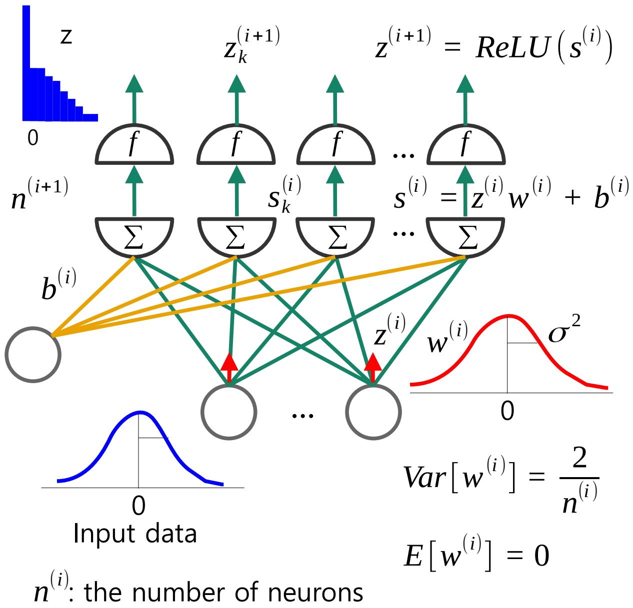


8. Weights Initialization

Part 1: Observation of the distribution of hidden layer outputs

This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

[MXDL-8-01]

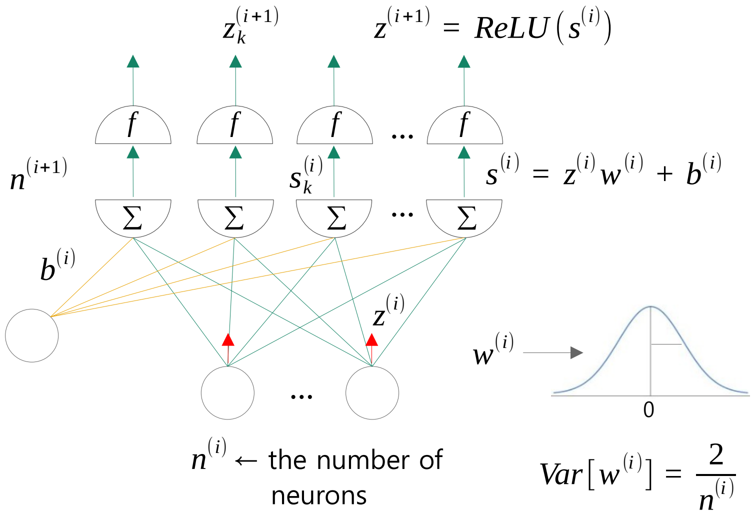
1. Parameter space and error
2. Observation of the distribution of hidden layer outputs according to initial weights
3. Formula for the variance of output

[MXDL-8-02]

4. Xavier Glorot initialization
   - Paper
   - Forward direction
   - Backward direction
   - Uniform distribution
   - Xavier Glorot intializer in Keras

[MXDL-8-03]

5. Kaiming He initialization
   - Paper
   - Forward direction
   - Backward direction
   - Uniform distribution
   - Kaiming He intializer in Keras

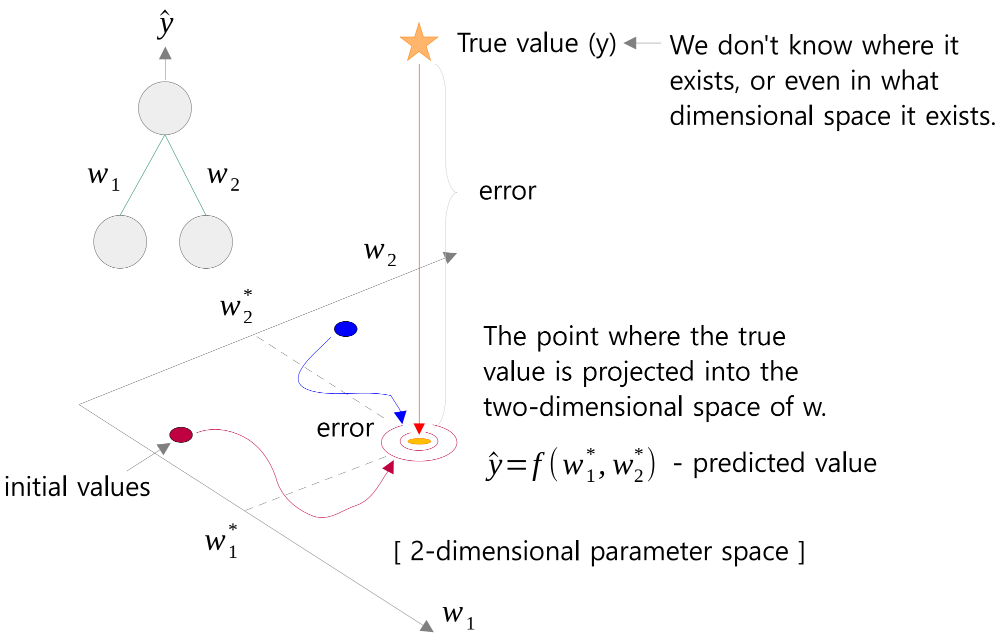


$$Var\left[w^{(i)}\right] = \frac{2}{n^{(i)}}$$

$$E\left[w^{(i)}\right] = 0$$

■ Parameter space and error

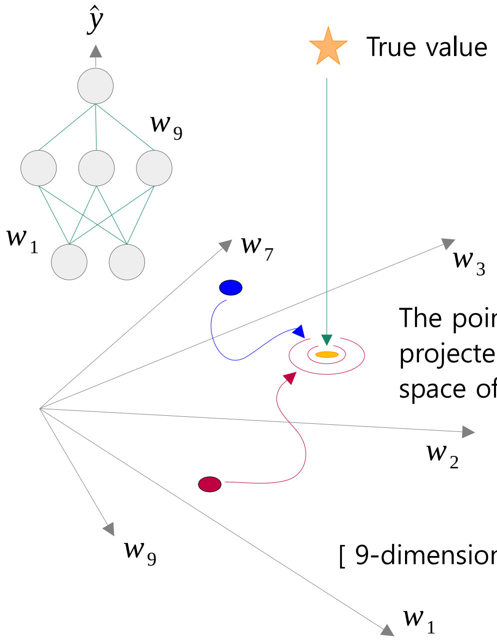
- Properly setting the number and initial values of parameters is a critical step in training a neural network.
- Since we don't know where the true values y are, we don't know how wide or deep to build a model. The number of parameters w is determined by the width and depth of a model. Models with a small number of parameters may have large errors and underfitting, while models with a large number of parameters may have small errors but overfitting.
- Additionally, the initial values of parameters also affect the performance of models. Depending on the initial values, vanishing or exploding gradients problems may occur, and local minimum problems may also increase.



The point where the true value is projected into the two-dimensional space of w.

$\hat{y}=f(w_1^*, w_2^*)$ - predicted value

[2-dimensional parameter space]



The point where the true value is projected into the 9-dimensional space of w.

