Binary classification

- Dataset: 50,000 IMDB reviews, labeled positive (1) or negative (0)
  - Included in Keras, with a 50/50 train-test split
- Each row is one review, with only the 10,000 most frequent words retained
- Each word is replaced by a *word index* (word ID)

```
Encoded review:  [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
Original review:  ? this film was just brilliant casting location scene
ry story
```

Preprocessing

- We can't input lists of categorical value to a neural net, we need to create tensors
- One-hot-encoding:
  - 10000 features, '1.0' if the word occurs
- Word embeddings (word2vec):
  - Map each word to a dense vector that represents it (it's *embedding*)
  - *Embedding* layer: pre-trained layer that looks up the embedding in a dictionary
  - Converts 2D tensor of word indices (zero-padded) to 3D tensor of embeddings
- Let's do One-Hot-Encoding for now. We'll come back to *Embedding* layers.
- Also vectorize the labels: from 0/1 to float
  - Binary classification works with one output node

```
Encoded review:  [1, 14, 22, 16, 43, 530, 973, 1622, 1385,
65]
One-hot-encoded review:  [0. 1. 1. 0. 1. 1. 1. 1. 1. 1.]
Label:  1.0
```

Building the network

- We can solve this problem using a network of *Dense* layers and the *ReLU* activation function.
- How many layers? How many hidden units for layer?
  - Start with 2 layers of 16 hidden units each
  - We'll optimize this soon
- Output layer: single unit with *sigmoid* activation function
  - Close to 1: positive review, close to 0: negative review

Cross-entropy loss

- We've seen *cross-entropy loss* (or *log loss*) over C classes before
    - Measures how similar the actual and predicted probability distributions are
    - Compute cross-entropy $H(y, \hat{y})$ between true y and predicted $\hat{y}$
    - Sum up over all training samples

$$H(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$$

- For binary classification, this simplifies to

$$-\sum_{c=0,1} y_c \log(\hat{y}_c) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

ml

For more control, you can explictly create the optimizer, loss, and metrics:

```
from keras import optimizers
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

Model selection

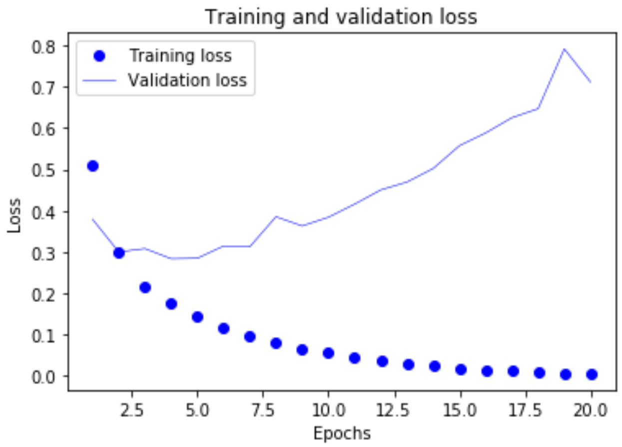
- How many epochs do we need for training?
- Take a validation set of 10,000 samples from the training set
- Train the neural net and track the loss after every iteration on the validation set
  - This is returned as a `History` object by the `fit()` function
- We start with 20 epochs in minibatches of 512 samples

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 4s 243us/step - loss: 0.
5085 - acc: 0.7814 - val_loss: 0.3794 - val_acc: 0.8693
Epoch 2/20
15000/15000 [==============================] - 2s 115us/step - loss: 0.
3005 - acc: 0.9046 - val_loss: 0.3002 - val_acc: 0.8899
Epoch 3/20
15000/15000 [==============================] - 2s 115us/step - loss: 0.
2179 - acc: 0.9285 - val_loss: 0.3082 - val_acc: 0.8715
Epoch 4/20
15000/15000 [==============================] - 2s 142us/step - loss: 0.
1751 - acc: 0.9437 - val_loss: 0.2838 - val_acc: 0.8835
Epoch 5/20
15000/15000 [==============================] - 2s 139us/step - loss: 0.
1427 - acc: 0.9543 - val_loss: 0.2848 - val_acc: 0.8865
Epoch 6/20
15000/15000 [==============================] - 2s 133us/step - loss: 0.
1150 - acc: 0.9652 - val_loss: 0.3146 - val_acc: 0.8774
Epoch 7/20
15000/15000 [==============================] - 2s 144us/step - loss: 0.
0980 - acc: 0.9707 - val_loss: 0.3126 - val_acc: 0.8843
Epoch 8/20
15000/15000 [==============================] - 2s 119us/step - loss: 0.
0807 - acc: 0.9763 - val_loss: 0.3855 - val_acc: 0.8651
Epoch 9/20
15000/15000 [==============================] - 2s 124us/step - loss: 0.
0661 - acc: 0.9821 - val_loss: 0.3632 - val_acc: 0.8779
Epoch 10/20
15000/15000 [==============================] - 2s 118us/step - loss: 0.
0557 - acc: 0.9852 - val_loss: 0.3842 - val_acc: 0.8791
Epoch 11/20
15000/15000 [==============================] - 2s 108us/step - loss: 0.
```

