ML Homework 3

December 11, 2017

1 ML Homework 3

Ivan Puhachov

1.1 Problem

Make a model which will be able to predict any of given feature (or a set of features) even with additional missing data.

Dataset: Letters - 17 features, 20 000 samples.
Training set - DATA variable, 17 features x 16 000
Testing set - TEST variable, 17 features x 4 000

1.2 Idea

Key idea is to train N neural nets (where N - number of given features in TS), each on full data without missing unit. Apply this neural nets to given query and give a result based on full output.

Example: if in query features 4,6,9 is requested '?' we apply correspondent neural nets independently.

Variables will be encoded by characteristics vector.

Idea of evaluation: explore how can we improve this "general-prediction" system by using **dropout** technique. Besides usual test of performance (accuracy on the test set), compare results of 2 systems: with and without dropout.

1.3 Summary

Based on "Letters" dataset * Chosen neural net architecture [input 282 binary - 100 relu - 60 relu - bunary output] is too complicated for predicting almost all features and leads to **overfitting**. On training they showed around 85% of accuracy, while on testing (even without missing data) accuracy is near 70% * Dropout is very powerfull tecnique to simulate missing variables and to prevent overfitting. Even when testing set includes very few missing values (query marks), prediction is 2% better. If missing values are more common, neural nets with dropout showed better performance (up to 6% increase).

1.4 Results

1.4.1 Accuracy of predicting each feature

When given test set has '?' only in this feature

Feature	lettr	xbox	ybox	width	high	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybr	xy2br	xege	xegvy	yege	yegvx
Accuracy:	90.5	72.1	60.1	70.5	73.8	70.1	65.3	63.8	64.6	62	68.8	67.5	63.5	73.5	69.3	69	65.5
Accuracy with Dropout:	92.7	74.7	61.9	72.1	74.6	70.4	67.8	65.7	65.8	63.5	71.2	69.7	65.8	75.4	71.6	70.3	67.1

1.4.2 Accuracy of prediction on noisy data

Accuracy of prediction 'Letter' feature, when adding 10 (30, 50, 100 etc.) missing values ('?') on each other features randomly.

Frequency	Accuracy (%)	Accuracy with Dropout:
10	90.5	92.2
30	90.1	91.8
50	89.6	91.4
100	88.7	90.5
200	86.5	88.8
400	82.4	85.2
800	73.5	77.7
1000	69.0	73.8

1.5 Realization

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

In [2]:
```
dataframe = pd.read_csv('Letters.csv')
#dataframe = pd.read_csv('anyCSVfile.csv')
DATA, TEST = train_test_split(dataframe, test_size=0.2, random_state=6) #random_state to
DATA.reset_index(drop=True, inplace=True)
TEST.reset_index(drop=True, inplace=True)
```

In [3]:
```
DATA[1:10]
```

Out[3]:
```
  lettr  xbox  ybox  width  high  onpix  xbar  ybar  x2bar  y2bar  xybar  \
1     H     5    10      8     8     10     7     6      6      4      7
2     Q     3     5      3     6      4     8     9      5      1      6
3     M     7    11      8     8      4     7     7     13      2      7
4     R     5    11      7     8      6     6     8      5      6      6
5     Y     2     1      3     1      0     7    10      3      1      7
6     Q     5     9      6     8      3     9     6      9      8      7
7     K     6    11      9     8      7     6     6      1      6      9
8     E     4     8      4     6      2     3     6      6     11      7
```

```
9     A     4     9      7     7      5     6     5      2      3      4

   x2ybr  xy2br  xege  xegvy  yege  yegvx
1      5      8     9      6    10     10
2      7     11     2      9     5      9
3     10      8     9      6     0      8
4      5      7     3      6     6      9
5     12      8     1     11     0      8
6      4      9     3      8     4      8
7      6     10     5      7     5      8
8      7     15     0      8     7      7
9      1      6     5      7     5      4
```

1.5.1 Data transformation

Transform data row. For each feature - transform it by one-hot-encoder or use 0-vector of suitable shape.

How it works: * program extracts each feature range out of dataset DATA (assuming features are categorical) to list in **feature_range** * function **encodeValue** takes feature *value* and *column* (featureNo) and transforms it to characteristics vector of appropriate size, according to corresponding feature range. If it meets new value (which is not present in corresponding feature_range) it returns **0-vector** * function **deencodeValue** takes characteristics vector and returns corresponding feature value * function **transformInput** transforms whole row from dataset to big vector (concatinated characteristics vectors) of length **inputvector_length**

Transformation of training set will be done in training part. It will be done by only using these functions.

```
In [4]: feature_range = np.array([DATA[i].unique().tolist() for i in DATA]) # list of all featur

        inputvector_length = sum([len(feature_range[i]) for i in range(len(feature_range))])

        def encodeValue(value, column):
            """
            create a characteristics vector, encoding value from column in data
            """
            encodedValue = np.zeros(len(feature_range[column]))
            try:
                index = feature_range[column].index(value)
                encodedValue[index]=1
            except:
                pass
            return encodedValue

        def deencodeValue(vector, column):
            """
            deencode characteristics vector given from column
            """
            index = np.argmax(vector)
```

```
        return feature_range[column][index]

    def transformInput(row):
        """
        return a transformed input query even if it has question marks (substitute with 0-ve
        """
        transformedRow = np.array([])
        for i in range(len(row)):
            value = row[i]
            encodedValue = encodeValue(value,i)
            transformedRow = np.concatenate((transformedRow,encodedValue))
            #print(encodedValue)
        return transformedRow
```

Example

```
In [14]: DATA.iloc[0,0]
```

```
Out[14]: 'I'
```

```
In [11]: vector = encodeValue(DATA.iloc[0,0],0)
         print(vector)
```

```
[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.]
```

```
In [13]: deencodeValue(vector,0)
```

```
Out[13]: 'I'
```

```
In [6]: tst = DATA.iloc[100,:].values
        print(transformInput(tst))
```

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  1.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

```
In [7]: print(len(transformInput(tst)))
```

```
282
```

```
In [8]: print(inputvector_length)
```

```
282
```

1.5.2 Creating neural nets

Training 17 neural nets (for dataset Letters - there are 17 features).

Function *buildANNtoPredict* returns a trained neural net, which can predict some particular feature (only one), basing on DATA. If parameter *dropout* is *True*, than it constructs neural net with dropout.

Architecture: inputvector_length binary -> 100 relu -> 60 relu ->output_length softmax

- inputvector_length is constant for each model (16x16 + 26 for "Letters" dataset)
- output_length depends on feature_range of particular feature we predicting
- **dropout** unit lies between 1 and 2 hidden layers

Process of training For feature number X: * create a true output - vector of this feature values from training set DATA and encode it * substitute all values of this feature in dataset DATA as '?'. Now this feature will always be encoded as 0-vector of appropriate length * encode input through characteristics vectors (function transformInput) * train neural net

- Save the result in array

```
In [105]: from keras.models import Sequential
          from keras.layers import Dense
          from keras.optimizers import Adam
          from keras.layers import Dropout
```

```
In [106]: def buildANNtoPredict(featureNo, dropout = False):
              """
              given a No. of column (feature) we want to predict, construct and train neural net
              """
              output_length = len(feature_range[featureNo])
              # Model construction
              model = Sequential()
              model.add(Dense(100, input_dim=inputvector_length, activation='relu'))
              if (dropout):
                  model.add(Dropout(0.2))
              model.add(Dense(60, activation='relu'))
              model.add(Dense(output_length, activation='softmax'))
              adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
```

```
        model.compile(loss='categorical_crossentropy', optimizer=adam,metrics=['accuracy']
        # Creating training and testing sets
        trainLocalCopy = DATA.copy()
        y = trainLocalCopy.iloc[:,featureNo].values
        yEncoded=np.zeros(0)
        for value in y:
            encodedValue = encodeValue(value,featureNo)
            yEncoded = np.append(yEncoded,encodedValue)
        yEncoded.shape = (-1,output_length)
        trainLocalCopy.iloc[:,featureNo] = '?'
        xEncoded = np.array([transformInput(trainLocalCopy.iloc[i,:].values) for i in rang
        # Training
        #(xTrain, xTest, yTrain, yTest) = train_test_split(xEncoded, yEncoded, test_size =
        model.fit(xEncoded, yEncoded, batch_size=32, epochs=20,verbose=2)
        return model
```

