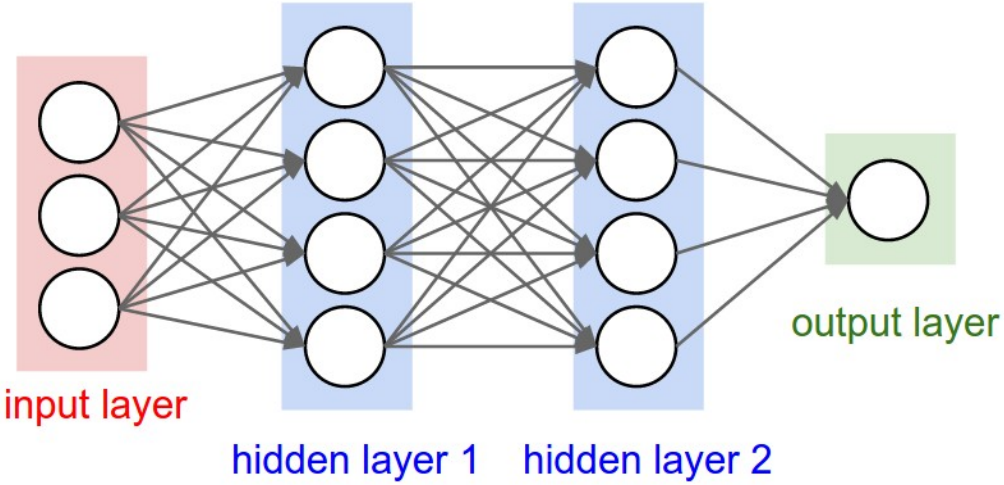
Weight 초기화

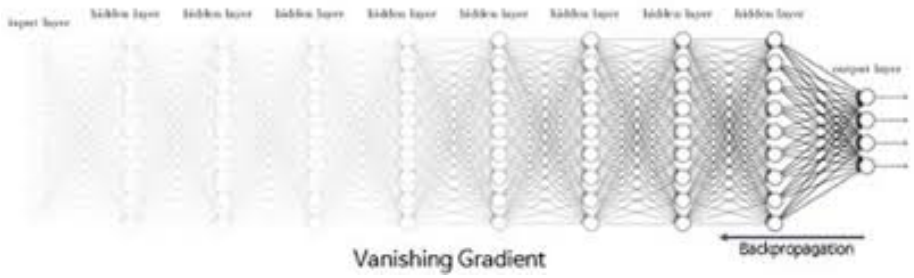
명지대학교
MYONGJI UNIVERSITY

Fully Connected Neural Network



https://cs231n.github.io/assets/nn1/neural_net2.jpeg

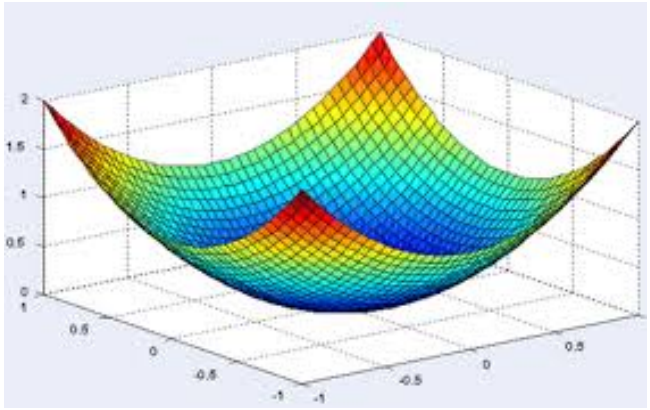
Vanishing gradient

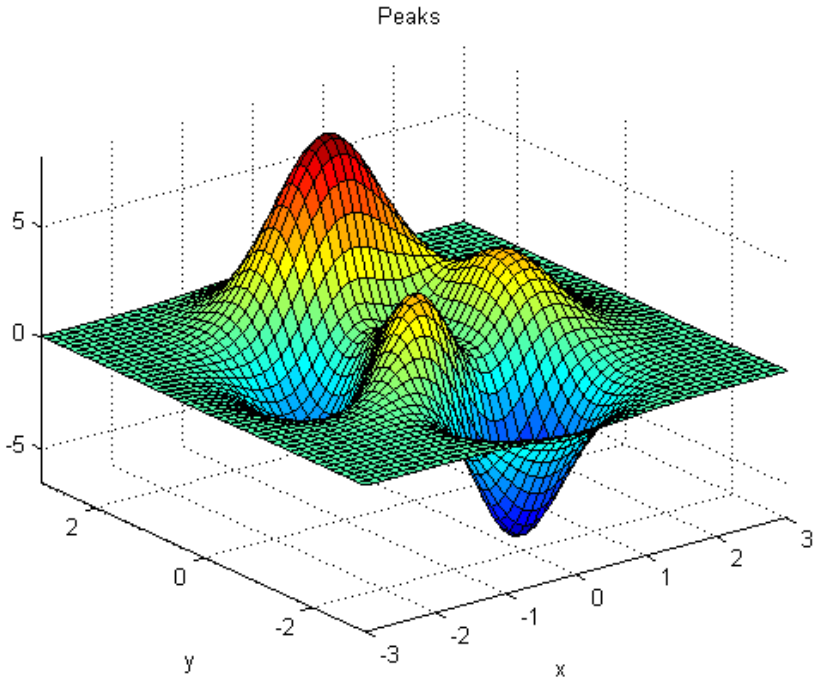


기타 activation function

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

Loss





Data Load

```
def load_mnist() :
    digits = load_digits()
    x_data = digits.data
    y_data = digits.target
    x_trainf, x_test, y_trainf, y_test = train_test_split(x_data, y_data, test_size=0.3)
    X_valid, X_train = x_trainf[:50] , x_trainf[50:]
    y_valid, y_train = y_trainf[:50], y_trainf[50:]


    return X_valid,X_train,y_valid,y_train,x_test,y_test
```

가중치 초기화 - 기존 랜덤

```