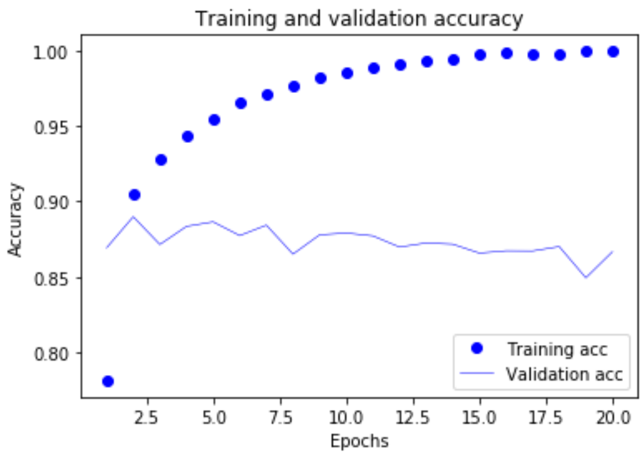
```
0450 - acc: 0.9889 - val_loss: 0.4160 - val_acc: 0.8772
Epoch 12/20
15000/15000 [==============================] - 2s 114us/step - loss: 0.
0385 - acc: 0.9913 - val_loss: 0.4505 - val_acc: 0.8698
Epoch 13/20
15000/15000 [==============================] - 2s 110us/step - loss: 0.
0299 - acc: 0.9930 - val_loss: 0.4696 - val_acc: 0.8725
Epoch 14/20
15000/15000 [==============================] - 2s 111us/step - loss: 0.
0247 - acc: 0.9948 - val_loss: 0.5025 - val_acc: 0.8717
Epoch 15/20
15000/15000 [==============================] - 2s 110us/step - loss: 0.
0174 - acc: 0.9981 - val_loss: 0.5576 - val_acc: 0.8658
Epoch 16/20
15000/15000 [==============================] - 2s 111us/step - loss: 0.
0139 - acc: 0.9983 - val_loss: 0.5885 - val_acc: 0.8673
Epoch 17/20
15000/15000 [==============================] - 2s 130us/step - loss: 0.
0126 - acc: 0.9981 - val_loss: 0.6253 - val_acc: 0.8672
Epoch 18/20
15000/15000 [==============================] - 2s 118us/step - loss: 0.
0106 - acc: 0.9979 - val_loss: 0.6471 - val_acc: 0.8701
Epoch 19/20
15000/15000 [==============================] - 2s 124us/step - loss: 0.
0055 - acc: 0.9996 - val_loss: 0.7920 - val_acc: 0.8494
Epoch 20/20
15000/15000 [==============================] - 2s 114us/step - loss: 0.
0067 - acc: 0.9993 - val_loss: 0.7118 - val_acc: 0.8667
```

We can now retrieve visualize the loss on the validation data

- The training loss keeps decreasing, due to gradient descent
- The validation loss peaks after a few epochs, after which the model starts to overfit



We can also visualize the accuracy, with similar findings



Early stopping

One simple technique to avoid overfitting is to use the validation set to 'tune' the optimal number of epochs

```
Epoch 1/4
25000/25000 [==============================] - 2s 83us/step - loss: 0.2
226 - acc: 0.9457
Epoch 2/4
25000/25000 [==============================] - 2s 78us/step - loss: 0.1
419 - acc: 0.9582
Epoch 3/4
25000/25000 [==============================] - 2s 77us/step - loss: 0.1
122 - acc: 0.9668
Epoch 4/4
25000/25000 [==============================] - 2s 77us/step - loss: 0.0
905 - acc: 0.9714
25000/25000 [==============================] - 4s 162us/step
Loss: 0.5101, Accuracy: 0.8592
```