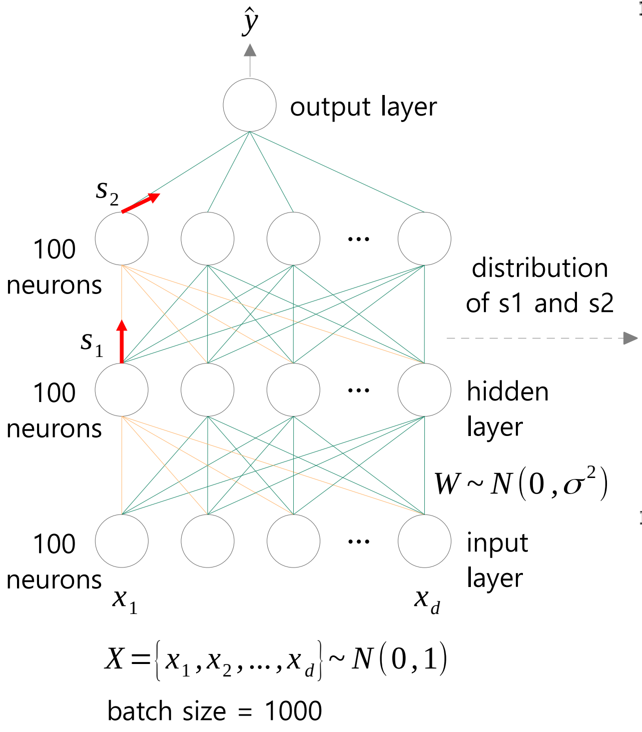
$\hat{y}=f(w_1^*, w_2^*, \dots, w_9^*)$

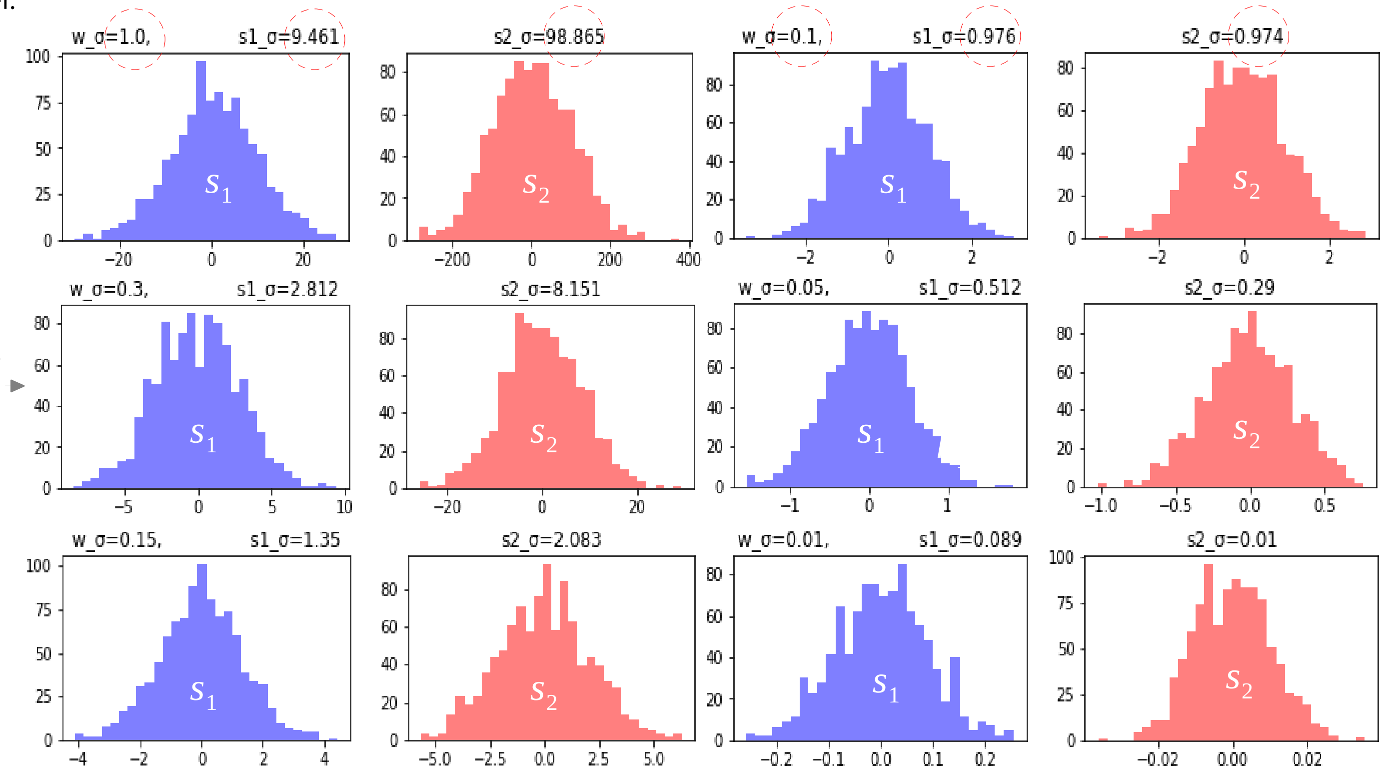
[9-dimensional parameter space]

MX-AI

Distribution of hidden layer outputs according to initial weights

- The distribution of the input data X is a normal distribution with mean 0 and variance 1, and the distribution of the initial weights in the hidden layer is a normal distribution with mean 0 and variance σ^2. Let's observe the distribution of hidden layer outputs as the σ of initial w changes.
- The distribution of a hidden layer output values changes depending on the initial values of w.
- When w_σ is 1.0, the variances of s1 and s2 are significantly different. When w_σ is 0.1, the variances of s1 and s2 are similar.
- The more similar the variance, the better.

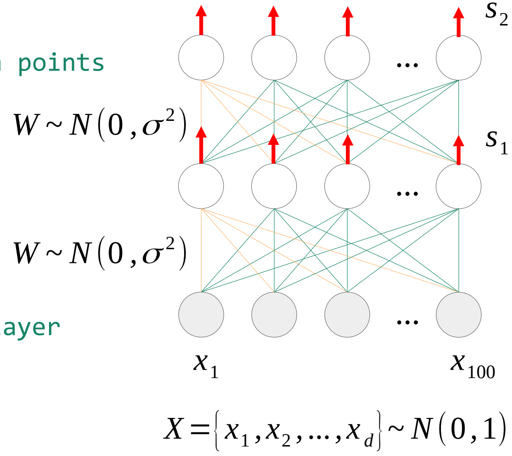




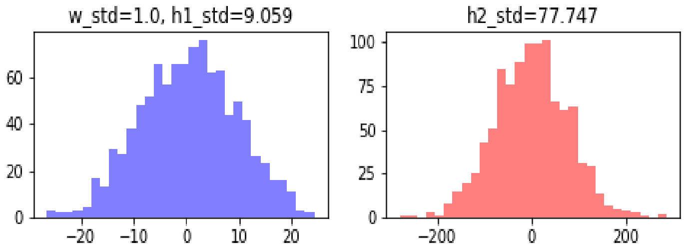
MX-AI

■ Observation of the distribution of hidden layer outputs according to initial weights

```
# [MXDL-8-01] 1.h_distribution.py
# Observe the distribution of hidden layer outputs according
# to initial weights
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import initializers
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt

# Generate a simple dataset with 1000 data points
x = np.random.normal(size=(1000, 100))

# Create an ANN model with 2 hidden layers
n_input = x.shape[1]
n_hidden = 100

# Let's check the distribution of hidden layer
# outputs by changing std in N(0, std).
std = [1.0, 0.3, 0.15, 0.1, 0.05, 0.01]
for sigma in std:
    # Create an ANN model
    w0 = initializers.RandomNormal(mean=0.0, stddev=sigma)
    x_input = Input(batch_shape=(None, n_input))
    s1 = Dense(n_hidden, kernel_initializer = w0)(x_input)
    s2 = Dense(n_hidden, kernel_initializer = w0)(s1)
```

