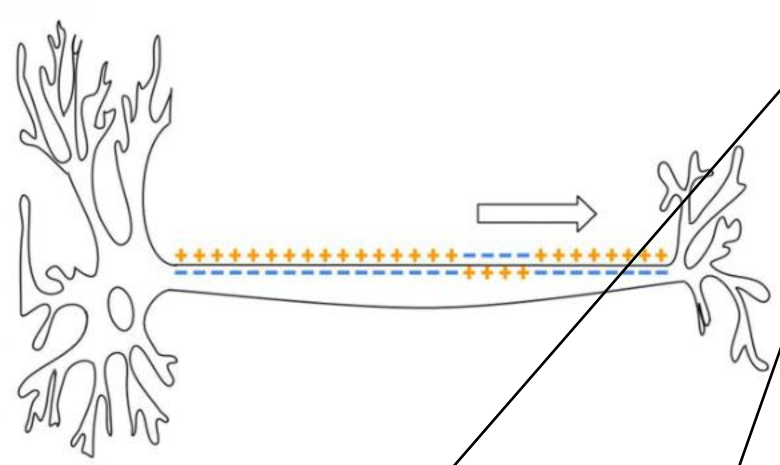
Technique for improving the speed, performance, and stability of artificial neural networks

Dongmyeong Lee

- Sigmoid/ReLU, Softmax
- Perceptron
- Weight initialization, Optimization
- Regularization : Dropout
- Batch Normalization

Sigmoid/ReLU, Softmax(1) - Linearity

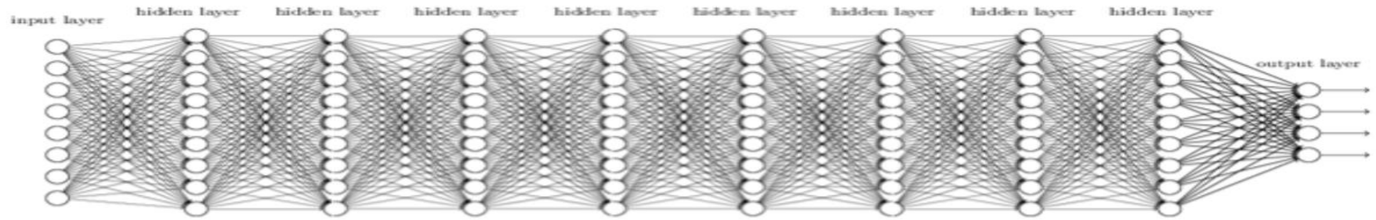
Action Potential(Threshold)



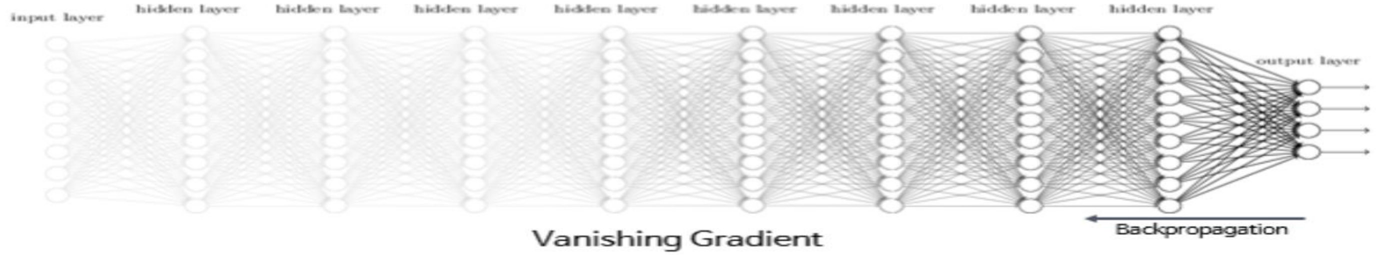
$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} = \max\{0, x\} = x\mathbf{1}_{x>0}$$

