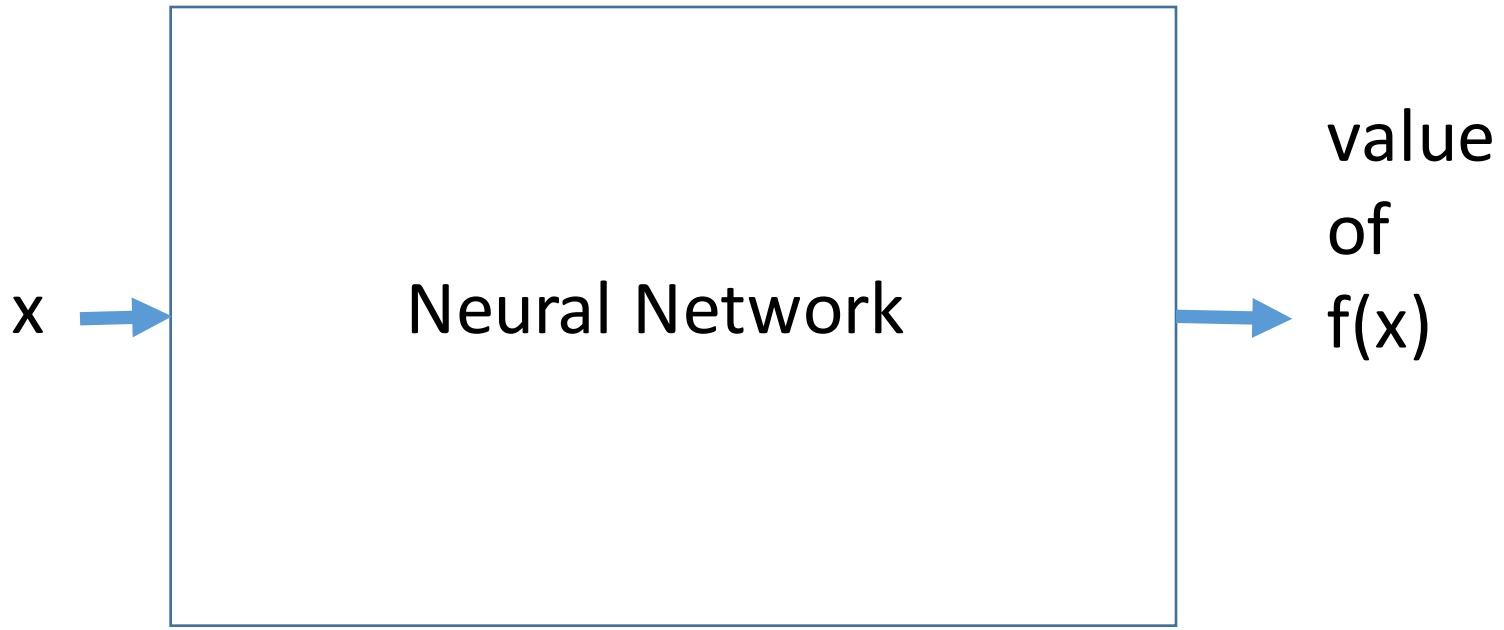
CSCE 636 Neural Networks (Deep Learning)

Lecture 5: Getting Started with Neural Networks (continued)

Anxiao (Andrew) Jiang

What a neural network does: learn a function



The neural network learns the function f(x), either exactly or approximately.

Multi-class Classification

More specifically:
the example here is single-label, multi-class classification

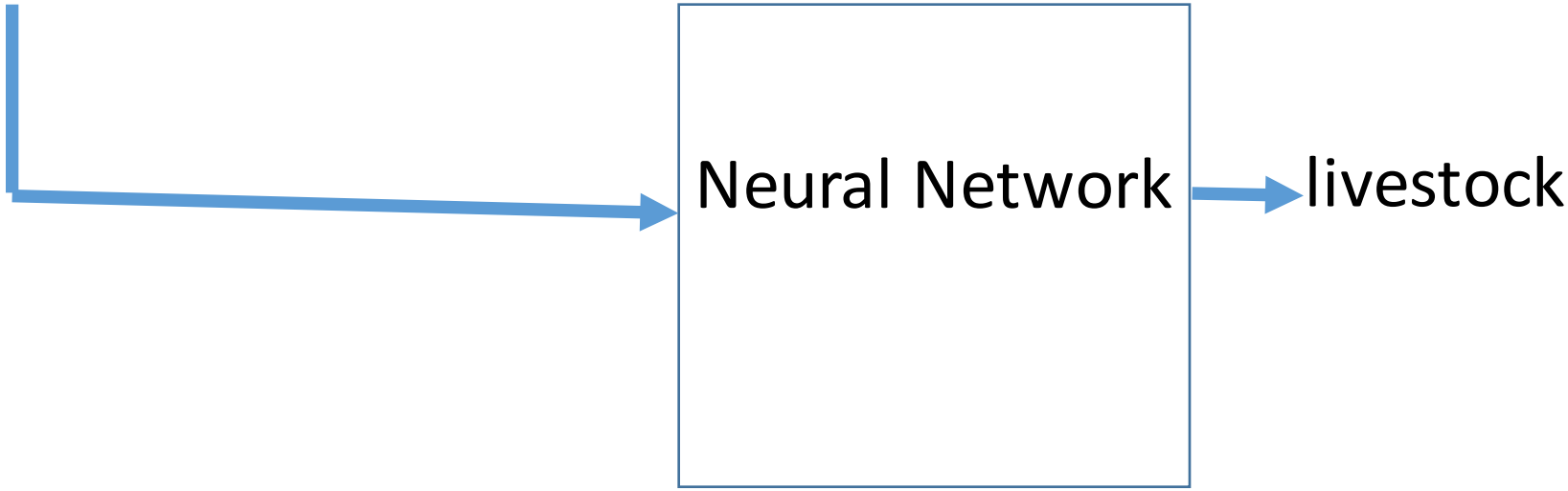
Because every sample belongs to just one class

Application: Topic classification for texts

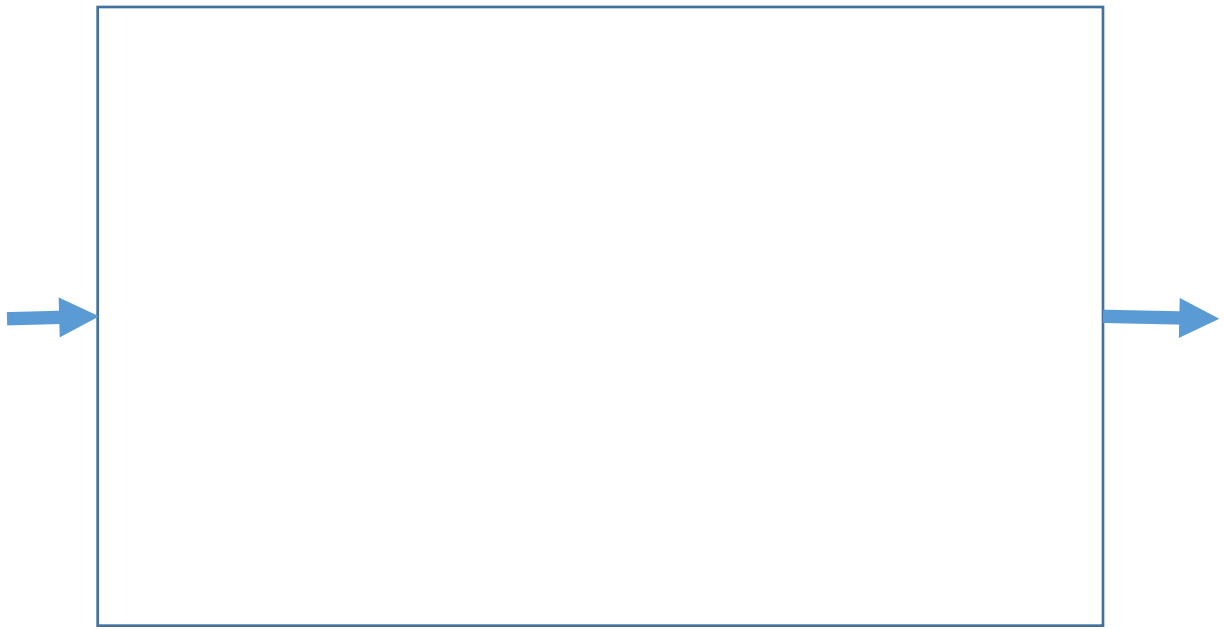
Task: Classify a newswire text by its topic

<DATELINE> CHICAGO, March 2 - </DATELINE><BODY>The American Pork Congress kicks off tomorrow, March 3, in Indianapolis with 160 of the nations pork producers from 44 member states determining industry positions on a number of issues, according to the National Pork Producers Council, NPPC.

