Biomedical Text Mining using Neural Networks

[Appendix] Techniques for Deep Learning – Part 2

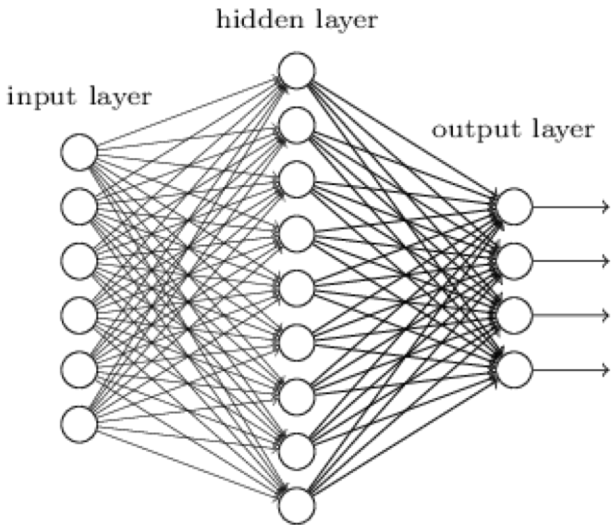
서울대학교병원 정보화실

고태훈 (taehoonko@snuh.org)

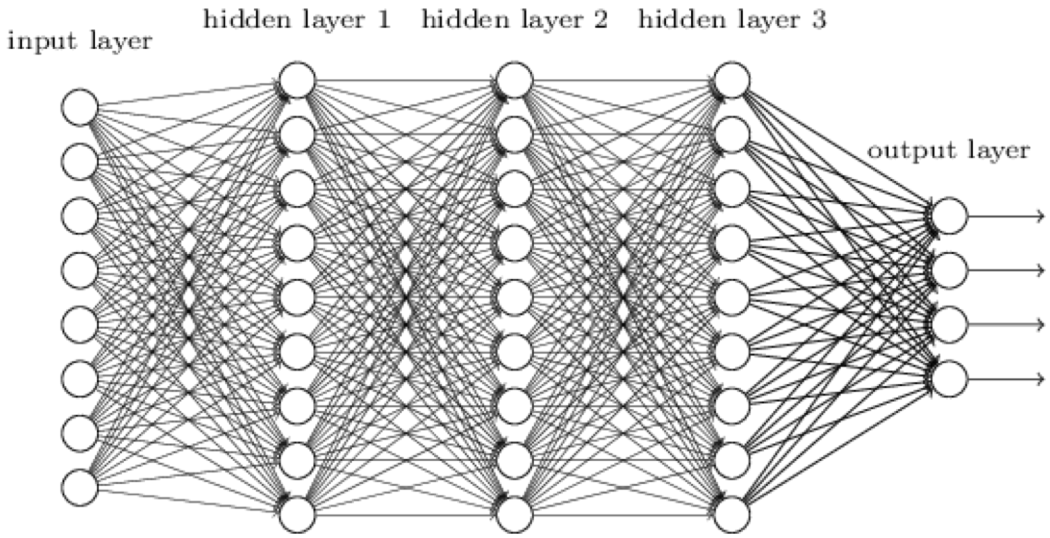
최세원 (swc@snuh.org)

Single (hidden) layer vs. Multiple layers

Number of input nodes: n
Number of hidden nodes: p *(per each hidden layer)*
Number of output nodes: l



number of weights: $np + pl$
number of biases: $p + l$



number of weights: $np + 2p^2 + pl$
number of biases: $3p + l$

The deeper the model, the greater the number of parameters in the model.

Single (hidden) layer vs. Multiple layers

- **It is more difficult to train a network with multiple hidden layers than with a hidden layer.**
- **Why?**
  - **First hypothesis: underfitting**
    - 아무리 학습을 해도 손실함수(loss function) 또는 비용함수(cost function)이 일정 이상 줄어들지 않는다 (분명 더 학습이 이루어질 수 있음에도 불구하고…).
  - **Second hypothesis: overfitting**
    - 학습데이터에서의 성능에 비해 테스트데이터에 대한 성능이 떨어진다.
    - 머신러닝 모델이 데이터에 비해 지나치게 복잡할수록 overfitting할 가능성이 높다. 신경망 모델도 예외가 아니다.

How to solve underfitting problems

- **Why? ➔ Vanishing gradient problem**
- **Weight initialization & input scaling**

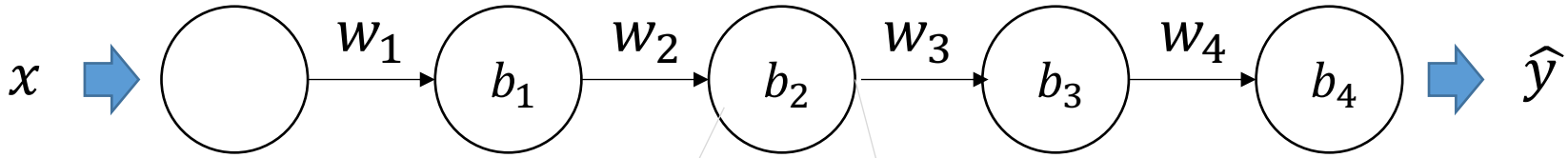
Why my network is underfitted?

- **Vanishing gradient problem (경사 소멸 문제)**
  - Backpropagation 으로 가중치를 학습할 때, 입력층에 가까울수록 각 가중치에 대한 gradient값이 매우 작아져서 학습이 잘 이루어지지 않는 문제
- **Vanishing gradient problem in deep NN**
  - It is a difficulty found in training artificial neural networks with gradient-based learning methods and backpropagation.
  - Backpropagation computes gradients by the "chain rule".
    - The gradients decreases exponentially with the number of layers.
    - The front layers, which are close to the input layer, train very slowly.

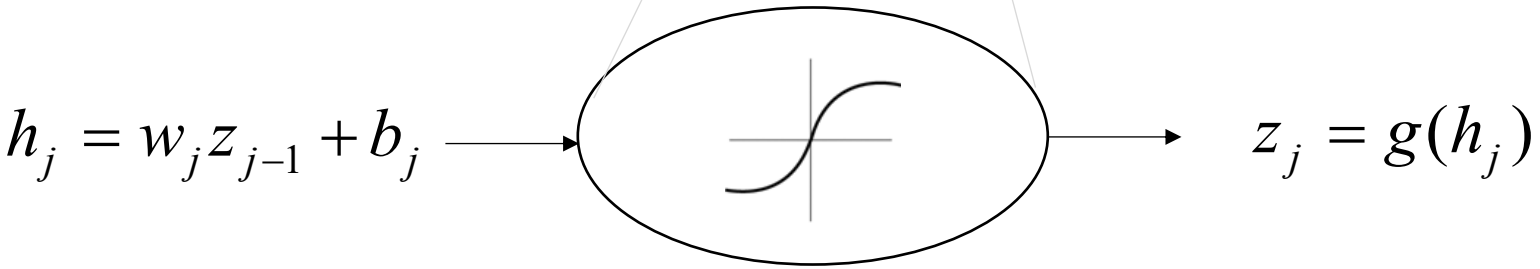
https://en.wikipedia.org/wiki/Vanishing_gradient_problem

Vanishing gradient problem in deep NN

- **Example: A network with three hidden layers**
  - Just a single node in each layer
  - Weights: w_j / Biases: b_j / input data: (x,y) / Predicted output: $\hat{y}$

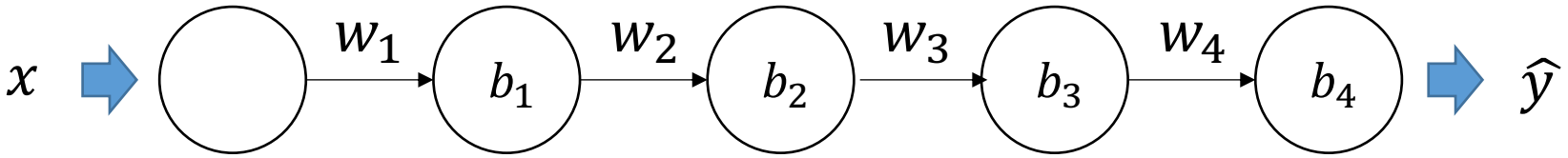


  - Input value: h_j / Output value: z_j / Activation funcion: g



Vanishing gradient problem in deep NN

- **Example: A network with three hidden layers**



$$h_1 = w_1 x + b_1 \qquad z_1 = g(h_1)$$

$$h_2 = w_2 z_1 + b_2 \qquad z_2 = g(h_2)$$

$$h_3 = w_3 z_2 + b_3 \qquad z_3 = g(h_3)$$

$$h_4 = w_4 z_3 + b_4 \qquad z_4 = g(h_4)$$

$$\hat{y} = z_4$$

Loss function: L

Gradient for parameter θ: $\dfrac{\partial L}{\partial \theta}$

Vanishing gradient problem in deep NN

$$\frac{\partial L}{\partial w_4} = \frac{\partial h_4}{\partial w_4} \cdot \frac{\partial z_4}{\partial h_4} \cdot \frac{\partial L}{\partial z_4} = z_3 \cdot g'(h_4) \cdot \frac{\partial L}{\partial z_4}$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial h_3}{\partial w_3} \cdot \frac{\partial z_3}{\partial h_3} \cdot \frac{\partial h_4}{\partial z_3} \cdot \frac{\partial z_4}{\partial h_4} \cdot \frac{\partial L}{\partial z_4}$$

$$= z_2 \cdot g'(h_3) \cdot w_4 \cdot g'(h_4) \cdot \frac{\partial L}{\partial z_4} = (z_2 w_4) \cdot g'(h_3) \cdot g'(h_4) \cdot \frac{\partial L}{\partial z_4}$$

$$\vdots$$

$$\frac{\partial L}{\partial w_1} = k \cdot g'(h_1) \cdot g'(h_2) \cdot g'(h_3) \cdot g'(h_4) \cdot \frac{\partial L}{\partial z_4} \quad (k : \text{constant})$$

"chain rule"