Name	Plot	Equation	Derivative (with respect to x)	Range	Order of continuity	Monotonic	Monotonic derivative	Approximates identity near the origin
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	C^∞	Yes	Yes	Yes
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}	Yes	No	No
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f'(x) = f(x)(1-f(x))$	$(0, 1)$	C^∞	Yes	No	No
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^∞	Yes	No	Yes
Rectified linear unit (ReLU)[12]		$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} = \max\{0, x\} = x\mathbf{1}_{x>0}$	$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	$[0, \infty)$	C^0	Yes	Yes	No
Gaussian Error Linear Unit (GELU)[7]		$f(x) = x\Phi(x) = x(1 + \text{erf}(x/\sqrt{2}))/2$	$f'(x) = \Phi(x) + x\phi(x)$	$(\approx -0.17, \infty)$	C^∞	No	No	No
SoftPlus[13]		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞	Yes	Yes	No
Exponential linear unit (ELU)[14]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C^1 & \text{when } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$	Yes iff $\alpha \geq 0$	Yes iff $0 \leq \alpha \leq 1$	Yes iff $\alpha = 1$
Scaled exponential linear unit (SELU)[15]		$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ with $\lambda = 1.0507$ and $\alpha = 1.67326$	$f'(\alpha, x) = \lambda \begin{cases} \alpha(e^x) & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\lambda\alpha, \infty)$	C^0	Yes	No	No
Leaky rectified linear unit (Leaky ReLU)[16]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Parameteric rectified linear unit (PReLU)[17]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$[2]	C^0	Yes iff $\alpha \geq 0$	Yes	Yes iff $\alpha = 1$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	C^∞	Yes	No	Yes



Deep Neural Network



Vanishing Gradient

Sigmoid/ReLU, Softmax(2)

The distribution [edit]

The Boltzmann distribution is a probability distribution that gives the probability of a certain state as a function of that state's energy and temperature of the system to which the distribution is applied.[6] It is given as

$$p_i = \frac{1}{Q} e^{-\varepsilon_i/kT} = \boxed{\frac{e^{-\varepsilon_i/kT}}{\sum_{j=1}^{M} e^{-\varepsilon_j/kT}}}$$

where p_i is the probability of state i, ε_i the energy of state i, k the Boltzmann constant, T the temperature of the system and M is the number of all states accessible to the system of interest.[6][5] Implied parentheses around the denominator kT are omitted for brevity. The normalization denominator Q (denoted by some authors by Z) is the canonical partition function

$$Q = \sum_{i=1}^{M} e^{-\varepsilon_i/kT}$$

It results from the constraint that the probabilities of all accessible states must add up to 1.

Softmax function

From Wikipedia, the free encyclopedia

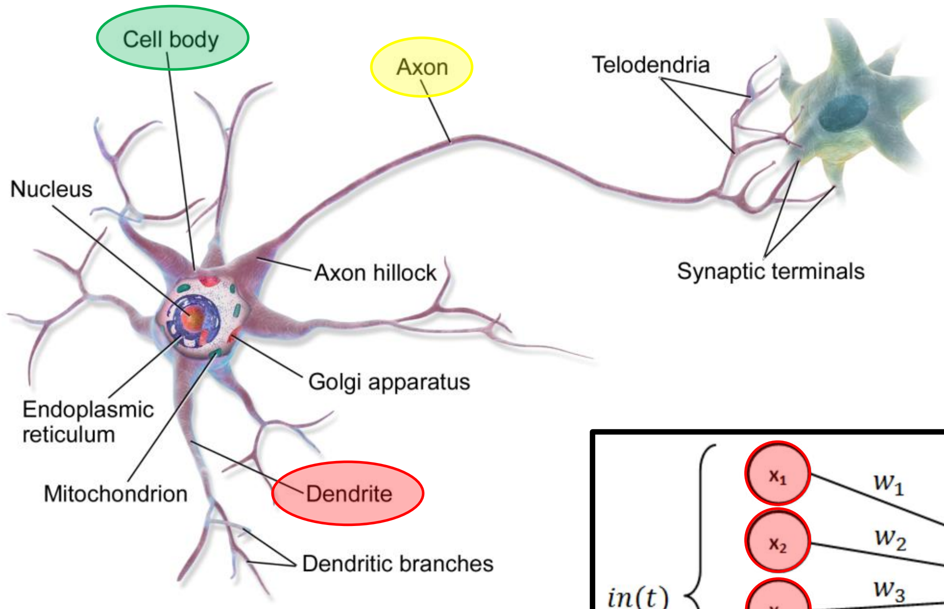
This article is about the smooth approximation of arg max. For the smooth approximation of max, see LogSumExp.

