UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Hyperparam. Tuning Batch Normalization Multiclass Class.

Lesson #10

Hyperparameter Tuning

Hyperparameters

α

β

$\#hidden\ units$

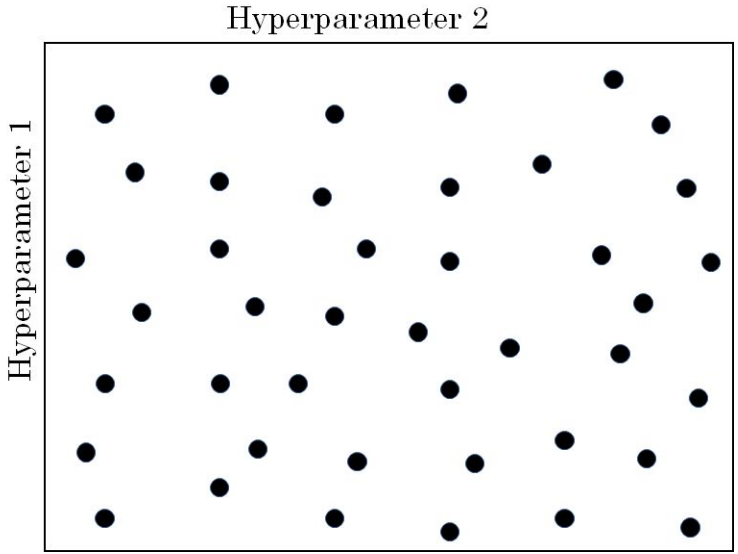
$mini-batch\ size$

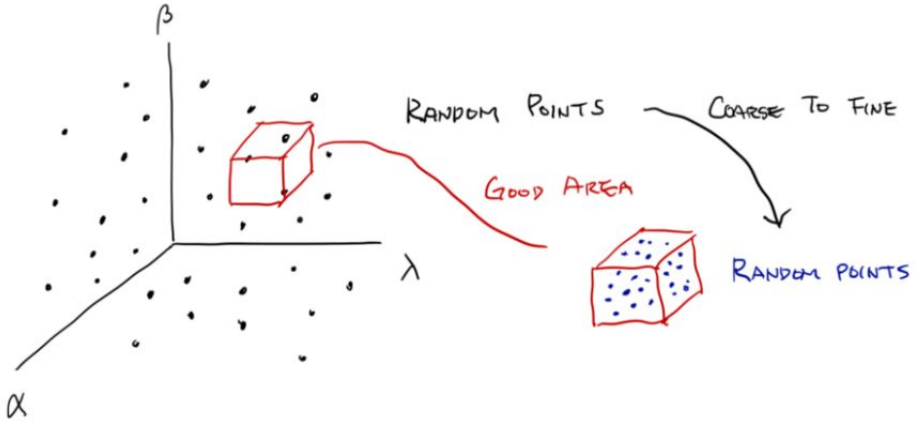
$\beta_1,\ \beta_2,\ \epsilon$

$\#layers$

$learning\ rate\ decay$

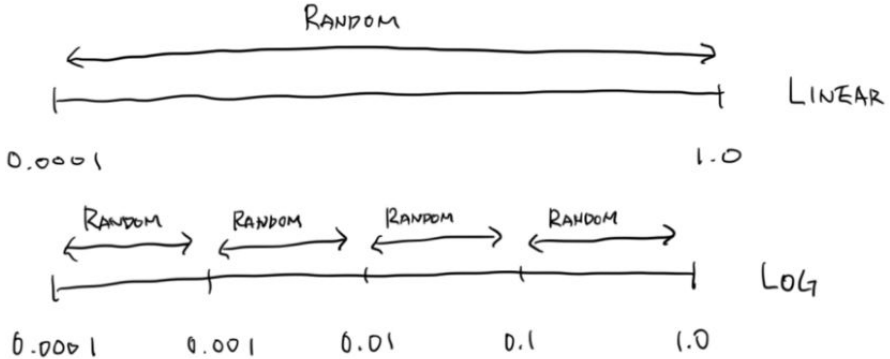
Try random values: don't use a grid





Feurer et al. Auto-Sklearn 2.0: The Next Generation, Jul. 2020

Picking Hyperparameters at Random



$n^{[l]}$ → np.random.randint(50,101)
L → np.random.randint(2,5)

$$r = \log_{10} 1 \rightarrow 10^r = 10^0 \rightarrow r = 0$$

$$r = \log_{10} 0.0001 \rightarrow 10^r = 10^{-4} \rightarrow r = -4$$

r = -4 np.random.rand( )
α = np.power(10,r)

