22년 삼성 AI 전문가과정
6월 9일 목요일 1교시
장병탁

인공지능

16차시 : Deep Learning

서울대학교 컴퓨터공학부
담당 교수: 장병탁

Seoul National University
Byoung-Tak Zhang

Lecture Overview

인공지능

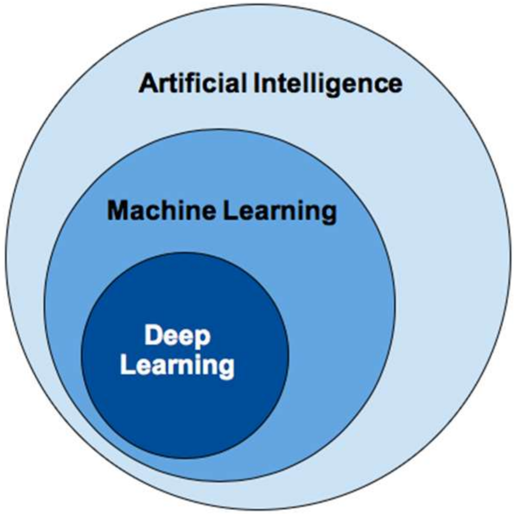
16차시 : Deep Learning

서울대학교 컴퓨터공학부
담당 교수: 장병탁

Seoul National University
Byoung-Tak Zhang

Introduction: Deep Learning

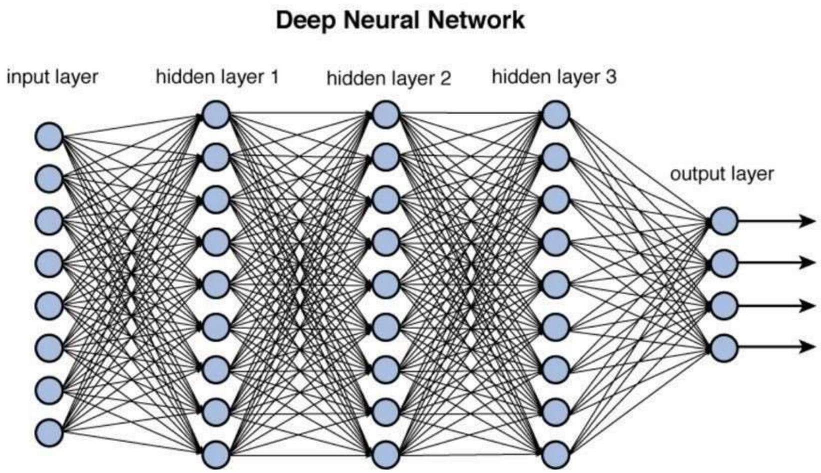
- **Machine Learning (Previous lecture)**
  - **Machine learning** improves its performance after observations about the world
  - Models: decision trees, linear models, nonparametric models, ensemble models, etc.
- **Deep Learning (This lecture)**
  - **Deep learning** is a broad family of techniques for ML
  - The word "deep" refers to the fact that the circuits are typically organized into many layers
  - Applications: visual object recognition, machine translation, speech recognition, etc.
  - Models: feedforward neural networks, convolutional neural networks, recurrent neural networks, etc.



Why Deep Learning?

❑ Why Deep Learning?

- Expressive power of deep models
- Stacking many layers can generate highly-complex nonlinear decision boundaries that separate the complex patterns in the data



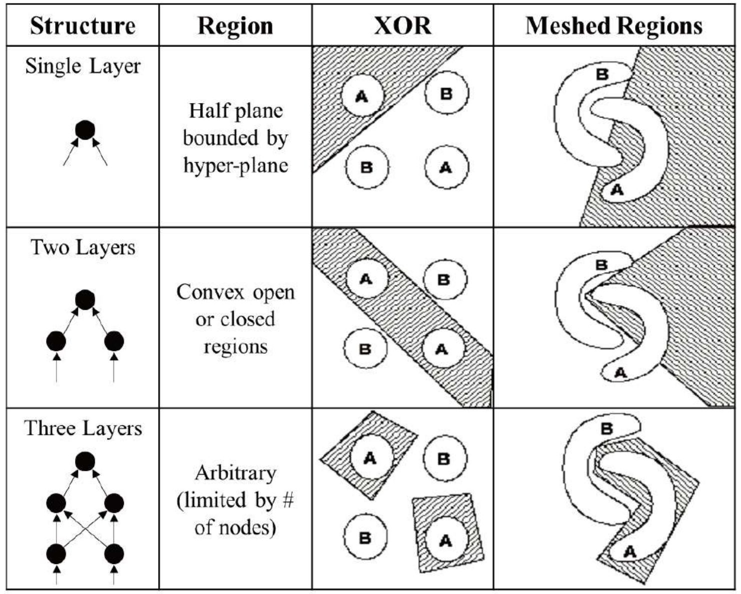


그림 3.2 딥구조 학습 모델의 필요성

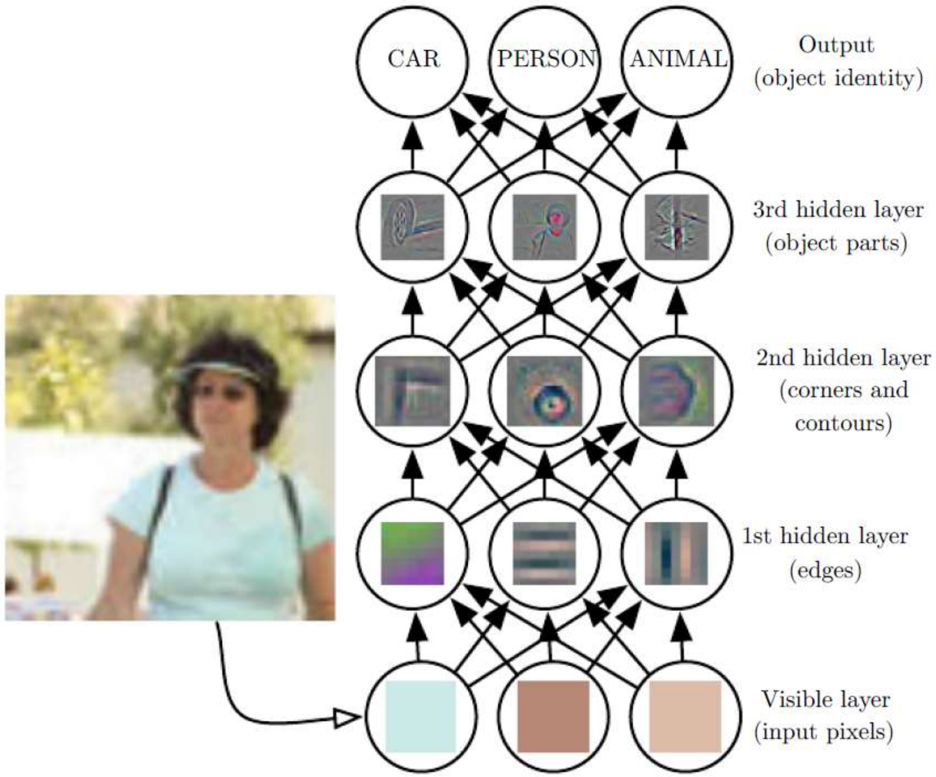
출처: Figure #2

Computation Graphs for Deep Learning

❑ Hidden layers

- Values computed at each layer of the network is a different representation for the input **x**

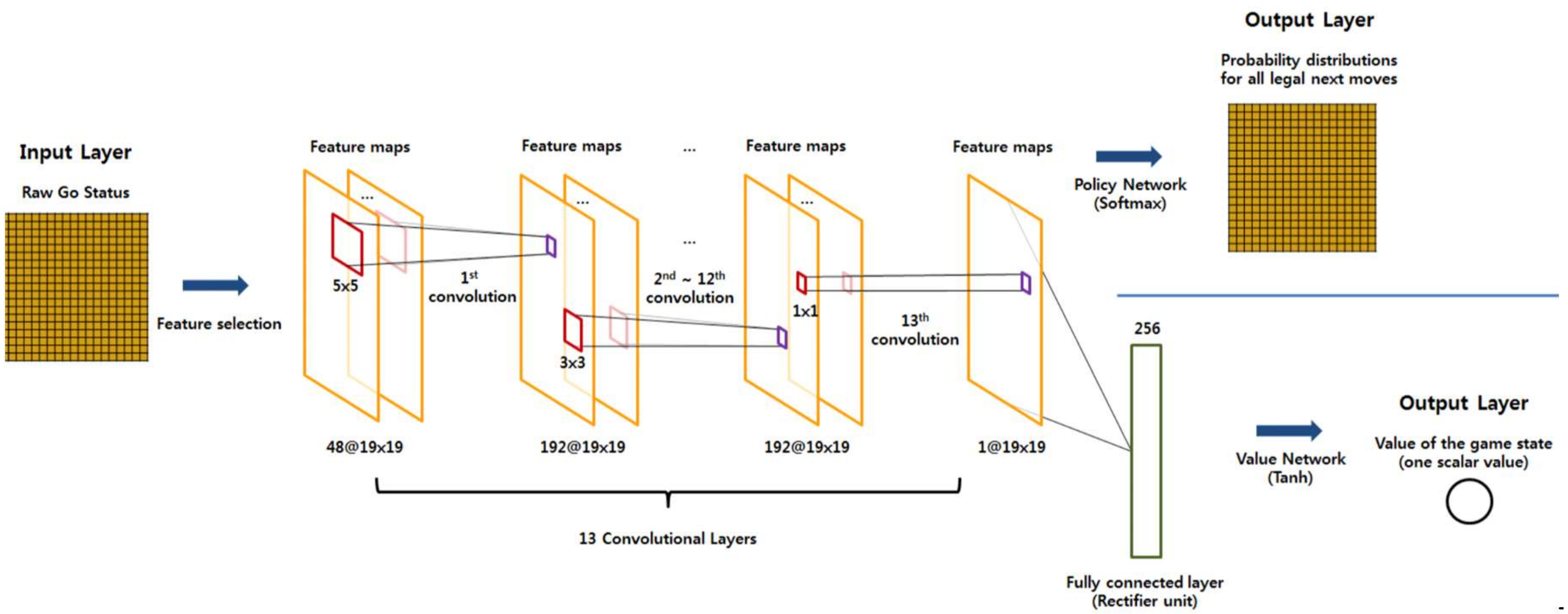
Example: a network learning to recognize complex objects in images may form internal layers that detect useful subunits: edges, corners, ellipses, eyes, faces—cats