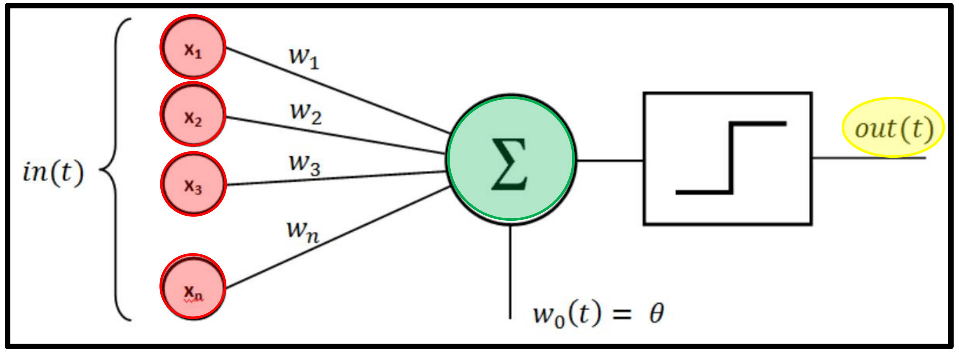
In mathematics, the **softmax function,** also known as **softargmax**[1]:184 or **normalized exponential function,**[2]:198 is a function that takes as input a vector z of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval $(0, 1)$, and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities. Softmax is often used in neural networks, to map the non-normalized output of a network to a probability distribution over predicted output classes.

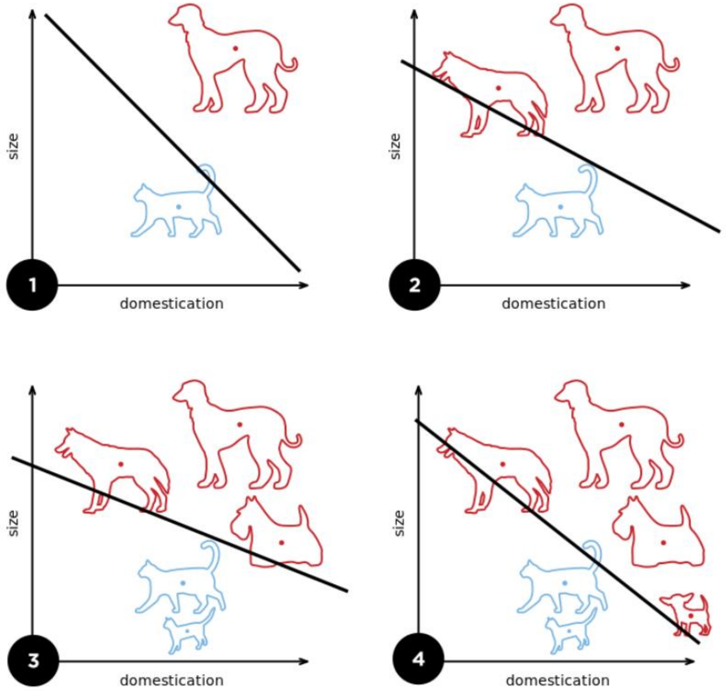
The standard (unit) softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is defined by the formula

$$\boxed{\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}} \text{ for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

