بسم الله الرحمن الرحیم

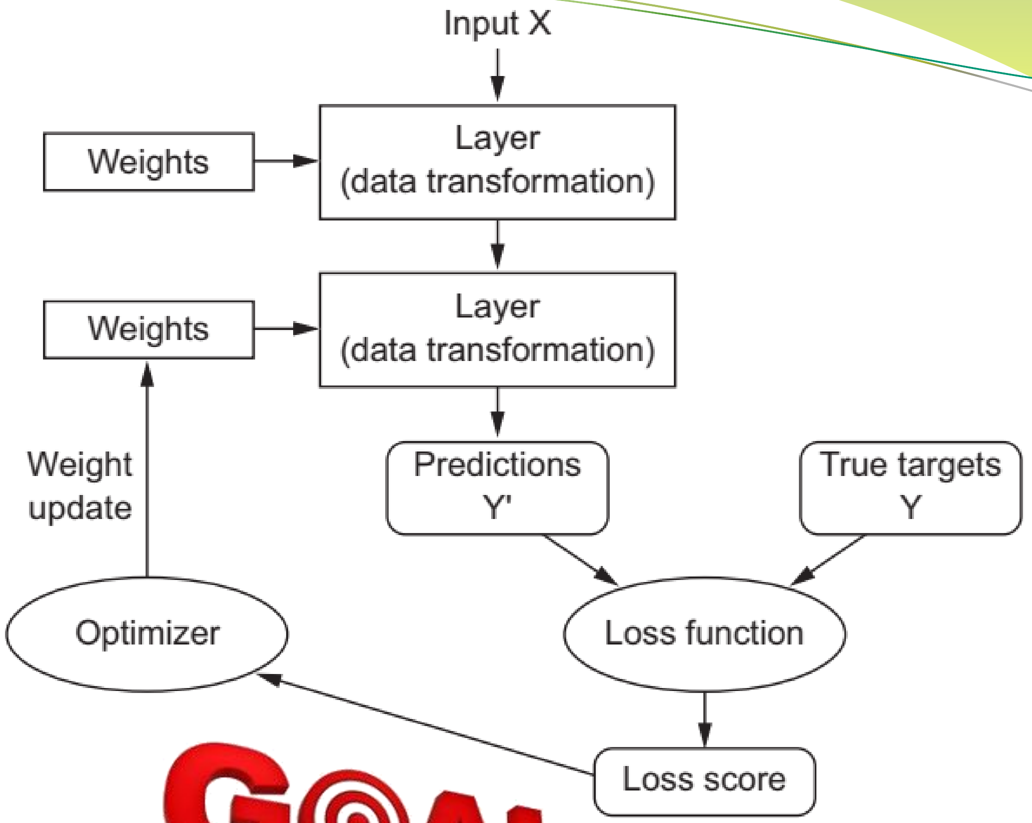
Deep Learning

Mohammad Reza Mohammadi

2021

How deep learning works?