Delegates to the three day Congress will be considering 26 resolutions concerning various issues, including the future direction of farm policy and the tax law as it applies to the agriculture sector. The delegates will also debate whether to endorse concepts of a national PRV (pseudorabies virus) control and eradication program, the NPPC said.



How to start?



Step 1: Load the dataset

- **Reuters dataset**: a set of short newswires and their topics.
- There are 46 different topics. Some topics are more represented than others.

Step 1: Load the dataset

Listing 3.12 Loading the Reuters dataset

```
from keras.datasets import reuters

(train_data, train_labels), (test_data, test_labels) = reuters.load_data(
    num_words=10000)
```

There are 8,982 training examples and 2,246 test examples.

Each example is a list of integers (word indices).

<DATELINE> CHICAGO, March 2 - </DATELINE><BODY>The American Pork Congress kicks off tomorrow, March 3, in Indianapolis with 160 of the nations pork producers from 44 member states determining industry positions on a number of issues, according to the National Pork Producers Council, NPPC.

Delegates to the three day Congress will be considering 26 resolutions concerning various issues, including the future direction of farm policy and the tax law as it applies to the agriculture sector. The delegates will also debate whether to endorse concepts of a national PRV (pseudorabies virus) control and eradication program, the NPPC said.

Words are represented by a list of integers.

Step 2: Prepare the data

One-hot encode training and test data

We get:

x_train (newswires in training data): shape is 8982 x 10000

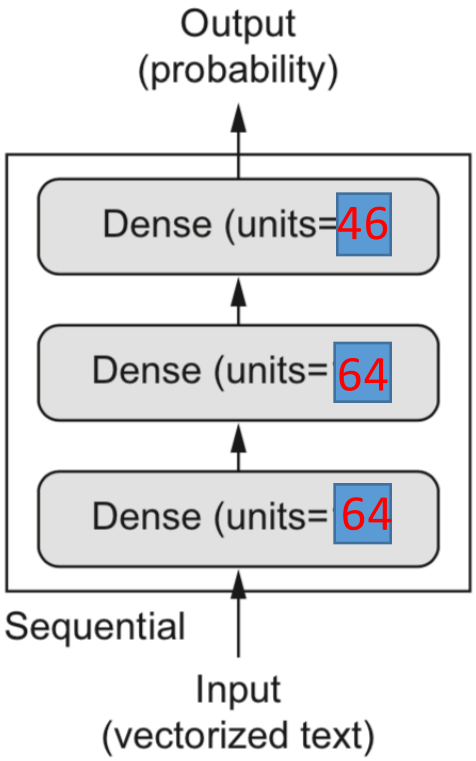
x_test (newswires in test data): shape is 2246 x 10000

one_hot_train_labels (labels in training data): shape is 8982 x 46

one_hot_test_labels (labels in test data): shape is 2246 x 46

[1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74, 2979,
3554, 14, 46, 4689, 4329, 86, 61, 3499, 4795, 14, 61, 451, 4329, 17, 12]

Step 3: Build neural network

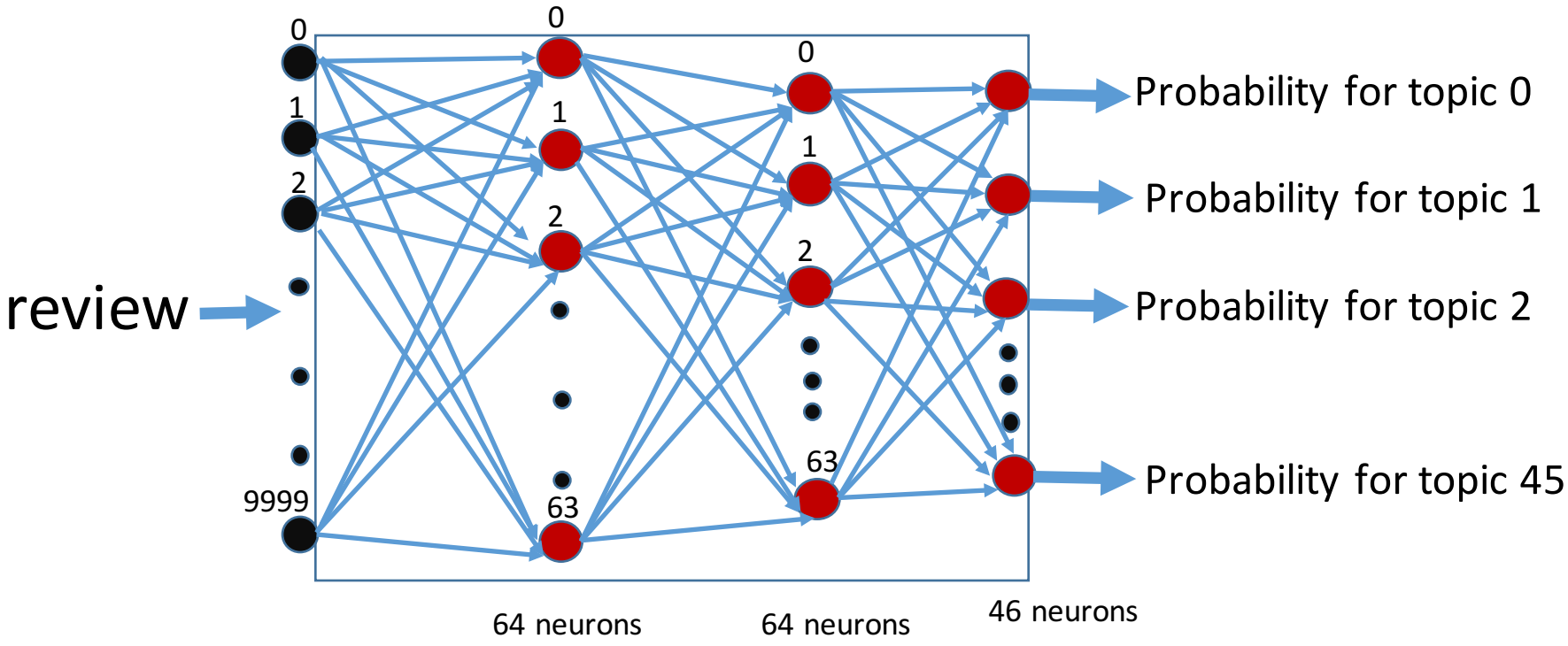


The three-layer network

Listing 3.15 Model definition

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```



Step 4: choose loss function, optimizer, and target metrics

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

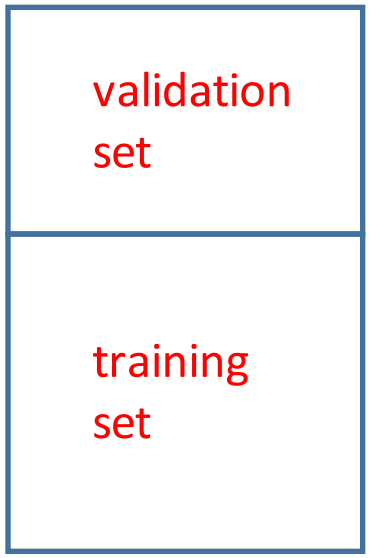
$$CCE = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{J} y_j \cdot log(\hat{y}_j) + (1 - y_j) \cdot log(1 - \hat{y}_j)$$

Step 4: get validation set

Listing 3.17 Setting aside a validation set

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```



1,000 newswires and their labels.