def makemodel(X_train, y_train, X_valid, y_valid):
    model = keras.models.Sequential()
    model.add(keras.layers.Flatten(input_shape=[28, 28]))
    model.add(keras.layers.Dense(300, activation="relu"))
    model.add(keras.layers.Dense(100, activation="relu"))
    model.add(keras.layers.Dense(10, activation="softmax"))

    model.summary()
```

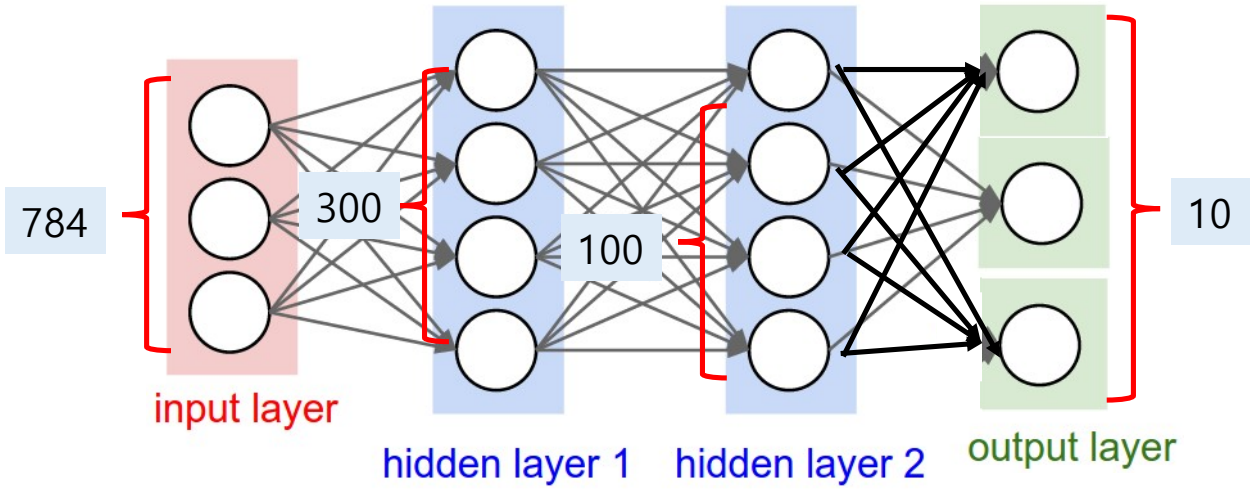
가중치 초기화 - 서로 다른 초기화 방법

```
def makemodel(X_train, y_train, X_valid, y_valid, weight_init,):
    model = keras.models.Sequential()
    model.add(keras.layers.Flatten(input_shape=[28, 28]))
    model.add(keras.layers.Dense(300, weight_init, activation="relu"))
    model.add(keras.layers.Dense(100, weight_init, activation="relu"))
    model.add(keras.layers.Dense(10, weight_init, activation="softmax"))

    model.summary()
```

