Keras

Deep Learning Framework

What is Keras ?

- Deep neural network library in Python
  - High-level neural networks API
  - Modular – Building model is just stacking layers
  - Runs on top of either TensorFlow
- Why use Keras ?
  - Useful for fast prototyping, ignoring the details of implementing backprop or
  - writing optimization procedure
  - Supports Convolution, Recurrent layer and combination of both.
  - Runs seamlessly on CPU and GPU
  - Almost any architecture can be designed using this framework
  - Open-Source code – Large community support

Documentation: http://keras.io/

Three API styles (Keras Models)

- The Sequential Model
  - Dead simple
  - Only for single-input, single-output, sequential layer stacks
  - Allows you to create models layer-layer by stacking them
- The functional API
  - Flexible model architecture (each layer can be connected in a pairwise fashion).
  - Multi-input, multi-output, arbitrary static graph topologies
  - Can create complex network such as Residual Network.
- Model subclassing
  - More Flexible than Functional or Sequential.
  - Larger potential error surface

Keras Sequential model

- The Sequential model is a linear stack of layers
- You can create a Sequential model by passing a list of layer instances to the constructor:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

You can also simply add layers via the `.add()` method:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

The functional API

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Model subclassing

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)
```

Sequential model – steps

- **Specifying the input shape**
  - Input_shape
  - Input_dim
- **Compiling**
  - Optimizer
  - Loss function
  - Metrics
- **Training**
  - Optimizer
  - Loss function
  - Metrics