Vanishing gradient problem in deep NN

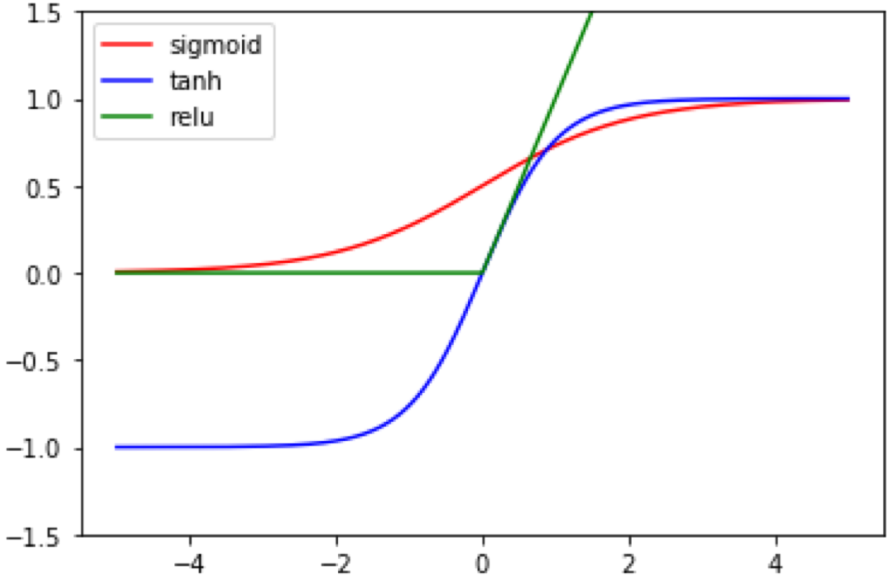
$$\frac{\partial L}{\partial w_1} = k \cdot \underbrace{g'(h_1) \cdot g'(h_2) \cdot g'(h_3) \cdot g'(h_4)}_{\text{“chain rule”}} \cdot \frac{\partial L}{\partial z_4} \quad (k : \text{constant})$$

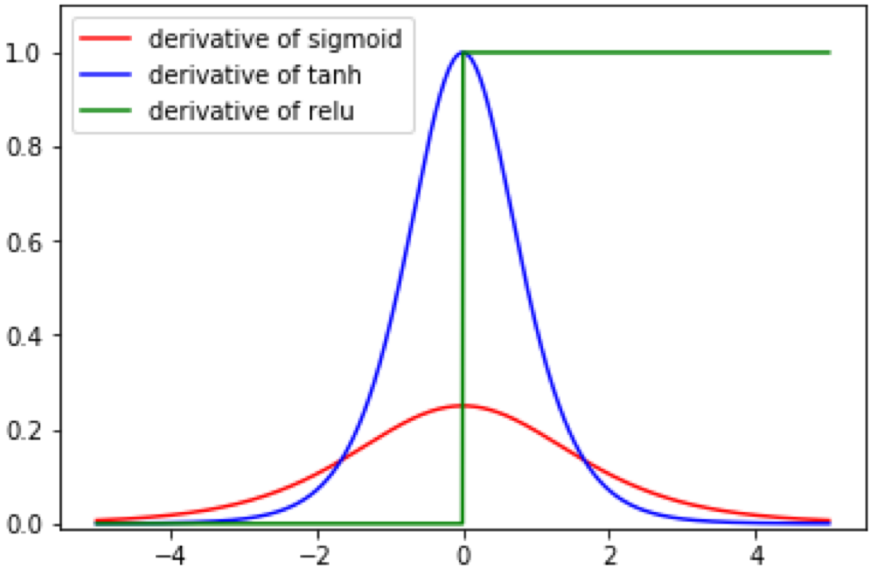
If $g(x)$ is a softmax function, $g'(x) = g(x)(1 - g(x))$ ➔ $0 < g'(x) \leq 0.25$

If $g(x)$ is a hyperbolic tangent function, $g'(x) = 1 - \big(g(x)\big)^2$ ➔ $0 < g'(x) \leq 1$

If $g(x)$ is a ReLU, $g'(x) = \begin{cases} 1 & \text{when } x \geq 0 \\ 0 & \text{when } x < 0 \end{cases}$

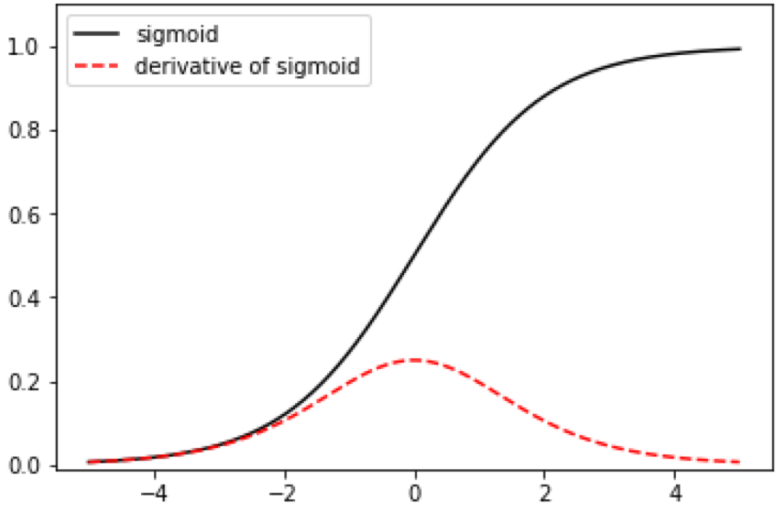
Derivatives of activation functions

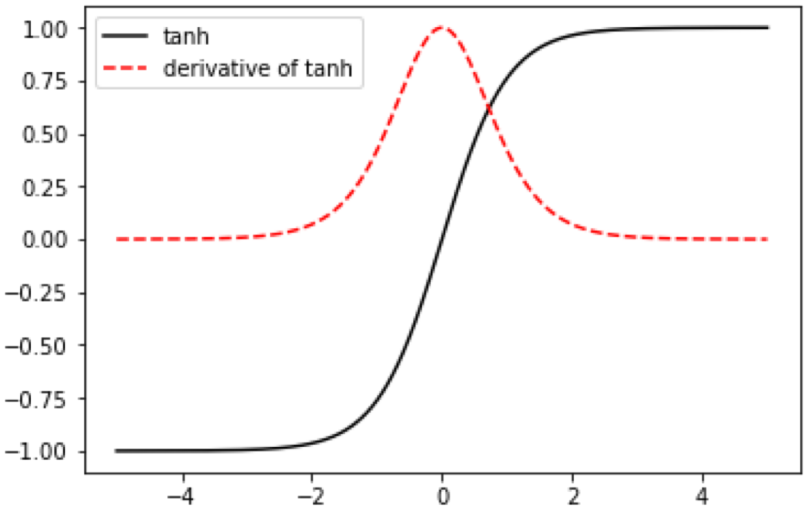


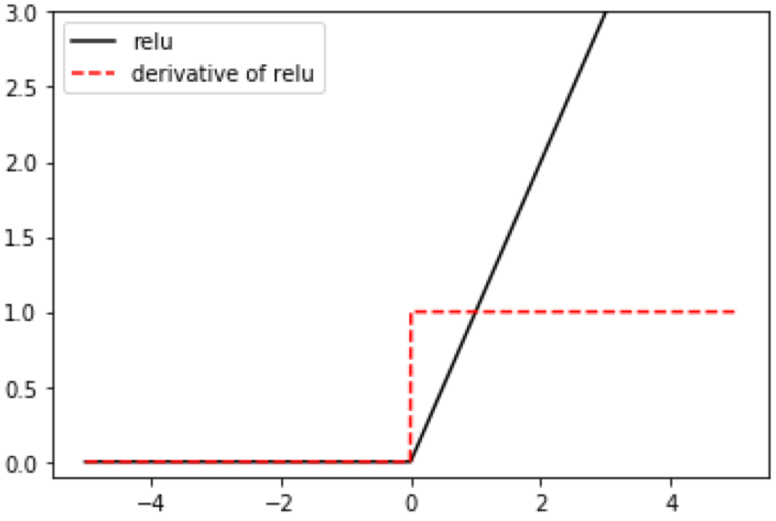


Appendix_activation_fuction_and_their_derivatives.ipynb 참조

Derivatives of activation functions







Appendix_activation_fuction_and_their_derivatives.ipynb 참조

Vanishing gradient problem in deep NN

- **Backpropagation computes gradients by the "chain rule".**
  - The gradients decreases exponentially with the number of layers.
  - The front layers, which are close to the input layer, are trained very slowly.
    - The gradient for the activation function is 1 or less.
      - In case of the softmax function, it is less than 0.25.
    - Multiple derivatives are multiplied. ➔ Gradient of parameter is decreased.
    - If a derivative almost zero is multiplied, what happen?

- **For this reason, the ReLU is often used as an active function of the hidden node.**
  - When the hidden node is activated, the gradient is **1**.

How to solve underfitting problems

- **Why? ➔ Vanishing gradient problem**
- **Weight initialization & input scaling**

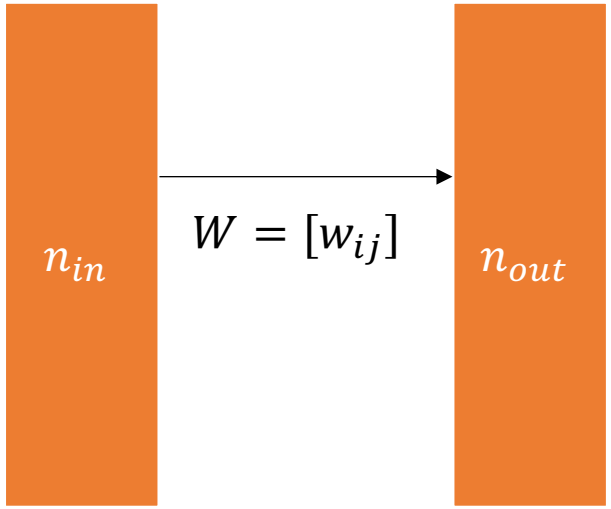
Why my network is underfitted?