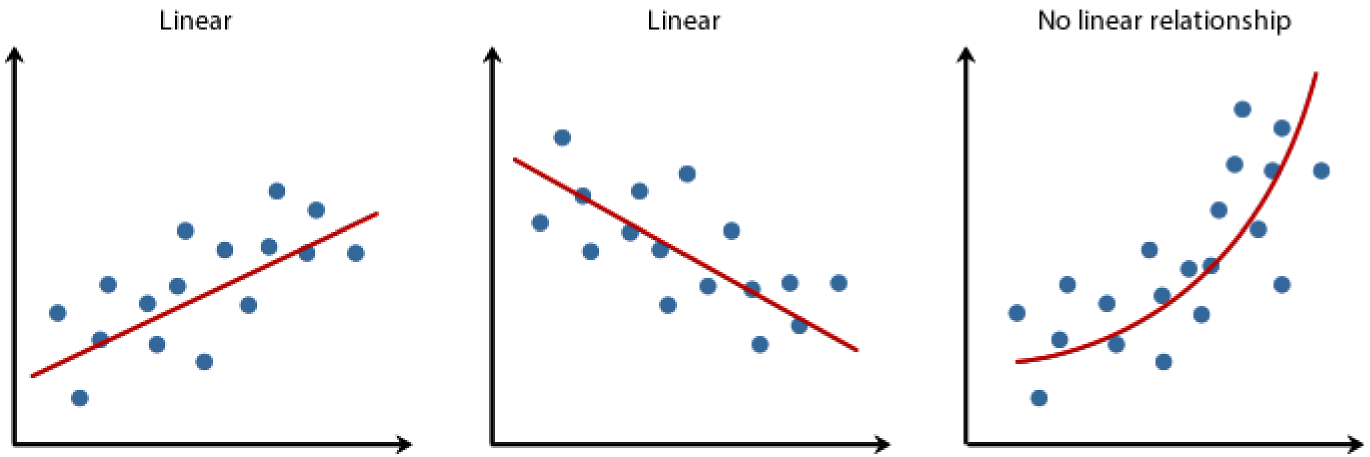
- Last-layer activation:
  - This establishes useful constraints on the network's output
- Loss function:
  - This should match the type of problem you're trying to solve
  - It isn't always possible to directly optimize for the metric that measures success on a problem
  - Need to be computable given only a mini-batch of data and must be differentiable



Regression

- Consists of predicting a continuous value
- For instance, predicting the temperature tomorrow, given meteorological data
- We can use linear function as the last-layer activation function
- What is the appropriate loss?
  - mean squared error!

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \|\boldsymbol{y} - f(\boldsymbol{x};\boldsymbol{\theta})\|^2$$



Predicting house prices

- The Boston Housing Price dataset
  - Relatively small: 404 training, 102 test
  - Features of data have different scales
  - Targets are in thousands of dollars

```
from keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()
```

```
print(train_data.shape)
print(train_targets.shape)
```

```
(404, 13)
(404,)
```

```
std = train_data.std(axis=0)
print(std)
```

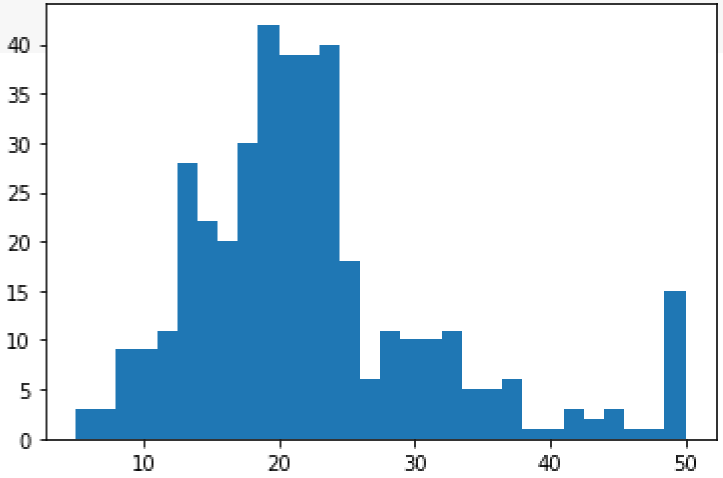
```
[9.22929073e+00 2.37382770e+01 6.80287253e+00 2.40939633e-01
 1.17147847e-01 7.08908627e-01 2.79060634e+01 2.02770050e+00
 8.68758849e+00 1.66168506e+02 2.19765689e+00 9.39946015e+01
 7.24556085e+00]
```

Predicting house prices

- The Boston Housing Price dataset
  - Relatively small: 404 training, 102 test
  - Features of data have different scales
  - Targets are in thousands of dollars

```
print(train_targets)
plt.hist(train_targets, 30)
```

```
[ 15.2, 42.3, 50. ... 19.4, 19.4, 29.1]
```



Predicting house prices

- Problematic to feed values that all take wildly different ranges
  - Makes learning more difficult
- Solution:
  - Feature-wise Normalization
    - Subtract the mean of the feature and divide by the standard deviation
    - Feature is centered around 0 and has a unit standard deviation

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

https://colab.research.google.com/drive/1Rj2owHgHtI5HqC15WkmeR2WozXbOhLV0?usp=sharing