```
model = Sequential()
model.add(Dense(32, input_shape=(784,)))
```

```
model = Sequential()
model.add(Dense(32, input_dim=784))
```

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
# For a single-input model with 2 classes (binary classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

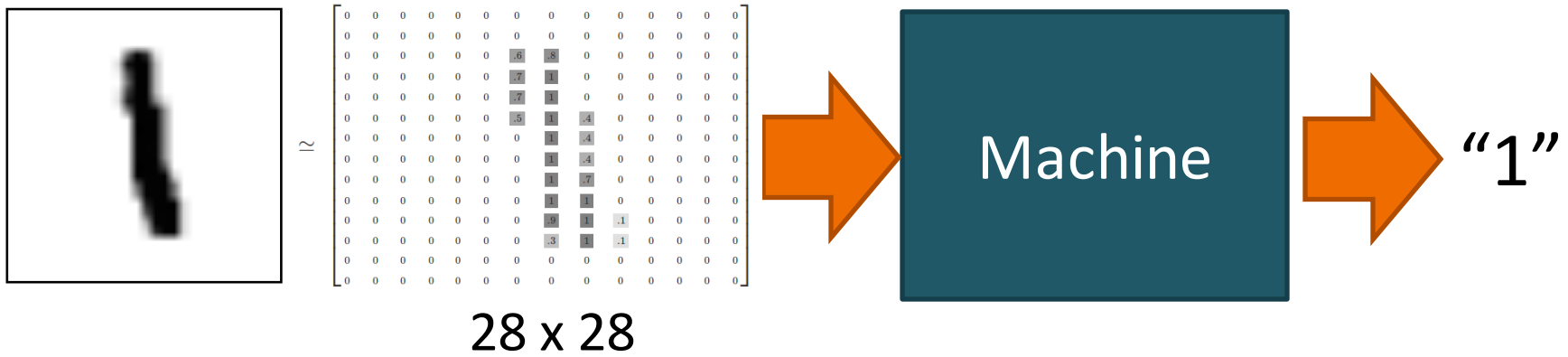
Model parameters

- Neural Network Model Parameters
  - No. of layers
  - No. of neurons in each layer
  - Connection between neurons from different layers
  - Activation Function
  - Optimization Method
  - Error Function

- Training (Time) Parameter
  - No of epochs
  - Batch size

- Evaluation
  - Measurement
  - Training/Validation/Test

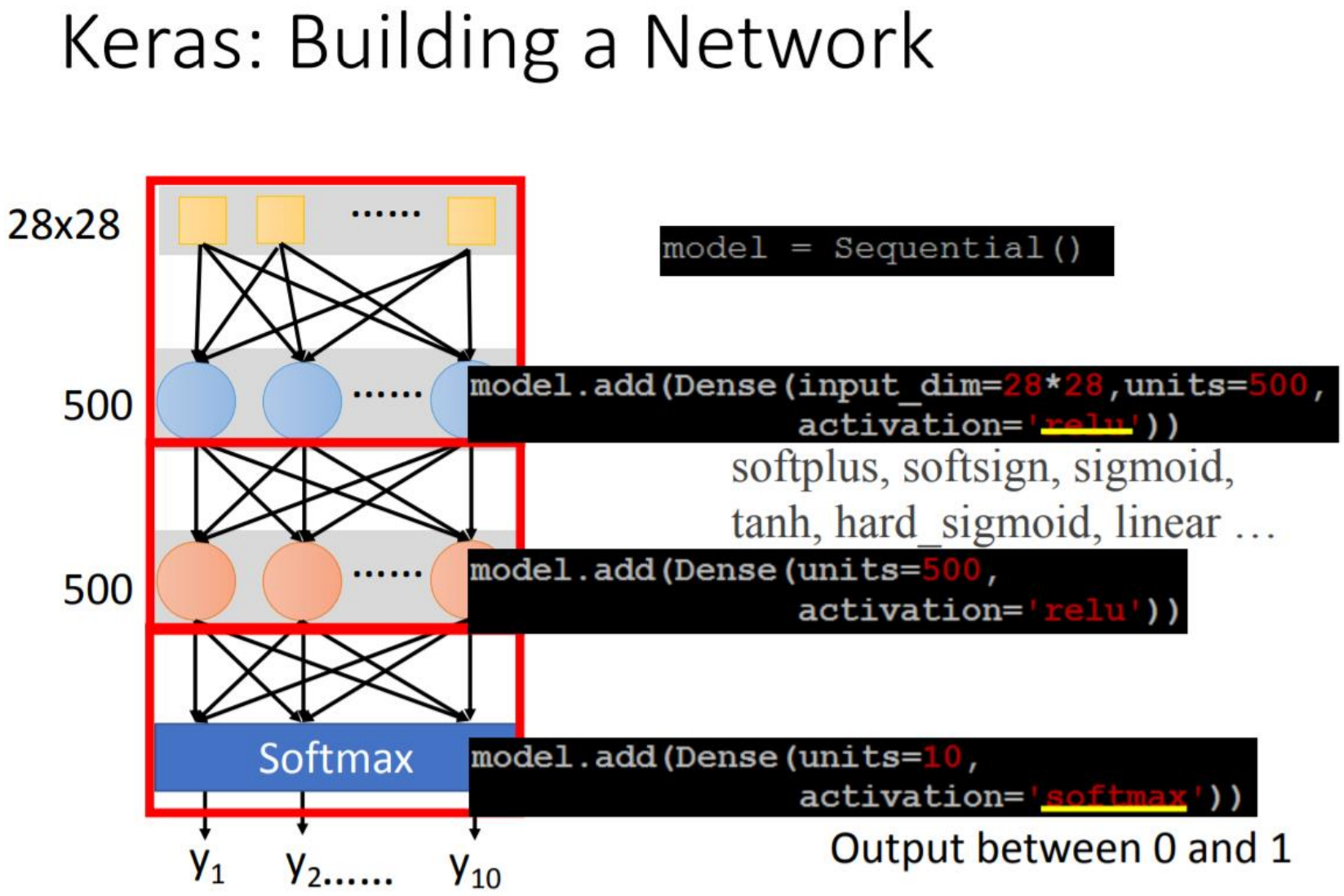
Handwriting Digit Recognition



MNIST Data: http://yann.lecun.com/exdb/mnist/

Keras provides data sets loading function: http://keras.io/datasets/

Keras: Building a Network



Configuration

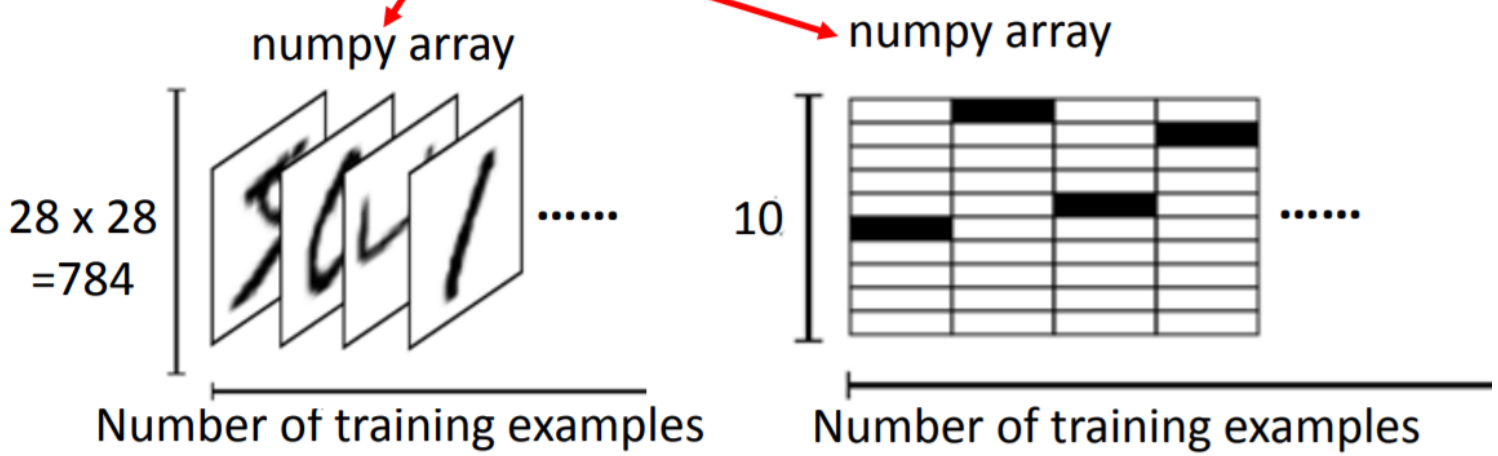
Several alternatives: https://keras.io/objectives/

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam

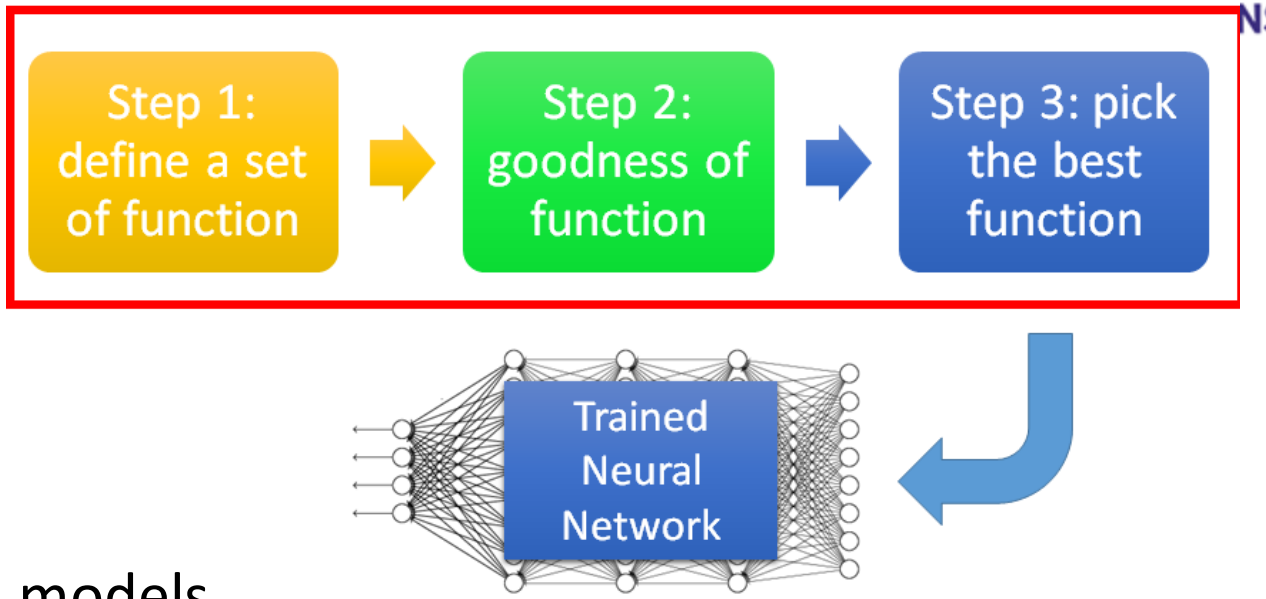
Pick the best function