- **Weight initialization problem (가중치 초기화 문제)**
  - 네트워크의 구조가 깊고 복잡할수록 가중치의 초기화가 학습 속도 및 성능에 꽤 큰 영향을 미침
  - 이는 입력 데이터의 scaling과도 크게 연관이 있음

- **기존 가중치 초기화 방법**
  - All zero or small constant? ➔ It is a wrong way…
  - Uniform distribution
  - Random normal dist. with zero mean and small standard deviation ➔ $N \sim (0, \sigma^2)$
  - Random truncated normal dist. with zero mean and small standard deviation
    - Truncated normal = bounded normal

Recommended methods for initializing weights

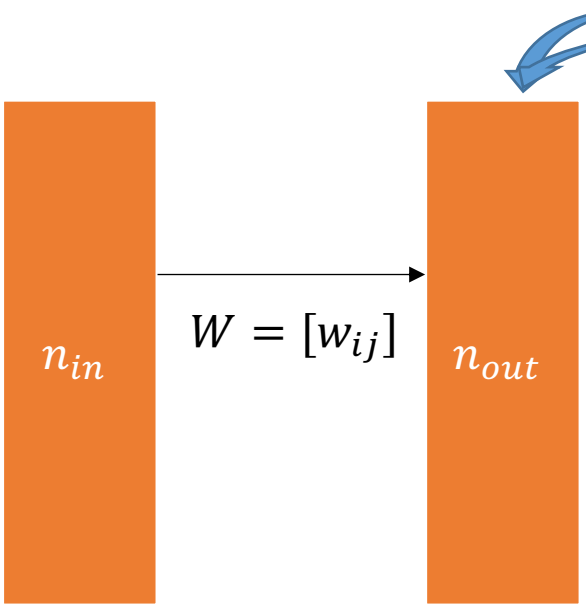
- **Xavier (or Glorot) initialization**
  - Goal: The variance of gradients should be the same in each layer.
  - If $n_{in} = n_{out}$ and $var(w_{ij}) = \frac{2}{n_{in}+n\_out}$, then we can achieve above goal.
    - Even if $n_{in} \neq n_{out}$, it works well for learning the network in practice.
  - 2 version
    - Normal dist.: $w_{ij} \sim N\left(0, \frac{2}{n_{in}+n_{out}}\right)$
    - Uniform dist.: $w_{ij} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$

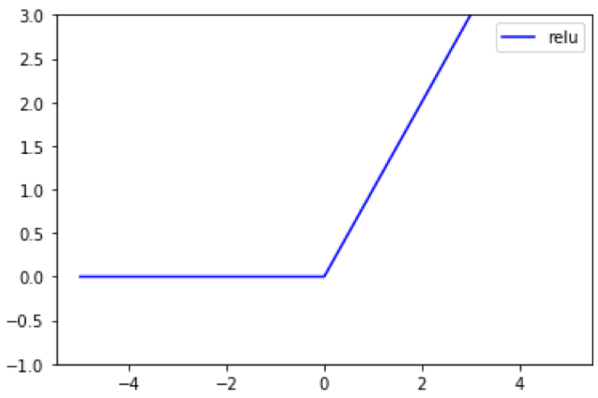


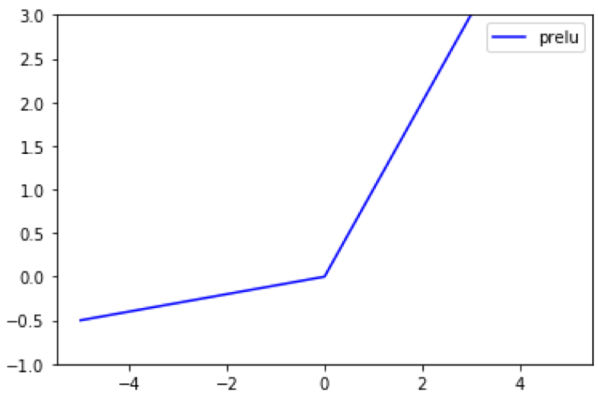
http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

Recommended methods for initializing weights

- **He initialization**
  - This method has became famous thanks to the ResNet model.
    - ResNet (to be covered) : Winner model in 2015 ImageNet Challenge
  - He initialization method is very effective when the ReLU or PReLU units are used.
  - $w_{ij} \sim N\left(0, \frac{2}{n_{in}}\right)$







https://arxiv.org/pdf/1502.01852v1.pdf

Xavier and He initialization in TensorFlow

- **Xavier initialization**
  - https://www.tensorflow.org/api_docs/python/tf/contrib/layers/xavier_initializer
  - Uniform distribution을 사용할지 Normal distribution을 사용할지 사용자 정의
- **He initialization**
  - https://www.tensorflow.org/api_docs/python/tf/contrib/layers/variance_scaling_initializer
  - 기본 파라미터가 He initialization이며 몇 가지 옵션을 더 부여할 수 있음
- **How to use? ➔ Very simple!**
  - weight = **tf.get_variable**(name='w1', shape=[784, 200], initializer=**tf.contrib.layers.xavier_initializer()**)
  - weight = **tf.get_variable**(name='w1', shape=[784, 200], initializer=**tf.contrib.layers.variance_scaling_initializer()**)

Input data preprocessing

- **1) Zero-centered data**
  - Subtract the mean in each dimension.
  - Why? ➔ Characteristics of activation function
    - Sigmoid and tanh are symmetric with respect to the input value 0.
    - ReLU activates the signal when the input value is greater than 0.
- **2) Scaling the data**
  - Method 1: Divide the standard deviation in each dimension. (standardization)
  - Method 2: Normalize each dimension so that the min and max along the dimension is -1 and 1 respectively. (min-max scaling, or normalization)
    - The normalization range may vary.
      ➔ For example, VGGnet normalizes the data into range [-0.5, 0.5].
  - Why?
    - To control the variance. (related to the initialization of weights)
    - Because the effect of backpropagation is equally distributed across all dimensions

참고: Feature scaling

- **Standardization (Z-score normalization)**
  - The features will be rescaled so that they'll have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$.

$$Z = \frac{X - \mu(X)}{\sigma(X)}$$

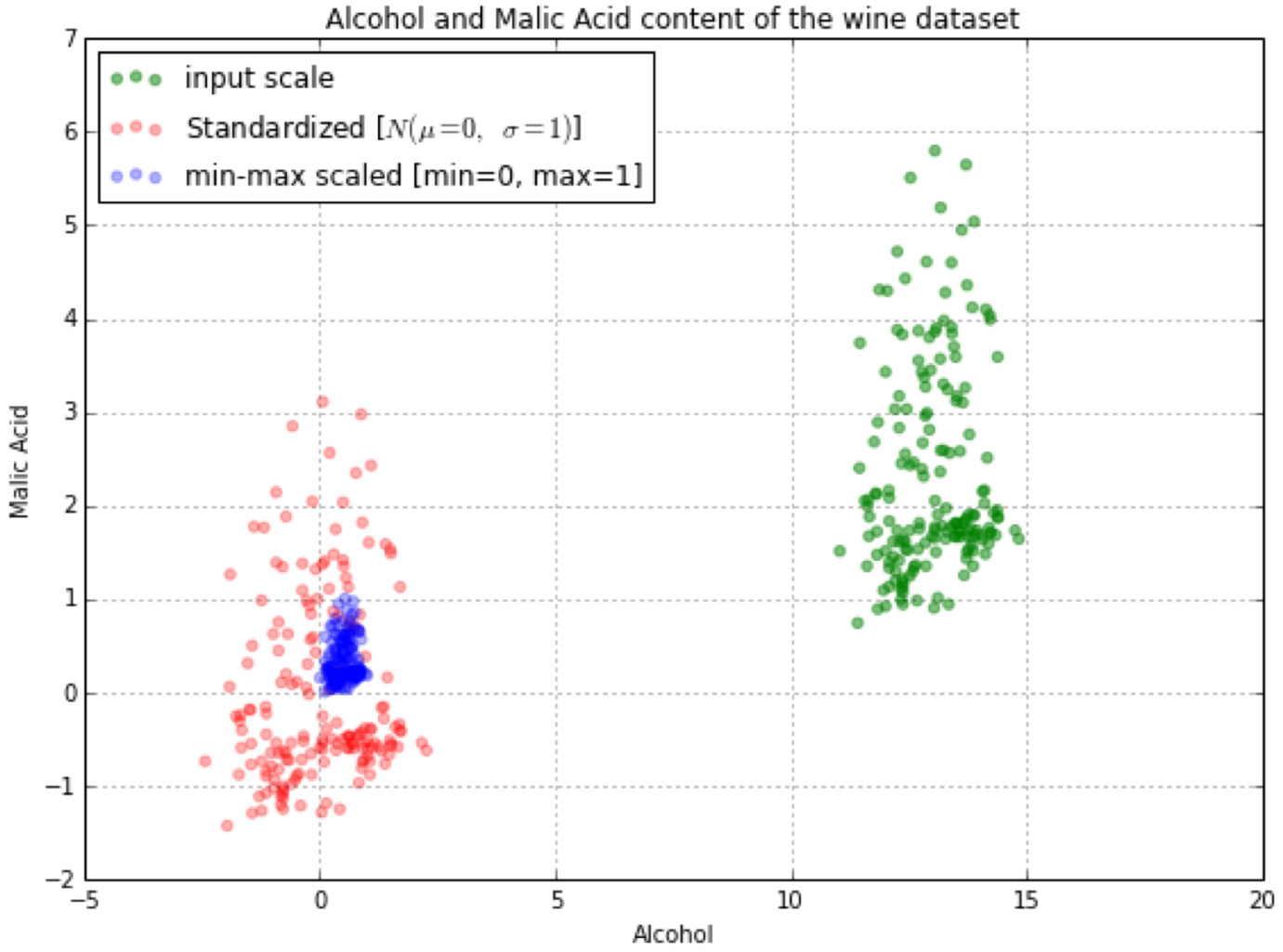
- **Min-Max scaling (Normalization)**
  - The feature is rescaled to a fixed range [0,1].

$$X_{norm} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

  - The feature is rescaled to a fixed range [-1, 1].