7982 reviews and their labels.

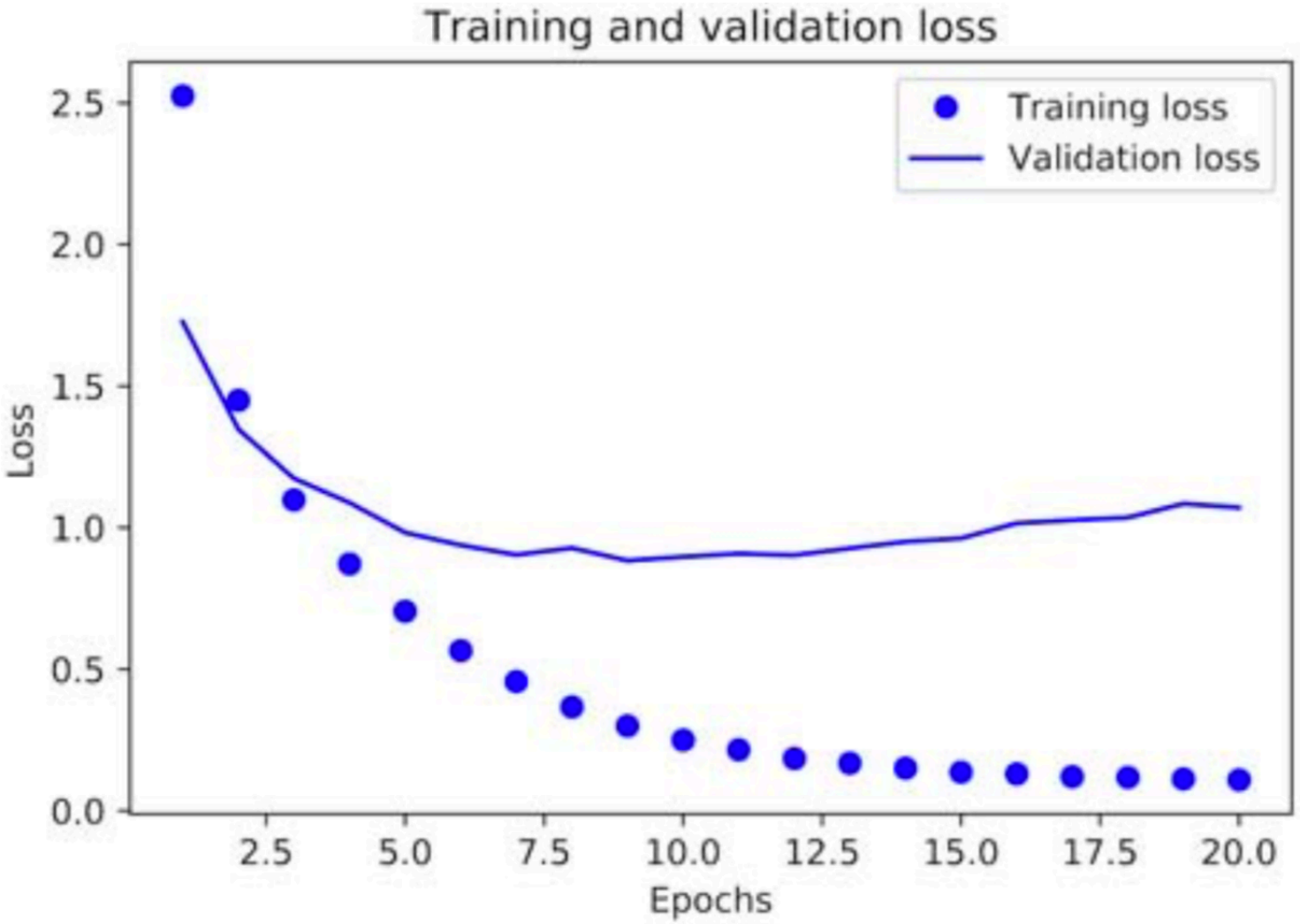
Shape of x_val: 1000 x 10000
Shape of partial_x_train: 7982 x 10000

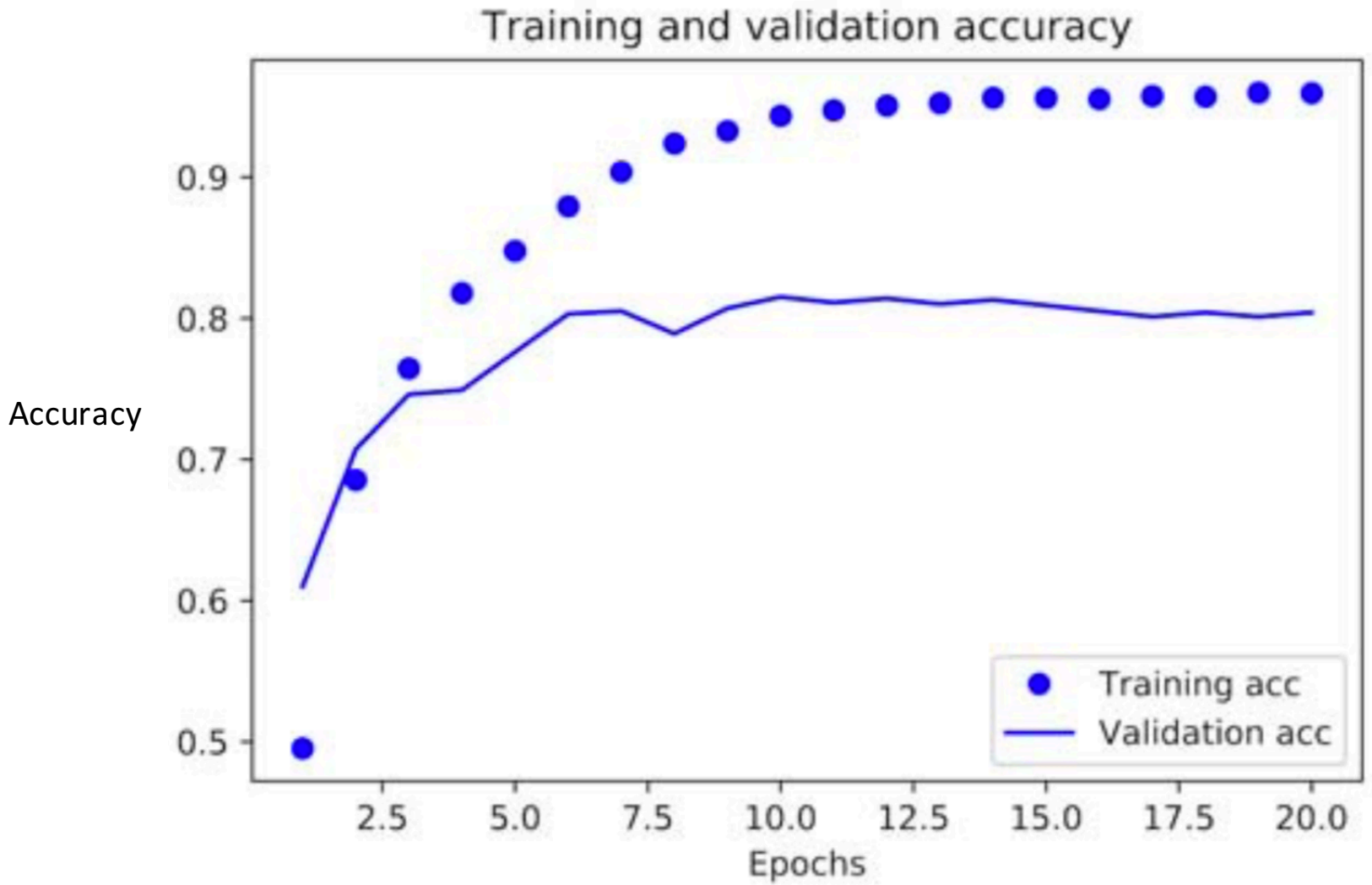
Shape of y_val: 1000 x 46
Shape of partial_y_train: 7982 x 46

