3. Gradient Descent and Batch Normalization

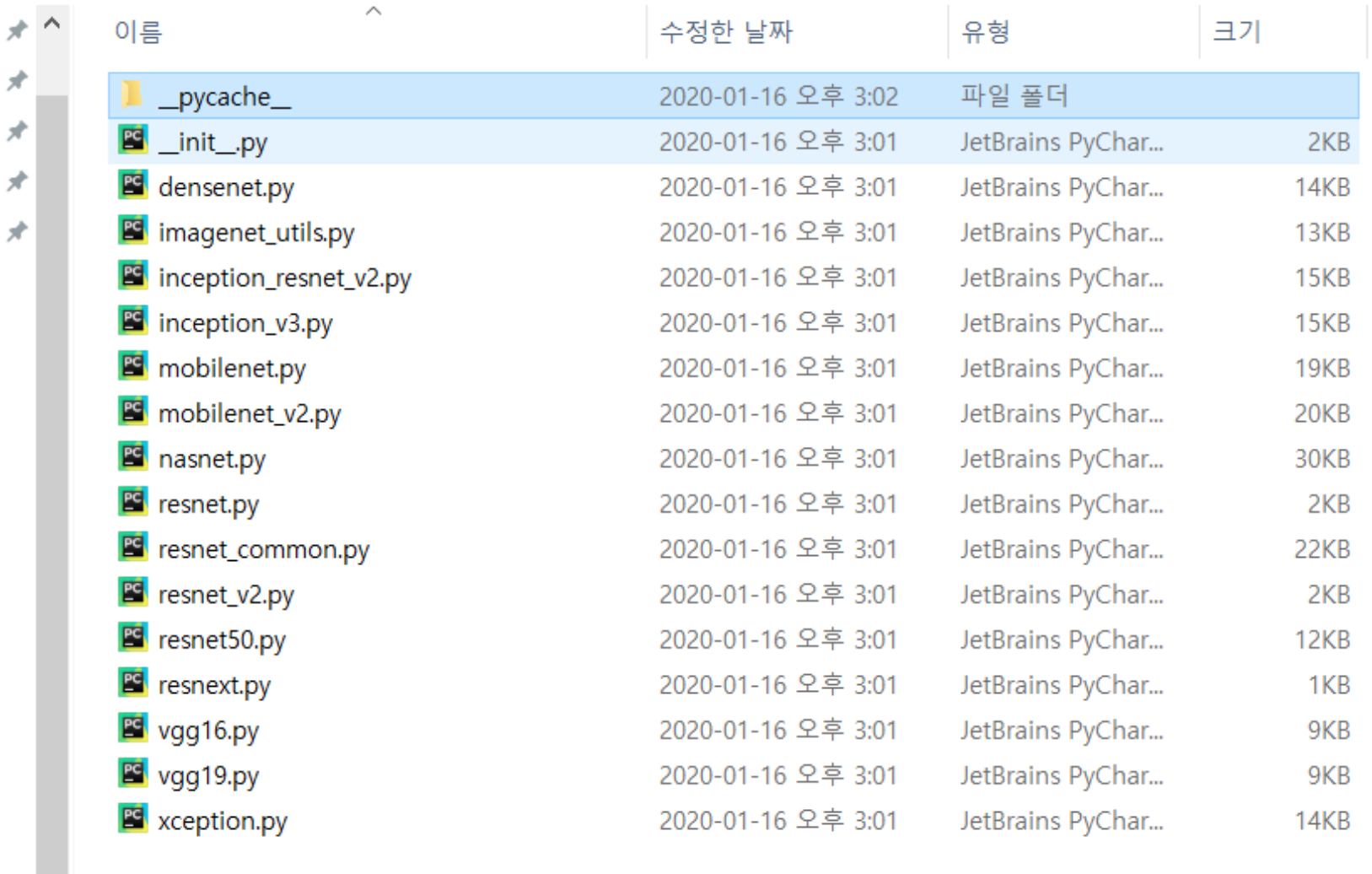
Dong-Gyu, Lee

Dept. of Statistics, KU

2020, Feb 11

Contents

Keras Models



이름	수정한 날짜	유형	크기
__pycache__	2020-01-16 오후 3:02	파일 폴더	
__init__.py	2020-01-16 오후 3:01	JetBrains PyChar...	2KB
densenet.py	2020-01-16 오후 3:01	JetBrains PyChar...	14KB
imagenet_utils.py	2020-01-16 오후 3:01	JetBrains PyChar...	13KB
inception_resnet_v2.py	2020-01-16 오후 3:01	JetBrains PyChar...	15KB
inception_v3.py	2020-01-16 오후 3:01	JetBrains PyChar...	15KB
mobilenet.py	2020-01-16 오후 3:01	JetBrains PyChar...	19KB
mobilenet_v2.py	2020-01-16 오후 3:01	JetBrains PyChar...	20KB
nasnet.py	2020-01-16 오후 3:01	JetBrains PyChar...	30KB
resnet.py	2020-01-16 오후 3:01	JetBrains PyChar...	2KB
resnet_common.py	2020-01-16 오후 3:01	JetBrains PyChar...	22KB
resnet_v2.py	2020-01-16 오후 3:01	JetBrains PyChar...	2KB
resnet50.py	2020-01-16 오후 3:01	JetBrains PyChar...	12KB
resnext.py	2020-01-16 오후 3:01	JetBrains PyChar...	1KB
vgg16.py	2020-01-16 오후 3:01	JetBrains PyChar...	9KB
vgg19.py	2020-01-16 오후 3:01	JetBrains PyChar...	9KB
xception.py	2020-01-16 오후 3:01	JetBrains PyChar...	14KB

Today's Goal

- In this session, we will learn what the Gradient Descent(GD) and Batch Normalization(BN) are and how these work.
- Before we start studying BN, we will look at a simple gradient descent algorithm first.
