Introduction

1) What is Deep Learning?

Artificial Intelligence is mimicing human intelligence

Machine Learning is data-driven approach.

when we want to solve a problem, we use dataset to find a solution

Deep learning is a part of Machine Learning.

Deep learning uses Neural Networks to solve a problem

p.38 Cross entropy loss

Linear regressio

- Given a dataset, our goal is to choose the weights **w** and the bias b that make our model's predictions fit the true price observed in that data as closely as possible.

Model

Input → d features. (area와 age를 집 가격을 예측한다면 Features=[area, age])

$$\hat{y} = w_1 x_1 + \cdots + w_d x_d + b$$

- w: weight
- b: bias

⇒ Affine transformation of input features

- Weight : a value that adjust the influence of each input feature on the output
- bias : a constant value that adjusts the output of the model, allowing it to shift away from the origin.

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b \quad \because [\ ] = d \begin{bmatrix} \mathbf{w}^T \\ \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ d \end{bmatrix}$$

Weight vector ↵ ↳ Feature vector

Single instance가 아닌 n개 examples (entire data set)

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

↳ Design matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ (row: Point(observations), column: features)

⇒ Broadcasting 기능 사용

$\therefore \hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$

- Given the features of a training dataset $\mathbf{X}$ and corresponding labels $\mathbf{y}$, the goal of linear regression is to find the weight vector $\mathbf{w}$ and the bias b With the lowest prediction error.

Model 평가 → Loss Funtion (실제값 y 와 예측값 $\hat{y}$의 차이 측정)

; 한개 example에 대해

Squared error $l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$ – 미분할때 계산 편하게 $\frac{1}{2}$ 사용

n개 example (entire dataset) → average (or sum) the losses

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left(\underbrace{\mathbf{w}^T \mathbf{x}^{(i)} + b}_{\hat{y}} - y^{(i)}\right)^2$$

loss를 최소화하는 $(\mathbf{w}^*, b^*)$ 찾기

$$\mathbf{w}^*, b^* = \underset{\mathbf{w}, b}{\operatorname{argmin}}\ L(\mathbf{w}, b)$$

Analytic Solution

Linear regression은 we can find optimal parameters analytically

minimize $\|y - Xw\|^2$ — squared error의 합 목표

↳ 전체 제곱 오차, (제곱 오차들의 합)

↓

w에 대해 편미분 $\partial_w \|y - Xw\|^2 = 2X^T(Xw - y) = 0$

↓

$w^* = (X^TX)^{-1}X^Ty$

w^*값 찾기.

But, 복잡해서 Analytical 이라고만!

Stochastic Gradient Descent

Gradient Descent

loss function을 줄이는 방향으로 parameter를 update 하며 error를 줄이는 방법.

→ 근데 너무 느림.

⇒ 극단적으로 전체의 데이터 포인트 중 1개만 사용하는게 SGD

→ 좋은 방법으로 mini batch SGD가 있음.

전체 data set에서 몇개만 선택해 mini batch를 만듦.

$$(w, b) \leftarrow (w, b) - \frac{\eta}{|B|} \sum_{i \in B} \partial_{(w,b)} l^{(i)}(w, b)$$

B: minibatch (샘플의 집합)

$|B|$: minibatch의 크기 (샘플 갯수)

η: learning rate (얼마큼 parameter를 크게 작게 바꿀지)

∴ SGD iteratively reduces the error by updating parameters using only a single example at a time in the direction that lowers the loss function.

Maximum Likelihood Learning

$x \sim N(\mu, \sigma^2)$ mean: μ, Variance σ^2

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

Error가 Normal dist를 따른다 가정.

$$y = w^T x + b + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2)$$

Likelihood function (특정 x에 대해 y가 나올 확률)

$$p(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - w^T x - b)^2\right)$$

$y - (w^T x + b)$

$\hat{y} \rightarrow E[y|x]$ (μ 대신 들어감)

※ The best values of w, b are those that maximize the likelihood of the entire dataset.

For entire dataset,

$$p(Y|X) = \prod_{i=1}^{n} p(y^{(i)}|x^{(i)})$$

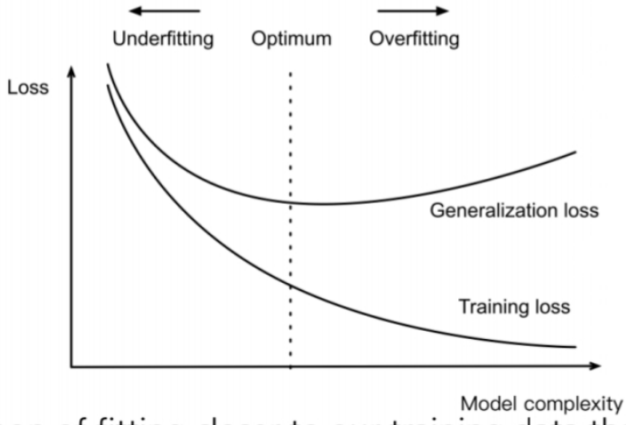
○ likelihood를 maximize하는 대신, Negative log-likelihood를 minimize할 수 있다.
(계산 쉬우려고)

$$-\log p(Y|X) = \sum_{i=1}^{n} \left(\frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(y^{(i)} - w^T x^{(i)} - b)^2\right)$$

If σ is fixed, first term 무시 가능

- In fact, as the solution does not depend on σ, minimizing the mean squared error is equivalent to the maximum likelihood estimation of a linear model under the assumption of additive Gaussian noise.

Underfitting or Overfitting



- The phenomenon of fitting closer to our training data than to the underlying distribution of called **overfitting**, and techniques for combatting overfitting are often called **regularization** methods.

o Underfitting

: underfitting occurs when a model is too simple to capture underlying patterns in the training data. In this case, both the training loss and generalization loss are high, resulting in poor model performance.

o Overfitting

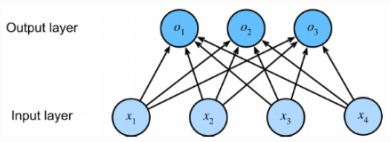
: overfitting occurs when a model is too complex and fits the training data closely, resulting in poor performance on new data. In this case, the training loss is low, but the generalization loss is high, leading to decreased predictive power.

Classification

ex) $o_{1,2,3}$ = 개, 고양이, 닭

$x_{1,2,3,4}$ = 몸무게, 털 길이, 날개 길이, 귀 형태

Model



개의 카테고리를 분류

$$o_1 = x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1$$

$$o_2 = x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2$$

$$o_3 = x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3$$

class 갯수 ↑ ↑ d features.

$$\Rightarrow \quad \mathbf{o} = W\mathbf{x} + \mathbf{b}, \text{ where } W \in \mathbb{R}^{3\times 4},\ \mathbf{b} \in \mathbb{R}^3$$

Softmax

: In the classification setting, output follows a categorical distribution (i.e., nonnegative and sum up to one.)

⇒ Softmax utilizes exponential function to model categorical dist,

$$P(y=i) \propto \exp(o_i)$$

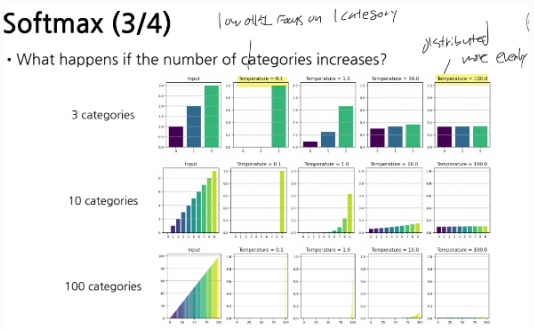
$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}), \text{ where } \hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)}$$

⇒ 여기에 temperature τ 를 더해서 Softmax의 output 조절

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}), \text{ where } \hat{y}_i = \frac{\exp(o_i/\tau)}{\sum_j \exp(o_j/\tau)}$$

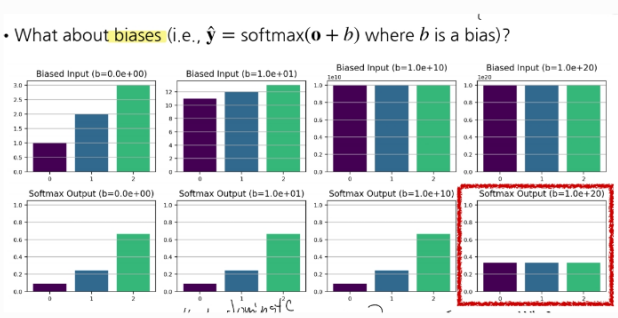
lower τ : sharper distribution (뚜렷해짐)

higher τ : more even distribution



biases추가 $\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}+\mathbf{b})$

b가 커지면서 Input이 바뀌지만 Class들의 확률은 바뀌지 않음.



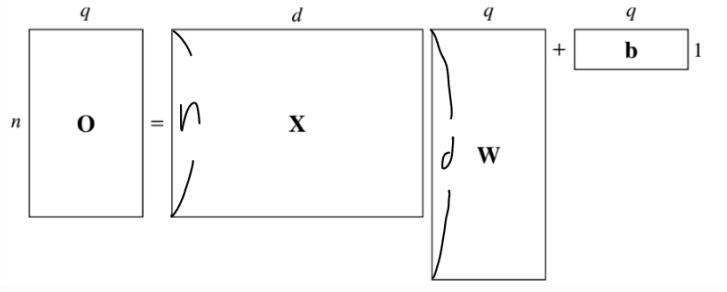
Vectorization (in minibatches) ⇒ 여러 이미지를 한번에!
— 미니배치 데이터셋

$$O = XW + lb$$

$$\hat{Y} = \text{softmax}(O)$$ — 최종 각 클래스의 확률.

where $O \in \mathbb{R}^{n \times q}$, $X \in \mathbb{R}^{n \times d}$, $W \in \mathbb{R}^{d \times q}$, $lb \in \mathbb{R}^{1 \times q}$, $\hat{Y} \in \mathbb{R}^{n \times q}$

n : 이미지 샘플 갯수
q : 클래스 수.