Step 5: Train neural network

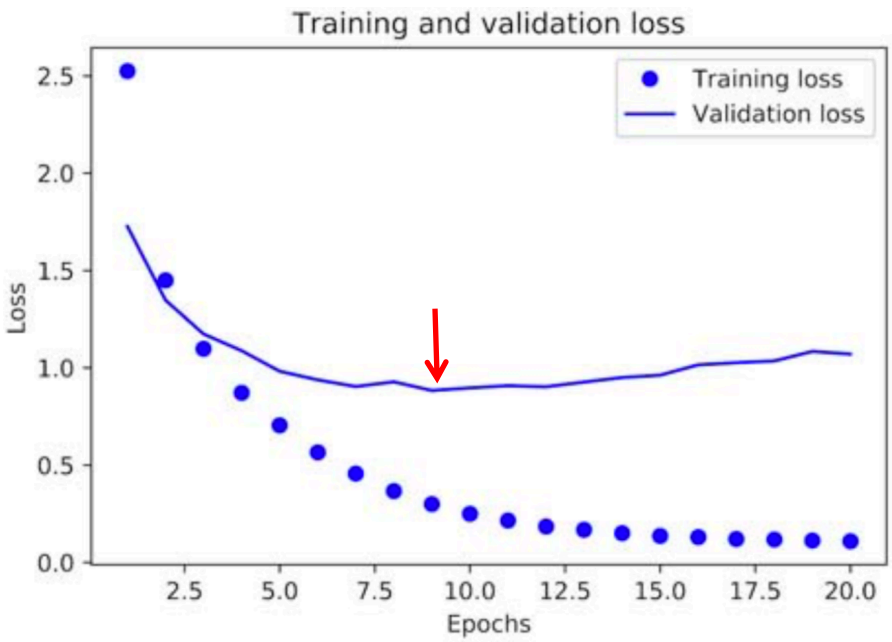
Listing 3.18 Training the model

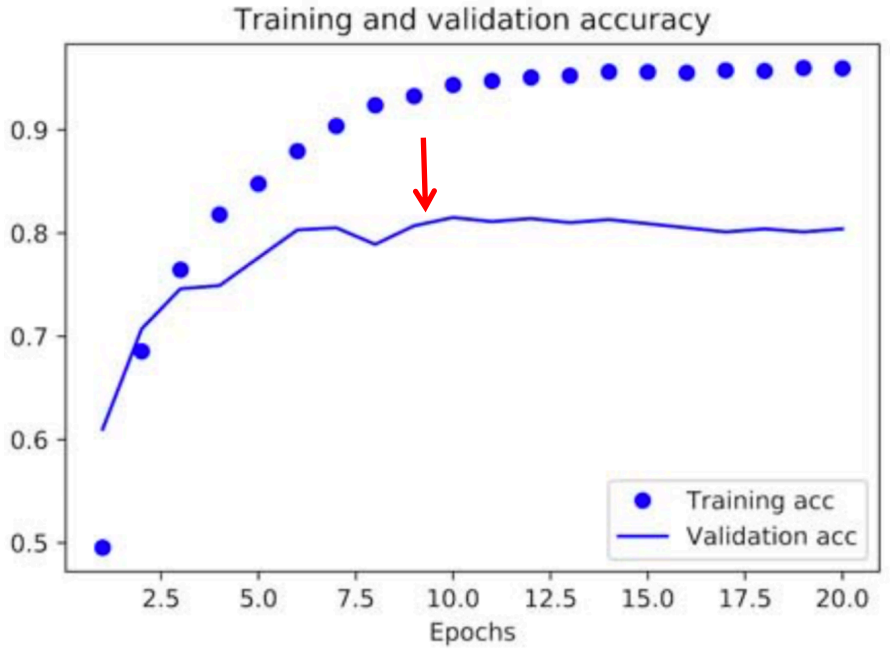
```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```


Training and validation loss
Training loss
Validation loss
Loss
Epochs
0.0
0.5
1.0
1.5
2.0
2.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Training and validation accuracy
Accuracy
0.9
0.8
0.7
0.6
0.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs
Training acc
Validation acc

NN starts to overfit after about 9 epochs





Train a new network from scratch for 9 epochs

```
model.fit(partial_x_train,
          partial_y_train,
          epochs=9,
          batch_size=512,
          validation_data=(x_val, y_val))
```

Step 6: Test the trained neural network

```
results = model.evaluate(x_test, one_hot_test_labels)
```

Show test results:

```
>>> results
[0.9565213431445807, 0.7969723953695458]
```

loss

Accuracy: about 80%

Step 7: Use trained network for prediction

Listing 3.22 Generating predictions for new data

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

Each entry in `predictions` is a vector of length 46:

```
>>> predictions[0].shape
(46,)
```

The coefficients in this vector sum to 1:

```
>>> np.sum(predictions[0])
1.0
```

The largest entry is the predicted class—the class with the highest probability:

```
>>> np.argmax(predictions[0])
4
```

A different way to handle the labels and the loss

We mentioned earlier that another way to encode the labels would be to cast them as an integer tensor, like this:

```
y_train = np.array(train_labels)
y_test = np.array(test_labels)
```

Shape of y_train: (8982,)
Shape of y_test: (2246,)

The only thing this approach would change is the choice of the loss function. The loss function used in listing 3.21, `categorical_crossentropy`, expects the labels to follow a categorical encoding. With integer labels, you should use `sparse_categorical_crossentropy`:

```
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])
```

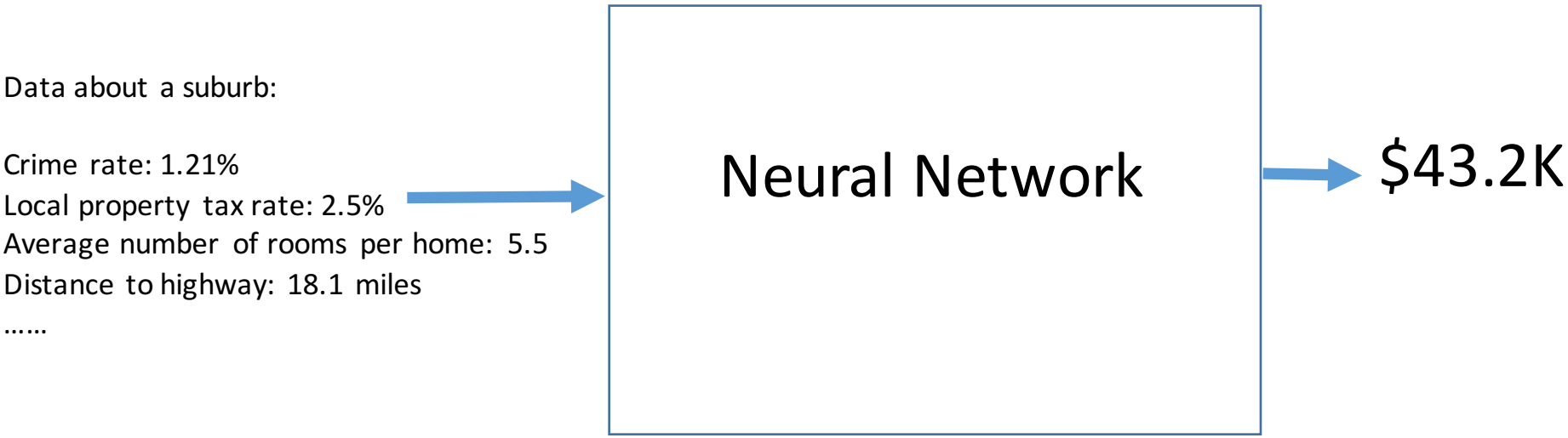
This new loss function is still mathematically the same as `categorical_crossentropy`; it just has a different interface.

Regression

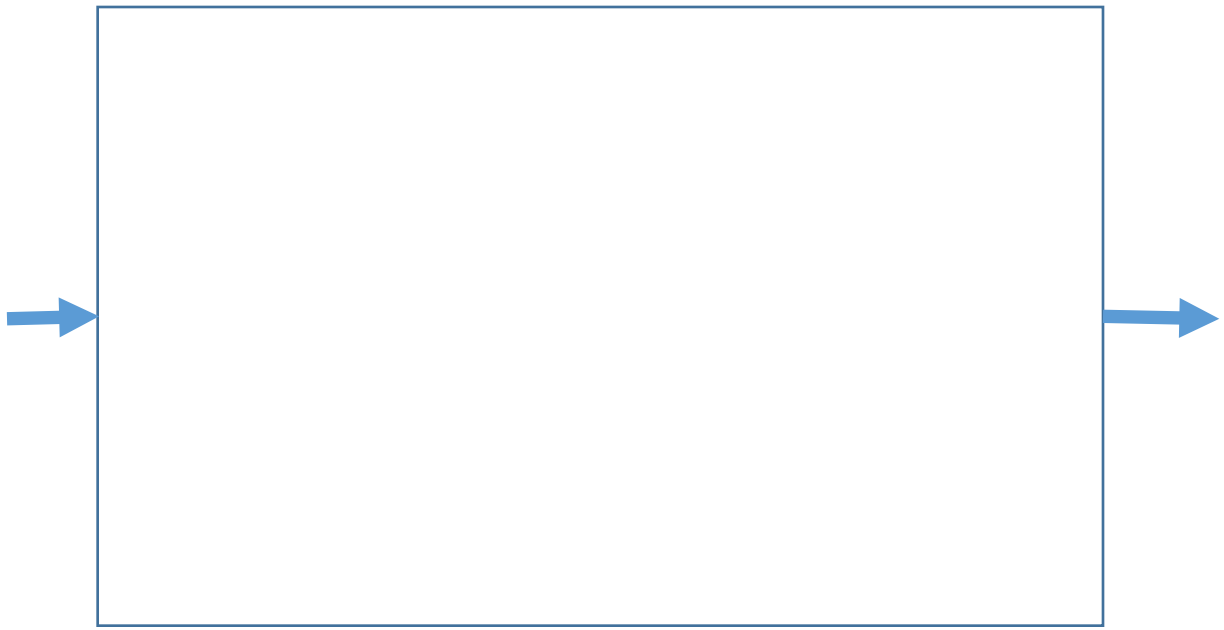
Predict a continuous value (e.g., price, temperature ….) instead of a discrete label.

Application: Predicting House Prices

Task: Given data points about a suburb, predict the median price of homes in the suburb.



How to start?



Step 1: Load the dataset

- **Boston Housing Price dataset**: data about 506 suburbs in Boston in the mid-1970s, and the median home price of each suburb.
- Dataset is very small: only 506 samples.
- Training set: 404 samples.
- Test set: 102 samples.
- Each sample has 13 numerical features in its input data.

Step 1: Load the dataset

Listing 3.24 Loading the Boston housing dataset

```
from keras.datasets import boston_housing

(train_data, train_targets), (test_data, test_targets) =
➥boston_housing.load_data()
```

train_data: input training data of shape (404, 13)

train_targets: output training data of shape (404,)

test_data: input test data of shape (102, 13)

test_targets: output test data of shape (102,)

Data about a suburb:

Crime rate: 1.21%
Local property tax rate: 2.5%
Average number of rooms per home: 5.5
Distance to highway: 18.1 miles
……

Step 2: Prepare the data

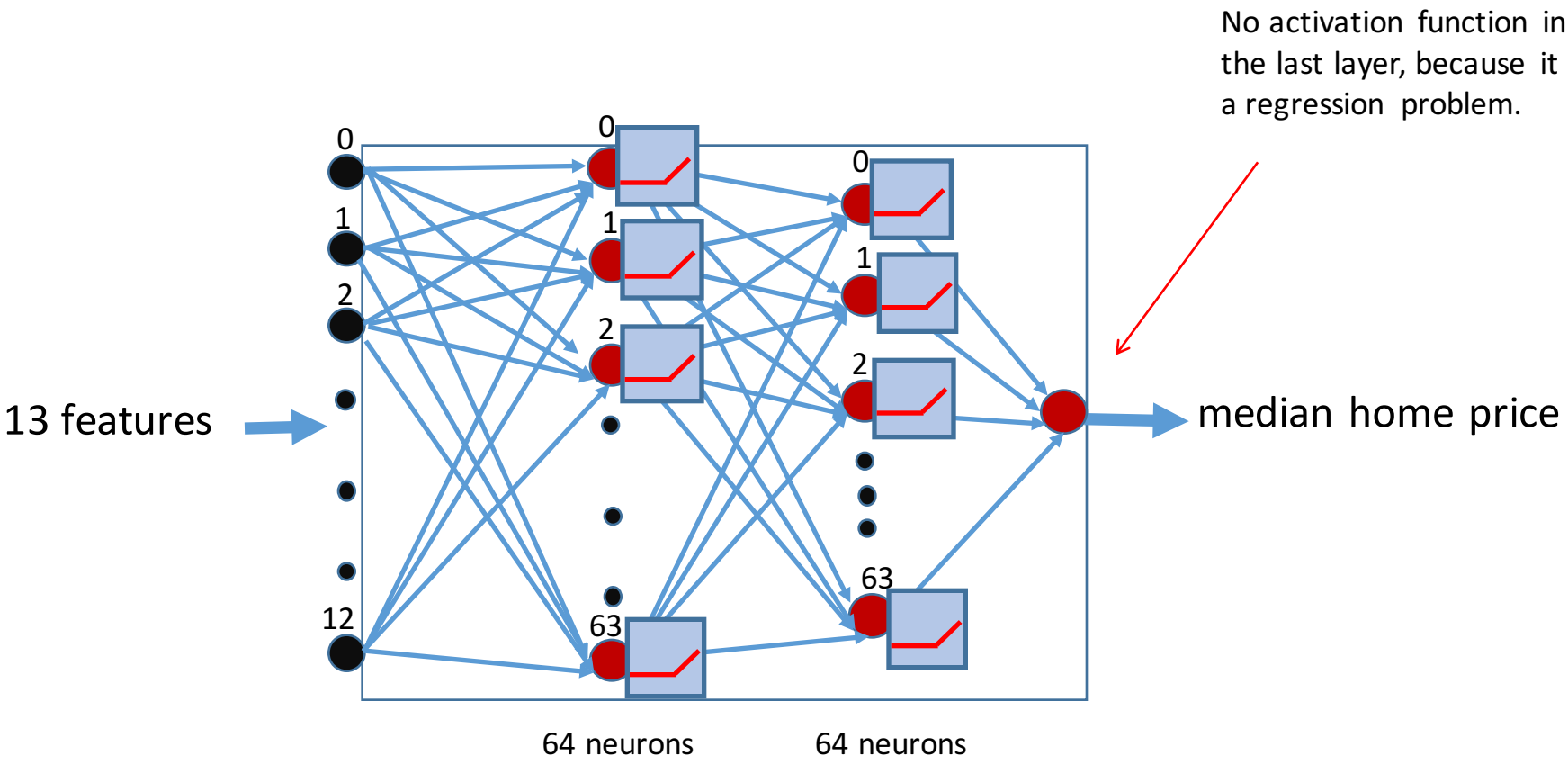
- The 13 features in input data have quite different range.
- Good solution: normalize features to mean 0 and variance 1.
- Benefit: Make the NN easier to train.

Listing 3.25 Normalizing the data

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

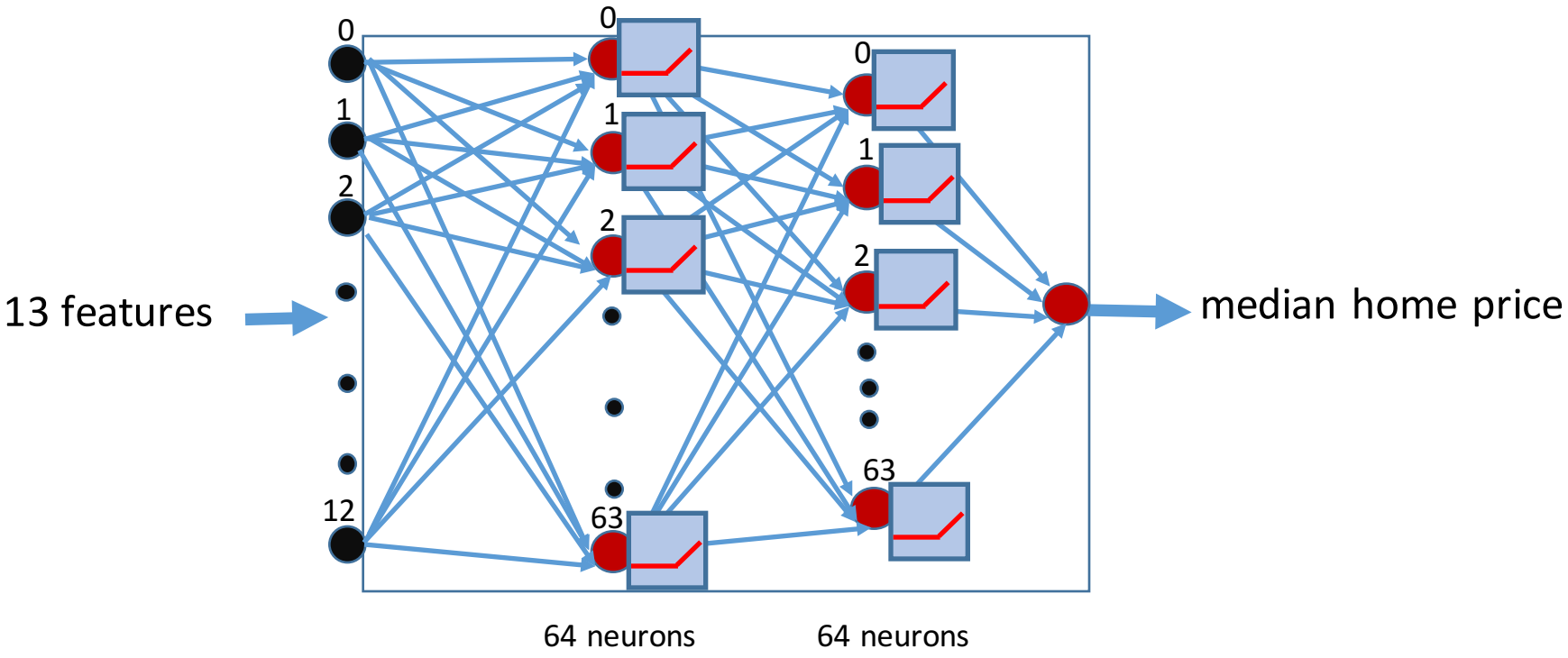
Step 3: Build the neural network, compile it