- Also, we will learn what batch is.

What is the Gradient?

- As you know, gradient means slope and is usually defined as a multidimensional vector.

Figure 1: Tilt in the Mountains

GD

- Gradient Descent(GD) is a term that means an algorithm.
- And the purpose of this algorithm is to find the variable ω that minimizes the target function(i.e. Loss Function).

$$\omega_{t+1} \leftarrow \omega_t - \eta \frac{\partial L(\omega_t)}{\partial \omega_t}$$

ω_t : *Parameters*, $L(\omega_t)$: *Loss Function*, η : *Learning Rate*

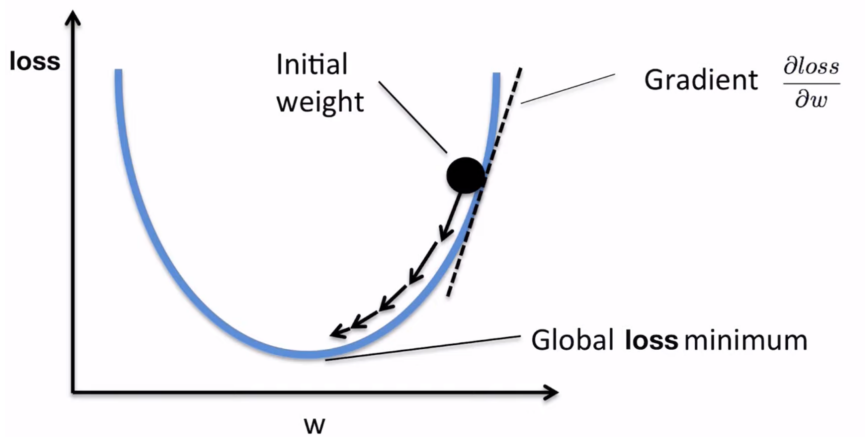


Figure 2: Gradient Descent

GD

- One thing to note is that the number of data does not affect the updating of variables(parameters).
- In statistics, the Normal equation for simple version of LSE like this: $(X^T X)\underline{\beta} = X^T \underline{y}$
- Also $\underline{\beta}$ is the value calculated by $argmin_{\underline{\beta}}(\underline{y} - X\underline{\beta})^T(\underline{y} - X\underline{\beta})$
- Moreover, $\underline{\beta}$ can be calculated(updated) regardless of the number of data.
- And the same process in statistics applies to Deep Learning if we consider loss function as $(\underline{y} - X\underline{\beta})^T(\underline{y} - X\underline{\beta})$

- Let's check the intuitive understanding through the MLP.
- Each input node has a multidimensional vector as many as the number of data.