$\hat{Y} =$  ← 이미지 i의 고양이일 확률, 개일 확률, 새일 확률

Loss Function

Softmax의 결과 → $\hat{y}$: condition probabilities of each class, given input x

(e.g. $\hat{y}_1 = P(y = \text{cat} \mid x)$)

Likelihood

$$P(Y \mid X) = \prod_{i=1}^{n} P(y^{(i)} \mid x^{(i)})$$

↓ negative log-likelihood

$$-\log P(Y \mid X) = \sum_{i=1}^{n} -\log P(y^{(i)} \mid x^{(i)}) = \sum_{i=1}^{n} l(y^{(i)} \mid \hat{y}^{(i)})$$

where $l(y, \hat{y}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j$ ⇒ Cross-entropy loss

Cross-Entropy Loss ← Essential of classification prob

$$l(y, \hat{y}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j = -\sum_{j=1}^{q} y_j \log \frac{\exp(o_k)}{\sum_{k=1}^{q} \exp(o_k)} = \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j$$

$$= -\sum_{j=1}^{q} y_j \log \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} = \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j$$

$$\hat{y}(\text{j 클래스의 확률}) = \frac{\text{j 클래스의 exp 값}}{\text{모든 클래스의 exp 값}}$$

$\text{lse}(o) = \log \sum_{k=1}^{q} \exp(o_k)$ is

↓

log-softmax

$$\max(o) \le \text{lse}(o) \le \max(o) + \log q$$

⇒ give upper bound (최대값 상한 설정)

The derivative of $l(y, \hat{y})$ w.r.t o_j:

$$\partial_{o_j} l(y, \hat{y}) = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \text{softmax}(o_j) - y_j$$

$\frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} = \hat{y}_j$

⇒ difference between $\hat{y}$ and y

$\hat{y}$: Probability assigned by model

y: label

Information Theory

: to quantify the amount of information contained in data.

For discrete dist P, entropy is

$$H[P] = \sum_j -P(j) \log P(j) = \mathbb{E}[-\log P]$$

확률 분포의 불확실성. -

=) expected surprisal (i.e., lower probability, greater surprisal)

negative log prob (확률과 반비례)

Cross-entropy from P to Q

$$H(P, Q) = \sum_j -P(j) \log Q(j)$$

Linear model (affine transformation) 에선 input 과 output의 관계가 linearity 라는 가정을 함.

⇒ But, 실제에선 대부분의 문제가 Non-linear 하다.

⇒ 이걸 극복하기 위해, find a suitable representation of inputs,

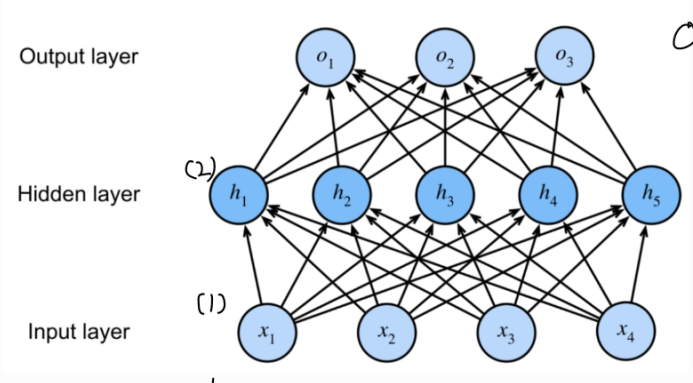
↓

input을 더 적절한 형태로 변형. (원본에 제곱이나 로그등을 씌워서 새로운 input 만들기)

on top of which a single linear model would suffice

↓

변형된 input에 linear model 적용



One hidden layer MLP.

→ H : h hidden units

$X \in \mathbb{R}^{n \times d}$: mini-batch of n examples with d inputs (features)

$H \in \mathbb{R}^{n \times h}$: outputs of the hidden layer. h hidden units

$$\Rightarrow H = XW^{(1)} + b^{(1)}$$
$$O = HW^{(2)} + b^{(2)}$$

However!

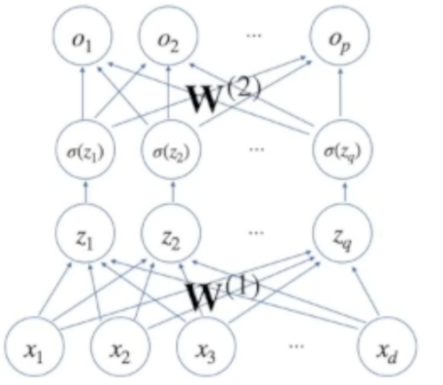
$$O = (XW^{(1)} + b^{(1)})W^{(2)} + b^{2} = XW + b$$

→ hidden layer를 추가했어도 최종 결과에 변화X.

Nonlinear activation function σ 필요.

- Popular choice is ReLU (Rectified Linear Unit), $\sigma(x) = \max(0, x)$

$$H = \sigma(XW^{(1)} + b^{(1)})$$
$$O = HW^{(2)} + b^{(2)}$$

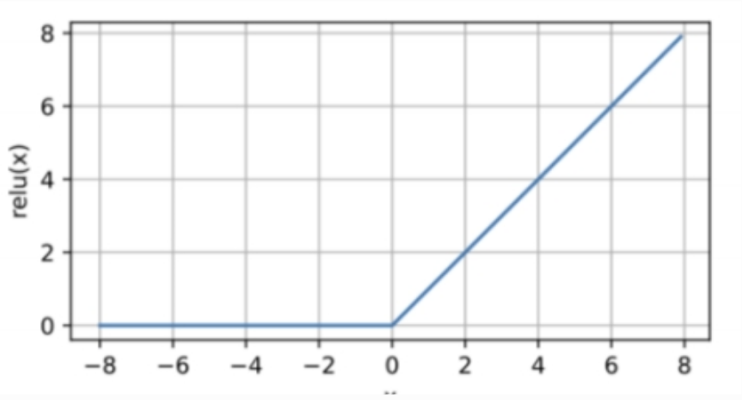


Activation functions.

; differentiable operators, adding non-linearity

여러개가 있는데 각각 장,단점이 있음 ⇒ Activate function을 고르는 것에 따라 How well network perform이 달라짐.

ReLU Function: $ReLU(x) = \max(x, 0)$

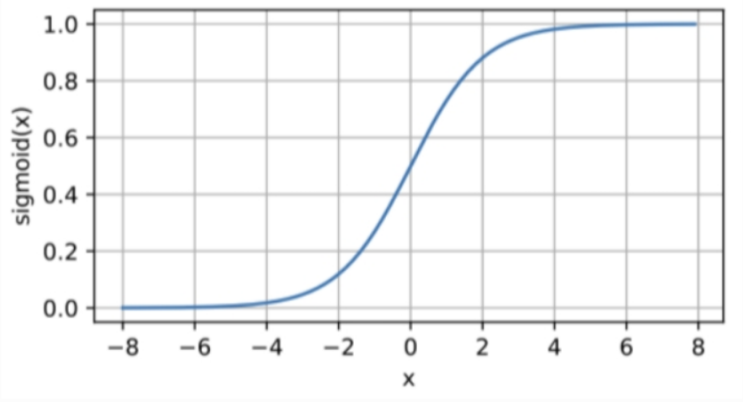


- Vanishing Gradient issue를 다소기 함음.

✱ However, "Dead relu" 문제 발생가능

⇒ Input이 negative 면(bias가 large negative라던지, 항상 output이 0.

Sigmoid Function: $Sigmoid(x) = \frac{1}{1+\exp[-x]}$



- Output이 (0,1) 사이에 존재 (확률로 표현 가능)

✱ - gradient vanishing problem 발생가능 (input이 너무 작으면 gradient도 매우 작아짐)
(or 너무 크면)

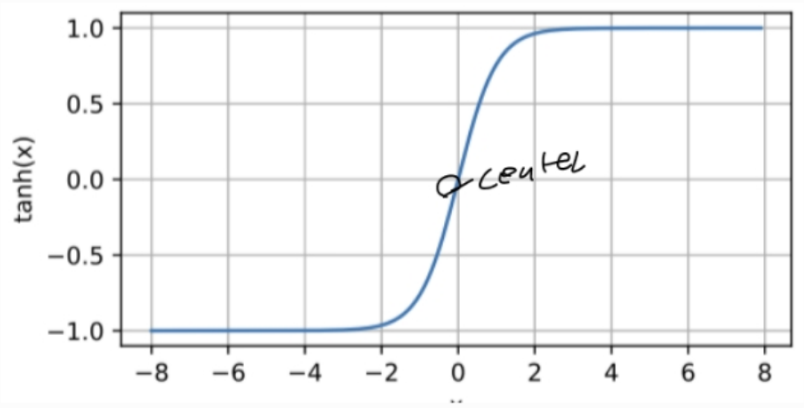
- outputs are not zero-centered.

- useful property

$$\frac{d}{dx} Sigmoid(x) = Sigmoid(x)(1 - Sigmoid(x))$$

⇒ 계산식을

Tanh Function: $\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$



- Output이 (-1, 1) 사이 ⇒ Center 기준 Symmetric

- Vanish gradient problem.

