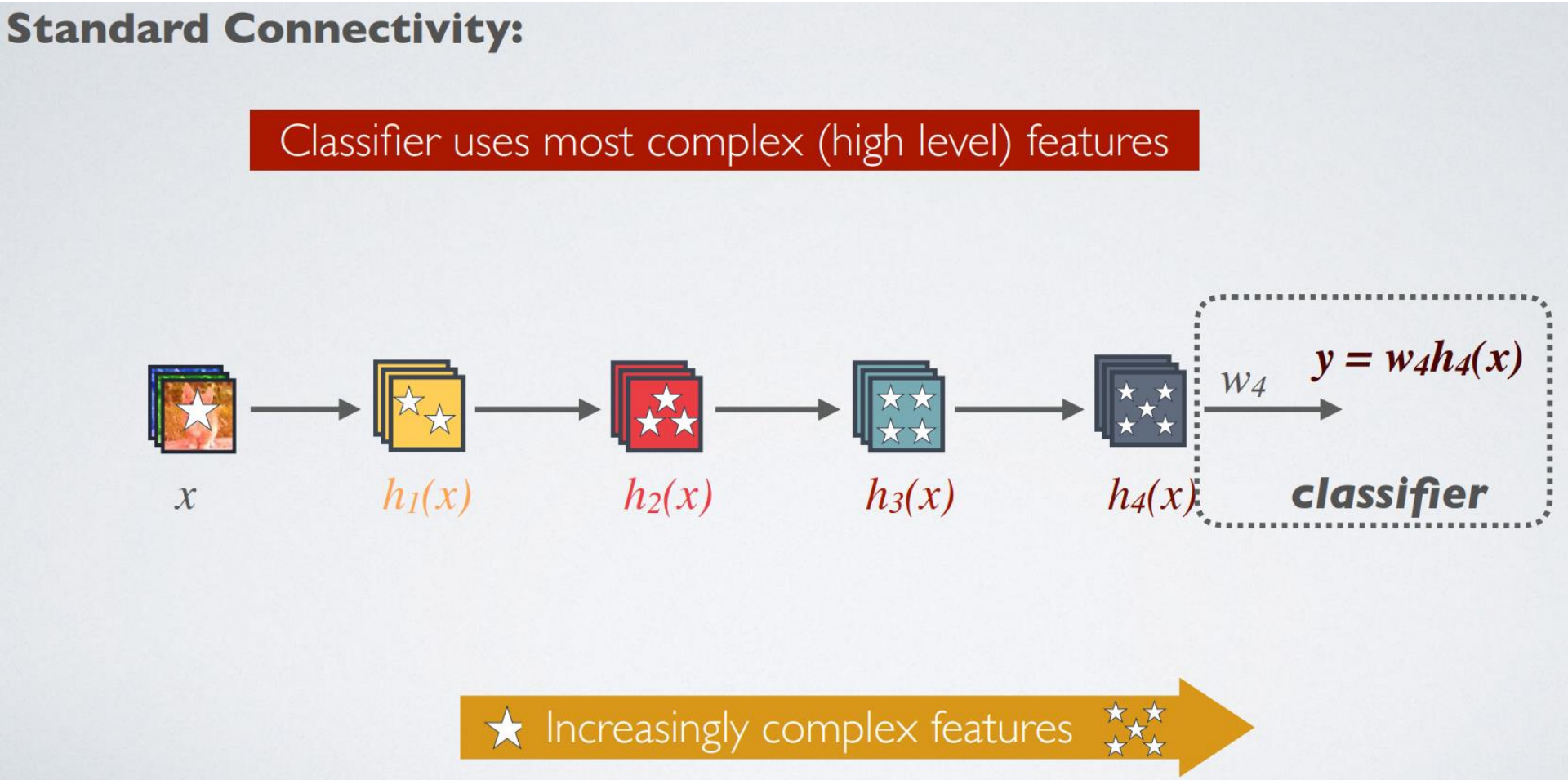
CNN Architecture 5: DenseNet

- Advantage of DenseNet 2: Fewer parameters



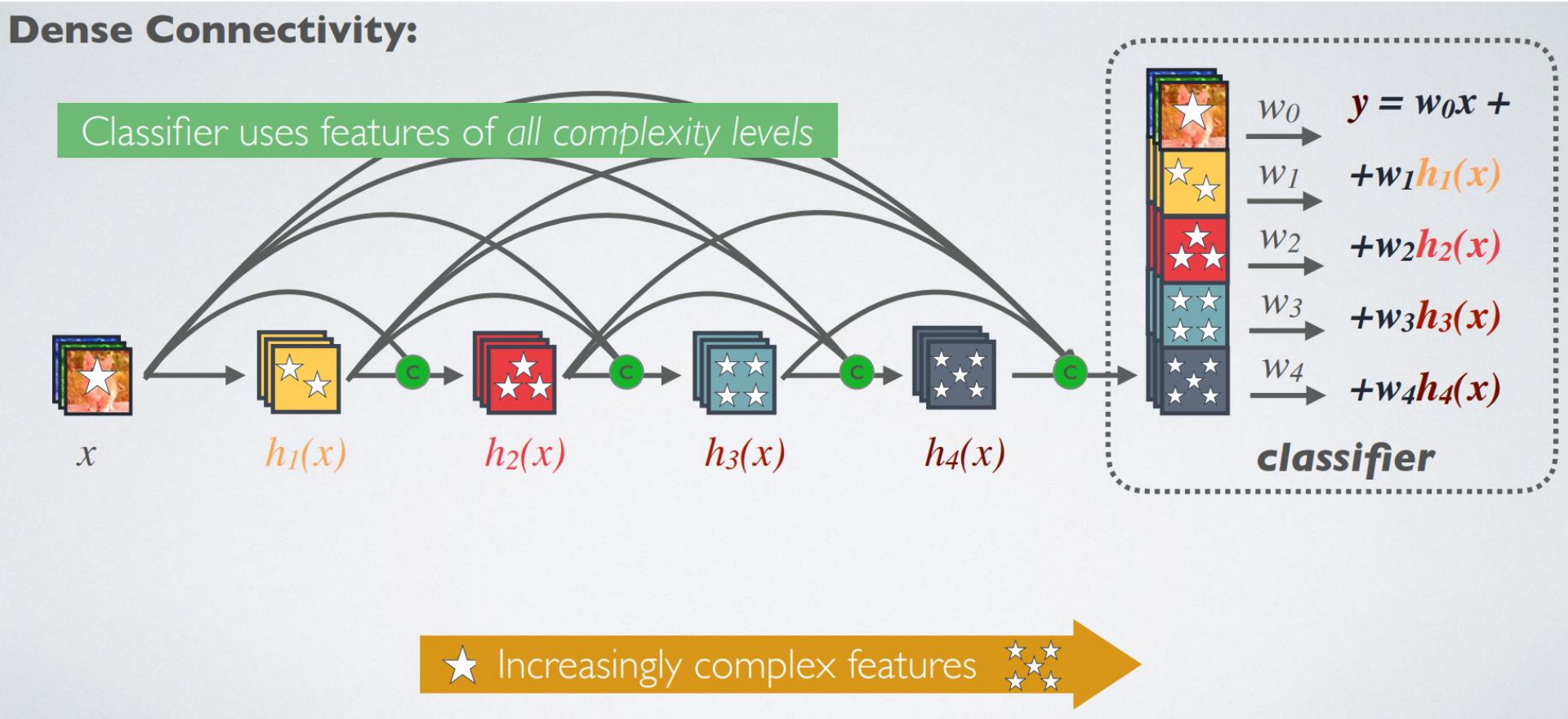
CNN Architecture 5: DenseNet

- Advantage of DenseNet 3: Feature-level diversity can be obtained



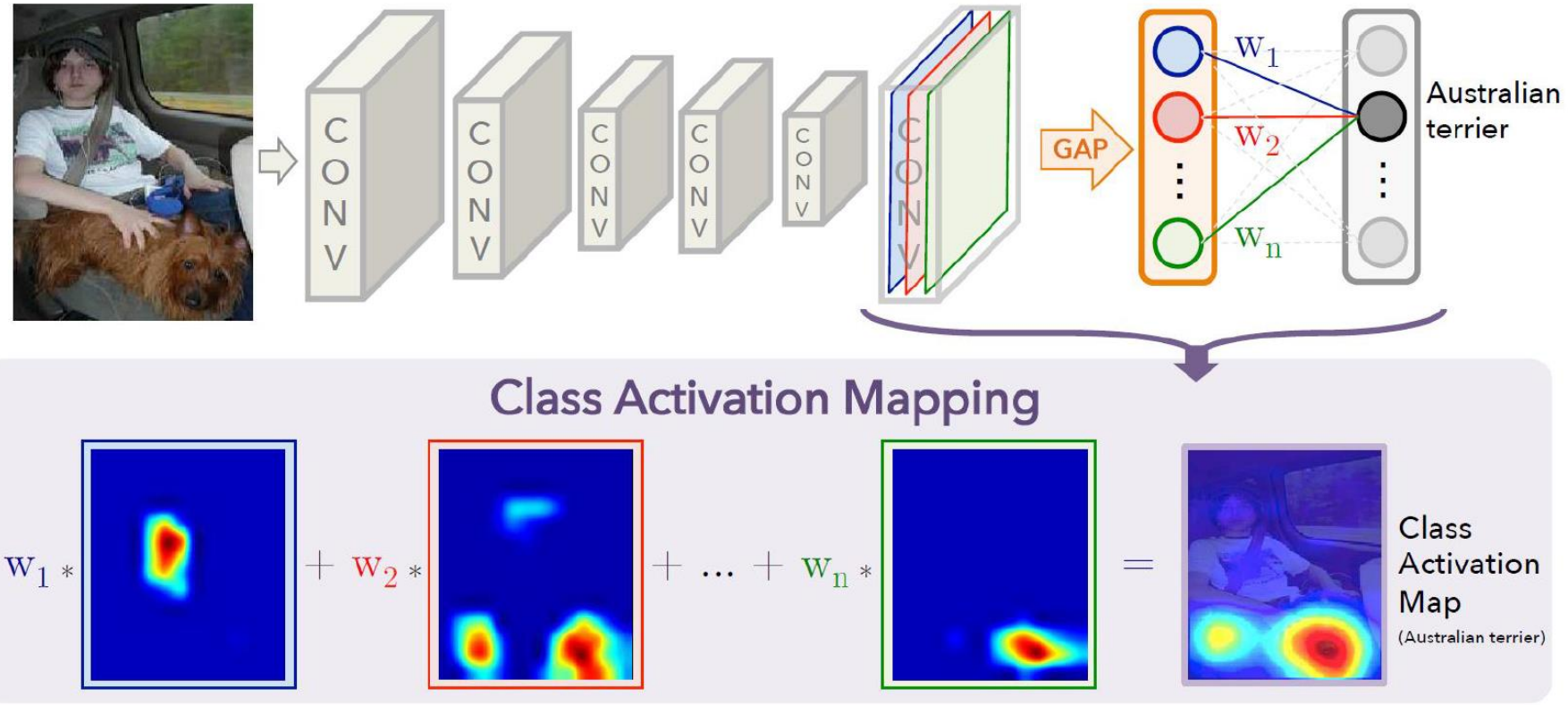
CNN Architecture 5: DenseNet

- Advantage of DenseNet 3: Feature-level diversity can be obtained



CNN Localization

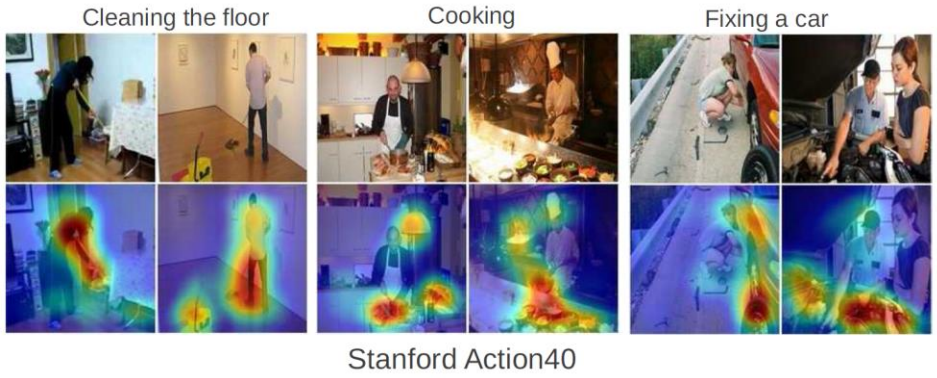
- Image Localization
  - ✓ Class Activation Map (CAM)
    - Localize significant areas based only on class label information

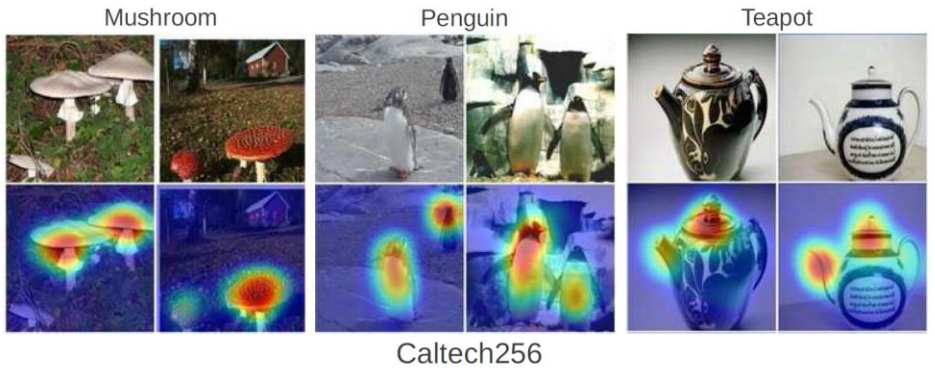


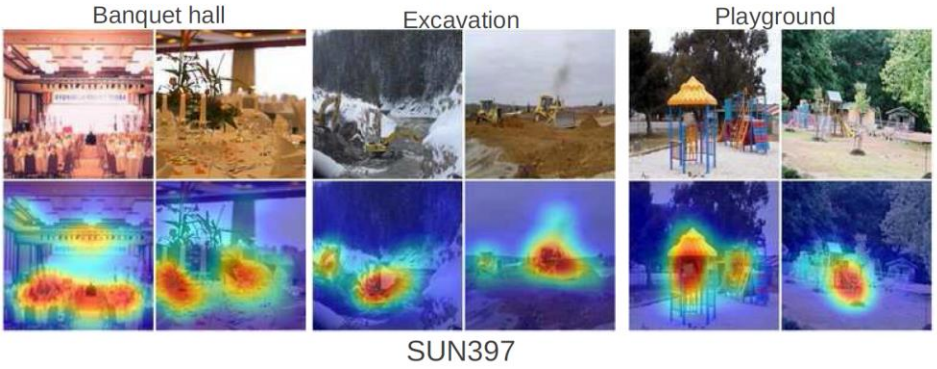
Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2921-2929).

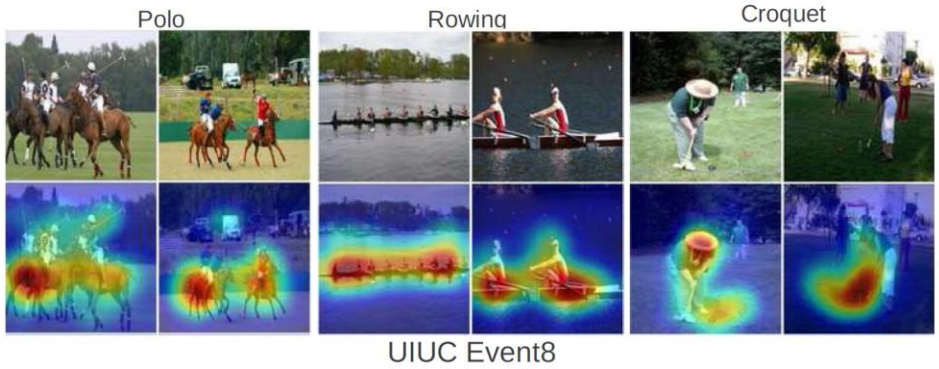
CNN Localization

- Image Localization
  - ✓ Class Activation Map (CAM)
    - Localize significant areas based only on class label information







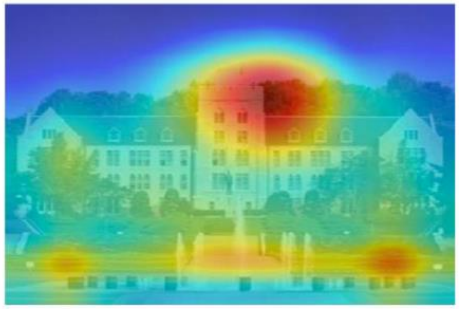


CNN Localization

- Image Localization
  - ✓ Class Activation Map (CAM)
    - Localize significant areas based only on class label information

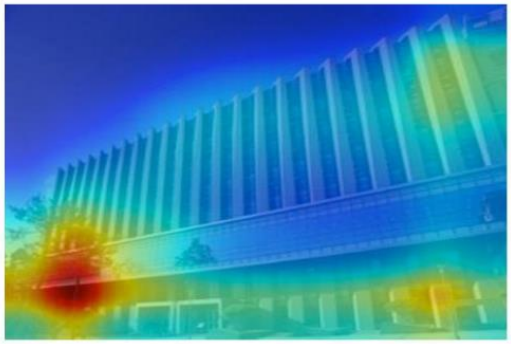
Predictions:

- **Type of environment:** outdoor
- **Semantic categories:** palace:0.23, formal_garden:0.20, mansion:0.15, castle:0.07, courthouse:0.06
- **SUN scene attributes:** man-made, openarea, naturallight, grass, vegetation, foliage, leaves, directsunsunny, trees, vacationingtouring
- **Informative region for the category *palace* is:**



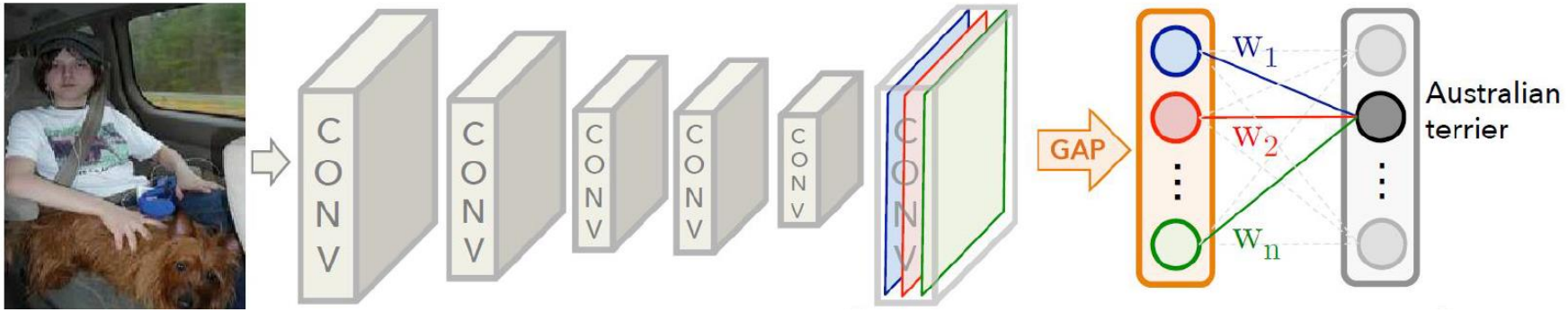
Predictions:

- **Type of environment:** outdoor
- **Semantic categories:** office_building:0.33, building_facade:0.24, hospital:0.17, skyscraper:0.06,
- **SUN scene attributes:** man-made, naturallight, openarea, mostlyverticalcomponents, directsunsunny, clouds, glass, mostlyhorizontalcomponents, semi-enclosedarea, metal
- **Informative region for the category *office_building* is:**



Class Activation Map

- CAM Process



✓ $f_k(x, y)$: activation of the last convolution layer unit k (x, y): spatial location

✓ Global average pooling:

$$F_k = \sum_{x,y} f_k(x, y)$$

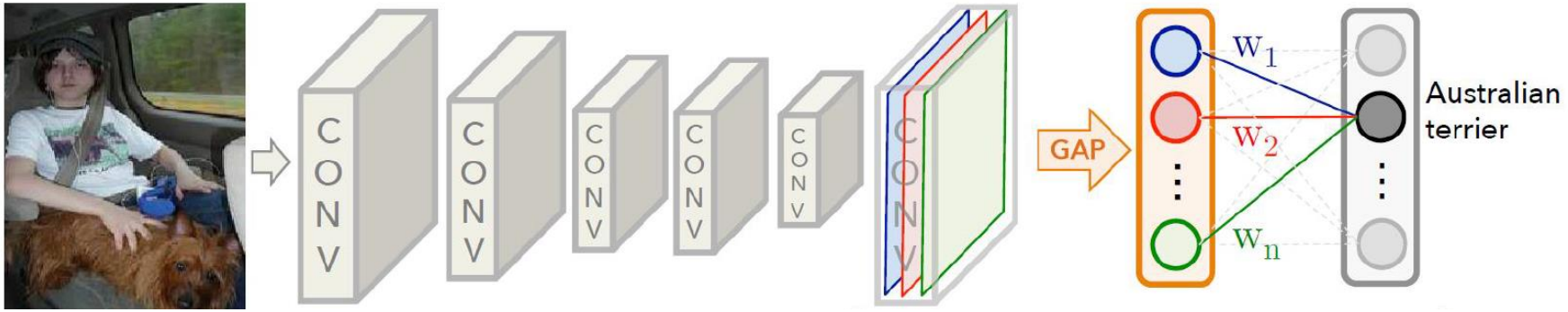
✓ Input for the Softmax function of class C

$$S_c = \sum_k w_k^c F_k$$

- w_k^c : weight of unit k for class C

Class Activation Map

- CAM Process



✓ Output of the softmax for class C

$$\frac{\exp(S_c)}{\sum_c \exp(S_c)}$$

✓ S_c can be defined as follows. M_c can be class activation map for class C

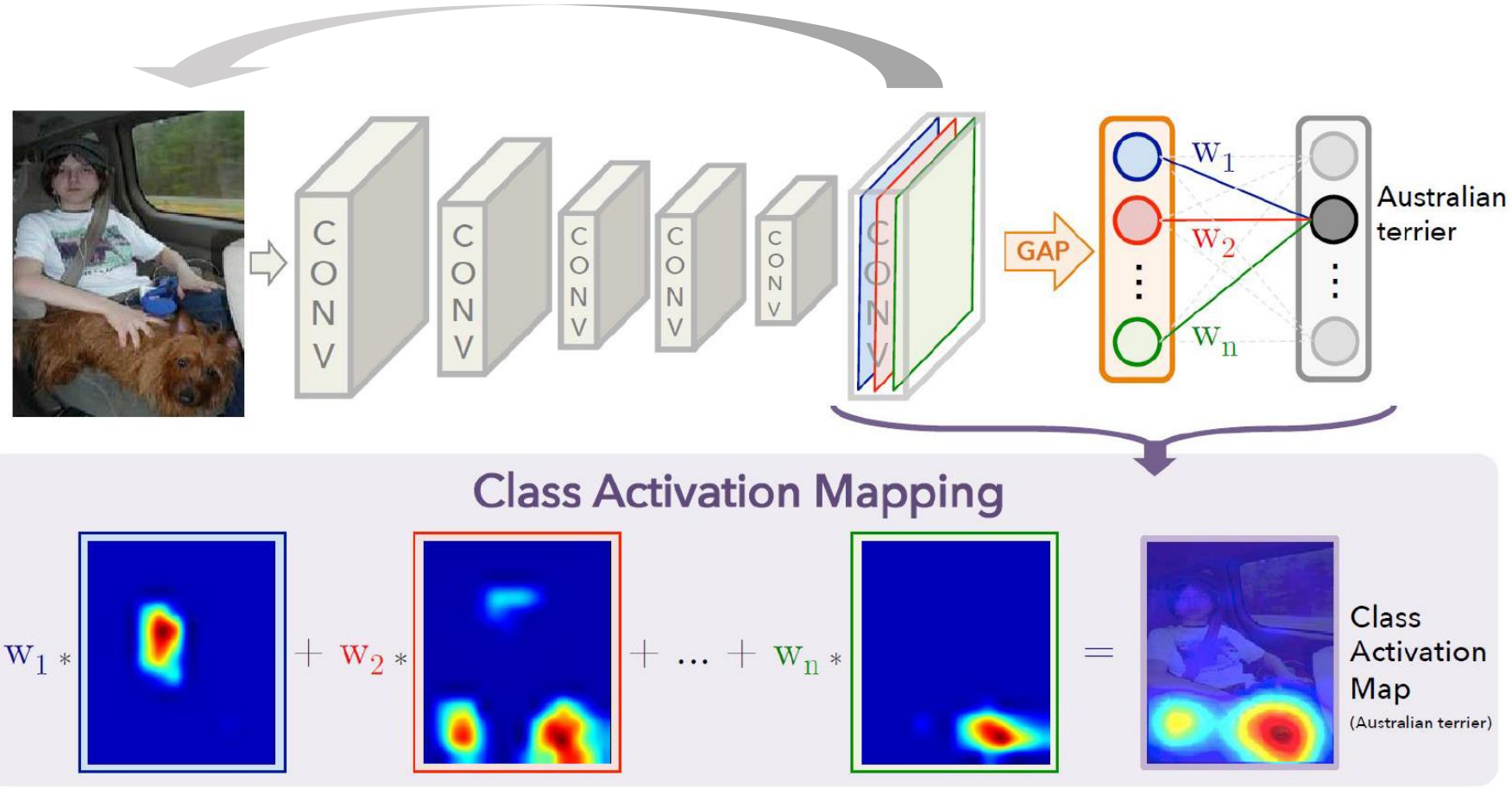
$$S_c = \sum_k w_k^c \sum_{x,y} f_k(x,y) = \sum_{x,y} \sum_k w_k^c f_k(x,y)$$

$$M_c(x,y) = \sum_k w_k^c f_k(x,y)$$

Class Activation Map

- CAM Process

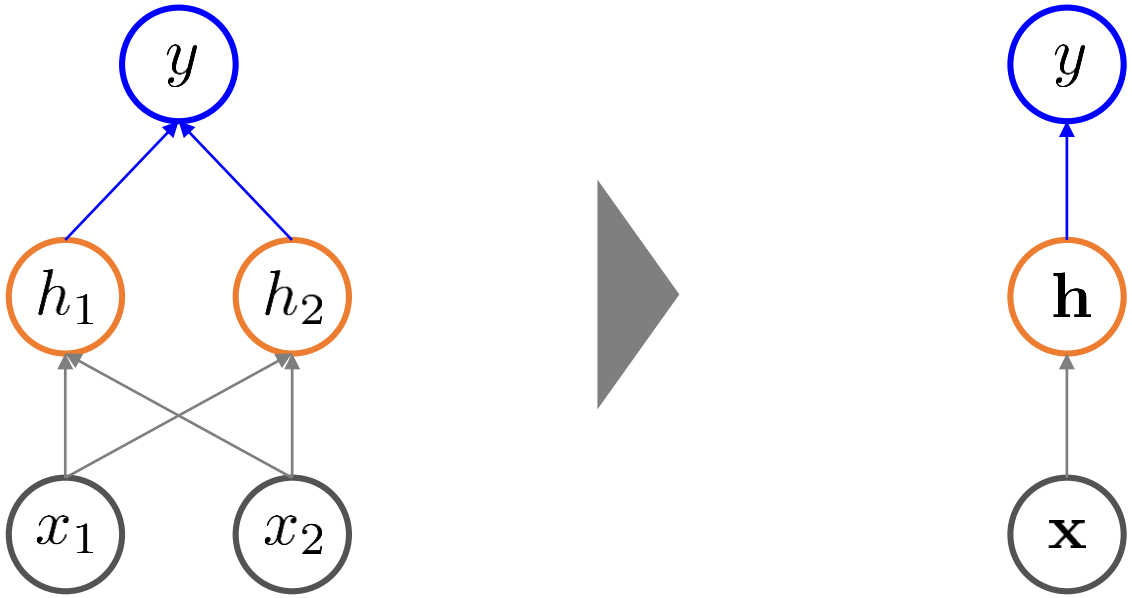
Upsampling is commomly used because the raw image size (I) and feature map size(F) are different (I > F)