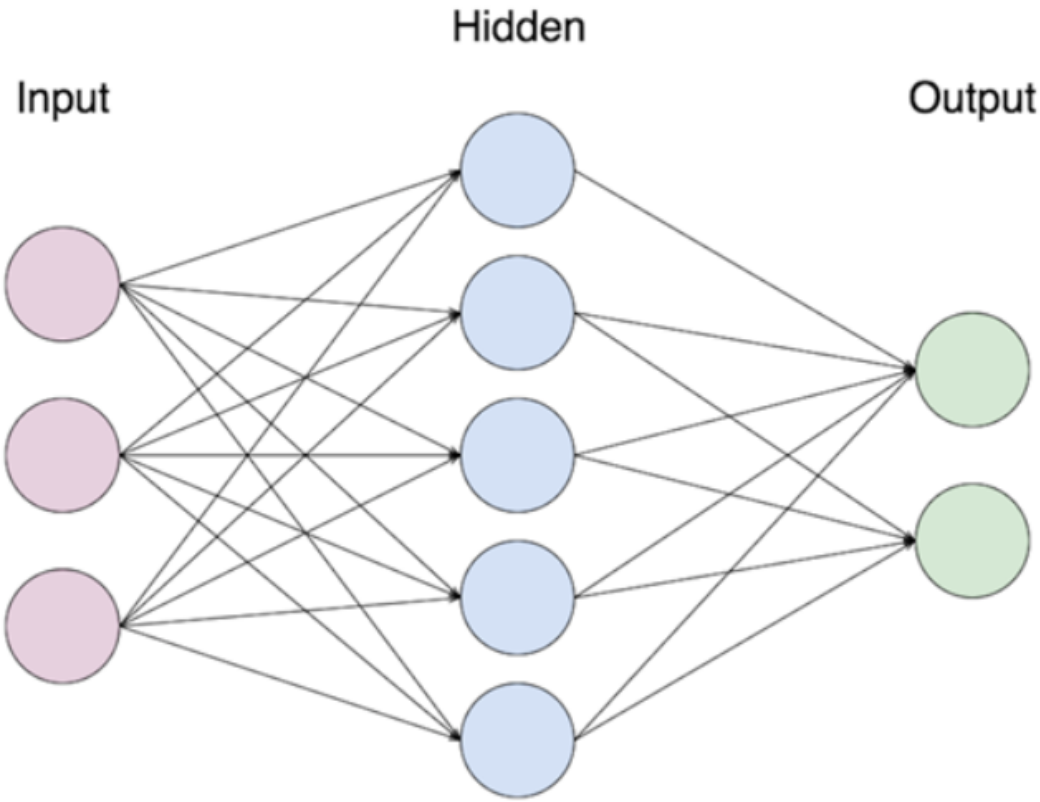


Figure 3: MLP

What is the Batch?

- Batch is a bundle of data and the Batch size is the number of data you enter into the model to update the parameters.
- Usually size of the Batch is power of 2
- The reason is that when you upload data to the GPU, space in the GPU is power of 2.
- Therefore, the calculating speed difference between Batch size 64 and 65 might be enormous, while 65 and 128 might be insignificant.
- Among developers, smaller Batch sizes(ex. 16,32) are known to have good performance.
- However, the calculation takes longer.

BGD

- The Batch Gradient Descent(BGD) stands for an GD algorithm where Batch size is equal to the entire data size.
- It offers very stable learning process.
- Furthermore, it has a small number of calculations in terms of time complexity due to the low number of total iterations.
- If the number of train data is 10,000, gradient update is performed by averaging the 10,000 gradients which is respectively calculated.