Training

```
In [89]: MODELS = [buildANNtoPredict(i) for i in range(DATA.shape[1])]
```

```
Epoch 1/20
1s - loss: 1.6638 - acc: 0.5706
Epoch 2/20
0s - loss: 0.6678 - acc: 0.8126
Epoch 3/20
0s - loss: 0.4912 - acc: 0.8566
Epoch 4/20
0s - loss: 0.3936 - acc: 0.8840
Epoch 5/20
0s - loss: 0.3237 - acc: 0.9051
Epoch 6/20
0s - loss: 0.2759 - acc: 0.9175
Epoch 7/20
0s - loss: 0.2293 - acc: 0.9347
Epoch 8/20
1s - loss: 0.1940 - acc: 0.9428
Epoch 9/20
0s - loss: 0.1643 - acc: 0.9529
Epoch 10/20
0s - loss: 0.1372 - acc: 0.9604
Epoch 11/20
0s - loss: 0.1153 - acc: 0.9686
Epoch 12/20
0s - loss: 0.0956 - acc: 0.9741
Epoch 13/20
0s - loss: 0.0824 - acc: 0.9783
Epoch 14/20
0s - loss: 0.0669 - acc: 0.9836
```

```
Epoch 15/20
0s - loss: 0.0550 - acc: 0.9873
Epoch 16/20
0s - loss: 0.0451 - acc: 0.9908
Epoch 17/20
0s - loss: 0.0413 - acc: 0.9906
Epoch 18/20
0s - loss: 0.0323 - acc: 0.9941
Epoch 19/20
0s - loss: 0.0253 - acc: 0.9961
Epoch 20/20
0s - loss: 0.0223 - acc: 0.9970
Epoch 1/20
1s - loss: 1.2332 - acc: 0.5307
Epoch 2/20
0s - loss: 0.7979 - acc: 0.6661
Epoch 3/20
0s - loss: 0.6903 - acc: 0.7090
Epoch 4/20
0s - loss: 0.6151 - acc: 0.7441
Epoch 5/20
0s - loss: 0.5566 - acc: 0.7688
Epoch 6/20
0s - loss: 0.5071 - acc: 0.7897
Epoch 7/20
0s - loss: 0.4679 - acc: 0.8068
Epoch 8/20
0s - loss: 0.4305 - acc: 0.8234
Epoch 9/20
0s - loss: 0.4012 - acc: 0.8356
Epoch 10/20
0s - loss: 0.3709 - acc: 0.8497
Epoch 11/20
0s - loss: 0.3436 - acc: 0.8616
Epoch 12/20
0s - loss: 0.3209 - acc: 0.8712
Epoch 13/20
0s - loss: 0.2936 - acc: 0.8822
Epoch 14/20
0s - loss: 0.2740 - acc: 0.8926
Epoch 15/20
0s - loss: 0.2582 - acc: 0.9010
Epoch 16/20
0s - loss: 0.2358 - acc: 0.9086
Epoch 17/20
0s - loss: 0.2218 - acc: 0.9159
Epoch 18/20
0s - loss: 0.2045 - acc: 0.9245
```

```
Epoch 19/20
0s - loss: 0.1919 - acc: 0.9274
Epoch 20/20
0s - loss: 0.1812 - acc: 0.9338
Epoch 1/20
1s - loss: 1.5968 - acc: 0.4458
Epoch 2/20
0s - loss: 1.0775 - acc: 0.5722
Epoch 3/20
0s - loss: 0.9520 - acc: 0.6009
Epoch 4/20
0s - loss: 0.8683 - acc: 0.6321
Epoch 5/20
0s - loss: 0.8077 - acc: 0.6533
Epoch 6/20
0s - loss: 0.7564 - acc: 0.6757
Epoch 7/20
0s - loss: 0.7134 - acc: 0.6930
Epoch 8/20
0s - loss: 0.6734 - acc: 0.7091
Epoch 9/20
0s - loss: 0.6433 - acc: 0.7222
Epoch 10/20
0s - loss: 0.6100 - acc: 0.7366
Epoch 11/20
0s - loss: 0.5795 - acc: 0.7542
Epoch 12/20
0s - loss: 0.5555 - acc: 0.7658
Epoch 13/20
0s - loss: 0.5314 - acc: 0.7781
Epoch 14/20
0s - loss: 0.5089 - acc: 0.7879
Epoch 15/20
0s - loss: 0.4889 - acc: 0.7977
Epoch 16/20
0s - loss: 0.4674 - acc: 0.8106
Epoch 17/20
0s - loss: 0.4472 - acc: 0.8164
Epoch 18/20
0s - loss: 0.4295 - acc: 0.8232
Epoch 19/20
0s - loss: 0.4095 - acc: 0.8350
Epoch 20/20
0s - loss: 0.3937 - acc: 0.8412
Epoch 1/20
1s - loss: 1.3147 - acc: 0.4843
Epoch 2/20
0s - loss: 0.8637 - acc: 0.6344
```

```
Epoch 3/20
0s - loss: 0.7475 - acc: 0.6821
Epoch 4/20
0s - loss: 0.6738 - acc: 0.7148
Epoch 5/20
0s - loss: 0.6177 - acc: 0.7389
Epoch 6/20
0s - loss: 0.5712 - acc: 0.7588
Epoch 7/20
0s - loss: 0.5302 - acc: 0.7775
Epoch 8/20
0s - loss: 0.4998 - acc: 0.7926
Epoch 9/20
0s - loss: 0.4702 - acc: 0.8048
Epoch 10/20
0s - loss: 0.4393 - acc: 0.8196
Epoch 11/20
0s - loss: 0.4125 - acc: 0.8310
Epoch 12/20
0s - loss: 0.3885 - acc: 0.8411
Epoch 13/20
0s - loss: 0.3644 - acc: 0.8521
Epoch 14/20
0s - loss: 0.3423 - acc: 0.8634
Epoch 15/20
0s - loss: 0.3235 - acc: 0.8711
Epoch 16/20
0s - loss: 0.3061 - acc: 0.8809
Epoch 17/20
0s - loss: 0.2879 - acc: 0.8864
Epoch 18/20
0s - loss: 0.2705 - acc: 0.8950
Epoch 19/20
0s - loss: 0.2552 - acc: 0.9004
Epoch 20/20
0s - loss: 0.2410 - acc: 0.9073
Epoch 1/20
1s - loss: 1.1950 - acc: 0.5907
Epoch 2/20
0s - loss: 0.7167 - acc: 0.7203
Epoch 3/20
0s - loss: 0.6240 - acc: 0.7425
Epoch 4/20
0s - loss: 0.5578 - acc: 0.7611
Epoch 5/20
0s - loss: 0.5050 - acc: 0.7828
Epoch 6/20
0s - loss: 0.4631 - acc: 0.7996
```

```
Epoch 7/20
0s - loss: 0.4293 - acc: 0.8142
Epoch 8/20
0s - loss: 0.3967 - acc: 0.8314
Epoch 9/20
0s - loss: 0.3750 - acc: 0.8386
Epoch 10/20
0s - loss: 0.3518 - acc: 0.8482
Epoch 11/20
0s - loss: 0.3249 - acc: 0.8633
Epoch 12/20
0s - loss: 0.3061 - acc: 0.8708
Epoch 13/20
0s - loss: 0.2867 - acc: 0.8800
Epoch 14/20
0s - loss: 0.2685 - acc: 0.8905
Epoch 15/20
0s - loss: 0.2531 - acc: 0.8970
Epoch 16/20
0s - loss: 0.2411 - acc: 0.9003
Epoch 17/20
0s - loss: 0.2241 - acc: 0.9101
Epoch 18/20
0s - loss: 0.2108 - acc: 0.9169
Epoch 19/20
0s - loss: 0.2008 - acc: 0.9214
Epoch 20/20
0s - loss: 0.1892 - acc: 0.9254
Epoch 1/20
1s - loss: 1.3592 - acc: 0.4654
Epoch 2/20
0s - loss: 0.8951 - acc: 0.6252
Epoch 3/20
0s - loss: 0.7671 - acc: 0.6765
Epoch 4/20
0s - loss: 0.6867 - acc: 0.7106
Epoch 5/20
0s - loss: 0.6253 - acc: 0.7399
Epoch 6/20
0s - loss: 0.5751 - acc: 0.7618
Epoch 7/20
0s - loss: 0.5319 - acc: 0.7794
Epoch 8/20
0s - loss: 0.4904 - acc: 0.8000
Epoch 9/20
0s - loss: 0.4612 - acc: 0.8142
Epoch 10/20
0s - loss: 0.4310 - acc: 0.8274
```

```
Epoch 11/20
0s - loss: 0.4023 - acc: 0.8389
Epoch 12/20
0s - loss: 0.3798 - acc: 0.8479
Epoch 13/20
0s - loss: 0.3519 - acc: 0.8630
Epoch 14/20
0s - loss: 0.3273 - acc: 0.8741
Epoch 15/20
0s - loss: 0.3065 - acc: 0.8826
Epoch 16/20
0s - loss: 0.2912 - acc: 0.8903
Epoch 17/20
0s - loss: 0.2735 - acc: 0.8972
Epoch 18/20
0s - loss: 0.2517 - acc: 0.9060
Epoch 19/20
0s - loss: 0.2339 - acc: 0.9143
Epoch 20/20
0s - loss: 0.2184 - acc: 0.9211
Epoch 1/20
1s - loss: 1.5290 - acc: 0.4290
Epoch 2/20
0s - loss: 1.0797 - acc: 0.5706
Epoch 3/20
0s - loss: 0.9323 - acc: 0.6221
Epoch 4/20
0s - loss: 0.8300 - acc: 0.6671
Epoch 5/20
0s - loss: 0.7548 - acc: 0.6947
Epoch 6/20
0s - loss: 0.6925 - acc: 0.7204
Epoch 7/20
0s - loss: 0.6437 - acc: 0.7406
Epoch 8/20
0s - loss: 0.5967 - acc: 0.7641
Epoch 9/20
0s - loss: 0.5542 - acc: 0.7797
Epoch 10/20
0s - loss: 0.5213 - acc: 0.7934
Epoch 11/20
0s - loss: 0.4883 - acc: 0.8104
Epoch 12/20
0s - loss: 0.4583 - acc: 0.8216
Epoch 13/20
0s - loss: 0.4332 - acc: 0.8313
Epoch 14/20
0s - loss: 0.4061 - acc: 0.8439
```

```
Epoch 15/20
0s - loss: 0.3859 - acc: 0.8510