- Useful property

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$

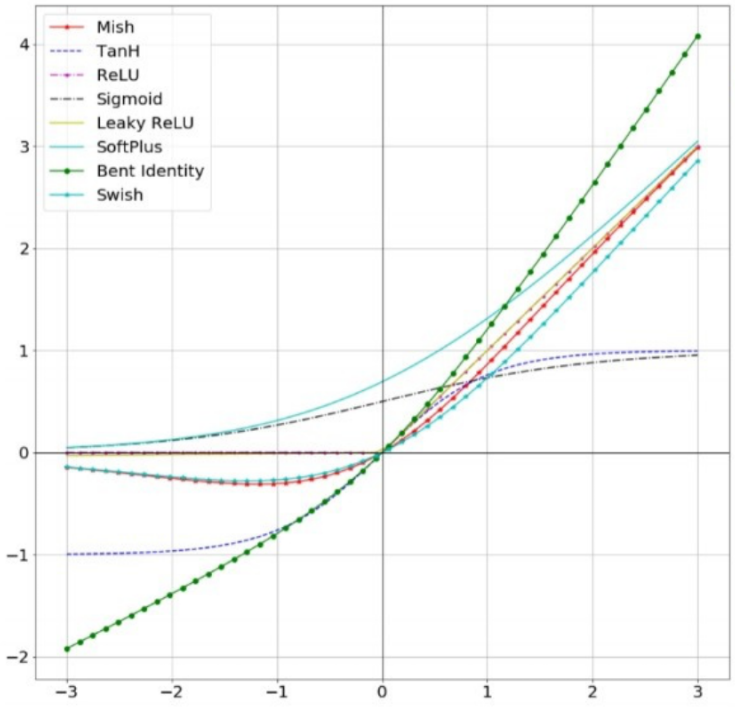
⇒ 계산 쉬움.

그 외

Activation Functions

-Problem에서 크게 벗어나지 않음.

Activation value is always zero



- The **relu** is notorious for the dead relu problem.
- To handle this, the **elu** function was proposed. However, it introduces a longer computation time due to the exponential operation included.

$$\text{elu}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

- The **leaky relu** function also avoids the dead relu problem and is fast. However, we have to tune the parameter α.

$$\text{lrelu}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & x < 0 \end{cases}$$

- The **gelu** function works well in NLP, specifically Transformer models, as it is fast.

$$\text{gelu}(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$$

- The **swish** function is continuous and differentiable at all points. And it works well on standard image datasets (CIFAR or ImageNet) compared to others (relu, lrelu, elu, gelu).

$$\text{swish}(x) = x(1 + e^{-x})^{-1}$$

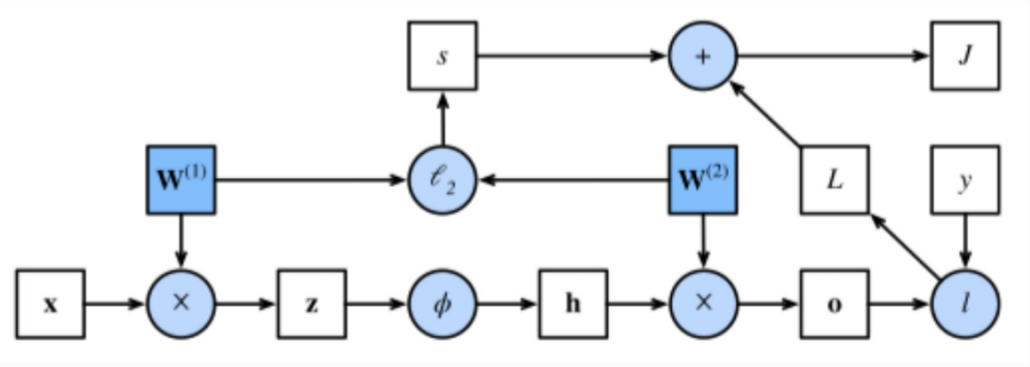
- The **mish** function is C^∞-continuous and approximates identity near the origin. In some experiments, the **mish** works better than **swish** activations.

zero

$$\text{mish}(x) = x\tanh(\text{softplus}(x))$$

Back Propagation

Forward Propagation



1) Input : $x \in \mathbb{R}^d$ (no bias for simplicity) ← $d \times 1$

2) Intermediate Variable : $z = W^{(1)}x \in \mathbb{R}^h$, where $W^{(1)} \in \mathbb{R}^{h \times d}$ (weight parameter of the hidden layer) ← hidden layer의 node 갯수

3) Hidden activation vector : $h = \phi(z) \in \mathbb{R}^h$ ⇒ hidden layer
↳ activation function

4) Hidden layer output (MLP 최종 output)
: $o = W^{(2)}h \in \mathbb{R}^q$, where $W^{(2)} \in \mathbb{R}^{q \times h}$
(weight parameter of the hidden layer)

5) Loss term (for a single data) : $L = l(o, y)$ ← Loss function

6) Regulation term : $s = \frac{\lambda}{2}(\|W^{(1)}\|_F^2 + \|W^{(2)}\|_F^2)$ ← 정규화의 정도
⇒ Prevent over fitting

$\|W\|_F$ ⇒ Frobenius Norm
$\sqrt{w_1^2 + w_2^2 + w_3^2 + \cdots}$

7) Model's regularized loss : $J = L + s$

⇒ J를 Minimize 하는게 목표

Back propagation

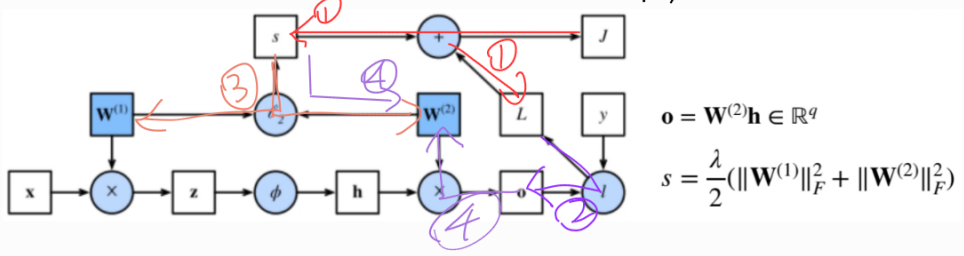
: Calculating the gradient of neural network parameters (to update weights)

→ Chain rule을 사용해 output에서 input으로 탐색

Chain rule

$Y = f(X),\ Z = g(Y)$

$$\frac{\partial Z}{\partial X} = \Pi\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X}\right)$$



$W^{(2)}$

1) Calculate the gradients of the objective $J = L + S$ w.r.t L and S.

Loss term regularization term

$$\frac{\partial J}{\partial L} = 1, \quad \frac{\partial J}{\partial S} = 1$$

$$\Leftarrow \frac{\partial (L+S)}{\partial L} = 1$$ (상수 취급)

2) Compute the gradient of J w.r.t the output layer o : o가 J에 얼마나 영향을 미치는가

$$\frac{\partial J}{\partial o} = \Pi\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial o}\right) = \frac{\partial L}{\partial o} \in \mathbb{R}^q$$

$\Pi\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial o}\right) = \frac{\partial J}{\partial L} \times \frac{\partial L}{\partial o}$

$\frac{\partial J}{\partial L}$: 1

$\frac{\partial L}{\partial o}$: 각 클래스별 하나씩. ⇒ $\mathbb{R}^q$

⇒ chain rule

3) Calculate the gradients of S w.r.t Parameters $W^{(1)}, W^{(2)}$

$$\frac{\partial S}{\partial W^{(1)}} = \lambda W^{(1)}, \quad \frac{\partial S}{\partial W^{(2)}} = \lambda W^{(2)}$$

$$S = \frac{\lambda}{2}\left(\|W^{(1)}\|_F^2 + \|W^{(2)}\|_F^2\right)$$

$$\Leftarrow$$ 제곱해 편미분

$$\frac{\partial S}{\partial W^{(1)}} = \frac{\lambda}{2} \cdot 2W^{(1)} + 0$$

4) Calculate the gradient $\frac{\partial J}{\partial W^{(2)}} \in \mathbb{R}^{q \times h}$

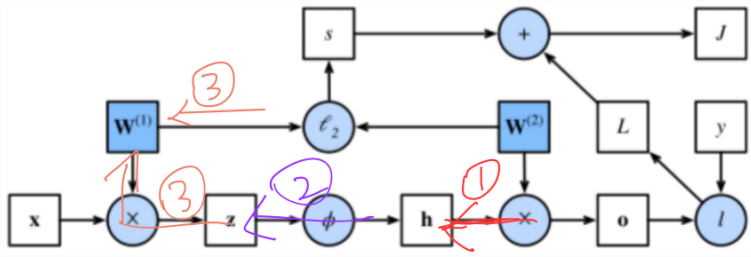
곱셈에서 chain rule 적용하기 위해

$$\frac{\partial J}{\partial W^{(2)}} = \Pi\left(\frac{\partial J}{\partial o}, \frac{\partial o}{\partial W^{(2)}}\right) + \Pi\left(\frac{\partial J}{\partial S}, \frac{\partial S}{\partial W^{(2)}}\right) = \frac{\partial J}{\partial o} h^T + \lambda W^{(2)}$$

$\frac{\partial J}{\partial o}$: $\frac{\partial L}{\partial o}$ / $\frac{\partial J}{\partial S}$: 1 / $\frac{\partial S}{\partial W^{(2)}}$: $\lambda W^{(2)}$

$o = W^{(2)}h$

$\frac{\partial o}{\partial W^{(2)}} = h$



$W^{(1)}$

1) To obtain the gradient w.r.t $W^{(1)}$,
we need to continue back propagation along the output layer to hidden layer

$$\frac{\partial J}{\partial h} = \prod\left(\frac{\partial J}{\partial o}, \frac{\partial o}{\partial h}\right) = W^{(2)\top} \frac{\partial J}{\partial o}$$

$\frac{\partial o}{\partial h} = W^{(2)}$, $W^{(2)}$: $h \times q$, $\frac{\partial J}{\partial o}$: $q \times 1$

2) Since the activation function ϕ applies element wise. (원소별)

$$\frac{\partial J}{\partial z} = \prod\left(\frac{\partial J}{\partial h}, \frac{\partial h}{\partial z}\right) = \frac{\partial J}{\partial h} \odot \phi'(z)$$

$h = \phi(z)$

원소별 곱 $\begin{bmatrix} a \\ b \\ c \end{bmatrix}, \begin{bmatrix} d \\ e \\ f \end{bmatrix}$

$a \odot b = \begin{bmatrix} a \times d \\ b \times e \\ c \times f \end{bmatrix}$