Predictions

Out of curiosity, let's look at a few predictions:

Review 0: ? please give this one a miss br br ? ? and the rest of the cast rendered terrible performances the show is flat flat flat br br i don't know how michael madison could have allowed this one on his plate he almost seemed to know this wasn't going to work out and his performa nce was quite ? so all you madison fans give this a miss
Predicted positiveness: [0.016]

Review 9: ? this film is where the batman franchise ought to have stop ped though i will ? that the ideas behind batman forever were excellent and could have been easily realised by a competent director as it turne d out this was not to be the case br br apparently warner brothers exec utives were disappointed with how dark this second batman film from tim burton turned out apart from the idiocy of expecting anything else from burton and the conservative ? of their subsequent decision to turn the franchise into an homage to the sixties tv series i fail to understand how batman returns can be considered at all disappointing br br true it is not quite the equal of the first film though it ? all the minor ? of style found in batman a weaker script that ? the ? between not just two but three characters invites ? comparisons to the masterful pairing of keaton and jack nicholson as the joker in the first film yet for all th is it remains a ? dark film true to the way the batman was always meant to be and highly satisfying br br michael keaton returns as the batman and his alter ego bruce wayne with ? max ? christopher walken named in honour of the 1920s german silent actor his partner in crime ? ? the pe nguin danny ? in brilliant makeup reminiscent of laurence ? richard iii and ? kyle the ? michelle pfeiffer whom wayne romances both as himself and as the batman the four principals turn in excellent performances es pecially walken and ? while together keaton and pfeiffer explore the da rker side of double identities br br there are some intriguing concepts in this film about the only weakness i can really point out is a certai n to the script in some places which i think is due mostly to the way t his film is a four ? fight there simply isn't enough time to properly e xplore what's going on br br nevertheless this is a damn good film i hi

Takeaways

- Neural nets require a lot of preprocessing to create tensors
- Dense layers with ReLU activation can solve a wide range of problems
- Binary classification can be donw with a Dense layer with a single unit, sigmoid activation, and binary cross-entropy loss
- Neural nets overfit easily
- Many design choices have an effect on accuracy and overfitting. Try:
    - 1 or 3 hidden layers
    - more or fewer hidden units (e.g. 64)
    - MSE loss instead of binary cross-entropy
    - `tanh` activation instead of `ReLU`

Wrapping Keras models as scikit-learn estimators

- Model selection can be tedious in pure Keras
- We can use all the power of scikit-learn by wrapping Keras models

```
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
clf = KerasClassifier(model)
```

```
Epoch 1/1
16666/16666 [==============================] - 4s 232us/step - loss: 0.
3399 - acc: 0.8594
8334/8334 [==============================] - 1s 169us/step
16666/16666 [==============================] - 1s 70us/step
Epoch 1/1
16667/16667 [==============================] - 3s 181us/step - loss: 0.
3364 - acc: 0.8611
8333/8333 [==============================] - 1s 161us/step
16667/16667 [==============================] - 1s 71us/step
Epoch 1/1
16667/16667 [==============================] - 2s 134us/step - loss: 0.
3379 - acc: 0.8615
8333/8333 [==============================] - 1s 146us/step
16667/16667 [==============================] - 1s 65us/step
Epoch 1/1
16666/16666 [==============================] - 2s 143us/step - loss: 0.
3411 - acc: 0.8642
8334/8334 [==============================] - 1s 155us/step
16666/16666 [==============================] - 1s 84us/step
Epoch 1/1
16667/16667 [==============================] - 2s 134us/step - loss: 0.
3377 - acc: 0.8631
8333/8333 [==============================] - 1s 139us/step
16667/16667 [==============================] - 1s 73us/step
Epoch 1/1
16667/16667 [==============================] - 3s 163us/step - loss: 0.
3468 - acc: 0.8592
8333/8333 [==============================] - 1s 165us/step
16667/16667 [==============================] - 2s 113us/step
Epoch 1/1
16666/16666 [==============================] - 4s 237us/step - loss: 0.
3404 - acc: 0.8587
```

```
Epoch 8/10
16667/16667 [==============================] - 2s 133us/step - loss: 0.
0783 - acc: 0.9743
Epoch 9/10
16667/16667 [==============================] - 2s 113us/step - loss: 0.
0664 - acc: 0.9776
Epoch 10/10
16667/16667 [==============================] - 2s 110us/step - loss: 0.
0549 - acc: 0.9818
8333/8333 [==============================] - 726s 87ms/step
16667/16667 [==============================] - 5s 305us/step
Epoch 1/1
25000/25000 [==============================] - 6s 257us/step - loss: 0.
3218 - acc: 0.8688
```

```
Out[11]: GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=<keras.wrappers.scikit_learn.KerasClassifier object at
0x1c2e90d0f0>,
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'epochs': [1, 5, 10], 'hidden_size': [32, 64, 256]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

Out[12]:

		mean_test_score	mean_train_score
param_epochs	**param_hidden_size**		
1	**32**	0.89	0.94
	64	0.89	0.93
	256	0.89	0.93
5	**32**	0.88	0.97
	64	0.88	0.98
	256	0.88	0.97
10	**32**	0.87	0.99
	64	0.87	0.99
	256	0.87	0.99

Multi-class classification (topic classification)

- Dataset: 11,000 news stories, 46 topics
  - Included in Keras, with a 50/50 train-test split
- Each row is one news story, with only the 10,000 most frequent words retained
- Each word is replaced by a *word index* (word ID)