```
model.fit(x_train,y_train,batch_size=100,epochs=20)
```



https://www.tensorflow.org/versions/r0.8/tutorials/mnist/beginners/index.html

Keras



Save and load models

http://keras.io/getting-started/faq/#how-can-i-save-a-keras-model

How to use the neural network (testing):

case 1:

```
score = model.evaluate(x_test,y_test)
print('Total loss on Testing Set:', score[0])
print('Accuracy of Testing Set:', score[1])
```

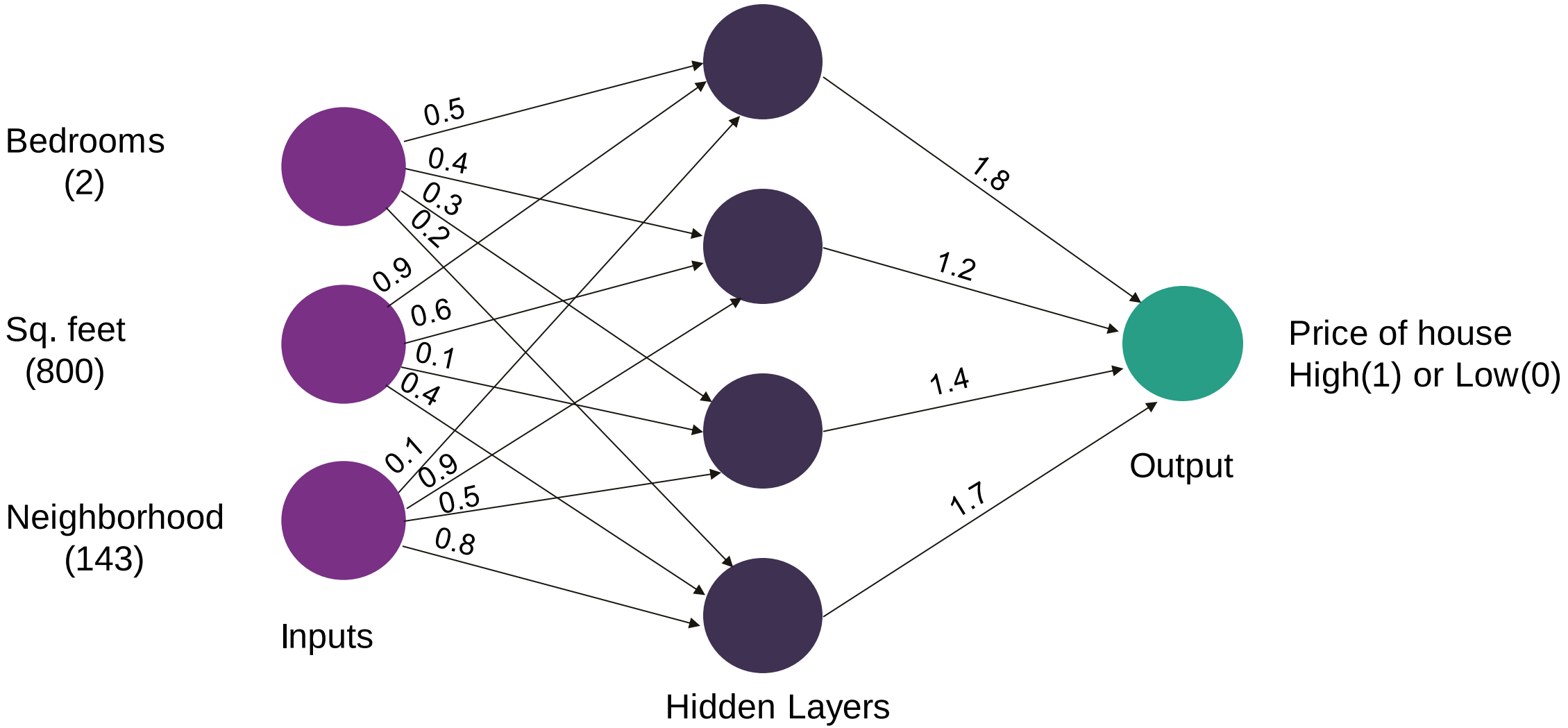
case 2:

```
result = model.predict(x_test)
```

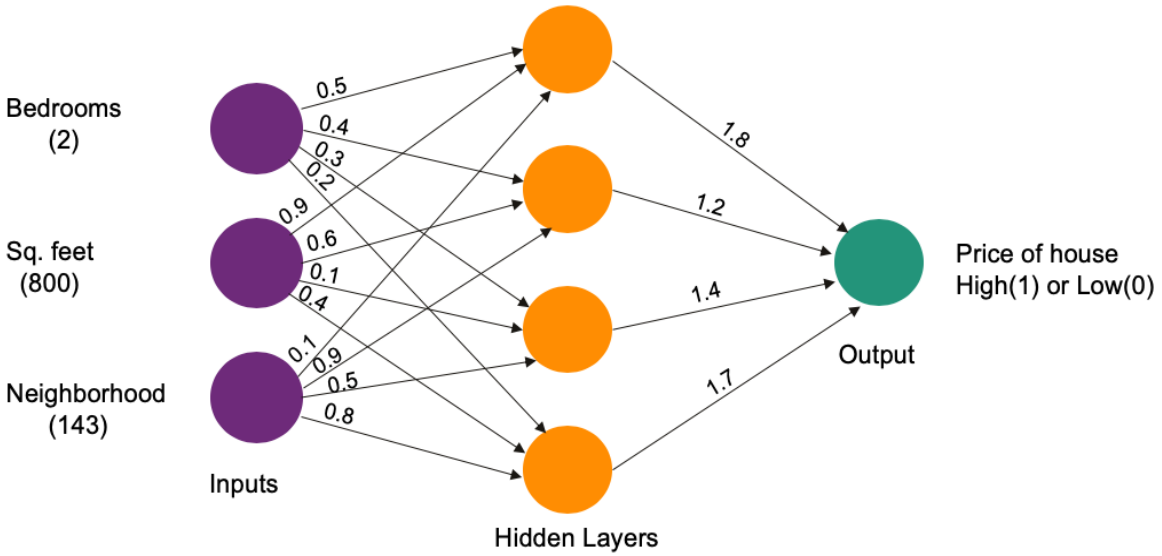
Example: Estimate the price of a house based on its attributes.

Bedrooms	Sq. Feet	Neighborhood (no. of houses in the locality)	Price high or low? High (1) , Low (0)
3	2000	90	1
2	800	143	0
2	850	167	0
1	550	267	0
4	2000	396	1

MLP Architecture

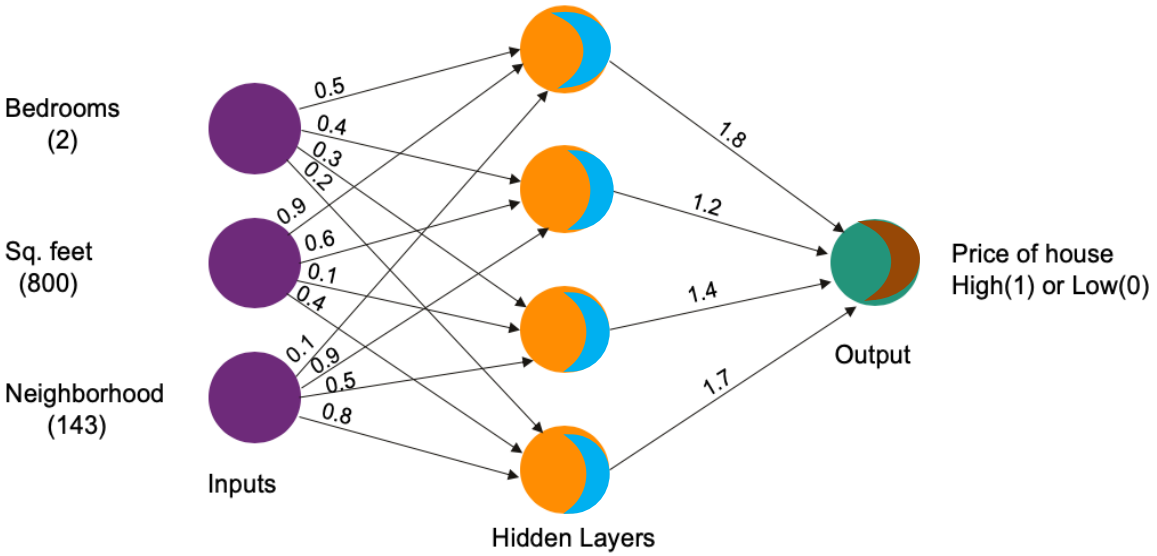


MLP Architecture

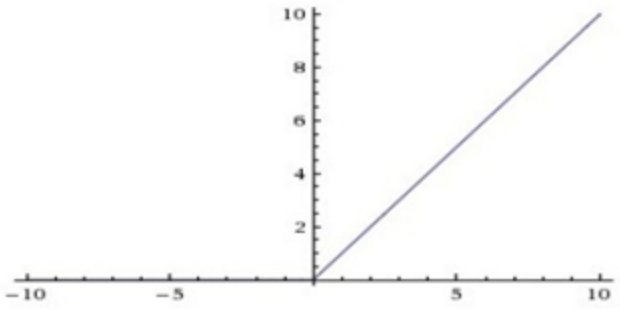



```
input_dim = X_train.shape[1]  # 3

# Housing price MLP
model = Sequential()
model.add(Dense(4, input_dim=input_dim))
model.add(Dense(1))
```

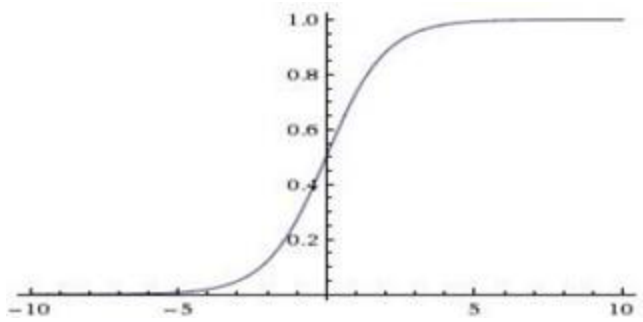
MLP Architecture : output as sigmoid




```
input_dim = X_train.shape[1]  # 3

# Housing price MLP
model = Sequential()
model.add(Dense(4, input_dim=input_dim))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