Predicting house prices

- Small data, small network

```
model = keras.models.Sequential()
model.add(keras.layers.Input(train_data.shape[1]))
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dense(1))
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.summary()
```

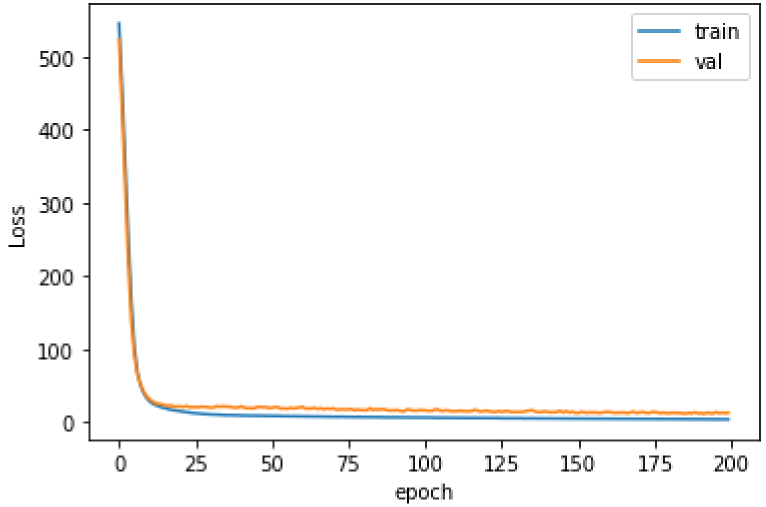
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 64)                896
_________________________________________________________________
dense_4 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 5,121
Trainable params: 5,121
Non-trainable params: 0
```

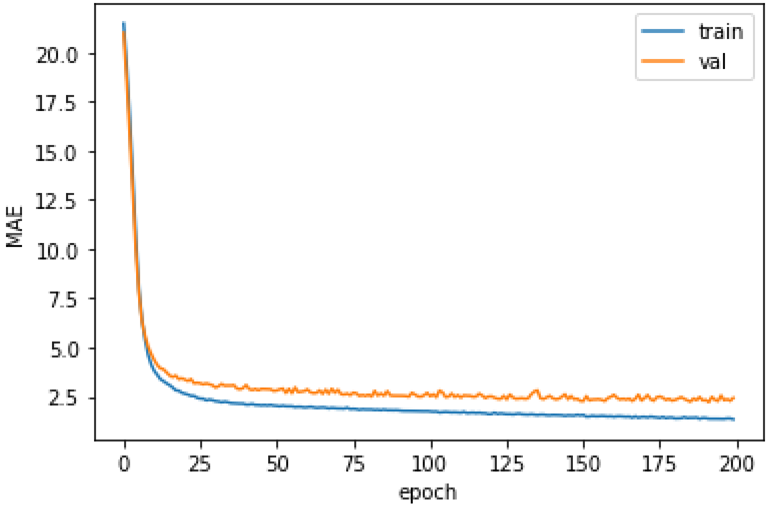
https://colab.research.google.com/drive/1Rj2owHgHtI5HqC15WkmeR2WozXbOhLV0?usp=sharing

Predicting house prices

- Small data, small network

```
history = model.fit(train_data, train_targets,
                    validation_data=(test_data, test_targets),
                    batch_size=32,
                    epochs=200)
```