Hyperparameters for Exponentially Weighted Averages

					$r \in [-3,-1]$
ß	0.9	0.9005	0.999	0.9995	$1 - 10^r$
1-ß	0.1	0.0995	0.001	0.0005	10^r
Average 1/(1-ß)	10	10.05	1000	2000	

ß

random random

0.9 0.99 0.999

1-ß

random random

0.1 0.01 0.001

Keras Tuner

```python
from tensorflow import keras
from tensorflow.keras import layers
from kerastuner.tuners import RandomSearch


def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Dense(units=hp.Int('units',
                                        min_value=32,
                                        max_value=512,
                                        step=32),
                           activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(
        optimizer=keras.optimizers.Adam(
            hp.Choice('learning_rate',
                      values=[1e-2, 1e-3, 1e-4],
                      sampling='LOG')),
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return model
```

```python
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=5,
    executions_per_trial=3,
    directory='my_dir',
    project_name='helloworld')
```

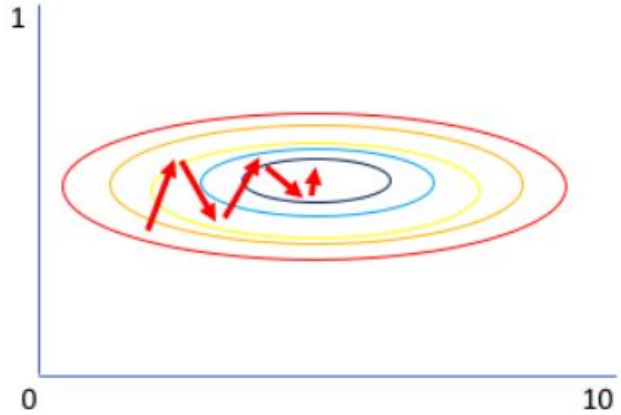
```python
tuner.search(train_x,
             train_y,
             epochs = 500,
             verbose=0,
             batch_size=32,
             validation_data = (test_x, test_y))
```

https://keras-team.github.io/keras-tuner/

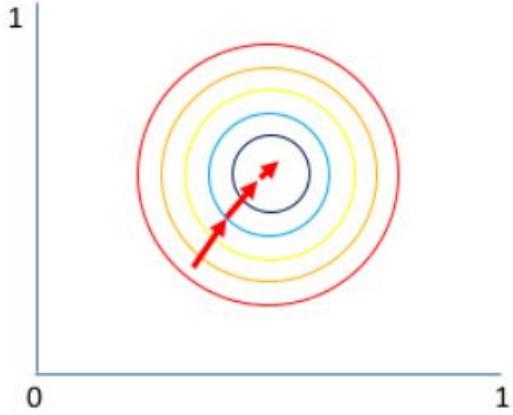
> Lesson 03 Task01
-Hyperparameter Tuning.ipynb