출처: **Figure #8**

Applications

- **AlphaGo**
  - Deep learning + reinforcement learning, deep Q-network, value network, policy network, policy gradient



출처: Figure #25

Outline (Lecture 16)

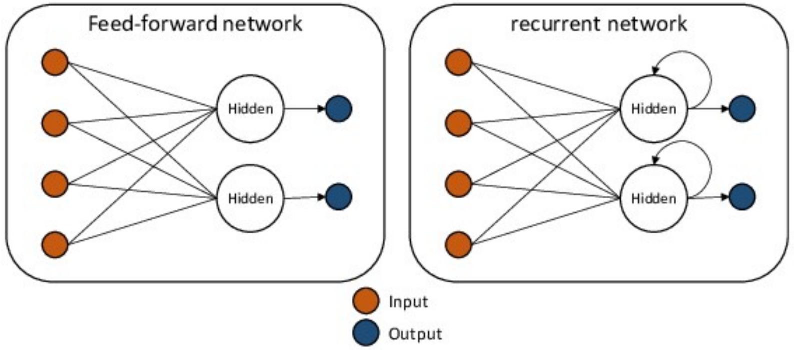
Stuart Russell & Peter Norvig (2021), Artificial Intelligence: A Modern Approach (4th Edition)

16.1 Simple Feedforward Networks

16.1 Simple Feedforward Networks (1/7)

Feedforward Network

- Feedforward network
  - has connections only in one direction—directed acyclic graph
- Recurrent network
  - feeds its intermediate or final outputs back into its own inputs



출처: Figure #2

Unit

- Weighted sum of the inputs from predecessor nodes + nonlinear function

$$a_j = g_j\left(\sum_i w_{i,j} a_i\right) \equiv g_j(in_j) = g_j(\mathbf{w}^\top \mathbf{x})$$

a_j: output of unit
$w_{i,j}$: weight from unit i to j
g_j: nonlinear activation function
in_j: weighted sum of inputs to j
$\mathbf{w}, \mathbf{x}$: vector of weights and inputs

16.1 Simple Feedforward Networks (2/7)

Activation functions

Sigmoid function	**ReLU function**
The logistic or sigmoid function, which is also used in logistic regression $\sigma(x) = 1/(1 + e^{-x})$	is an abbreviation for rectified linear unit $\mathrm{ReLU}(x) = \max(0, x)$
Softplus function	**Tanh function**
a smooth version of the ReLU function $\mathrm{softplus}(x) = \log(1 + e^{x})$	$\tanh(x) = \dfrac{e^{2x} - 1}{e^{2x} + 1}$

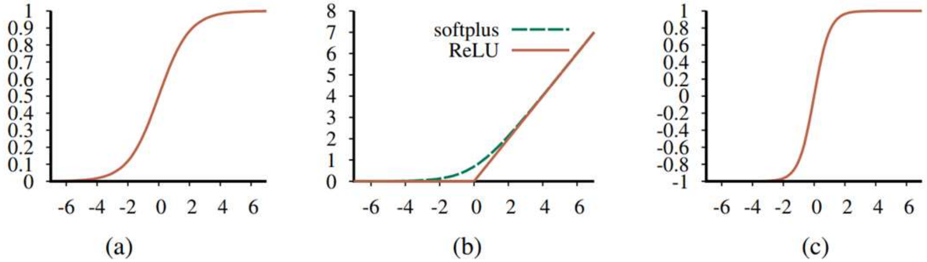


Figure 22.2 Activation functions commonly used in deep learning systems: (a) the logistic or sigmoid function; (b) the ReLU function and the softplus function; (c) the tanh function.

g: monotonically nondecreasing

$g' \geq 0$

출처: Figure #3

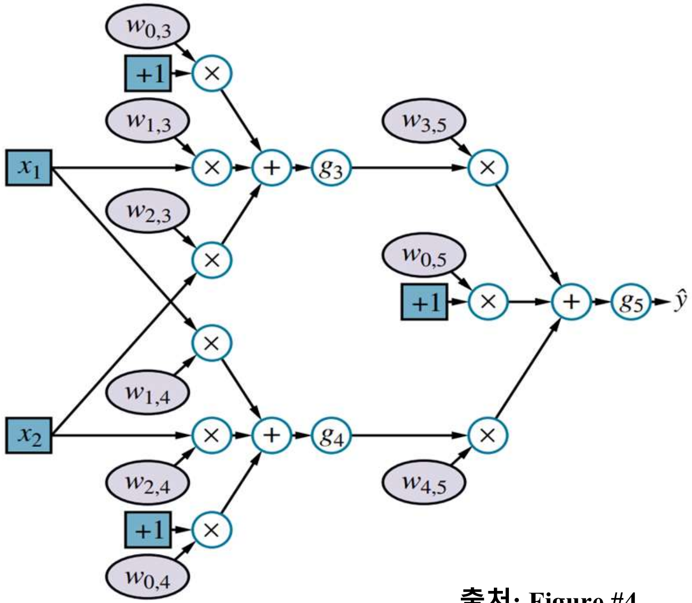
16.1 Simple Feedforward Networks (3/7)

Fully connected Feedforward Network

- Fully connected
  - Every node in each layer is connected to every node in the next layer
- Adjusted weights $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$ to fit the data
- $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{g}^{(2)}(\mathbf{W}^{(2)}\mathbf{g}^{(1)}(\mathbf{W}^{(1)}\mathbf{x}))$

$\mathbf{W}^{(1)}$: weights in the first layer $(w_{1,3}, w_{1,4}, etc)$
$\mathbf{W}^{(2)}$: weights in the second layer $(w_{3,5}, etc)$
$\mathbf{g}^{(1)}, \mathbf{g}^{(2)}$: activation functions in the first and second layers

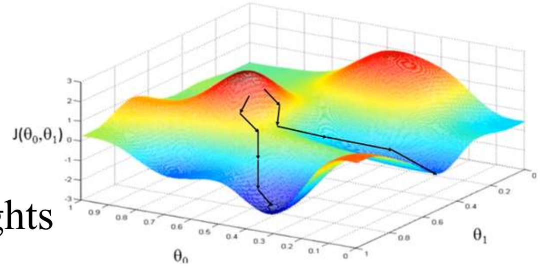
Not every network is fully connected!
See Section 22.3 that choosing the connectivity of the network is also important in achieving effective learning



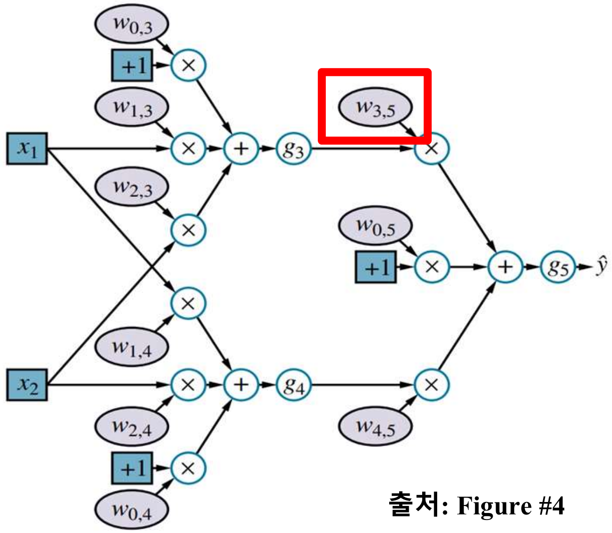
출처: Figure #4

16.1 Simple Feedforward Networks (4/7)

Gradients and learning



- Gradient descent
  - Calculate the gradient of the loss function with respect to the weights
  - Adjust the weights along the gradient direction to reduce the loss
  - $Loss(h_{\mathbf{w}}) = L_2(y, h_{\mathbf{w}}(\mathbf{x})) = \| y - h_{\mathbf{w}}(\mathbf{x}) \|^2 = (y - \hat{y})^2$
  - Chain rule: $\frac{\partial g(f(x))}{\partial x} = \frac{g'(f(x))\partial f(x)}{\partial x}$



- **Weights leading into units in the output layer:**

$$\frac{\partial}{\partial w_{3,5}} Loss(h_{\mathbf{w}}) = \frac{\partial}{\partial w_{3,5}} (y - \hat{y})^2 = -2(y - \hat{y}) \frac{\partial \hat{y}}{\partial w_{3,5}}$$

$$= -2(y - \hat{y}) \frac{\partial}{\partial w_{3,5}} g_5(in_5) = -2(y - \hat{y}) g'_5(in_5) \frac{\partial}{\partial w_{3,5}} (in_5)$$

$$= -2(y - \hat{y}) g'_5(in_5) \frac{\partial}{\partial w_{3,5}} \left(w_{0,5} + w_{3,5} a_3 + w_{4,5} a_4\right)$$