Model: sequential

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 784)	0
dense_6 (Dense)	(None, 300)	235500
dense_7 (Dense)	(None, 100)	30100
dense_8 (Dense)	(None, 10)	1010



가중치 초기화

기본 가중치 초기화 방법

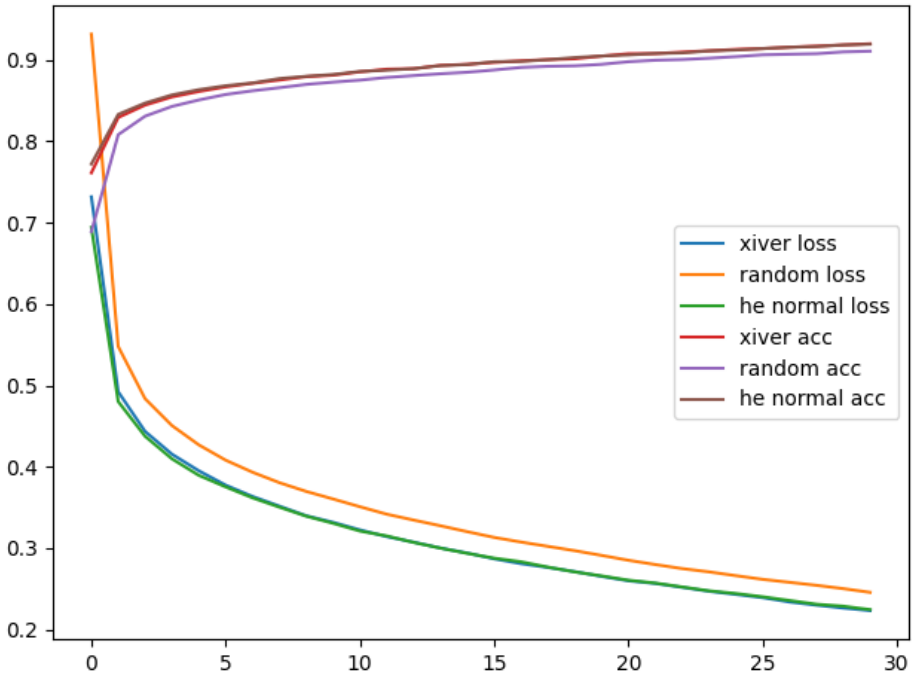
- 케라스 레이어의 가중치 초기화 방식은 일정 구간 내에서 랜덤하게 찍는 random_uniform
- 오차 역전파(back propagation) 과정에서 미분한 gradient가 지나치게 커지거나(exploding gradient) 소실되는(vanishing gradient) 문제에 빠질 위험성

어떻게 가중치를 초기화할 것인가

- LeCun 초기화(lecun_uniform, lecun_normal): 98년도에 얀 르쿤이 제기한 방법으로 최근에는 Xavier나 He 초기화 방식에 비해 덜 사용되는 편이다.
- Xavier 초기화(glorot_uniform, glorot_normal): 케라스에서는 glorot이라는 이름으로 되어있는데, 일반적으로는 Xavier Initialization이라고 알려져 있다. 사실 초기화 방식이 제안된 논문의 1저자 이름이 Xavier Glorot이다(출처). 2저자는 유명한 Yoshua Bengio.
- He 초기화(he_uniform, he_normal): ResNet으로도 유명한 마이크로소프트(현재는 Facebook)의 Kaiming He가 2015년에 제안한 가장 최신의 초기화 방식이다. 수식을 보면 Xavier Initialization을 조금 개선한 것인데, 경험적으로 더 좋은 결과를 내었다고 한다.

가중치 초기화 - 서로 다른 초기화 방법

```
model_xavier,hist_xavier= makemodel(X_train, y_train, X_valid, y_valid,'glorot_uniform')
model_RandomNormal,hist_RandomNormal= makemodel(X_train, y_train, X_valid, y_valid,'RandomNormal')
model_he,hist_he= makemodel(X_train, y_train, X_valid, y_valid,'he_normal')
```



He vs Xavier 초기화

Xavier

- ReLu 등장 후에 Glorot이 2010년에 제안한 방법으로 vanshing gradient 문제를 해결하기 위해 만들어짐
- Input 과 output neuron의 수에 기반해서 초기화의 스케일을 정함
- Sigmoid 나 tanh 사용할 경우 Xaiver