```
News wire:  ? ? ? said as a result of its december acquisition of space
co it expects earnings per share in 1987 of 1 15 to 1 30 dlrs per share
up from 70 cts in 1986 the company said pretax net should rise to nine
to 10 mln dlrs from six mln dlrs in 1986 and rental operation revenues
to 19 to 22 mln dlrs from 12 5 mln dlrs it said cash flow per share thi
s year should be 2 50 to three dlrs reuter 3
Encoded:  [1, 2, 2, 8, 43, 10, 447, 5, 25, 207, 270, 5, 3095, 111, 16,
369, 186, 90, 67, 7]
Topic:  3
```

Preparing the data

- We have to vectorize the data again (using one-hot-encoding)
- We have to vectorize the labels as well, also using one-hot-encoding
  - We can use Keras' `to_categorical` again
  - This yields a vector of 46 floats (0/1) for every sample

Building the network

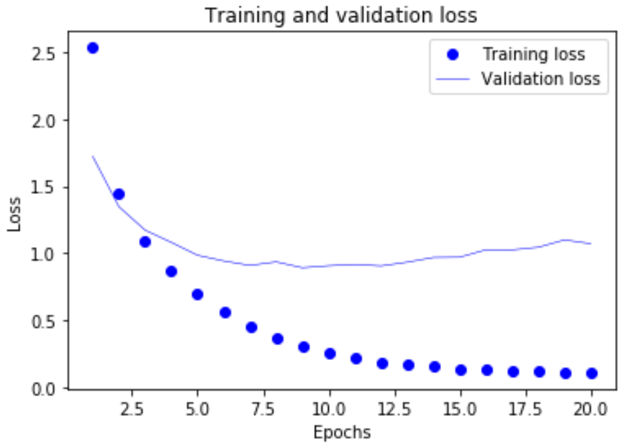
- *Information bottleneck*: Every layer can `drop` some information, which can never be recovered by subsequent layers
- 16 hidden units may be too limited to learn 46 topics, hence we use 64 in each layer
- The output layer now needs 46 units, one for each topic
  - We use `softmax` activation for the output to get probabilities]
- The loss function is now `categorical_crossentropy`

Model selection

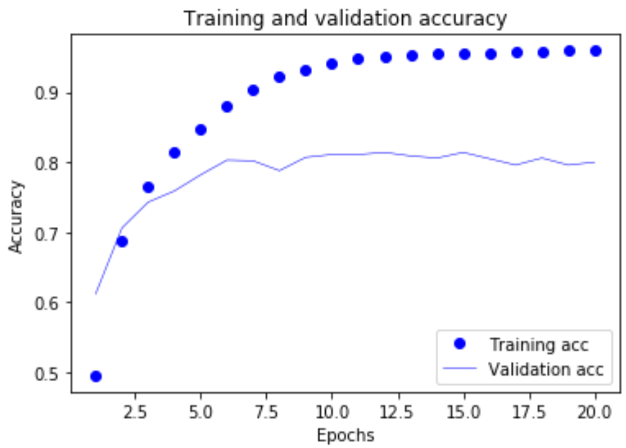
- Take a validation set from the training set
- Fit again with 20 epochs