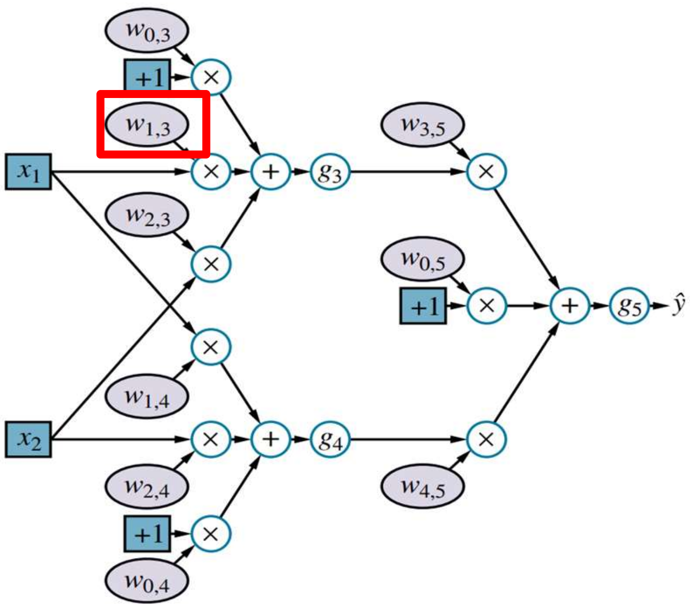
$$= -2(y - \hat{y}) g'_5(in_5) a_3$$

출처: Figure #4

16.1 Simple Feedforward Networks (5/7)

- **Weights leading into units in the hidden layer**

$$\frac{\partial}{\partial w_{1,3}} Loss(h_{\mathbf{W}})$$
$$= -2(y-\hat{y})g'_5(in_5)\frac{\partial}{\partial w_{1,3}}\left(w_{0,5}+w_{3,5}a_3+w_{4,5}a_4\right)$$
$$= -2(y-\hat{y})g'_5(in_5)w_{3,5}\frac{\partial}{\partial w_{1,3}}a_3$$
$$= -2(y-\hat{y})g'_5(in_5)w_{3,5}\frac{\partial}{\partial w_{1,3}}g_3(in_3)$$
$$= -2(y-\hat{y})g'_5(in_5)w_{3,5}g'_3(in_3)\frac{\partial}{\partial w_{1,3}}in_3$$
$$= -2(y-\hat{y})g'_5(in_5)w_{3,5}g'_3(in_3)\frac{\partial}{\partial w_{1,3}}\left(w_{0,3}+w_{1,3}x_1+w_{2,3}x_2\right)$$
$$= -2(y-\hat{y})g'_5(in_5)w_{3,5}g'_3(in_3)x_1$$



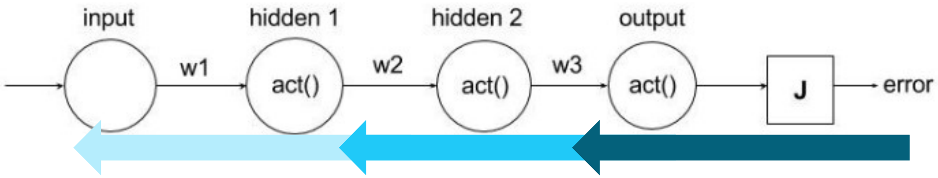
출처: Figure #4

For multiple examples, the gradient is just the sum of the gradients for the individual examples

16.1 Simple Feedforward Networks (6/7)

Back-propagation

- Error at the output is passed back through the network
- Perceived error: $\Delta_5 = 2(\hat{y} - y)g'_5(in_5)$
- Gradient w.r.t. $w_{3,5}$: $\Delta_5 a_3$
- Perceived error: $\Delta_3 = \Delta_5 w_{3,5} g'_3(in_3)$
- Gradient w.r.t. $w_{1,3}$: $\Delta_3 x_1$



출처: Figure #5

Reading assignment (from textbook 807pg)

if Δ_5 is positive, that means $\hat{y}$ is too big (recall that g' is always nonnegative); if a_3 is also positive, then increasing $w_{3,5}$ will only make things worse, whereas if a_3 is negative, then increasing $w_{3,5}$ will reduce the error. The magnitude of a_3 also matters: if a_3 is small for this training example, then $w_{3,5}$ didn't play a major role in producing the error and doesn't need to be changed much.

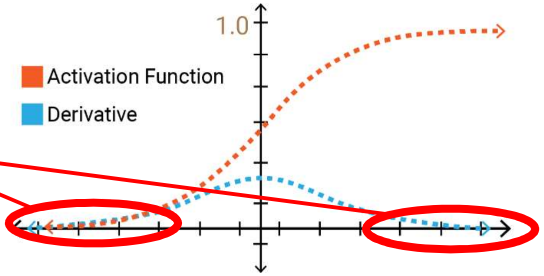
출처: Figure #6

16.1 Simple Feedforward Networks (7/7)

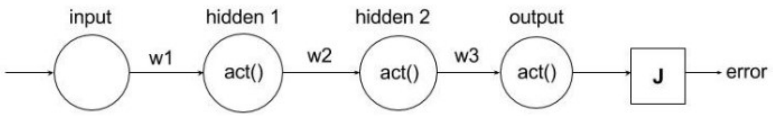
Vanishing gradient

- If $g'_j \approx 0$, small change of weights
- Error signals are extinguished altogether as they are propagated back through the network

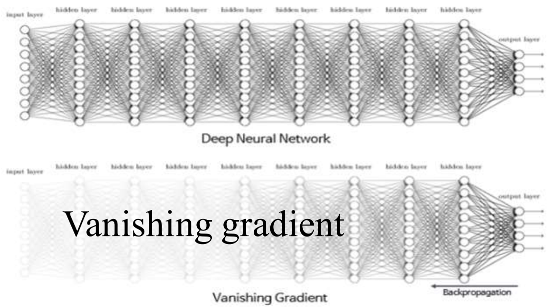
$$\frac{\partial Sigmoid(z_3)}{\partial z_3} \ll 1$$



출처: Figure #7



$output = Sigmoid(z_3)$,
$z_3 = h_2 w_3 + b_3$

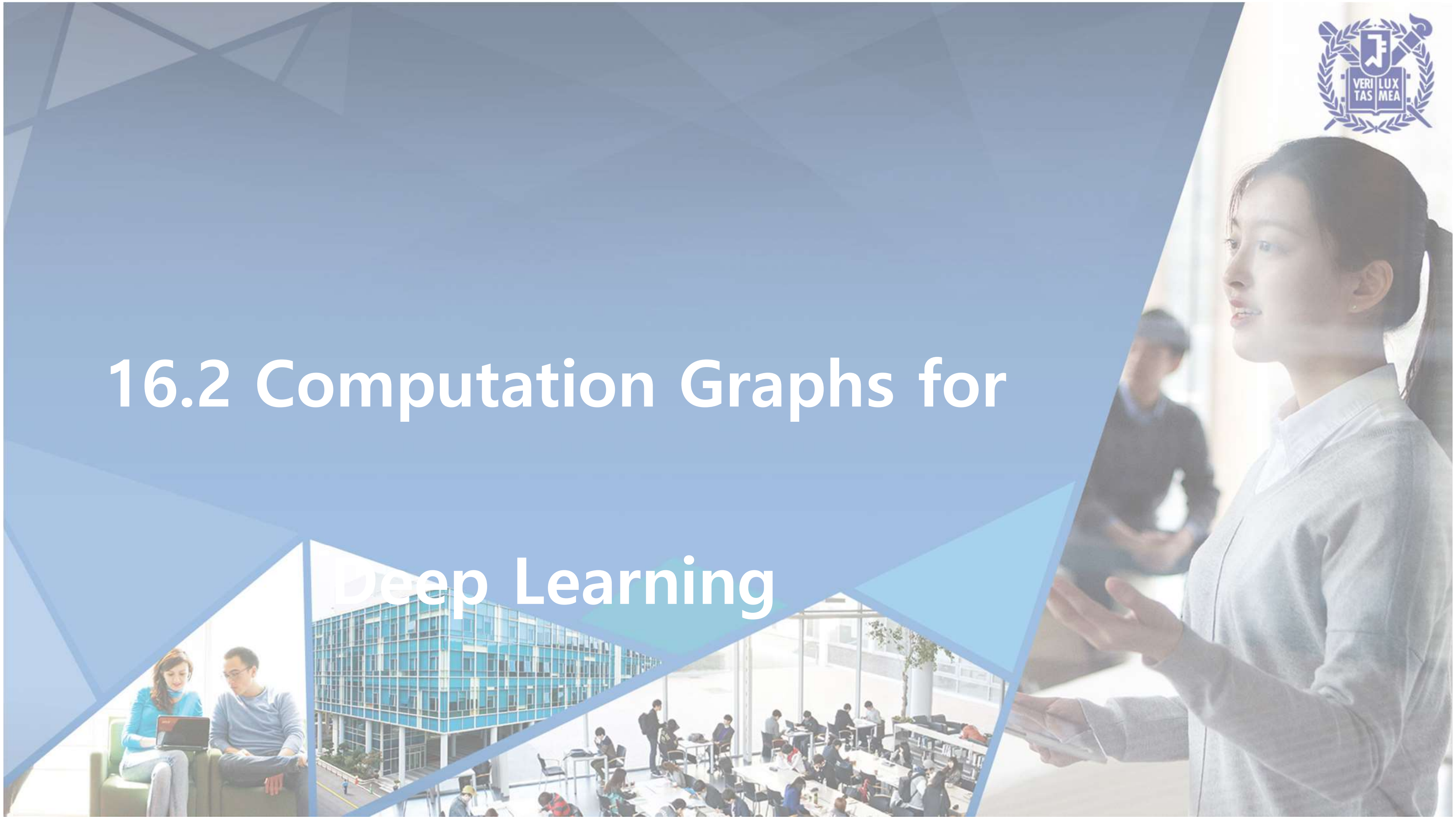


$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial output}\frac{\partial output}{\partial w_3} = \frac{\partial L}{\partial output}\frac{\partial Sigmoid(z_3)}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

$$\frac{\partial L}{\partial w_2} = \left(\frac{\partial L}{\partial output}\frac{\partial Sigmoid(z_3)}{\partial z_3}\right)\frac{\partial z_3}{\partial hidden_2}\frac{\partial Sigmoid(z_2)}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

$$\frac{\partial L}{\partial w_1} = \left(\left(\frac{\partial L}{\partial output}\frac{\partial Sigmoid(z_3)}{\partial z_3}\right)\frac{\partial z_3}{\partial hidden_2}\frac{\partial Sigmoid(z_2)}{\partial z_2}\right)\frac{\partial z_2}{\partial hidden_1}\frac{\partial Sigmoid(z_1)}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