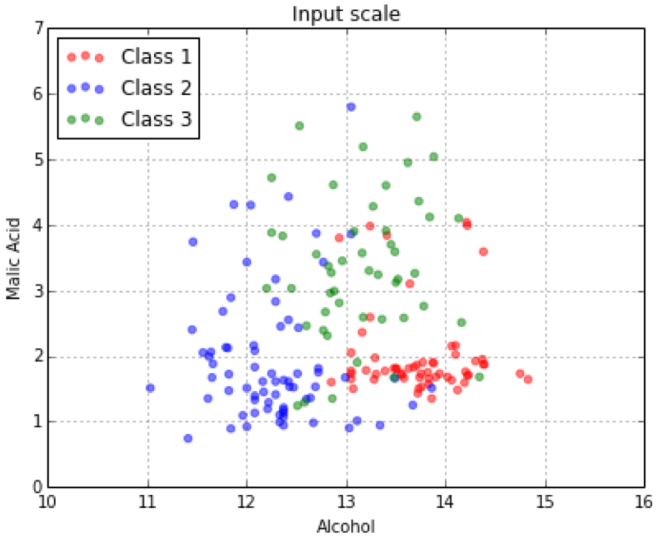
$$X_{norm} = 2 \times \frac{X - X_{\min}}{X_{\max} - X_{\min}} - 1$$

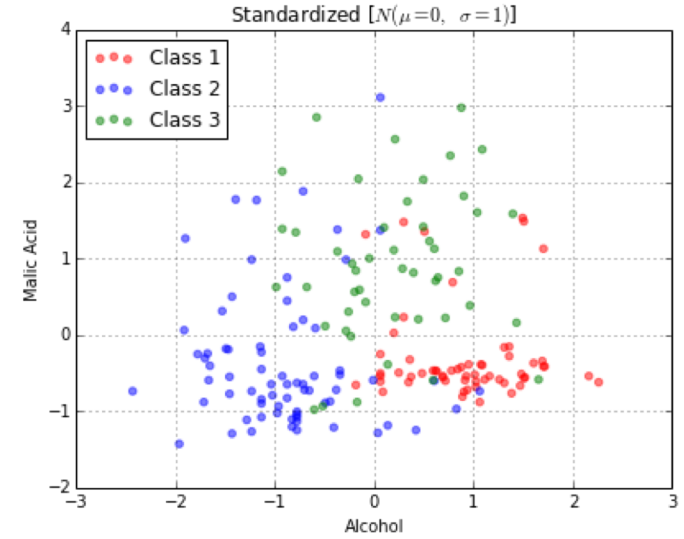
참고: Feature scaling

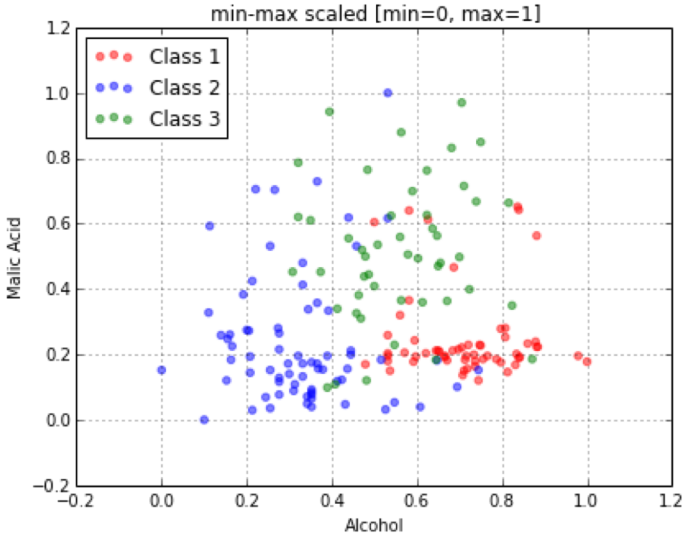


http://sebastianraschka.com/Articles/2014_about_feature_scaling.html

참고: Feature scaling







http://sebastianraschka.com/Articles/2014_about_feature_scaling.html

How to solve overfitting problems

- **Dropout**
- **L2-regularization**

Dropout

- **Large network ➔ a large number of parameters ➔ Overfitting**
- **How to avoid overfitting in neural network ➔ "Dropout"**
  - Idea: Randomly drop units (along with their connections) from the neural network during training.
  - During training, dropout samples from an exponential number of different "thinned" networks.
  - At test time, it is easy to *approximate the effect of averaging the predictions of all these thinned networks* by simply using a single unthinned network that has smaller weights.
    - **Regularization**
    - **Similar to ensemble models**

Dropout

- 딥 뉴럴넷의 오버피팅을 방지하기 위한 제약 (regularization) 방법
- 노랑 -> 방법 / 파랑 -> 학습 단계 / 초록 -> 테스트 단계

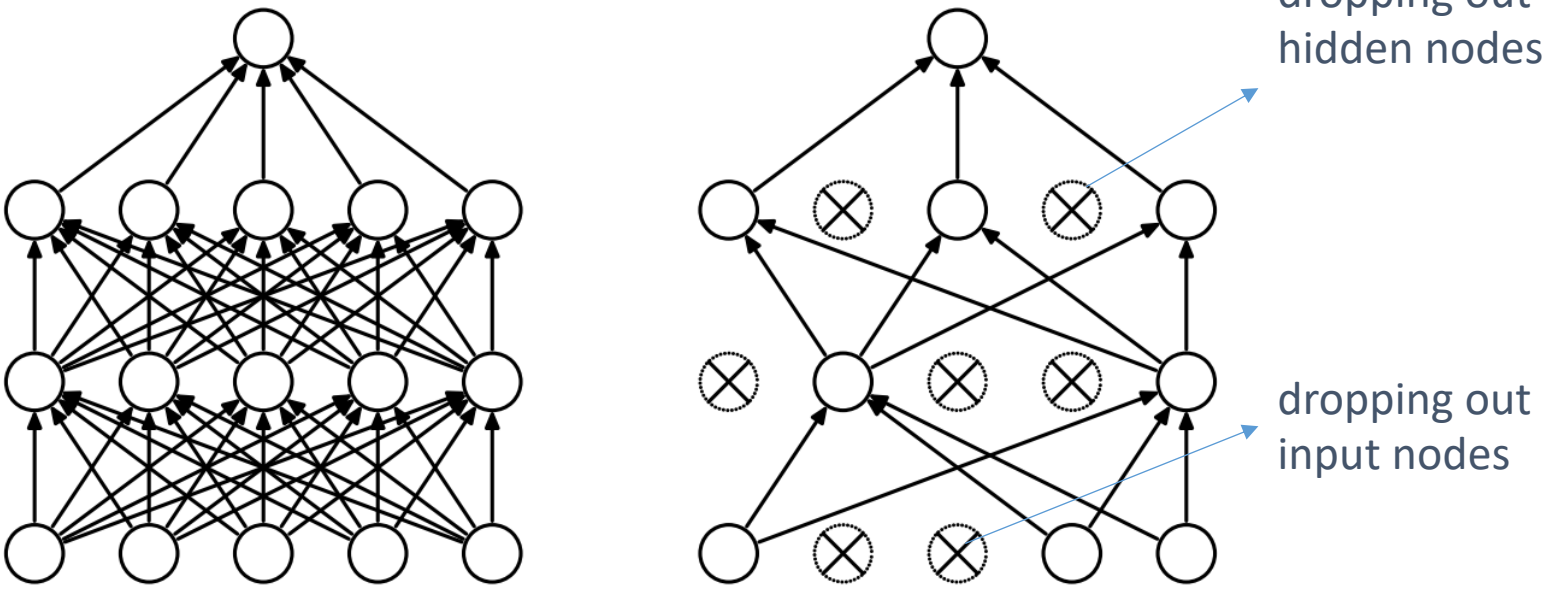
Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

Keywords: neural networks, regularization, model combination, deep learning

Dropout

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting.*Journal of Machine Learning Research*, *15*(1), 1929-1958.



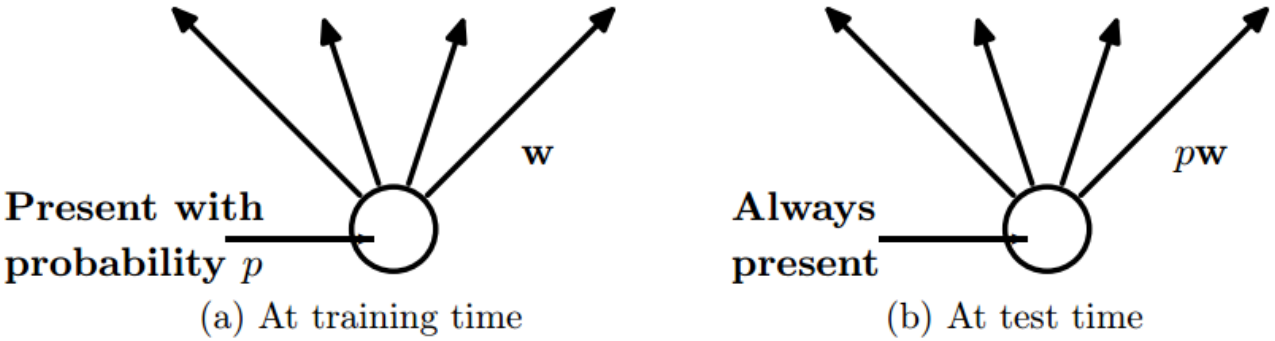
(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Dropout

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting.*Journal of Machine Learning Research*, *15*(1), 1929-1958.

- **Probability of retention *p***
  - For hidden units
    - Simply $p = 0.5$
    - You can choice p using a validation set
  - For input/visible units
    - p is usually closer to 1 than 0.5

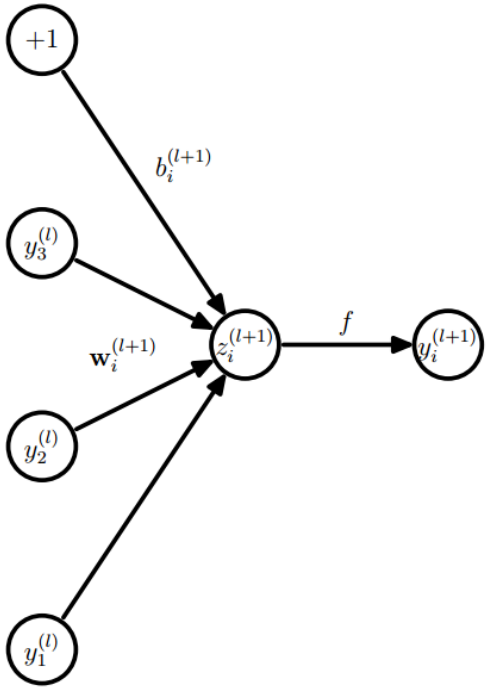


(a) At training time (b) At test time

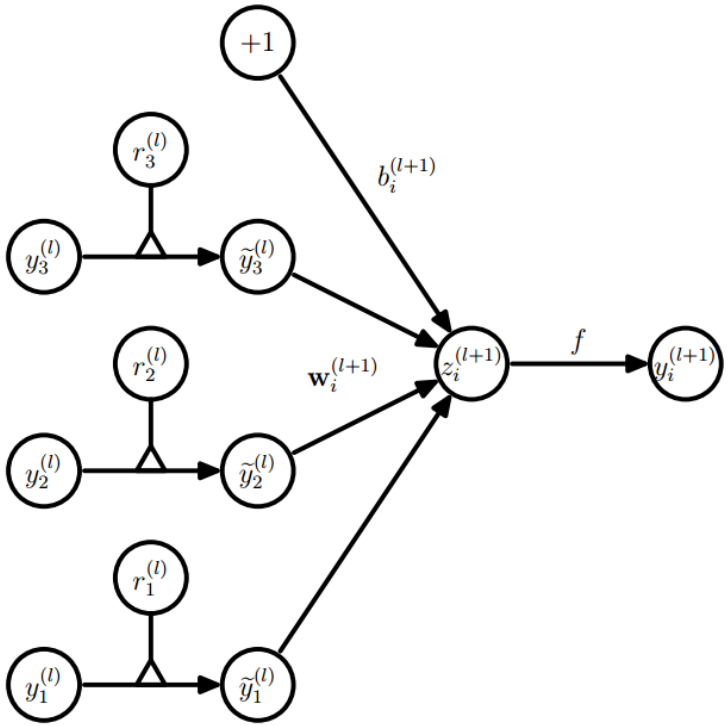
Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Dropout

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting.*Journal of Machine Learning Research*, *15*(1), 1929-1958.