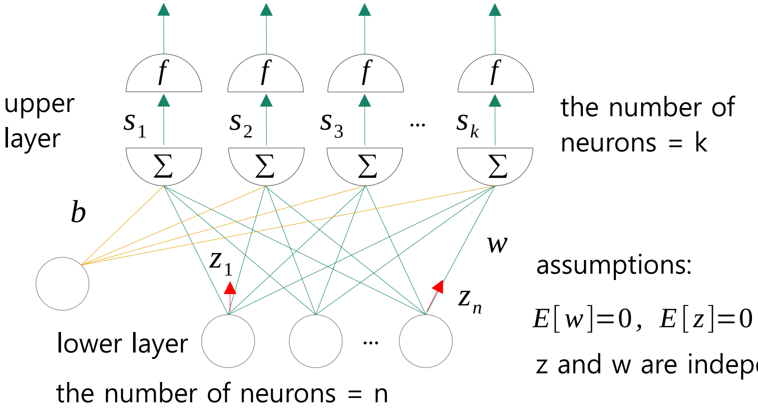



```
    s1_model = Model(x_input, s1)
    s2_model = Model(x_input, s2)

    i = 0
    s1_out = s1_model.predict(x, verbose=0)[:, i]
    s2_out = s2_model.predict(x, verbose=0)[:, i]

    # Check the distribution of the hidden layer outputs.
    plt.figure(figsize=(8,2))
    plt.subplot(121)
    plt.hist(s1_out, bins=30, color='blue', alpha=0.5)
    plt.title('w_std=' + str(sigma) + ', \
              s1_std=' + str(s1_out.std().round(3)))

    plt.subplot(122)
    plt.hist(s2_out, bins=30, color='red', alpha=0.5)
    plt.title('s2_std=' + str(s2_out.std().round(3)))
    plt.show()
```



MX-AI

■ Formula for the variance of output



assumptions:

$E[w]=0,\ E[z]=0$

z and w are independent.

$$S = Z \cdot W + B$$

$$S = \begin{pmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,n} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,n} \\ & \cdots & & \\ z_{m,1} & z_{m,2} & & z_{m,n} \end{pmatrix} \cdot \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w_{2,1} & w_{2,2} & \cdots & w_{1,k} \\ & \cdots & & \\ w_{n,1} & w_{n,2} & \cdots & w_{n,k} \end{pmatrix} + [b_1, b_2, b_3, \ldots, b_k]$$

(m, n) (m: batch size) (n, k)

$$S = \begin{array}{c} \begin{pmatrix} \boxed{\begin{matrix} \sum_{j=1}^{n} z_{1,j} w_{j,1} + b_1 \\ \sum_{j=1}^{n} z_{2,j} w_{j,1} + b_1 \\ \vdots \\ \sum_{j=1}^{n} z_{m,j} w_{j,1} + b_1 \end{matrix}} & \begin{matrix} \sum_{j=1}^{n} z_{1,j} w_{j,2} + b_2 \\ \sum_{j=1}^{n} z_{2,j} w_{j,2} + b_2 \\ \vdots \\ \sum_{j=1}^{n} z_{m,j} w_{j,2} + b_2 \end{matrix} & \begin{matrix} \cdots \\ \cdots \\ \\ \cdots \end{matrix} & \begin{matrix} \sum_{j=1}^{n} z_{1,j} w_{j,k} + b_k \\ \sum_{j=1}^{n} z_{2,j} w_{j,k} + b_k \\ \vdots \\ \sum_{j=1}^{n} z_{m,j} w_{j,k} + b_k \end{matrix} \end{pmatrix} \\ \begin{matrix} s_1 & \qquad\qquad s_2 & \qquad\qquad\qquad s_k \end{matrix} \end{array} \quad (m,\ k)$$

$Var[s_1]$

Let: $z_i = z_{1:m,i}$ (b = constant)

$$Var\left[\sum_{j=1}^{n} z_j w_{j,1}\right] = Var[z_1 w_{1,1} + z_2 w_{2,1} + \ldots + z_n w_{n,1}]$$

$$= Var[z_1 w_{1,1}] + Var[z_2 w_{2,1}] + \ldots + Var[z_n w_{n,1}]$$

Assumptions: s, z and w are i.i.d.

$$Var[z_1 w_{1,1}] \approx Var[z_2 w_{2,1}] \approx \ldots \approx Var[z_n w_{n,1}]$$

$$Var[s_1] \approx Var[s_2] \approx \ldots \approx Var[s_k]$$

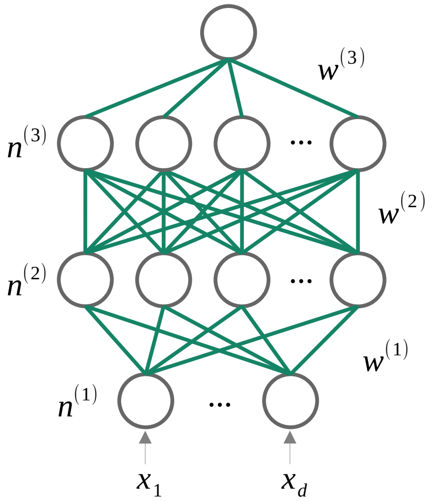
$$Var(s_1) = Var\left[\sum_{j=1}^{n} z_{1:m,j} w_{j,1}\right] = n \cdot Var[z_1 w_{1,1}] = n \cdot Var[z_2 w_{2,1}] = \ldots$$

$$\boxed{Var[S] = n \cdot Var[Z \cdot W]}$$

* Reference: https://www.pinecone.io/learn/weight-initialization/
https://towardsdatascience.com/xavier-glorot-initialization-in-neural-networks-math-proof-4682bf5c6ec3

$$w^{(i)} \sim N\left(0,\ \frac{2}{n^{(i)} + n^{(i+1)}}\right)$$

$$w^{(i)} \sim U\left(-\sqrt{\frac{6}{n^{(i)} + n^{(i+1)}}},\ \sqrt{\frac{6}{n^{(i)} + n^{(i+1)}}}\right)$$



$$x = \{x_1, x_2, \dots, x_{50}\} \sim N(0, 1)$$

8. Weights Initialization

Part 2: Xavier Glorot initializer

This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

- Xavier Glorot paper (2010)

Understanding the difficulty of training deep feedforward neural networks

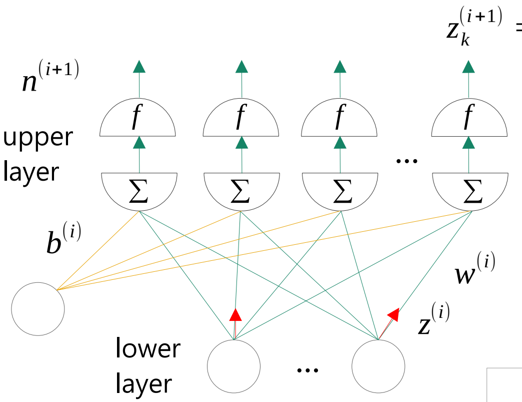
Xavier Glorot Yoshua Bengio
DIRO, Universite de Montr ´ eal, Montr ´ eal, Qu ´ ebec, Canada

Abstract

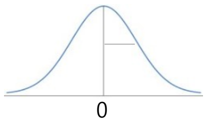
Whereas before 2006 it appears that deep multilayer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent relative successes and help design better algorithms in the future. We first observe the influence of the non-linear activations functions. We find that the logistic sigmoid activation is unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation. Surprisingly, we find that saturated units can move out of saturation by themselves, albeit slowly, and explaining the plateaus sometimes seen when training neural networks. We find that a new non-linearity that saturates less can often be beneficial. Finally, we study how activations and gradients vary across layers and during training, with the idea that training may be more difficult when the singular values of the Jacobian associated with each layer are far from 1. Based on these considerations, we propose a **new initialization scheme** that brings substantially faster convergence.