```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu',
                       input_shape=(train_data.shape[1],)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1)) # No activation function  in the last layer
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

Shape: (404, 13)

13



MSE: Mean Square Error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

MAE: Mean Absolute Error

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Optimizer: RMSprop and Adam (which is RMSprop + momentum method) are both good choices.

Step 4: partition training data into training data and validation data

Training set is too small: only 404 samples

Validation set will also be too small: say, only 101 samples

Validation performance can have high variance (i.e., not accurate and far from the test performance), making it hard to tune parameters (such as the number of epochs to train, and hyper-parameters in NN) well

Best practice in such situations

Use K-fold cross-validation

K-fold Cross Validation

- Split the training set into K parts (often K=4 or 5)
- Initiate K identical NN models
- Train each NN on K-1 parts of data and validate on the other part
- Final validation score: use the average of the K validation scores.

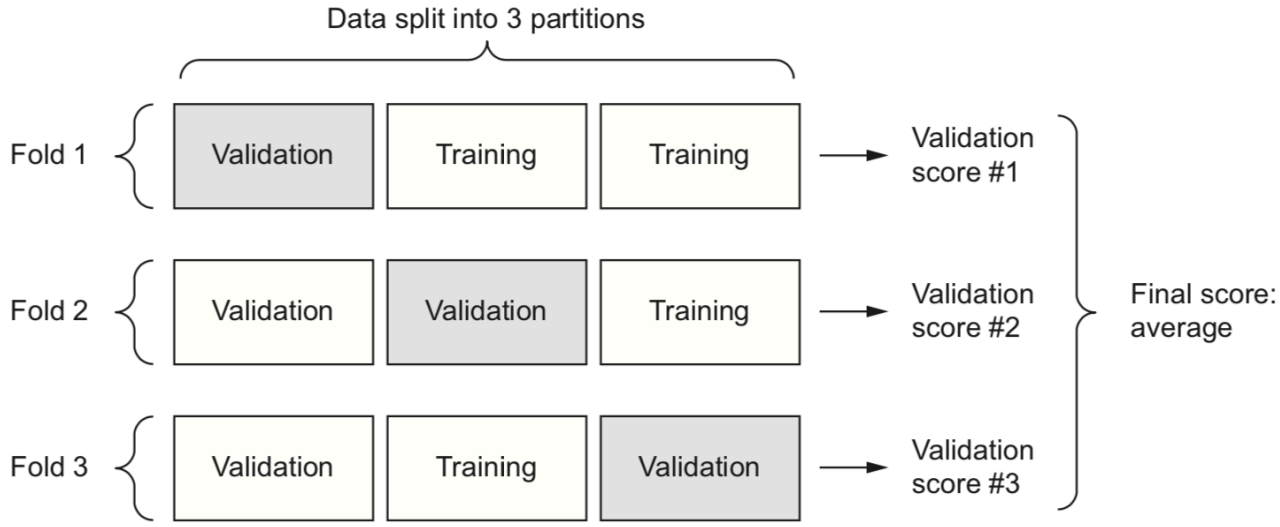


Figure 3.11 3-fold cross-validation

Step 5: Train and test neural network

Train neural network using K-fold Cross Validation.

Tune parameters based on the performance of K-fold Cross Validation.

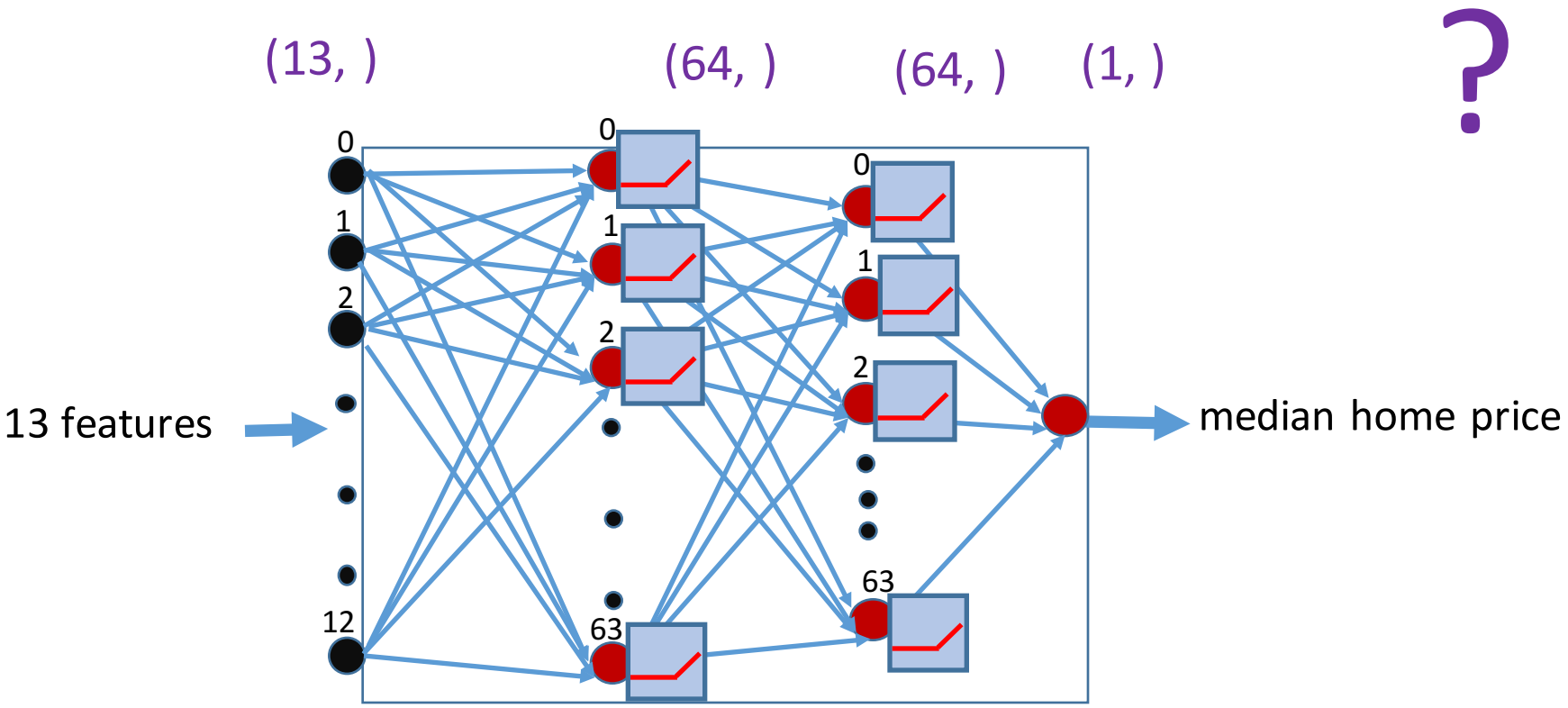
After parameters are tuned, train a final NN using all the training data.