16.2 Computation Graphs for Deep Learning



16.2 Computation Graphs for Deep Learning (1/4)

Basic idea of deep learning

- Represents hypotheses as computation graphs with tunable weights
- Computes gradient of the loss function w.r.t. those weights to fit the data

Input encoding

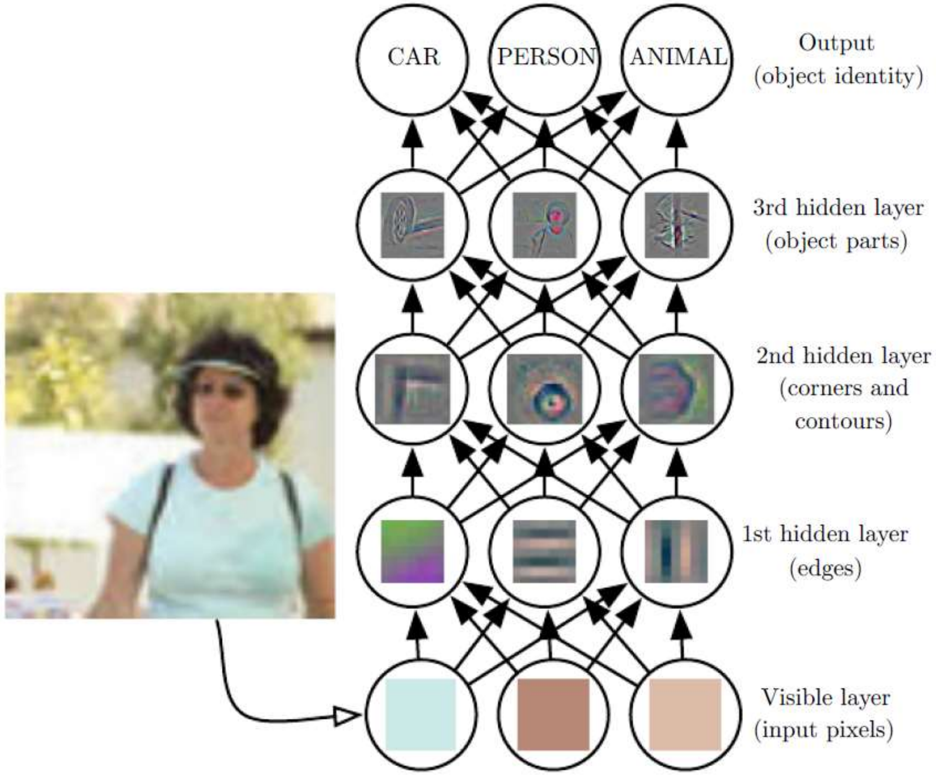
- Boolean attributes: true, false mapped to 1, 0 or +1, -1
- Numeric attributes (integer or real-valued): used as is or maybe scaled to fit within a fixed range (log scale)
- Categorical attributes with more than two values: one-hot encoding
  - Where {*French, Italian, Thai*} encodes *Thai* as 001 and encodes *French* as 100

16.2 Computation Graphs for Deep Learning (2/4)

Hidden layers

- Values computed at each layer of the network is a different representation for the input x

Example: a network learning to recognize complex objects in images may form internal layers that detect useful subunits: edges, corners, ellipses, eyes, faces—cats



출처: **Figure #8**

16.2 Computation Graphs for Deep Learning (3/4)

Output layers and loss functions

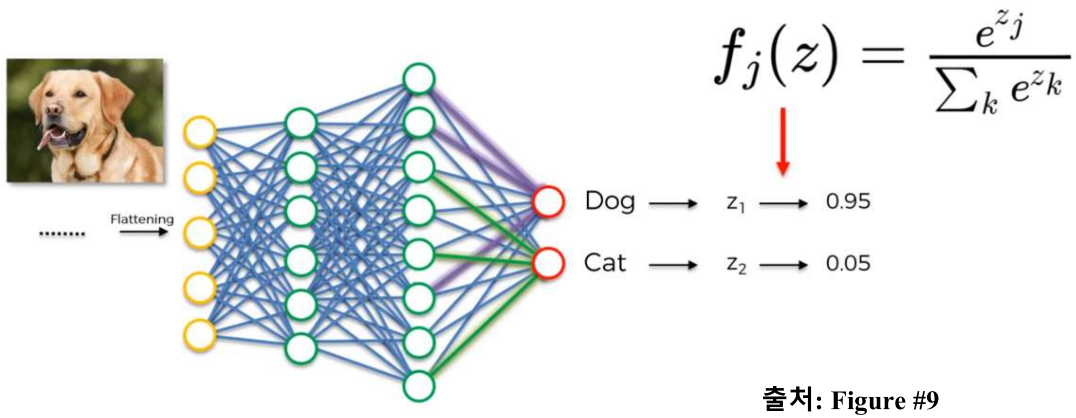
- Categorical attributes with more than two values: one-hot encoding
  - Where {*sun, rain, cloud, snow*} encodes *cloud* as 0010 and encodes *sun* as 1000
- Minimizing the cross-entropy loss $H(P, Q)$
  - Measure of dissimilarity between two distributions P and Q
  - $H(P, Q) = \mathbf{E}_{\mathbf{z} \sim P(\mathbf{z})}[\log Q(\mathbf{z})] = \int P(\mathbf{z}) \log Q(\mathbf{z}) d\mathbf{z}$
  - P: true distribution over training samples $P^*(\mathbf{x}, \mathbf{y})$
  - Q: predictive hypothesis $P_{\mathbf{W}}(\mathbf{y}|\mathbf{x})$
  - Equivalent to minimizing negative log likelihood

16.2 Computation Graphs for Deep Learning (4/4)

Output layers and loss functions

- Need to be able to interpret the output of the network as a probability
  - Example: classifiers used for object recognition need to recognize thousands of categories of objects
  - Use softmax layer:

$$\text{softmax}(\mathbf{in})_k = \frac{e^{in_k}}{\sum_{k'=1}^{d} e^{in_{k'}}}$$



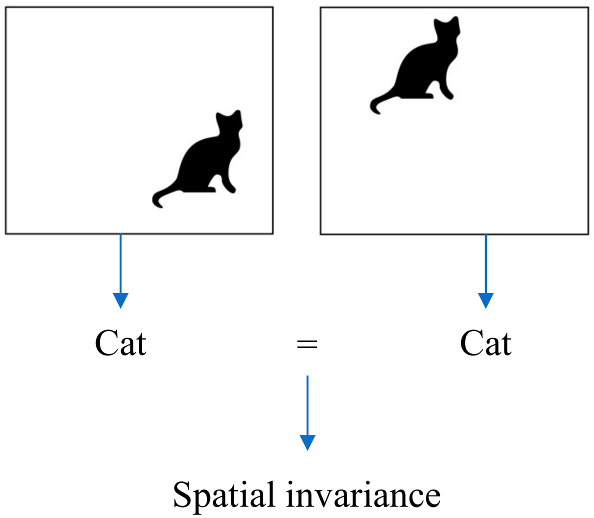
출처: Figure #9

16.3 Convolutional Networks

16.3 Convolutional Networks (1/7)

Motivation

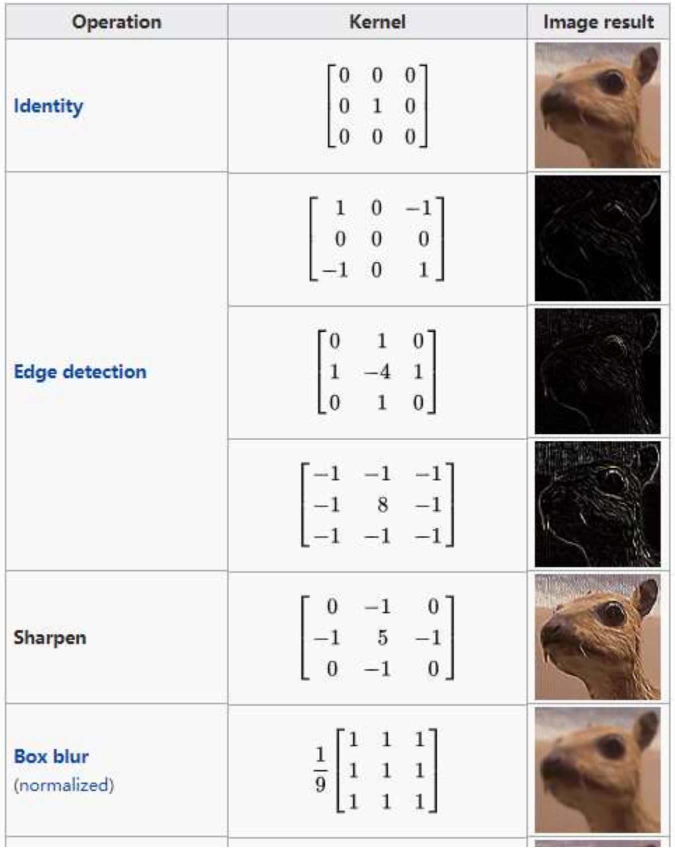
- Image not a simple vector since adjacency of pixels
  - A network with fully connected layers = training with randomly permuted pixels (no locality)
  - Vast parameter space with huge computational budget
  - Solution: each hidden unit receives input from only a small, local region of the image
- Spatial invariance
  - Detect the same feature wherever it appear in the image
  - Weight sharing: Weights connecting a local region to a unit in the hidden layer to be the same for each hidden unit



16.3 Convolutional Networks (2/7)

Kernel and Convolution

- **Kernel**
  - A pattern of weights that is replicated across multiple local regions
- **Convolution**
  - Process of applying the kernel to the pixels of the image
  - i.e. slide over the image spatially, to find shift-invariant patterns



Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

16.3 Convolutional Networks (3/7)

Convolutional operation

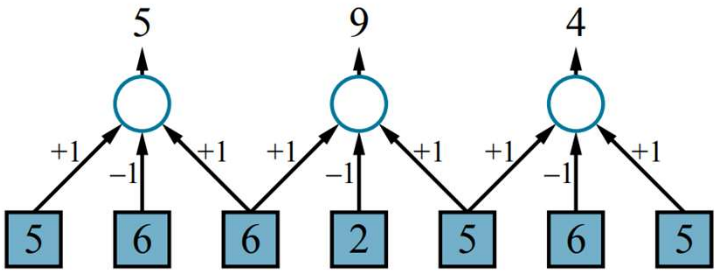


Figure 22.4 An example of a one-dimensional convolution operation with a kernel of size $l = 3$ and a stride $s = 2$. The peak response is centered on the darker (lower intensity) input pixel. The results would usually be fed through a nonlinear activation function (not shown) before going to the next hidden layer.