AGENDA

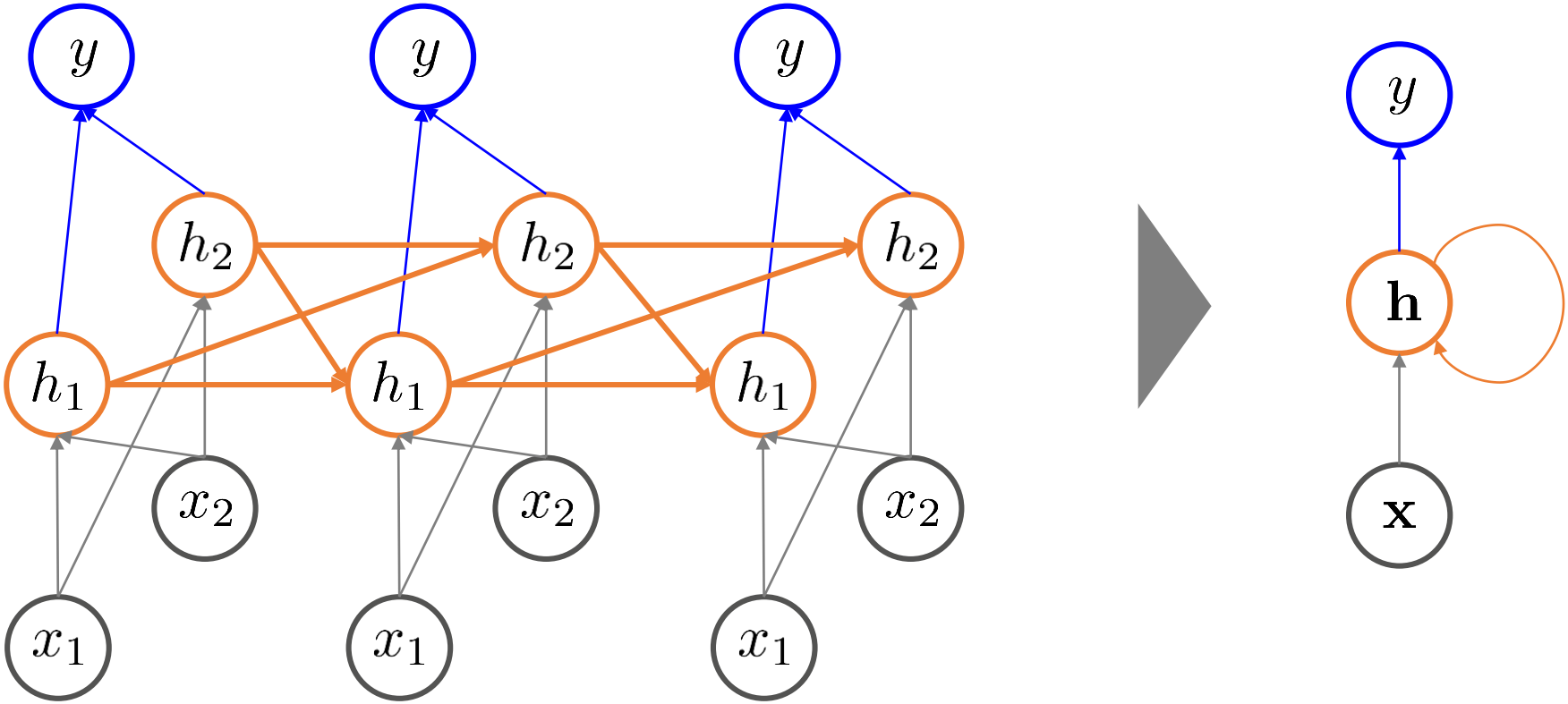
RNN Basics: Sequence

- NN structure without sequence



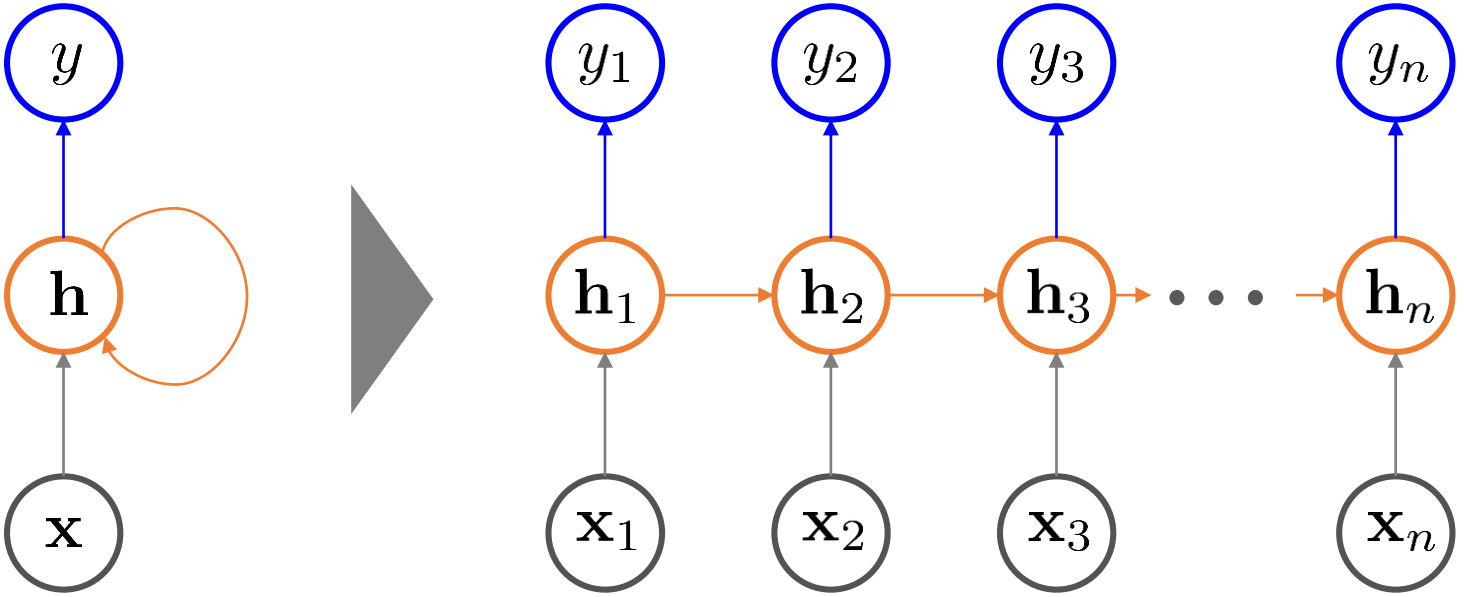
RNN Basics: Sequence

- NN structure with sequence



RNN Basics: Sequence

- NN structure with sequence (vector representation)



RNN Basics: Sequence

- NN structure with sequence
  - ✓ Input-Output structure

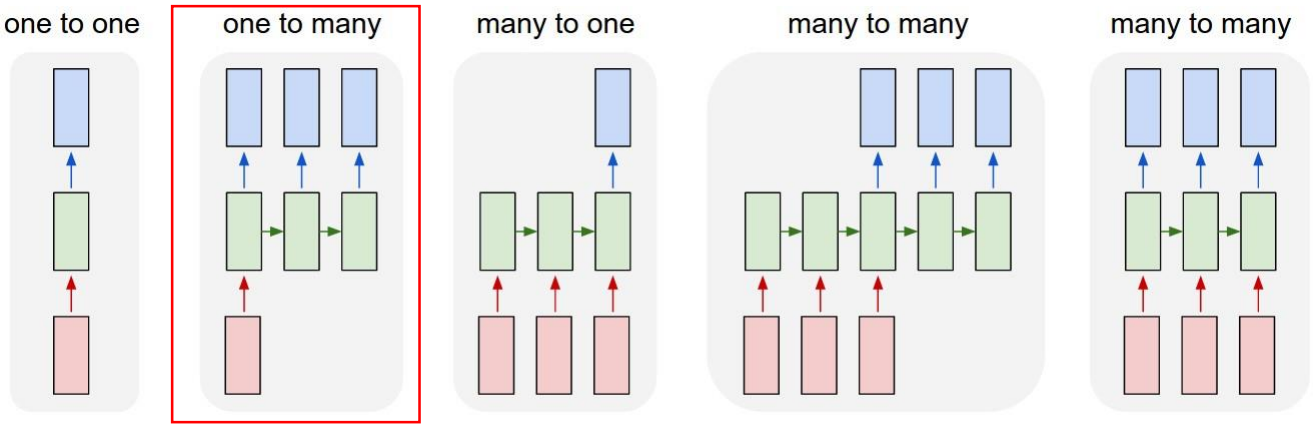
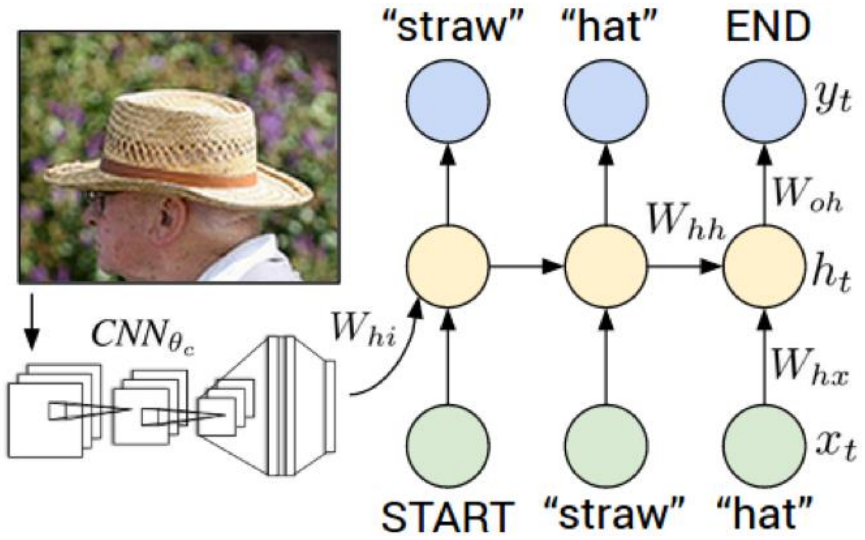
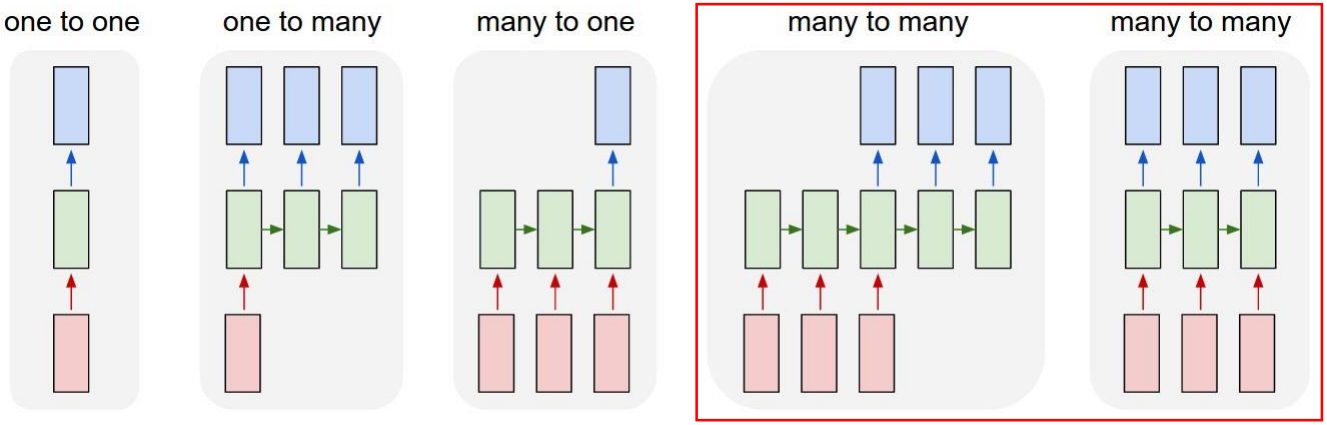


Image captioning

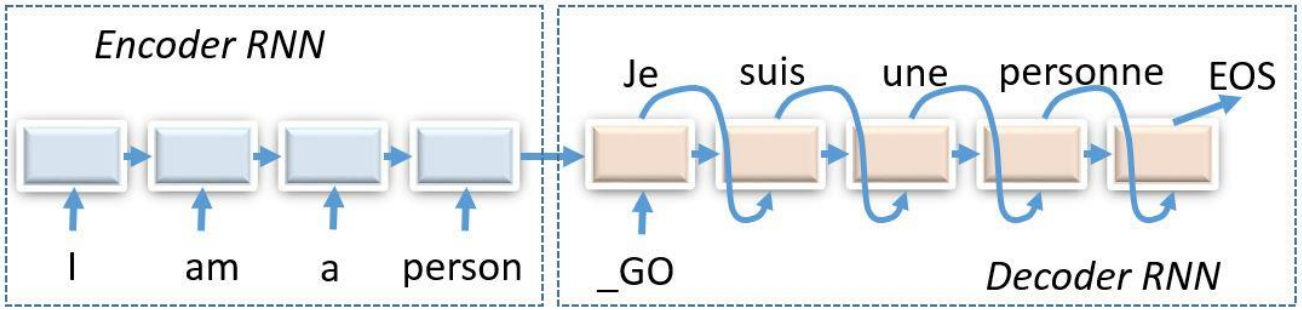


RNN Basics: Sequence

- NN structure with sequence
  - ✓ Input-Output structure



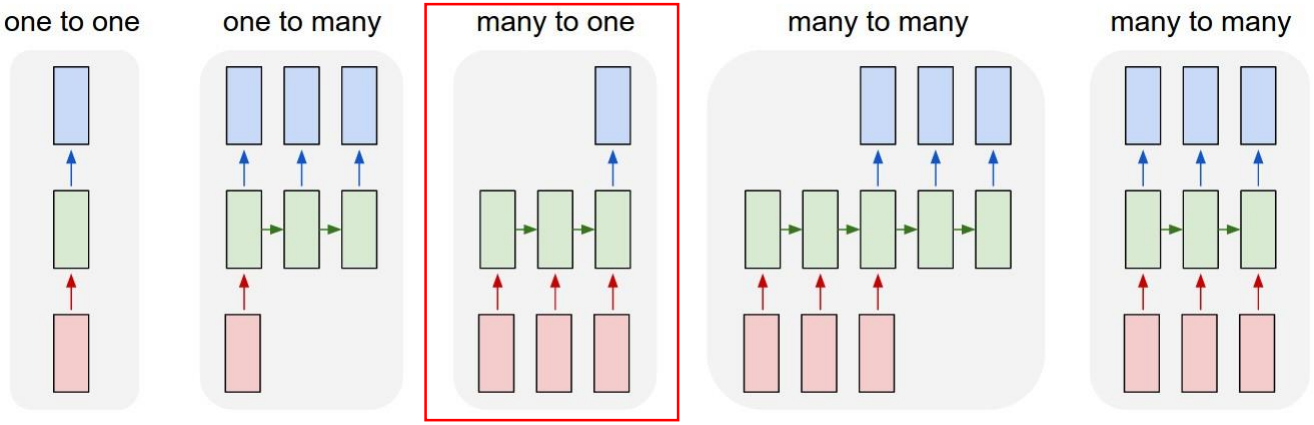
Machine translation/Dialog system



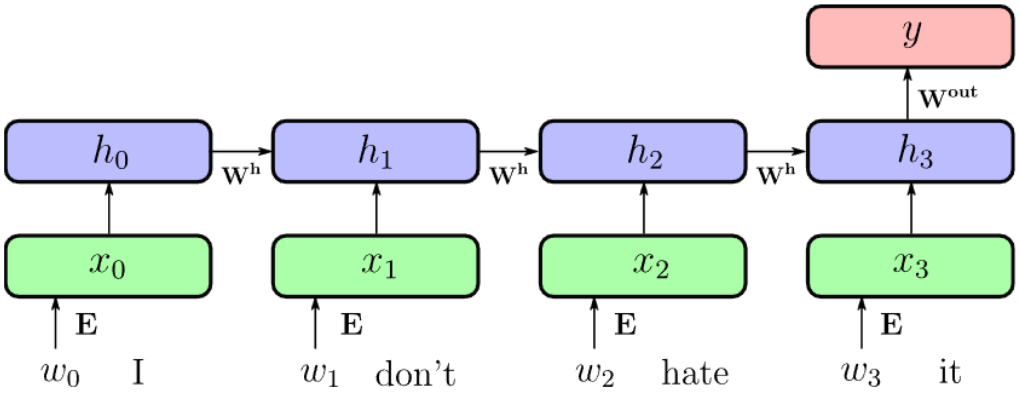
https://esciencegroup.com/2016/03/04/fun-with-recurrent-neural-nets-one-more-dive-into-cntk-and-tensorflow/

RNN Basics: Sequence

- NN structure with sequence
  - ✓ Input-Output structure

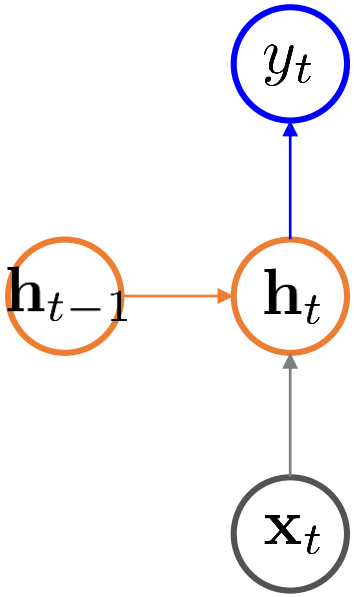


Text classification/Sentiment analysis



RNN Basics: Forward Path

- Information flow in Vanilla RNN
  - ✓ f( ) and g( ) are activation functions

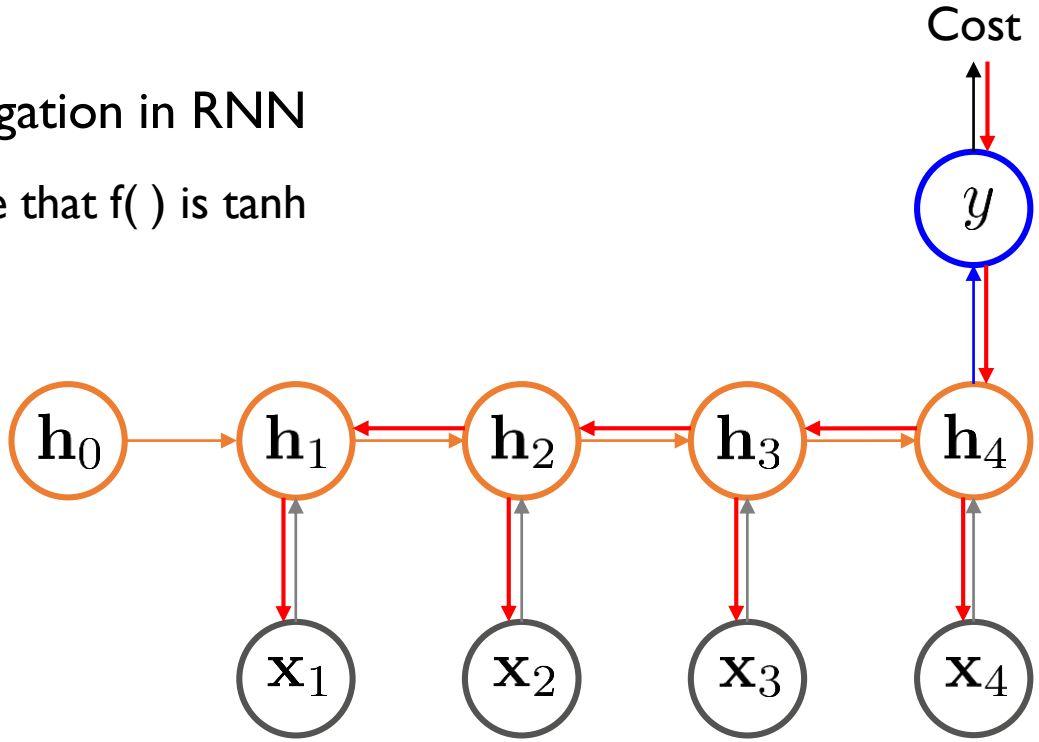


$$y_t = g(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y)$$

$$\mathbf{h}_t = f(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_x)$$

RNN Basics: Gradient Vanishing/Exploding Problem

- Backpropagation in RNN
  - ✓ Assume that f( ) is tanh



$$\frac{\partial Cost}{\partial \mathbf{W}_{xh}} = \frac{\partial Cost}{\partial y} \times \frac{\partial y}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{W}_{xh}} + \frac{\partial Cost}{\partial y} \times \frac{\partial y}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \times \frac{\partial \mathbf{h}_3}{\partial \mathbf{W}_{xh}}$$
$$+ \frac{\partial Cost}{\partial y} \times \frac{\partial y}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \times \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \times \frac{\partial \mathbf{h}_2}{\partial \mathbf{W}_{xh}}$$
$$+ \frac{\partial Cost}{\partial y} \times \frac{\partial y}{\partial \mathbf{h}_4} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \times \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \times \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \times \frac{\partial \mathbf{h}_1}{\partial \mathbf{W}_{xh}}$$

RNN Basics: Gradient Vanishing/Exploding Problem

- Backpropagation in RNN
  - ✓ In general

$$\frac{\partial Cost}{\partial \mathbf{W}_{xh}} = \sum_{i=1}^{n} \left(\frac{\partial Cost}{\partial y} \cdot \frac{\partial y}{\partial \mathbf{h}_n} \cdot \left(\prod_{j=i}^{n-1} \frac{\partial \mathbf{h}_{j+1}}{\partial \mathbf{h}_j} \right) \cdot \frac{\partial \mathbf{h}_i}{\partial \mathbf{W}_{xh}} \right)$$

  - ✓ Recall that f( ) is tanh and

$$\mathbf{h}_t = f(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_x) = tanh(\mathbf{z}_t)$$

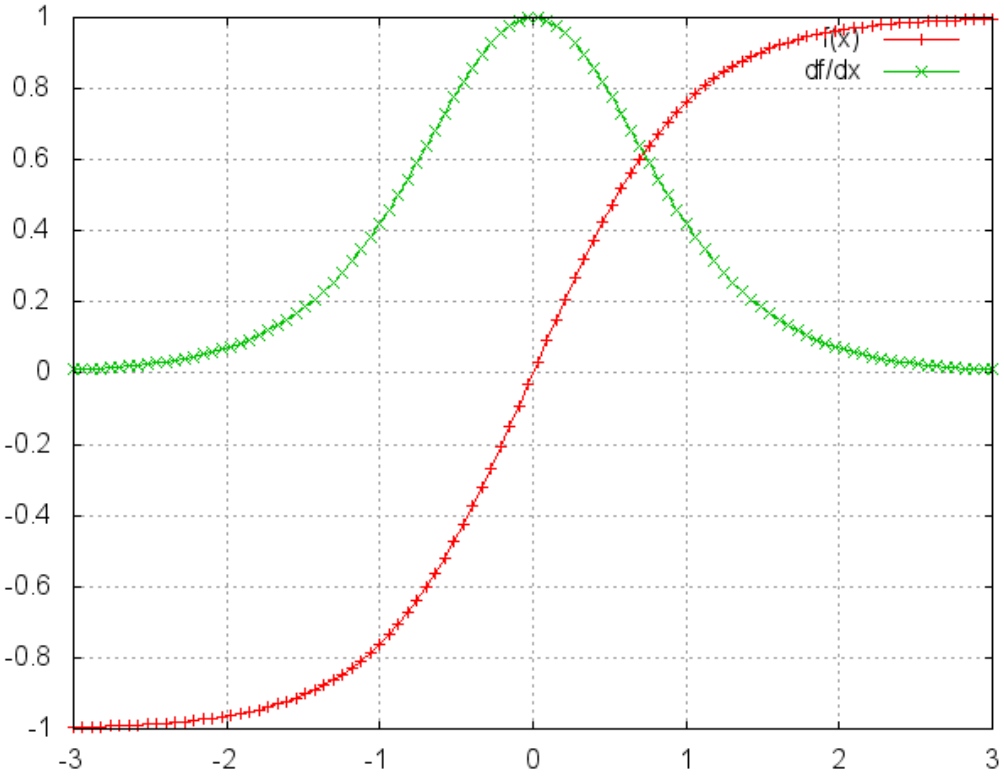
$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \times \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = (1 - tanh^2(\mathbf{z}_t)) \times \mathbf{W}_{hh}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{xh}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \times \frac{\partial \mathbf{z}_t}{\partial \mathbf{W}_{xh}} = (1 - tanh^2(\mathbf{z}_t)) \times \mathbf{x}_t$$

RNN Basics: Gradient Vanishing/Exploding Problem

- Backpropagation in RNN

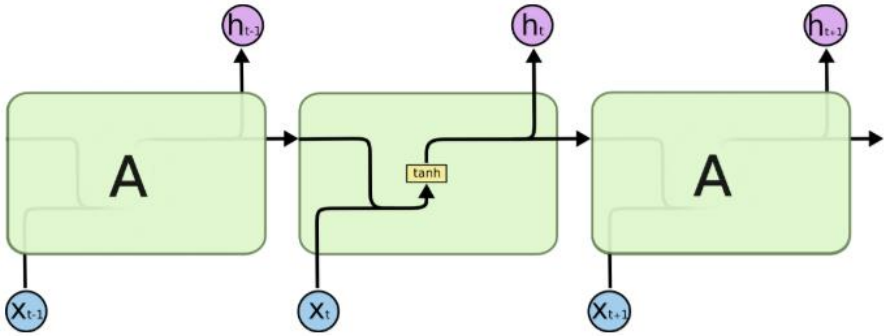
$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \times \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = \boxed{(1 - tanh^2(\mathbf{z}_t))} \times \mathbf{W}_{hh}$$