Epoch 16/20
0s - loss: 0.3610 - acc: 0.8624
Epoch 17/20
0s - loss: 0.3452 - acc: 0.8686
Epoch 18/20
0s - loss: 0.3265 - acc: 0.8784
Epoch 19/20
0s - loss: 0.3026 - acc: 0.8885
Epoch 20/20
0s - loss: 0.2892 - acc: 0.8940
Epoch 1/20
1s - loss: 1.5592 - acc: 0.4425
Epoch 2/20
0s - loss: 1.0833 - acc: 0.5674
Epoch 3/20
0s - loss: 0.9502 - acc: 0.6230
Epoch 4/20
0s - loss: 0.8485 - acc: 0.6662
Epoch 5/20
0s - loss: 0.7708 - acc: 0.6955
Epoch 6/20
0s - loss: 0.7074 - acc: 0.7209
Epoch 7/20
0s - loss: 0.6527 - acc: 0.7429
Epoch 8/20
0s - loss: 0.6064 - acc: 0.7596
Epoch 9/20
0s - loss: 0.5689 - acc: 0.7751
Epoch 10/20
0s - loss: 0.5298 - acc: 0.7954
Epoch 11/20
0s - loss: 0.4933 - acc: 0.8099
Epoch 12/20
0s - loss: 0.4624 - acc: 0.8221
Epoch 13/20
0s - loss: 0.4321 - acc: 0.8327
Epoch 14/20
0s - loss: 0.4086 - acc: 0.8452
Epoch 15/20
0s - loss: 0.3850 - acc: 0.8534
Epoch 16/20
0s - loss: 0.3624 - acc: 0.8672
Epoch 17/20
0s - loss: 0.3406 - acc: 0.8745
Epoch 18/20
0s - loss: 0.3211 - acc: 0.8827
```

```
Epoch 19/20
0s - loss: 0.3054 - acc: 0.8882
Epoch 20/20
0s - loss: 0.2844 - acc: 0.8956
Epoch 1/20
1s - loss: 1.6859 - acc: 0.3866
Epoch 2/20
0s - loss: 1.1302 - acc: 0.5639
Epoch 3/20
0s - loss: 0.9605 - acc: 0.6234
Epoch 4/20
0s - loss: 0.8447 - acc: 0.6660
Epoch 5/20
0s - loss: 0.7562 - acc: 0.7021
Epoch 6/20
0s - loss: 0.6863 - acc: 0.7293
Epoch 7/20
0s - loss: 0.6284 - acc: 0.7519
Epoch 8/20
0s - loss: 0.5836 - acc: 0.7705
Epoch 9/20
0s - loss: 0.5403 - acc: 0.7907
Epoch 10/20
0s - loss: 0.5034 - acc: 0.8039
Epoch 11/20
0s - loss: 0.4722 - acc: 0.8164
Epoch 12/20
0s - loss: 0.4416 - acc: 0.8305
Epoch 13/20
0s - loss: 0.4156 - acc: 0.8399
Epoch 14/20
0s - loss: 0.3936 - acc: 0.8489
Epoch 15/20
0s - loss: 0.3725 - acc: 0.8588
Epoch 16/20
0s - loss: 0.3549 - acc: 0.8636
Epoch 17/20
0s - loss: 0.3280 - acc: 0.8796
Epoch 18/20
0s - loss: 0.3096 - acc: 0.8849
Epoch 19/20
0s - loss: 0.2938 - acc: 0.8916
Epoch 20/20
0s - loss: 0.2762 - acc: 0.8986
Epoch 1/20
1s - loss: 1.7380 - acc: 0.3659
Epoch 2/20
0s - loss: 1.1894 - acc: 0.5331
```

```
Epoch 3/20
0s - loss: 1.0209 - acc: 0.5954
Epoch 4/20
0s - loss: 0.9063 - acc: 0.6362
Epoch 5/20
0s - loss: 0.8252 - acc: 0.6753
Epoch 6/20
0s - loss: 0.7550 - acc: 0.6973
Epoch 7/20
0s - loss: 0.6980 - acc: 0.7238
Epoch 8/20
0s - loss: 0.6517 - acc: 0.7403
Epoch 9/20
0s - loss: 0.6088 - acc: 0.7576
Epoch 10/20
0s - loss: 0.5720 - acc: 0.7720
Epoch 11/20
0s - loss: 0.5384 - acc: 0.7883
Epoch 12/20
0s - loss: 0.5067 - acc: 0.8030
Epoch 13/20
0s - loss: 0.4757 - acc: 0.8170
Epoch 14/20
0s - loss: 0.4533 - acc: 0.8234
Epoch 15/20
0s - loss: 0.4255 - acc: 0.8373
Epoch 16/20
0s - loss: 0.4065 - acc: 0.8446
Epoch 17/20
0s - loss: 0.3874 - acc: 0.8505
Epoch 18/20
0s - loss: 0.3631 - acc: 0.8648
Epoch 19/20
0s - loss: 0.3463 - acc: 0.8704
Epoch 20/20
0s - loss: 0.3263 - acc: 0.8785
Epoch 1/20
1s - loss: 1.6036 - acc: 0.4420
Epoch 2/20
0s - loss: 1.0804 - acc: 0.5976
Epoch 3/20
0s - loss: 0.9170 - acc: 0.6494
Epoch 4/20
0s - loss: 0.8075 - acc: 0.6885
Epoch 5/20
0s - loss: 0.7232 - acc: 0.7192
Epoch 6/20
0s - loss: 0.6556 - acc: 0.7466
```

```
Epoch 7/20
0s - loss: 0.5957 - acc: 0.7705
Epoch 8/20
0s - loss: 0.5461 - acc: 0.7899
Epoch 9/20
0s - loss: 0.5052 - acc: 0.8057
Epoch 10/20
0s - loss: 0.4676 - acc: 0.8246
Epoch 11/20
0s - loss: 0.4391 - acc: 0.8357
Epoch 12/20
0s - loss: 0.4074 - acc: 0.8470
Epoch 13/20
0s - loss: 0.3796 - acc: 0.8597
Epoch 14/20
0s - loss: 0.3543 - acc: 0.8654
Epoch 15/20
0s - loss: 0.3322 - acc: 0.8774
Epoch 16/20
0s - loss: 0.3116 - acc: 0.8884
Epoch 17/20
0s - loss: 0.2886 - acc: 0.8969
Epoch 18/20
0s - loss: 0.2743 - acc: 0.9009
Epoch 19/20
0s - loss: 0.2560 - acc: 0.9089
Epoch 20/20
0s - loss: 0.2376 - acc: 0.9134
Epoch 1/20
1s - loss: 1.5127 - acc: 0.4598
Epoch 2/20
0s - loss: 1.0493 - acc: 0.5901
Epoch 3/20
0s - loss: 0.9226 - acc: 0.6371
Epoch 4/20
0s - loss: 0.8261 - acc: 0.6741
Epoch 5/20
0s - loss: 0.7511 - acc: 0.7030
Epoch 6/20
0s - loss: 0.6800 - acc: 0.7338
Epoch 7/20
0s - loss: 0.6279 - acc: 0.7541
Epoch 8/20
0s - loss: 0.5812 - acc: 0.7765
Epoch 9/20
0s - loss: 0.5411 - acc: 0.7908
Epoch 10/20
0s - loss: 0.5074 - acc: 0.8033
```

```
Epoch 11/20
0s - loss: 0.4720 - acc: 0.8201
Epoch 12/20
0s - loss: 0.4462 - acc: 0.8300
Epoch 13/20
0s - loss: 0.4155 - acc: 0.8395
Epoch 14/20
0s - loss: 0.3895 - acc: 0.8553
Epoch 15/20
0s - loss: 0.3692 - acc: 0.8635
Epoch 16/20
0s - loss: 0.3440 - acc: 0.8731
Epoch 17/20
0s - loss: 0.3223 - acc: 0.8826
Epoch 18/20
0s - loss: 0.3063 - acc: 0.8876
Epoch 19/20
0s - loss: 0.2888 - acc: 0.8927
Epoch 20/20
0s - loss: 0.2702 - acc: 0.9002
Epoch 1/20
1s - loss: 1.5736 - acc: 0.4437
Epoch 2/20
0s - loss: 1.1313 - acc: 0.5690
Epoch 3/20
0s - loss: 0.9798 - acc: 0.6239
Epoch 4/20
0s - loss: 0.8706 - acc: 0.6608
Epoch 5/20
0s - loss: 0.7896 - acc: 0.6906
Epoch 6/20
0s - loss: 0.7232 - acc: 0.7159
Epoch 7/20
0s - loss: 0.6701 - acc: 0.7401
Epoch 8/20
0s - loss: 0.6249 - acc: 0.7538
Epoch 9/20
0s - loss: 0.5832 - acc: 0.7716
Epoch 10/20
0s - loss: 0.5467 - acc: 0.7917
Epoch 11/20
0s - loss: 0.5113 - acc: 0.7984
Epoch 12/20
0s - loss: 0.4822 - acc: 0.8151
Epoch 13/20
0s - loss: 0.4514 - acc: 0.8310
Epoch 14/20
0s - loss: 0.4274 - acc: 0.8411
```

```
Epoch 15/20
0s - loss: 0.3990 - acc: 0.8511
Epoch 16/20
0s - loss: 0.3811 - acc: 0.8589
Epoch 17/20
0s - loss: 0.3597 - acc: 0.8678
Epoch 18/20
0s - loss: 0.3369 - acc: 0.8764
Epoch 19/20
0s - loss: 0.3241 - acc: 0.8795
Epoch 20/20
0s - loss: 0.3014 - acc: 0.8906
Epoch 1/20
1s - loss: 1.2949 - acc: 0.5506
Epoch 2/20
0s - loss: 0.8315 - acc: 0.6911
Epoch 3/20
0s - loss: 0.7341 - acc: 0.7232
Epoch 4/20
0s - loss: 0.6661 - acc: 0.7499
Epoch 5/20
0s - loss: 0.6093 - acc: 0.7696
Epoch 6/20
0s - loss: 0.5552 - acc: 0.7914
Epoch 7/20
0s - loss: 0.5111 - acc: 0.8095
Epoch 8/20
0s - loss: 0.4725 - acc: 0.8224
Epoch 9/20
0s - loss: 0.4365 - acc: 0.8409
Epoch 10/20
0s - loss: 0.4042 - acc: 0.8498
Epoch 11/20
0s - loss: 0.3730 - acc: 0.8651
Epoch 12/20
0s - loss: 0.3440 - acc: 0.8723