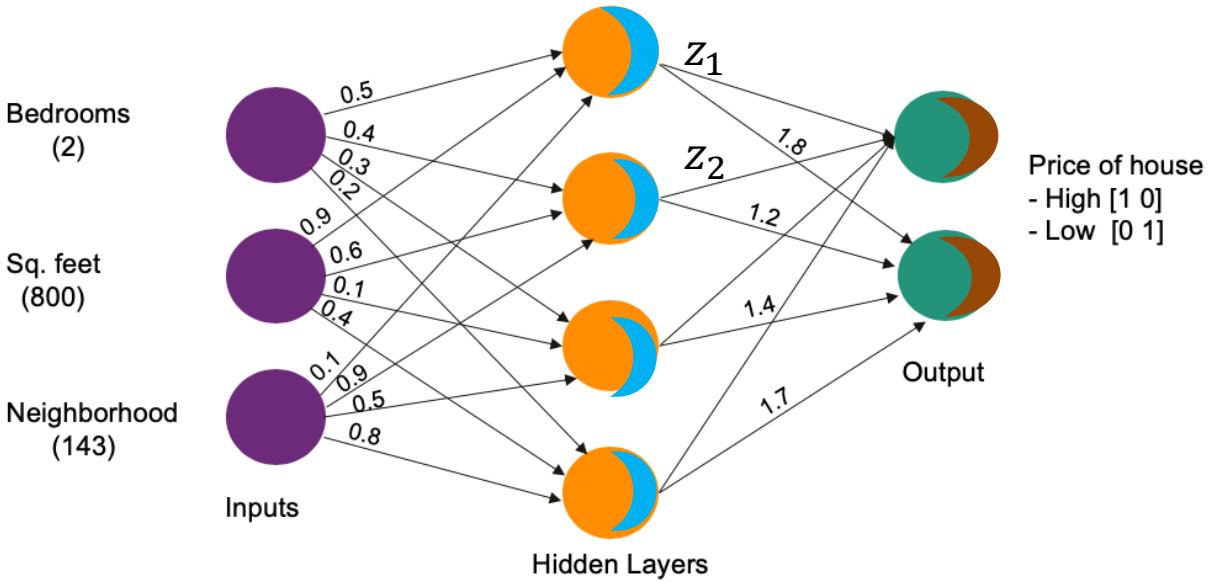


$$y = max(0, x)$$

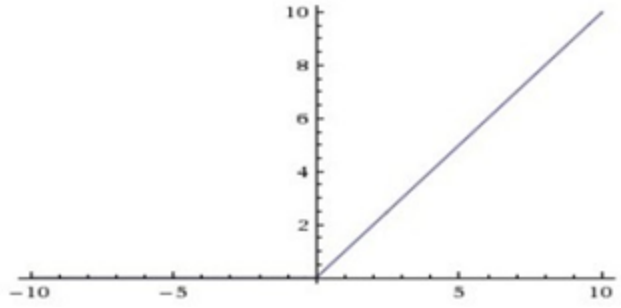


$$y = \frac{1}{1+e^{-x}}$$

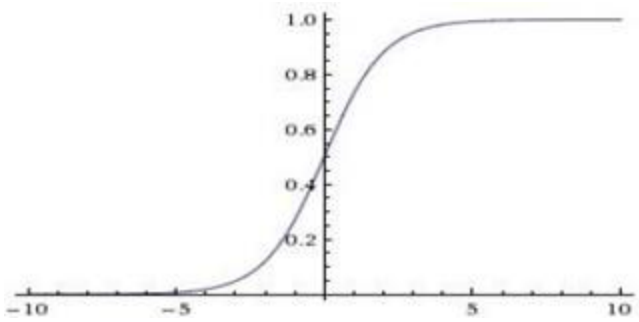
MLP Architecture : output as one-hot encoded class




```
input_dim = X_train.shape[1]  # 3