```
Epoch 200/200
13/13 [==============================] - 0s 5ms/step - loss: 4.4936 - mae: 1.4073 - val_loss: 12.7377 - val_mae: 2.4284
```

Binary classification

- Many tasks require predicting the value of a binary variable y
- A Bernoulli distribution is defined by just a single number
- The neural net needs to predict only $P(y = 1|\boldsymbol{x}) \in [0, 1]$

$$\text{e.g. } P(y = 1|\boldsymbol{x}) = \max\{0, \min\{1, \boldsymbol{w}^T\boldsymbol{h} + b\}\}$$

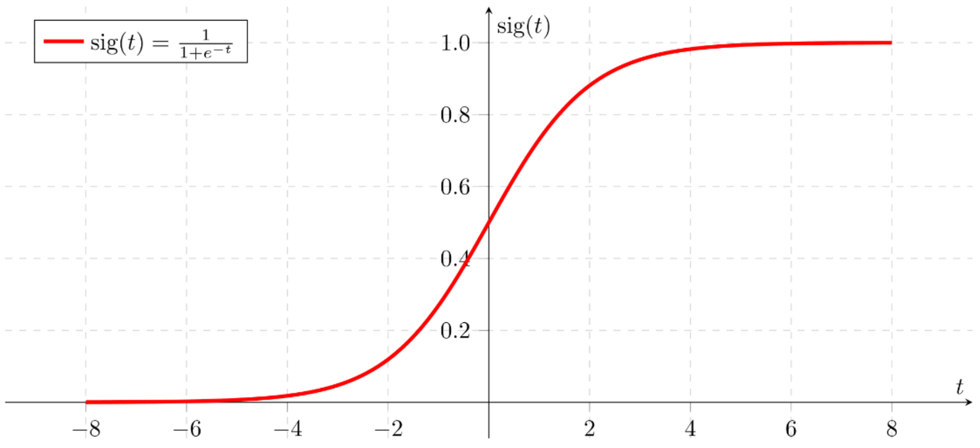
- We would not be able to train it very effectively with gradient descent

	Two Class Classification	
$y \in \{0, 1\}$	**1 or Positive Class**	**0 or Negative Class**
Email	Spam	Not Spam
Tumor	Malignant	Benign
Transaction	Fraudulent	Not Fraudulent

Binary classification

- It is better to use a different approach that ensures there is always a strong gradient whenever the model has the wrong answer
- A sigmoid output unit is defined by $P(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{h} + b)$
- Is MSE a good loss function in this case?