```
Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 68s 8ms/step - loss: 2.532
2 - acc: 0.4955 - val_loss: 1.7208 - val_acc: 0.6120
Epoch 2/20
7982/7982 [==============================] - 3s 411us/step - loss: 1.44
52 - acc: 0.6877 - val_loss: 1.3458 - val_acc: 0.7060
Epoch 3/20
7982/7982 [==============================] - 2s 209us/step - loss: 1.09
53 - acc: 0.7653 - val_loss: 1.1713 - val_acc: 0.7430
Epoch 4/20
7982/7982 [==============================] - 1s 152us/step - loss: 0.86
98 - acc: 0.8156 - val_loss: 1.0803 - val_acc: 0.7590
Epoch 5/20
7982/7982 [==============================] - 1s 174us/step - loss: 0.70
34 - acc: 0.8479 - val_loss: 0.9843 - val_acc: 0.7820
Epoch 6/20
7982/7982 [==============================] - 1s 166us/step - loss: 0.56
65 - acc: 0.8799 - val_loss: 0.9417 - val_acc: 0.8030
Epoch 7/20
7982/7982 [==============================] - 2s 203us/step - loss: 0.45
80 - acc: 0.9048 - val_loss: 0.9086 - val_acc: 0.8020
Epoch 8/20
7982/7982 [==============================] - 1s 172us/step - loss: 0.36
95 - acc: 0.9230 - val_loss: 0.9350 - val_acc: 0.7880
Epoch 9/20
7982/7982 [==============================] - 1s 157us/step - loss: 0.30
33 - acc: 0.9315 - val_loss: 0.8913 - val_acc: 0.8070
Epoch 10/20
7982/7982 [==============================] - 1s 143us/step - loss: 0.25
38 - acc: 0.9415 - val_loss: 0.9063 - val_acc: 0.8110
Epoch 11/20
7982/7982 [==============================] - 1s 150us/step - loss: 0.21
```

Loss curve:


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

Accuracy curve. Overfitting starts after about 8 epochs



Retrain with early stopping and validate

```
Epoch 1/8
7982/7982 [==============================] - 1s 132us/step - loss: 0.10
54 - acc: 0.9569
Epoch 2/8
7982/7982 [==============================] - 1s 133us/step - loss: 0.10
18 - acc: 0.9580
Epoch 3/8
7982/7982 [==============================] - 1s 129us/step - loss: 0.10
26 - acc: 0.9589
Epoch 4/8
7982/7982 [==============================] - 1s 162us/step - loss: 0.10
29 - acc: 0.9574
Epoch 5/8
7982/7982 [==============================] - 1s 158us/step - loss: 0.09
69 - acc: 0.9587
Epoch 6/8
7982/7982 [==============================] - 1s 140us/step - loss: 0.09
87 - acc: 0.9563
Epoch 7/8
7982/7982 [==============================] - 1s 131us/step - loss: 0.09
58 - acc: 0.9575
Epoch 8/8
7982/7982 [==============================] - 1s 134us/step - loss: 0.09
20 - acc: 0.9585
2246/2246 [==============================] - 0s 196us/step
Loss: 1.3821, Accuracy: 0.7640
```

Information bottleneck

- What happens if we create an information bottleneck on purpose
  - Use only 4 hidden units in the second layer
- Accuracy drops dramatically!
- We are trying to learn 64 separating hyperplanes from a 4-dimensional representation
  - It manages to save a lot of information, but also loses a lot