출처: Figure #9

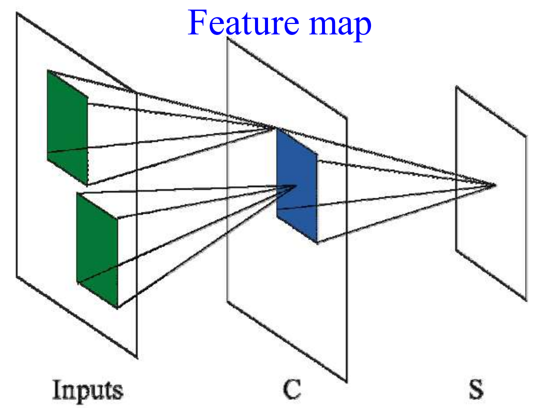


그림 3.4 컨볼루션 네트워크 (Hinton et al., 2006)

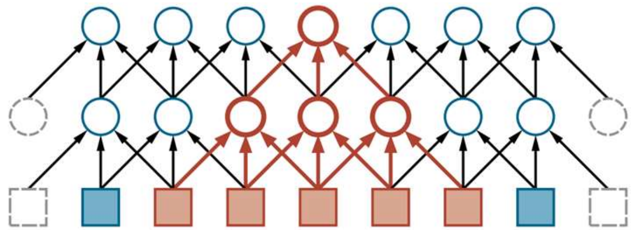
출처: Figure #10

Receptive field (shown in red)

Receptive field grows exponentially with depth!

$$O(ls^m)$$

Where m is the number of layers

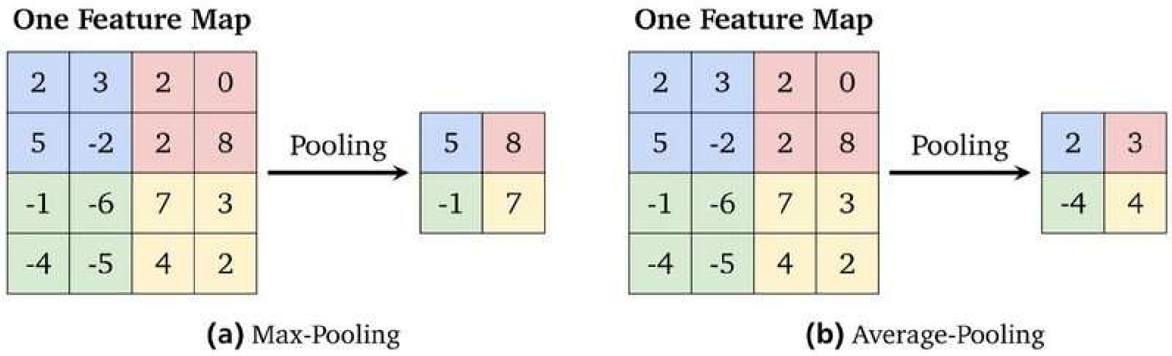


출처: Figure #11

16.3 Convolutional Networks (4/7)

Pooling and down-sampling

- **Pooling layer**
  - Summarizes a set of adjacent units with a single value to down-sample it
  - Reduces the number of weights required in subsequent layers
  - Max pooling, average pooling, min pooling, L2-norm pooling, etc.

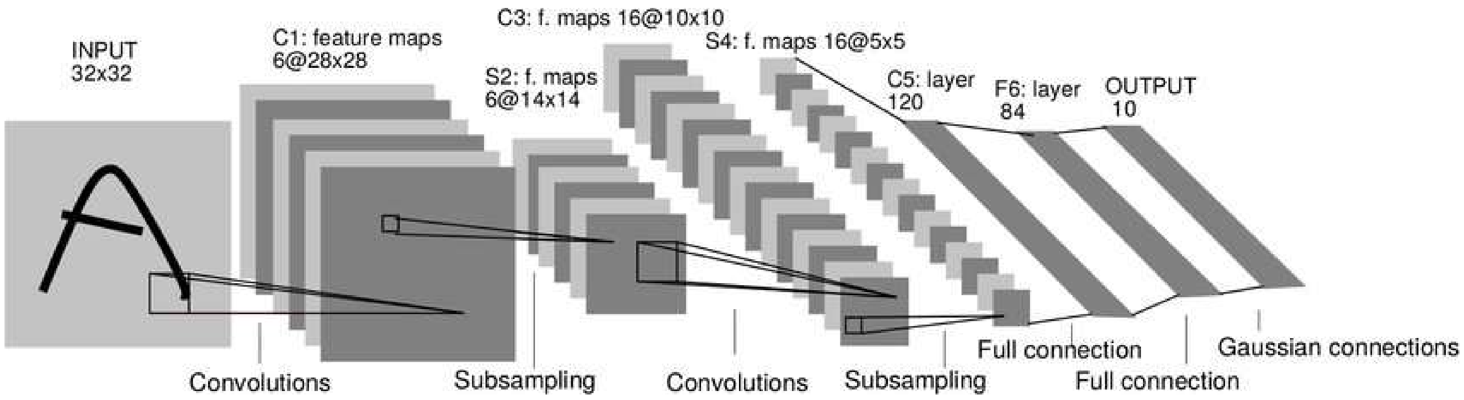


출처: Figure #12

16.3 Convolutional Networks (5/7)

Total structure of ConvNet (CNN)

- Convolution layer, subsampling (pooling) layer, fully-connected layer
- Feature maps, dimension reduction, classification



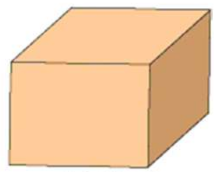
CNN for recognizing the character image "A" (LeCun et al., 1998)

출처: Figure #13

16.3 Convolutional Networks (6/7)

Tensor operations in CNNs

- **Tensor**: multidimensional arrays of any dimension
  - Vectors (one-dimensional) and matrices (two-dimensional) tensors
  - Image: (H, W, D) 3D tensor shape
  - GPU, TPU make available a high degree of parallelism

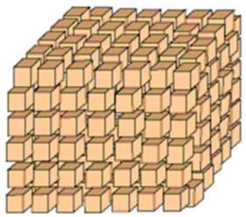


3 d -Tensor

- **Example**
  - Training a 256×256 RGB image with mini batch size of 64

    : 4D tensor of tensor size 256×256×3×64
  - Apply 96 kernels of size 5×5×3 with a stride of 2

    : output tensor of size 128×128×96×64

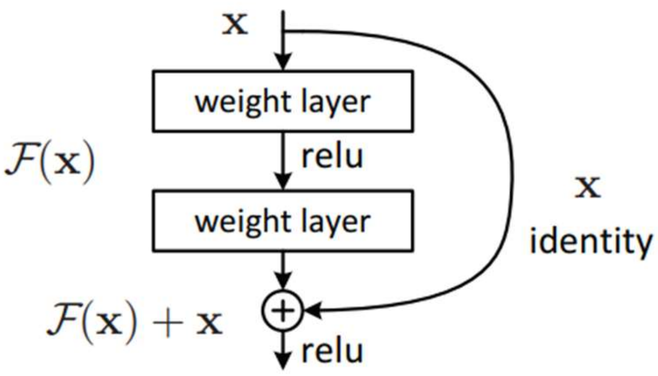


6 d -Tensor

16.3 Convolutional Networks (7/7)

Residual networks

- An approach to building very deep networks that avoid the problem of vanishing gradients
- Key idea of residual networks is that a layer should perturb the representation from the previous layer rather than replace it entirely
- $\mathbf{z}^{(i)} = \mathbf{g}_r^{(i)}(\mathbf{z}^{(i-1)} + f(\mathbf{z}^{(i-1)}))$
- $\mathbf{g}_r^{(i)}$ is ith activation functions
- $f(\mathbf{z}) = \mathbf{V}\mathbf{g}(\mathbf{W}\mathbf{z})$
- a general-purpose tool that makes deep networks more robust hence go deeper



출처: Figure #14

16.4 Learning Algorithms

16.4 Learning Algorithms (1/1)

Training a neural network = modifying the network's parameters so as to minimize the loss function

Optimization algorithm

- Each update step in the gradient descent

  $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w})$ α: learning rate

- Gradient descent: L is defined w.r.t. entire training set

- Stochastic gradient descent (SGD)

  : defined w.r.t minibatch of m examples chosen randomly at each step

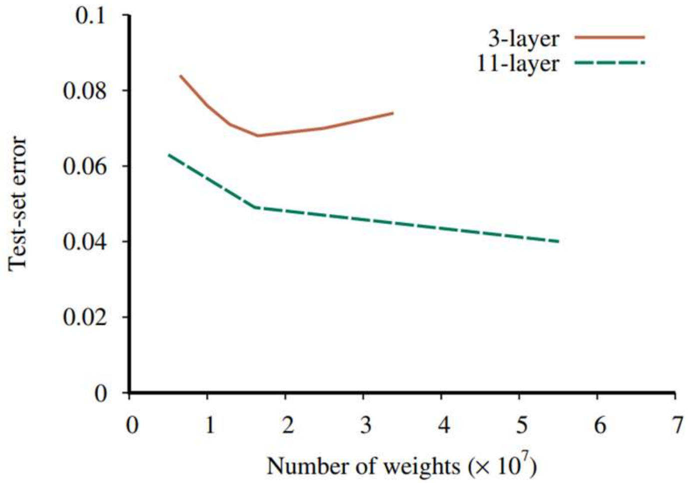
  - helps the algorithm escape small local minima
  - computational cost of each weight update step is a small constant
  - advantage of hardware parallelism in GPUs or TPUs
  - (More in Textbook Section 19.6.2 and 22.4)