Batch Normalization

Normalizing inputs to speed up learning

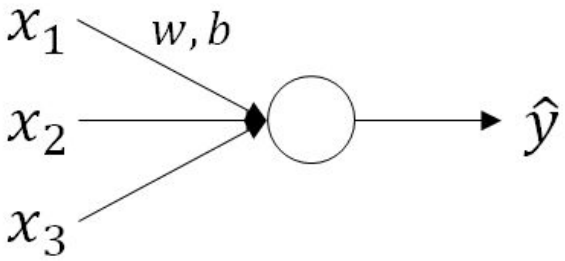


Gradient of larger parameter dominates the update

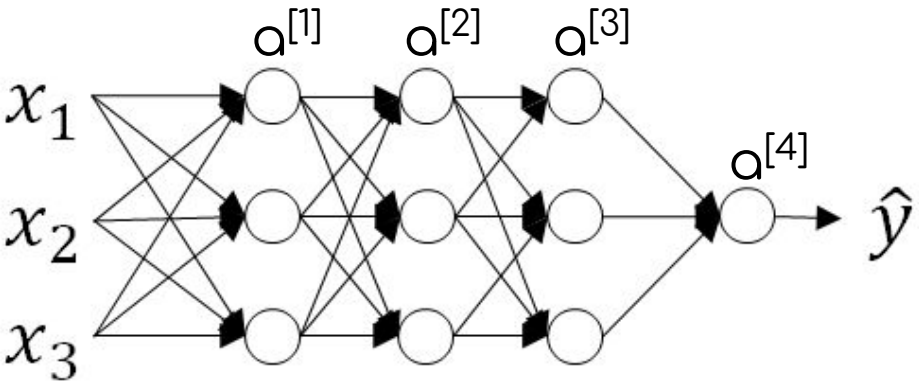


Both parameters can be updated in equal proportions

Normalizing inputs to speed up learning



$$X = \frac{X - \mu}{\sigma}$$



In practice, we actually **normalize $z^{[l]}$** which has the same effect as **normalizing $a^{[l]}$**.
Batch Norm (BN)

Implementing BN in a single layer

$$All\ \ z^{[l](i)} = \{z^{[l](1)}, z^{[l](2)}, \ldots, z^{[l](m)}\}$$

$$\mu = \frac{1}{m}\sum_i z^{[l](i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{[l](i)} - \mu)^2$$

$$z^{[l](i)}_{norm} = \frac{z^{[l](i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

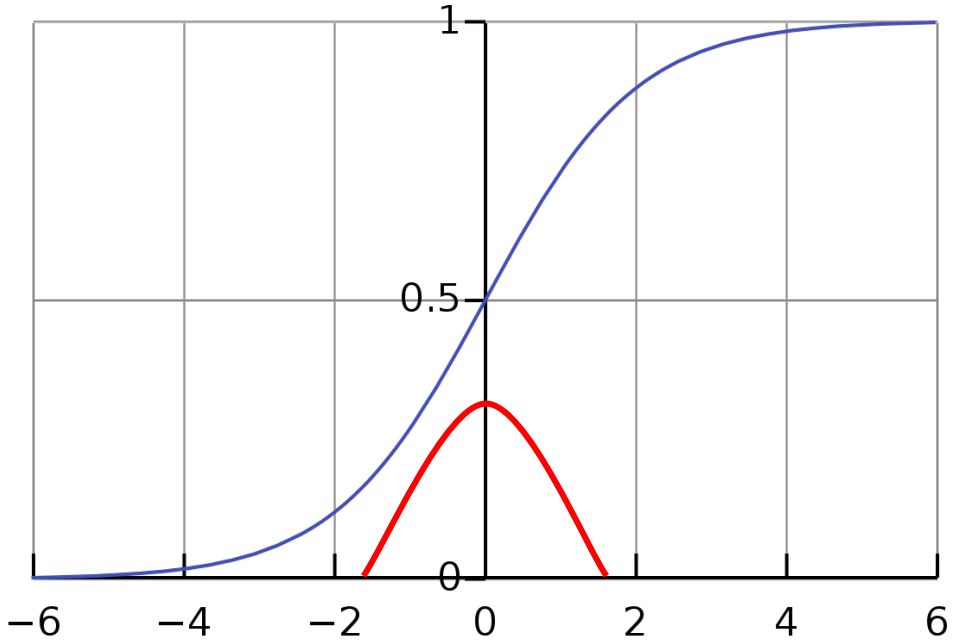
$$\tilde{z}^{[l](i)} = \gamma\, z^{[l](i)}_{norm} + \beta$$