Xavier Glorot initializer: Forward direction

- During forward propagation, we would like the variance of the activations to remain constant across layers.
- Let's find the optimal variance of the initial weight distribution in the forward direction.



$n^{(i)} \leftarrow$ the number of neurons



$$w^{(i)} \sim N\left(0, \frac{1}{n^{(i)}}\right)$$

Assumption: $E[z^{(i)}] = E[w^{(i)}] = 0$

Objective: $\forall i, Var[z^{(i)}] = Var[z^{(i+1)}] \leftarrow$ We want the variance of the activations to remain constant across layers.

$Var[z^{(i+1)}] = Var[f(s^{(i+1)})] \approx Var[s^{(i+1)}] \leftarrow$ Assume f is linear near 0. (ex: sigmoid, tanh)

$= n^{(i)} Var[z^{(i)} w^{(i)}] \leftarrow$ by the formula for the variance of output

$$= n^{(i)} [Var[z^{(i)}]Var[w^{(i)}] + Var[z^{(i)}](E[w^{(i)}])^2 + Var[w^{(i)}](E[z^{(i)}])^2]$$

$$= n^{(i)} Var[z^{(i)}]Var[w^{(i)}]$$

$$Var[z^{(i+1)}] = n^{(i)} Var[z^{(i)}] \cdot Var[w^{(i)}]$$

$$Var[w^{(i)}] = \frac{1}{n^{(i)}}$$

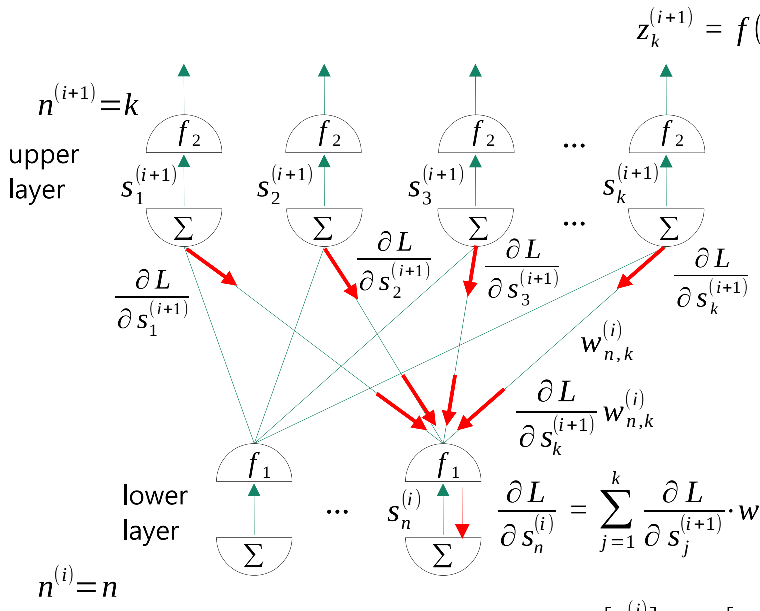
For example, if the number of neurons in the lower layer is 100, the initial weights of the upper layer are set by drawing them from a normal distribution with a mean of 0 and a variance of 0.01. N(0, 1/100)

* Reference: https://www.pinecone.io/learn/weight-initialization/
https://towardsdatascience.com/xavier-glorot-initialization-in-neural-networks-math-proof-4682bf5c6ec3

* reference: https://towardsdatascience.com/xavier-glorot-initialization-in-neural-networks-math-proof-4682bf5c6ec3

Xavier Glorot initializer: Backward direction

- During backpropagation, we would like the variance of the gradients of loss with respect to the output s to remain constant across layers.



$$\frac{\partial L}{\partial s_n^{(i)}} = \sum_{j=1}^{k} \frac{\partial L}{\partial s_j^{(i+1)}} \cdot w_{n,j}^{(i)} \cdot f_1{}'(s_n^{(i)})$$

Assumption: $E[s^{(i)}] = E[w^{(i)}] = 0$

Objective:

$$\forall i, Var\left[\frac{\partial L}{\partial s^{(i+1)}}\right] = Var\left[\frac{\partial L}{\partial s^{(i)}}\right]$$

We want the variance of the gradients to remain constant across layers.

$$\frac{\partial L}{\partial s_n^{(i)}} = \sum_{j=1}^{k} \frac{\partial L}{\partial s_j^{(i+1)}} \cdot w_{n,j}^{(i)} \cdot f_1{}'(s_n^{(i)})$$

Assume f_1 is linear near 0. (ex: sigmoid, tanh)
$f_1(x)=x$, $f'_1(x)=1$

- Chan rule (single path)

$$s^{(i+1)} = f_1(s^{(i)}) \cdot w^{(i)} + b^{(i)}$$

$$\frac{\partial L}{\partial s^{(i)}} = \frac{\partial L}{\partial s^{(i+1)}} \frac{\partial s^{(i+1)}}{\partial f_1(s^{(i)})} \frac{\partial f_1(s^{(i)})}{\partial s^{(i)}}$$

$$Var\left[\frac{\partial L}{\partial s_n^{(i)}}\right] = Var\left[\sum_{j=1}^{k} \frac{\partial L}{\partial s_j^{(i+1)}} \cdot w_{n,j}^{(i)}\right]$$

$$Var\left[\frac{\partial L}{\partial s^{(i)}}\right] = n^{(i+1)} Var\left[\frac{\partial L}{\partial s^{(i+1)}} \cdot w^{(i)}\right]$$

$$= n^{(i+1)}\left[Var\left[\frac{\partial L}{\partial s^{(i+1)}}\right] Var[w^{(i)}] + Var\left[\frac{\partial L}{\partial s^{(i+1)}}\right](\underbrace{E[w^{(i)}]}_{0})^2 + Var[w^{(i)}]\left(\underbrace{E\left[\frac{\partial L}{\partial s^{(i+1)}}\right]}_{0}\right)^2\right]$$

(assumption)

$$Var\left[\frac{\partial L}{\partial s^{(i)}}\right] = n^{(i+1)} Var\left[\frac{\partial L}{\partial s^{(i+1)}}\right] \cdot Var[w^{(i)}]$$

$$\boxed{Var[w^{(i)}] = \frac{1}{n^{(i+1)}}}$$

- Find the average of the forward and backward directions.