Epoch 13/20
0s - loss: 0.3201 - acc: 0.8832
Epoch 14/20
0s - loss: 0.2969 - acc: 0.8938
Epoch 15/20
0s - loss: 0.2774 - acc: 0.8996
Epoch 16/20
0s - loss: 0.2550 - acc: 0.9115
Epoch 17/20
0s - loss: 0.2316 - acc: 0.9162
Epoch 18/20
0s - loss: 0.2163 - acc: 0.9261
```

```
Epoch 19/20
0s - loss: 0.2025 - acc: 0.9303
Epoch 20/20
0s - loss: 0.1872 - acc: 0.9369
Epoch 1/20
1s - loss: 1.2273 - acc: 0.5681
Epoch 2/20
0s - loss: 0.8919 - acc: 0.6627
Epoch 3/20
0s - loss: 0.7904 - acc: 0.6957
Epoch 4/20
0s - loss: 0.7131 - acc: 0.7273
Epoch 5/20
0s - loss: 0.6473 - acc: 0.7502
Epoch 6/20
0s - loss: 0.5918 - acc: 0.7728
Epoch 7/20
0s - loss: 0.5446 - acc: 0.7914
Epoch 8/20
0s - loss: 0.4999 - acc: 0.8064
Epoch 9/20
0s - loss: 0.4571 - acc: 0.8251
Epoch 10/20
0s - loss: 0.4246 - acc: 0.8400
Epoch 11/20
0s - loss: 0.3899 - acc: 0.8539
Epoch 12/20
0s - loss: 0.3622 - acc: 0.8680
Epoch 13/20
0s - loss: 0.3354 - acc: 0.8756
Epoch 14/20
0s - loss: 0.3051 - acc: 0.8883
Epoch 15/20
0s - loss: 0.2850 - acc: 0.8964
Epoch 16/20
0s - loss: 0.2638 - acc: 0.9072
Epoch 17/20
0s - loss: 0.2425 - acc: 0.9146
Epoch 18/20
0s - loss: 0.2244 - acc: 0.9226
Epoch 19/20
0s - loss: 0.2098 - acc: 0.9247
Epoch 20/20
0s - loss: 0.1889 - acc: 0.9340
Epoch 1/20
1s - loss: 1.4737 - acc: 0.4701
Epoch 2/20
0s - loss: 0.9710 - acc: 0.6368
```

```
Epoch 3/20
0s - loss: 0.8356 - acc: 0.6851
Epoch 4/20
0s - loss: 0.7389 - acc: 0.7216
Epoch 5/20
0s - loss: 0.6618 - acc: 0.7485
Epoch 6/20
0s - loss: 0.6007 - acc: 0.7699
Epoch 7/20
0s - loss: 0.5529 - acc: 0.7899
Epoch 8/20
0s - loss: 0.5096 - acc: 0.8037
Epoch 9/20
0s - loss: 0.4708 - acc: 0.8234
Epoch 10/20
0s - loss: 0.4383 - acc: 0.8339
Epoch 11/20
0s - loss: 0.4080 - acc: 0.8469
Epoch 12/20
0s - loss: 0.3773 - acc: 0.8596
Epoch 13/20
0s - loss: 0.3482 - acc: 0.8716
Epoch 14/20
0s - loss: 0.3258 - acc: 0.8816
Epoch 15/20
0s - loss: 0.3015 - acc: 0.8912
Epoch 16/20
0s - loss: 0.2824 - acc: 0.8967
Epoch 17/20
0s - loss: 0.2623 - acc: 0.9089
Epoch 18/20
0s - loss: 0.2411 - acc: 0.9154
Epoch 19/20
0s - loss: 0.2299 - acc: 0.9192
Epoch 20/20
0s - loss: 0.2115 - acc: 0.9264
Epoch 1/20
1s - loss: 1.4431 - acc: 0.4815
Epoch 2/20
0s - loss: 1.0864 - acc: 0.5913
Epoch 3/20
0s - loss: 0.9476 - acc: 0.6392
Epoch 4/20
0s - loss: 0.8440 - acc: 0.6808
Epoch 5/20
0s - loss: 0.7614 - acc: 0.7041
Epoch 6/20
0s - loss: 0.6949 - acc: 0.7344
```

```
Epoch 7/20
0s - loss: 0.6376 - acc: 0.7509
Epoch 8/20
0s - loss: 0.5884 - acc: 0.7713
Epoch 9/20
0s - loss: 0.5453 - acc: 0.7885
Epoch 10/20
0s - loss: 0.5047 - acc: 0.8045
Epoch 11/20
0s - loss: 0.4715 - acc: 0.8187
Epoch 12/20
0s - loss: 0.4353 - acc: 0.8365
Epoch 13/20
0s - loss: 0.4041 - acc: 0.8459
Epoch 14/20
0s - loss: 0.3751 - acc: 0.8609
Epoch 15/20
0s - loss: 0.3551 - acc: 0.8686
Epoch 16/20
0s - loss: 0.3293 - acc: 0.8791
Epoch 17/20
0s - loss: 0.3044 - acc: 0.8892
Epoch 18/20
0s - loss: 0.2885 - acc: 0.8938
Epoch 19/20
0s - loss: 0.2650 - acc: 0.9080
Epoch 20/20
0s - loss: 0.2496 - acc: 0.9120
```

```
In [107]: MODELSDROPOUT = [buildANNtoPredict(i, dropout=True) for i in range(DATA.shape[1])]
```

```
Epoch 1/20
1s - loss: 1.8503 - acc: 0.5118
Epoch 2/20
0s - loss: 0.8329 - acc: 0.7618
Epoch 3/20
0s - loss: 0.6346 - acc: 0.8168
Epoch 4/20
0s - loss: 0.5261 - acc: 0.8441
Epoch 5/20
0s - loss: 0.4422 - acc: 0.8689
Epoch 6/20
0s - loss: 0.3931 - acc: 0.8794
Epoch 7/20
0s - loss: 0.3440 - acc: 0.8942
Epoch 8/20
0s - loss: 0.3025 - acc: 0.9052
```

```
Epoch 9/20
0s - loss: 0.2766 - acc: 0.9136
Epoch 10/20
0s - loss: 0.2508 - acc: 0.9216
Epoch 11/20
0s - loss: 0.2278 - acc: 0.9264
Epoch 12/20
0s - loss: 0.2070 - acc: 0.9341
Epoch 13/20
0s - loss: 0.1942 - acc: 0.9381
Epoch 14/20
0s - loss: 0.1843 - acc: 0.9392
Epoch 15/20
0s - loss: 0.1651 - acc: 0.9461
Epoch 16/20
0s - loss: 0.1599 - acc: 0.9484
Epoch 17/20
0s - loss: 0.1477 - acc: 0.9499
Epoch 18/20
0s - loss: 0.1359 - acc: 0.9545
Epoch 19/20
0s - loss: 0.1338 - acc: 0.9558
Epoch 20/20
0s - loss: 0.1194 - acc: 0.9613
Epoch 1/20
1s - loss: 1.2698 - acc: 0.5112
Epoch 2/20
0s - loss: 0.8578 - acc: 0.6431
Epoch 3/20
0s - loss: 0.7553 - acc: 0.6851
Epoch 4/20
0s - loss: 0.6867 - acc: 0.7120
Epoch 5/20
0s - loss: 0.6376 - acc: 0.7322
Epoch 6/20
0s - loss: 0.5940 - acc: 0.7528
Epoch 7/20
0s - loss: 0.5629 - acc: 0.7611
Epoch 8/20
0s - loss: 0.5342 - acc: 0.7767
Epoch 9/20
0s - loss: 0.5067 - acc: 0.7898
Epoch 10/20
0s - loss: 0.4851 - acc: 0.7963
Epoch 11/20
0s - loss: 0.4609 - acc: 0.8072
Epoch 12/20
0s - loss: 0.4463 - acc: 0.8134
```

```
Epoch 13/20
0s - loss: 0.4379 - acc: 0.8147
Epoch 14/20
0s - loss: 0.4139 - acc: 0.8274
Epoch 15/20
0s - loss: 0.4007 - acc: 0.8346
Epoch 16/20
0s - loss: 0.3898 - acc: 0.8389
Epoch 17/20
0s - loss: 0.3797 - acc: 0.8403
Epoch 18/20
0s - loss: 0.3654 - acc: 0.8510
Epoch 19/20
0s - loss: 0.3544 - acc: 0.8532
Epoch 20/20
0s - loss: 0.3582 - acc: 0.8528
Epoch 1/20
1s - loss: 1.6639 - acc: 0.4299
Epoch 2/20
0s - loss: 1.1548 - acc: 0.5508
Epoch 3/20
0s - loss: 1.0364 - acc: 0.5763
Epoch 4/20
0s - loss: 0.9605 - acc: 0.5984
Epoch 5/20
0s - loss: 0.9022 - acc: 0.6193
Epoch 6/20
0s - loss: 0.8557 - acc: 0.6369
Epoch 7/20
0s - loss: 0.8203 - acc: 0.6442
Epoch 8/20
0s - loss: 0.7861 - acc: 0.6592
Epoch 9/20
0s - loss: 0.7607 - acc: 0.6734
Epoch 10/20
0s - loss: 0.7345 - acc: 0.6798
Epoch 11/20
0s - loss: 0.7135 - acc: 0.6912
Epoch 12/20
0s - loss: 0.6892 - acc: 0.7049
Epoch 13/20
0s - loss: 0.6771 - acc: 0.7079
Epoch 14/20
0s - loss: 0.6518 - acc: 0.7177
Epoch 15/20
0s - loss: 0.6488 - acc: 0.7169
Epoch 16/20
0s - loss: 0.6253 - acc: 0.7333
```

```
Epoch 17/20
0s - loss: 0.6124 - acc: 0.7355
Epoch 18/20
0s - loss: 0.6043 - acc: 0.7401
Epoch 19/20
0s - loss: 0.5919 - acc: 0.7469
Epoch 20/20
0s - loss: 0.5748 - acc: 0.7534
Epoch 1/20
1s - loss: 1.3595 - acc: 0.4607
Epoch 2/20
0s - loss: 0.9387 - acc: 0.6038
Epoch 3/20
0s - loss: 0.8241 - acc: 0.6552
Epoch 4/20
0s - loss: 0.7532 - acc: 0.6788
Epoch 5/20
0s - loss: 0.6972 - acc: 0.7037
Epoch 6/20
0s - loss: 0.6617 - acc: 0.7186
Epoch 7/20