$$if\ \ \gamma = \sqrt{\sigma^2 + \epsilon}\ \ and\ \ \beta = -\mu$$

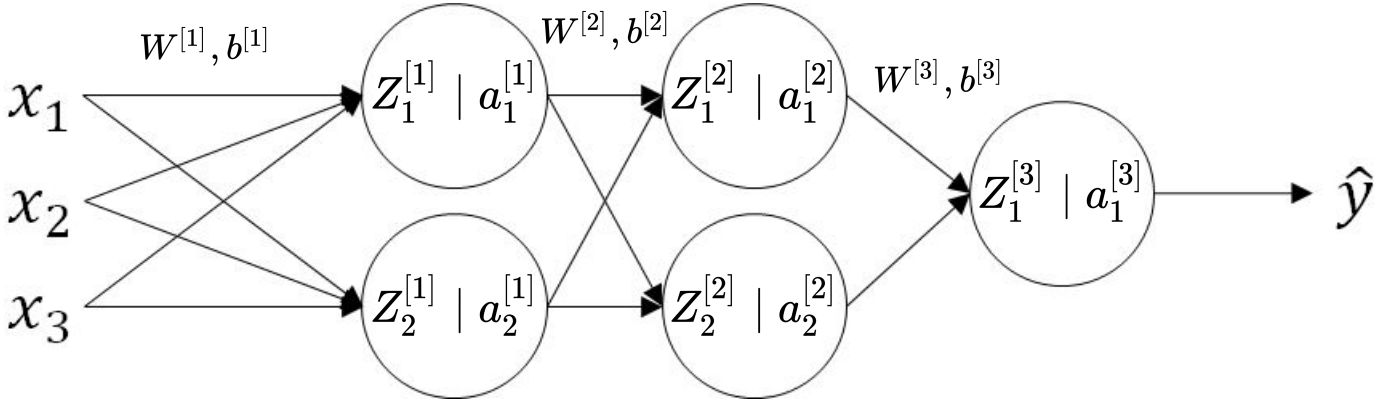
$$z^{[l](i)}_{norm} = \tilde{z}^{[l](i)}$$

- γ and β are learnable parameters (gradient descent)
- γ and β are used to avoid all layers have activations with mean zero and standard deviation equal to one.

Implementing BN in a single layer <<Intuition>>



Adding BN to a Network



$$X \xrightarrow{W^{[1]},b^{[1]}} Z^{[1]} \xrightarrow[\text{BN}]{\gamma^{[1]},\beta^{[1]}} \tilde{Z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{Z}^{[1]}) \xrightarrow{W^{[2]},b^{[2]}} Z^{[2]}$$

$$\xrightarrow[\text{BN}]{\gamma^{[2]},\beta^{[2]}} \tilde{Z}^{[2]} \rightarrow a^{[2]} = g^{[2]}(\tilde{Z}^{[2]}) \xrightarrow{W^{[3]},b^{[3]}} \dots$$

Parameters (learnable from Gradient Descent)

$$\begin{cases} W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \\ \gamma^{[1]}, \beta^{[1]}, \gamma^{[2]}, \beta^{[2]}, \gamma^{[3]}, \beta^{[3]} \end{cases}$$

Working with Mini-Batches

$$X^{\{1\}} \xrightarrow{W^{[1]},b^{[1]}} Z^{[1]} \xrightarrow[\text{BN}]{\gamma^{[1]},\beta^{[1]}} \tilde{Z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{Z}^{[1]}) \xrightarrow{W^{[2]},b^{[2]}} Z^{[2]}$$

$$\xrightarrow[\text{BN}]{\gamma^{[2]},\beta^{[2]}} \tilde{Z}^{[2]} \rightarrow a^{[2]} = g^{[2]}(\tilde{Z}^{[2]}) \xrightarrow{W^{[3]},b^{[3]}} \dots$$

$$\vdots$$

$$X^{\{t\}} \dots$$

BN calculates the normalization using data from individual batch t

Parameters (learnable from Gradient Descent)

$$\begin{cases} W^{[1]}, \cancel{b^{[1]}}, W^{[2]}, \cancel{b^{[2]}}, W^{[3]}, \cancel{b^{[3]}} \\ \gamma^{[1]}, \beta^{[1]}, \gamma^{[2]}, \beta^{[2]}, \gamma^{[3]}, \beta^{[3]} \end{cases}$$