Backward direction → $n^{(i+1)} Var[w^{(i)}] = 1$

Forward direction → $n^{(i)} Var[w^{(i)}] = 1$

→ $$\boxed{Var[w^{(i)}] = \frac{2}{n^{(i)} + n^{(i+1)}}}$$

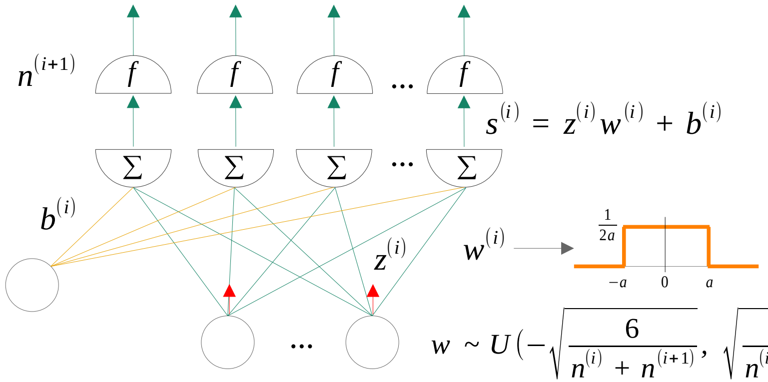
For example, if the number of neurons in the lower and upper layer is 100 each, the initial weights of the upper layer are set by drawing them from a normal distribution with a mean of 0 and a variance of 0.01. N(0, 0.01)

MX-AI

Xavier Glorot initializer: Uniform distribution

- Now that we know the optimal variance for the normal distribution, we can also find the range for the uniform distribution.

$$z^{(i+1)} = f(s^{(i)})$$



$$s^{(i)} = z^{(i)} w^{(i)} + b^{(i)}$$

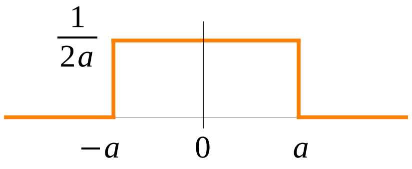
$$w \sim U\left(-\sqrt{\frac{6}{n^{(i)} + n^{(i+1)}}}, \sqrt{\frac{6}{n^{(i)} + n^{(i+1)}}}\right)$$

$n^{(i)} \leftarrow$ the number of neurons

assumption: $E[z^{(i)}] = E[w^{(i)}] = 0$

* reference: https://towardsdatascience.com/xavier-glorot-initialization-in-neural-networks-math-proof-4682bf5c6ec3

- continuous uniform distribution



$$\begin{cases} g(x) = \frac{1}{2a}, & \text{if } -a \le x \le a \\ g(x) = 0 & \text{otherwise} \end{cases}$$

$$Var[x] = \int_{-a}^{a} x^2 \cdot \frac{1}{2a} dx = \left[\frac{1}{3} x^3 \frac{1}{2a}\right]_{-a}^{a} = \frac{1}{3} a^2$$

- Forward and backward directions

(1) $\forall i, Var[z^{(i)}] = Var[z^{(i+1)}]$ ← Forward direction

(2) $\forall i, Var[\frac{\partial L}{\partial s^{(i)}}] = Var[\frac{\partial L}{\partial s^{(i+1)}}]$ ← Backward direction

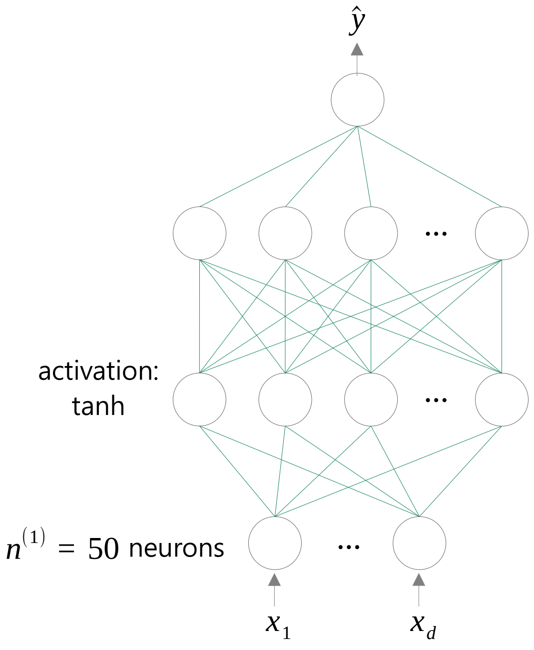
- We already know the variance that satisfies the above conditions, and the variance of the uniform distribution must also be equal to that variance.

$$\frac{1}{3} a^2 = \frac{2}{n^{(i)} + n^{(i+1)}} \longrightarrow \boxed{a = \sqrt{\frac{6}{n^{(i)} + n^{(i+1)}}}}$$

MX-AI

Xavier Glorot initializer: Example

- Given an ANN network like the one below, set appropriate initial distributions of $w^{(1)}$ and $w^{(2)}$.



$x = \{x_1, x_2, \ldots, x_{50}\} \sim N(0,1)$

- Normal distribution

$$w \sim N(0, \sigma^2) = N\left(0, \frac{2}{n^{(i)} + n^{(i+1)}}\right)$$

- Uniform distribution

$$w \sim U(-a, a) = U\left(-\sqrt{\frac{6}{n^{(i)} + n^{(i+1)}}}, \sqrt{\frac{6}{n^{(i)} + n^{(i+1)}}}\right)$$

$n^{(3)} = 100$ neurons

$$w^{(2)} \sim N\left(0, \frac{2}{80 + 100}\right) = N(0, 0.105^2)$$

$$w^{(2)} \sim U\left(-\sqrt{\frac{6}{80+100}}, \sqrt{\frac{6}{80+100}}\right) = U(-0.183, 0.183)$$

$n^{(2)} = 80$ neurons

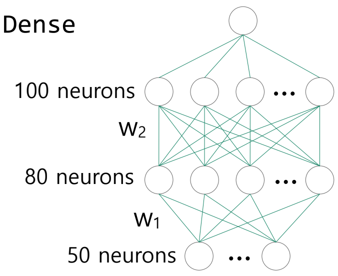
$$w^{(1)} \sim N\left(0, \frac{2}{50 + 80}\right) = N(0, 0.124^2)$$

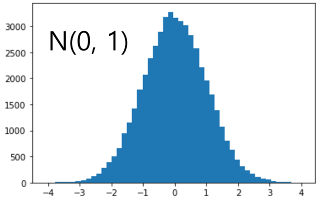
$$w^{(1)} \sim U\left(-\sqrt{\frac{6}{50+80}}, \sqrt{\frac{6}{50+80}}\right) = U(-0.215, 0.215)$$

MX-AI

■ Keras Xavier Glorot initializer

```python
# [MXDL-8-02] 2.xavier_glorot.py
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import initializers
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt

method = 'Normal'
if method == 'Normal':
    init_w1 = initializers.GlorotNormal()
    init_w2 = initializers.GlorotNormal()
else:
    init_w1 = initializers.GlorotUniform()
    init_w2 = initializers.GlorotUniform()

n_in = 50
n_s1 = 80
n_s2 = 100
x = np.random.normal(size=(1000, n_in))

# Create an ANN model
x_input = Input(batch_shape=(None, n_in))
s1 = Dense(n_s1, kernel_initializer=init_w1, activation='tanh',
           name='W1')(x_input)
s2 = Dense(n_s2, kernel_initializer=init_w2, activation='tanh',
           name='W2')(s1)
y_output = Dense(1, activation='sigmoid')(s2)
s1_model = Model(x_input, s1)
s2_model = Model(x_input, s2)
model = Model(x_input, y_output)

w1 = model.get_layer('W1').get_weights()[0].reshape(-1,)
w2 = model.get_layer('W2').get_weights()[0].reshape(-1,)

s1_out = s1_model.predict(x, verbose=0).reshape(-1,)
s2_out = s2_model.predict(x, verbose=0).reshape(-1,)
```