$$J(\boldsymbol{\theta}) = \big(y - \sigma(z)\big)^2, \qquad z = \boldsymbol{w}^T\boldsymbol{h} + b$$



- Example: if $z = 5.6, y = 0$

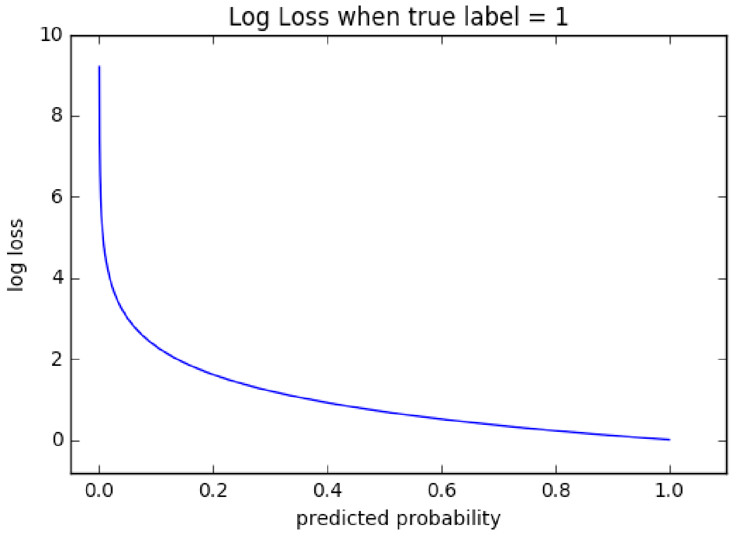
$$\frac{dJ}{dz} = -2\big(y - \sigma(z)\big)\sigma(z)\big(1 - \sigma(z)\big) = 0.0073$$

$$\approx -2(0 - 1)1(1 - 1)$$

Binary classification

- It is better to use a different approach that ensures there is always a strong gradient whenever the model has the wrong answer
- A sigmoid output unit is defined by $P(y=1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{h} + b)$
- Binary cross-entropy loss:

$$J(\boldsymbol{\theta}) = -y\log\sigma(z) - (1-y)\log\big(1-\sigma(z)\big)$$

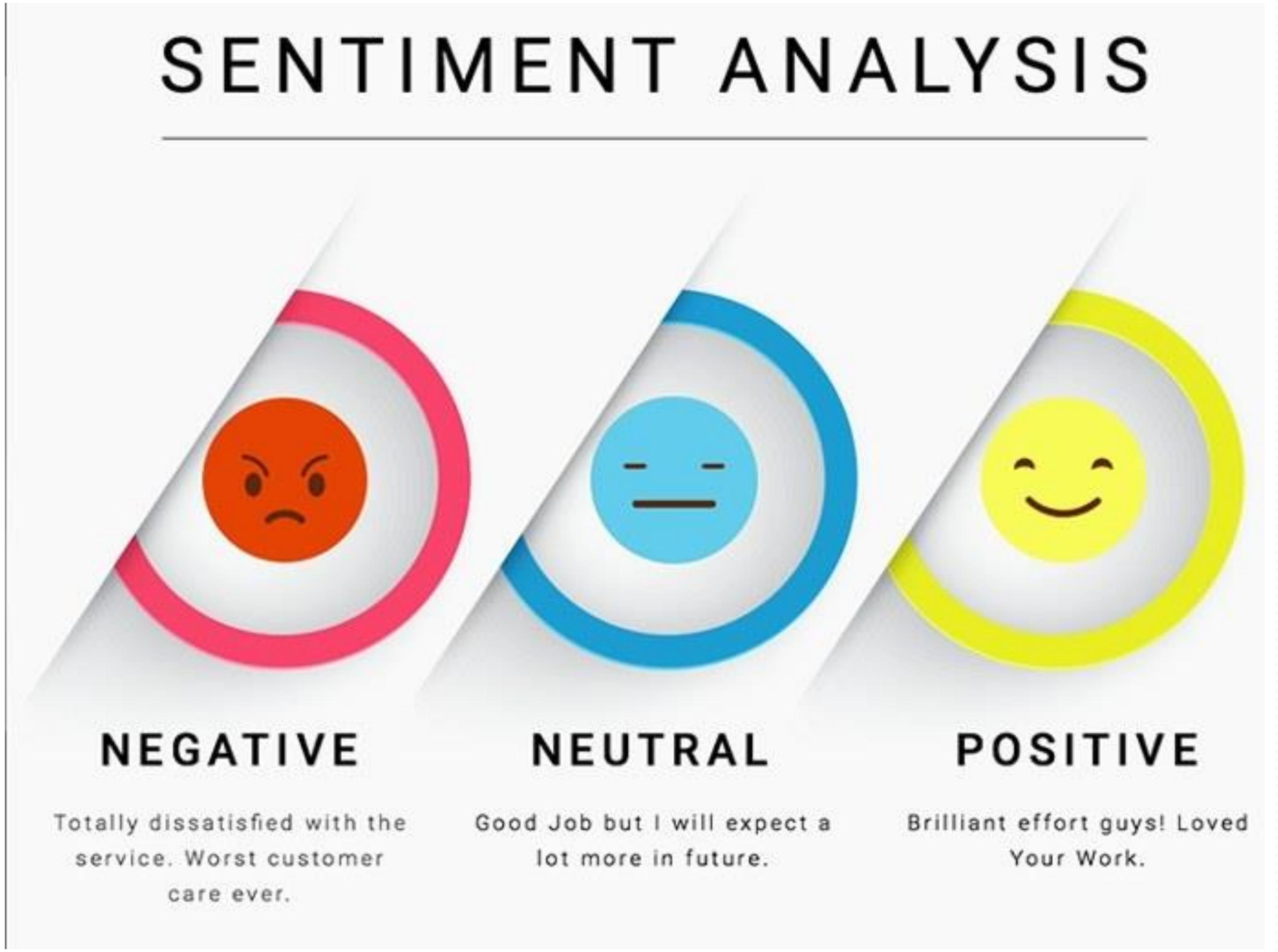


- Example: if $z = 5.6$, $y = 0$

$$\frac{dJ}{dz} = -\frac{-\sigma(z)\big(1-\sigma(z)\big)}{1-\sigma(z)} = \sigma(z) = 0.9963$$

Classifying movie reviews

- IMDB dataset
  - 50,000 highly polarized reviews
  - 25K training, 25K test, each 50-50%
- The reviews (sequences of words) have been turned into sequences of integers, where each integer stands for a specific word in a dictionary



Classifying movie reviews