$$\tilde{z}^{[l]\{i\}} = \gamma z_{norm}^{[l]\{i\}} + \beta$$

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Abstract

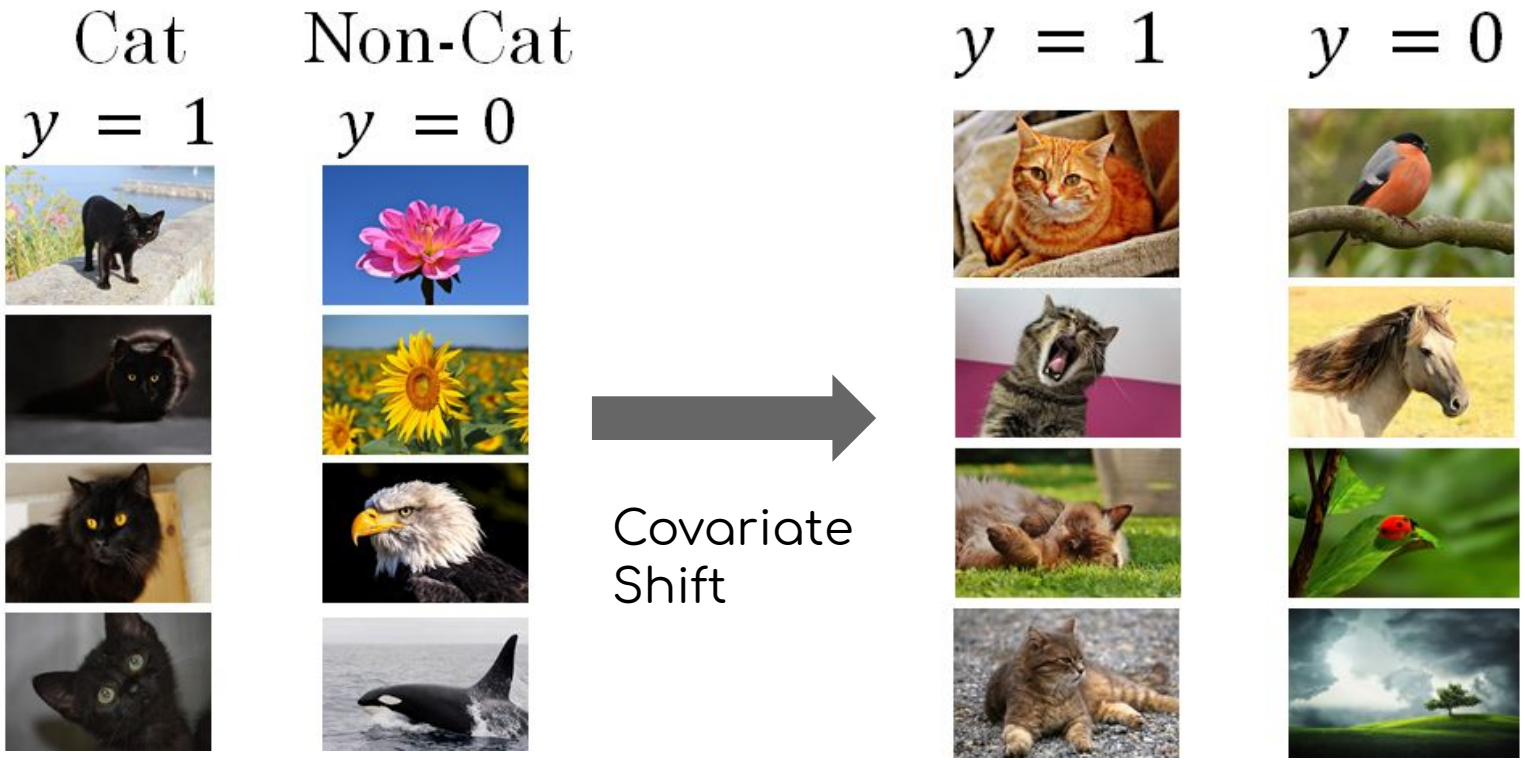
Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously 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,