```
Train on 7982 samples, validate on 1000 samples
Epoch 1/20
7982/7982 [==============================] - 3s 336us/step - loss: 2.46
95 - acc: 0.5199 - val_loss: 1.7777 - val_acc: 0.5980
Epoch 2/20
7982/7982 [==============================] - 2s 206us/step - loss: 1.51
49 - acc: 0.6124 - val_loss: 1.4549 - val_acc: 0.6210
Epoch 3/20
7982/7982 [==============================] - 2s 196us/step - loss: 1.22
49 - acc: 0.6738 - val_loss: 1.3334 - val_acc: 0.6820
Epoch 4/20
7982/7982 [==============================] - 2s 189us/step - loss: 1.05
68 - acc: 0.7453 - val_loss: 1.2732 - val_acc: 0.7100
Epoch 5/20
7982/7982 [==============================] - 2s 203us/step - loss: 0.94
27 - acc: 0.7612 - val_loss: 1.2580 - val_acc: 0.7210
Epoch 6/20
7982/7982 [==============================] - 2s 192us/step - loss: 0.85
41 - acc: 0.7813 - val_loss: 1.2851 - val_acc: 0.7090
Epoch 7/20
7982/7982 [==============================] - 2s 201us/step - loss: 0.78
32 - acc: 0.8007 - val_loss: 1.2925 - val_acc: 0.7290
Epoch 8/20
7982/7982 [==============================] - 2s 204us/step - loss: 0.72
41 - acc: 0.8156 - val_loss: 1.3219 - val_acc: 0.7290
Epoch 9/20
7982/7982 [==============================] - 2s 190us/step - loss: 0.67
16 - acc: 0.8309 - val_loss: 1.3542 - val_acc: 0.7300
Epoch 10/20
7982/7982 [==============================] - 2s 205us/step - loss: 0.62
71 - acc: 0.8366 - val_loss: 1.4113 - val_acc: 0.7250
Epoch 11/20
7982/7982 [==============================] - 2s 232us/step - loss: 0.58
```

```
90 - acc: 0.8429 - val_loss: 1.4427 - val_acc: 0.7290
Epoch 12/20
7982/7982 [==============================] - 2s 230us/step - loss: 0.55
86 - acc: 0.8464 - val_loss: 1.4668 - val_acc: 0.7290
Epoch 13/20
7982/7982 [==============================] - 1s 186us/step - loss: 0.53
07 - acc: 0.8483 - val_loss: 1.5093 - val_acc: 0.7220
Epoch 14/20
7982/7982 [==============================] - 2s 245us/step - loss: 0.50
56 - acc: 0.8540 - val_loss: 1.5416 - val_acc: 0.7220
Epoch 15/20
7982/7982 [==============================] - 2s 220us/step - loss: 0.48
46 - acc: 0.8602 - val_loss: 1.5974 - val_acc: 0.7190
Epoch 16/20
7982/7982 [==============================] - 1s 187us/step - loss: 0.46
51 - acc: 0.8608 - val_loss: 1.5933 - val_acc: 0.7230
Epoch 17/20
7982/7982 [==============================] - 1s 183us/step - loss: 0.44
92 - acc: 0.8641 - val_loss: 1.6393 - val_acc: 0.7160
Epoch 18/20
7982/7982 [==============================] - 2s 198us/step - loss: 0.43
27 - acc: 0.8687 - val_loss: 1.7431 - val_acc: 0.7140
Epoch 19/20
7982/7982 [==============================] - 1s 177us/step - loss: 0.41
71 - acc: 0.8703 - val_loss: 1.7400 - val_acc: 0.7130
Epoch 20/20
7982/7982 [==============================] - 2s 211us/step - loss: 0.40
61 - acc: 0.8779 - val_loss: 1.8052 - val_acc: 0.7140
```