```
from keras.datasets import imdb
(train_data,train_labels), (test_data,test_labels) = imdb.load_data(num_words=1000)
```

```
d_view = [(v, k) for k, v in imdb.get_word_index().items()]
d_view.sort()
for v, k in d_view[:20]:
    print("{}:\t{}".format(k, v))
for v, k in d_view[-20:]:
    print("{}:\t{}".format(k, v))
```

the:	1	percival:	88565
and:	2	lubricated:	88566
a:	3	heralding:	88567
of:	4	baywatch':	88568
to:	5	odilon:	88569
is:	6	'solve':	88570
br:	7	guard's:	88571
in:	8	nemesis':	88572
it:	9	airsoft:	88573
i:	10	urrrghhh:	88574
this:	11	ev:	88575
that:	12	chicatillo:	88576
was:	13	transacting:	88577
as:	14	sics:	88578
for:	15	wheelers:	88579
with:	16	pipe's:	88580
movie:	17	copywrite:	88581
but:	18	artbox:	88582
film:	19	voorhees':	88583
on:	20	'l':	88584

<START> this film was just brilliant casting location scenery story direction

1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, …

1, 14, 22, 16, 43, 530, 973, 2, 2, 65, 458, 2, …

Classifying movie reviews

```
from keras.datasets import imdb
(train_data,train_labels), (test_data,test_labels) = imdb.load_data(num_words=1000)
```

- train_labels and test_labels are lists of 0s and 1s, where 0 stands for negative and 1 stands for positive
- We have to turn the lists (with different lengths) into tensors

```
np.array([len(d) for d in train_data])
```

```
array([218, 189, 141, ..., 184, 150, 153])
```

  - Using Embedding layer: Same length by padding the sentences
  - One-hot encoding: Vectors of 0’s and 1’s, turning the sequence [3, 5] into a 10,000-dimensional vector that would be all 0s except for indices 3 and 5, which would be 1s

Classifying movie reviews

```
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

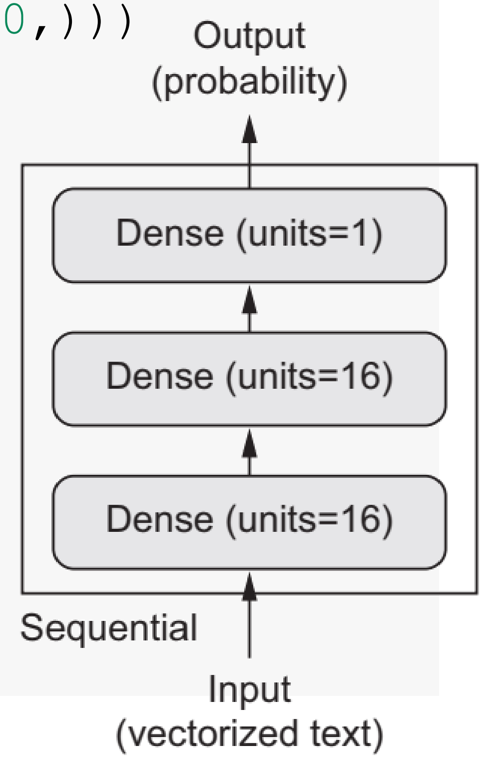
```
print(x_train.shape)
print(x_train[0])
```