0s - loss: 0.6246 - acc: 0.7354
Epoch 8/20
0s - loss: 0.5933 - acc: 0.7462
Epoch 9/20
0s - loss: 0.5732 - acc: 0.7538
Epoch 10/20
0s - loss: 0.5454 - acc: 0.7687
Epoch 11/20
0s - loss: 0.5344 - acc: 0.7727
Epoch 12/20
0s - loss: 0.5164 - acc: 0.7829
Epoch 13/20
0s - loss: 0.4959 - acc: 0.7881
Epoch 14/20
0s - loss: 0.4844 - acc: 0.7953
Epoch 15/20
0s - loss: 0.4701 - acc: 0.8007
Epoch 16/20
0s - loss: 0.4578 - acc: 0.8065
Epoch 17/20
0s - loss: 0.4474 - acc: 0.8109
Epoch 18/20
0s - loss: 0.4357 - acc: 0.8159
Epoch 19/20
0s - loss: 0.4310 - acc: 0.8179
Epoch 20/20
0s - loss: 0.4196 - acc: 0.8249
```

```
Epoch 1/20
1s - loss: 1.2690 - acc: 0.5716
Epoch 2/20
0s - loss: 0.7945 - acc: 0.7004
Epoch 3/20
0s - loss: 0.6823 - acc: 0.7239
Epoch 4/20
0s - loss: 0.6169 - acc: 0.7432
Epoch 5/20
0s - loss: 0.5694 - acc: 0.7606
Epoch 6/20
0s - loss: 0.5357 - acc: 0.7718
Epoch 7/20
0s - loss: 0.4970 - acc: 0.7857
Epoch 8/20
0s - loss: 0.4705 - acc: 0.7974
Epoch 9/20
0s - loss: 0.4501 - acc: 0.8048
Epoch 10/20
0s - loss: 0.4301 - acc: 0.8160
Epoch 11/20
0s - loss: 0.4188 - acc: 0.8237
Epoch 12/20
0s - loss: 0.4025 - acc: 0.8269
Epoch 13/20
0s - loss: 0.3807 - acc: 0.8382
Epoch 14/20
0s - loss: 0.3723 - acc: 0.8404
Epoch 15/20
0s - loss: 0.3582 - acc: 0.8438
Epoch 16/20
0s - loss: 0.3456 - acc: 0.8518
Epoch 17/20
0s - loss: 0.3372 - acc: 0.8561
Epoch 18/20
0s - loss: 0.3353 - acc: 0.8569
Epoch 19/20
0s - loss: 0.3228 - acc: 0.8629
Epoch 20/20
0s - loss: 0.3158 - acc: 0.8631
Epoch 1/20
1s - loss: 1.4140 - acc: 0.4430
Epoch 2/20
0s - loss: 0.9792 - acc: 0.5824
Epoch 3/20
0s - loss: 0.8657 - acc: 0.6284
Epoch 4/20
0s - loss: 0.7879 - acc: 0.6669
```

```
Epoch 5/20
0s - loss: 0.7283 - acc: 0.6895
Epoch 6/20
0s - loss: 0.6888 - acc: 0.7099
Epoch 7/20
0s - loss: 0.6556 - acc: 0.7224
Epoch 8/20
0s - loss: 0.6300 - acc: 0.7352
Epoch 9/20
0s - loss: 0.6004 - acc: 0.7478
Epoch 10/20
0s - loss: 0.5696 - acc: 0.7610
Epoch 11/20
0s - loss: 0.5627 - acc: 0.7590
Epoch 12/20
0s - loss: 0.5357 - acc: 0.7767
Epoch 13/20
0s - loss: 0.5199 - acc: 0.7823
Epoch 14/20
0s - loss: 0.5041 - acc: 0.7924
Epoch 15/20
0s - loss: 0.4875 - acc: 0.7957
Epoch 16/20
0s - loss: 0.4780 - acc: 0.8013
Epoch 17/20
0s - loss: 0.4607 - acc: 0.8108
Epoch 18/20
0s - loss: 0.4563 - acc: 0.8071
Epoch 19/20
0s - loss: 0.4434 - acc: 0.8165
Epoch 20/20
0s - loss: 0.4349 - acc: 0.8193
Epoch 1/20
1s - loss: 1.6137 - acc: 0.4037
Epoch 2/20
0s - loss: 1.1697 - acc: 0.5351
Epoch 3/20
0s - loss: 1.0298 - acc: 0.5817
Epoch 4/20
0s - loss: 0.9352 - acc: 0.6202
Epoch 5/20
0s - loss: 0.8755 - acc: 0.6403
Epoch 6/20
0s - loss: 0.8142 - acc: 0.6624
Epoch 7/20
0s - loss: 0.7745 - acc: 0.6802
Epoch 8/20
0s - loss: 0.7358 - acc: 0.6971
```

```
Epoch 9/20
0s - loss: 0.7147 - acc: 0.7051
Epoch 10/20
0s - loss: 0.6913 - acc: 0.7166
Epoch 11/20
0s - loss: 0.6598 - acc: 0.7301
Epoch 12/20
0s - loss: 0.6389 - acc: 0.7363
Epoch 13/20
0s - loss: 0.6286 - acc: 0.7431
Epoch 14/20
0s - loss: 0.6077 - acc: 0.7519
Epoch 15/20
0s - loss: 0.5899 - acc: 0.7583
Epoch 16/20
0s - loss: 0.5739 - acc: 0.7667
Epoch 17/20
0s - loss: 0.5621 - acc: 0.7680
Epoch 18/20
0s - loss: 0.5460 - acc: 0.7741
Epoch 19/20
0s - loss: 0.5399 - acc: 0.7801
Epoch 20/20
0s - loss: 0.5315 - acc: 0.7811
Epoch 1/20
1s - loss: 1.5774 - acc: 0.4310
Epoch 2/20
0s - loss: 1.1537 - acc: 0.5451
Epoch 3/20
0s - loss: 1.0375 - acc: 0.5842
Epoch 4/20
0s - loss: 0.9577 - acc: 0.6121
Epoch 5/20
0s - loss: 0.8937 - acc: 0.6394
Epoch 6/20
0s - loss: 0.8430 - acc: 0.6588
Epoch 7/20
0s - loss: 0.7983 - acc: 0.6787
Epoch 8/20
0s - loss: 0.7659 - acc: 0.6931
Epoch 9/20
0s - loss: 0.7364 - acc: 0.7011
Epoch 10/20
0s - loss: 0.7039 - acc: 0.7186
Epoch 11/20
0s - loss: 0.6818 - acc: 0.7228
Epoch 12/20
0s - loss: 0.6555 - acc: 0.7328
```

```
Epoch 13/20
0s - loss: 0.6398 - acc: 0.7416
Epoch 14/20
0s - loss: 0.6190 - acc: 0.7496
Epoch 15/20
0s - loss: 0.6058 - acc: 0.7568
Epoch 16/20
0s - loss: 0.5978 - acc: 0.7604
Epoch 17/20
0s - loss: 0.5762 - acc: 0.7681
Epoch 18/20
0s - loss: 0.5650 - acc: 0.7728
Epoch 19/20
0s - loss: 0.5532 - acc: 0.7746
Epoch 20/20
0s - loss: 0.5421 - acc: 0.7818
Epoch 1/20
1s - loss: 1.8276 - acc: 0.3413
Epoch 2/20
0s - loss: 1.2688 - acc: 0.5111
Epoch 3/20
0s - loss: 1.1040 - acc: 0.5656
Epoch 4/20
0s - loss: 0.9943 - acc: 0.6061
Epoch 5/20
0s - loss: 0.9177 - acc: 0.6334
Epoch 6/20
0s - loss: 0.8572 - acc: 0.6553
Epoch 7/20
0s - loss: 0.8098 - acc: 0.6784
Epoch 8/20
0s - loss: 0.7655 - acc: 0.6905
Epoch 9/20
0s - loss: 0.7309 - acc: 0.7049
Epoch 10/20
0s - loss: 0.7025 - acc: 0.7156
Epoch 11/20
0s - loss: 0.6769 - acc: 0.7308
Epoch 12/20
0s - loss: 0.6572 - acc: 0.7379
Epoch 13/20
0s - loss: 0.6349 - acc: 0.7422
Epoch 14/20
0s - loss: 0.6159 - acc: 0.7565
Epoch 15/20
0s - loss: 0.5917 - acc: 0.7630
Epoch 16/20
0s - loss: 0.5908 - acc: 0.7659
```

```
Epoch 17/20
0s - loss: 0.5702 - acc: 0.7739
Epoch 18/20
0s - loss: 0.5422 - acc: 0.7842
Epoch 19/20
0s - loss: 0.5438 - acc: 0.7843
Epoch 20/20
0s - loss: 0.5291 - acc: 0.7861
Epoch 1/20
1s - loss: 1.7968 - acc: 0.3416
Epoch 2/20
0s - loss: 1.2795 - acc: 0.4955
Epoch 3/20
0s - loss: 1.1240 - acc: 0.5519
Epoch 4/20
0s - loss: 1.0239 - acc: 0.5824
Epoch 5/20
0s - loss: 0.9453 - acc: 0.6178
Epoch 6/20
0s - loss: 0.8873 - acc: 0.6422
Epoch 7/20
0s - loss: 0.8492 - acc: 0.6591
Epoch 8/20
0s - loss: 0.8038 - acc: 0.6685
Epoch 9/20
0s - loss: 0.7740 - acc: 0.6866
Epoch 10/20
0s - loss: 0.7478 - acc: 0.6949
Epoch 11/20
0s - loss: 0.7237 - acc: 0.7042
Epoch 12/20
0s - loss: 0.7052 - acc: 0.7139
Epoch 13/20
0s - loss: 0.6829 - acc: 0.7236
Epoch 14/20
0s - loss: 0.6637 - acc: 0.7270
Epoch 15/20
0s - loss: 0.6478 - acc: 0.7335
Epoch 16/20
0s - loss: 0.6326 - acc: 0.7457
Epoch 17/20
0s - loss: 0.6155 - acc: 0.7507
Epoch 18/20
0s - loss: 0.5980 - acc: 0.7551
Epoch 19/20
0s - loss: 0.5875 - acc: 0.7605
Epoch 20/20
0s - loss: 0.5778 - acc: 0.7624
```

```
Epoch 1/20
1s - loss: 1.7179 - acc: 0.4074
Epoch 2/20
0s - loss: 1.1938 - acc: 0.5512
Epoch 3/20
0s - loss: 1.0249 - acc: 0.6101