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs

Learning on Shifting Input Distribution



Batch Normalization as Regularization

- Batch norm has slight regularization effect because it kind of **can cancel out large W**, adding noise.
  - Each mini-batch is scaled by the mean/standard deviation computed on just that mini-batch. This adds some noise to the values $Z^{[l]}$ within that mini-batch. So similar to dropout, it adds some noise to each hidden layer's activations.
- Its regularization effect gets weaker as batch size gets larger because there is less noise as batch size gets larger

```
# Batch Normalization After Activation Function
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(25, activation=tf.nn.relu,kernel_initializer="he_uniform"))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(12, activation=tf.nn.relu,kernel_initializer="he_uniform"))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(6, activation=tf.nn.softmax,kernel_initializer="he_uniform"))
```

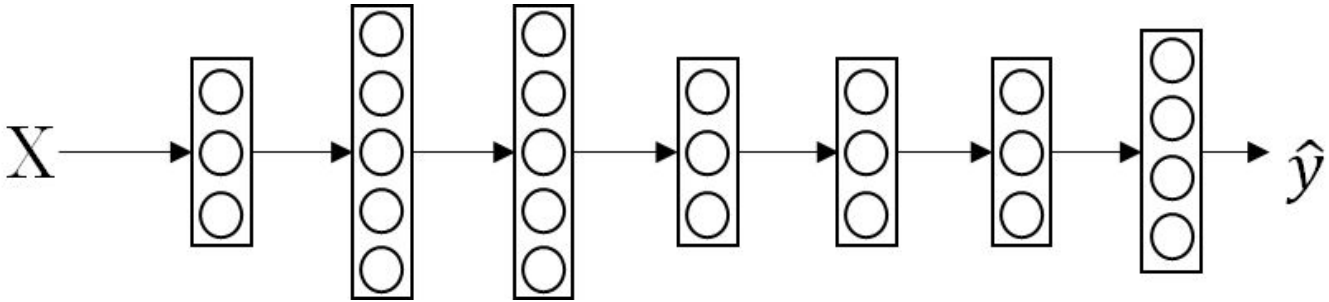
```
# Batch Normalization Before Activation Function
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(25, kernel_initializer="he_uniform"))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Activation(tf.nn.relu))

model.add(tf.keras.layers.Dense(12,kernel_initializer="he_uniform"))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Activation(tf.nn.relu))
```