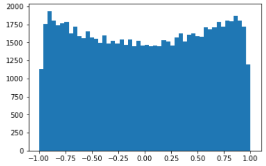




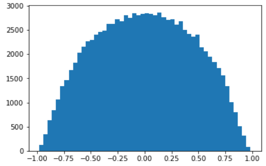

```python
plt.hist(x.reshape(-1,), bins=50); plt.show()
plt.hist(s1_out, bins=50); plt.show()
plt.hist(s2_out, bins=50); plt.show()

if method == 'Normal':
    print('[Keras   ] σ of w1 = {:.3f}'.format(w1.std()))
    print('[Formula] σ of w1 = {:.3f}'.\
          format(np.sqrt(2 / (n_in + n_s1))))
else:
    print('[Keras   ] a of w1 = {:.3f} ~ {:.3f}'.\
          format(w1.min(), w1.max()))
    print('[Formula] a of w1 = ±{:.3f}'.\
          format(np.sqrt(6 / (n_in + n_s1))))

if method == 'Normal':
    print('\n[Keras   ] σ of w2 = {:.3f}'.format(w2.std()))
    print('[Formula] σ of w2 = {:.3f}'.
          format(np.sqrt(2 / (n_s1 + n_s2))))
else:
    print('\n[Keras] a of w2 = {:.3f} ~ {:.3f}'.
          format(w2.min(), w2.max()))
    print('[Formula] a of w2 = ±{:.3f}'.
          format(np.sqrt(6 / (n_s1 + n_s2))))
```

```
method = ‘Normal’:
[Keras   ] σ of w1 = 0.125
[Formula] σ of w1 = 0.124

[Keras   ] σ of w2 = 0.105
[Formula] σ of w2 = 0.105

method = ‘Uniform’:
[Keras   ] a of w1 = -0.215 ~ 0.214
[Formula] a of w1 = ±0.215

[Keras] a of w2 = -0.182 ~ 0.183
[Formula] a of w2 = ±0.183
```

s1_out →



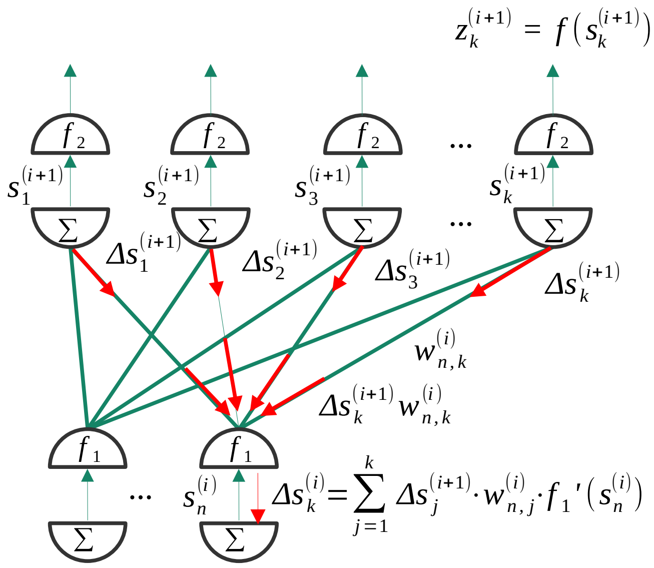
s2_out →



MX-AI

$$w^{(i)} \sim N(0, \frac{2}{n^{(i)}}), \text{ or } N(0, \frac{2}{n^{(i+1)}})$$

$$w^{(i)} \sim U(-\sqrt{\frac{6}{n^{(i)}}}, \sqrt{\frac{6}{n^{(i)}}}), \text{ or } U(-\sqrt{\frac{6}{n^{(i)}+1}}, \sqrt{\frac{6}{n^{(i+1)}}})$$



8. Weights Initialization

Part 3: Kaiming He initializer

This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

■ Kaiming He paper (2015)

Delving Deep into Rectifiers:
Surpassing Human-Level Performance on ImageNet Classification

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

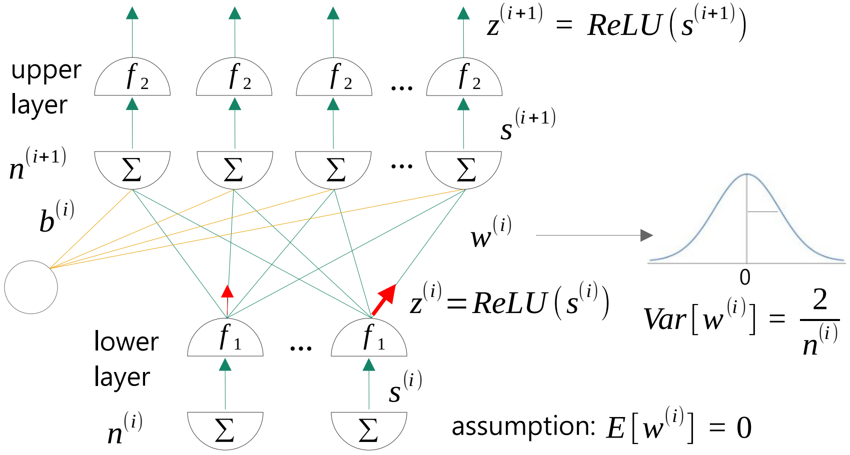
Abstract

Rectified activation units (rectifiers) are essential for state-of-the-art neural networks. In this work, we study rectifier neural networks for image classification from two aspects. **First**, we propose a Parametric Rectified Linear Unit (PReLU) that generalizes the traditional rectified unit. PReLU improves model fitting with nearly zero extra computational cost and little overfitting risk. **Second**, we derive a robust initialization method that particularly considers the rectifier nonlinearities. This method enables us to train extremely deep rectified models directly from scratch and to investigate deeper or wider network architectures. Based on our PReLU networks (PReLU-nets), we achieve 4.94% top-5 test error on the ImageNet 2012 classification dataset. This is a 26% relative improvement over the ILSVRC 2014 winner (GoogLeNet, 6.66% [29]). To our knowledge, our result is the first to surpass human-level performance (5.1%, [22]) on this visual recognition challenge.