Epoch 4/20
0s - loss: 0.9258 - acc: 0.6458
Epoch 5/20
0s - loss: 0.8529 - acc: 0.6729
Epoch 6/20
0s - loss: 0.7989 - acc: 0.6924
Epoch 7/20
0s - loss: 0.7459 - acc: 0.7126
Epoch 8/20
0s - loss: 0.7077 - acc: 0.7256
Epoch 9/20
0s - loss: 0.6817 - acc: 0.7359
Epoch 10/20
0s - loss: 0.6433 - acc: 0.7486
Epoch 11/20
0s - loss: 0.6220 - acc: 0.7606
Epoch 12/20
0s - loss: 0.5973 - acc: 0.7699
Epoch 13/20
0s - loss: 0.5781 - acc: 0.7751
Epoch 14/20
0s - loss: 0.5586 - acc: 0.7812
Epoch 15/20
0s - loss: 0.5425 - acc: 0.7889
Epoch 16/20
0s - loss: 0.5254 - acc: 0.7965
Epoch 17/20
0s - loss: 0.5100 - acc: 0.8029
Epoch 18/20
0s - loss: 0.4942 - acc: 0.8067
Epoch 19/20
0s - loss: 0.4817 - acc: 0.8095
Epoch 20/20
0s - loss: 0.4723 - acc: 0.8148
Epoch 1/20
1s - loss: 1.6294 - acc: 0.4229
Epoch 2/20
0s - loss: 1.1386 - acc: 0.5544
Epoch 3/20
0s - loss: 1.0113 - acc: 0.6016
Epoch 4/20
0s - loss: 0.9257 - acc: 0.6304
```

```
Epoch 5/20
0s - loss: 0.8546 - acc: 0.6601
Epoch 6/20
0s - loss: 0.8054 - acc: 0.6800
Epoch 7/20
0s - loss: 0.7598 - acc: 0.6957
Epoch 8/20
0s - loss: 0.7282 - acc: 0.7087
Epoch 9/20
0s - loss: 0.6896 - acc: 0.7240
Epoch 10/20
0s - loss: 0.6660 - acc: 0.7321
Epoch 11/20
0s - loss: 0.6344 - acc: 0.7452
Epoch 12/20
0s - loss: 0.6174 - acc: 0.7531
Epoch 13/20
0s - loss: 0.5965 - acc: 0.7616
Epoch 14/20
0s - loss: 0.5760 - acc: 0.7721
Epoch 15/20
0s - loss: 0.5603 - acc: 0.7754
Epoch 16/20
0s - loss: 0.5607 - acc: 0.7740
Epoch 17/20
0s - loss: 0.5393 - acc: 0.7860
Epoch 18/20
0s - loss: 0.5272 - acc: 0.7876
Epoch 19/20
0s - loss: 0.5166 - acc: 0.7947
Epoch 20/20
0s - loss: 0.5028 - acc: 0.7973
Epoch 1/20
1s - loss: 1.6371 - acc: 0.4154
Epoch 2/20
0s - loss: 1.2020 - acc: 0.5441
Epoch 3/20
0s - loss: 1.0715 - acc: 0.5811
Epoch 4/20
0s - loss: 0.9854 - acc: 0.6101
Epoch 5/20
0s - loss: 0.9188 - acc: 0.6368
Epoch 6/20
0s - loss: 0.8616 - acc: 0.6562
Epoch 7/20
0s - loss: 0.8157 - acc: 0.6734
Epoch 8/20
0s - loss: 0.7823 - acc: 0.6928
```

```
Epoch 9/20
0s - loss: 0.7511 - acc: 0.6975
Epoch 10/20
0s - loss: 0.7279 - acc: 0.7105
Epoch 11/20
0s - loss: 0.7015 - acc: 0.7199
Epoch 12/20
0s - loss: 0.6768 - acc: 0.7292
Epoch 13/20
0s - loss: 0.6575 - acc: 0.7393
Epoch 14/20
0s - loss: 0.6453 - acc: 0.7426
Epoch 15/20
0s - loss: 0.6252 - acc: 0.7524
Epoch 16/20
0s - loss: 0.6085 - acc: 0.7576
Epoch 17/20
0s - loss: 0.6025 - acc: 0.7578
Epoch 18/20
0s - loss: 0.5787 - acc: 0.7693
Epoch 19/20
0s - loss: 0.5789 - acc: 0.7666
Epoch 20/20
0s - loss: 0.5580 - acc: 0.7786
Epoch 1/20
1s - loss: 1.3657 - acc: 0.5151
Epoch 2/20
0s - loss: 0.9077 - acc: 0.6585
Epoch 3/20
0s - loss: 0.8035 - acc: 0.6969
Epoch 4/20
0s - loss: 0.7432 - acc: 0.7192
Epoch 5/20
0s - loss: 0.6913 - acc: 0.7399
Epoch 6/20
0s - loss: 0.6394 - acc: 0.7568
Epoch 7/20
0s - loss: 0.6075 - acc: 0.7683
Epoch 8/20
0s - loss: 0.5754 - acc: 0.7804
Epoch 9/20
0s - loss: 0.5536 - acc: 0.7907
Epoch 10/20
0s - loss: 0.5330 - acc: 0.8002
Epoch 11/20
0s - loss: 0.5046 - acc: 0.8112
Epoch 12/20
0s - loss: 0.4864 - acc: 0.8149
```

```
Epoch 13/20
0s - loss: 0.4731 - acc: 0.8203
Epoch 14/20
0s - loss: 0.4527 - acc: 0.8272
Epoch 15/20
0s - loss: 0.4411 - acc: 0.8324
Epoch 16/20
0s - loss: 0.4246 - acc: 0.8397
Epoch 17/20
0s - loss: 0.4138 - acc: 0.8455
Epoch 18/20
0s - loss: 0.4048 - acc: 0.8439
Epoch 19/20
0s - loss: 0.3785 - acc: 0.8546
Epoch 20/20
0s - loss: 0.3817 - acc: 0.8525
Epoch 1/20
1s - loss: 1.2870 - acc: 0.5456
Epoch 2/20
0s - loss: 0.9405 - acc: 0.6455
Epoch 3/20
0s - loss: 0.8518 - acc: 0.6739
Epoch 4/20
0s - loss: 0.7887 - acc: 0.6979
Epoch 5/20
0s - loss: 0.7335 - acc: 0.7167
Epoch 6/20
0s - loss: 0.6941 - acc: 0.7311
Epoch 7/20
0s - loss: 0.6616 - acc: 0.7439
Epoch 8/20
0s - loss: 0.6263 - acc: 0.7595
Epoch 9/20
0s - loss: 0.5989 - acc: 0.7675
Epoch 10/20
0s - loss: 0.5739 - acc: 0.7790
Epoch 11/20
0s - loss: 0.5532 - acc: 0.7886
Epoch 12/20
0s - loss: 0.5293 - acc: 0.7944
Epoch 13/20
0s - loss: 0.5148 - acc: 0.7964
Epoch 14/20
0s - loss: 0.4942 - acc: 0.8060
Epoch 15/20
0s - loss: 0.4728 - acc: 0.8184
Epoch 16/20
0s - loss: 0.4710 - acc: 0.8173
```

```
Epoch 17/20
0s - loss: 0.4498 - acc: 0.8218
Epoch 18/20
0s - loss: 0.4385 - acc: 0.8284
Epoch 19/20
0s - loss: 0.4307 - acc: 0.8343
Epoch 20/20
0s - loss: 0.4148 - acc: 0.8360
Epoch 1/20
1s - loss: 1.5633 - acc: 0.4419
Epoch 2/20
0s - loss: 1.0642 - acc: 0.5978
Epoch 3/20
0s - loss: 0.9282 - acc: 0.6442
Epoch 4/20
0s - loss: 0.8400 - acc: 0.6807
Epoch 5/20
0s - loss: 0.7747 - acc: 0.7011
Epoch 6/20
0s - loss: 0.7233 - acc: 0.7188
Epoch 7/20
0s - loss: 0.6887 - acc: 0.7339
Epoch 8/20
0s - loss: 0.6413 - acc: 0.7498
Epoch 9/20
0s - loss: 0.6184 - acc: 0.7579
Epoch 10/20
0s - loss: 0.5934 - acc: 0.7665
Epoch 11/20
0s - loss: 0.5689 - acc: 0.7772
Epoch 12/20
0s - loss: 0.5545 - acc: 0.7841
Epoch 13/20
0s - loss: 0.5301 - acc: 0.7935
Epoch 14/20
0s - loss: 0.5104 - acc: 0.8006
Epoch 15/20
0s - loss: 0.4964 - acc: 0.8057
Epoch 16/20
0s - loss: 0.4779 - acc: 0.8138
Epoch 17/20
0s - loss: 0.4702 - acc: 0.8148
Epoch 18/20
0s - loss: 0.4593 - acc: 0.8201
Epoch 19/20
0s - loss: 0.4432 - acc: 0.8254
Epoch 20/20
0s - loss: 0.4391 - acc: 0.8306
```

```
Epoch 1/20
1s - loss: 1.4871 - acc: 0.4561
Epoch 2/20
0s - loss: 1.1544 - acc: 0.5611
Epoch 3/20
0s - loss: 1.0256 - acc: 0.6049
Epoch 4/20
0s - loss: 0.9366 - acc: 0.6409
Epoch 5/20
0s - loss: 0.8706 - acc: 0.6604
Epoch 6/20
0s - loss: 0.8221 - acc: 0.6766
Epoch 7/20
0s - loss: 0.7782 - acc: 0.6977
Epoch 8/20
0s - loss: 0.7424 - acc: 0.7056
Epoch 9/20
0s - loss: 0.7144 - acc: 0.7194
Epoch 10/20
0s - loss: 0.6785 - acc: 0.7331
Epoch 11/20
0s - loss: 0.6583 - acc: 0.7402
Epoch 12/20
0s - loss: 0.6324 - acc: 0.7532
Epoch 13/20
0s - loss: 0.6098 - acc: 0.7572
Epoch 14/20
0s - loss: 0.6015 - acc: 0.7598
Epoch 15/20
0s - loss: 0.5857 - acc: 0.7681
Epoch 16/20
0s - loss: 0.5615 - acc: 0.7786
Epoch 17/20
0s - loss: 0.5565 - acc: 0.7784
Epoch 18/20
0s - loss: 0.5407 - acc: 0.7871
Epoch 19/20
0s - loss: 0.5284 - acc: 0.7890
Epoch 20/20
0s - loss: 0.5156 - acc: 0.7950
```