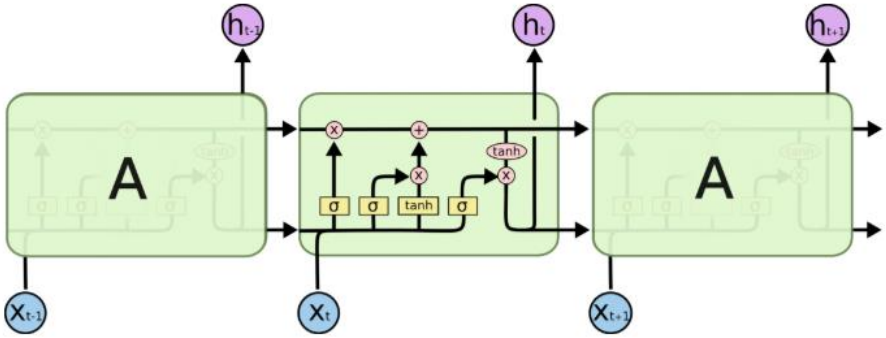
RNN Basic: LSTM

- LSTM: Long Short-Term Memory
  - ✓ Able to learn long-term dependency by preventing gradient exploding/vanishing problem

Vanilla RNN

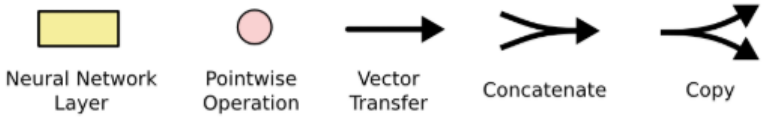


LSTM



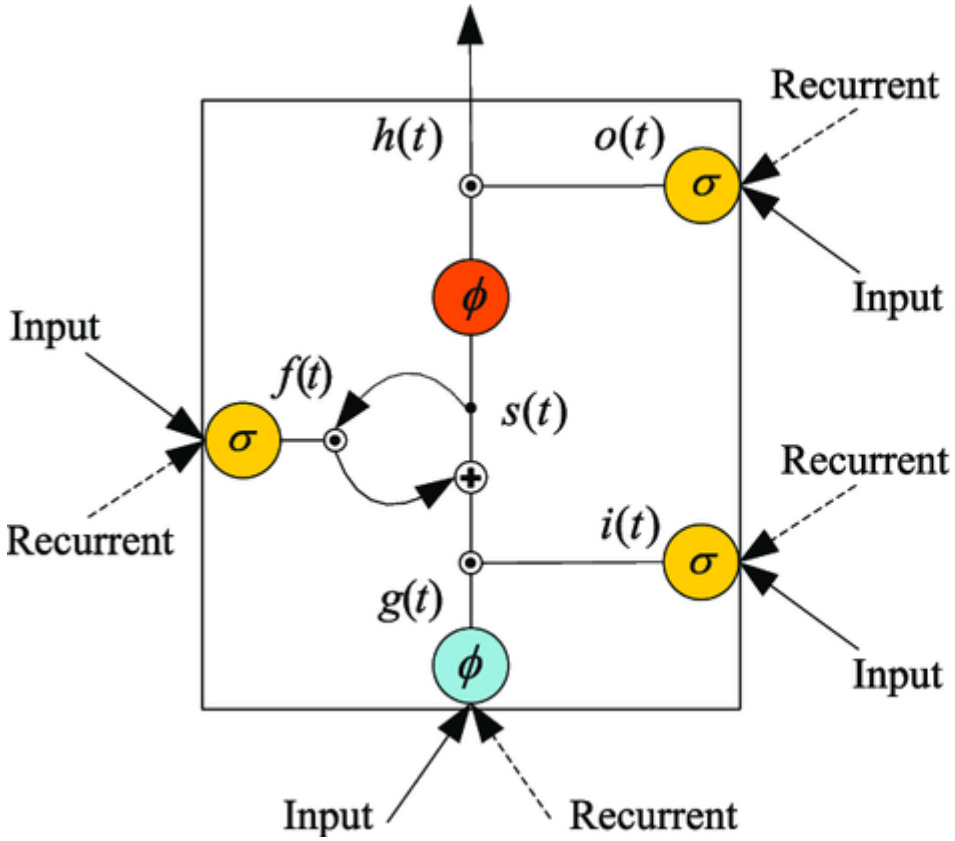
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

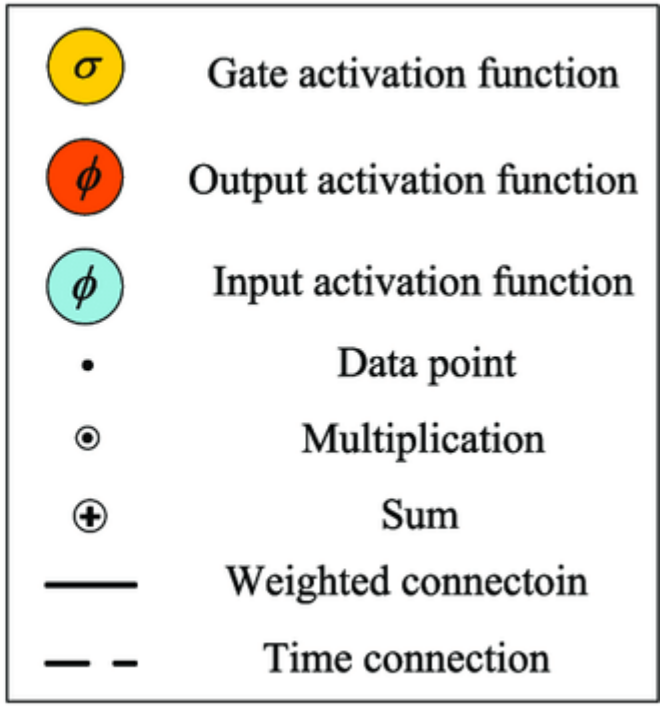
Operation symbols



RNN Basic: LSTM

- LSTM: Long Short-Term Memory
  - ✓ Various diagrams exist for LSTM structure

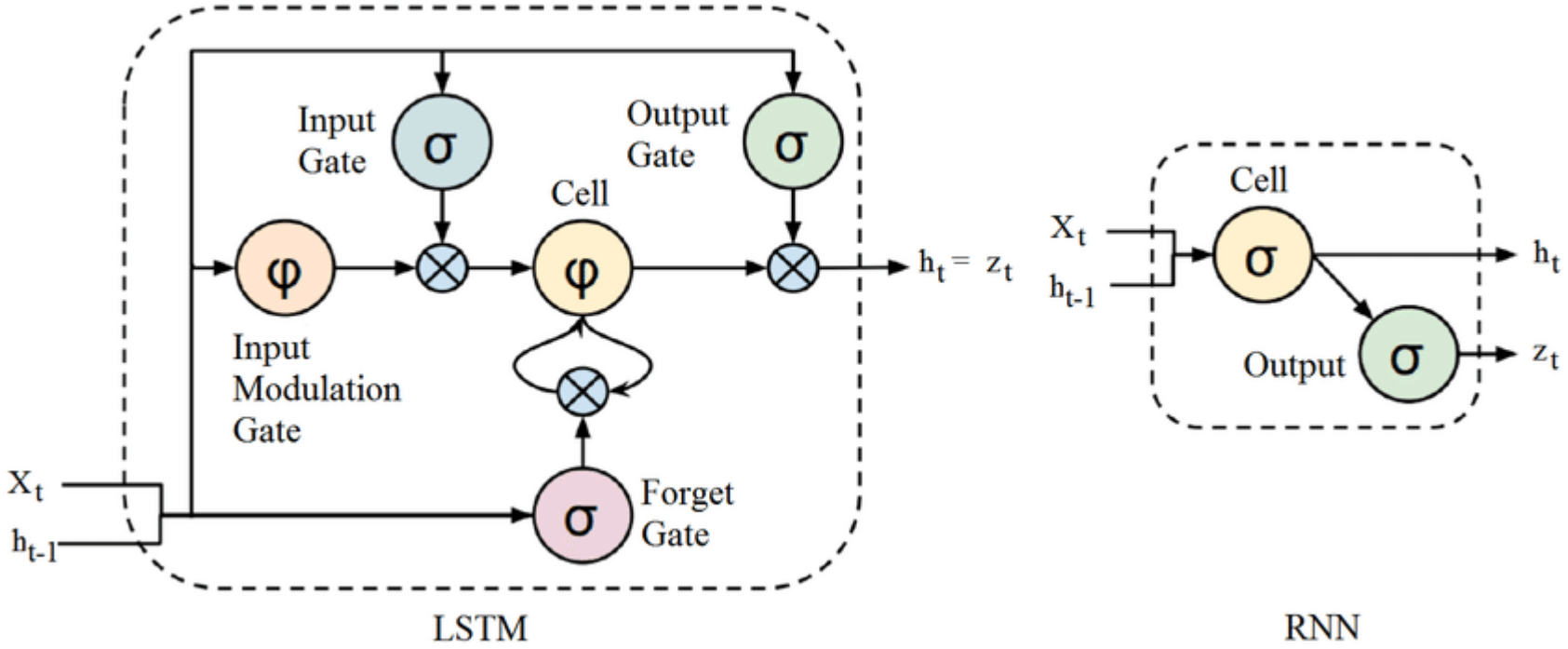




https://www.researchgate.net/figure/The-architecture-of-a-LSTM-memory-cell_fig2_314251130

RNN Basic: LSTM

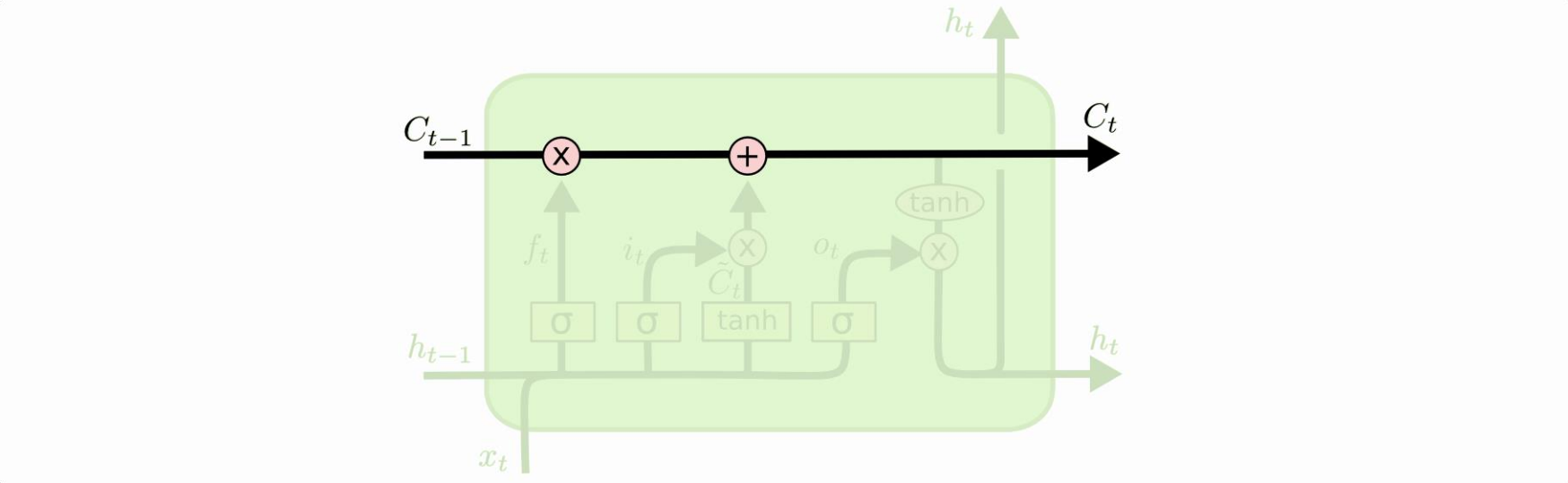
- LSTM: Long Short-Term Memory
  - ✓ Various diagrams exist for LSTM structure



https://www.researchgate.net/figure/An-example-of-a-basic-LSTM-cell-left-and-a-basic-RNN-cell-right-Figure-follows-a_fig2_306377072

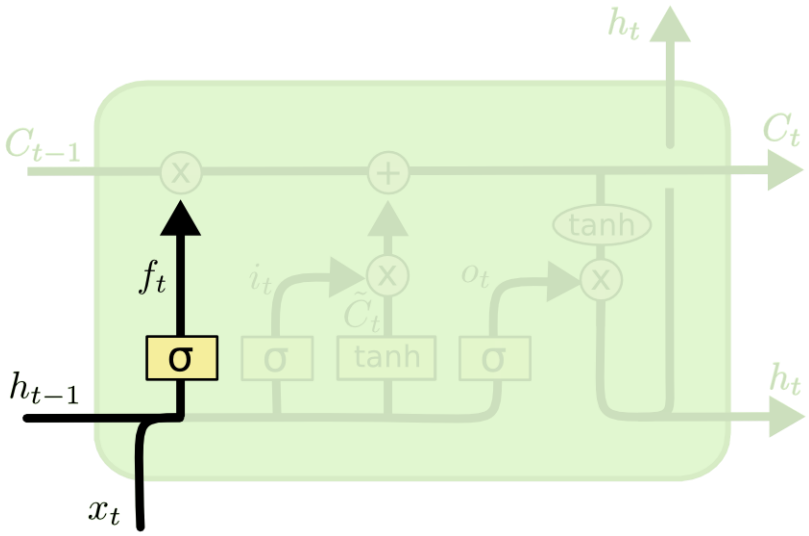
RNN Basic: LSTM

- Cell state
  - ✓ The key to LSTM, the horizontal line running through the top of the diagram



RNN Basic: LSTM

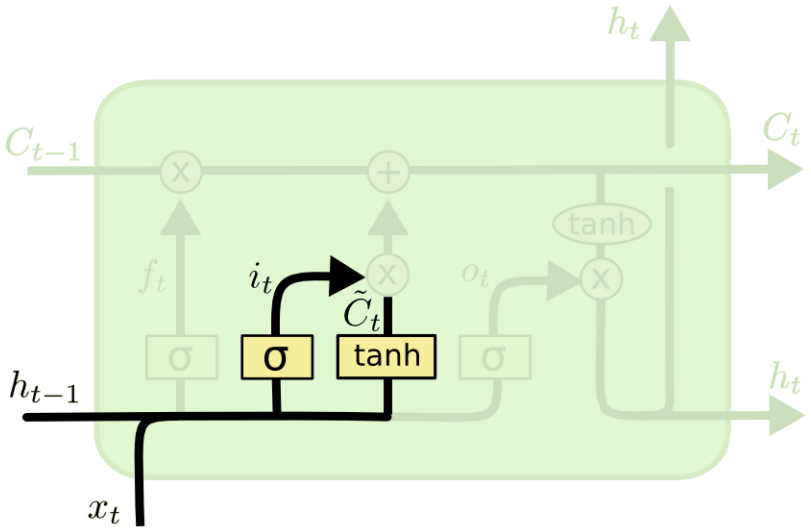
- Step 1: Decide what information we are going to throw away from the cell state
  - ✓ Forget gate: a sigmoid layer that takes the previous hidden state h_{t-1} and the current input x_t and outputs a number between 0 and 1 for the previous cell state
    - 1: completely keep the information in the previous cell state
    - 0: completely ignore the information



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

RNN Basic: LSTM

- Step 2: Decide what new information we are going to store in the cell state
  - ✓ Input gate: a sigmoid layer that decides which values we will update
  - ✓ A tanh layer create a vector of new candidate cell state value $\tilde{C}_t$

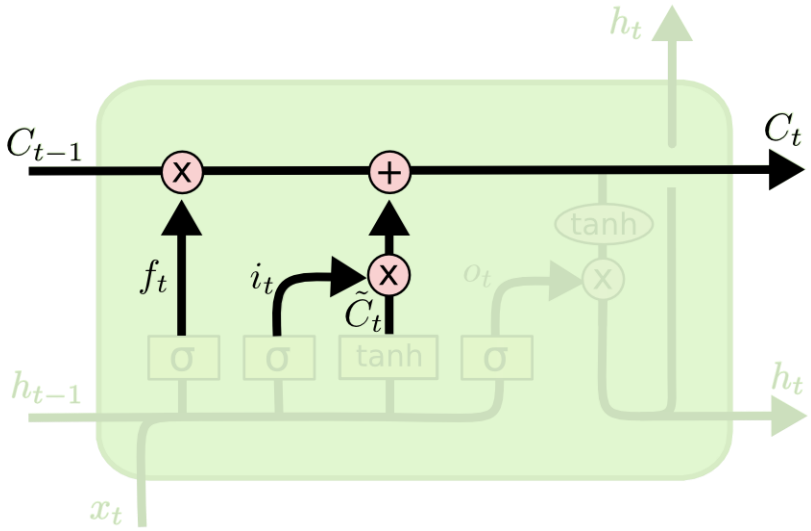


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

RNN Basic: LSTM

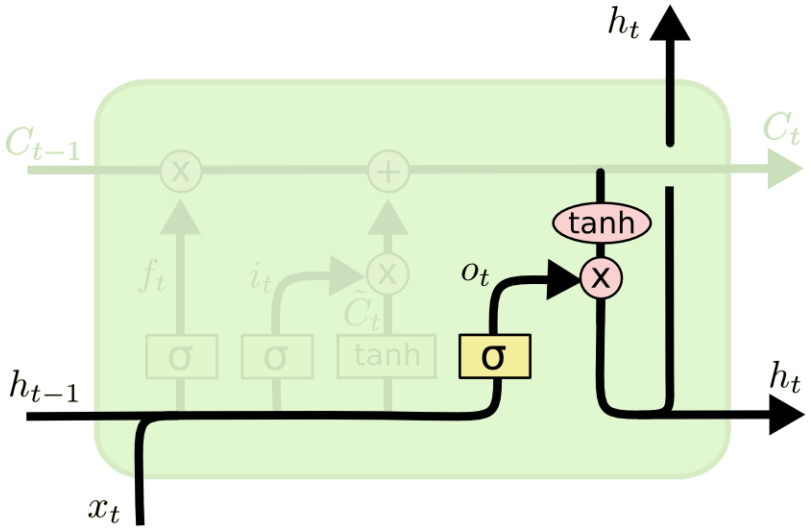
- Step 3: Update the old cell state into the new cell state
  - ✓ Multiply the old state C_{t-1} by f_t and add $i_t * \tilde{C}_t$, which is a new candidate value scaled by how much we decided to update each state value



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

RNN Basic: LSTM

- Step 4: Decide what we are going to output
  - ✓ Based on the current cell state but will be a filtered version

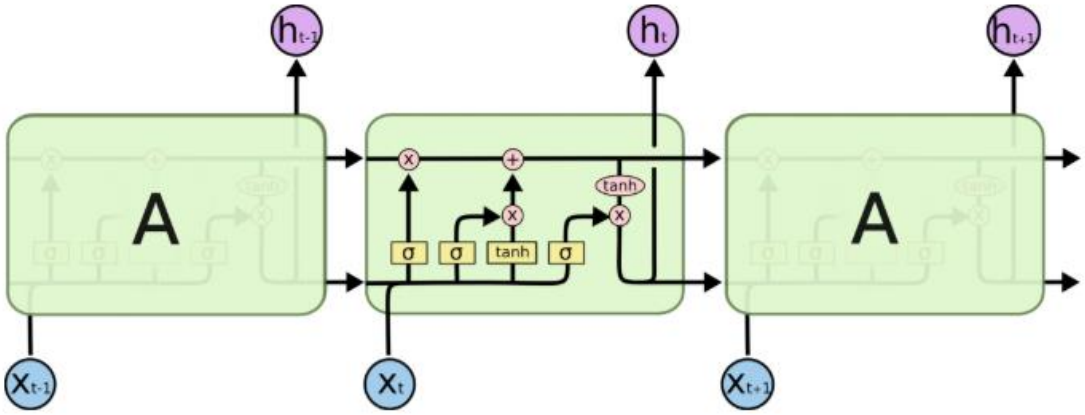


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

RNN Basic: LSTM

- Summary



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

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

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

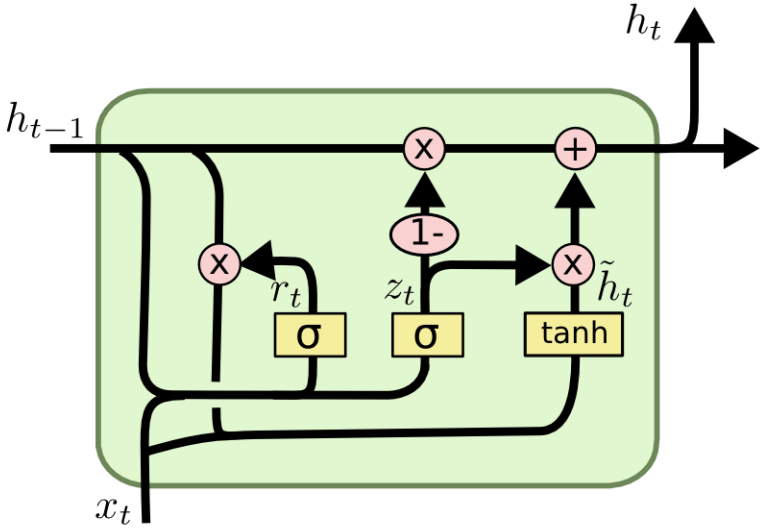
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

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

RNN Basic: GRU

- GRU: Gated Recurrent Unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \quad \text{Update gate}$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \quad \text{Reset gate}$$

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

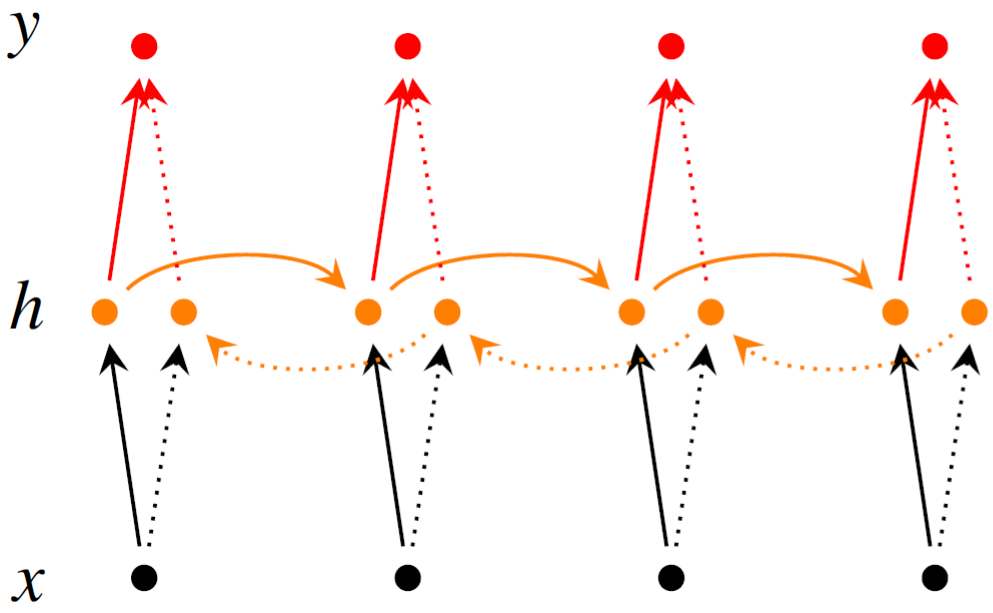
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

✓ Combines the forget and input gates into a single "update gate"

✓ Merges the cell state and hidden state

RNN Variations: Bidirectional RNN

- Bidirectional RNN
  - ✓ Attempts to incorporate information from words both preceding and following



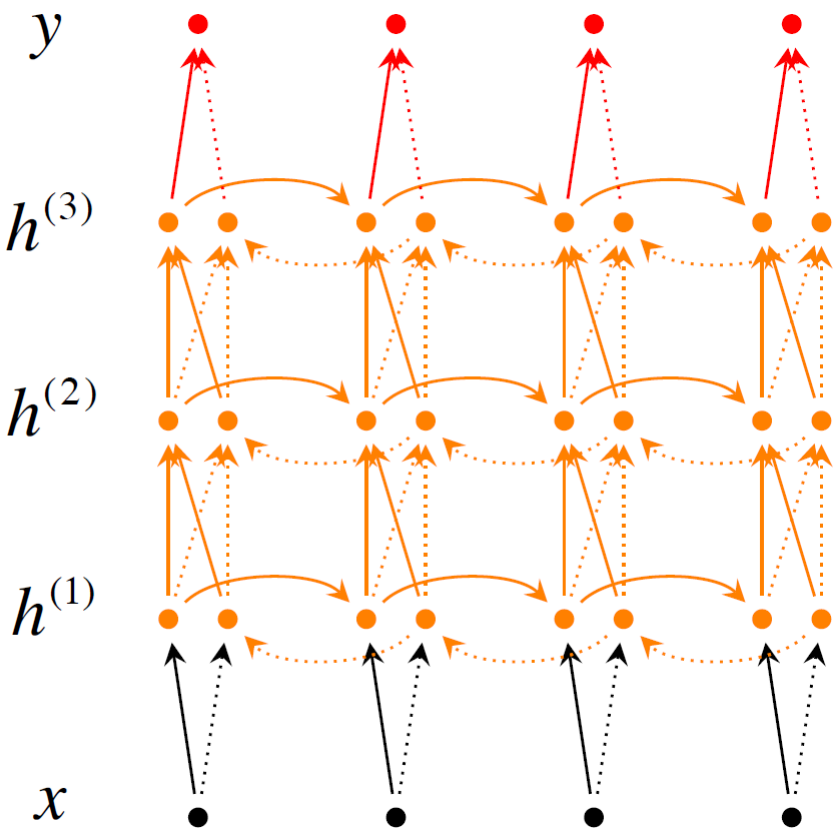
$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$

RNN Variations: Bidirectional RNN

- Deep-Bidirectional RNN
  - ✓ Why not more than one layer of RNN?