Figure 4: Batch Gradient Descent

SGD

- The Stochastic Gradient Descent(SGD)[4] is a GD algorithm with only 1 Batch size.
- For this reason, the learning is performed in a state of severe shaking such as Random Walk.
- Compared to GD, the time complexity is larger, but the convergence speed is faster.
- In addition, shooting takes place, so the risk of falling into the local minima is relatively less than the GD.

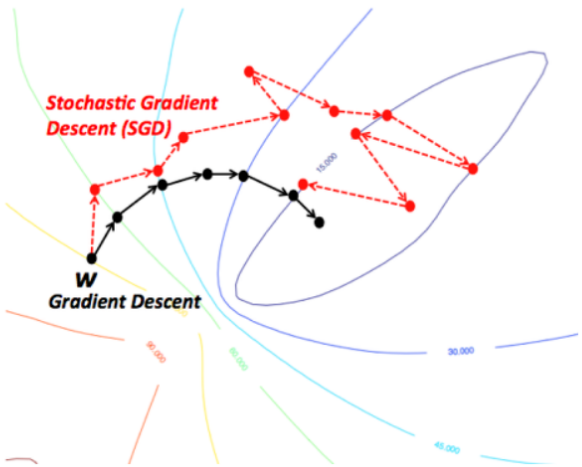


Figure 5: BGD vs SGD

MSGD(MBSGD)

- Mini-Batch Stochastic Gradient Descent(MSGD or MBSGD) is an algorithm that combines the benefits of BGD and SGD.
- The Batch size is determined by the Rule of Thumb.

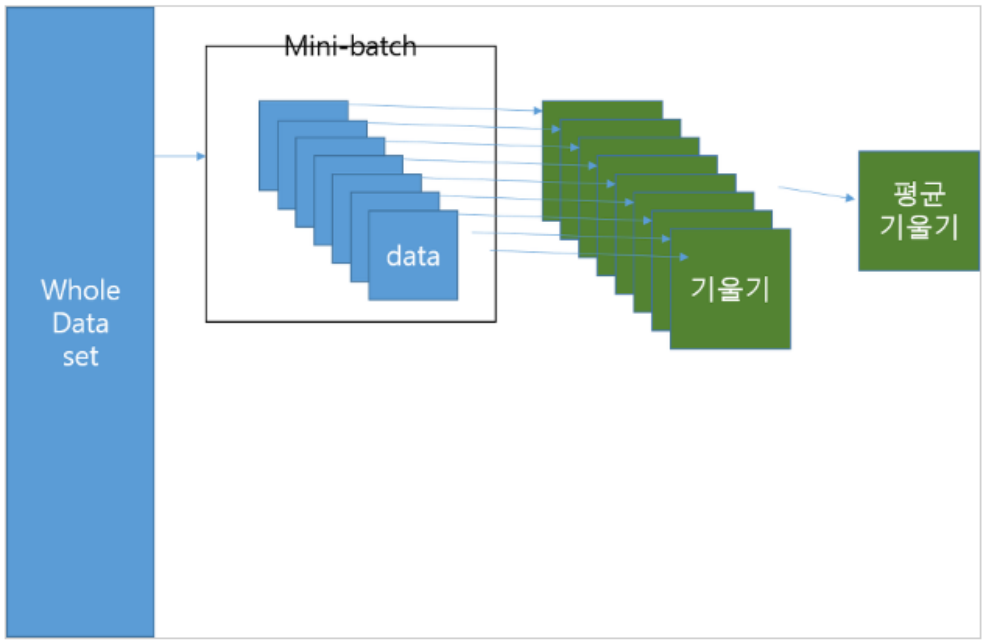
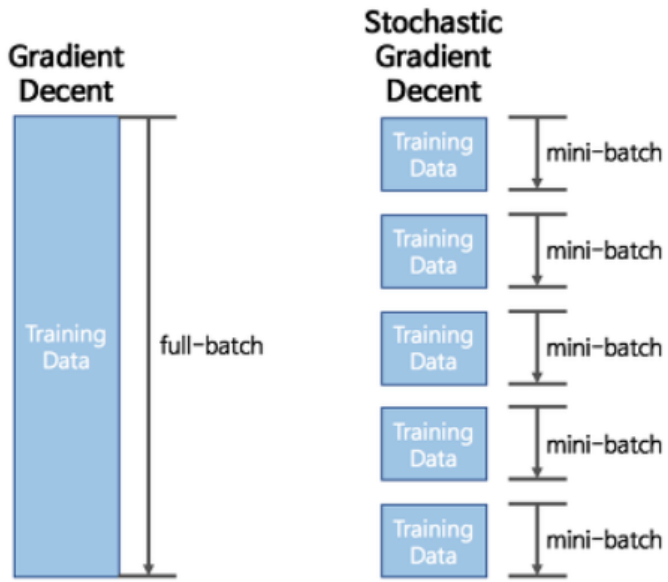