16.5 Generalization

16.5 Generalization (1/3)

Choosing a network architecture

- Neural network architectures designed to generalize on particular type of data
  - CNN: same feature extractor is useful at all locations across a spatial grid (e.g. image)
  - RNN: same update rule is useful at all points in a stream of sequential data (e.g. text, audio)
- Depth matters



16.5 Generalization (2/3)

Penalizing large weights

- **Regularization**
  - Limiting the complexity of a model for generalization (Section 19.4.3 in textbook)
- **Weight decay**
  - Adding a penalty $\lambda \sum_{i,j} W_{i,j}^2$ to the loss function
  - Larger hyperparmeter λ encourages the weights to become small
  - Common to use weight decay with λ near 10^{-4}
  - In networks with sigmoid activation, weight decay keeps the activations near the linear part, avoiding vanishing gradients

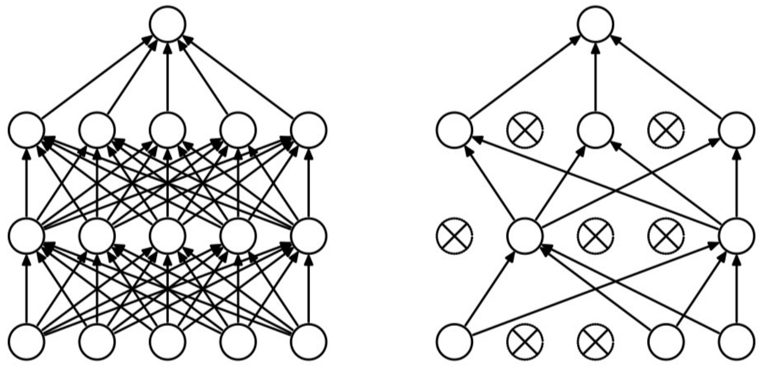
Dropout

- Randomly set neurons to zero in the forward pass
- Dropout forces the model to learn multiple, robust explanations for each input

16.5 Generalization (3/3)

Dropout

- Randomly set neurons to zero in the forward pass
- No change in backward pass
- Dropout forces the model to learn multiple, robust explanations for each input
- One of regularization techniques



출처: Figure #15

(a) Standard Neural Net (b) After applying dropout.

16.6 Recurrent Neural Networks

16.6 Recurrent Neural Networks (1/2)

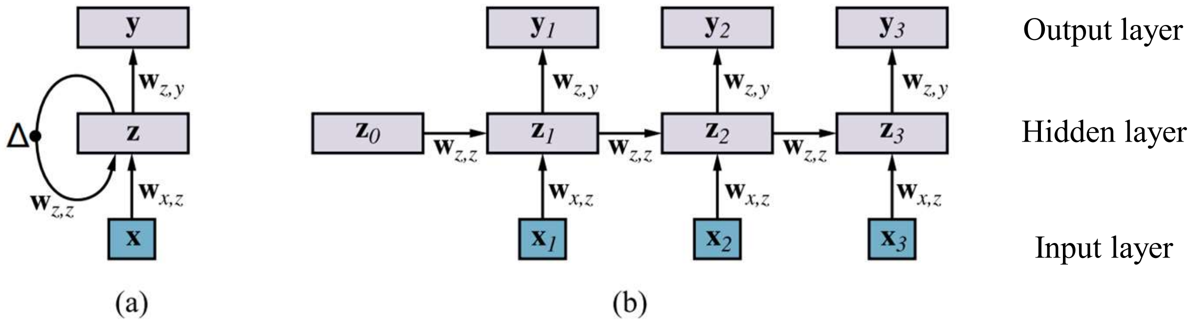
Recurrent neural networks (RNNs)

- Units take as input a value computed from their own output at an earlier step
  - Allows the RNN to have internal state, or memory

$$\mathbf{z}_t = f_{\mathbf{w}}(\mathbf{z}_{t-1}, \mathbf{x}_t) = \mathbf{g}_z(\mathbf{W}_{z,z}\mathbf{z}_{t-1} + \mathbf{W}_{x,z}\mathbf{x}_t)$$

$$\hat{\mathbf{y}}_t = \mathbf{g}_y(\mathbf{W}_{z,y}, \mathbf{z}_t)$$

- To analyze sequential data, new input vector $\mathbf{x}_t$ arrives at each time step

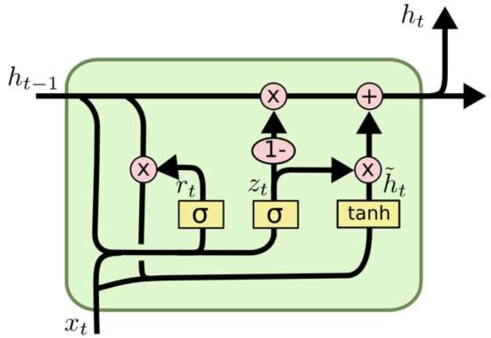


출처: Figure #16

16.6 Recurrent Neural Networks (2/2)

Long short-term memory RNNs (LSTM)

- Vanilla RNNs suffer from the vanishing gradient problem when $w_{z,z} < 1$ or exploding gradient problem when $w_{z,z} > 1$
- LSTM has the goal of enabling information to be preserved over many time steps
- New information enters the memory by adding updates so that the gradient expressions do not accumulate multiplicatively over time



출처: Figure #17

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

16.7 Unsupervised Learning and Transfer Learning

16.7 Unsupervised Learning and Transfer Learning (1/4)

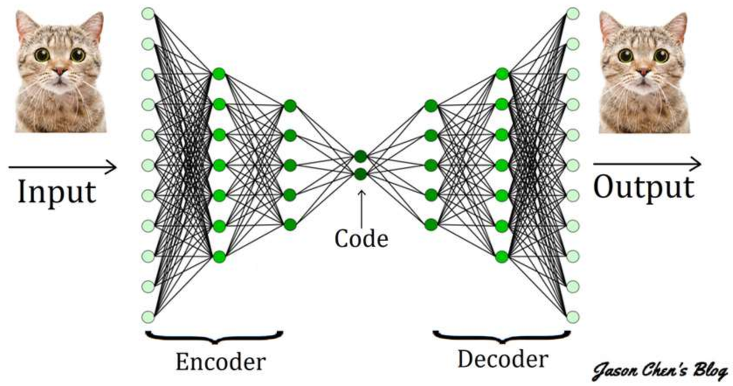
Deep learning systems we have discussed so far are based on supervised learning but…

- Supervised learning
  - Requires each training example to be labeled
- Often requires far more labeled data than a human would for the same task
  - A child needs to see only one picture of a giraffe to recognize giraffes
- Large data sets usually require scarce and expensive human labor

16.7 Unsupervised Learning and Transfer Learning (2/4)

Unsupervised learning

- Learns solely from unlabeled inputs **x**
- Produces generative models, which can produce realistic text, images, audio, and video
- **Example**
  - Probabilistic PCA, autoencoders, deep autoregressive models, generative adversarial networks (GANs), unsupervised translation

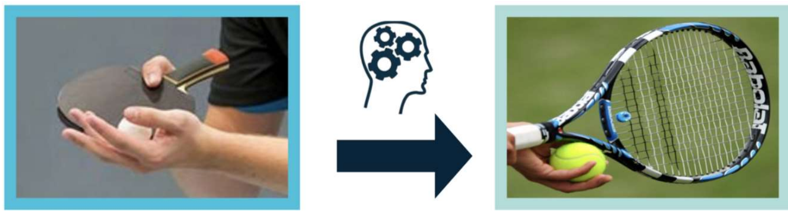


출처: Figure #18

16.7 Unsupervised Learning and Transfer Learning (3/4)

Transfer learning

- Requires some labeled examples but is able to improve their performance further by studying labeled examples for different tasks
- **Motivation**
  - A person who has already learned to play table tennis will find it easier to learn related sports such as tennis

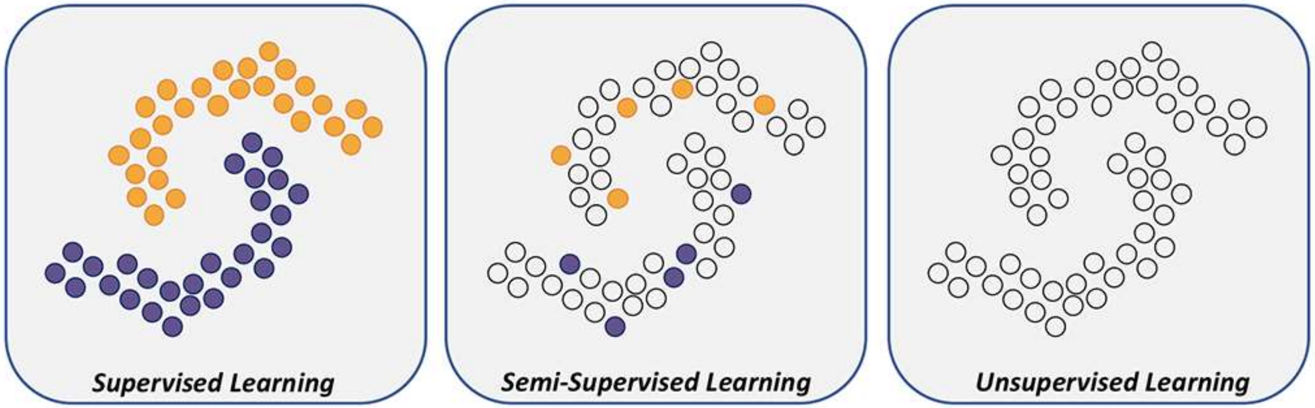


- **Usage example**
  - Copy over the weights learned for task A to a network that will be trained for task B