# convert list of labels to binary class matrix
y_train = np_utils.to_categorical(labels)
nb_classes = y_train.shape[1]

# Housing price MLP
model = Sequential()
model.add(Dense(4, input_dim=input_dim))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```



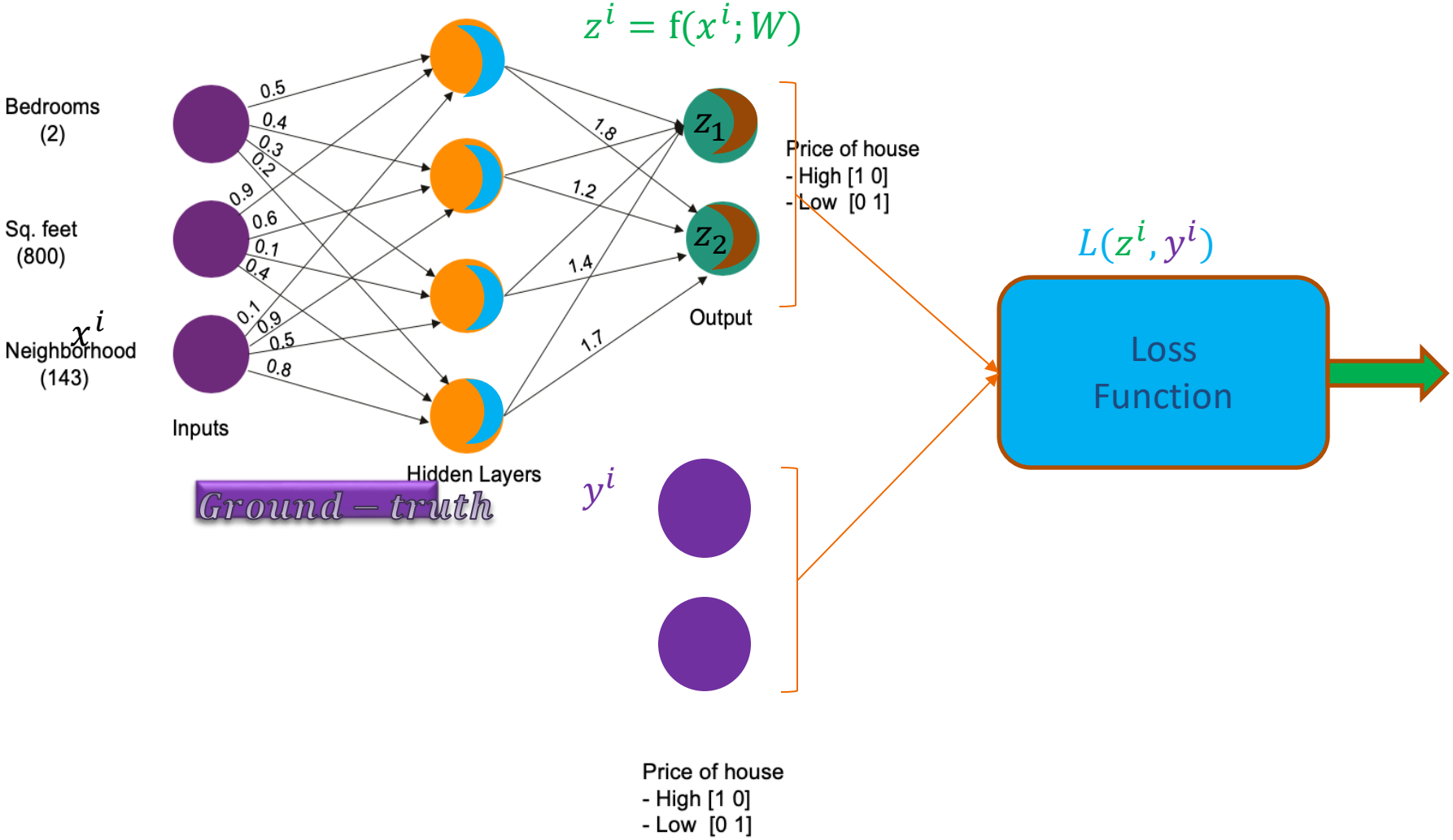
$$y = max(0, x)$$