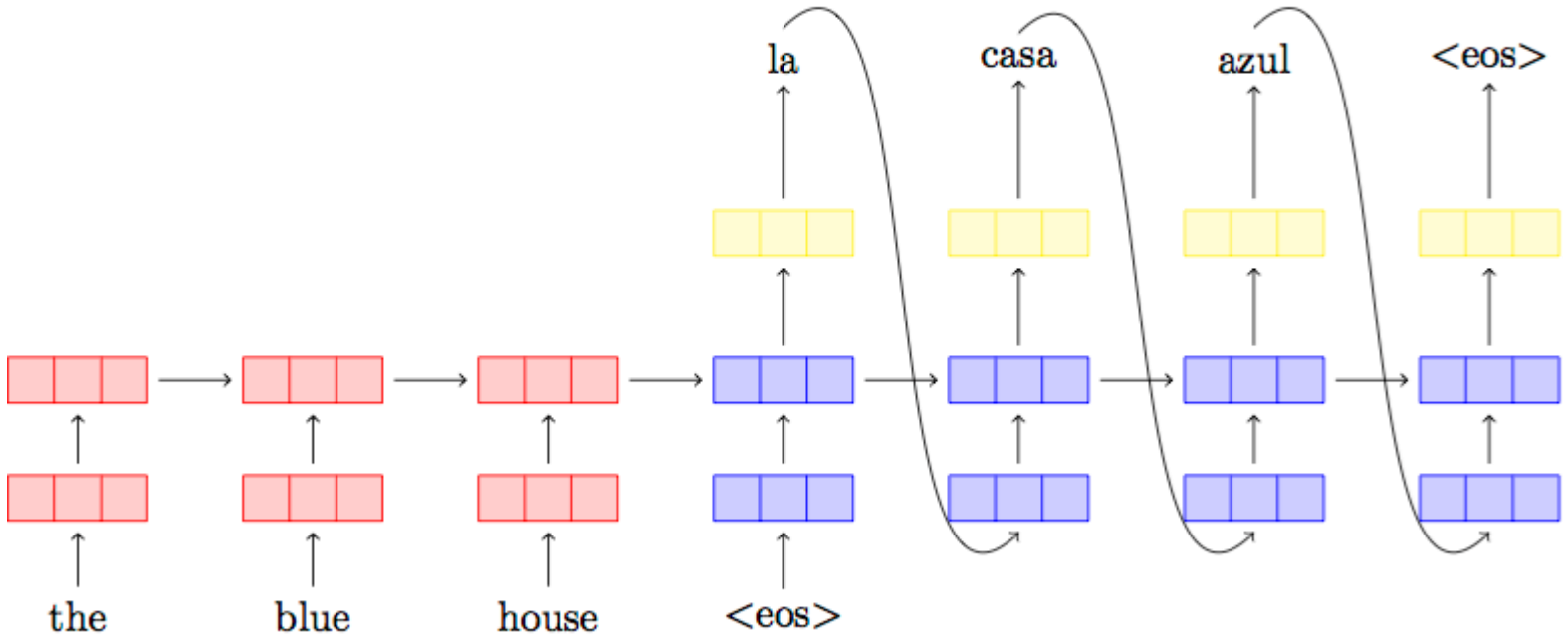
$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

RNN Variations: Sequence to Sequence (Seq2seq)

- Sequence-to-sequence model
  - ✓ Encoder takes a raw input text data and outputs a neural representation
  - ✓ Decoder generates a target sequence given the neural representation of the input sequences and the previously generated output sequences



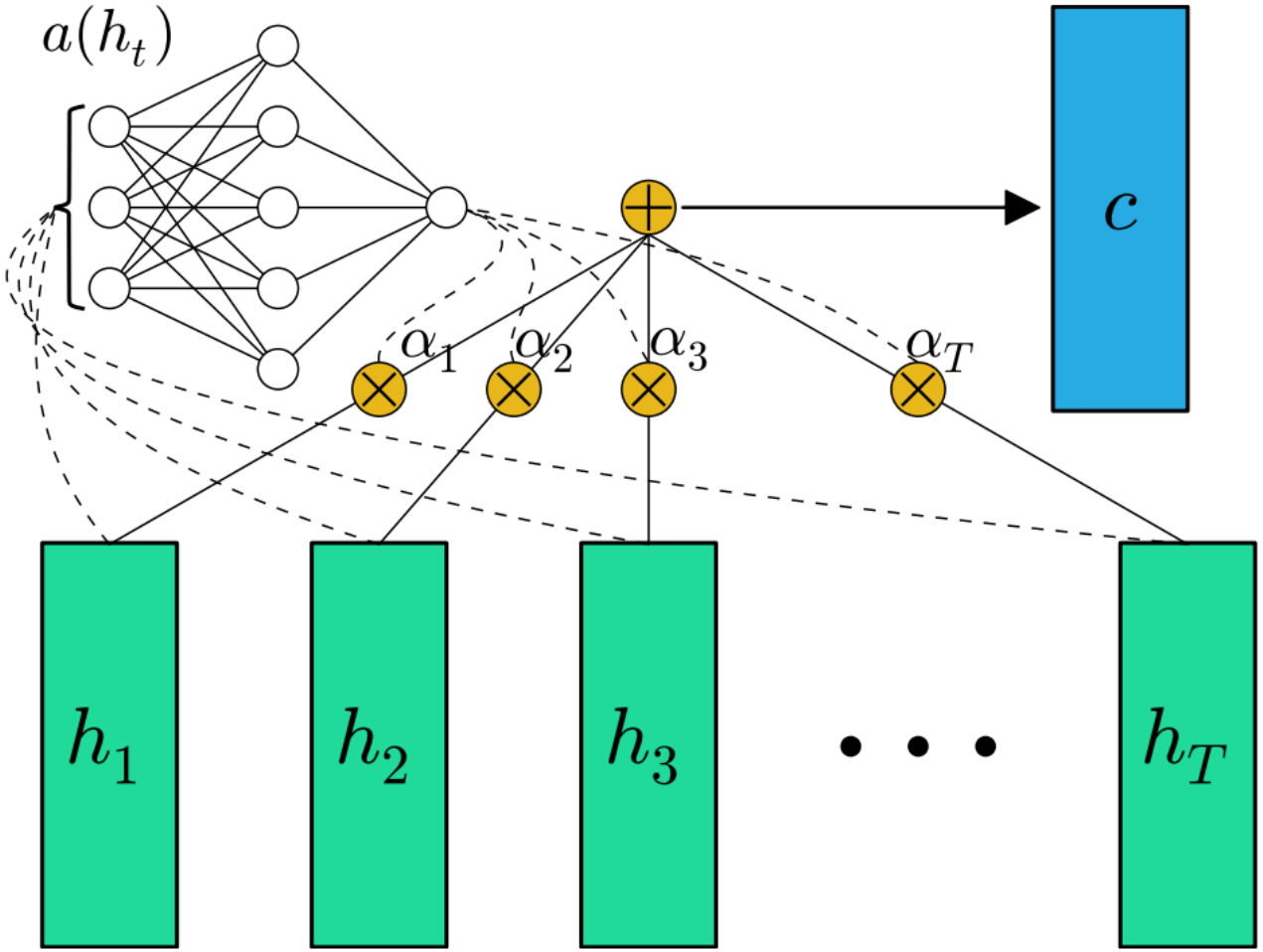
https://tex.stackexchange.com/questions/357613/how-to-draw-a-seq2seq-rnn-model-with-tikz?rq=1

RNN Variations: Sequence to Sequence (Seq2seq)

- Seq2seq Applications
  - Machine Translation (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014b; Cho et al., 2014; Bahdanau et al., 2014; Luong et al., 2015)
  - Question Answering (Hermann et al., 2015)
  - Conversation (Vinyals and Le, 2015) (Serban et al., 2016)
  - Parsing (Vinyals et al., 2014)
  - Speech (Chorowski et al., 2015; Chan et al., 2015)
  - Caption Generation (Karpathy and Li, 2015; Xu et al., 2015; Vinyals et al., 2015)
  - Video-Generation (Srivastava et al., 2015)
  - NER/POS-Tagging (Gillick et al., 2016)
  - Summarization (Rush et al., 2015)

RNN: Attention

- Attention mechanism for finding significant words in document classification

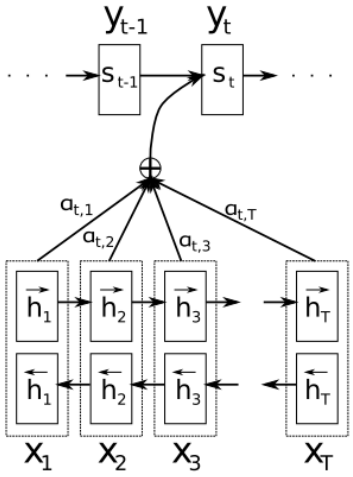


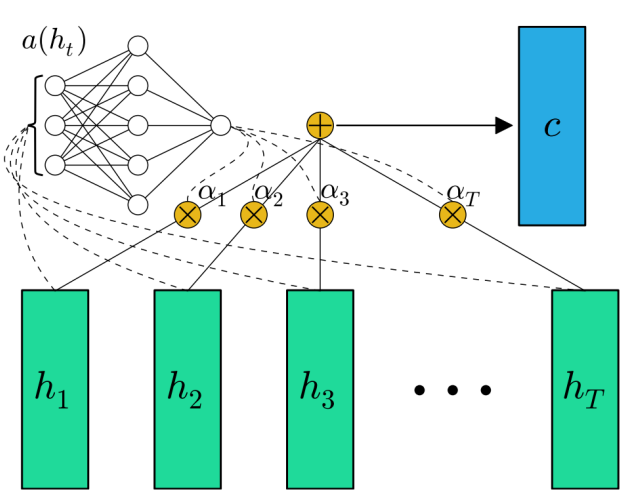
Source: (Raffel and Ellis, 2016)

RNN: Attention

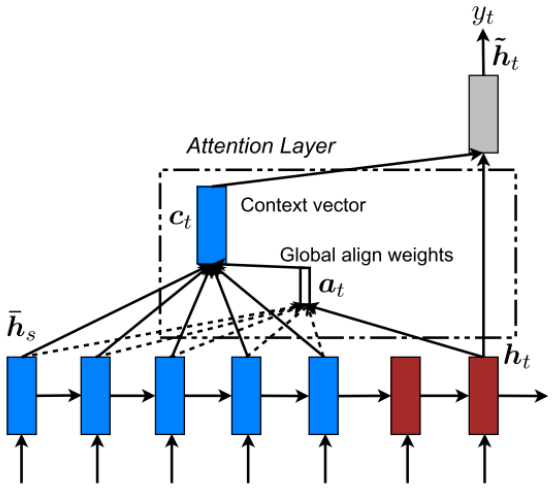
- Two main attention mechanisms
  - ✓ Bahadanau attention (Bahdanau et al., 2015)
    - Attention scores are separated trained, the current hidden state is a function of the context vector and the previous hidden state
  - ✓ Luong attention (Luong et al., 2015)
    - Attention scores are not trained, the new current hidden state is the simple tanh of the weigthed concatenation of the context vector and the current hidden state of the decoder

Bahdanau attention





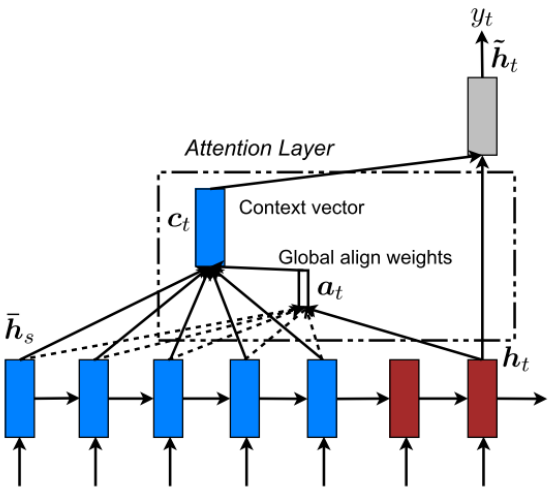
Luong attention



RNN: Attention

- Luong attention
  - ✓ New hidden state of the decoder is the simple tanh of the weigthed concatenation of the context vector and the current hidden state of the decoder:

$$\tilde{\mathbf{h}}_t = tanh(\mathbf{W_c}[\mathbf{c}_t; \mathbf{h}_t])$$



  - ✓ The attention vector is fed through the softmax layer to produce the predictive distribution:

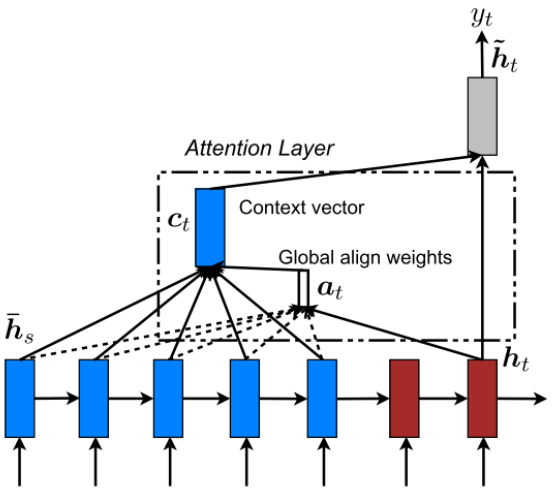
$$p(y_t|y_{y<t}, x) = \text{softmax}(\mathbf{W_s}\tilde{\mathbf{h}}_t)$$

RNN: Attention

- Luong attention
  - ✓ A variable-length alignment vector:

$$\mathbf{a}_t(s) = \text{align}(\mathbf{h}_t, \overline{\mathbf{h}}_s)$$

$$= \frac{exp\Big(\text{score}(\mathbf{h}_t, \overline{\mathbf{h}}_s)\Big)}{\sum_{s'} exp\Big(\text{score}(\mathbf{h}_t, \overline{\mathbf{h}}'_s)\Big)}$$



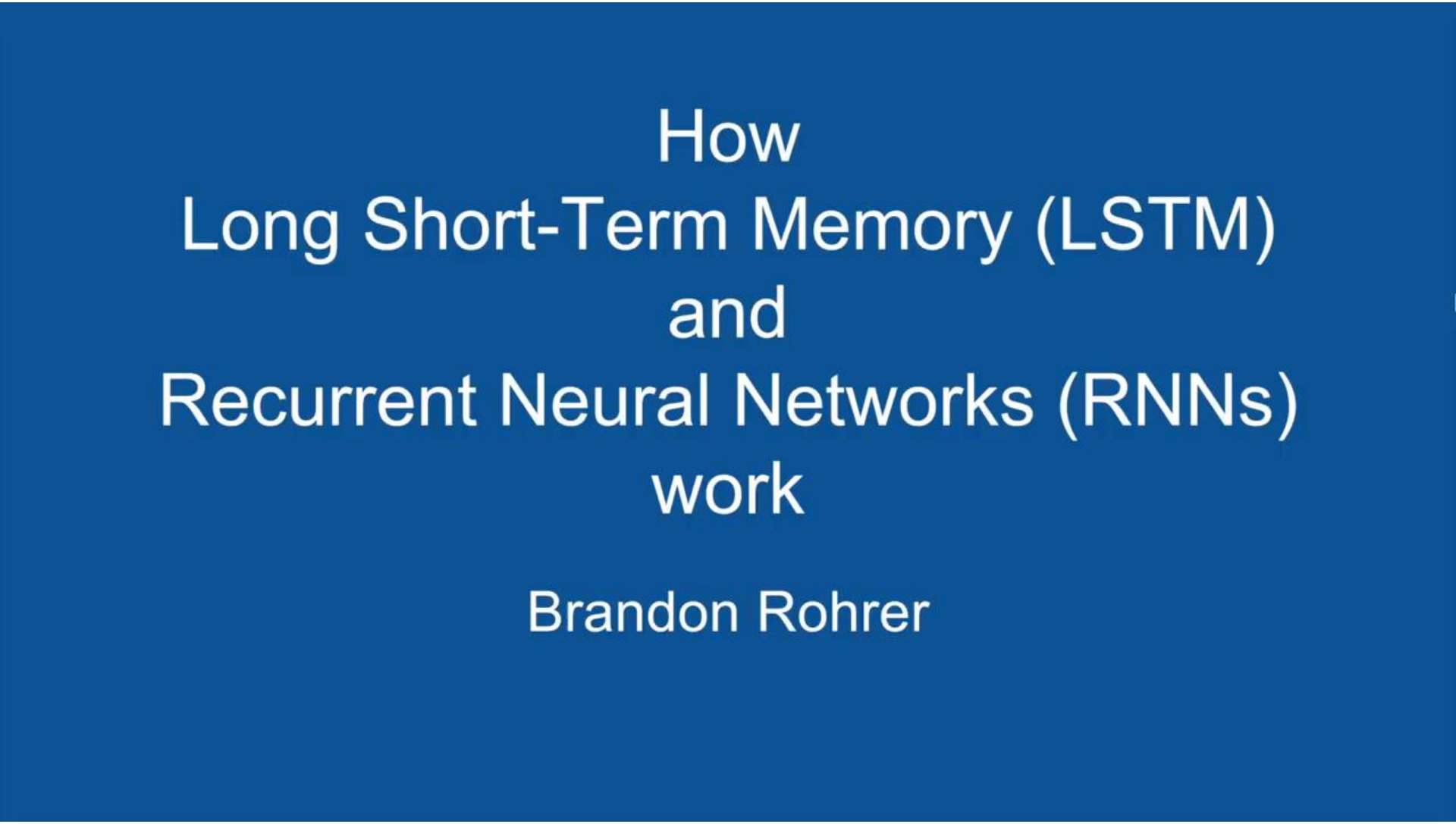
  - ✓ **score** is referred as a context-based function:

$$\text{score}(\mathbf{h}_t, \overline{\mathbf{h}}_s) = \begin{cases} \mathbf{h}_t{}^T \overline{\mathbf{h}}_s, & dot \\ \mathbf{h}_t{}^T \mathbf{W_a} \overline{\mathbf{h}}_s, & general \\ \mathbf{v}_a^T tanh(\mathbf{W_c}[\mathbf{c}_t; \mathbf{h}_t]), & concat \end{cases}$$

  - ✓ Context vector

$$\mathbf{c}_t = \overline{\mathbf{h}}_s \mathbf{a}_t$$

RNN Procedure

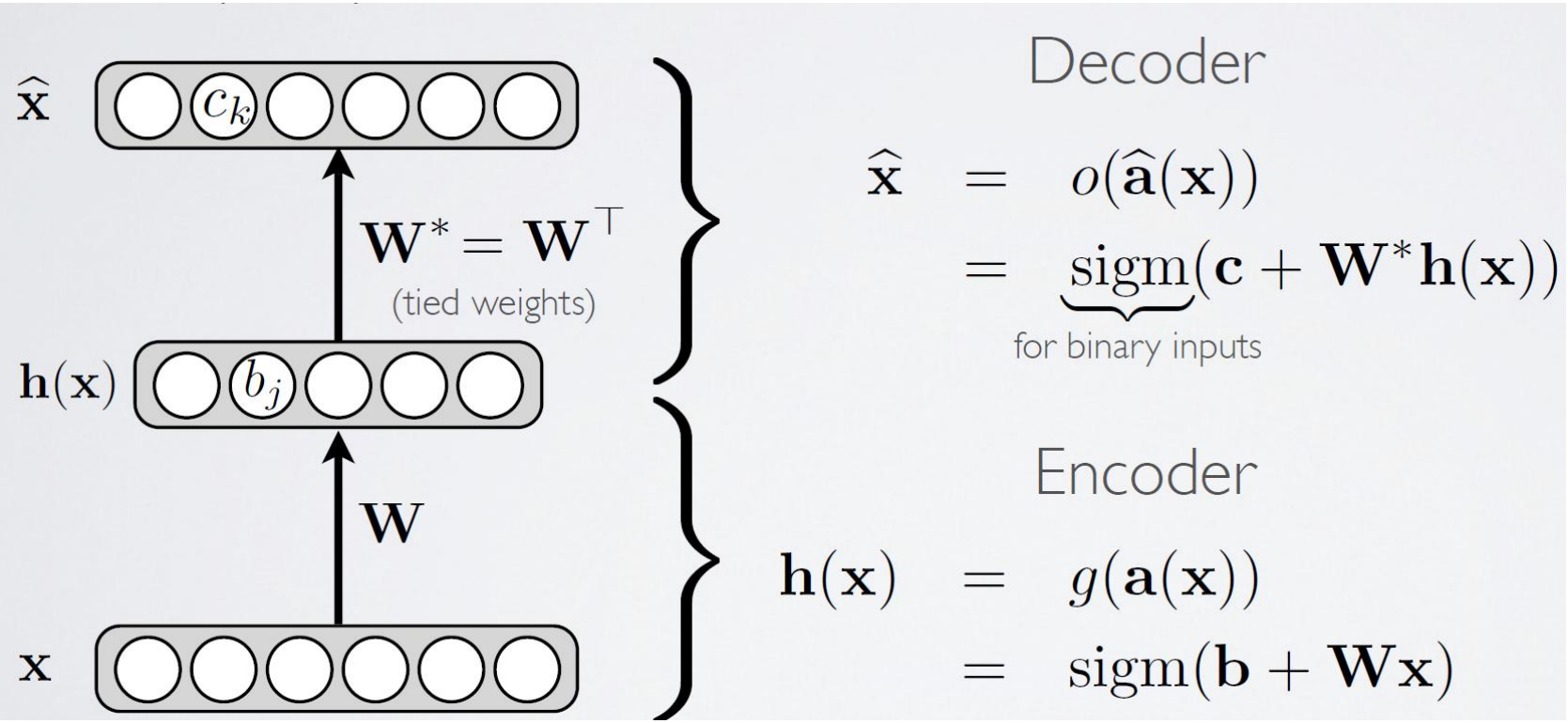


https://www.youtube.com/watch?v=WCUNPb-5EYI

AGENDA

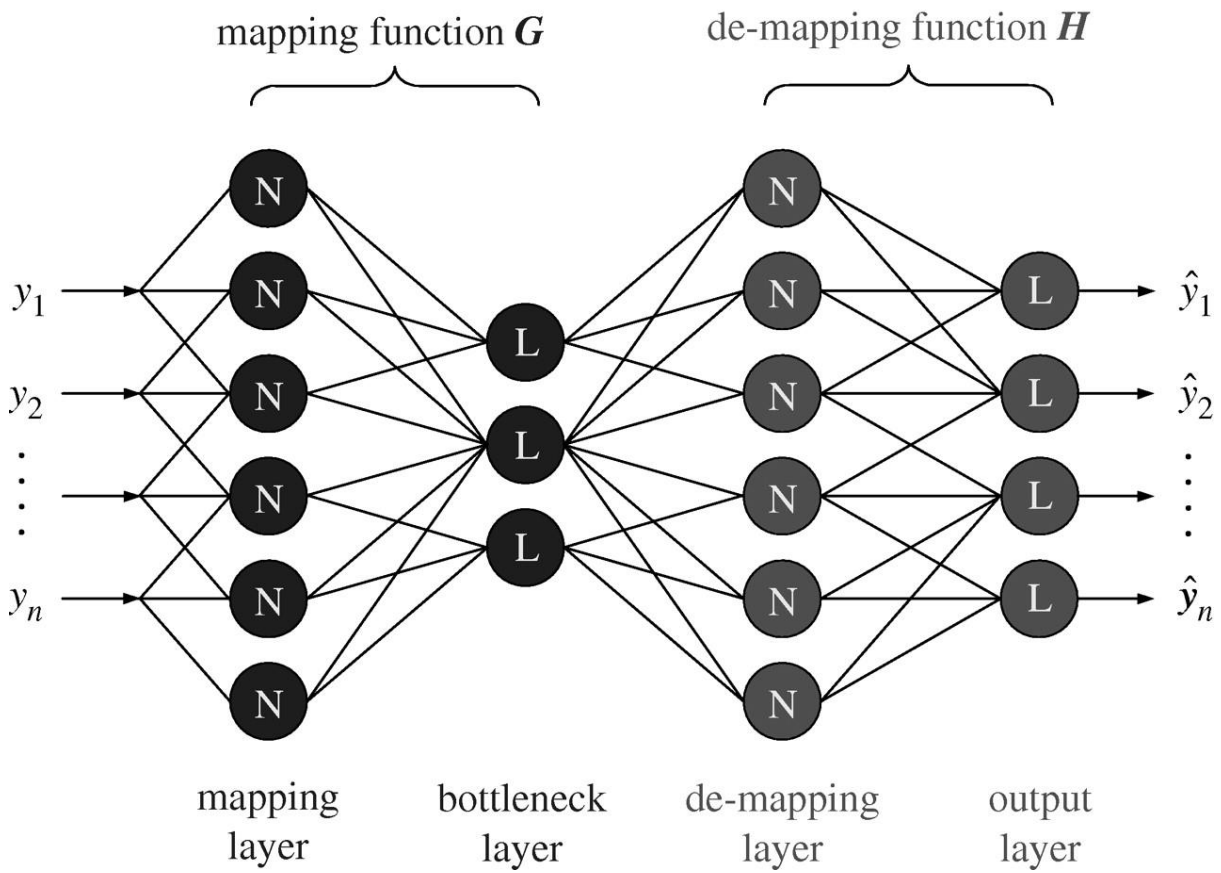
Auto-Encoder

- Auto-Encoder (Auto-Associative Neural Network)
  - ✓ Neural network with the same input and output
    - Loss function: $l(f(\mathbf{x})) = \frac{1}{2}\sum_k (\hat{x}_k - x_k)^2$



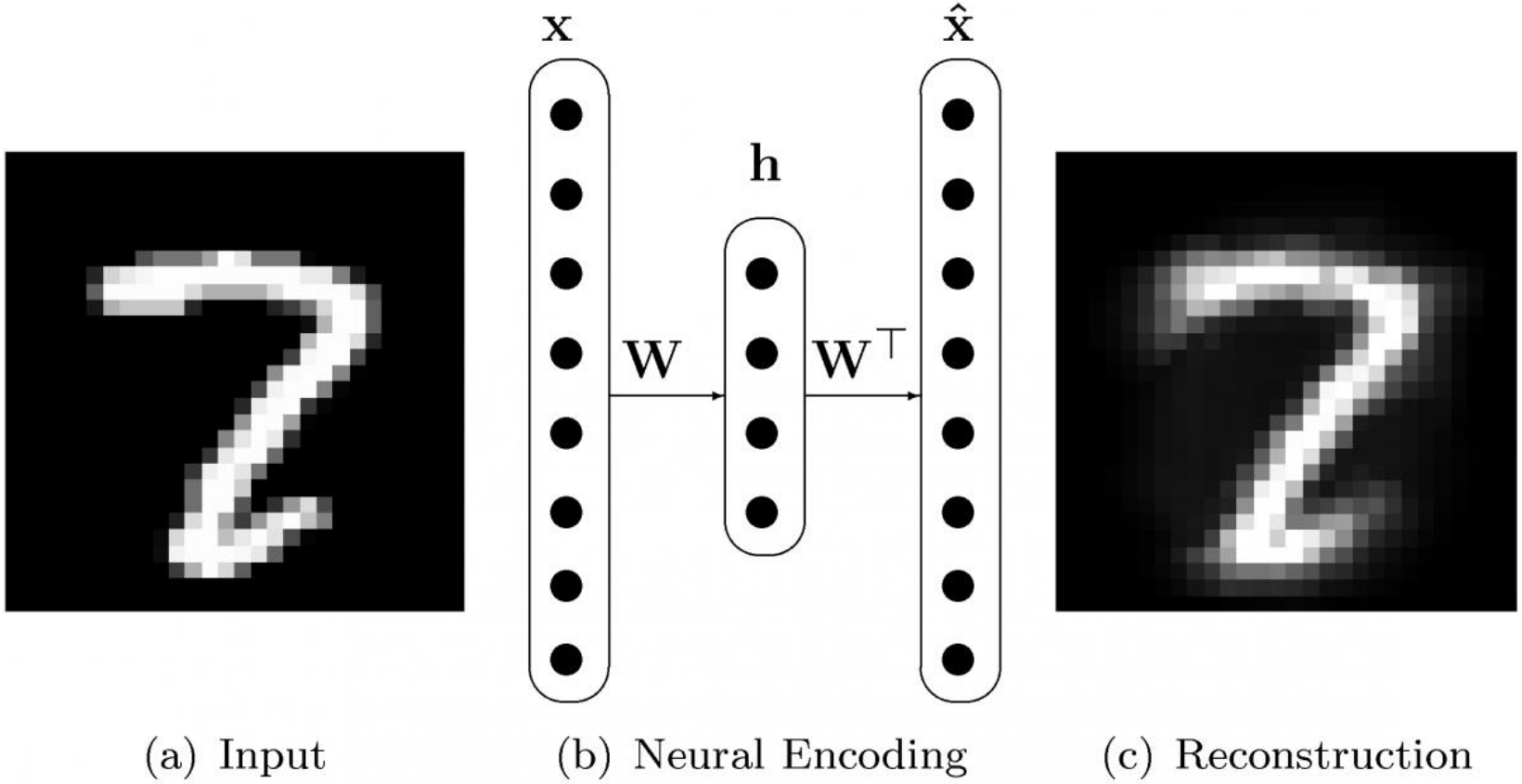
Auto-Encoder

- Auto-Encoder (Auto-Associative Neural Network)
  - ✓ Hidden layer with fewer nodes than the input layer must exist
    - Information is compressed in this layer



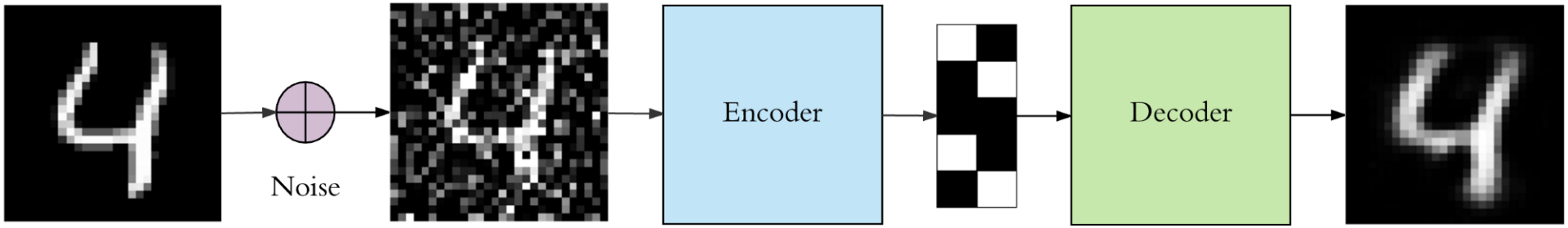
Auto-Encoder

- Auto-Encoder (Auto-Associative Neural Network) Example
  - ✓ AE learning the digit 2 → If the digit 5 is provided, it will not generate 5 (Loss will be large)



Denoising Auto-Encoder

- Limitations of NN including Auto-Encoder
  - ✓ Sensitive to small input perturbations
- Add the noise during training process to make the model more robust!