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Classifying movie reviews

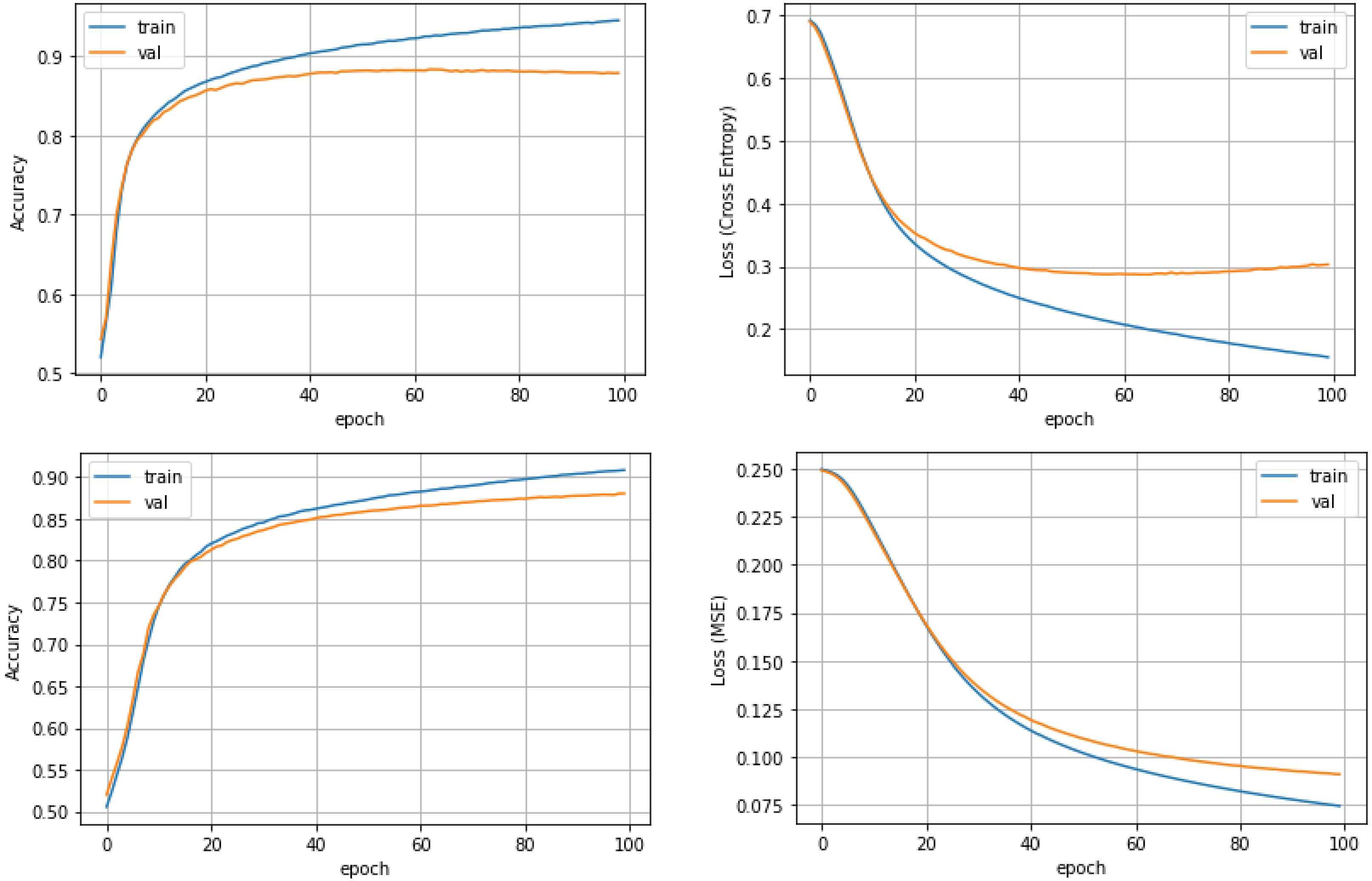
- Building the network

```
model = keras.models.Sequential()
model.add(keras.layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(keras.layers.Dense(16, activation='relu'))
model.add(keras.layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    validation_data=(x_val, y_val),
                    epochs=100,
                    batch_size=512)
```