Perceptron







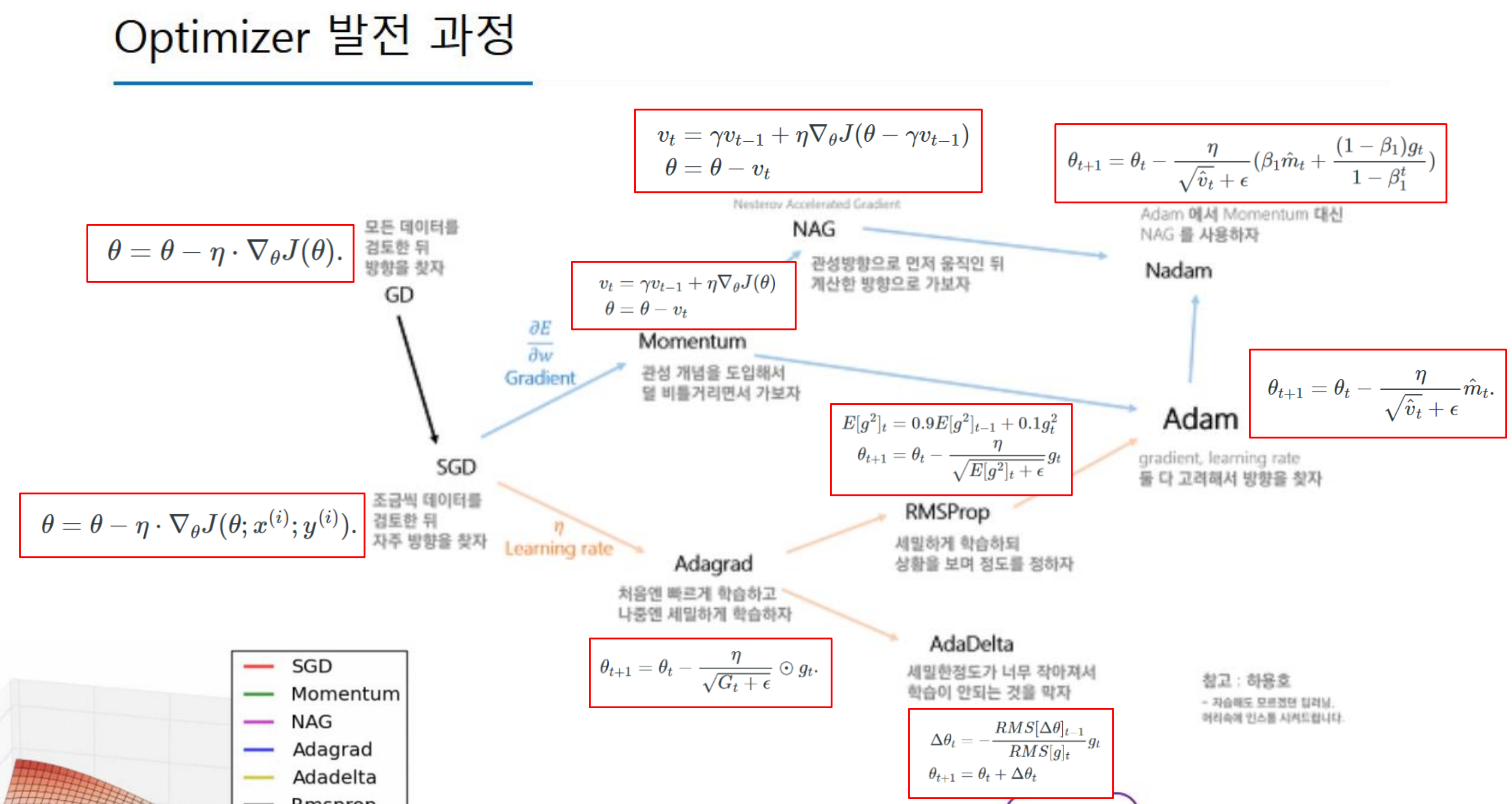
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

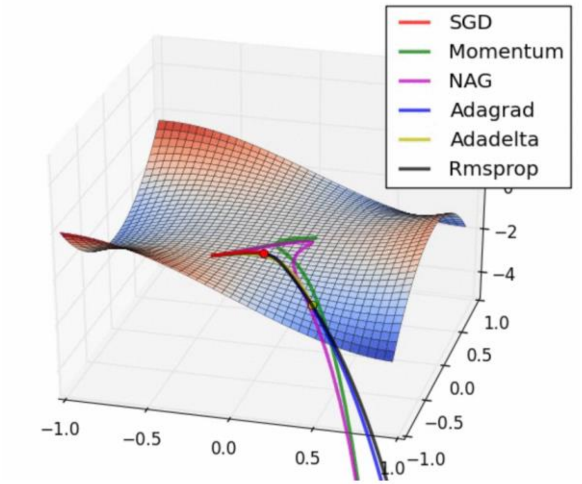
$f(\mathbf{x}; \boxed{w_1, w_2, \ldots, w_m})$

LOOP

X	f(X)	Y
X_1	$f(X_1)$	Y_1
X_2	$f(X_2)$	Y_2
X_3	$f(X_3)$	Y_3
…	…	…
…	…	…
X_{n-2}	$f(X_{n-2})$	Y_{n-2}
X_{n-1}	$f(X_{n-1})$	Y_{n-1}
X_n	$f(X_n)$	Y_n

X	Y	Loss
X_1	Y_1	$Y_1 - f(X_1)$
X_2	Y_2	$Y_2 - f(X_2)$
X_3	Y_3	$Y_3 - f(X_3)$
…	…	
…	…	
X_{n-2}	Y_{n-2}	$Y_{n-2} - f(X_{n-2})$
X_{n-1}	Y_{n-1}	$Y_{n-1} - f(X_{n-1})$
X_n	Y_n	$Y_n - f(X_n)$