3) the gradient $\frac{\partial J}{\partial W^{(1)}} \in \mathbb{R}^{h \times d}$

$$\frac{\partial J}{\partial W^{(1)}} = \prod\left(\frac{\partial J}{\partial z}, \frac{\partial z}{\partial W^{(1)}}\right) + \prod\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial W^{(1)}}\right) = \frac{\partial J}{\partial z} x^\top + \lambda W^{(1)}$$

$z = W^{(1)} x$, $\frac{\partial s}{\partial W^{(1)}} = 1$

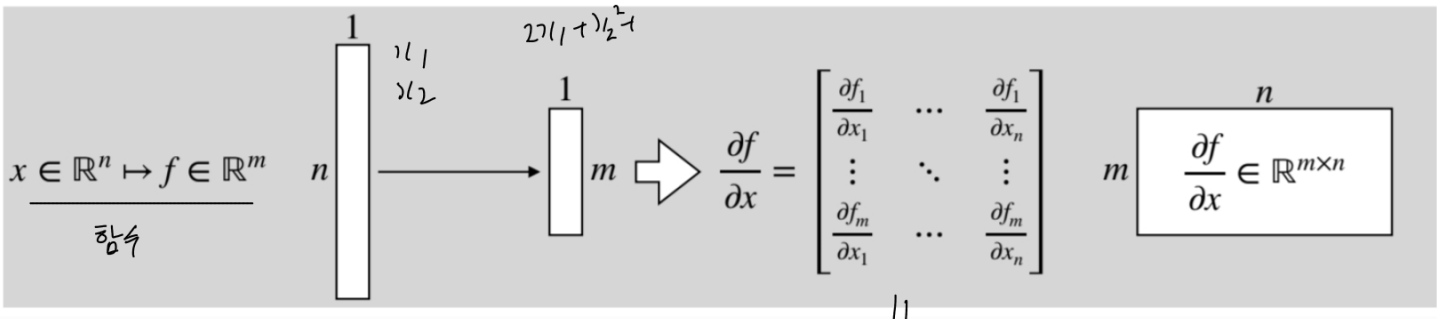
$\frac{\partial z}{\partial W^{(1)}} = x$

$$= \left(\frac{\partial J}{\partial h} \odot \phi'(z)\right) x^\top + \lambda W^{(1)}$$

$W^{(2)\top} \frac{\partial J}{\partial o}$

$$= \left(\left(W^{(2)\top} \frac{\partial L}{\partial o} \odot \phi'(W^{(1)} x)\right) x^\top + \lambda W^{(1)}\right)$$

Matrix Calculus (행렬 미적분학)



Jacobian Matrix

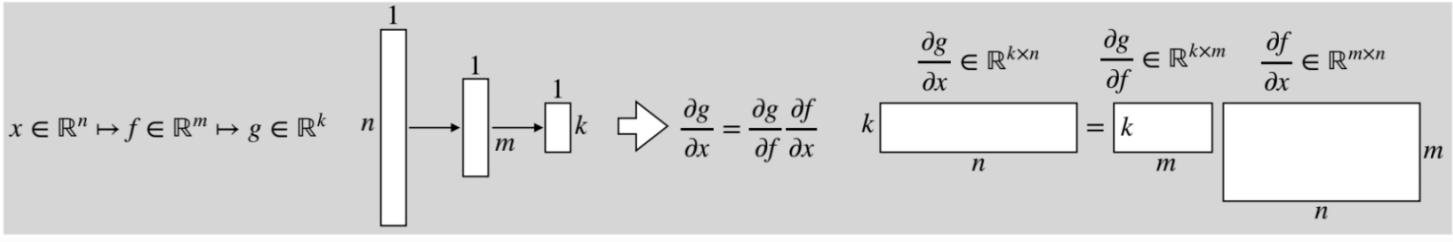
Ex)

$x \in \mathbb{R}^2$, $x = [x_1, x_2]^T$

$f \in \mathbb{R}^3$, $f(x) = [2x_1 + x_2^2, -x_1^2 + 3x_2, 4x_1x_2]^T$

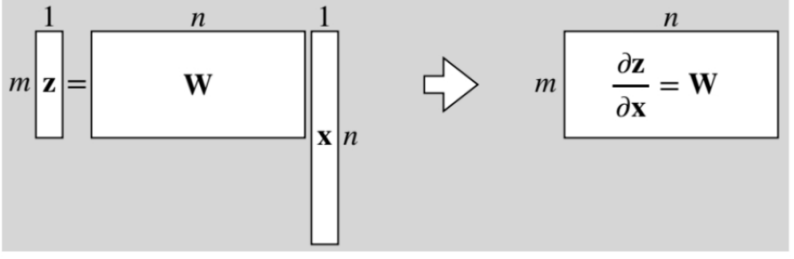
$\frac{\partial f}{\partial x} = \begin{bmatrix} 2 & 2x_2 \\ -2x_1 & 3 \\ 4x_2 & 4x_1 \end{bmatrix}$

$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ $\begin{pmatrix} 2x_1 + x_2^2 \\ -x_1^2 + 3x_2 \\ 4x_1x_2 \end{pmatrix}$



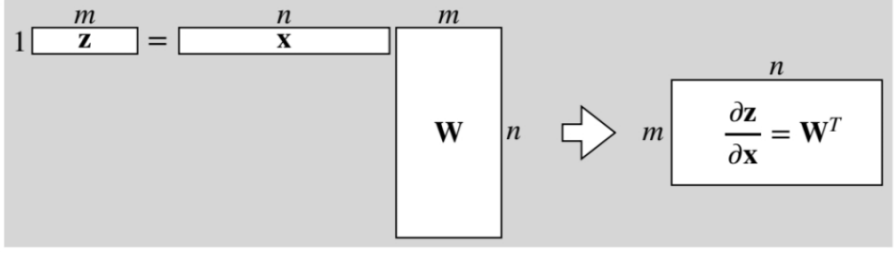
Chain rule

- Matrix times column vector: $\mathbf{z} = \mathbf{W}\mathbf{x}$ where $\mathbf{z} \in \mathbb{R}^{m \times 1}$, $\mathbf{x} \in \mathbb{R}^{n \times 1}$, and $\mathbf{W} \in \mathbb{R}^{m \times n}$. Then, $\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \mathbf{W} \in \mathbb{R}^{m \times n}$.



- Row vector times matrix: $\mathbf{z} = \mathbf{x}\mathbf{W}$ where $\mathbf{z} \in \mathbb{R}^{1 \times m}$, $\mathbf{x} \in \mathbb{R}^{1 \times n}$, and $\mathbf{W} \in \mathbb{R}^{n \times m}$. Then, $\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \mathbf{W}^T \in \mathbb{R}^{m \times n}$.

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



1) $Z = X \cdot \frac{\partial Z}{\partial X} = I \cdot \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}$

identity matrix

2) An elementwise function applied to a vector $Z = f(X)$, wher $z_i = f(x_i)$

$$\frac{\partial Z}{\partial X} = \text{diag}(f'(X)) \quad \text{or} \quad \odot f'(X)$$

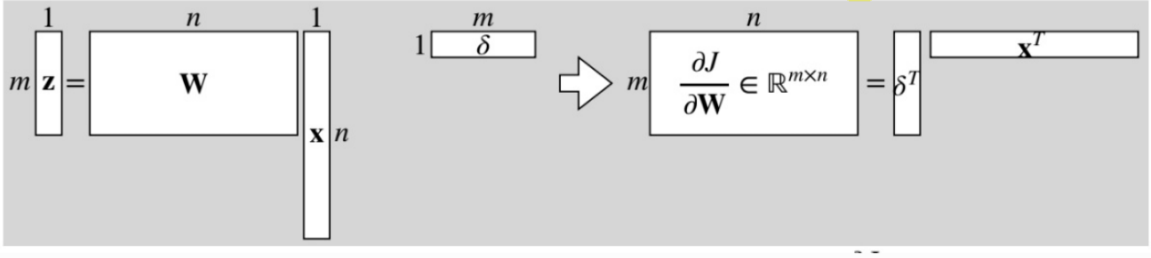
Ex) $X = [x_1, x_2, x_3]^T$

$f(X) = [x_1^2, x_2^2, x_3^2]^T$

$$\frac{\partial Z}{\partial X} = \begin{bmatrix} 2x_1 & & \\ & 2x_2 & \\ & & 2x_3 \end{bmatrix}$$

↳ gradient of loss function J w.r.t Z

- Matrix times column vector: $\mathbf{z} = \mathbf{W}\mathbf{x}$ and $\delta = \frac{\partial J}{\partial \mathbf{z}} \in \mathbb{R}^{1\times m}$. Then, $\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial J}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{W}} = \delta\frac{\partial \mathbf{z}}{\partial \mathbf{W}} = \delta^T\mathbf{x}^T \in \mathbb{R}^{m\times n}$



δ : 1xm

X : nx1

$\delta^T \cdot X^T$ = mx1 · 1xn = mxn

- Row vector times Matrix: $\mathbf{x} \in \mathbb{R}^{1\times n}$, $\mathbf{W} \in \mathbb{R}^{n\times m}$, $\mathbf{z} = \mathbf{x}\mathbf{W} \in \mathbb{R}^{1\times m}$, and $\delta = \frac{\partial J}{\partial \mathbf{z}} \in \mathbb{R}^{1\times m}$.
  - Then, $\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial J}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{W}} = \delta\frac{\partial \mathbf{z}}{\partial \mathbf{W}} = \mathbf{x}^T\delta \in \mathbb{R}^{n\times m}$

nx1 1xm

Cross-entropy loss w.r.t. logits

↳ Multi classification problem.

$J = CE(y, \hat{y})$. $\hat{y} = Softmax(z) \in \mathbb{R}^{1 \times k}$

↓ logit (마지막 output)

$\frac{\partial J}{\partial z} = \hat{y} - y \in \mathbb{R}^{1 \times k}$. k is the number of classes

↓

gradient 0

Vanishing and Exploding Gradients.