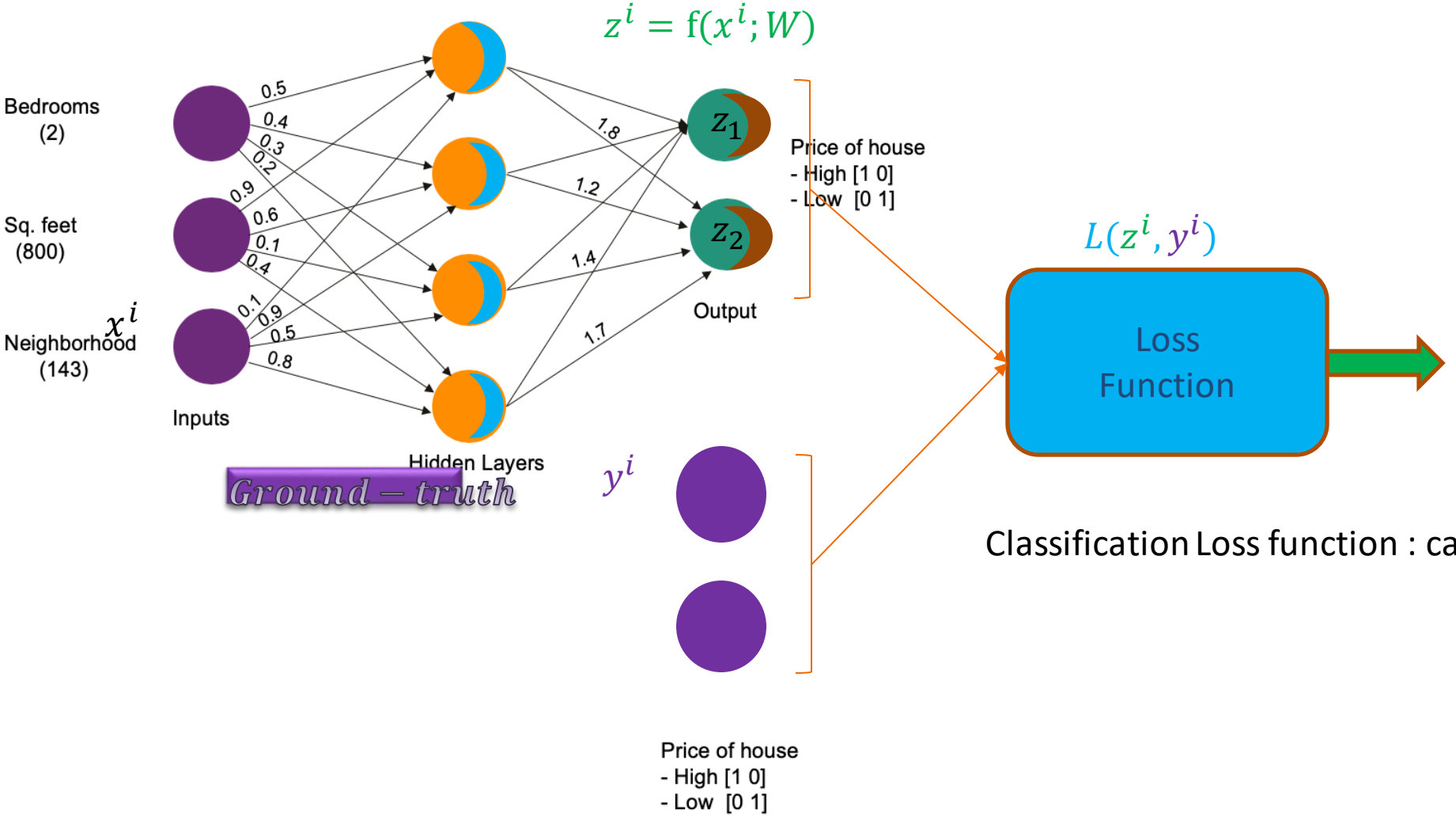
$$y = \frac{1}{1 + e^{-x}}$$

$$s_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

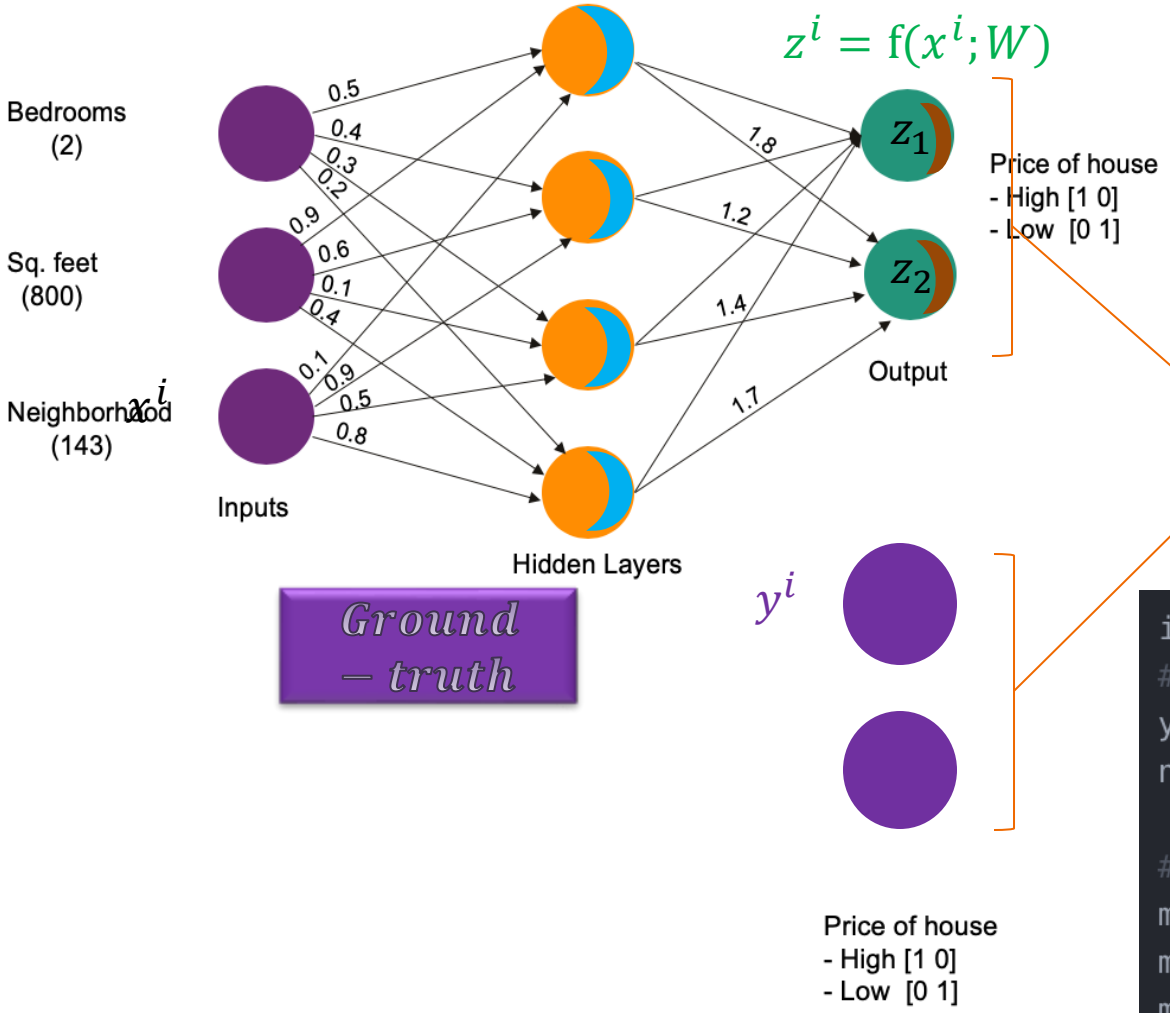
Loss / Objective



Loss / Objective

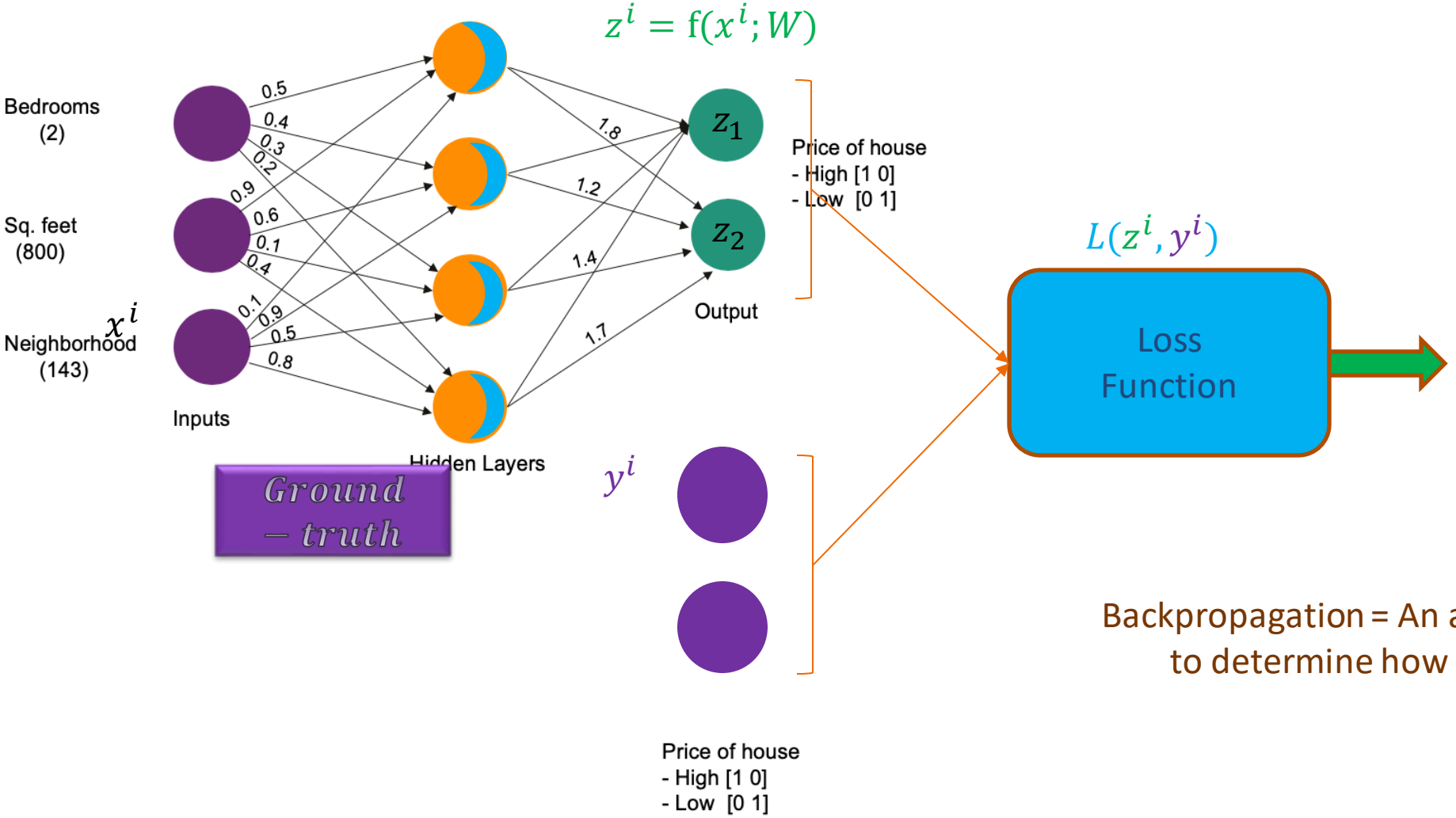


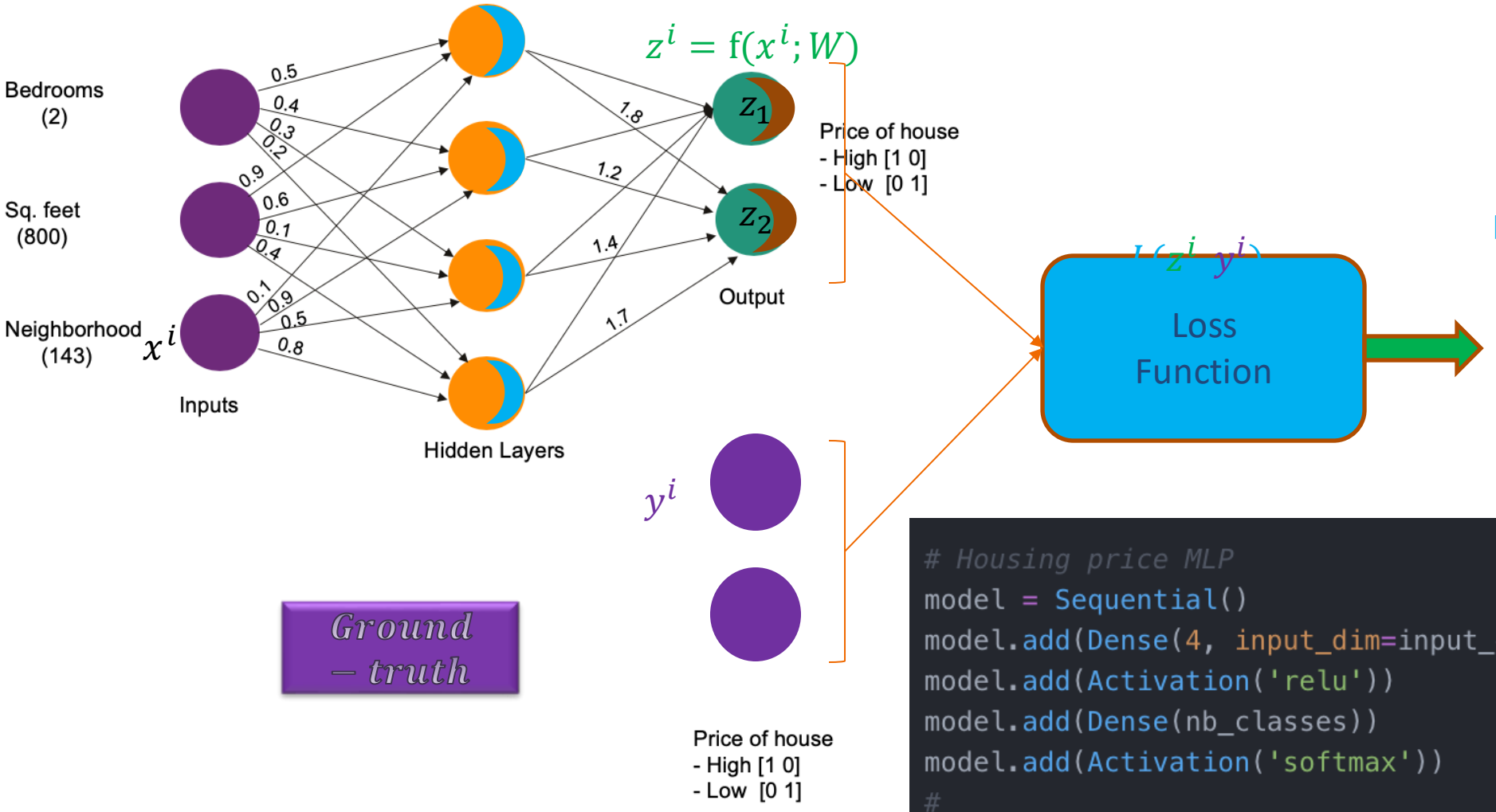
Loss / Objective