Gradient Descent Optimization Algorithms at Saddle Point

$$w^+ = w - \boxed{\eta} * \boxed{\frac{\partial E}{\partial w}}$$

learning rate :
한번에 얼마나 학습할지

gradient :
어떤 방향으로 학습할지

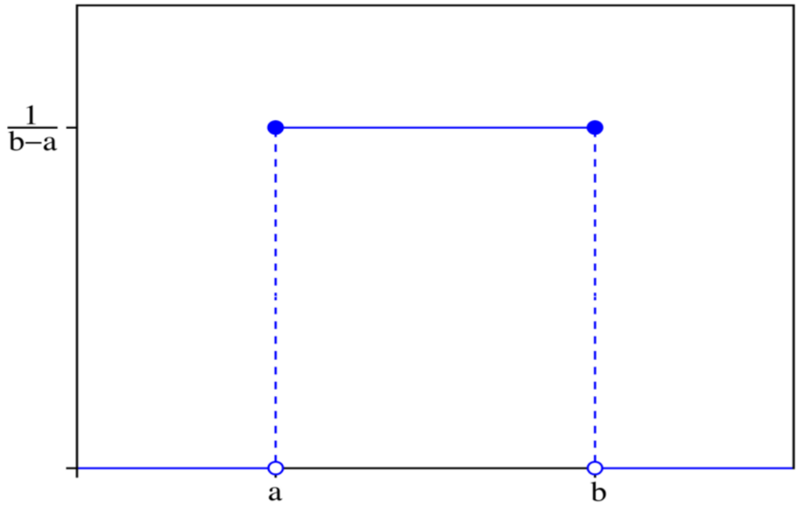
Weight initialization, Optimization(2)

`torch.nn.init.xavier_uniform_(tensor: torch.Tensor, gain: float = 1.0)` → torch.Tensor [SOURCE]

Fills the input *Tensor* with values according to the method described in *Understanding the difficulty of training deep feedforward neural networks* - Glorot, X. & Bengio, Y. (2010), using a uniform distribution. The resulting tensor will have values sampled from $\mathcal{U}(-a, a)$ where

$$a = \text{gain} \times \sqrt{\frac{6}{\text{fan\_in} + \text{fan\_out}}}$$

Also known as Glorot initialization.

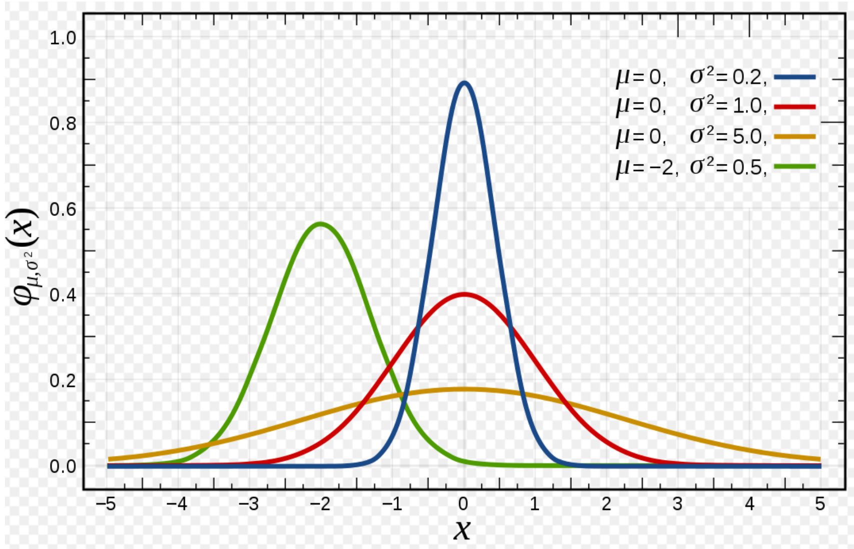


`torch.nn.init.xavier_normal_(tensor: torch.Tensor, gain: float = 1.0)` → torch.Tensor [SOURCE]

Fills the input *Tensor* with values according to the method described in *Understanding the difficulty of training deep feedforward neural networks* - Glorot, X. & Bengio, Y. (2010), using a normal distribution. The resulting tensor will have values sampled from $\mathcal{N}(0, \text{std}^2)$ where

$$\text{std} = \text{gain} \times \sqrt{\frac{2}{\text{fan\_in} + \text{fan\_out}}}$$

Also known as Glorot initialization.