Original Image

Noisy Input

Code

Output

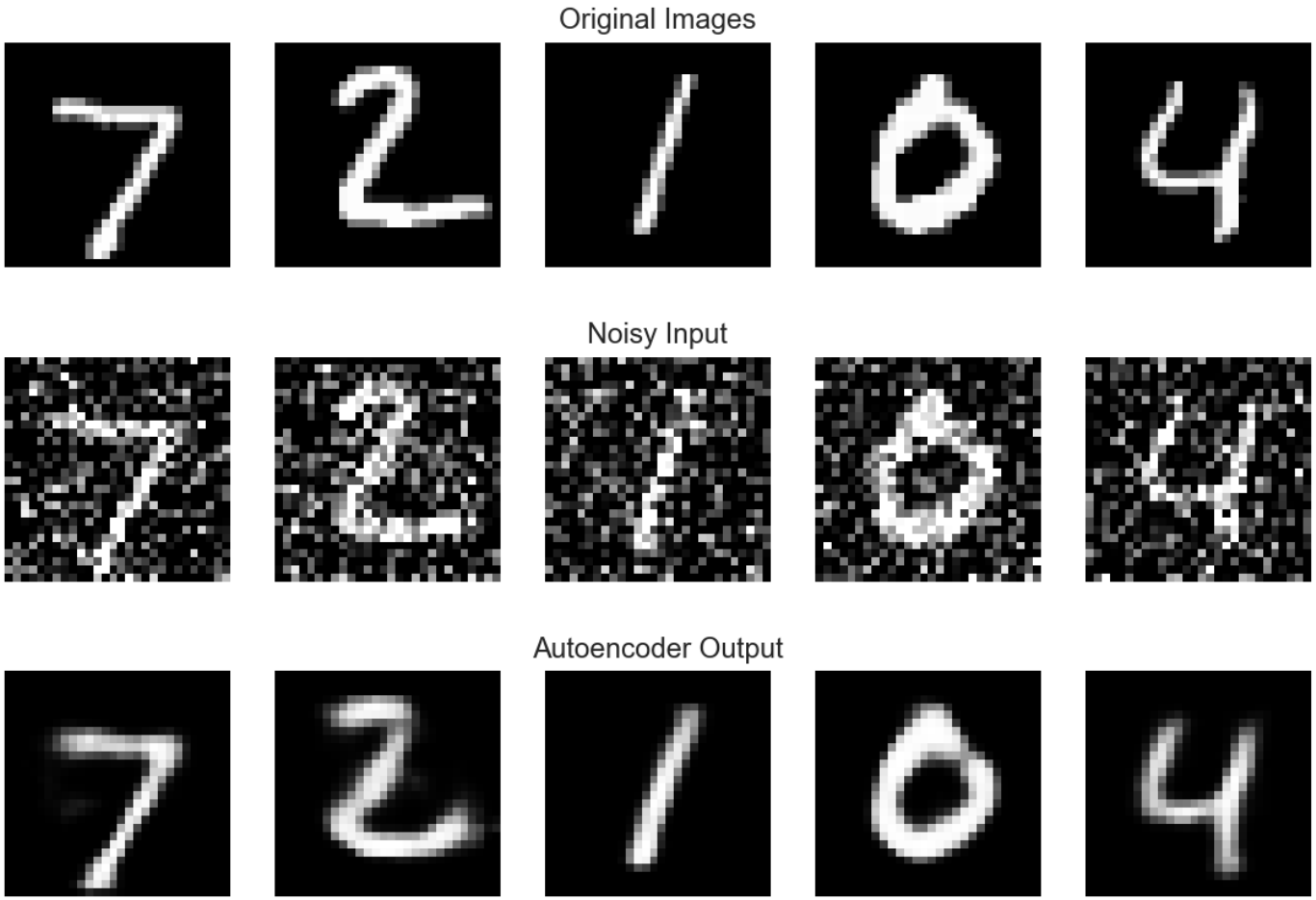
입력에 노이즈를 첨가해도

노이즈가 제거된 결과값이 나오도록

https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Denoising Auto-Encoder

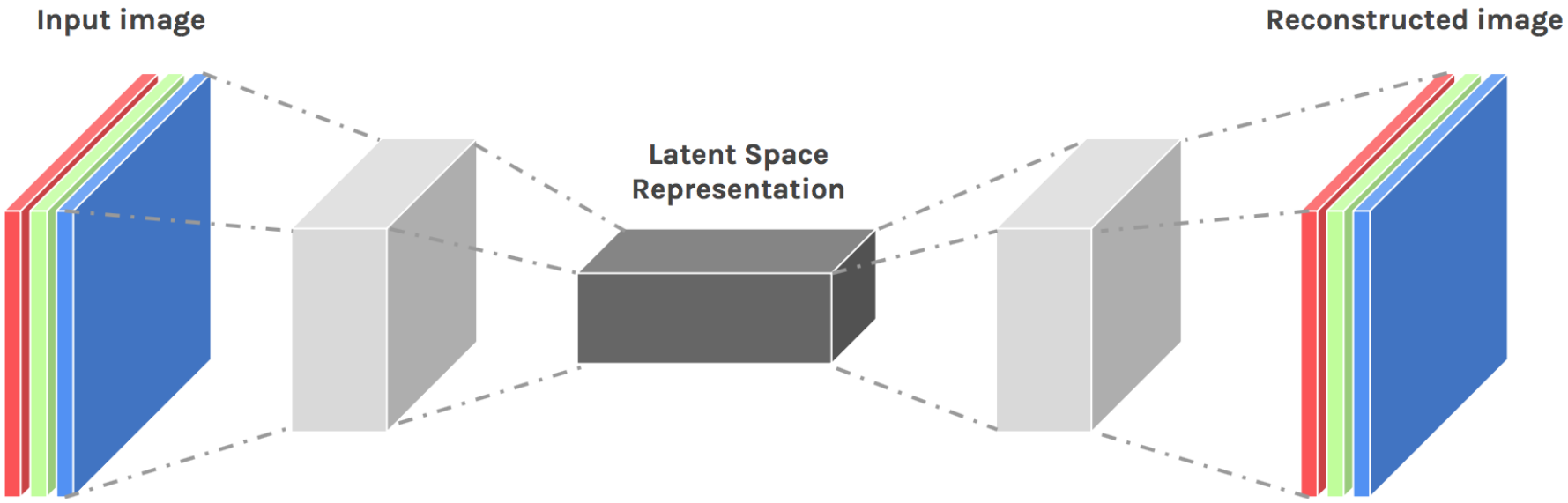
- How to add noise?
  - ✓ Random Gaussian noise is commonly used



https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Convolutional Auto-Encoder

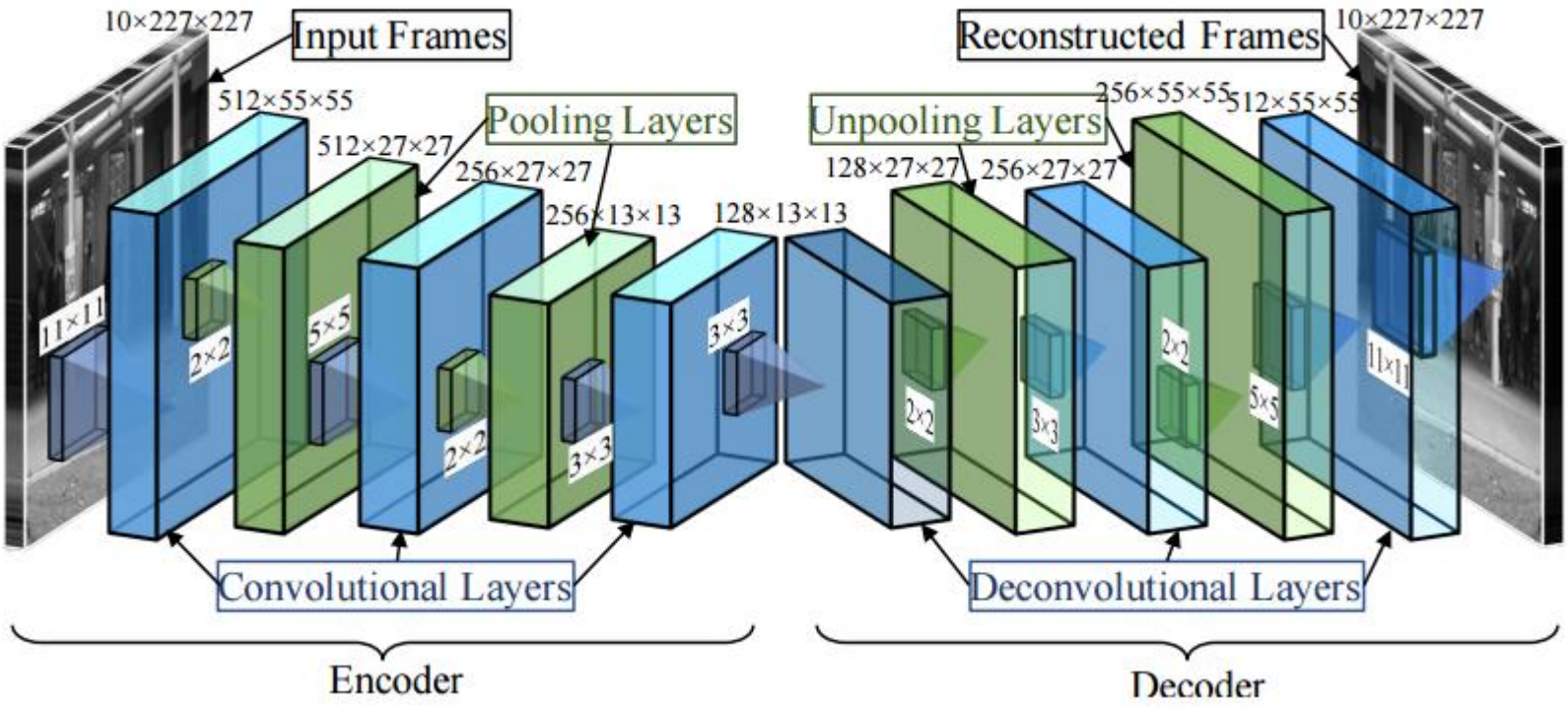
- Hand-written digit in the previous example use a vector as an input
  - ✓ 16 by 16 matrix is transformed into 256-dimensional vector
- CAE = Auto-Encoder that takes an image as the input and output



https://blog.manash.me/implementing-pca-feedforward-and-convolutional-autoencoders-and-using-it-for-image-reconstruction-8ee44198ea55

Convolutional Auto-Encoder

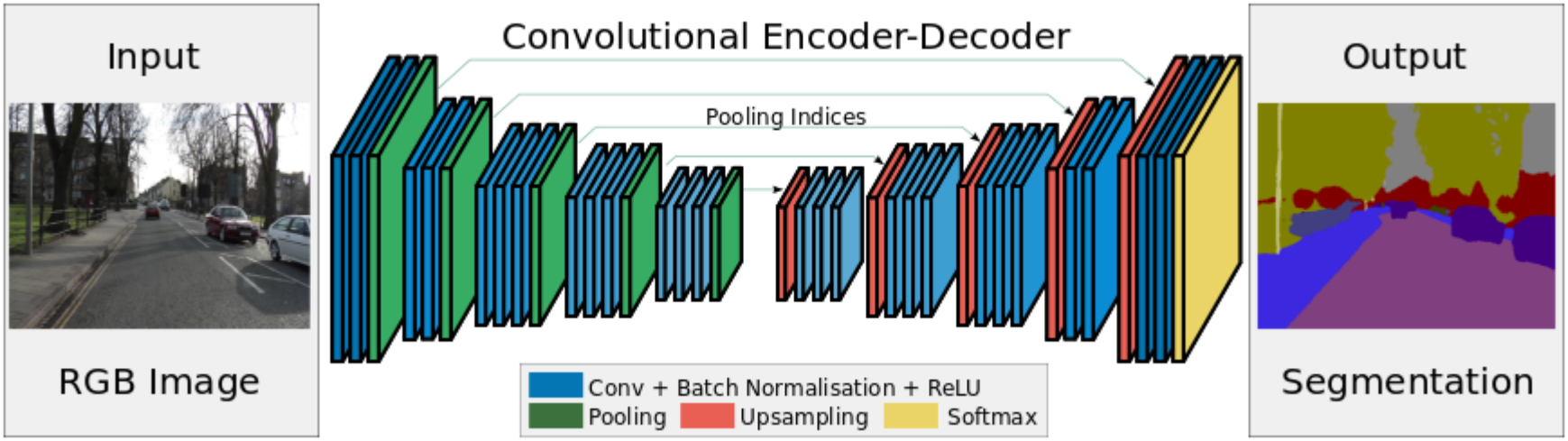
- CAE = Auto-Encoder that takes an image as the input and output



https://blog.manash.me/implementing-pca-feedforward-and-convolutional-autoencoders-and-using-it-for-image-reconstruction-8ee44198ea55

Convolutional Auto-Encoder

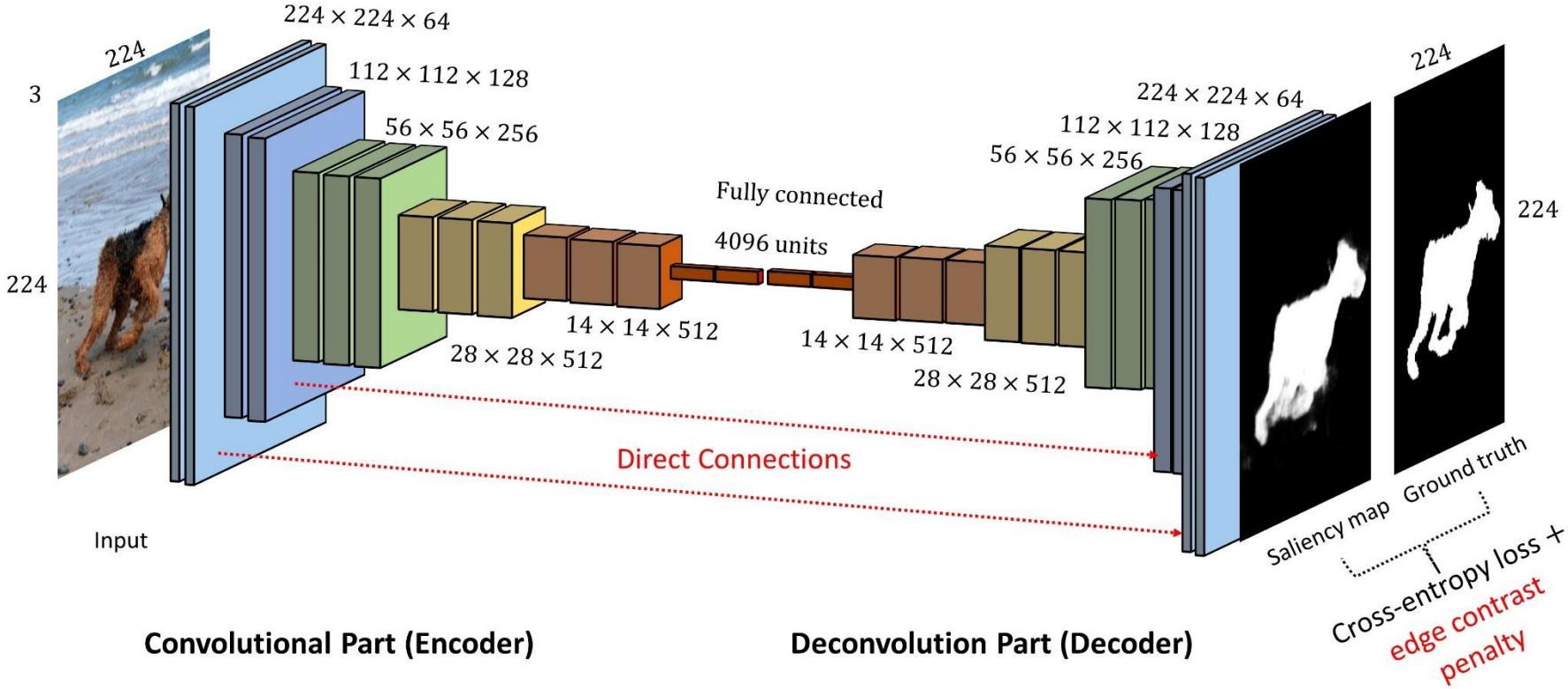
- It is not necessary to use CAE to reconstruct the original image
  - ✓ Image segmentation
    - Input: Image
    - Output: pixel-wise class (road, car, sky, etc.)



https://github.com/arahusky/Tensorflow-Segmentation

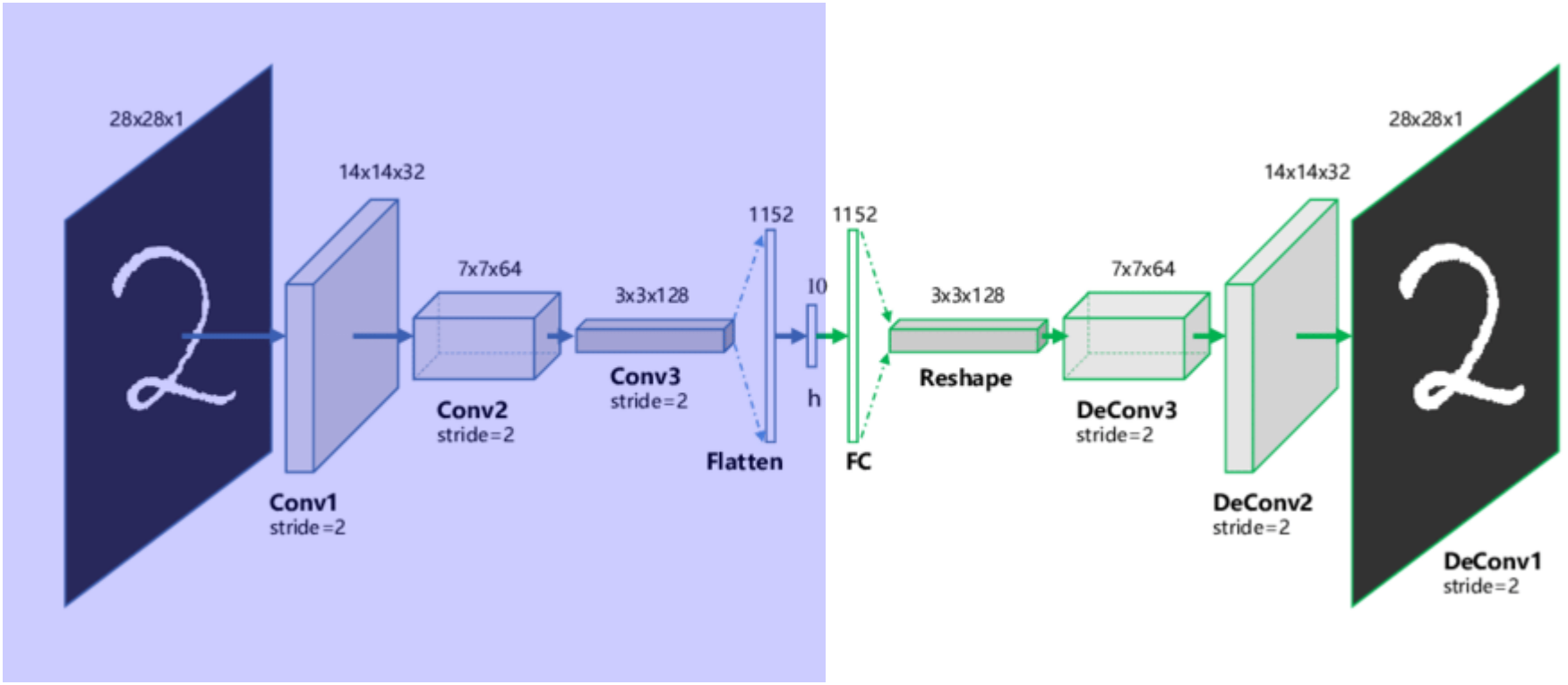
Convolutional Auto-Encoder

- It is not necessary to use CAE to reconstruct the original image
  - ✓ Saliency detection
    - Input: Image
    - Output: Most important area



Convolutional Auto-Encoder

- Issue: How to increase the size of feature map for training the decoder part?

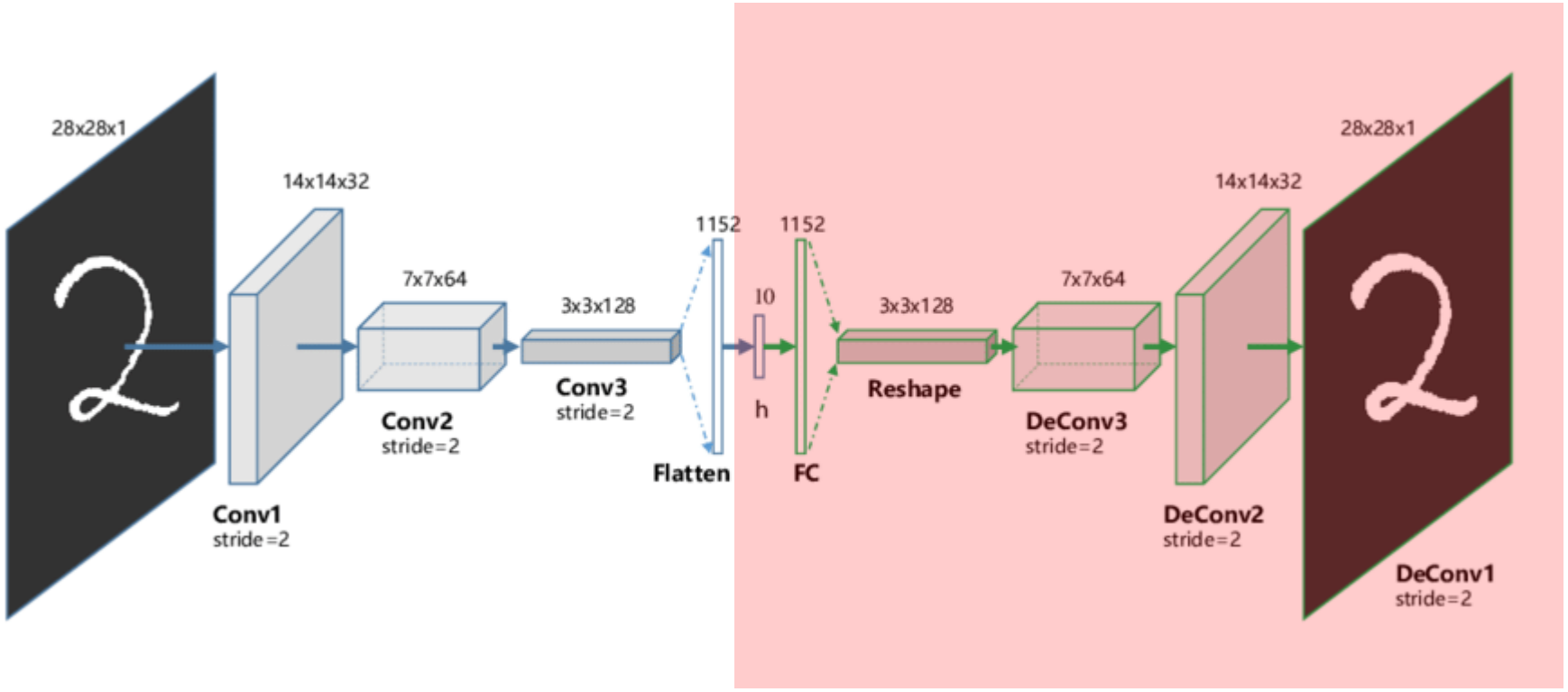


Encoder part: CNN forward path

https://www.researchgate.net/figure/The-structure-of-proposed-Convolutional-AutoEncoders-CAE-for-MNIST-In-the-middle-there_fig1_320658590

Convolutional Auto-Encoder

- Issue: How to increase the size of feature map for training the decoder part?