Other practice problems beyond binary classification

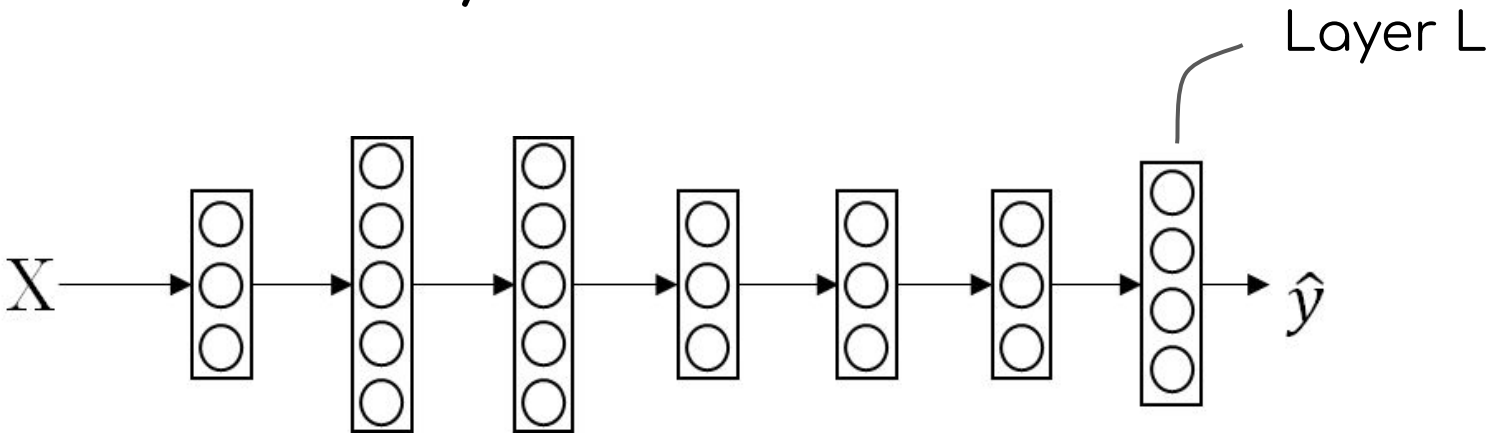
Softmax Regression

3 1 2 0 3 2 0 1



$P(cat \mid x) = ?$
$P(dog \mid x) = ?$
$P(bc \mid x) = ?$
$P(koala \mid x) = ?$

Softmax Layer

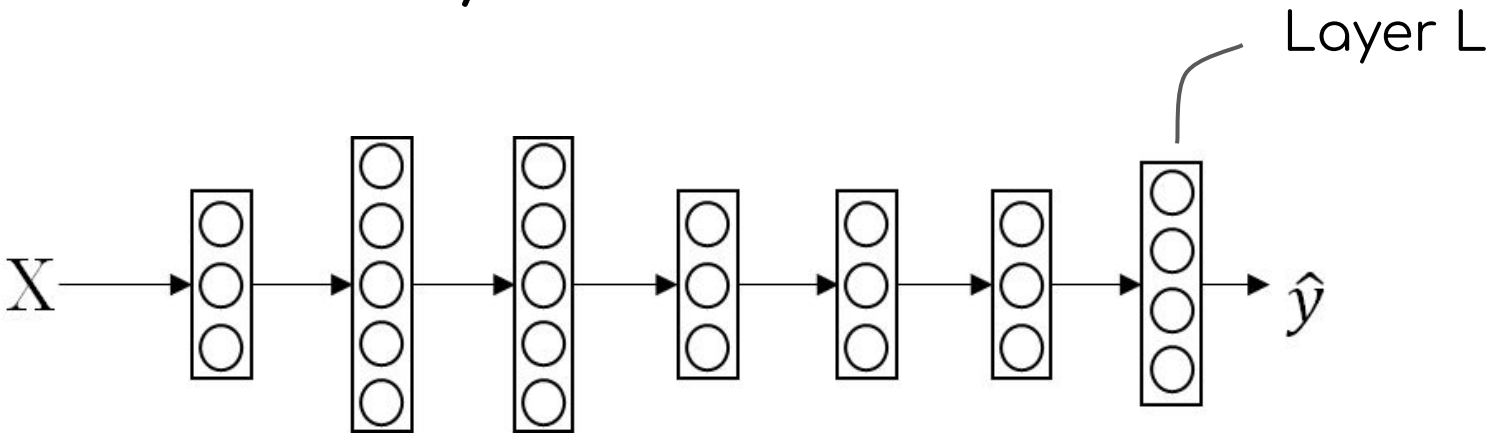


$$Z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]}$$

$$a_i^{[L]} = \frac{e^{Z_i^{[L]}}}{\sum_{j=1}^{n^{[L]}} e^{Z_j^{[L]}}}$$

$$Z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \rightarrow a^{[L]} = \begin{bmatrix} \frac{e^5}{e^5+e^2+e^{-1}+e^3} \\ \frac{e^2}{e^5+e^2+e^{-1}+e^3} \\ \frac{e^{-1}}{e^5+e^2+e^{-1}+e^3} \\ \frac{e^3}{e^5+e^2+e^{-1}+e^3} \end{bmatrix} = \begin{bmatrix} 0.8420 \\ 0.0419 \\ 0.0020 \\ 0.1137 \end{bmatrix} = \hat{y}$$

Softmax Layer - Loss Function



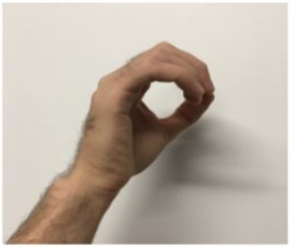
$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{n^{[L]}} y_j \log \hat{y}_j$$

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathcal{L}(\hat{y}, y) = -y_1 \log \hat{y}_1$$

$$\mathcal{L}(\hat{y}, y) = -y_1 \log a_1^{[L]}$$

> Lesson 10 Task 02 - Batch Normalization, Multiclass Classification.ipynb



y = 0

y = 1

y = 2

y = 3

y = 4

y = 5