- L layers, input x, output ℓ
- each layer l defined by f_l parametrized by $W^{(l)}$

  whose hidden layer output $h^{(l)}$ (i.e. $h^{(0)} = x$)

  $h^{(l)} = f_l(h^{(l-1)})$, $\ell = f_L \circ \cdots \circ f_1(x)$

  ↓

  ex) $h^{(1)} = f_1(x)$

  $h^{(2)} = f_2(h^{(1)}) = f_2(f_1(x))$

- gradient of ℓ w.r.t. $W^{(l)}$

$$\partial_{W^{(l)}} \ell = \partial_{h^{(L-1)}} h^{(L)} \cdots \partial_{h^{l}} h^{(l+1)} \partial_{W^{(l)}} h^{(l)}$$

↳

$\frac{\partial \ell}{\partial W^{(l)}}$

W가 변했을 때 ℓ가 어떻게 변하는지

Other Issues

Nonparametrics. ppt 참고

MLP → parametric (have fixed # of params)

Nonparametric model
: a level of complexity grows as the amount of available data grows.

=) Because neural networks are over-parametrized,
they tend to interpolate the training data (fitting it perfectly) having the same property as nonparamtric models.

Regularization in Neural Networks.
(Prevent overfitting) ppt 참고

- Early Stopping
- weight decay
- Dropout.

Invariance

- Despite Waldo's characteristic outfit, what Waldo looks like does not depend upon where is located. 위치와 상관 X
- We could sweep the image with a Waldo detector that could assign a score to each patch, indicating the likelihood that the patch contains Waldo.
- Convolutional neural networks (CNNs) systematize this idea of spatial invariance, exploiting it to learn useful representations with fewer parameters. → more effective to image Prob

No matter where an object appears in image, it's feature remain consistent

Convolutions (in single processing)

Convolution between 2 functions, f and g

$$(f * g)(x) = \int f(z)\, g(x-z)\, dz$$

⇒ g 함수를 flip, x 만큼 shift.

o Discrete objects in 2-dimensional spaces.

$$(f * g)(i,j) = \sum_a \sum_b f(a,b)\, g(i-a, j-b)$$ ⇒ 이미지 처리와 관련.

f(a,b) ⇒ 이미지의 측정 픽셀 (좌표)

g(i-a, j-b) ⇒ 적용할 필터의 값

Convolutions (in CNNs)

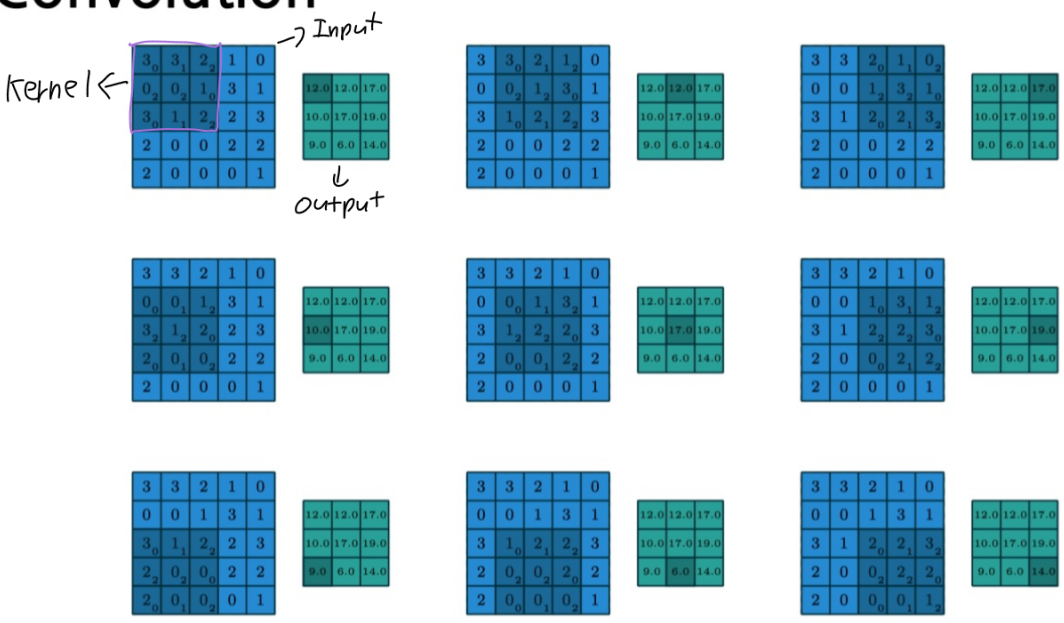
To exploit the translation invariance, an output of a convolutional layer is computed as:

$$[H]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [V]_{a,b}\, [X]_{i+a, j+b}$$

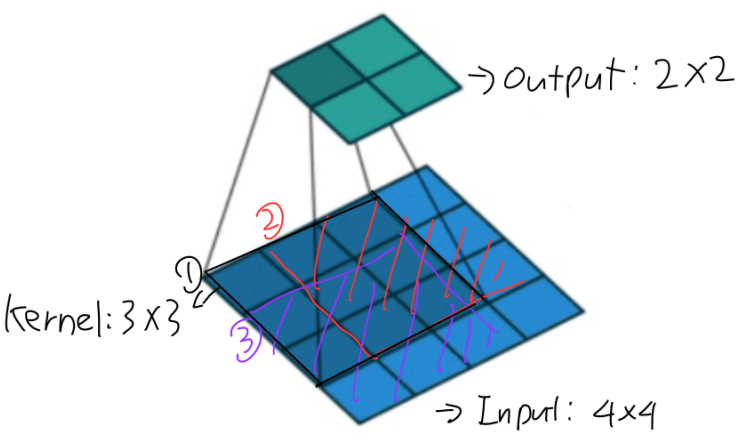
- $[H]_{i,j}$: Output, Pixel position
- u: bias
- Δ: Size of filter.
- $[V]_{a,b}$: Convolution filter (kernel)
- $[X]_{i+a, j+b}$: Input

The number of parameters for the layer : $(2\Delta+1)^2$

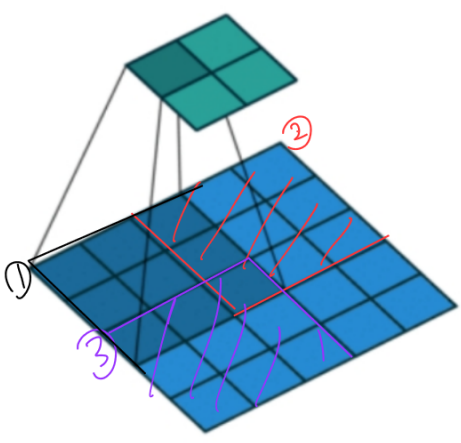
Convolution



Convolution (Padding and Strides)



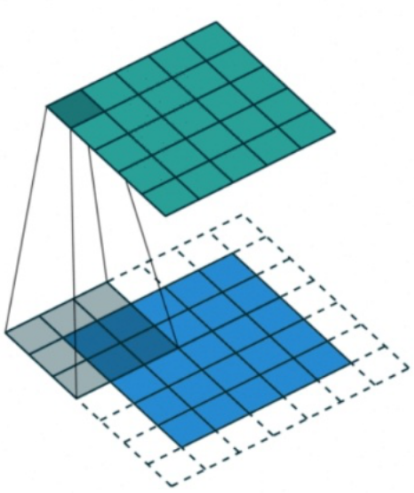
padding=0 and stride=1



padding=0 and stride=2

↳ 한번에 이동하는 갯수

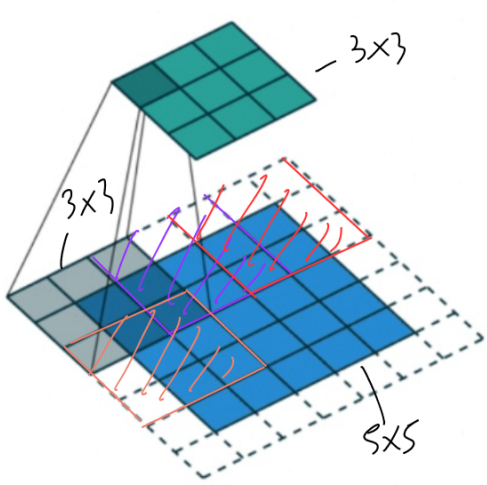
Blue maps are inputs and cyan maps are outputs.



padding=1 and stride=1

↓

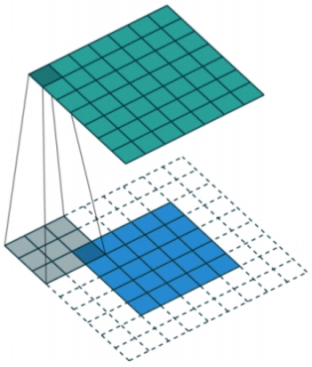
가장자리에 1줄 추가



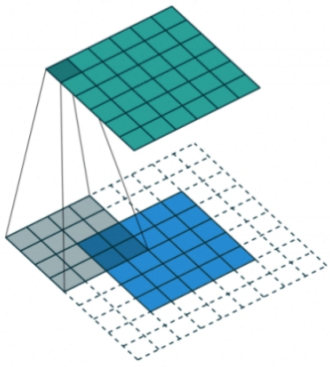
padding=1 and stride=2

↙

2칸씩 이동하니까 output이 작아짐



padding=2 and stride=1

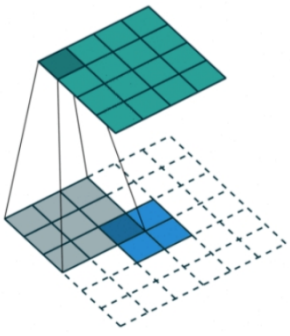


padding=2 and stride=1

Note that the sizes of the output features vary!