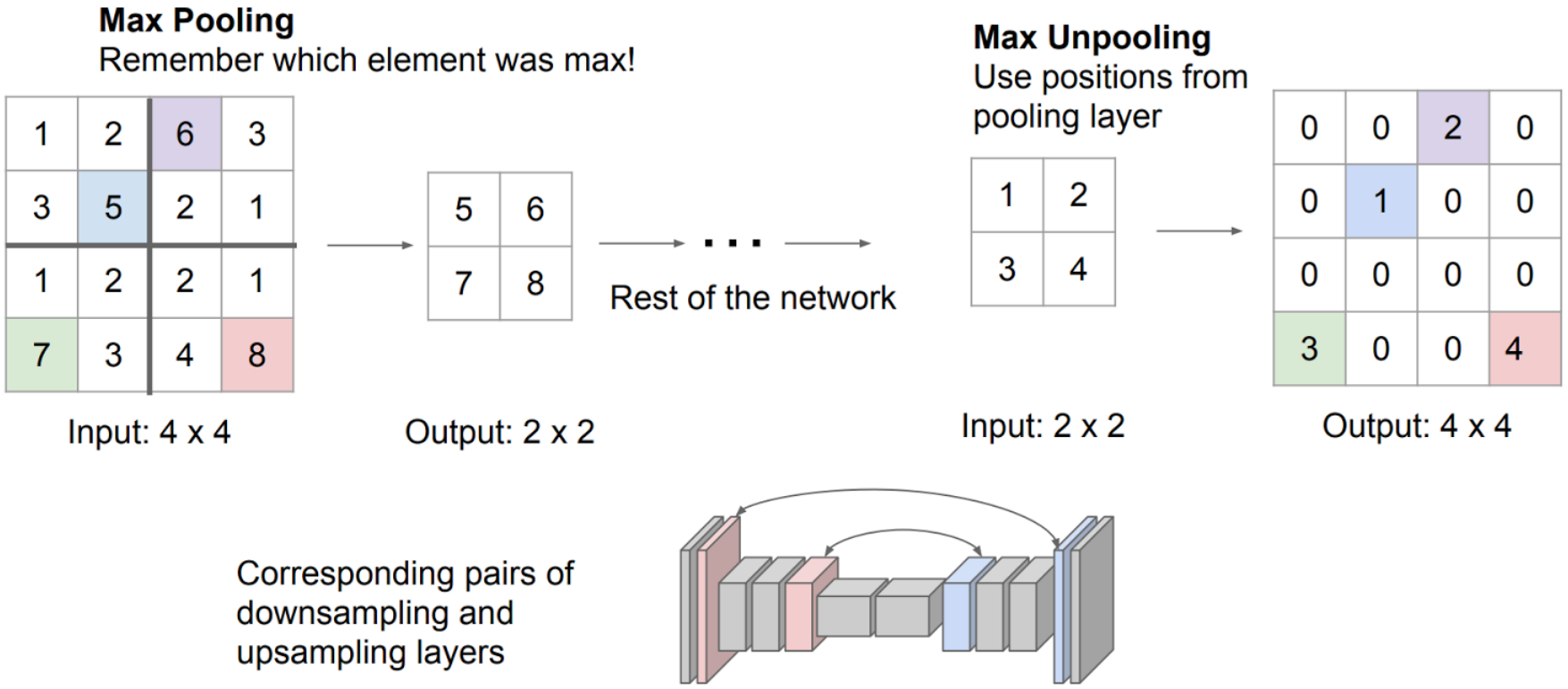


How to increase the feature map size?

https://www.researchgate.net/figure/The-structure-of-proposed-Convolutional-AutoEncoders-CAE-for-MNIST-In-the-middle-there_fig1_320658590

Convolutional Auto-Encoder

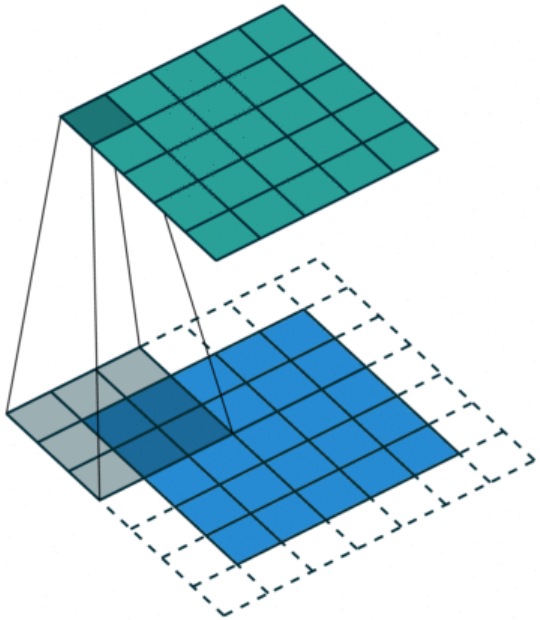
- Unpooling:
  - ✓ Remember the location information of max pooling



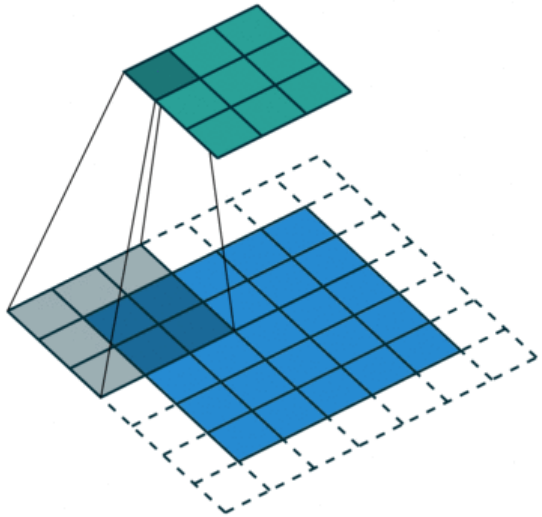
http://jihochoi.blogspot.com/2018/01/fully-convolutional-network-for.html

Convolutional Auto-Encoder

- Transpose convolution
  - ✓ Increase the feature map size using convolution-like operation
    - Convolution



- Feature map: 3 by 3
- Padding: 1
- Stride: 1



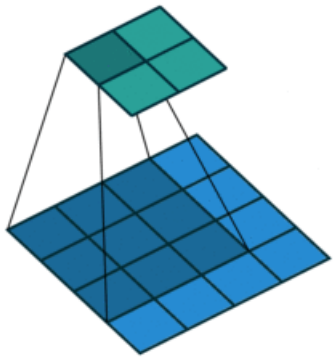
- Feature map: 3 by 3
- Padding: 1
- Stride: 2

http://www.cvc.uab.es/people/joans/slides_tensorflow/tensorflow_html/layers.html

Convolutional Auto-Encoder

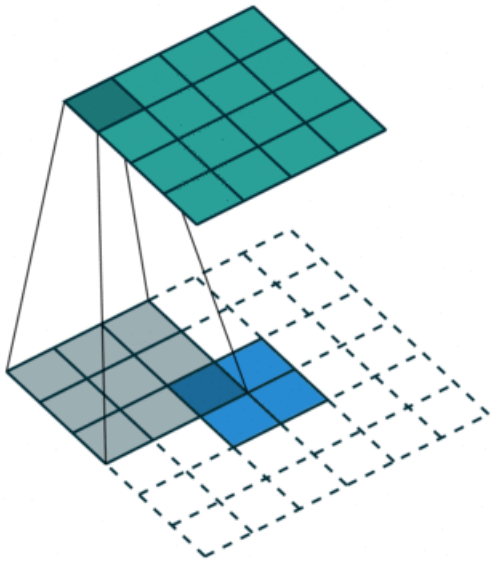
- Transpose convolution

Convolution



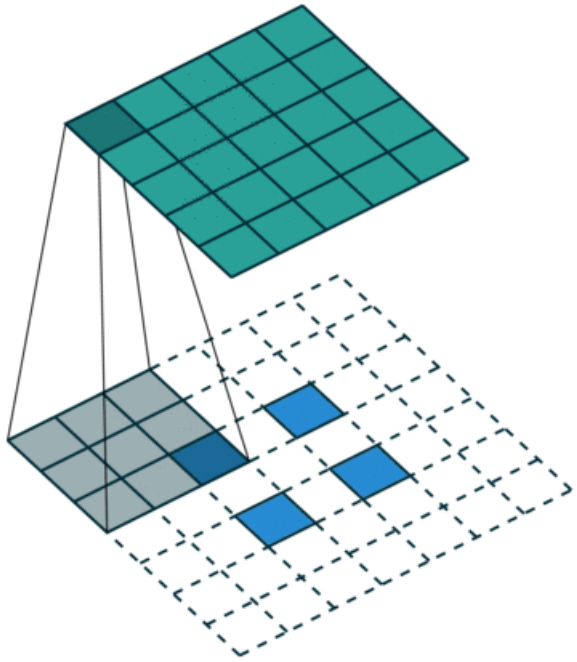
- Feature map: 3 by 3
- Padding: 0
- Stride: 1

Transpose Convolution



- Feature map: 3 by 3
- Padding: 0
- Stride: 1

Transpose Convolution

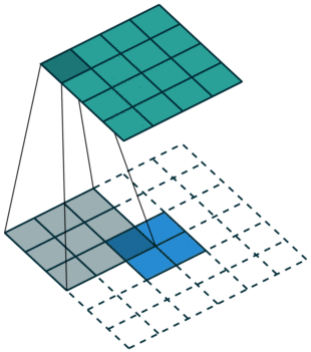


- Feature map: 3 by 3
- Padding: 0
- Stride: 2

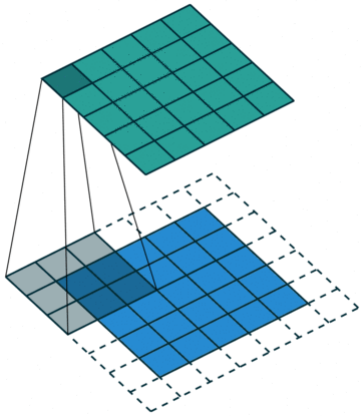
http://www.cvc.uab.es/people/joans/slides_tensorflow/tensorflow_html/layers.html

Convolutional Auto-Encoder

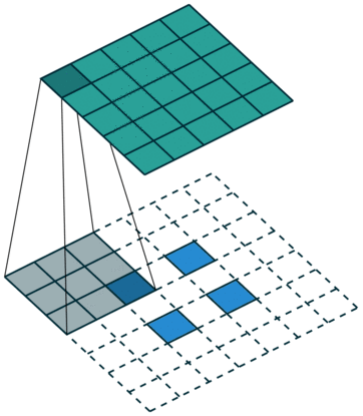
- (Caution) Padding in Transpose convolution
  - ✓ Padding = 1 in convolution: add pad (with 0 value) with 1 pixel to the feature map
  - ✓ Padding = 0 in transpose convolution: add pad (with 0 value) with (filter width (or height)-1) pixels to the feature map (add 2 pixels for 3 by 3 filter size)
  - ✓ Padding = 1 in transpose convolution: add pad (with 0 value) with (filter width (or height)-1)-1 pixels to the feature map (add 1 pixels for 3 by 3 filter size)



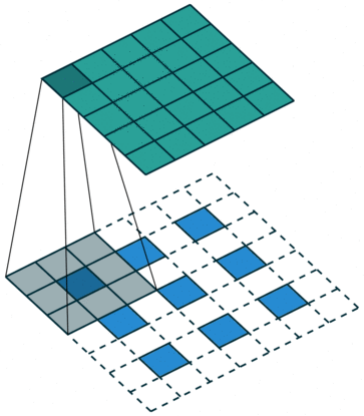
- Feature map: 3 by 3
- Padding: 0
- Stride: 0



- Feature map: 3 by 3
- Padding: 1
- Stride: 0



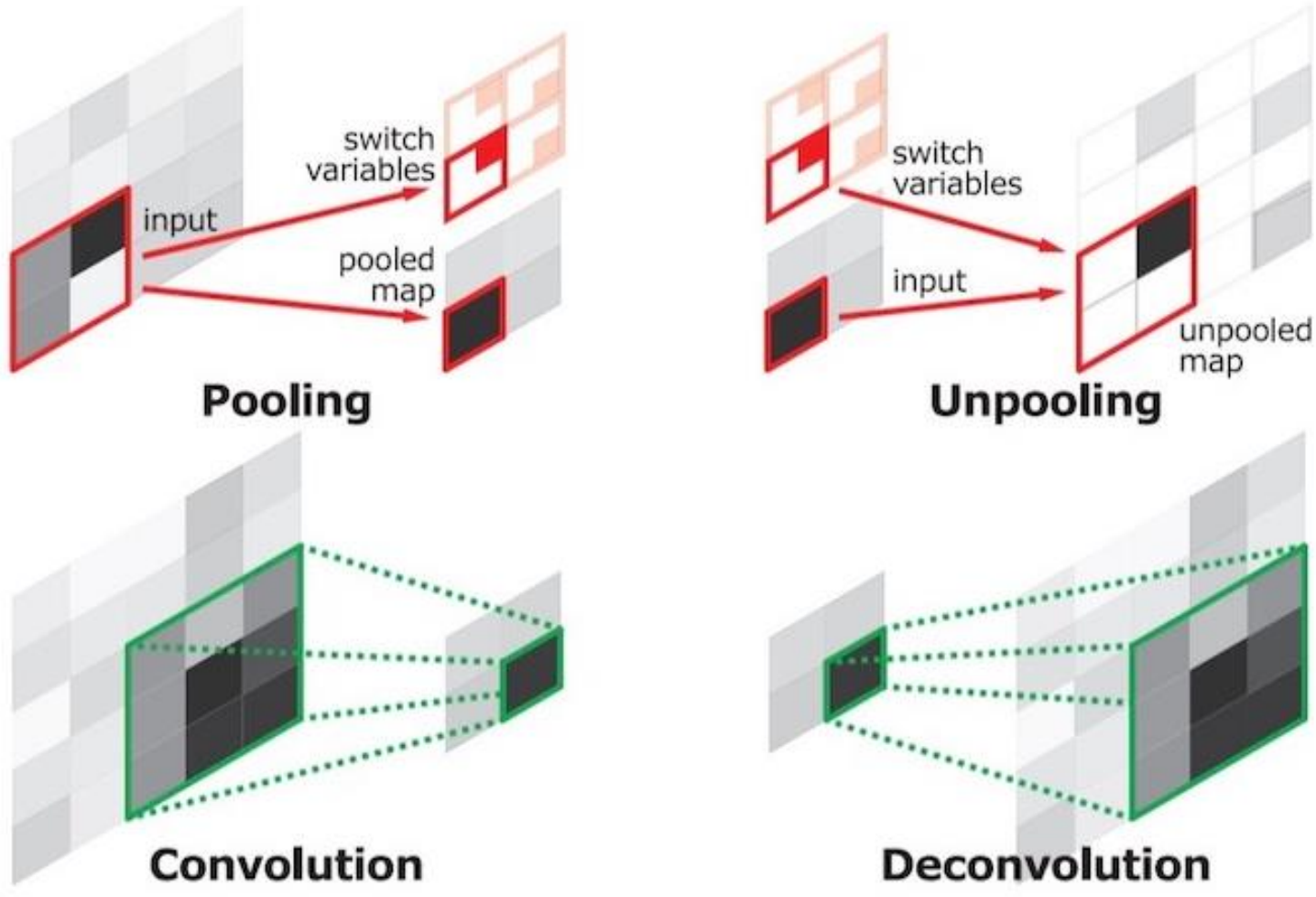
- Feature map: 3 by 3
- Padding: 0
- Stride: 1



- Feature map: 3 by 3
- Padding: 1
- Stride: 1

Convolutional Auto-Encoder

- Comparison between Transpose convolution and Unpooling



https://modulabs-biomedical.github.io/Learning_Deconvolution_Network_for_Semantic_Segmentation

RNN Auto-Encoder

- RNN Auto-Encoder
  - ✓ Input & Output: A sequence of words
  - ✓ Different from sequence to sequence (seq2seq) model
    - RNN-AE: Input and outputs are identical
    - Seq2seq: Takes sequences as input and output but they can be different
    - RNN-AE is a subset of seq2seq model

RNN-AE

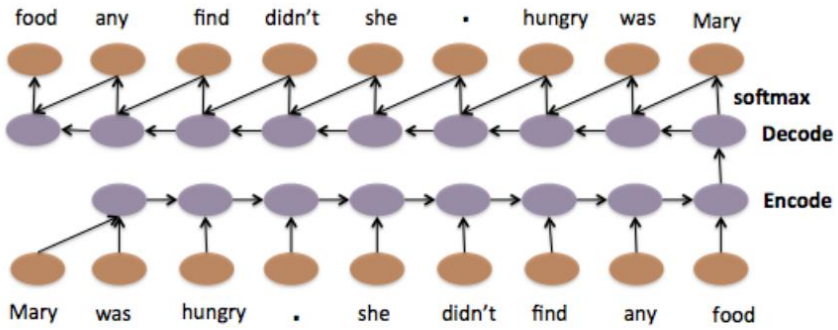
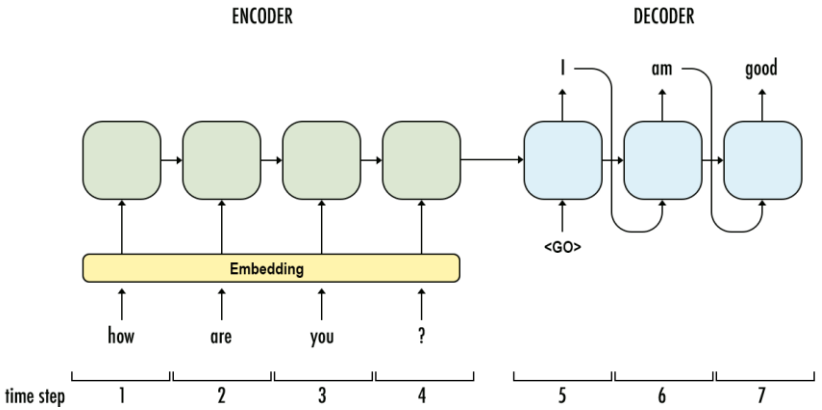


Figure 1: Standard Sequence to Sequence Model.

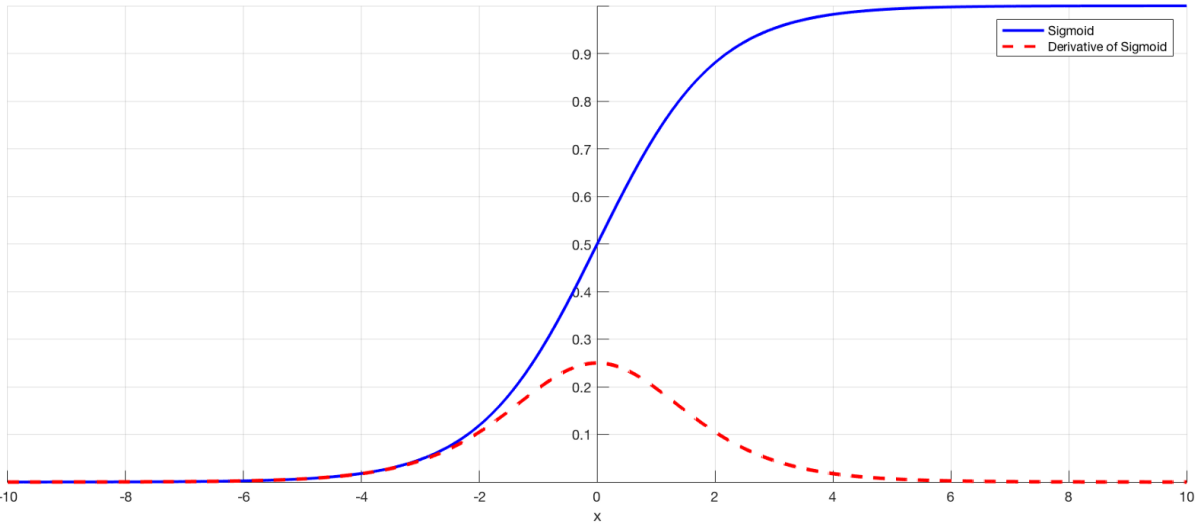
Seq2seq



AGENDA

Weight Initialization

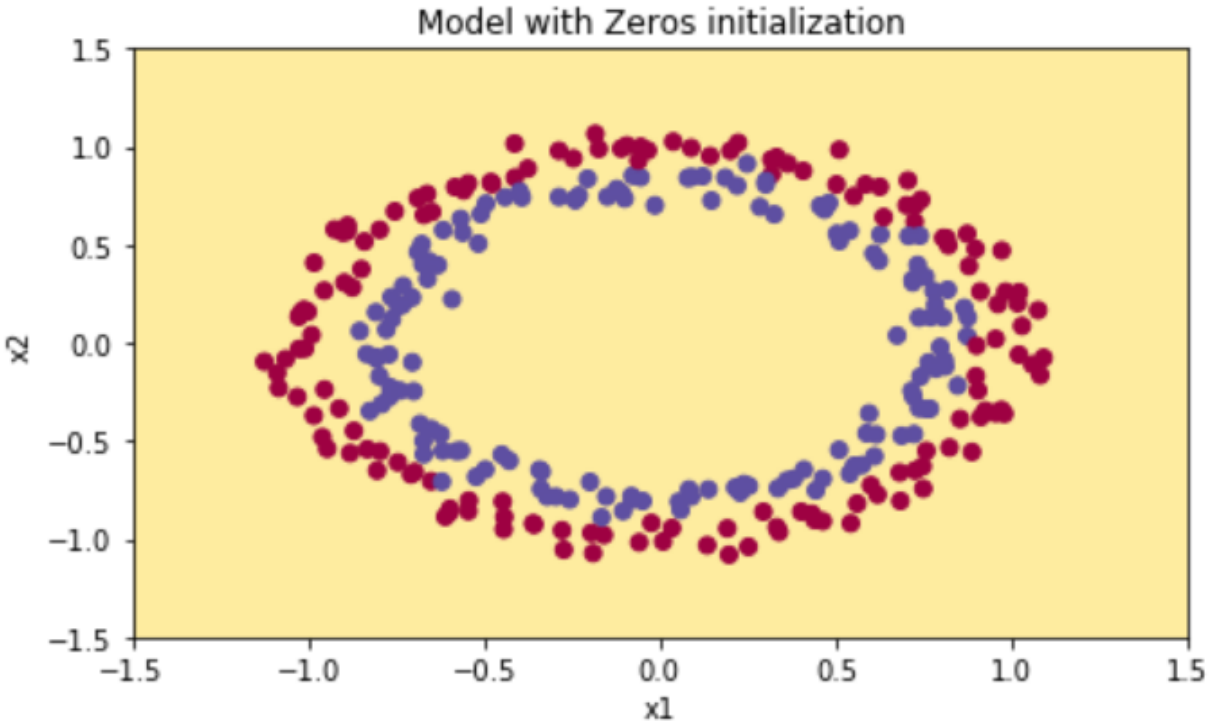
- Weight Initialization
  - ✓ How to initialize the weights?
    - We need outputs (regardless of whether they are good or not) to train the network
- Strategy 1: Set all weights to 0
  - ✓ Zero initialization: set all weights and biases to 0
  - ✓ Gradients of all hidden units in the same layer become identical → Linear model



https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

Weight Initialization

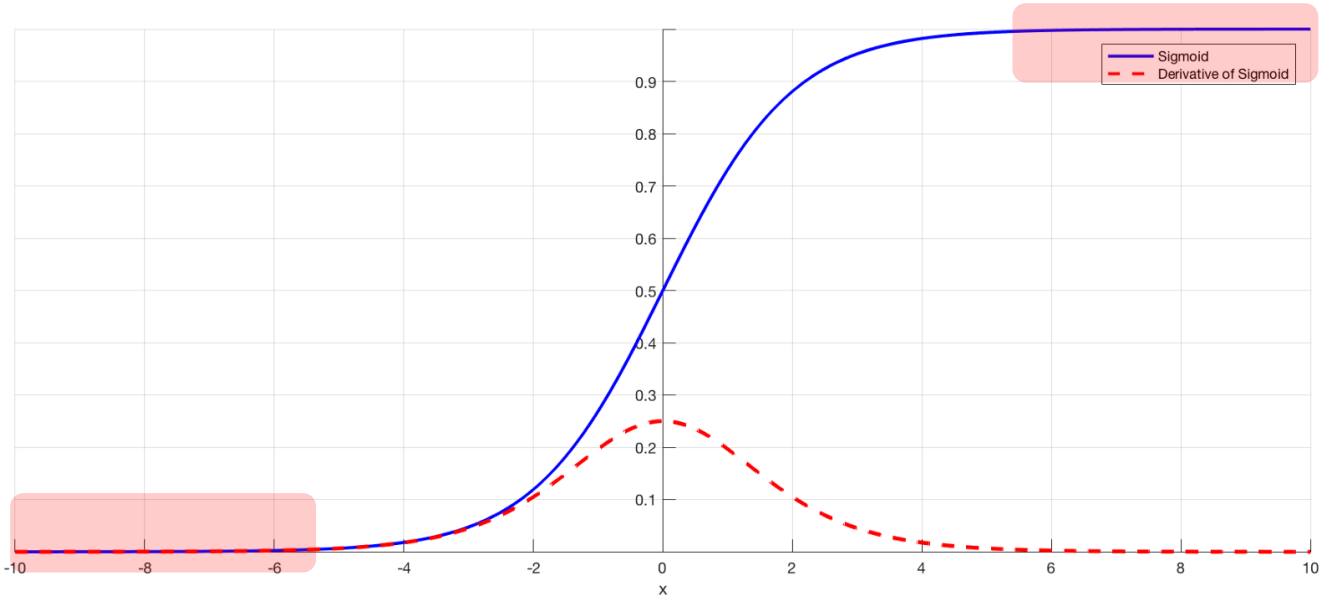
- Strategy 1: Set all weights to 0
  - ✓ For the dataset below
  - ✓ iteration = 15,000, loss = 0.693, accuracy = 50%



https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

Weight Initialization

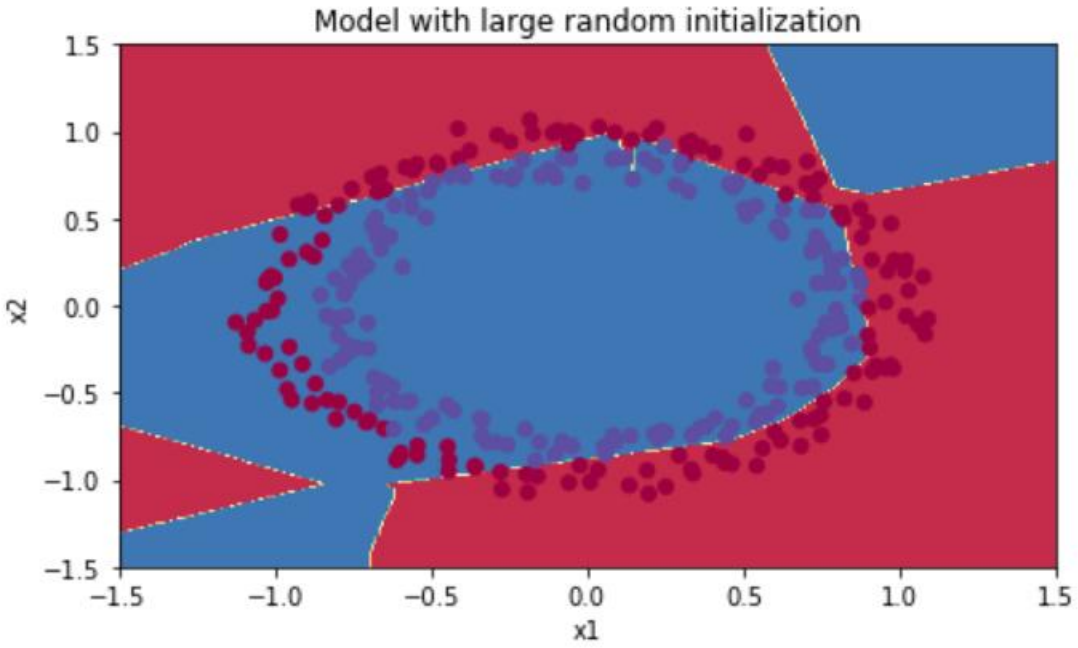
- Strategy 2: Sample random numbers from a Normal (Uniform) distribution to make some variations
  - ✓ Mean: 0, variance: a certain value…
  - ✓ Limitation: The magnitudes of weights depend on the variance
    - If the W is too large/small, Wx+b becomes large/small
    - Gradient of sigmoid activation function becomes close to 0 → Gradient vanishing problem



https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

Weight Initialization

- Strategy 2: Sample random numbers from a Normal (Uniform) distribution to make some variations
  - ✓ Same dataset as in Strategy 1
  - ✓ Iteration = 15,000, loss = 0.383, accuracy = 86%
  - ✓ Better than Zero initialization but needs to be improved



https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

Weight Initialization

- Strategy 3: Set an appropriate variance of Normal Distribution
  - ✓ Xavier initialization: Tanh activation function
  - ✓ Make the variances of input and the output of the L^{th} layer the same
  - ✓ It can be proved that the variance is 1/[No. of hidden nodes in the $(L-1)^{th}$ layer]
  - ✓ Weight sampling distribution
    - Consider the forward path only

$$N\left(0, \frac{1}{size^{(L-1)}}\right) \; or \; \left[-\sqrt{\frac{3}{size^{(L-1)}}}, \sqrt{\frac{3}{size^{(L-1)}}}\right]$$