```
input_dim = X_train.shape[1]  # 3
# convert list of labels to binary class matrix
y_train = np_utils.to_categorical(labels)
nb_classes = y_train.shape[1]

# Housing price MLP
model = Sequential()
model.add(Dense(4, input_dim=input_dim))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
#
model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
```

Backpropagation





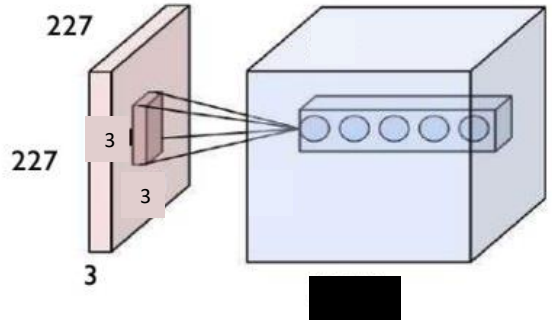
How much are we are "off" from correct prediction ?

```
# Housing price MLP
model = Sequential()
model.add(Dense(4, input_dim=input_dim))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
#
model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
# Training using backpropagation
model.fit(X_train, y_train, nb_epoch=10, batch_size=16)
```

Measuring Classification Performance

```
input_dim = X_train.shape[1]  # 3
# convert list of labels to binary class matrix
y_train = np_utils.to_categorical(labels)
nb_classes = y_train.shape[1]

# Housing price MLP
model = Sequential()
model.add(Dense(4, input_dim=input_dim))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
#
model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
# Training using backpropagation
model.fit(X_train, y_train, nb_epoch=10, batch_size=16)
print("Generating test predictions...")
preds = model.predict_classes(X_test, verbose=0)

print("Performance Measure Summary ..")
labels_pred = np.argmax(preds, axis=1)

# Print scores
print(accuracy_score(labels_test, labels_pred , average="macro")) # macro => all datapoints are treated equal
print(precision_score(labels_test, labels_pred , average="macro"))
print(recall_score(labels_test, labels_pred , average="macro"))
print(f1_score(labels_test, labels_pred , average="macro"))
```

CNN example (CIFAR-10)




```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)

# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

https://keras.io/examples/cifar10_cnn/

Thanks!!

Questions?