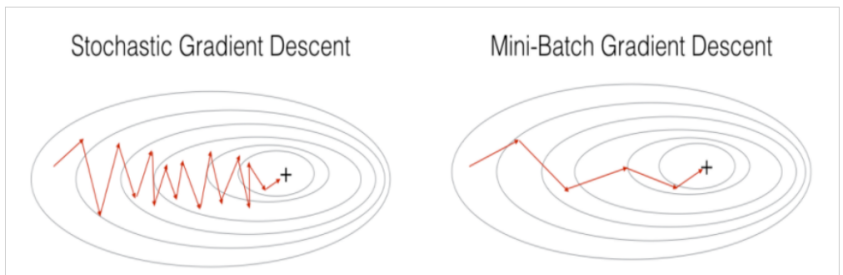
Figure 6: Calculation Using the MSGD Method

MSGD(MBSGD)

- The pictures below can help you understand MBSGD effectively.



(a) BGD vs MSGD



(b) SGD vs MSGD

Figure 7: MSGD

BN

- Batch Normalization(BN, also known as Whitening)[3] ,created for efficiency in learning, is a regularization technique.
- In addition to LSTM, the BN is a widely used technique currently.
- The term normalization in BN is equivalent to standarization in statistics.
- The expected effects of BN are:
  1. By setting a high learning rate, learning of model can be improved.
  2. Through normalizing per hidden layer, the dependency on the initial value of the parameter ω is reduced.
  3. Also, BN can reduce the risk of overfitting. This can be used to replace Dropout.
  4. The problem of vanishing gradient and exploding gradient can also be avoided.

BN

- Where to put the BN layer is currently under study[2].
- In general, the BN layer is situated before the activation function.