Saving / loading trained models In case it is no time to compute it again (it will work only if the splitting were done with the same random_state)

```
In [11]: def saveMODELS():
             for i in range(len(MODELS)):
                 filename = "ML3hw_model" + str(i) + ".h5"
```

```
            MODELS[i].save(filename)

    from keras.models import load_model

    def loadMODELS():
        models = []
        for i in range(len(feature_range)):
            filename = "ML3hw_model" + str(i) + ".h5"
            model = load_model(filename)
            models.append(model)
        return models

    #saveMODELS()
```

```
In [12]: #MODELS = loadMODELS()
```

1.5.3 Prediction

```
In [184]: def predictByANN(transformedRow,indexToPredict,models=MODELS):
              """
              row - numpy (binary) array of length inputvector_length
              """
              prediction = models[indexToPredict].predict(np.array([transformedRow]))[0]
              output = np.zeros(len(feature_range[indexToPredict]))
              output[np.argmax(prediction)] = 1
              return output
```

Example

```
In [235]: rowtopredict = 0
          featuretopredict = 1

          row = TEST.iloc[rowtopredict,:].values.copy()

          y_true = row[featuretopredict]

          row[featuretopredict] = '?'
          transformedrow = transformInput(row)

          y_pred_encoded = predictByANN(transformedrow, featuretopredict)
          y_pred = deencodeValue(y_pred_encoded,featuretopredict)
          print("\n Without dropout")
          print("Encoded prediction: ", y_pred_encoded)
          print("Deencoded prediction: ", deencodeValue(y_pred_encoded,featuretopredict))
          print("True value: ", y_true)
          %timeit predictByANN(transformedrow, featuretopredict)

          y_pred_encoded = predictByANN(transformedrow, featuretopredict,MODELSDROPOUT)
```

```
    y_pred = deencodeValue(y_pred_encoded,featuretopredict)
    print("\n\n With dropout")
    print("Encoded prediction: ", y_pred_encoded)
    print("Deencoded prediction: ", deencodeValue(y_pred_encoded,featuretopredict))
    print("True value: ", y_true)
    %timeit predictByANN(transformedrow, featuretopredict,MODELSDROPOUT)
```

```
 Without dropout
Encoded prediction:  [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
Deencoded prediction:  4
True value:  4
485 ţs ś 30.4 ţs per loop (mean ś std. dev. of 7 runs, 1000 loops each)


 With dropout
Encoded prediction:  [ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
Deencoded prediction:  4
True value:  4
500 ţs ś 24 ţs per loop (mean ś std. dev. of 7 runs, 1000 loops each)
```