Test the trained NN's performance on the test data.

A few more words on the shapes of data …

Say that the mini-batch size is 5 during training.

What is the shape of data in each layer?

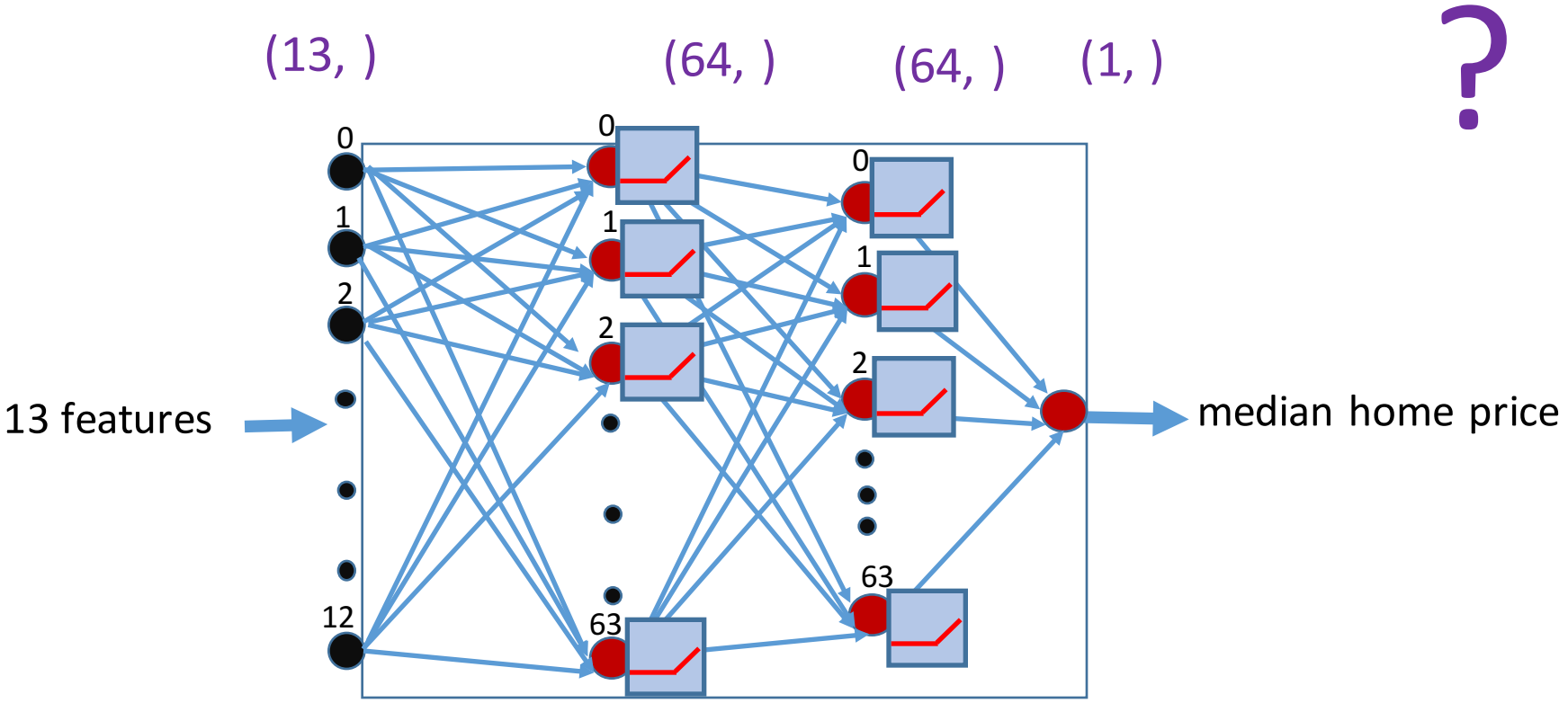


Say that the mini-batch size is 5 during training.

What is the shape of data in each layer?

It is OK to theoretically think so.

But in reality …

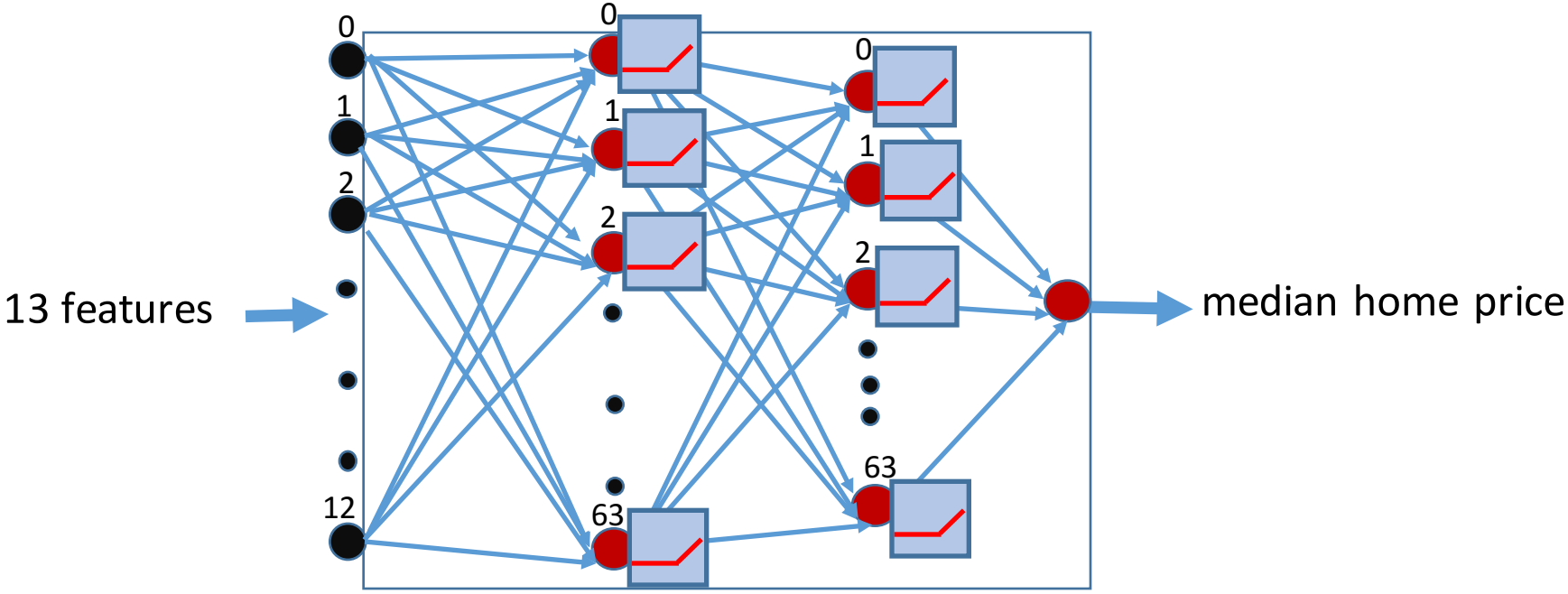


Say that the mini-batch size is 5 during training.