16.7 Unsupervised Learning and Transfer Learning (4/4)

Semi-supervised learning

- Requires some labeled examples but is able to improve their performance further by also studying unlabeled examples



출처: Figure #18

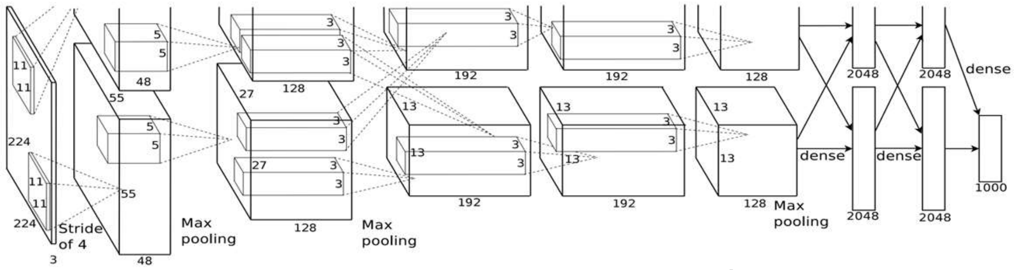
16.8 Applications

16.8 Applications (1/8)

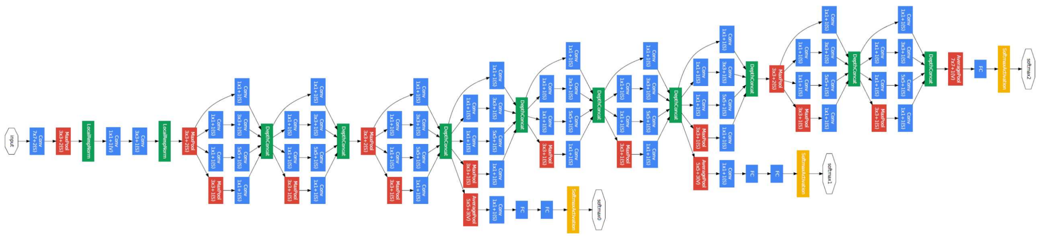
1) ImageNet Challenge

- ILSVRC (ImageNet Large-Scale Visual Recognition Challenge)
- Image classification/localization, 1.2M labeled images, 1000 classes

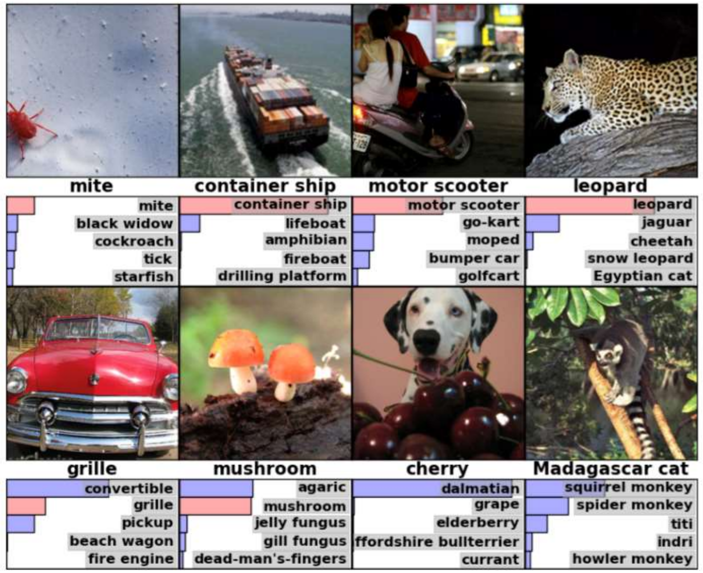
출처: Figure #20



AlexNet (2012)   출처: Figure #18



GoogLeNet (2015)   출처: Figure #19

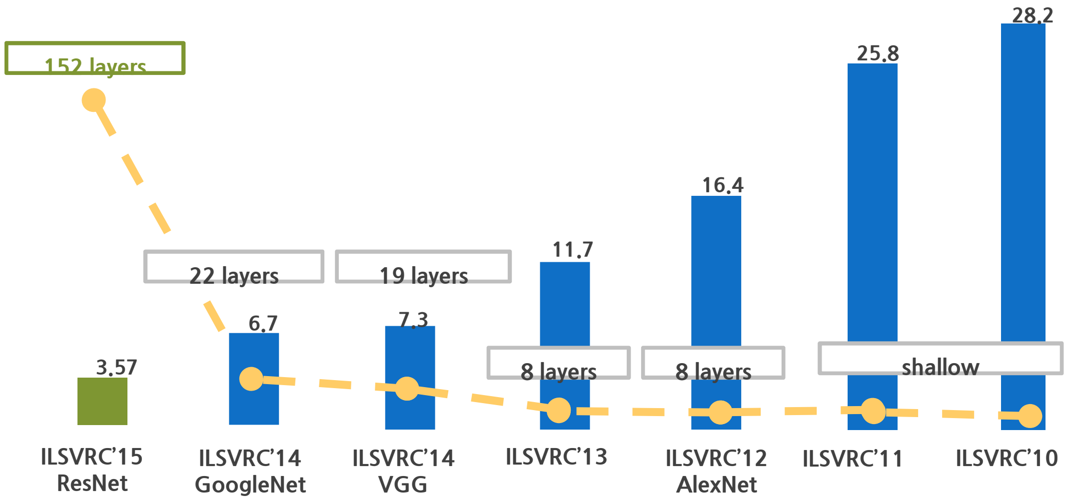


16.8 Applications (2/8)

2) AlexNet Breakthrough

- Eight weighted layers (5 convolution layers, 3 fully connected layers)
- GPU parallel processing: GTX 580 GPU (3GB) x 2
  - Single GPU is not capable of learning 1.2 million training images
- Distributing the entire network to each GPU

3) Deeper and Deeper!

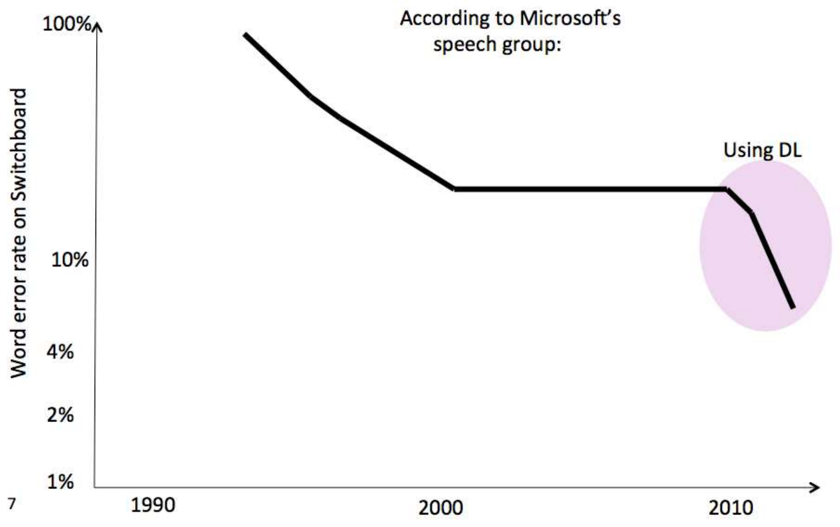


출처: Figure #21

16.8 Applications (3/8)

4) Speech Recognition

- ~2010 GMM-HMM
  (Dynamic Bayesian Models)
- ~2013 DNN-HMM
  (Deep Neural Networks)
- ~Current LSTM-RNN layers
  - 5 convolution layers
  - 3 fully connected layers



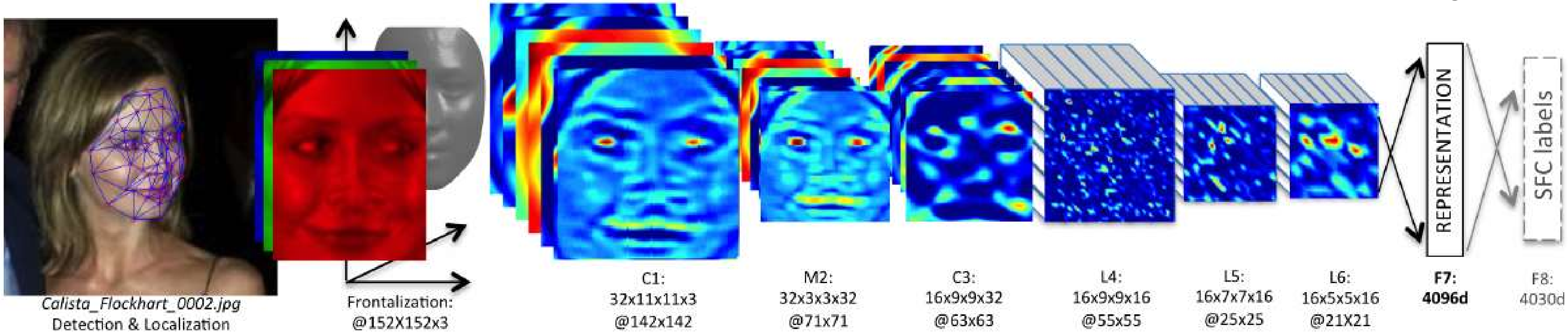
출처: Figure #22

16.8 Applications (4/8)

5) Face Recognition

- Minimizing cross-entropy loss for each learning sample: $L = -\log p_k$
- Standard back-propagation algorithm and stochastic gradient descent (SGD)
- Employing the ReLU activation function: $\max(0, x)$
- Sparsity: 75% of feature elements have 0 value
- Guarantee of sparcity using dropout (generalization method)