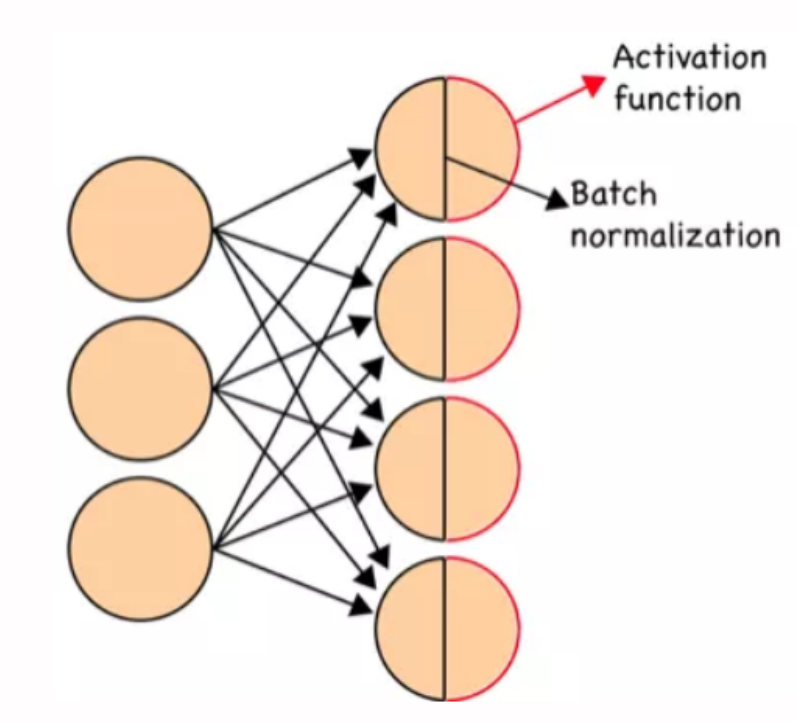


Figure 8: Batch Normalization

BN

- The BN paper says that tensors(or data) go through an "Internal Covariate Shift" every time when they pass through a layer.
- Internal covariate shift is a phenomenon in which the distribution of outputs differs from the distribution of inputs as one step of the Network goes on.
- Thus, through normalization(standardization) in the Batch, the BN corrects this shift.
- Also add appropriate parameters γ,β to compensate for the loss of parameters(intercept term) that occur during normalization.
- Of course, we assume that the data in the batch are independent of each other.

BN: NN case

- The BN method differs slightly between NN and CNN.
- Let's start with the NN case.
- When we say $B = \{x_{1...m}\}$ is input and y_i is output(i.e. $BN_{\gamma,\beta} : x_{1...m} \rightarrow y_{1...m}$), the algorithm for calculating BN is as follows:

  1. $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$: mini-batch mean
  2. $\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$: mini-batch variance
  3. $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$: normalize
  4. $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \boldsymbol{BN}_{\gamma,\beta}(x_i)$: scale and shift

- Note that the pair γ, β has the number of Batches(not Batch size) multiplied by the Dimension of x(equal to the number of Node).
- And, the overall algorithm of BN is shown in the following slide.

BN: NN case

Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network

2: **for** $k = 1 \ldots K$ **do**

3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)

4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead

5: **end for**

6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters

8: **for** $k = 1 \ldots K$ **do**

9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$

12: **end for**

Figure 9: Batch Normalization Algorithm

BN: NN case

- We are currently considering 1 epoch.
- In Figure 9 algorithm, notation "K" means the number of Batch.
- In Figure 9 algorithm, step1-6 and step7-12 are each a bundle.
- If you think that μ_B is the sample mean and σ_B^2 is the sample variance, it is easier to understand.
- In Figure 9 algorithm, The purpose of the first "for" statement is to learn the parameters in step6, and the purpose of the second "for" statement is to get the using(new) result(y) from the learned parameters.
- In step10 we eventually use $E[X]$ and $V[X]$ which are difficult to calculate directly due to memory problems.
- So, $E[X]$ and $V[X]$ are usually obtained from Mmoving Average(MA) or Exponential Moving Average(EMA). Usually the latter is used.
- In step11, the expression for $y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \cdot x + (\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}})$ is the same as $y = \gamma \frac{(x-E[x])}{\sqrt{Var[x]+\epsilon}} + \beta$.