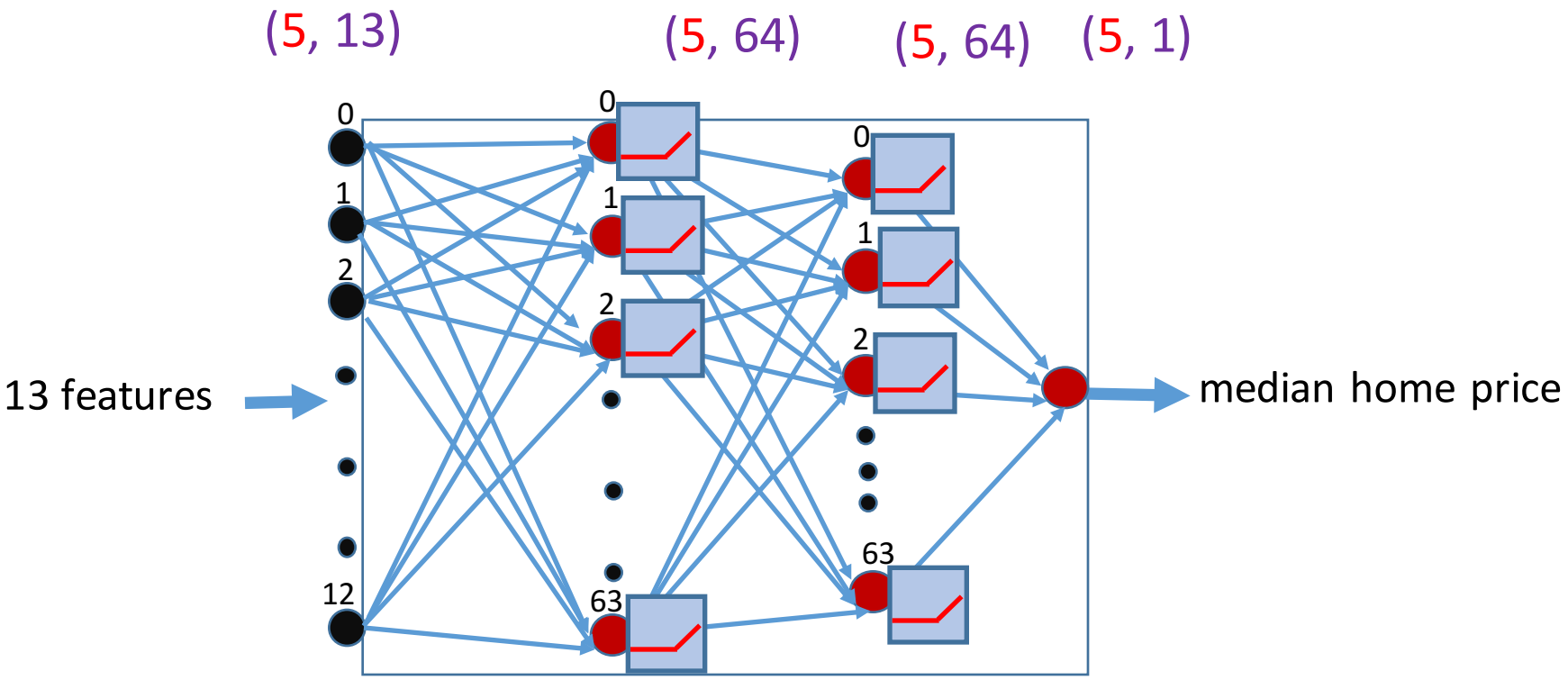
What is the shape of data in each layer?

In reality:

(5, 13) (5, 64) (5, 64) (5, 1)

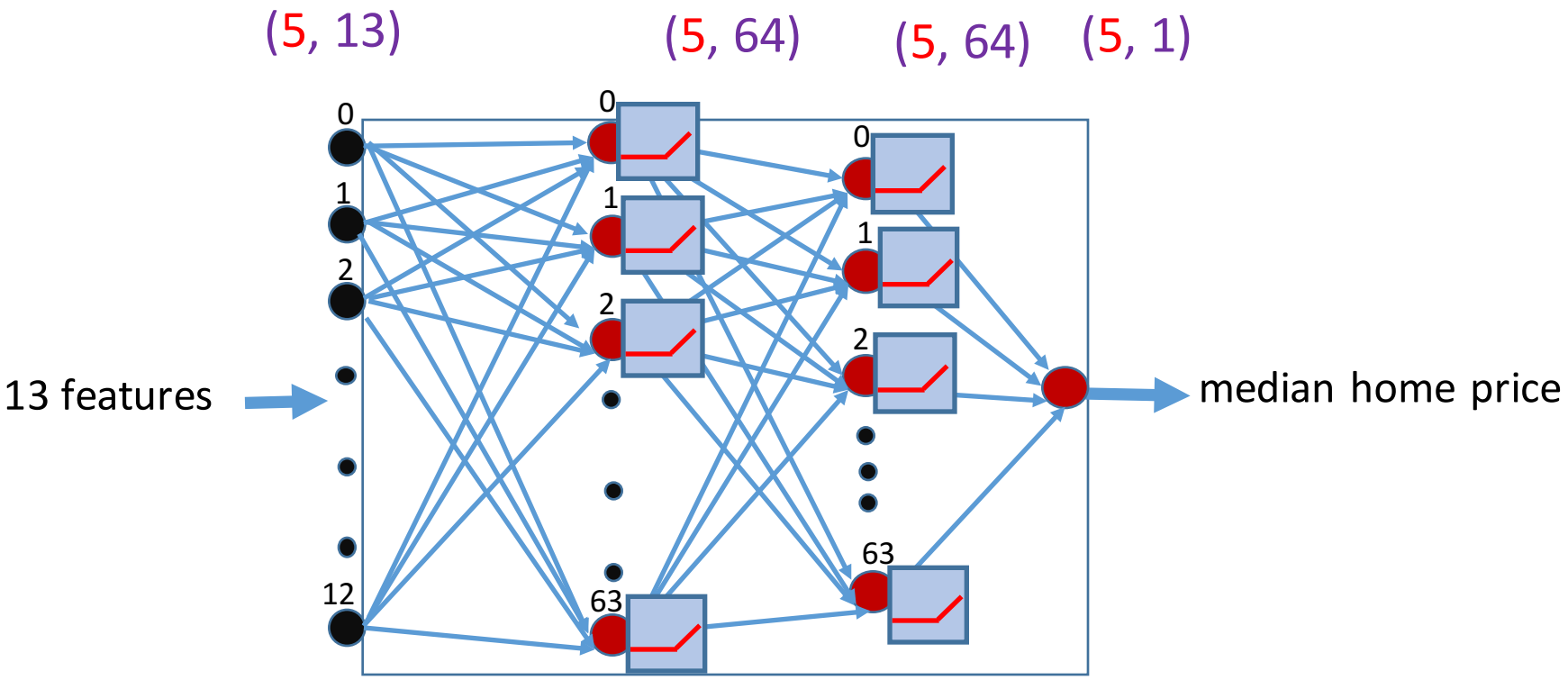


For GPU, same type of tensor operation (just one more dimension). Nearly same speed.



For GPU, same type of tensor operation (just one more dimension). Nearly same speed.

So we can choose larger mini-batch size to speed up training, as long as it does not exceed the memory size of the GPU.



For GPU, same type of tensor operation (just one more dimension). Nearly same speed.

So we can choose larger mini-batch size to speed up training, as long as it does not exceed the memory size of the GPU.

However, the mini-batch size should not be too large, either. Study shows that when a mini-batch size is too large, the gradient-descent method may easily get trapped and stop too early. The randomness in the descent directions of mini-batches actually helps somehow.