    - Consider both forward and backward paths

$$N\left(0, \frac{2}{size^{(L-1)} + size^{(L)}}\right) \; or \; \left[-\sqrt{\frac{6}{size}}, \sqrt{\frac{6}{size}}\right]$$

$$(size = size^{(L-1)} + size^{(L)})$$

https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

Weight Initialization

- Strategy 3: Set an appropriate variance of Normal Distribution
  - ✓ He initialization: LeLU activation function
  - ✓ Make the variances of input and the output of the L^{th} layer the same
  - ✓ It can be proved that the variance is

$$\frac{2}{size^{(L-1)}}$$

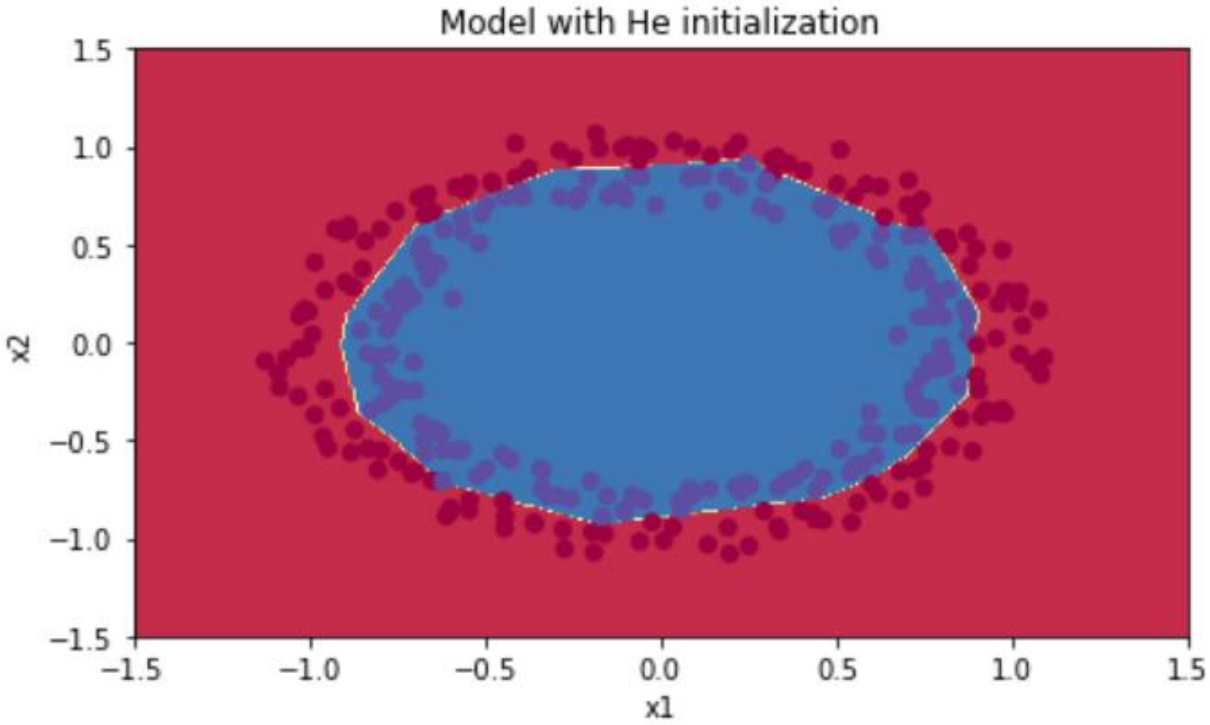
  - ✓ Weight sampling distribution
    - The more units in the previous layer, the smaller values are initialized

$$N\left(0, \frac{2}{size^{(L-1)}}\right) \; or \; \left[-\sqrt{\frac{6}{size^{(L-1)}}}, \sqrt{\frac{6}{size^{(L-1)}}}\right]$$

https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

Weight Initialization

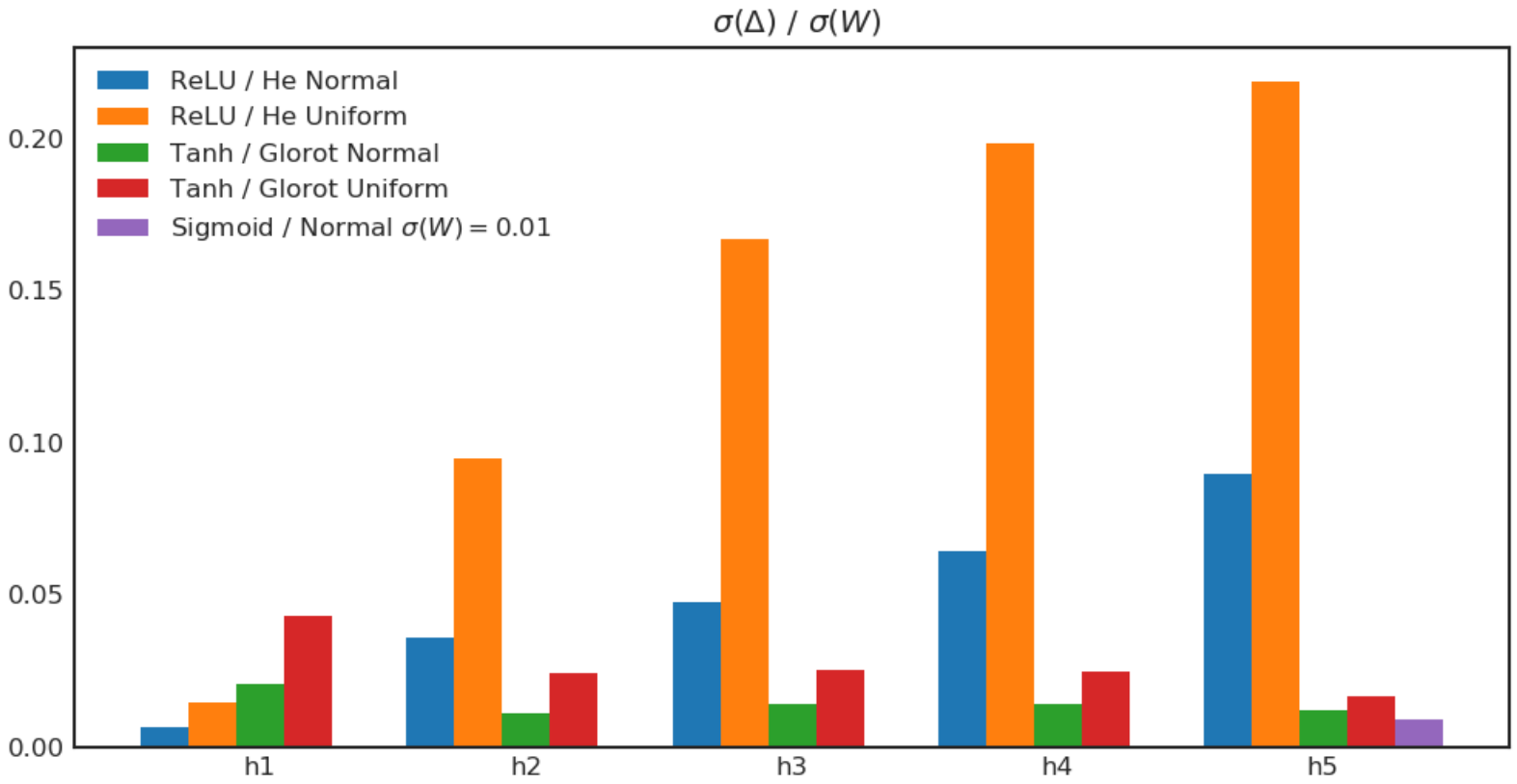
- Strategy 3: Set an appropriate variance of Normal Distribution
  - ✓ Same dataset as in Strategy 1 & 2
  - ✓ Iteration = 15,000, loss = 0.074, accuracy = 96%
  - ✓ Significant improvement although only the size of layers is taken into account



https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

Weight Initialization

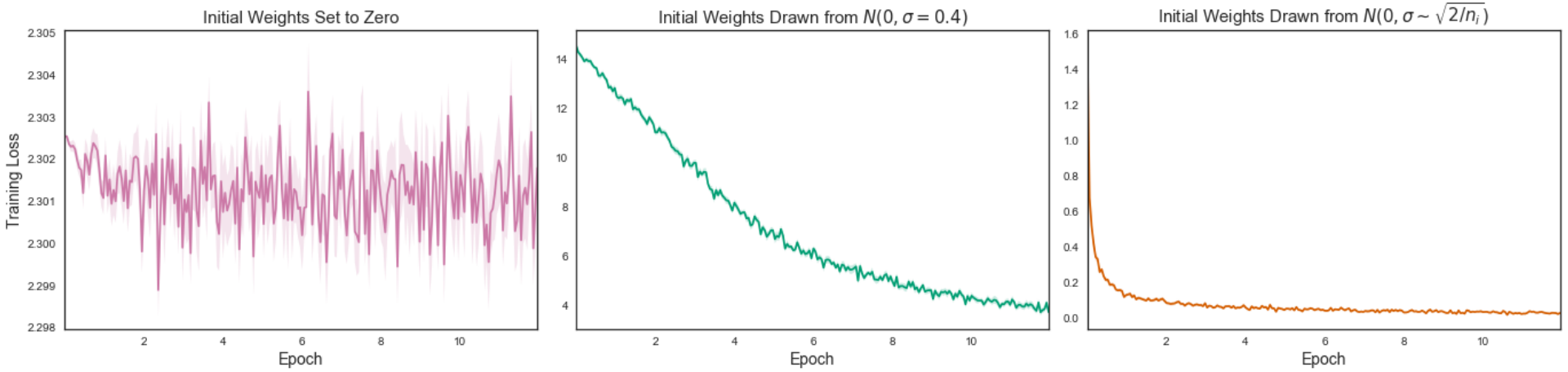
- Average gradient w.r.t. weight initialization methods and random distributions



https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404

Weight Initialization

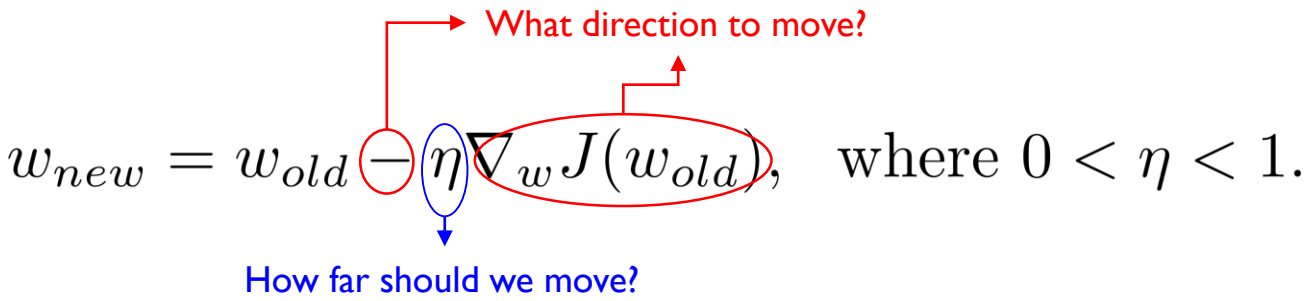
- Training loss w.r.t. weight initialization methods
  - ✓ left: Zero initialization
  - ✓ middle: Random initialization with a fixed variance
  - ✓ right: Xavier initialization



https://intoli.com/blog/neural-network-initialization/

Gradient Descent Variations

- Gradient Descent revisited
  - ✓ Batch gradient descent: use the average gradient of all training examples



$$w_{new} = w_{old} - \eta \nabla_w J(w_{old}), \quad \text{where } 0 < \eta < 1.$$

  - ✓ Stochastic gradient descent: use the gradient of an indivisual example

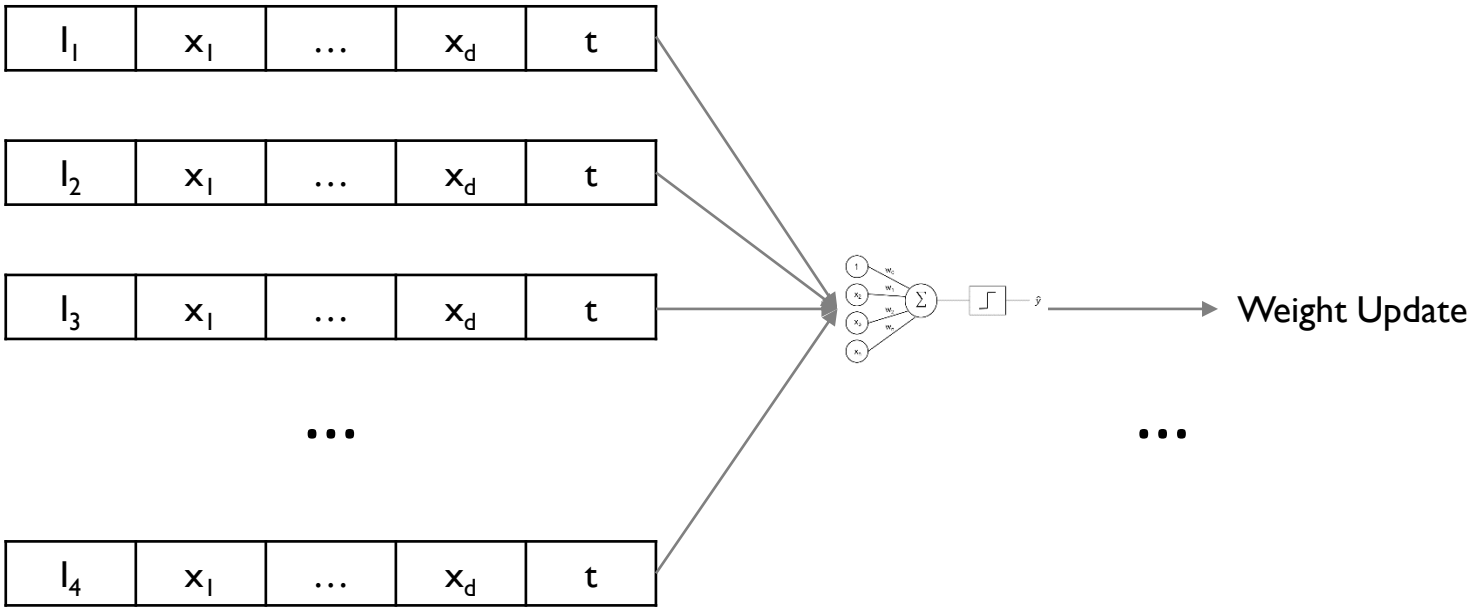
$$w_{new} = w_{old} - \eta \nabla_w J(w_{old} | \mathbf{x}^{(i)}, y^{(i)})$$

  - ✓ Mini-batch gradient descent: use the average gradient of n mini-batch examples

$$w_{new} = w_{old} - \eta \nabla_w J(w_{old} | \mathbf{x}^{(i:i+n)}, y^{(i:i+n)})$$

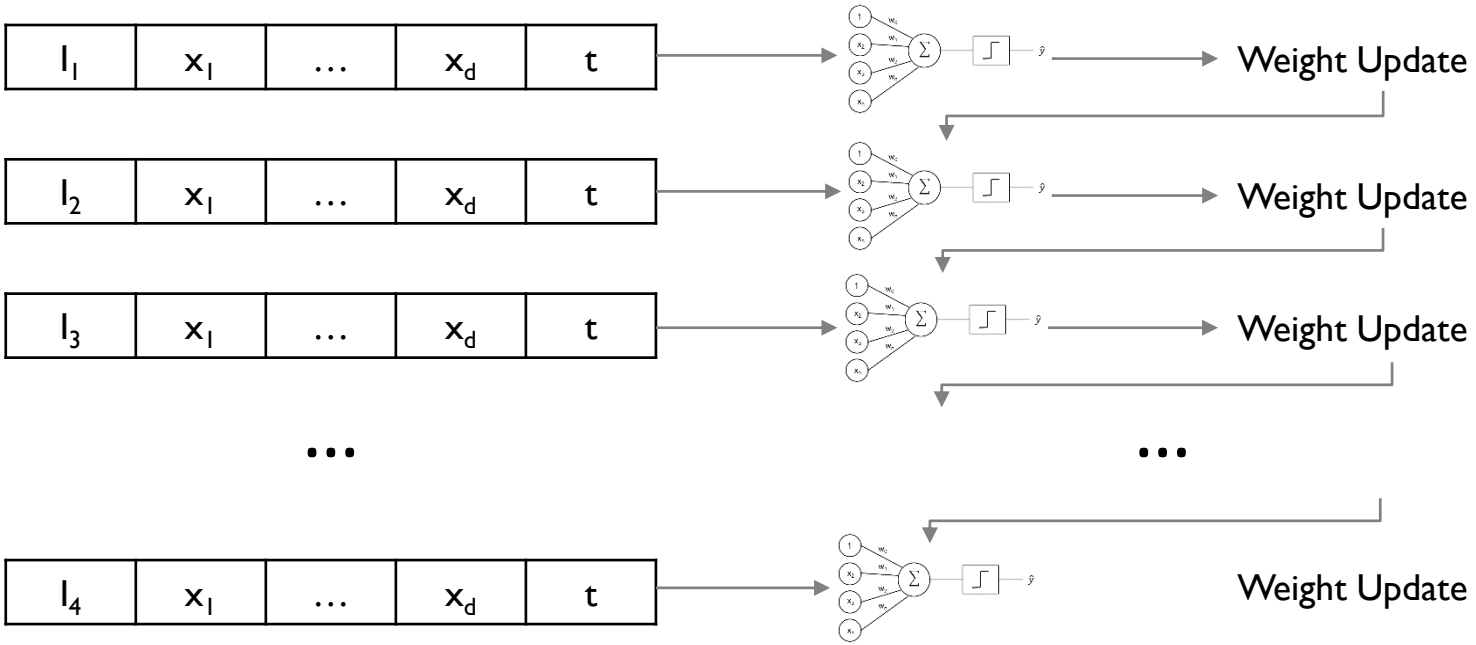
Gradient Descent Variations

- Batch Gradient Descent (BGD)
  - ✓ Fix the network weights, compute the cost function using all training examples, compute the gradient, and update the weights



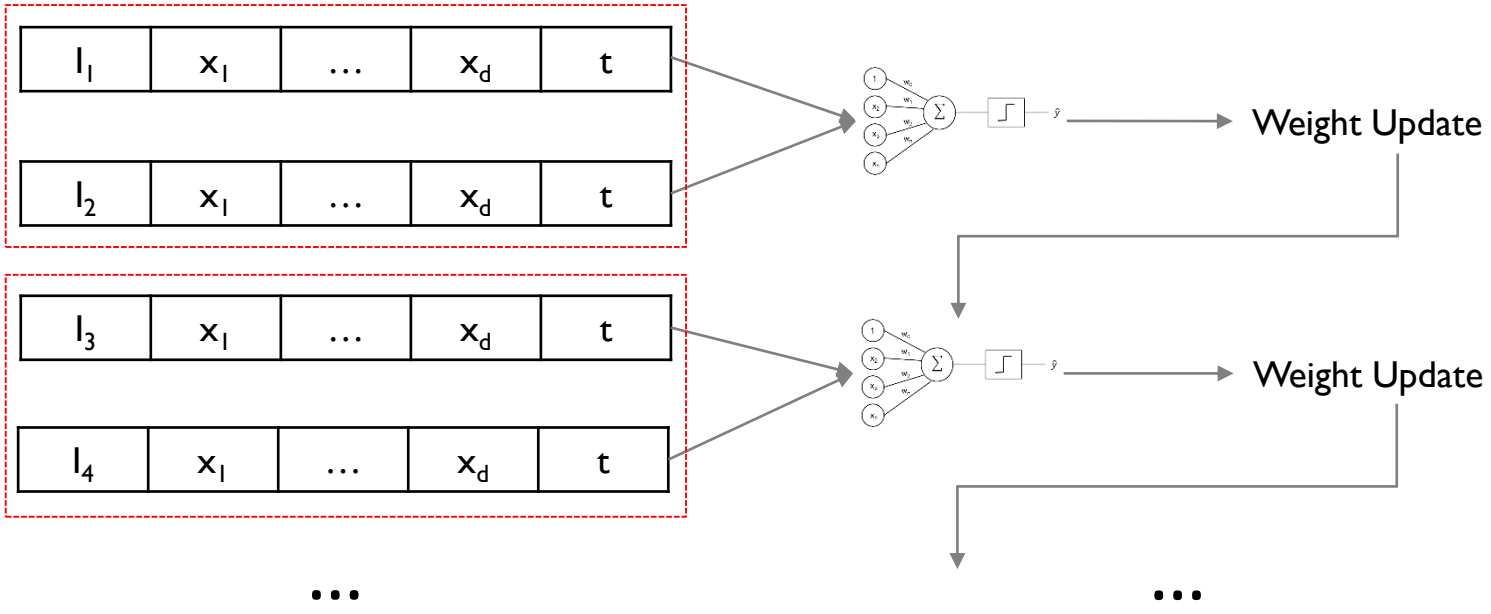
Gradient Descent Variations

- Stochastic Gradient Descent (SGD)
  - ✓ Compute the loss function for an individual training example and update the gradients



Gradient Descent Variations

- Mini-Batch Gradient Descent
  - ✓ A strategy between SGD and BGD
  - ✓ Construct a mini-batch with n examples from N training examples and compute the gradient using the n examples (when the mini-batch size is set to 2)