BN: NN case

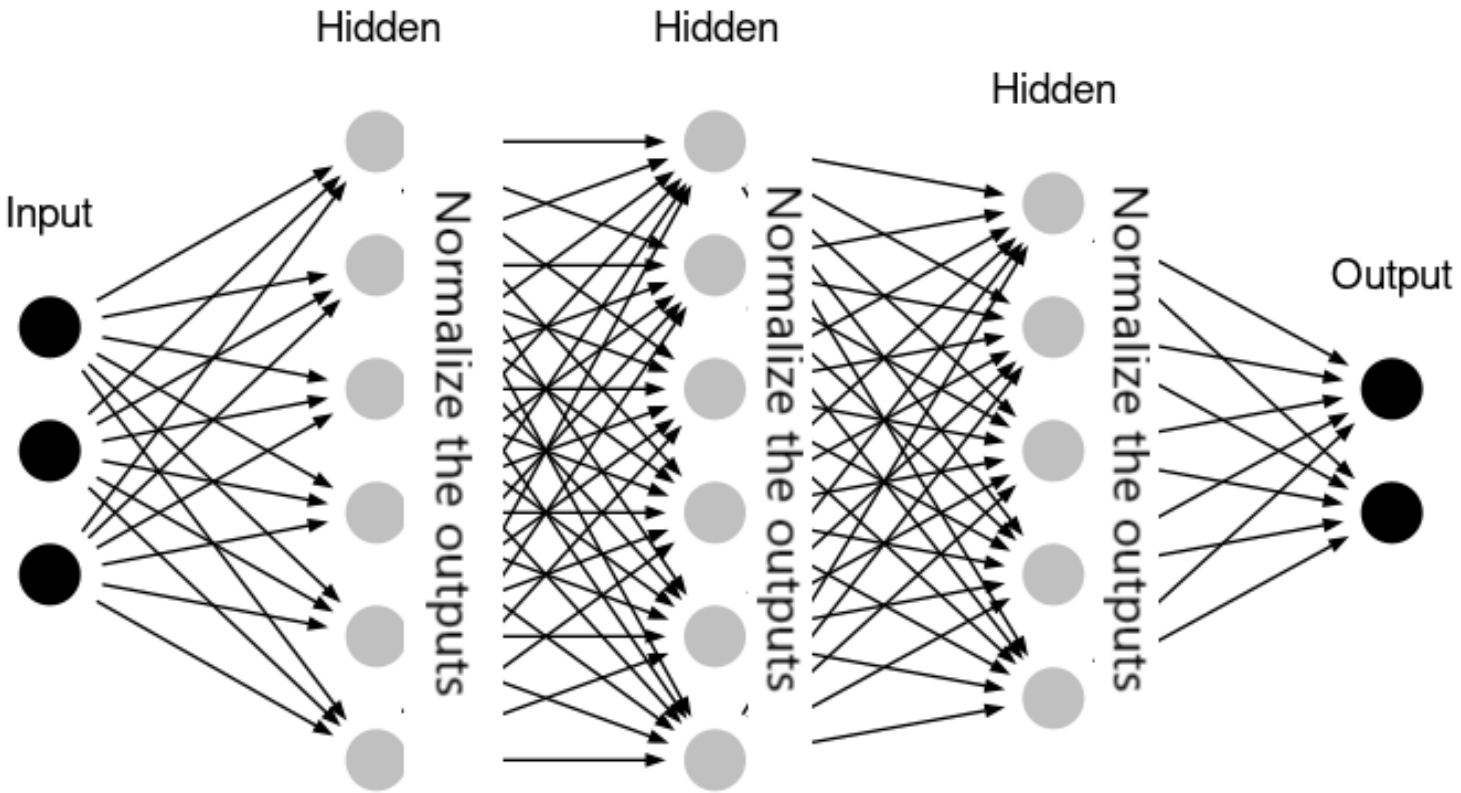
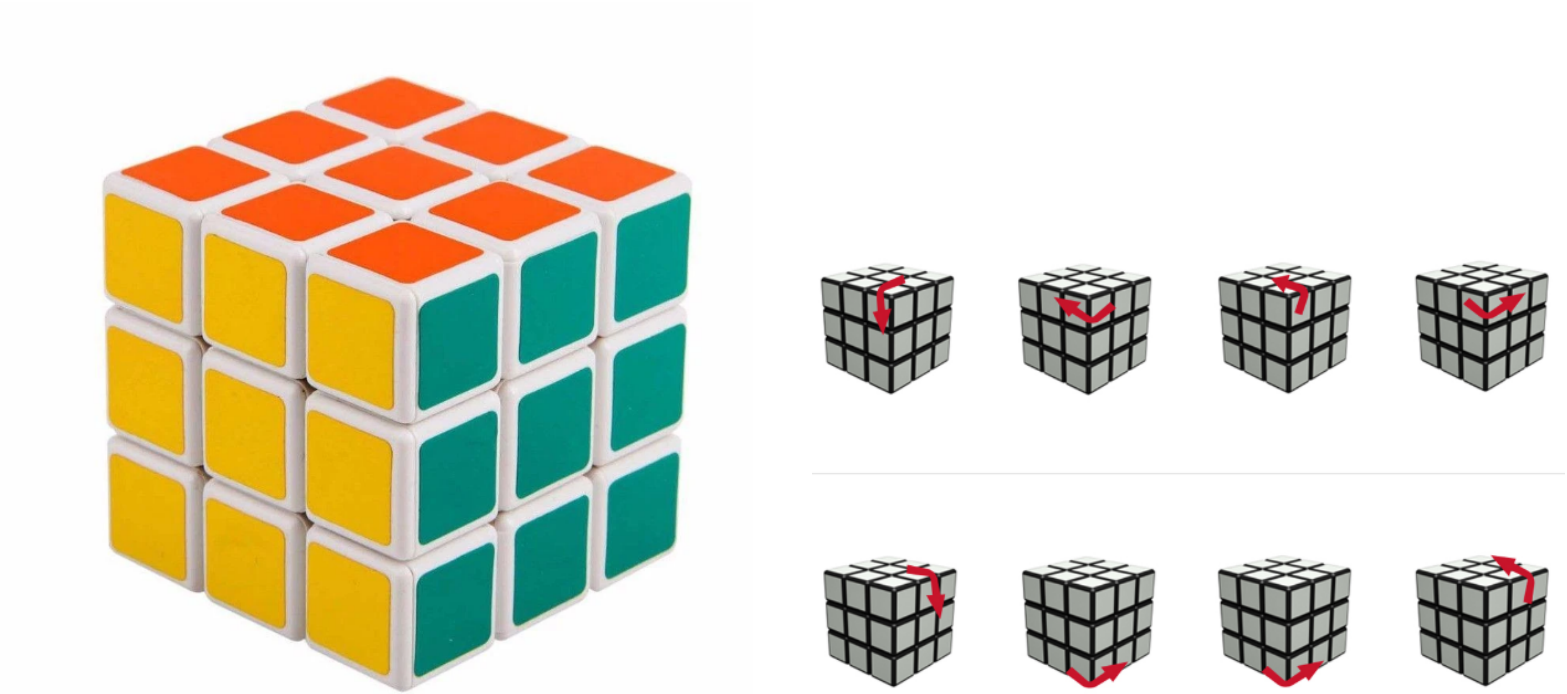