Regularization : Dropout

Dilution (neural networks)

From Wikipedia, the free encyclopedia
(Redirected from Dropout (neural networks))

This article's **factual accuracy is disputed**. Relevant discussion may be found on the talk page. Please help to ensure that disputed statements are reliably sourced. *(April 2020) (Learn how and when to remove this template message)*

Dilution (also called **Dropout**) is a regularization technique for reducing overfitting in artificial neural networks by preventing complex co-adaptations on training data. It is an efficient way of performing model averaging with neural networks.[1] The term *dilution* refers to the thinning of the weights.[2] The term *dropout* refers to randomly "dropping out", or omitting, units (both hidden and visible) during the training process of a neural network.[3][4][1] Both the thinning of weights and dropping out units trigger the same type of regularization, and often the term *dropout* is used when referring to the dilution of weights.

$$y_i = \sum_j w_{ij} x_j$$

- y_i – output from node i
- w_{ij} – real weight before dilution, also called the Hebb connection strength
- x_i – input from node j

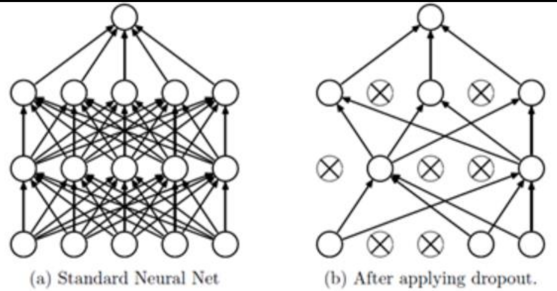
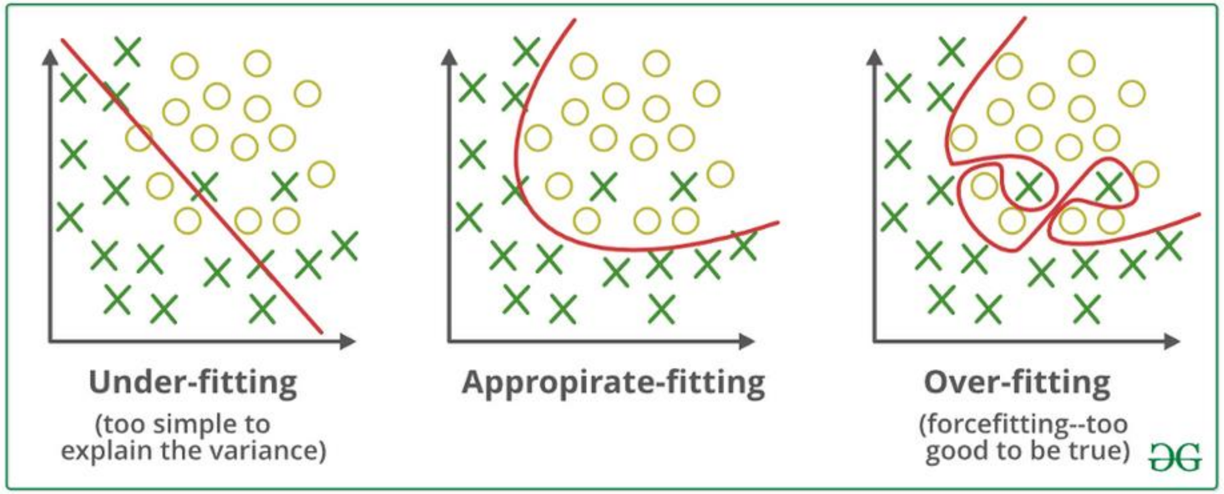


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

$$\hat{\mathbf{w}}_j = \begin{cases} \mathbf{w}_j, & \text{with } P(c) \\ \mathbf{0}, & \text{otherwise} \end{cases}$$

- $P(c)$ – the probability c to keep a row in the weight matrix
- $\mathbf{w}_j$ – real row in the weight matrix before dropout
- $\hat{\mathbf{w}}_j$ – diluted row in the weight matrix



Batch Normalization

2 Towards Reducing Internal Covariate Shift

We define *Internal Covariate Shift* as the change in the distribution of network activations due to the change in network parameters during training. To improve the training, we seek to reduce the internal covariate shift. By fixing the distribution of the layer inputs x as the training progresses, we expect to improve the training speed. It has been long known (LeCun et al., 1998b; Wiesler & Ney, 2011) that the network training converges faster if its inputs are whitened – i.e., linearly transformed to have zero means and unit variances, and decorrelated. As each layer observes the inputs produced by the layers below, it would be advantageous to achieve the same whitening of the inputs of each layer. By whitening the inputs to each layer, we would take a step towards achieving the fixed distributions of inputs that would remove the ill effects of the internal covariate shift.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

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