1.6 Evaluating results

1.6.1 Accuracy on testing set

Run out prediction on noiseless data and compute the accuracy for each feature

Feature	**lettr**	**xbox**	**ybox**	**width**	**high**	**onpix**	**xbar**	**ybar**	**x2bar**	**y2bar**	**xybar**	**x2ybr**	**xy2br**	**xege**	**xegvy**	**yege**	**yegvx**
Accuracy:	90.5	72.1	60.1	70.5	73.8	70.1	65.3	63.8	64.6	62	68.8	67.5	63.5	73.5	69.3	69	65.5
Accuracy with Dropout:	92.7	74.7	61.9	72.1	74.6	70.4	67.8	65.7	65.8	63.5	71.2	69.7	65.8	75.4	71.6	70.3	67.1

Results

1.6.2 Accuracy on noisy testing set

Compute accuracy of **letter** column on noisy test set (with additional query marks) with different noise level (frequency of missing values). However, the same can be done for any column.

Algorithm of computing accuracy * Create N (10, 30, 50, etc.) missing values in each column independently * Make a prediction on Letter column and compute accuracy * Repeat 10 times -

compute average result

Results | Frequency | Accuracy (%) | Accuracy with Dropout: | |-----------|:------------:|------------------------| | 10 | 90.5 | 92.2 | | 30 | 90.1 | 91.8 | | 50 | 89.6 | 91.4 | | 100 | 88.7 | 90.5 | | 200 | 86.5 | 88.8 | | 400 | 82.4 | 85.2 | | 800 | 73.5 | 77.7 | | 1000 | 69.0 | 73.8 | #### Realization

```
In [296]: number_of_NAs = 1000
          testWithNA = TEST.copy()
          columnToPred = 0
          testWithNA.iloc[:,columnToPred] = '?'
          for i in range(testWithNA.shape[1]):
              for k in range(number_of_NAs):
                  j = np.random.randint(testWithNA.shape[0])
                  testWithNA.iloc[j,i] = '?'
                  #testWithNA.iloc[j,0] = '?'

          testWithNA.iloc[:1,:]
```

```
Out[296]:   lettr xbox ybox width high onpix xbar ybar x2bar y2bar xybar x2ybr xy2br  \
          0     ?    4    5     5    6     ?    8    8     5     2     ?     ?    10

            xege xegvy yege yegvx
          0    3     9    5     ?
```

Making a prediction

```
In [297]: testWithNAnumpy = testWithNA.values.copy()
          testWithNAnumpy2 = testWithNA.values.copy()
          testWithNAprediction = testWithNA.values.copy()
          testnumpy = TEST.values.copy()
          models = MODELSDROPOUT

          for i in range(testWithNAnumpy.shape[0]):
              row = testWithNAnumpy[i]
              if '?' in row:
                  newrow = row.copy()
                  transformedRow = transformInput(row)
                  missingValuesIndeces = np.where(row=='?')[0]
                  for k in missingValuesIndeces:
                      newrow[k] = deencodeValue(predictByANN(transformedRow,k,models),k)
                  testWithNAprediction[i]=newrow

          actual = testnumpy[np.where(testWithNAnumpy2[:,columnToPred]=='?')][:,columnToPred]
          predicted = testWithNAprediction[np.where(testWithNAnumpy2[:,columnToPred]=='?')][:,co
          print(sum(actual==predicted) / len(actual))
```

```
0.74175
```