(a) Standard network



(b) Dropout network

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

Dropout

- **노랑 -> Forward phase**
  - (Original paper에서는) 하나의 case (or point) 에 대해 dropout을 수행하여 thinned network를 생성하고,
- **파랑 -> Backpropagation phase**
  - weight update는 mini-batch 단위로 수행

5.1 Backpropagation

Dropout neural networks can be trained using stochastic gradient descent in a manner similar to standard neural nets. The only difference is that for each training case in a mini-batch, we sample a thinned network by dropping out units. Forward and backpropagation for that training case are done only on this thinned network. The gradients for each parameter are averaged over the training cases in each mini-batch. Any training case which does not use a parameter contributes a gradient of zero for that parameter. Many methods have been used to improve stochastic gradient descent such as momentum, annealed learning rates and L2 weight decay. Those were found to be useful for dropout neural networks as well.

Dropout

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting.*Journal of Machine Learning Research*, *15*(1), 1929-1958.

- **Dropout beats regular backpropagation on many datasets.**
  - For more details, please refer to above paper.

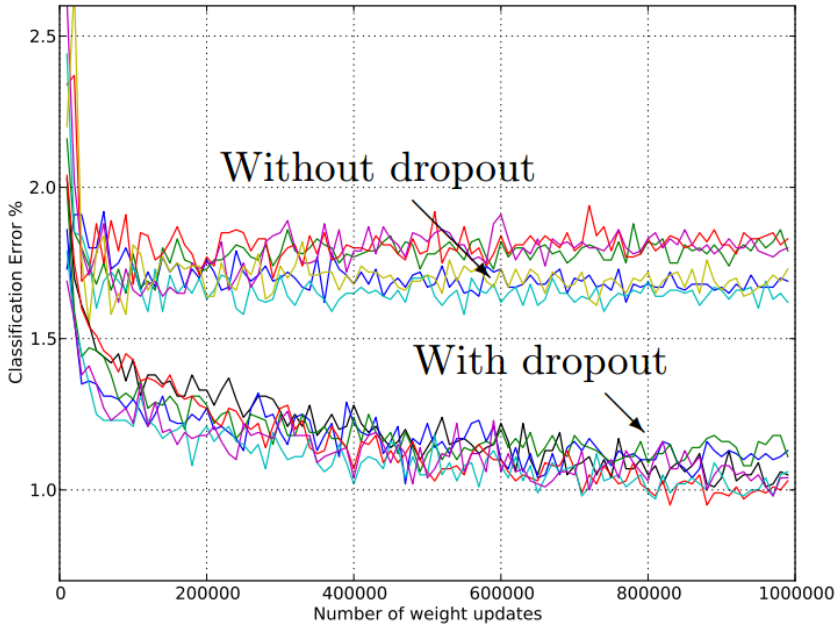


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Dropout

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting.*Journal of Machine Learning Research*, *15*(1), 1929-1958.

- **Appendix A. A Practical Guide for Training Dropout Networks**
  - Network size: 스탠다드 네트워크에서 hidden layer에 n개의 units을 사용했다면, dropout을 사용할 때에는 n / p 개의 units을 사용
  - Learning rate and momentum
    - A dropout net should typically use 10-100 times the learning rate that was optimal for a standard neural net.
    - While momentum values of 0.9 are common for standard nets, with dropout we found that values around 0.95 to 0.99 work quite a lot better.
    - 학습률과 모멘텀을 크게 가져가면 학습 속도가 빨라져 converge가 빠르게 이루어지길 기대해 볼 수 있다.

Dropout in CNN

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting.*Journal of Machine Learning Research*, *15*(1), 1929-1958.

- 모든 연구자들이 fully-connected layer에서의 dropout은 성과가 있다고 하지만, convolutional layer에서의 dropout에 대해서는 의견이 나뉨
- (Srivastava et al., 2014) ➔ Convolution layer에서의 dropout이 효과가 있다

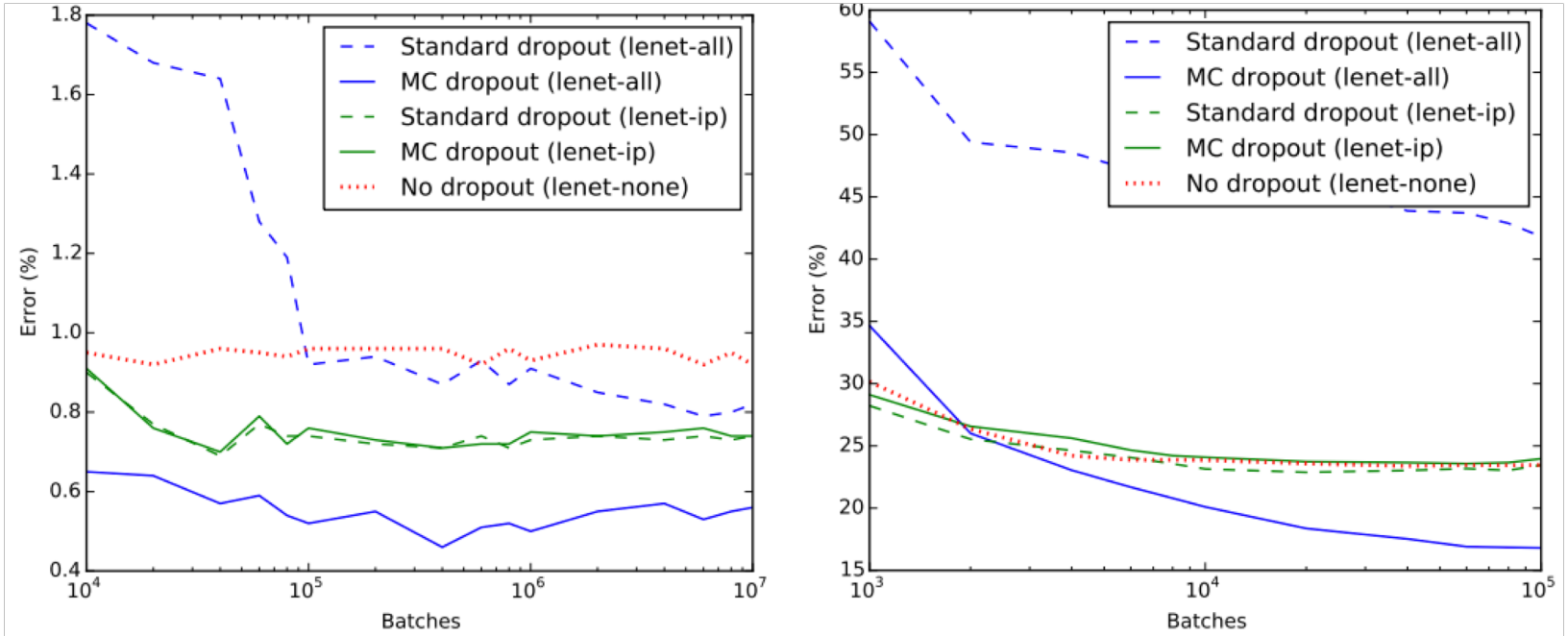
Method	Error %
Binary Features (WDCH) (Netzer et al., 2011)	36.7
HOG (Netzer et al., 2011)	15.0
Stacked Sparse Autoencoders (Netzer et al., 2011)	10.3
KMeans (Netzer et al., 2011)	9.4
Multi-stage Conv Net with average pooling (Sermanet et al., 2012)	9.06
Multi-stage Conv Net + L2 pooling (Sermanet et al., 2012)	5.36
Multi-stage Conv Net + L4 pooling + padding (Sermanet et al., 2012)	4.90
Conv Net + max-pooling	3.95
Conv Net + max pooling + dropout in fully connected layers	3.02
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	2.80
Conv Net + max pooling + dropout in all layers	2.55
Conv Net + maxout (Goodfellow et al., 2013)	**2.47**
Human Performance	2.0

Table 3: Results on the Street View House Numbers data set.

Dropout in CNN

Gal, Y., & Ghahramani, Z. (2015). Bayesian convolutional neural networks with Bernoulli approximate variational inference. arXiv preprint arXiv:1506.02158.

- **(Gal and Ghahramani, 2015)**
  - Convolution layer에 대한 (standard) dropout은 별 효과가 없다.
  - 대신, Monte Carlo dropout을 convolutions에 적용하는 것은 효과가 있다.



Standard dropout (lenet-all): 모든 레이어에 dropout 적용
Standard dropout (lenet-ip): Fully-connected 레이어에만 dropout 적용

Dropout in RNN

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. arXiv preprint arXiv:1409.2329.