MX-AI

Kaiming He initializer: Forward direction

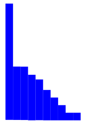
- During forward propagation, we would like the variance of the activations to remain constant across layers.
- Let's find the optimal variance of the initial weight distribution in the forward direction.



assumption: $E[w^{(i)}] = 0$

$(E[z^{(i)}] \neq 0 \rightarrow E[(z^{(i)})^2] \neq Var[z^{(i)}])$

distribution of z



0

Objective: $\forall i, Var[s^{(i+1)}] = Var[s^{(i)}]$ ← We want the variance of the activations to remain constant across layers.

$$Var[s^{(i+1)}] = n^{(i)} Var[z^{(i)} w^{(i)}]$$

$$= n^{(i)} [Var[z^{(i)}] Var[w^{(i)}] + Var[z^{(i)}](E[w^{(i)}])^2 + Var[w^{(i)}](E[z^{(i)}])^2]$$

$$= n^{(i)} [Var[z^{(i)}] Var[w^{(i)}] + Var[w^{(i)}](E[z^{(i)}])^2]$$

$$= n^{(i)} [Var[z^{(i)}] Var[w^{(i)}] + Var[w^{(i)}](E[(z^{(i)})^2] - Var[z^{(i)}])]$$

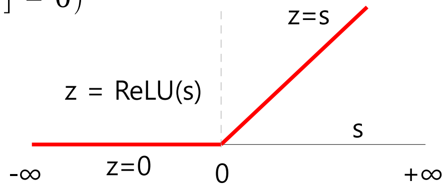
$$= n^{(i)} Var[w^{(i)}] E[(z^{(i)})^2]$$

$$E[(z^{(i)})^2] = \int_{-\infty}^{\infty} ReLU(s^{(i)})^2 p(s) ds = \int_{0}^{\infty} (s^{(i)})^2 p(s) ds = \frac{1}{2} \int_{-\infty}^{\infty} (s^{(i)})^2 p(s) ds$$

$$= \frac{1}{2} E[(s^{(i)})^2] = \frac{1}{2} Var[s^{(i)}] \quad (E[s^{(i)}] = 0)$$

$$Var[s^{(i+1)}] = \frac{1}{2} n^{(i)} Var[w^{(i)}] Var[s^{(i)}]$$

$$Var[w^{(i)}] = \frac{2}{n^{(i)}}$$

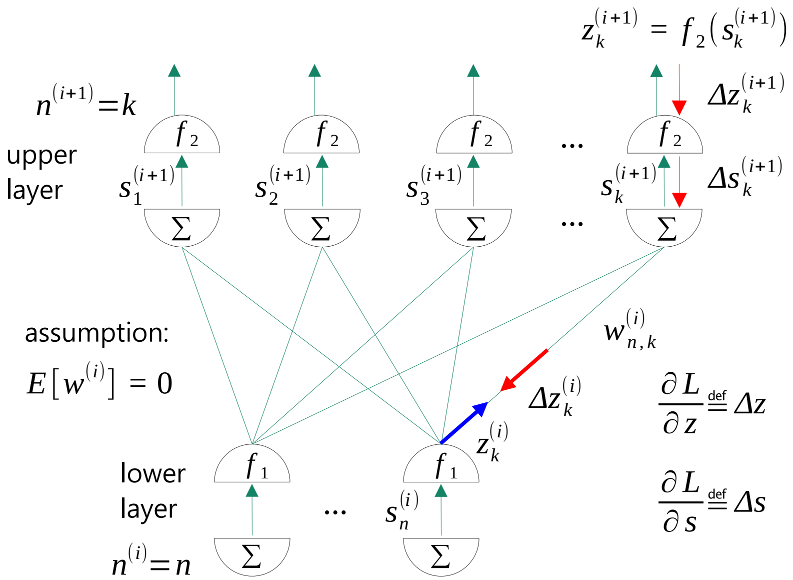


For example, if the number of neurons in the lower layer is 100, the initial weights of the upper layer are set by drawing them from a normal distribution with a mean of 0 and a variance of 0.02. N(0, 2/100)

MX-AI

Kaiming He initializer: Backward direction

- During backpropagation, we would like the variance of the gradients of loss with respect to the output z to remain constant across layers.



Objective: $\forall i, Var[\frac{\partial L}{\partial z^{(i+1)}}] = Var[\frac{\partial L}{\partial z^{(i)}}] \rightarrow Var[\Delta z^{(i+1)}] = Var[\Delta z^{(i)}]$

$$\Delta z^{(i)} = \frac{\partial L}{\partial s^{(i+1)}} \frac{\partial s^{(i+1)}}{\partial z^{(i)}} = \Delta s^{(i+1)} w^{(i)} \qquad (s^{(i+1)} = z^{(i)} \cdot w^{(i)} + b^{(i)})$$

$$E[\Delta z^{(i)}] = E[\Delta s^{(i+1)} w^{(i)}] = E[\Delta s^{(i+1)}] E[w^{(i)}] = 0$$ (Δs and w are independent, E[w] = 0)

$$\Delta s^{(i+1)} = \frac{\partial L}{\partial s^{(i+1)}} = \frac{\partial L}{\partial z^{(i+1)}} \frac{\partial z^{(i+1)}}{\partial s^{(i+1)}} = \Delta z^{(i+1)} f_2'(s^{(i+1)})$$

$\begin{cases} f'(s) = 1, \text{ if } s > 0 \\ f'(s) = 0, \text{ if } s < 0 \end{cases}$ with equal probabilities

$$E[\Delta s^{(i+1)}] = E[\Delta z^{(i+1)} f_2'(s^{(i+1)})] = \frac{1}{2} E[\Delta z^{(i+1)} \times 1] + \frac{1}{2} E[\Delta z^{(i+1)} \times 0] = 0$$

$$Var[\Delta s^{(i+1)}] = Var[\Delta z^{(i+1)} f_2'(s^{(i+1)})]$$

$$= E[(\Delta z^{(i+1)})^2] E[f_2'(s^{(i+1)})^2] - E[\cancelto{0}{\Delta z^{(i+1)}}]^2 E[f_2'(s^{(i+1)})]^2$$

$$= \frac{1}{2} Var[\Delta z^{(i+1)}] \qquad (E[f'(s^{(i)})^2] = \frac{0^2}{2} + \frac{1^2}{2} = \frac{1}{2})$$

$$Var[\Delta z^{(i)}] = n^{(i+1)} Var[\Delta s^{(i+1)} w^{(i)}] \longleftarrow \Delta z^{(i)} = \sum_{i=1}^{k} \Delta s_i^{(i+1)} w^{(i)}$$

$$= n^{(i+1)} Var[\Delta s^{(i+1)}] Var[w^{(i)}] = n^{(i+1)} \frac{1}{2} Var[\Delta z^{(i+1)}] Var[w^{(i)}]$$

$$\boxed{Var[w^{(i)}] = \frac{2}{n^{(i+1)}}}$$

For example, if the number of neurons in the lower layer is 80 and the number of neurons in the upper layer is 100, the initial weights between the lower and upper layer are set by drawing them from a normal distribution with a mean of 0, and a variance of either 2/100 or 2/80.

Forward direction: $Var[w^{(i)}] = \frac{2}{n^{(i)}}$

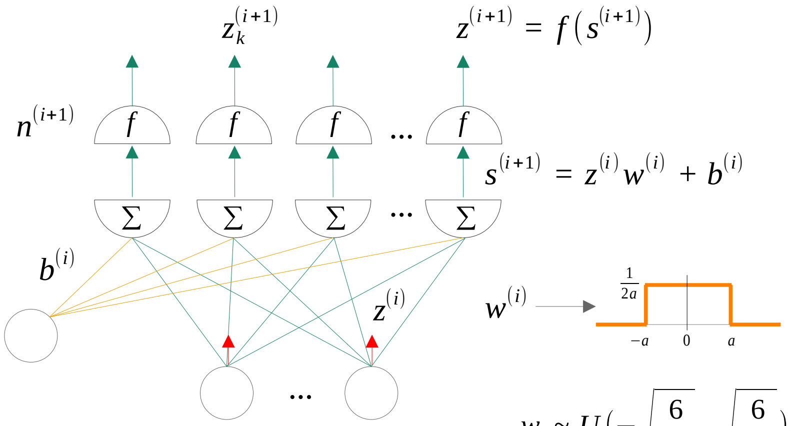
Backward direction: $Var[w^{(i)}] = \frac{2}{n^{(i+1)}}$

Either one can be used.