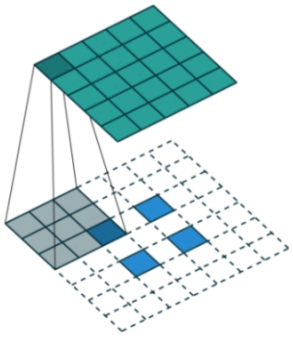
Transposed Convolution (Padding and Strides)

↳ 작은 input으로 큰 output (upsampling)



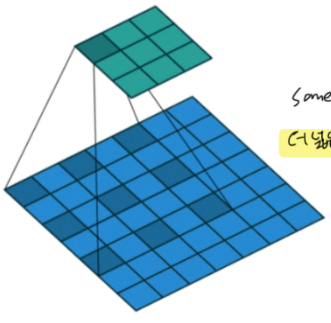
padding=0 and stride=0

↳ 간격



padding=0 and stride=1

Dilated Convolution (Padding and Strides)



Some Input pixel들을 skip한

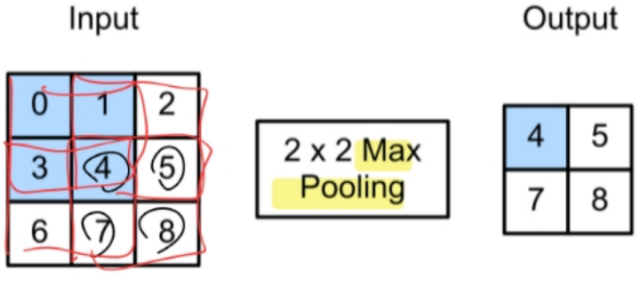
더 넓은 Input 영역을 cover,

without increasing the number of parameters

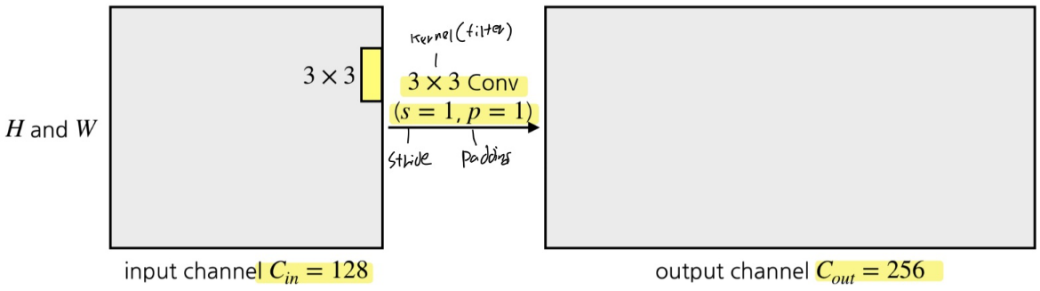
padding=0 and stride=1

Pooling

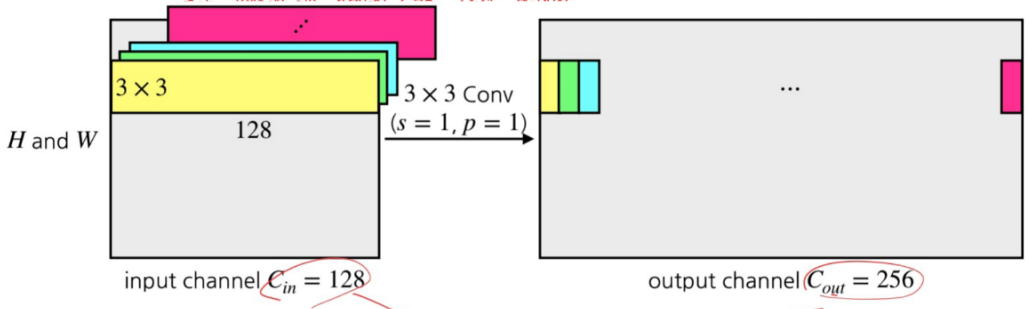
way of reducing dimension



Channel



- What is the number of parameters of this 3×3 convolution layer?

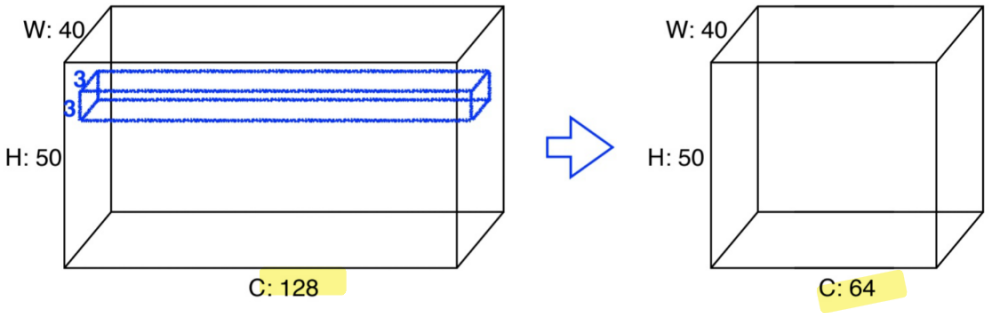


- What is the number of parameters of this 3×3 convolution layer?
  - Each kernel has a dimension of $128 \times 3 \times 3$
  - And we have 256 kernels.
  - The number of parameters is $128 \times 256 \times 3 \times 3 = 294{,}912$

17

$3 \times 3 \times 128 \times 256$

- Padding (1), Stride (1), 3×3 Kernel



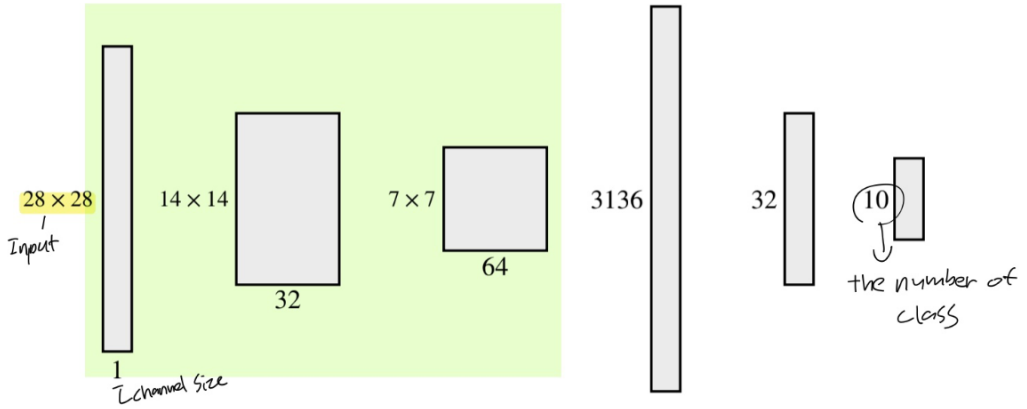
What is the **number of parameters** of this model?

The answer is $3 \times 3 \times 128 \times 64 = 73{,}728$

19

$128 \times 64 \times 3 \times 3$

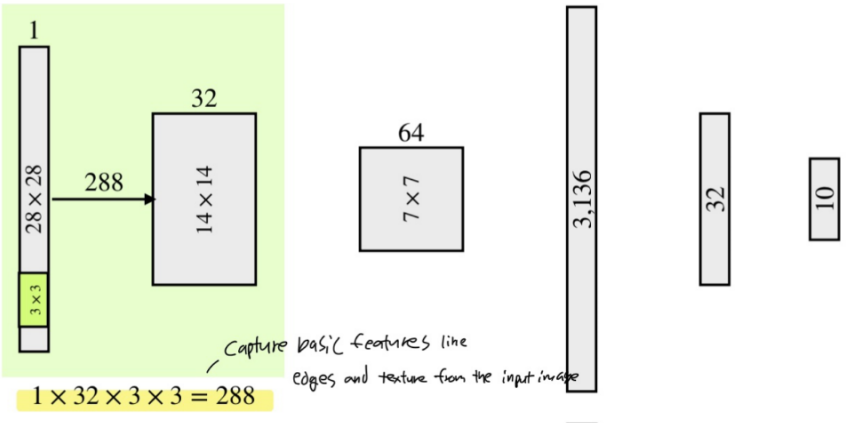
Convolutional Neural Network



- Convolutions with 3×3 kernels, stride is two, and padding is one.
- Max-pooling with 2×2 kernels.