출처: Figure #23



16.8 Applications (5/8)

6) Google Neural Machine Translation

➢ NeuralMT, end-to-end learning, recurrent neural networks, GPUs

출처: Figure #24

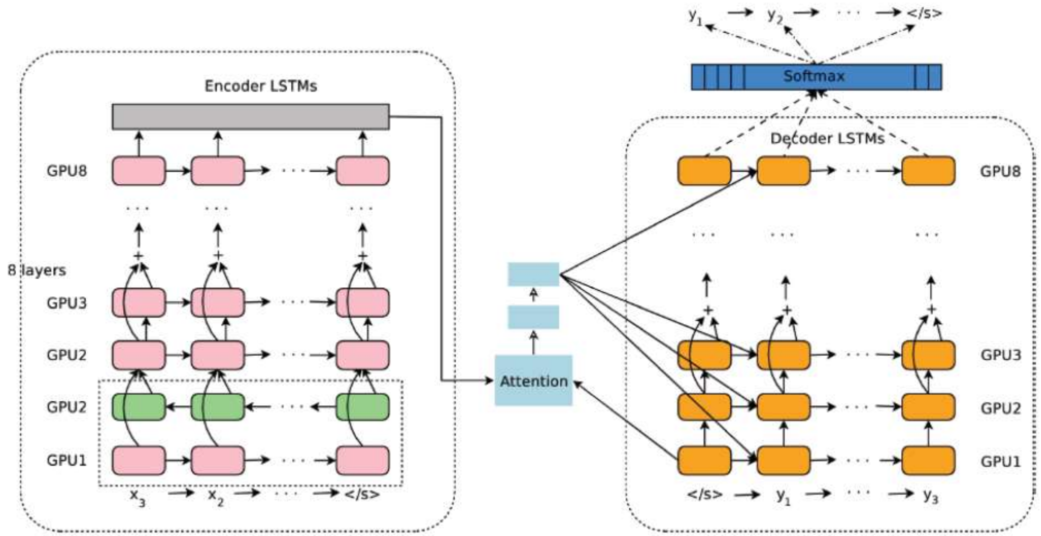
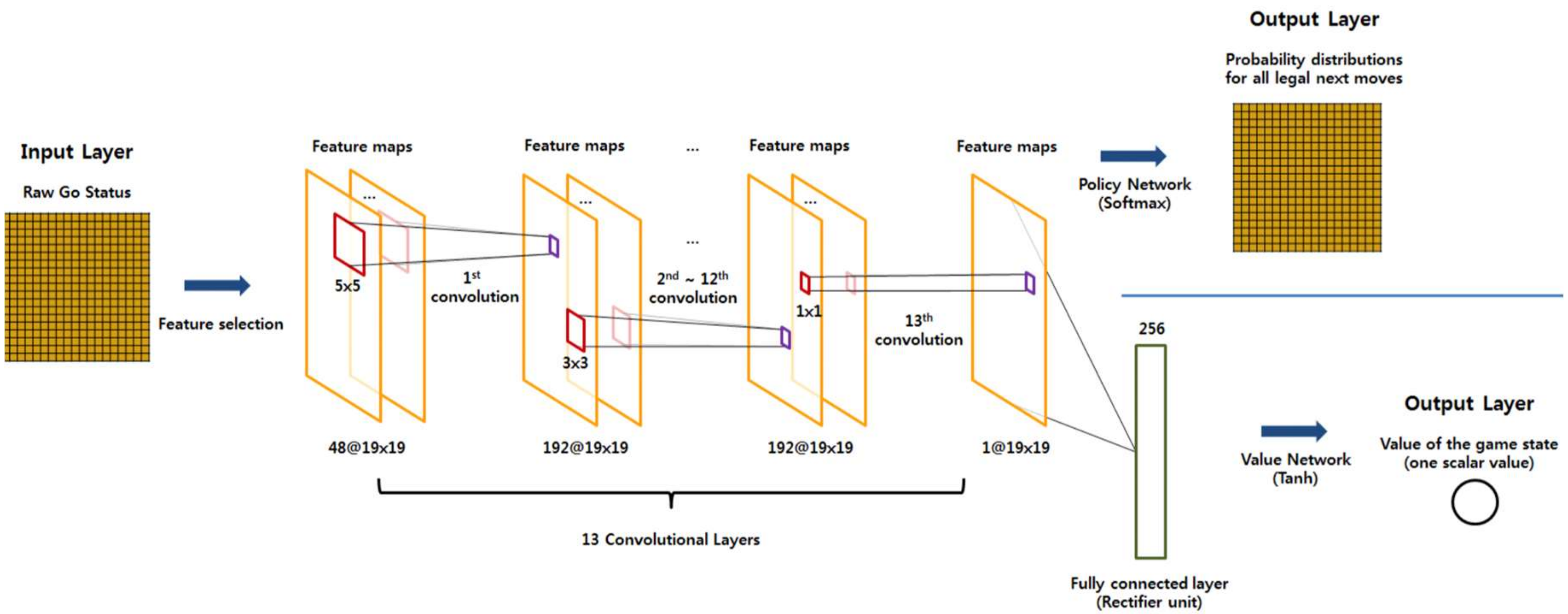


그림 10.14 신경기계번역의 예, 구글의 신경기계번역 시스템 (Wu et al., 2016)

16.8 Applications (6/8)

7) AlphaGo

- Deep learning + reinforcement learning, deep Q-network, value network, policy network, policy gradient

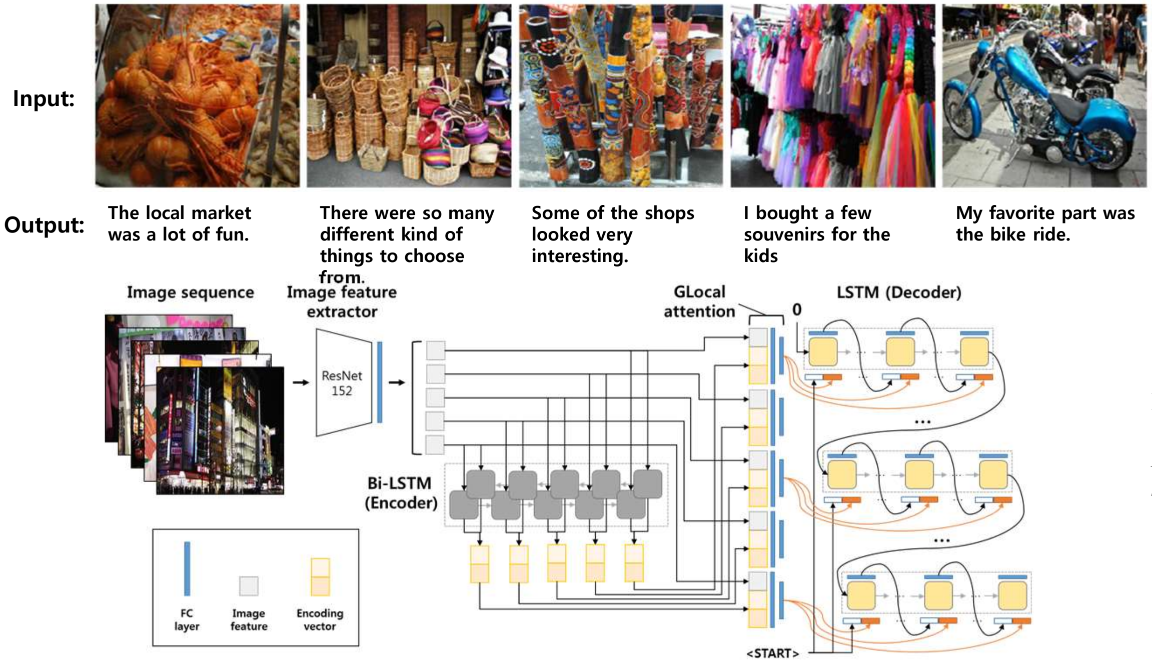


출처: Figure #25

16.8 Applications (7/8)

8) Visual Storytelling (VIST)

- Visual story = photo sequence + sentence sequence
- SNU 1st Place in VIST Challenge 2018



GLAC Net: GLocal Attention Cascading Networks for Multi-image Cued Story Generation, T. Kim et al. *NAACL 2018 Workshop on Storytelling, 2018.*

출처: Figure #26

16.8 Applications (8/8)

9) Image Synthesis by GAN (Generative Adversarial Nets): Videos

Summary

Methods for learning functions represented by deep computational graphs

- Neural networks represent complex nonlinear functions with a network of parameterized linear-threshold units.
- The back-propagation algorithm implements a gradient descent in parameter space to minimize the loss function.
- Deep learning works well for visual object recognition, speech recognition, natural language processing, and reinforcement learning in complex environments.
- Convolutional networks are particularly well suited for image processing and other tasks where the data have a grid topology.
- Recurrent networks are effective for sequence-processing tasks including language modeling and machine translation.

References

Figures

#1, 2, 7, 10, 24 장병탁. 장교수의 딥러닝. 홍릉과학출판사, 2017.

#3, 4, 5, 6, 9, 11, 15, 16 Stuart J. Russell and Peter Norvig (2021). Artificial Intelligence: A Modern Approach (4rd Edition). Pearson

#8 LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444

#12 Stanford Lecture CS231n

#13 LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

#14 He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

#15 Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.

#17 Colah’s blog. Understanding LSTM Networks

#18 Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).

References

Figures

#19 Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

#20 Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).

#21 Delivering competitive advantage to companies creating and using vision and machine learning technology https://www.bdti.com/InsideDSP/2017/06/29/Microsoft

#22 https://scienceon.kisti.re.kr/srch/selectPORSrchReport.do?cn=TRKO201800042056

#23 Taigman, Yaniv, et al. "Deepface: Closing the gap to human-level performance in face verification." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

#25 Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016)

#26 Kim, Taehyeong, et al. "Glac net: Glocal attention cascading networks for multi-image cued story generation." arXiv preprint arXiv:1805.10973 (2018).