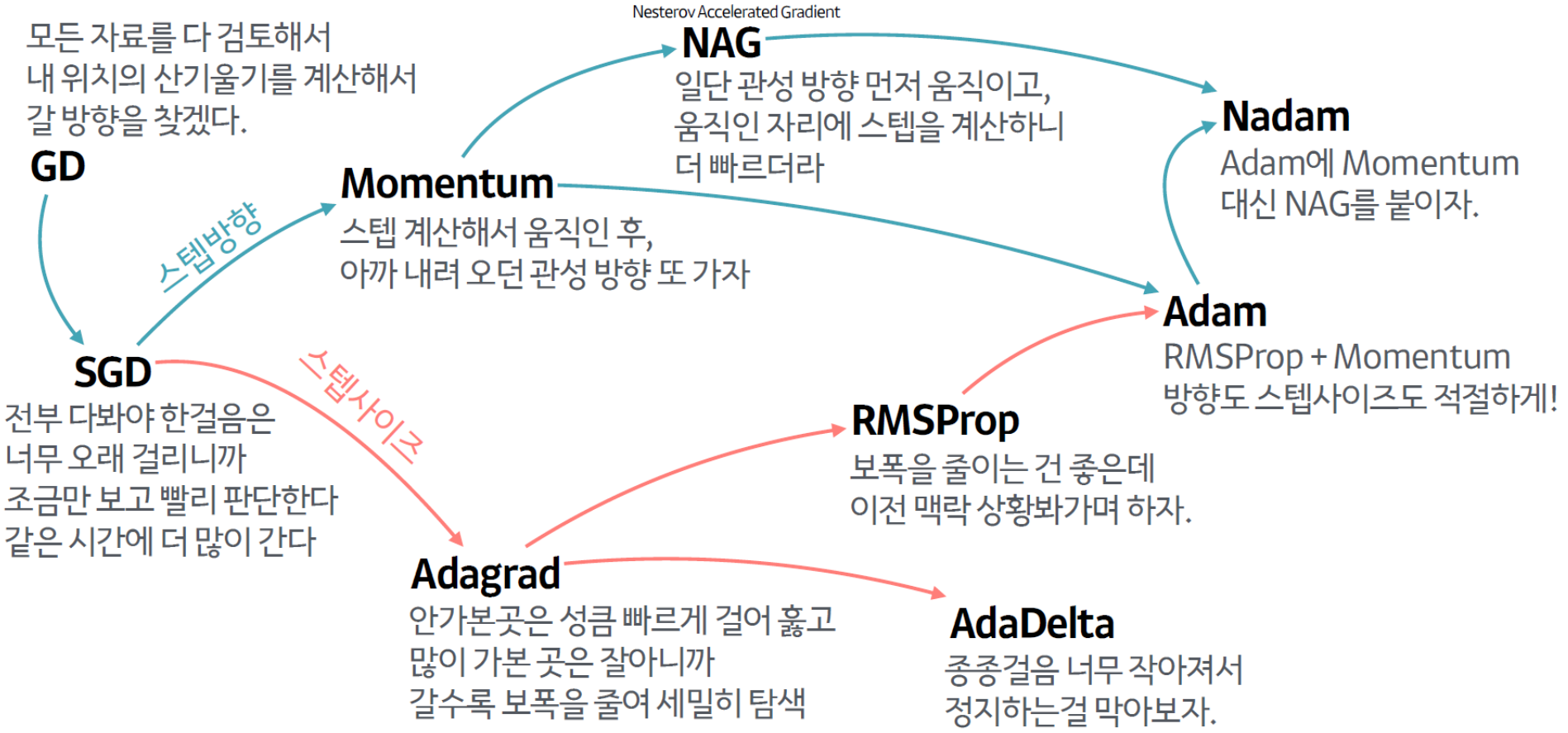


Gradient Descent Variations

- Challenges
  - ✓ Difficult to set an appropriate learning rate η
    - If it is too large, the solution either diverges or fluctuates around the optimum
    - If it is too small, convergence speed is too slow
  - ✓ Learning rate scheduling is also difficult
    - It is obvious that the learning rate should be large in early stages and should be decreased thereafter
    - How much and how frequently?
  - ✓ It can be also problematic that a single learning rate is used for all weights
    - The frequency of features is different
  - ✓ Non-convex objective function → numerical suboptimal local minima
    - Empirically, SGD cannot escape from the saddle points

Gradient Descent Variations

- Variations of Gradient Descent



Gradient Descent Variations

- Momentum

Consider the historical direction

$$v_t = \gamma v_{t-1} + \eta \nabla_w J(w_t)$$

$$w_t = w_{t+1} - v_t$$

Image 2: SGD without momentum

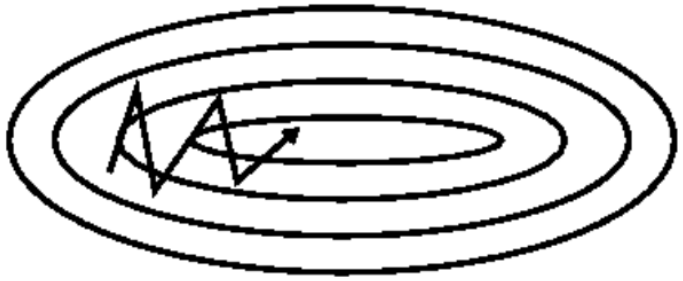


Image 3: SGD with momentum

http://ruder.io/optimizing-gradient-descent/index.html#challenges

Gradient Descent Variations

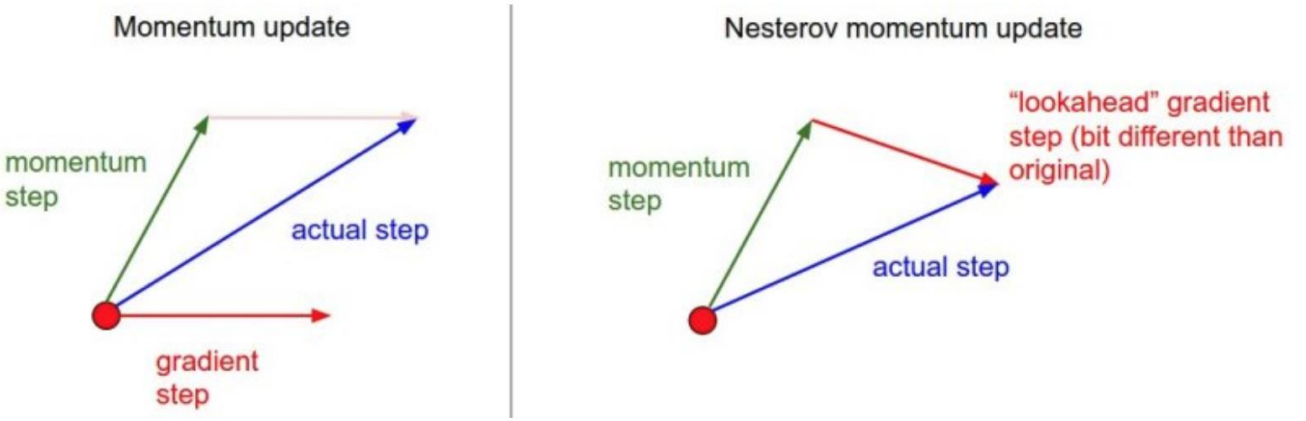
- Nesterov Accelerated Gradient (NAG)

Move w first using the previous gradient

$$v_t = \gamma v_{t-1} + \eta \nabla_w J(w_t - \gamma v_{t-1})$$

$$w_{t+1} = w_t - v_t$$

Compute the gradient



Nesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position.

http://cs231n.github.io/neural-networks-3/

Gradient Descent Variations

- Adagrad: Adapts the learning rate to the parameters
  - ✓ Gradient of weight i at time t

$$g_{t,i} = \nabla_w J(w_t - \gamma v_{t-1})$$

  - ✓ Consider the cumulative movements when computing the weight at time t+1

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

  - ✓ $G_{t,ii} \in R^{d \times d}$ diagonal matrix: cumulative squared gradient of w_i
    - Empirical analysis recommends the square root of G
  - ✓ Matrix-vector form

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

Gradient Descent Variations

- Adadelta
  - ✓ Adagrad tends to aggressively and monotonically decrease since the all past movements are recorded
  - ✓ Adadelta considers the recent gradient changes (learning rate can be increased if a weight has not been updated for a while)
  - ✓ Running average

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

  - ✓ Adadelta vs. Adagrad

Adadelta

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

과거 gradient의 제곱을 decaying하면서 누적

Adagrad

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

과거 gradient의 제곱을 전부 누적

http://ruder.io/optimizing-gradient-descent/index.html#challenges

Gradient Descent Variations

- RMSProp
  - ✓ Introduced by lecture slide by Geoffrey Hinton
  - ✓ Set the gamma to 0.9 in Adadelta (Independently proposed from Adadelta)

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

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

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{RMS[g]_t + \epsilon}} \odot g_t$$

http://ruder.io/optimizing-gradient-descent/index.html#challenges

Gradient Descent Variations

- Adam
  - ✓ Adaptive Moment Estimation
    - Actively adjust the learning rate such as Adadelta or RMSProp,
    - Consider the momentum of gradient
  - ✓ Moment term of Gradient: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  - ✓ Cumulative change of gradient: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  - ✓ Observation: If m_0 and v_0 are set to 0, they tend to be 0 during training
    - Solution: bias-corrected first and second moment 사용

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^2}$$

  - ✓ Weight update rule

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \odot \hat{m}_t$$

http://ruder.io/optimizing-gradient-descent/index.html#challenges

Regularization

- Cost function in machine learning

$$\text{Cost function} = \boxed{\text{Loss function}} + \text{Regularization term}$$

Fit the training data as much as possible

✓ Loss function

- Mean squared error for regression tasks
- Cross Entropy (Negative log-likelihood) for classification tasks
- KL-Divergence for Generative Adversarial Network (GAN)

Regularization

- Cost function in machine learning

$$\text{Cost function} = \text{Loss function} + \boxed{\text{Regularization term}}$$

Avoid overfitting
(Do not memorize them all!)

✓ Regularization term

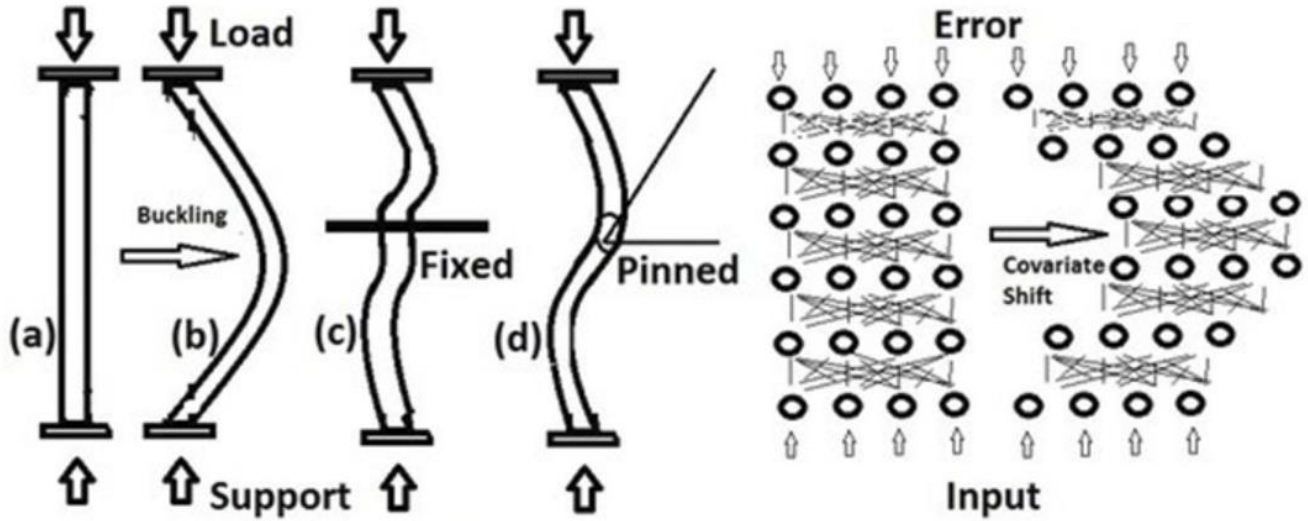
$$L_1 \text{ regularization} : \frac{\lambda}{2m} \sum_i |w_i|$$

$$L_2 \text{ regularization} : \frac{\lambda}{2m} \sum_i w_i^2$$

- lambda: hyperparameter, m: number of training examples

Batch Normalization

- Motivation
  - ✓ Reason of unstable training: Internal Covariance Shift
    - Change of input distributions of different layers in the network



https://m.blog.naver.com/laonple/220808903260

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Batch Normalization

- Initial solution: Whitening
  - ✓ Remove the correlation of input features, set the variance to 1
  - ✓ Problem: Computational cost due the covariance matrix and its inverse matrix
  - ✓ Effect of some parameters (especially bias term) decrease
- Batch Normalization (Ioffe & Szegedy, 2015)
  - ✓ Assumption: features are uncorrelated
  - ✓ For each feature, do the normalize using the mean and variance
  - ✓ If the mean and variance are fixed to 0 and 1, it will remove the non-linearity of activation function
    - Sigmoid has high linearity around 0
  - ✓ Add scale factor (gamma) and shift factor (beta) to the normalized values and learn them during the back-propagation process
  - ✓ Compute the mean and variance for each Mini-batch

https://shuuki4.wordpress.com/2016/01/13/batch-normalization-%EC%84%A4%EB%AA%85-%EB%B0%8F-%EA%B5%AC%ED%98%84/

Batch Normalization

- Batch Normalization: Algorithm

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

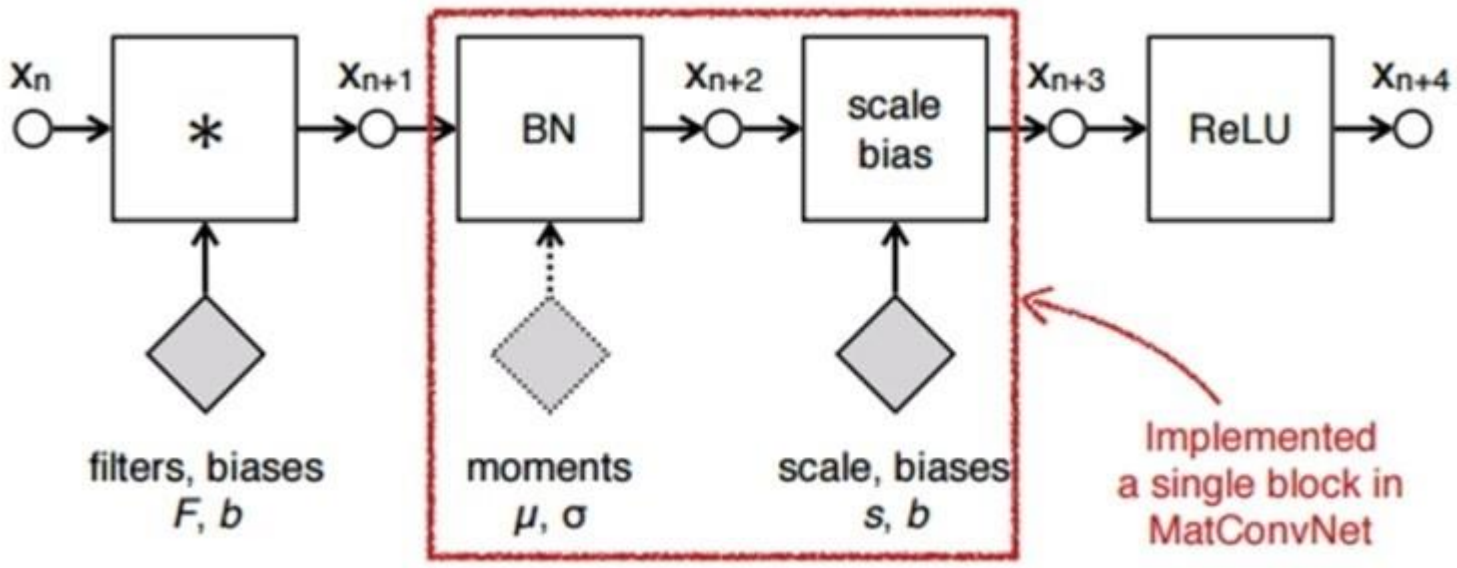
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization

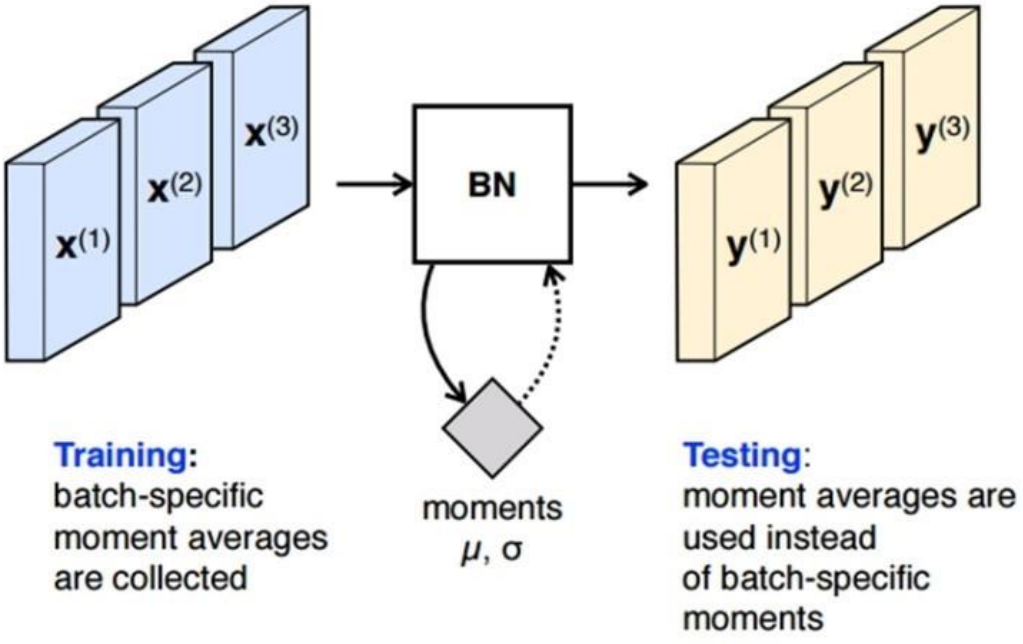
- Batch Normalization
  - ✓ Conducted before a Non-linear activation function



https://m.blog.naver.com/laonple/220808903260

Batch Normalization

- Batch Normalization: Training vs. Test
  - ✓ Training: compute gamma and beta for each mini-batch
  - ✓ Test: Use a weighted average of gamma and beta of mini-batches during training

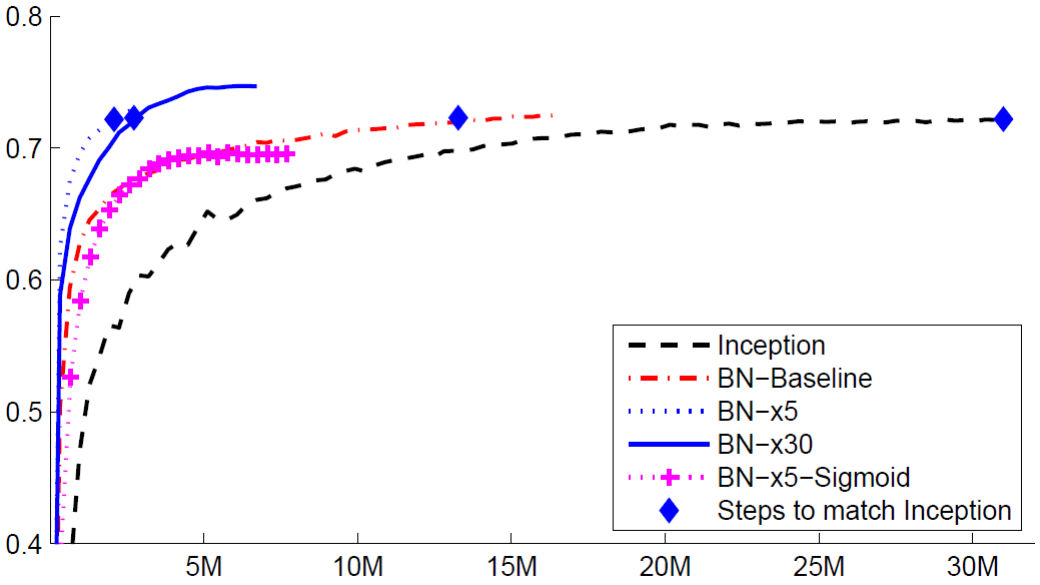


https://m.blog.naver.com/laonple/220808903260

Batch Normalization

- Batch Normalization: Effect

Training is faster



Performance is also improved

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Data Augmentation

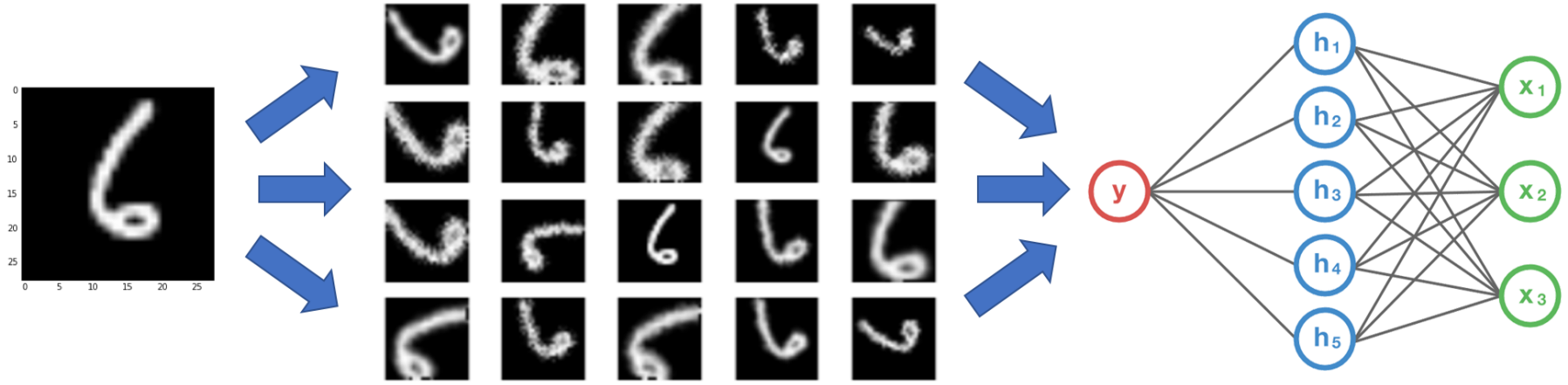
- Circumstance
  - ✓ My model has a large number of parameters but I do not have sufficient training examples
- Purpose
  - ✓ Enlarge the existing dataset to help train the model

	VGGNet	DeepVideo	GNMT
Used For	Identifying Image Category	Identifying Video Category	Translation
Input	Image 	Video	English Text 
Output	1000 Categories	47 Categories	French Text
Parameters	140M	~100M	380M
Data Size	1.2M Images with assigned Category	1.1M Videos with assigned Category	6M Sentence Pairs, 340M Words
Dataset	ILSVRC-2012	Sports-1M	WMT'14

https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Data Augmentation: Image Data

- Example



With only one image

Feed 20 different images to the model

https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

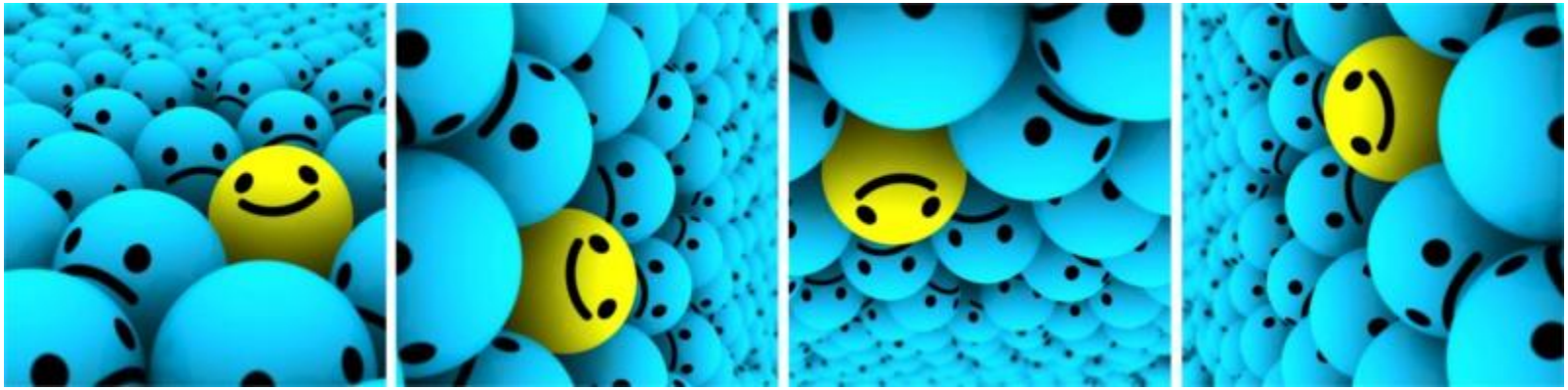
Data Augmentation: Image Data

- How to enlarge the original training examples?
  - ✓ Flip horizontally and vertically

https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Data Augmentation: Image Data

- How to enlarge the original training examples?
  - ✓ Rotation (image dimension is not be preserved unless the original image is a square)



https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Data Augmentation: Image Data

- How to enlarge the original training examples?
  - ✓ Scale outward or inward

https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Data Augmentation: Image Data

- How to enlarge the original training examples?
  - ✓ Crop: randomly sample a section from the original image

https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Data Augmentation: Image Data

- How to enlarge the original training examples?
  - ✓ Translation: moving the image along the X or Y direction or both

https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Data Augmentation: Image Data

- How to enlarge the original training examples?
  - ✓ Gaussian Noise: add artificially generated noise to the original image

https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Data Augmentation: Text Data

- Synonym Replacement (SR)
  - ✓ Randomly choose *n* words from the sentence that are not stop words. Replace each of these words with one of its synonyms chosen at random.

"A sad, superior human comedy played out on the back roads of life."

"A lamentable, superior human comedy played out on the backward roads of life."

Wei, J. X., & Zou, K. (2019).
EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. *arXiv preprint arXiv:1901.11196v1*.

Data Augmentation: Text Data

- Random Insertion (RI)
  - ✓ Find a random synonym of a random word in the sentence that is not a stop word. Insert that synonym into a random position in the sentence. Do this *n* times.

"A sad, superior human comedy played out on the back roads of life."

"A sad, superior human comedy played out on funniness the back roads of life."

Wei, J. X., & Zou, K. (2019).
EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. *arXiv preprint arXiv:1901.11196v1*.

Data Augmentation: Text Data

- Random Swap (RS)
  - ✓ Randomly choose two words in the sentence and swap their positions. Do this *n* times.

"A sad, superior human comedy played out on the back roads of life."

"A sad, superior human comedy played out on roads back the of life."

Wei, J. X., & Zou, K. (2019).
EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. *arXiv preprint arXiv:1901.11196v1*.

Data Augmentation: Text Data

- Random Deletion (RD)
  - ✓ For each word in the sentence, randomly remove it with probability *p*.

"A sad, superior human comedy played out on the back roads of life."

"A sad, superior human ~~comedy~~ ~~played~~ out on the ~~back~~ roads of life."

Wei, J. X., & Zou, K. (2019).
EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. *arXiv preprint arXiv:1901.11196v1*.


ANY
questions?

References

Research Papers

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
- Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).
- Lee, G., Jeong, J., Seo, S., Kim, C, & Kang, P. (2018). Sentiment classification with word localization based on weakly supervised learning with a convolutional neural network. Knowledge-Based Systems, 152, 70-82.
- Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025.
- Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1717-1724).
- Raffel, C., & Ellis, D. P. (2015). Feed-forward networks with attention can solve some long-term memory problems. arXiv preprint arXiv:1512.08756.

References

Research Papers

- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).
- Taigman, Y. et al. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification, CVPR'14.
- Wei, J. X., & Zou, K. (2019). EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. arXiv preprint arXiv:1901.11196v1.
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (pp. 2048-2057).

Other Materials

- http://machinelearninguru.com/computer_vision/basics/convolution/convolution_layer.html
- An overview of gradient descent optimization algorithms: http://ruder.io/optimizing-gradient-descent/
- Convolutional Neural Network: http://cs231n.github.io/convolutional-networks/
- Understanding LSTM Network
  - ✓ http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (English)
  - ✓ https://brunch.co.kr/@chris-song/9 (Korean)
- https://github.com/harvardnlp/seq2seq-talk/blob/master/slidescmu.pdf