Figure 10: Batch Normalization in MLP

BN: CNN case

- In CNN, this is different a little from the usual NN described earlier, because we want to receive the input as high dimensional tensor itself(or to maintain the properties of convolution).
- In convolution, consider $y = XW$ instead of $y = XW + b$. This is because b is deleted when we use the batch normalization.
- Let's say p(width)$\times q$(height) $\times n$(filters) is the dimension of the feature map we got after the convolution operation.
- If m is the Batch size, then the tensor is $m \times p \times q \times n$ dimensions.(that is, there are m three-dimensional wooden blocks)
- And we use the same γ and β for $p \times q \times 1$ to maintain the convolution properties.
- That is, the required number of γ , β parameters is $2n$(calculated based on a filter) in one batch.
- The same is true for $E[X]$, $V[X]$. Hence, total parameters are $4n$
- Then follow the algorithm procedure in slide 17 and figure 9.

BN: CNN case



(a) $3 \times 3 \times 3$ tensor

(b) $8 \times 3 \times 3 \times 3$ tensor

Figure 11: Batch size$= 8$

Another Normalization

- In summary, the introduction of the normalization layer can increase the learning efficiency by making the input distribution equal.
- There are other normalization methods besides BN: Weight Normalization[5], Layer Normalization[1], Group Normalization[6], etc.
- Weight Normalization : It is designed to normalize parameters.
- Layer Normalization : It uses the number of nodes instead of batch size.
- Group Normalization : For small Batch sizes, it is designed for better performance.

Next

- Next time, we'll deal with the followings:
  1. Various optimization methods.
  2. Initial value problems in parameters.
  3. Type of loss function.
  4. More complex CNN models.
  5. About the RNN Model.
- Of course, Python code learning proceeds at the same time.

Reference I

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[4] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[5] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909, 2016.

Reference II

[6] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.