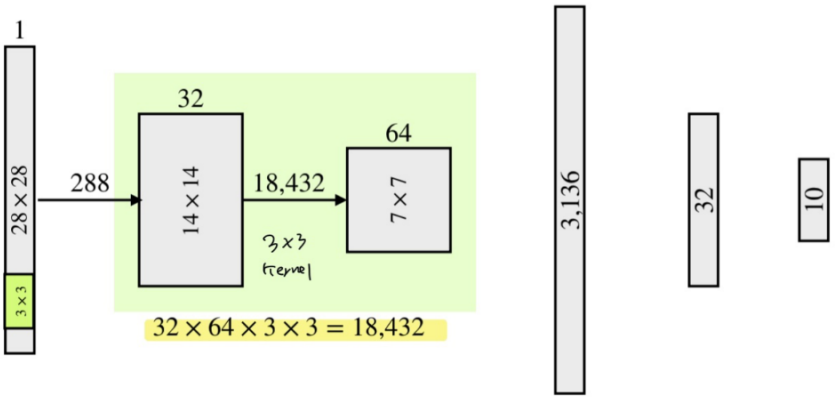
20

Parameter 수 계산

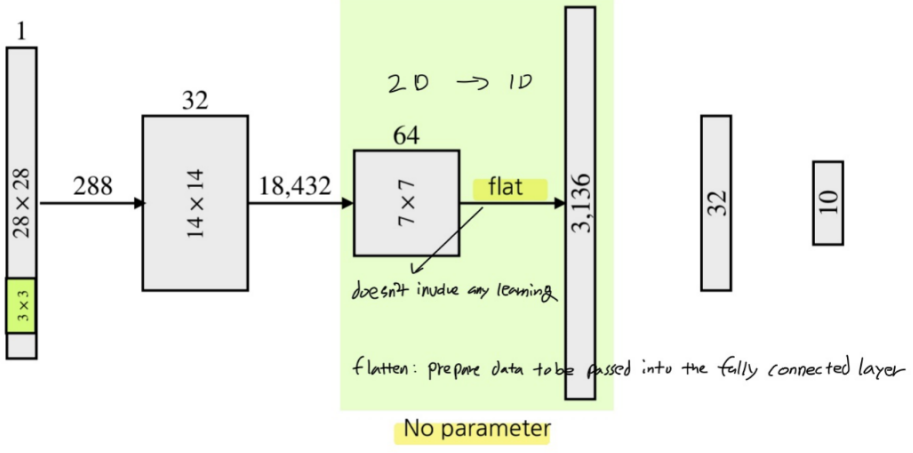


Capture basic features like edges and texture from the input image

$1 \times 32 \times 3 \times 3 = 288$



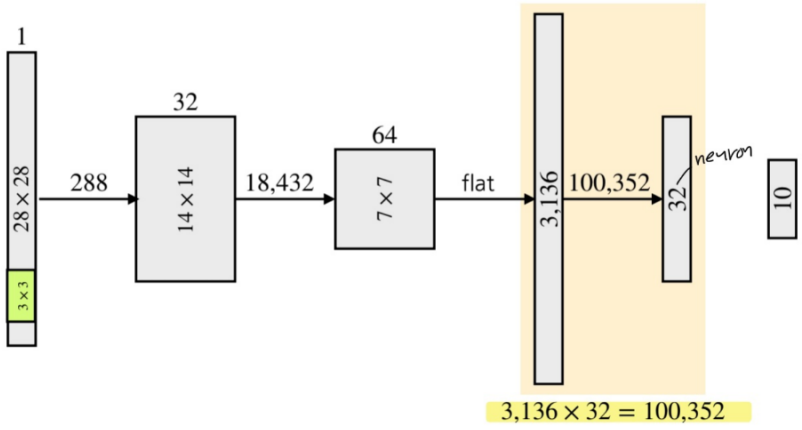
$32 \times 64 \times 3 \times 3 = 18,432$

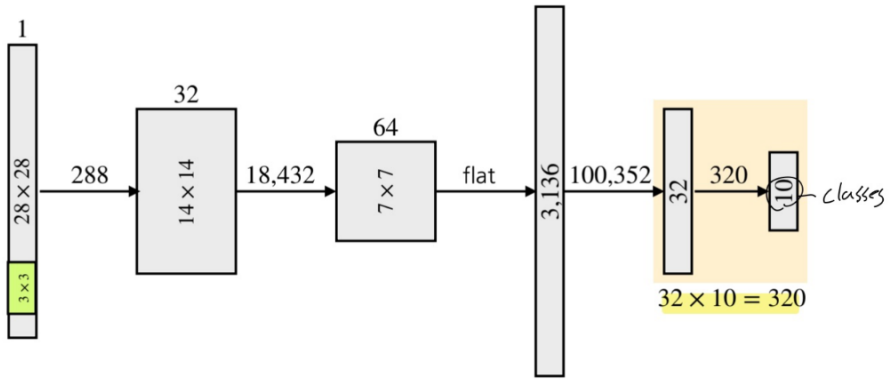


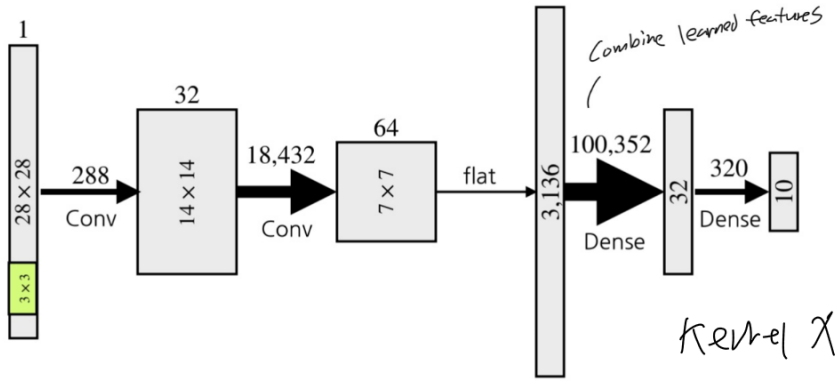
flatten: prepare data to be passed into the fully connected layer

No parameter

Convolutional Neural Network

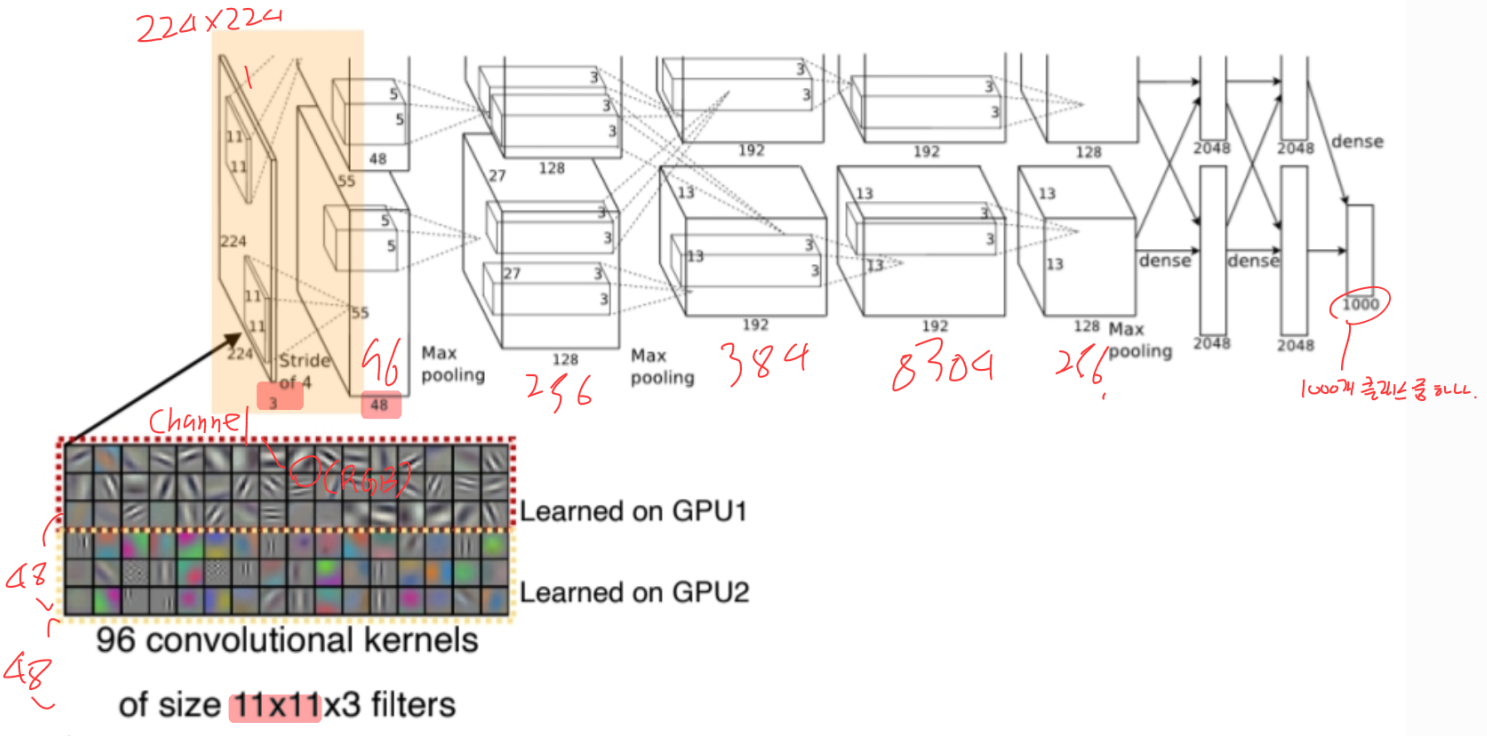






AlexNet

Architecture



Number of parameters

- Kernel: [out_channels x in_channels x kernel_height x kernel_width]

FC (1000)	$4,096 \times 1,000 = 4,096,000$
FC (4096)	$4,096 \times 4,096 = 16,777,216$
FC (4096)	$4,096 \times 6,400 = 26,214,400$
3 × 3 MaxPool, stride 2	~~$256 \times 384 \times 3 \times 3 = 884,736$~~
3 × 3 Conv (256), pad 1	$256 \times 384 \times 3 \times 3 = 884,736$
3 × 3 Conv (384), pad 1	$384 \times 384 \times 3 \times 3 = 1,327,104$
3 × 3 Conv (384), pad 1	$384 \times 256 \times 3 \times 3 = 884,736$
3 × 3 MaxPool, stride 2	
5 × 5 Conv (256), pad 2	$256 \times 96 \times 5 \times 5 = 614,400$
3 × 3 MaxPool, stride 2	
11 × 11 Conv (96), stride 4	$96 \times 3 \times 11 \times 11 = 34,848$
Image (3 × 224 × 224)	

Output channel / Input channel

Implementation Details

- Rectified Linear Unit (ReLU) activation
- Training on Multiple GPUs (2 GPUs) → spread network.
  Put half of kernels on each GPU
- Local Response Normalization
  helps reduce overfitting by normalizing activation
- Overlapping Pooling
  Stride is less than pooling window size,
  meaning the pooling windows overlap by one pixel
- Reducing Overfitting
  - Data Augmentation
  - Dropout

VGGNet

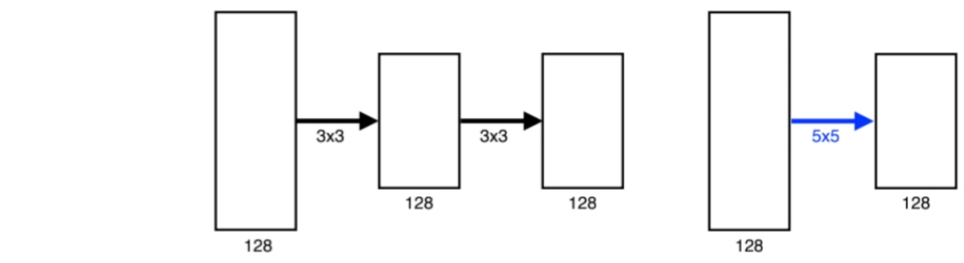
Enable deep networks using very small convolutional filters (3x3)

Basic building blocks.