- (Zaremba et al., 2014)

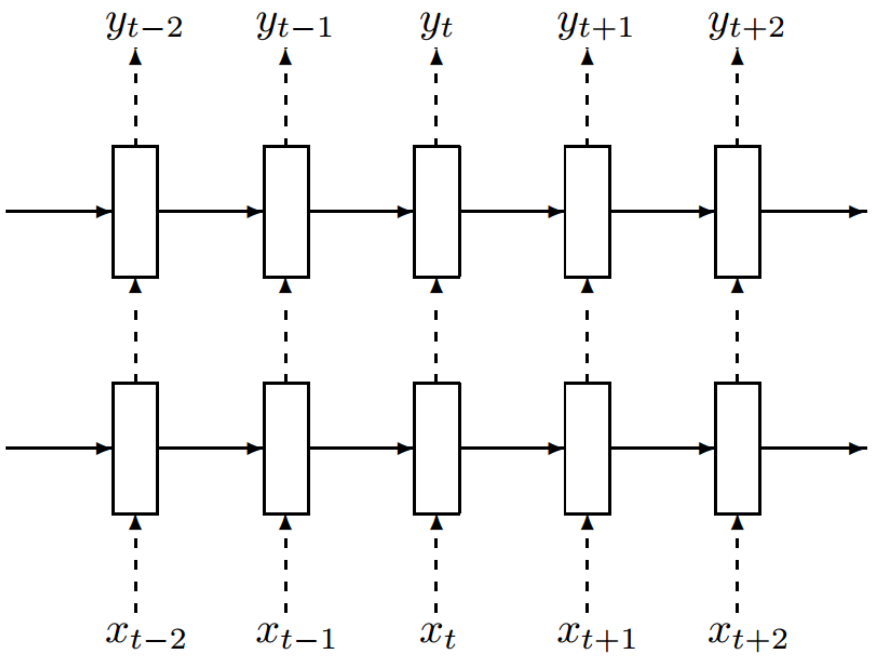
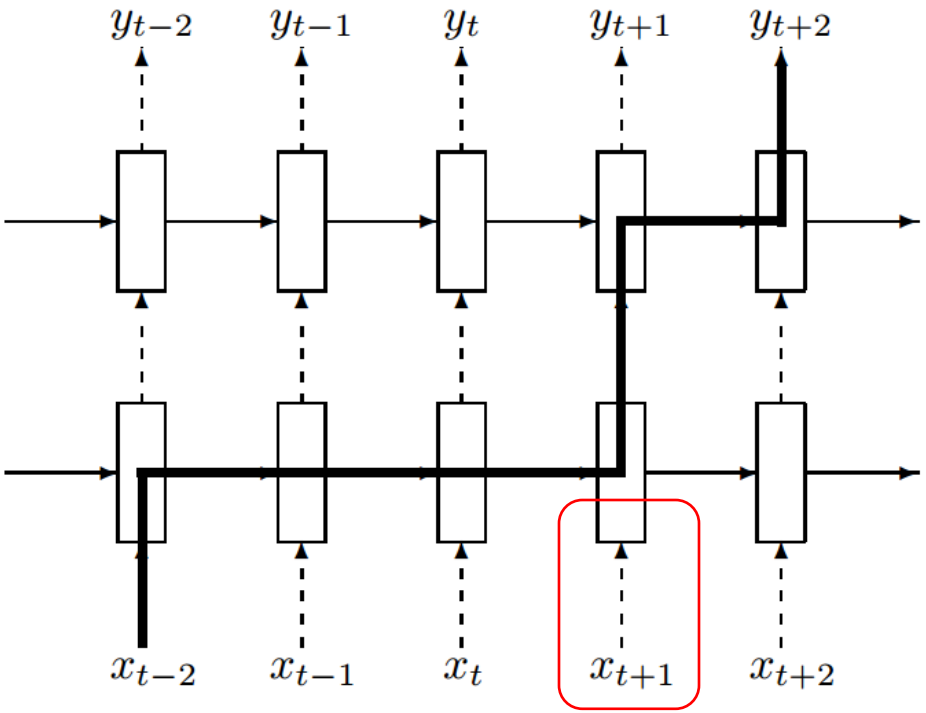


Figure 2: Regularized multilayer RNN. The dashed arrows indicate connections where dropout is applied, and the solid lines indicate connections where dropout is not applied.

Dropout in RNN

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. arXiv preprint arXiv:1409.2329.

- (Zaremba et al., 2014)



The information is corrupted by the dropout operator exactly L+1 time, and it is independent of the number of timesteps traversed by the information.

Figure 3: The thick line shows a typical path of information flow in the LSTM. The information is affected by dropout $L + 1$ times, where L is depth of network.

References: Dropout

- Gal, Y., & Ghahramani, Z. (2015). Bayesian convolutional neural networks with Bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*.
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.
- Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

How to solve overfitting problems

- **Dropout**
- **L2-regularization**

L2-regularization

- **(Softmax regression과 마찬가지로) 학습 가중치에 penalty를 부여**
  - $Cost = Loss + Penalty$
  - In general, $Penalty = \frac{1}{2}\lambda \|w\|^2$
  - 지나치게 특정 가중치의 크기가 증가하는 것에 제약을 걸어줌으로써, 가중치 벡터들이 고르게 퍼지도록 하는 효과

- **L1-regularization, Elastic net-regularization도 가능하나, 일반적으로 L2-regularization을 많이 사용**

Batch normalization

Batch normalization

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of The 32nd International Conference on Machine Learning (pp. 448-456).

- 노랑 -> 방법
- 파랑 -> 효과

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

Batch normalization

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of The 32nd International Conference on Machine Learning (pp. 448-456).

$$\mathrm{BN}_{\gamma,\beta} : x_{1\ldots m} \rightarrow y_{1\ldots m}$$

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

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch normalization

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of The 32nd International Conference on Machine Learning (pp. 448-456).

- **학습할 때에는,**
  - Activation function 직전에 배치로 들어오는 데이터를 그 batch 내에서 정규화
  - 정규화를 위해 해당 batch에서의 평균, 분산을 계산 ➔ 이를 저장해놓음
- **테스트할 때에는,**
  - 학습 단계에서 저장해놓은 이동평균 (moving average), 그리고 분산 추정치 (unbiased variance estimate)의 이동평균을 이용하여 전체 테스트 데이터를 정규화

Batch normalization

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of The 32nd International Conference on Machine Learning (pp. 448-456).

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**

Batch normalization

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of The 32nd International Conference on Machine Learning (pp. 448-456).

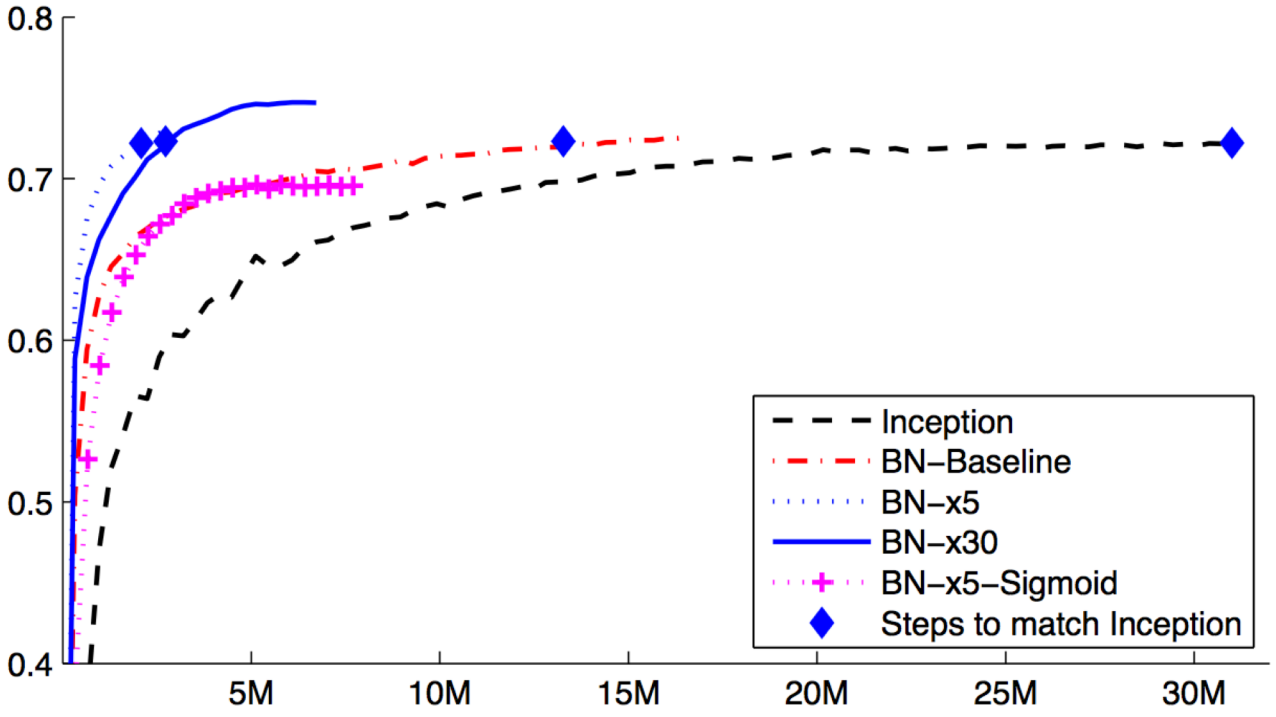


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Batch normalization

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of The 32nd International Conference on Machine Learning (pp. 448-456).

- **효과**
  - Batch normalization은 여러 측면으로 딥러닝 학습을 잘 하게 만드는 방법
  - Activation function 에 투입되는 값을 정규화하기 때문에, vanishing gradient 문제에도 적합하며 exploding gradient 문제도 완화시켜준다.
  - 논문에 의거하면 dropout을 대체할 수 있다고 함
    - Dropout보다 batch normalization을 사용할 때, 기대되는 학습시간이 적다.
    - 연구마다 dropout을 사용했는지 batch normalization을 사용했는지 확인해볼 필요가 있을 것이다.
    - 물론, 둘 다 사용하는 연구도 존재함

References: Batch normalization

- Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of The 32nd International Conference on Machine Learning (pp. 448-456).
- Laurent, C., Pereyra, G., Brakel, P., Zhang, Y., & Bengio, Y. (2016, March). Batch normalized recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on* (pp. 2657-2661). IEEE.