```
Out[22]: <keras.callbacks.History at 0x1c39100ef0>
```

Takeaways

- For a problem with C classes, the final Dense layer needs C units
- Use `softmax` activation and `categorical_crossentropy` loss
- Information bottleneck: when classifying many classes, the hidden layers should be large enough
- Many design choices have an effect on accuracy and overfitting. Try:
    - 1 or 3 hidden layers
    - more or fewer hidden units (e.g. 128)

Regression

- Dataset: 506 examples of houses and sale prices (Boston)
  - Included in Keras, with a 1/5 train-test split
- Each row is one house price, described by numeric properties of the house and neighborhood
- Small dataset, non-normalized features

Preprocessing

- Neural nets work a lot better if we normalize the features first.
- Keras has no built-in support so we have to do this manually (or with scikit-learn)
  - Again, be careful not to look at the test data during normalization

Building the network

- This is a small dataset, so easy to overfit
  - We use 2 hidden layers of 64 units each
- Use smaller batches, more epochs
- Since we want scalar output, the output layer is one unit without activation
- Loss function is Mean Squared Error (bigger penalty)
- Evaluation metric is Mean Abolute Error (more interpretable)
- We will also use cross-validation, so we wrap the model building in a function, so that we can call it multiple times

Cross-validation

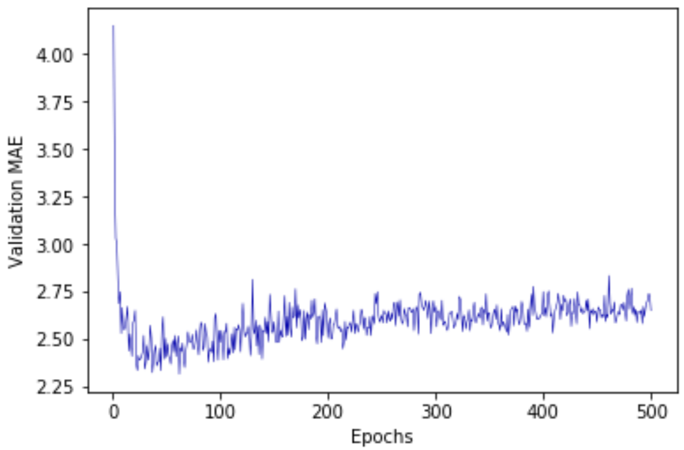
- Keras does not have support for cross-validation
- Luckily we can wrap a Keras model as a scikit-learn estimate
- We can also implement cross-validation ourselves
- Generally speaking, cross-validation is tricky with neural nets
  - Some fold may not converge, or fluctuate on random initialization

```
MAE:  15.93638613861
3859

processing fold
# 0
processing fold
# 1
processing fold
# 2
processing fold
# 3

MAE:  2.496337992720
0844
```

Train for longer and keep track of loss after every epoch

```
processing fold
# 0
processing fold
# 1
processing fold
# 2
processing fold
# 3
```

The model starts overfitting from epoch 80



Retrain with optimized number of epochs

```
102/102 [==============================] - 0s 640us/
step
MAE:  2.825951931523342
```

Takeaways

- Regression is usually done using MSE loss and MAE for evaluation
- Input data should always be scaled (independent from the test set)
- Small datasets:
  - Use cross-validation
  - Use simple (non-deep) networks
  - Smaller batches, more epochs

Regularization: build smaller networks

- The easiest way to avoid overfitting is to use a simpler model
- The number of learnable parameters is called the model *capacity*
- A model with more parameters has a higher *memorization capacity*
  - The entire training set can be `stored` in the weights
  - Learns the mapping from training examples to outputs
- Forcing the model to be small forces it to learn a compressed representation that generalizes better
  - Always a trade-off between too much and too little capacity
- Start with few layers and parameters, incease until you see diminisching returns

Let's try this on our movie review data, with 4 units per layer

The smaller model starts overfitting later than the original one, and it overfits more *slowly*

Regularization: Weight regularization

- As we did many times before, we can also add weight regularization to our loss function
- L1 regularization: leads to *sparse networks* with many weights that are 0
- L2 regularization: leads to many very small weights
  - Also called *weight decay* in neural net literature
- In Keras, add `kernel_regularizer` to every layer

L2 regularized model is much more resistant to overfitting, even though both have the same number of parameters

You can also try L1 loss or both at the same time

```python
from keras import regularizers

# L1 regularization
regularizers.l1(0.001)

# L1 and L2 regularization at the same time
regularizers.l1_l2(l1=0.001, l2=0.001)
```

Regularization: dropout

- One of the most effective and commonly used regularization techniques
- Breakes up accidental non-significant learned patterns
- Randomly set a number of outputs of the layer to 0
- *Dropout rate*: fraction of the outputs that are zeroed-out
  - Usually between 0.2 and 0.5
- Nothing is dropped out at test time, but the output values are scaled down by the dropout rate
  - Balances out that more units are active than during training
- In Keras: add `Dropout` layers between the normal layers

Regularization recap

- Get more training data
- Reduce the capacity of the network
- Add weight regularization
- Add dropout
- Either start with a simple model and add capacity
- Or, start with a complex model and then regularize by adding weight regularization and dropout