MX-AI

■ Kaiming He initializer: Uniform distribution

- Now that we know the optimal variance for the normal distribution, we can also find the range for the uniform distribution.



$w \sim U\left(-\sqrt{\frac{6}{n^{(i)}}}, \sqrt{\frac{6}{n^{(i)}}}\right)$

$n^{(i)} \leftarrow$ the number of neurons

- continuous uniform distribution

$$\begin{cases} g(x) = \dfrac{1}{2a}, & if \quad -a \le x \le a \\ g(x) = 0 & otherwise \end{cases}$$

$$Var[x] = \int_{-a}^{a} x^2 \cdot \frac{1}{2a} dx = \left[\frac{1}{3}x^3 \frac{1}{2a}\right]_{-a}^{a} = \frac{1}{3}a^2$$

- Forward and backward directions

(1) $\forall i, Var[z^{(i)}] = Var[z^{(i+1)}]$ ← Forward direction

(2) $\forall i, Var\left[\dfrac{\partial L}{\partial z^{(i)}}\right] = Var\left[\dfrac{\partial L}{\partial z^{(i+1)}}\right]$ ← Backward direction

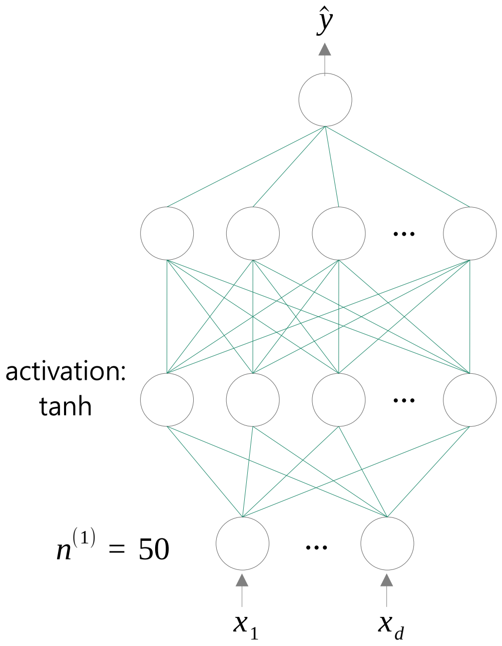
- We already know the variance that satisfies the above conditions, and the variance of the uniform distribution must also be equal to that variance.

$$\frac{1}{3}a^2 = \frac{2}{n^{(i)}} \longrightarrow \boxed{a = \sqrt{\frac{6}{n^{(i)}}}}$$

MX-AI

Kaiming He initializer: Example

- Given an ANN network like the one below, set appropriate initial distributions of $w^{(1)}$ and $w^{(2)}$.



$x = \{x_1, x_2, \ldots, x_{50}\} \sim N(0,1)$

- Normal distribution

$$w \sim N(0, \sigma^2) = N(0, \frac{2}{n^{(i)}})$$

- Uniform distribution

$$w \sim U(-a, a) = U(-\sqrt{\frac{6}{n^{(i)}}}, \sqrt{\frac{6}{n^{(i)}}})$$

$n^{(3)} = 100$

$$w^{(2)} \sim N(0, \frac{2}{80}) = N(0, 0.158^2)$$

$$w^{(2)} \sim U(-\sqrt{\frac{6}{80}}, \sqrt{\frac{6}{80}}) = U(-0.274, 0.274)$$

$n^{(2)} = 80$

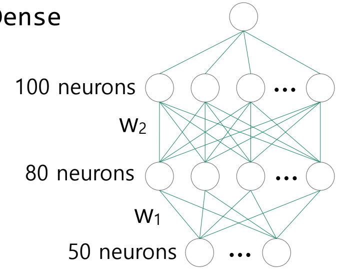
$$w^{(1)} \sim N(0, \frac{2}{50}) = N(0, 0.2^2)$$

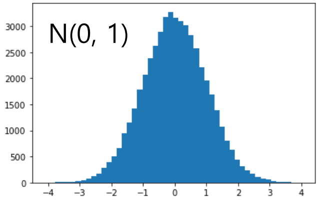
$$w^{(1)} \sim U(-\sqrt{\frac{6}{50}}, \sqrt{\frac{6}{50}}) = U(-0.346, 0.346)$$

MX-AI

■ Keras Kaiming He initializer

```
# [MXDL-8-03] 3.kaiming_he.py
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import initializers
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt

method = 'Normal'
if method == 'Normal':
    init_w1 = initializers.HeNormal()
    init_w2 = initializers.HeNormal()
else:
    init_w1 = initializers.HeUniform()
    init_w2 = initializers.HeUniform()

n_in = 50
n_s1 = 80
n_s2 = 100
x = np.random.normal(size=(1000, n_in))

# Create an ANN model
x_input = Input(batch_shape=(None, n_in))
s1 = Dense(n_s1, kernel_initializer=init_w1, activation='relu',
           name='W1')(x_input)
s2 = Dense(n_s2, kernel_initializer=init_w2, activation='relu',
           name='W2')(s1)
y_output = Dense(1, activation='sigmoid')(s2)
s1_model = Model(x_input, s1)
s2_model = Model(x_input, s2)
model = Model(x_input, y_output)

w1 = model.get_layer('W1').get_weights()[0].reshape(-1,)
w2 = model.get_layer('W2').get_weights()[0].reshape(-1,)

s1_out = s1_model.predict(x, verbose=0).reshape(-1,)
s2_out = s2_model.predict(x, verbose=0).reshape(-1,)
```

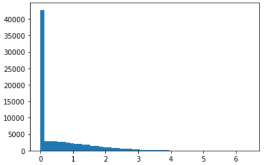





```
plt.hist(x.reshape(-1,), bins=50); plt.show()
plt.hist(s1_out, bins=50); plt.show()
plt.hist(s2_out, bins=50); plt.show()

if method == 'Normal':
    print('[Keras  ] σ of w1 = {:.3f}'.format(w1.std()))
    print('[Formula] σ of w1 = {:.3f}'.\
           format(np.sqrt(2 / n_in)))
else:
    print('[Keras  ] a of w1 = {:.3f} ~ {:.3f}'.\
           format(w1.min(), w1.max()))
    print('[Formula] a of w1 = ±{:.3f}'.format(np.sqrt(6 / n_in)))

if method == 'Normal':
    print('\n[Keras  ] σ of w2 = {:.3f}'.format(w2.std()))
    print('[Formula] σ of w2 = {:.3f}'.format(np.sqrt(2 / n_s1)))
else:
    print('\n[Keras] a of w2 = {:.3f} ~ {:.3f}'.\
           format(w2.min(), w2.max()))
    print('[Formula] a of w2 = ±{:.3f}'.format(np.sqrt(6 / n_s1)))
```

```
method = 'Normal':
[Keras  ] σ of w1 = 0.198
[Formula] σ of w1 = 0.200
[Keras  ] σ of w2 = 0.158
[Formula] σ of w2 = 0.158
```

s1_out →



```
method = 'Uniform':
[Keras  ] a of w1 = -0.346 ~ 0.346
[Formula] a of w1 = ±0.346
[Keras] a of w2 = -0.274 ~ 0.274
[Formula] a of w2 = ±0.274
```

s2_out →