Various optimization methods

- **(Vanilla) gradient descent**
- **AdaGrad**
- **RMSProp**
- **AdaDelta**
- **Adam**

GD vs. SGD

Gradient Decent

학습데이터 — full-batch

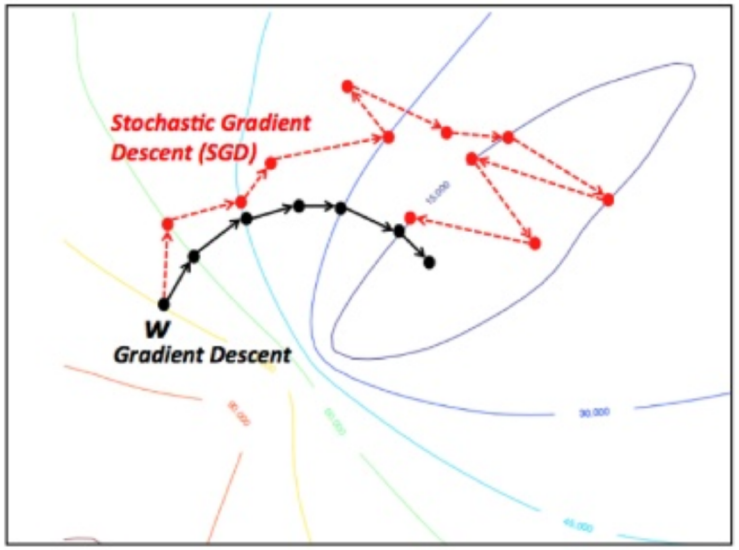
전부다 읽고나서
최적의 1스텝 간다.

Stochastic Gradient Decent

mini-batch
mini-batch
mini-batch
mini-batch
mini-batch

작은 토막마다
일단 1스텝간다.

느린 완벽보다 조금만 훑어보고
일단 빨리 가봅시다.

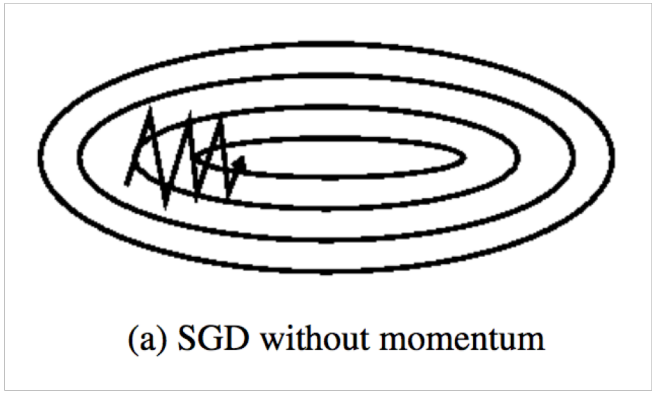


출처: 하용호@Kakao. 자습해도 모르겠던 딥러닝, 머리속에 인스톨시켜드립니다.

https://www.slideshare.net/yongho/ss-79607172

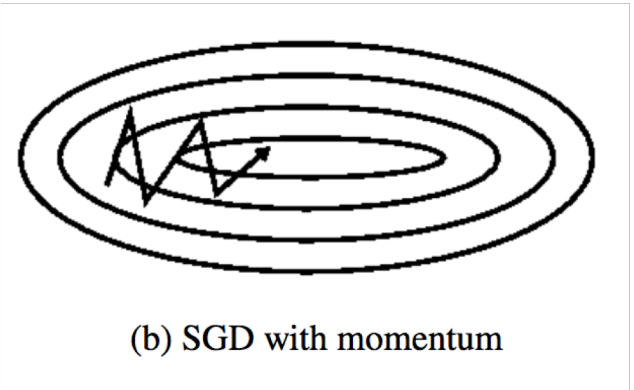
SGD without momentum vs. with momentum

- **Stochastic gradient descent**
  - $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta)$
    - η: learning rate
    - $J(\theta)$: cost (or loss) function



(a) SGD without momentum

- **Momentum**
  - $v \leftarrow \gamma v + \eta \nabla_\theta J(\theta),\ \theta \leftarrow \theta - v$
    - v: velocity vector
      - initial value of v = 0
    - γ: momentum term, friction factor
      - Usually 0.9



(b) SGD with momentum

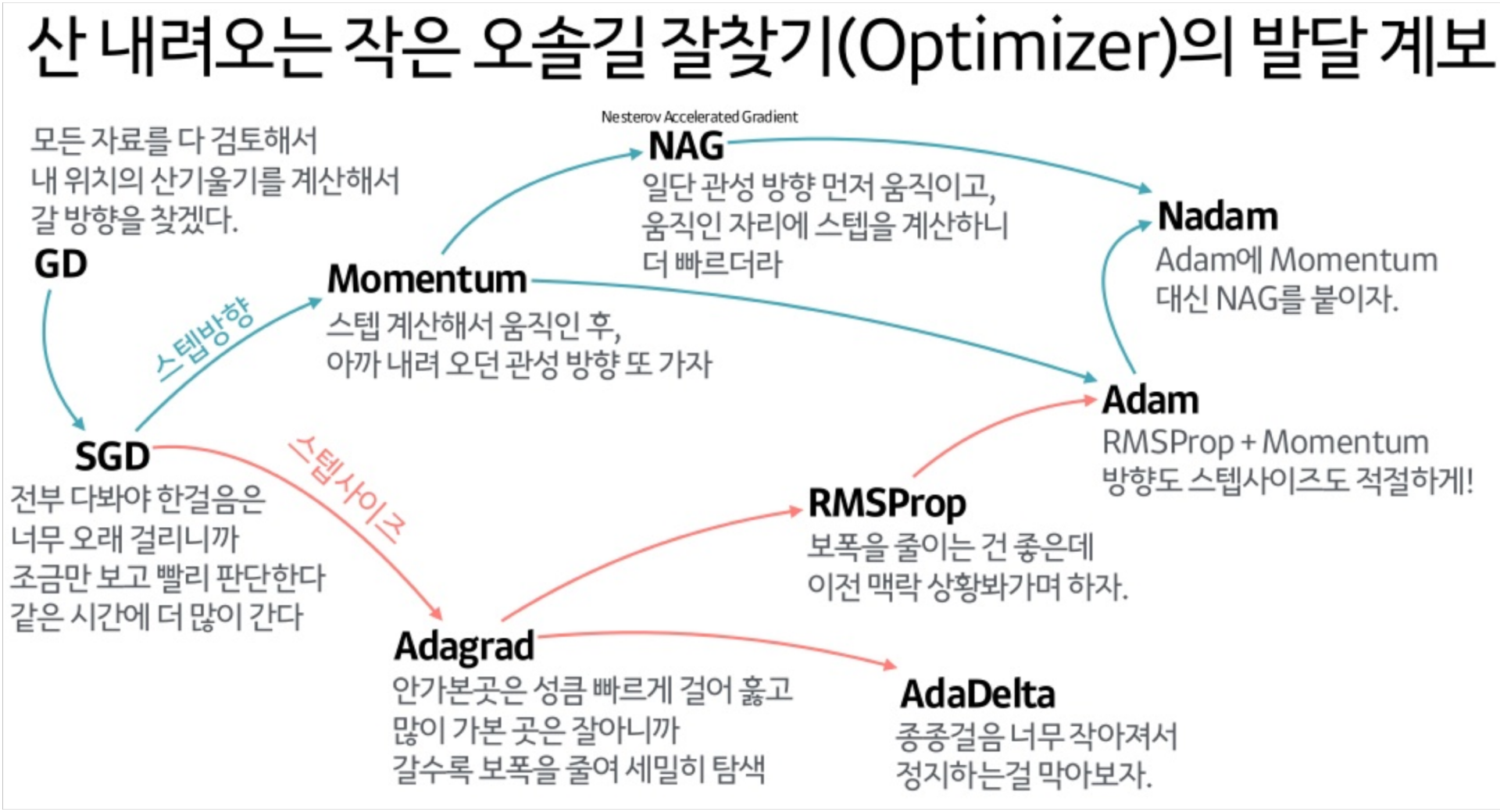
Momentum이 있을 때,
더 빨리 수렴하는 가능성 기대 ↑

Optimizers in TensorFlows

- **(Vanilla) gradient descent**
  - tf.train.GradientDescentOptimizer
- **AdaGrad**
  - tf.train.AdagradOptimizer
- **AdaDelta**
  - tf.train.AdadeltaOptimizer
- **RMSProp**
  - tf.train.RMSPropOptimizer
- **Adam**
  - tf.train.AdamOptimizer