He

- Glorot과 유사하지만 Neuron의 out size를 고려하진 않음
- Relu를 사용할 경우 He 초기화

둘다 초기 파라미터의 분포에 대한 좋은 variance를 찾는 방법.

tf.keras.initializers

https://www.tensorflow.org/api_docs/python/tf/keras/initializers

- `class GlorotNormal`: The Glorot normal initializer, also called Xavier normal initializer.
- `class GlorotUniform`: The Glorot uniform initializer, also called Xavier uniform initializer.
- `class HeNormal`: He normal initializer.
- `class HeUniform`: He uniform variance scaling initializer.
- `class Identity`: Initializer that generates the identity matrix.
- `class Initializer`: Initializer base class: all Keras initializers inherit from this class.
- `class LecunNormal`: Lecun normal initializer.
- `class LecunUniform`: Lecun uniform initializer.
- `class Ones`: Initializer that generates tensors initialized to 1.
- `class Orthogonal`: Initializer that generates an orthogonal matrix.
- `class RandomNormal`: Initializer that generates tensors with a normal distribution.
- `class RandomUniform`: Initializer that generates tensors with a uniform distribution.
- `class TruncatedNormal`: Initializer that generates a truncated normal distribution.
- `class VarianceScaling`: Initializer capable of adapting its scale to the shape of weights tensors.
- `class Zeros`: Initializer that generates tensors initialized to 0.
- `class constant`: Initializer that generates tensors with constant values.
- `class glorot_normal`: The Glorot normal initializer, also called Xavier normal initializer.
- `class glorot_uniform`: The Glorot uniform initializer, also called Xavier uniform initializer.
- `class he_normal`: He normal initializer.
- `class he_uniform`: He uniform variance scaling initializer.
- `class identity`: Initializer that generates the identity matrix.
- `class lecun_normal`: Lecun normal initializer.
- `class lecun_uniform`: Lecun uniform initializer.
- `class ones`: Initializer that generates tensors initialized to 1.
- `class orthogonal`: Initializer that generates an orthogonal matrix.
- `class random_normal`: Initializer that generates tensors with a normal distribution.
- `class random_uniform`: Initializer that generates tensors with a uniform distribution.
- `class truncated_normal`: Initializer that generates a truncated normal distribution.
- `class variance_scaling`: Initializer capable of adapting its scale to the shape of weights tensors.

tf.keras.optimizers

Model optimization

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

Classes

`class Adadelta` : Optimizer that implements the Adadelta algorithm.

`class Adagrad` : Optimizer that implements the Adagrad algorithm.

`class Adam` : Optimizer that implements the Adam algorithm.

`class Adamax` : Optimizer that implements the Adamax algorithm.

`class Ftrl` : Optimizer that implements the FTRL algorithm.

`class Nadam` : Optimizer that implements the NAdam algorithm.

`class Optimizer` : Base class for Keras optimizers.

`class RMSprop` : Optimizer that implements the RMSprop algorithm.

`class SGD` : Gradient descent (with momentum) optimizer.

tf.keras.losses

`class BinaryCrossentropy` : Computes the cross-entropy loss between true labels and predicted labels.

`class CategoricalCrossentropy` : Computes the crossentropy loss between the labels and predictions.

`class CategoricalHinge` : Computes the categorical hinge loss between `y_true` and `y_pred`.

`class CosineSimilarity` : Computes the cosine similarity between labels and predictions.

`class Hinge` : Computes the hinge loss between `y_true` and `y_pred`.

`class Huber` : Computes the Huber loss between `y_true` and `y_pred`.

`class KLDivergence` : Computes Kullback-Leibler divergence loss between `y_true` and `y_pred`.

`class LogCosh` : Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Loss` : Loss base class.

`class MeanAbsoluteError` : Computes the mean of absolute difference between labels and predictions.

`class MeanAbsolutePercentageError` : Computes the mean absolute percentage error between `y_true` and `y_pred`

`class MeanSquaredError` : Computes the mean of squares of errors between labels and predictions.

`class MeanSquaredLogarithmicError` : Computes the mean squared logarithmic error between `y_true` and `y_pred`

`class Poisson` : Computes the Poisson loss between `y_true` and `y_pred`.

`class Reduction` : Types of loss reduction.

`class SparseCategoricalCrossentropy` : Computes the crossentropy loss between the labels and predictions.

`class SquaredHinge` : Computes the squared hinge loss between `y_true` and `y_pred`.

https://www.tensorflow.org/api_docs/python/tf/keras/losses