- Convolutional layer with padding to maintain the resolution
- Relu
- max-pooling to reduce the resolution

key idea is to use multiple convolutions

- Why use 3x3 convolution



GoogleNet

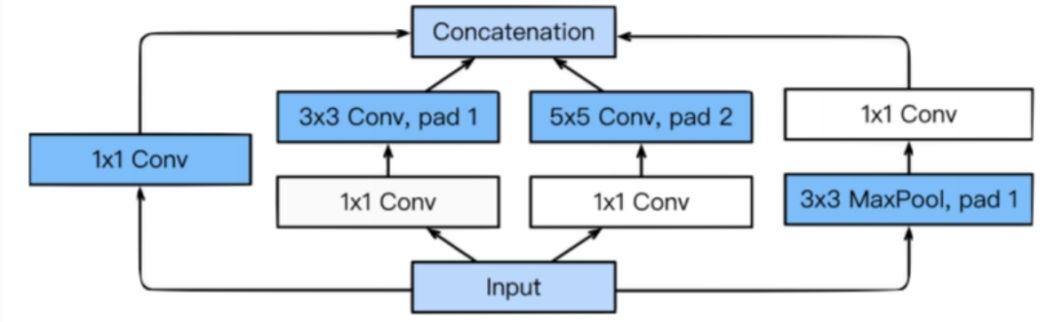
: Multi-Branch Networks

- stem: the first 2-3 convolution
- body: convolutional blocks
- head: maps the feature to the required problem

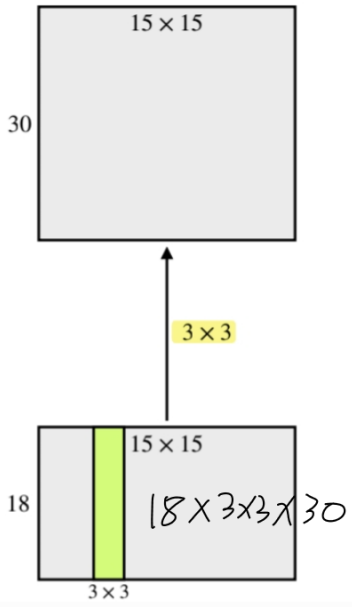
Inception blocks

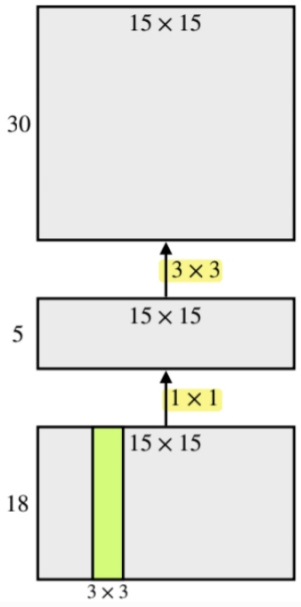
: reduce the number of parameter.

1x1 convolution can be seen as channel-wise dimension reduction



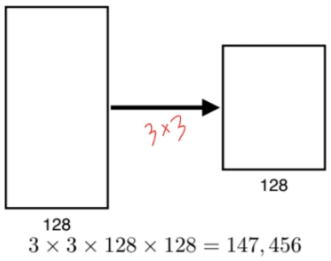
Why adding 1x1 Convolution?



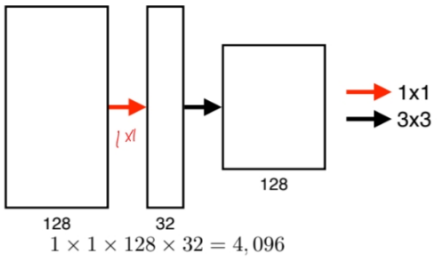


5X3X3X30

18X1X1X5



$3 \times 3 \times 128 \times 128 = 147,456$



$1 \times 1 \times 128 \times 32 = 4,096$

$3 \times 3 \times 32 \times 128 = 36,864$

$4,096 + 36,864 = 40,960$

1x1 convolution enables about 30% reduce of the number of parameters!

Resnet

- But, the **degradation** is not caused by overfitting, and adding more layers leads to higher training errors (and also higher test errors).

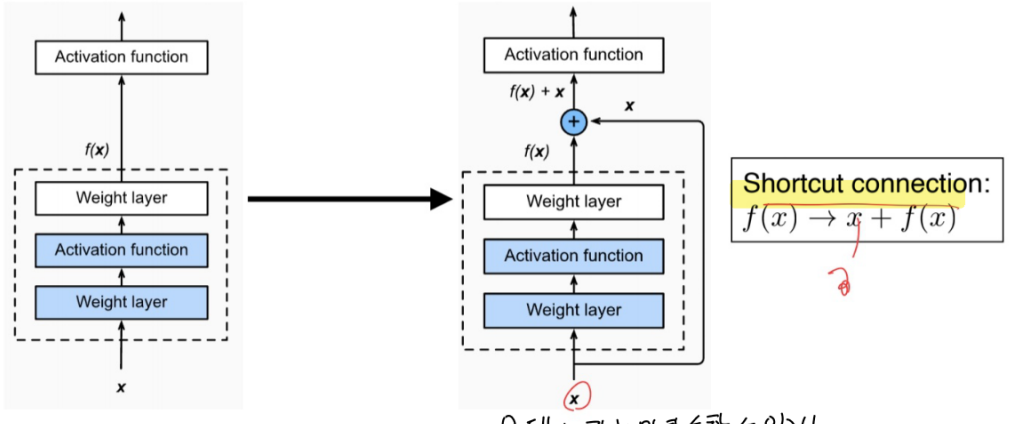
The key idea is that every additional layer should contain identity function as one of its elements.

f(x)=x, 원본을 그대로 가져가게

residual mapping

Residual Blocks

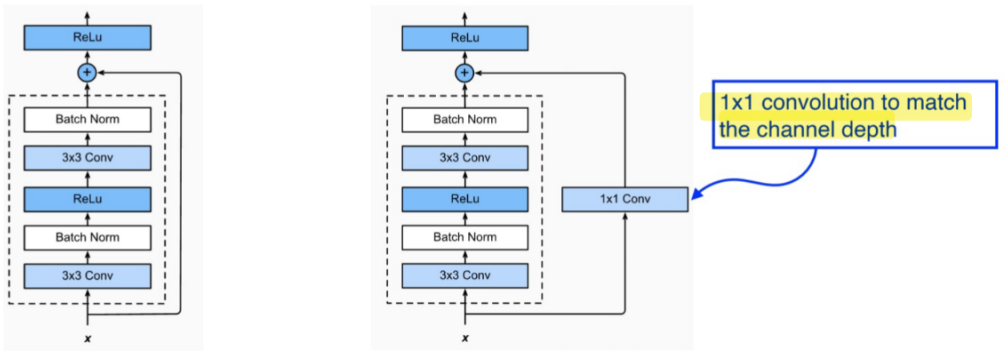
- Add an identity map (Shortcut connection) skip connection
- Shortcut allows the model to pass the input, and only learn the difference.



모델이 잔차만 학습할 수 있게

64

- Add an identity map after nonlinear activations.
- The projected shortcut uses a 1x1 convolution to match the number of channels.

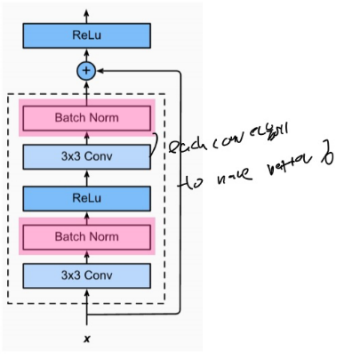


Simple Shortcut Projected Shortcut

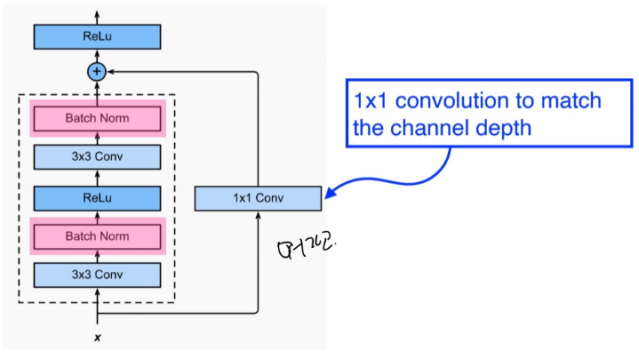
65

Residual Blocks

- Batch normalization after convolutions.



Simple Shortcut

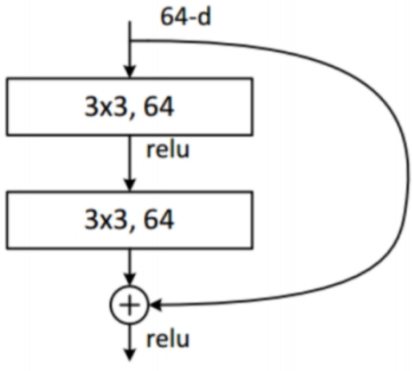


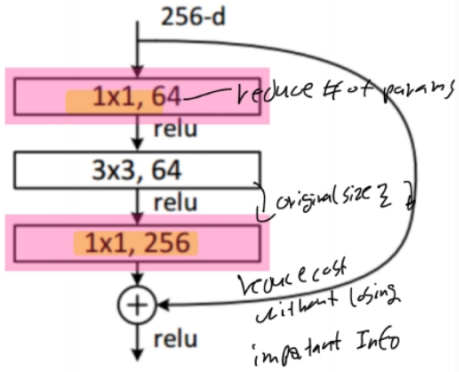
Projected Shortcut

66

Bottleneck Blocks

- Similar to GoogLeNet's Inception architecture.
- 1x1 convolution can be seen as channel-wise dimension reduction.





Performance increases while parameter size decreases.

Bottleneck Blocks

- Similar to GoogLeNet's Inception architecture.
- 1x1 convolution can be seen as channel-wise dimension reduction.