Please see
https://www.tensorflow.org/api_guides/python/train#Optimizers

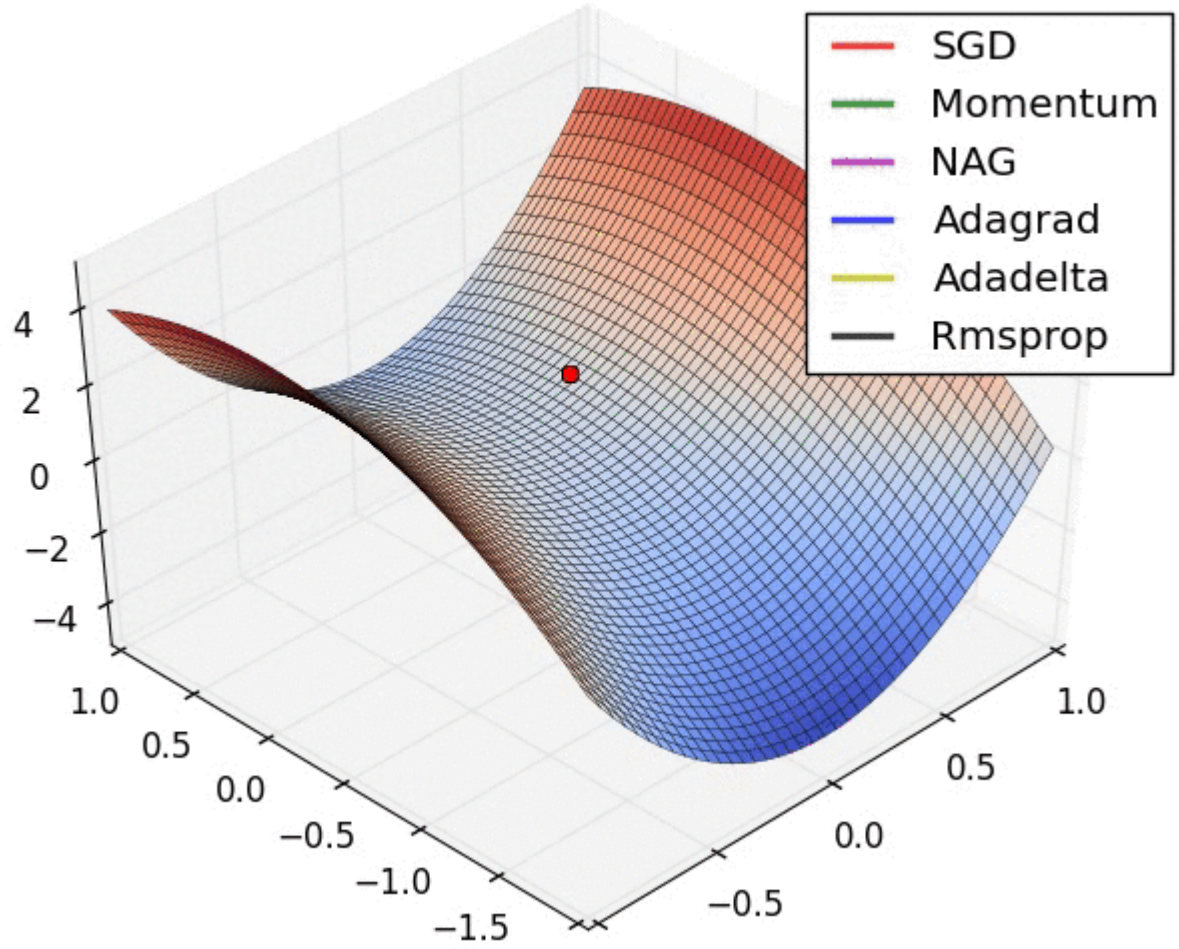
Various optimization methods



출처: 하용호@Kakao. 자습해도 모르겠던 딥러닝, 머리속에 인스톨시켜드립니다.

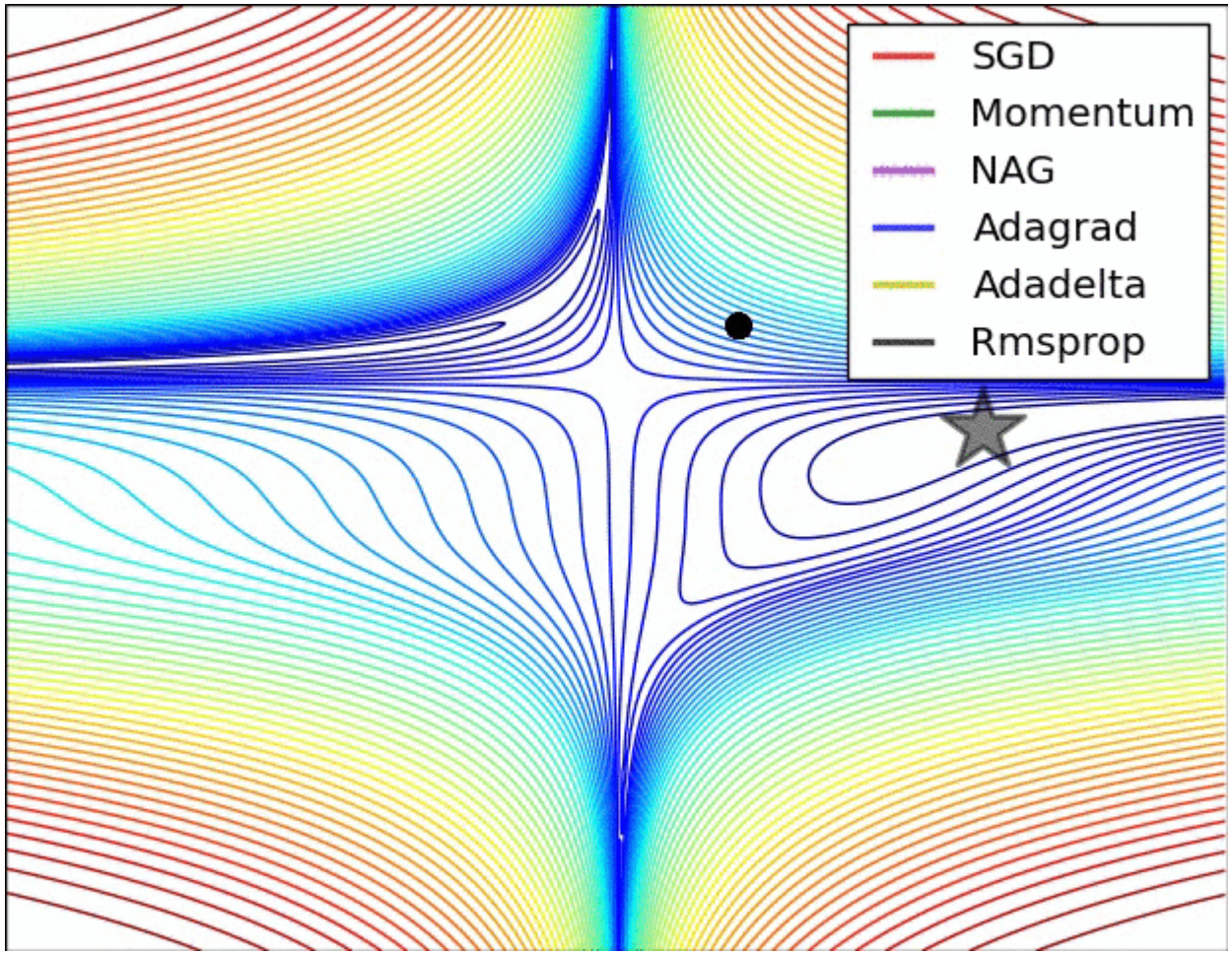
https://www.slideshare.net/yongho/ss-79607172

Various optimization methods



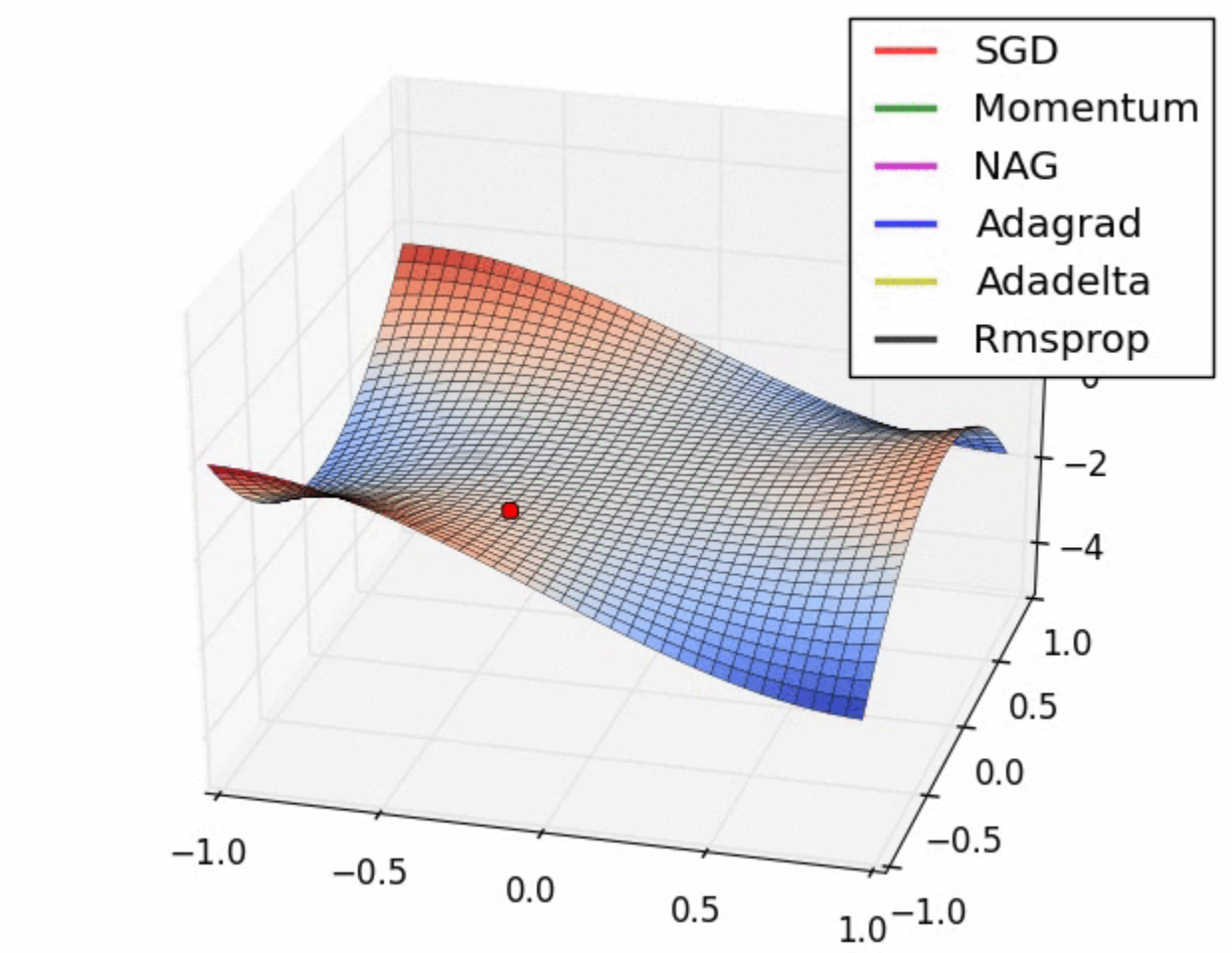
http://imgur.com/a/Hqolp

Various optimization methods



http://imgur.com/a/Hqolp

Various optimization methods



http://imgur.com/a/Hqolp

최적화 방법에 대해 더 잘 알고 싶다면,

- **Stanford CS231n**
  - http://cs231n.github.io/neural-networks-3/
  - 한글번역: http://aikorea.org/cs231n/neural-networks-3/
- **하용호@Kakao. 자습해도 모르겠던 딥러닝, 머리속에 인스톨시켜드립니다.**
  - https://www.slideshare.net/yongho/ss-79607172
- **On the origin of deep learning, Chapter 7**
  - Wang, H., Raj, B., & Xing, E. P. (2017). On the Origin of Deep Learning. *arXiv preprint arXiv:1702.07800*.
  - https://arxiv.org/abs/1702.07800
- **Sebastian Ruder's blog post**: http://ruder.io/optimizing-gradient-descent/
  - He also makes an article in arXiv.: http://arxiv.org/abs/1609.04747
  - https://www.slideshare.net/ssuser77b8c6/an-overview-of-gradient-descent-optimization-algorithms (nice review presentation of this paper)