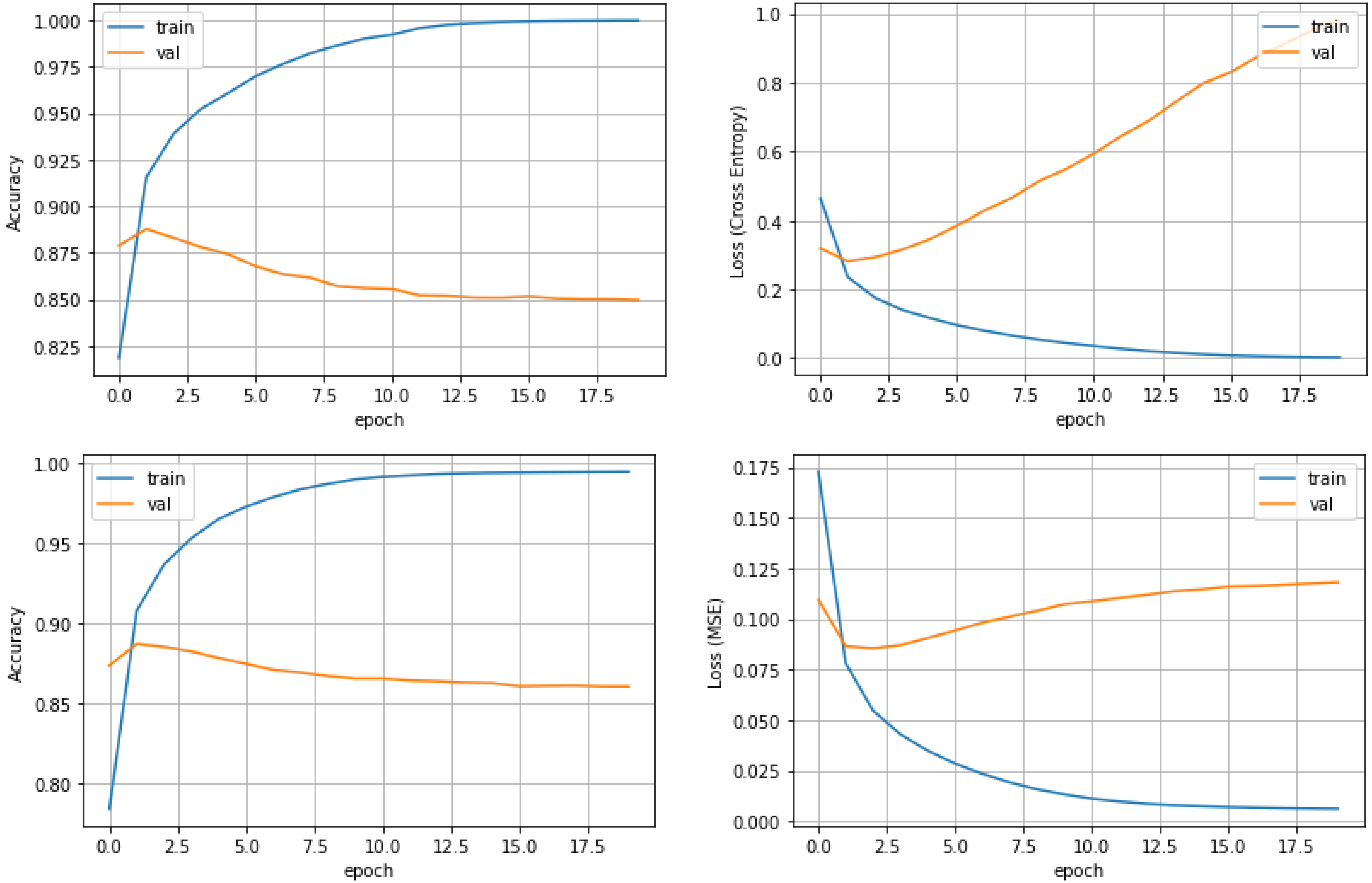
Classifying movie reviews

- SGD optimizer



Classifying movie reviews

- Adam optimizer



Multiclass classification

- Softmax can be seen as a generalization of the sigmoid function to represent a probability distribution over a discrete variable with n possible values
- We now need to produce a vector $\hat{\boldsymbol{y}}$, with $\hat{y}_i = P(y = i|\boldsymbol{x})$
- First, a linear layer predicts unnormalized log probabilities $\boldsymbol{z} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- Training the softmax using cross-entropy

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{C} y_i \log \hat{y}_i$$