Simple neural network classification over MNIST

Anton Antonov
MathematicaVsR at GitHub
May 2018

Introduction

This notebook is part of the MathematicaVsR at GitHub project “DeepLearningExamples”.

This notebook has code corresponding to neural network creation, training, and testing in RStudio’s opening Keras page: https://keras.rstudio.com/index.html .

(Also we can say that this notebook has code that corresponds to code in the book “Deep learning with R” by F. Chollet and J. J. Allaire. See the GitHub repository: https://github.com/jjallaire/deep-learning-with-r-notebooks ; specifically the notebook “A first look at a neural network”.)

Get code

Here we load a package with utilities:

```
In[1]:= Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/MathematicaForPredictionUtilities.m"]
```

» Importing from GitHub: `MosaicPlot.m`

» Importing from GitHub: `CrossTabulate.m`

Get data

We can get the data from Wolfram’s data repository:

```
In[2]:= (*ResourceObject["MNIST"]*)
```

or we can directly use `ExampleData`:

```
In[3]:= trainingData = ExampleData[{"MachineLearning", "MNIST"}, "TrainingData"];
testData = ExampleData[{"MachineLearning", "MNIST"}, "TestData"];
```

Here is a description of the variable names:

```
In[5]:= colNames = Flatten[List @@ ExampleData[{"MachineLearning", "MNIST"}, "VariableDescriptions"]];
GridTableForm[List /@ colNames]
```

Out[6]=

#	1
1	28x28 grayscale image
2	Integer in the range 1-10

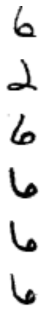
Data summaries

```
In[7]:= Dimensions[Flatten@*List @@@ trainingData]
```

```
Out[7]= {60 000, 2}
```

In[8]:=
```
Dimensions[Flatten@*List @@@ testData]
```
Out[8]=
```
{10 000, 2}
```

In[9]:=
```
Magnify[RecordsSummary[trainingData, Thread → True], 0.8]
```
Out[9]=



```
{ 1 column 1                  } → { 1 column 1     }
  (6 images)  1, 1, 1, 1, 1, 1        Min     0
  (Other)     59 994                  1st Qu  2
                                      Median  4
                                      Mean    4.45393
                                      3rd Qu  7
                                      Max     9
```

In[10]:=
```
Magnify[RecordsSummary[testData, Thread → True], 0.8]
```
Out[10]=

```
{ 1 column 1                  } → { 1 column 1     }
  (6 images)  1, 1, 1, 1, 1, 1        Min     0
  (Other)     9994                    1st Qu  2
                                      Median  4
                                      Mean    4.4434
                                      3rd Qu  7
                                      Max     9
```

In[11]:=
```
Tally[ImageDimensions /@ trainingData[[All, 1]]]
```
Out[11]=
```
{{{28, 28}, 60 000}}
```

Transform data: images into 1D arrays

In[24]:=
```
trainingData = Map[Flatten[ImageData[#[[1]]]] → #[[2]] &, trainingData];
```

In[25]:=
```
testData = Map[Flatten[ImageData[#[[1]]]] → #[[2]] &, testData];
```

The neural network

Build the network:

In[32]:=
```
model =
 NetChain[{
   ElementwiseLayer[Ramp],
   LinearLayer[256, "Input" → Length[First@trainingData〚All, 1〛]],
   DropoutLayer[0.4],
   ElementwiseLayer[Ramp],
   LinearLayer[128],
   DropoutLayer[0.3],
   LinearLayer[10],
   SoftmaxLayer["Output" -> NetDecoder[{"Class", Range[0, 9]}]]
  }]
```

Out[32]= NetChain[uninitialized

	Input	vector (size: 784)
1	Ramp	vector (size: 784)
2	LinearLayer	vector (size: 256)
3	DropoutLayer	vector (size: 256)
4	Ramp	vector (size: 256)
5	LinearLayer	vector (size: 128)
6	DropoutLayer	vector (size: 128)
7	LinearLayer	vector (size: 10)
8	SoftmaxLayer	vector (size: 10)
	Output	class

]

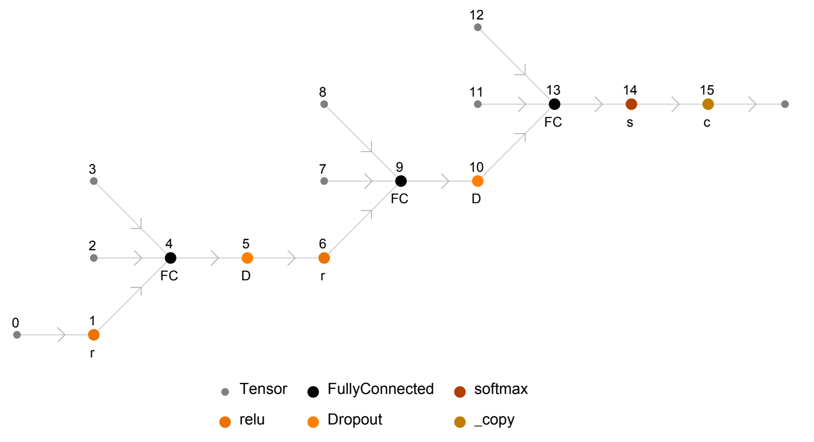
From the documentation:

CrossEntropyLossLayer["Index"] is used automatically by NetTrain when the final activation used for an output is a SoftmaxLayer.

Visualize the network:

In[33]:=
```
NetInformation[model, "MXNetNodeGraphPlot"]
```

Out[33]=



Train the network:

In[56]:=
```
tNet = NetTrain[model, trainingData, ValidationSet → Scaled[0.05], "MaxTrainingRounds" → 80]
```

Out[56]= NetChain[

	Input	vector (size: 784)
1	Ramp	vector (size: 784)
2	LinearLayer	vector (size: 256)
3	DropoutLayer	vector (size: 256)
4	Ramp	vector (size: 256)
5	LinearLayer	vector (size: 128)
6	DropoutLayer	vector (size: 128)
7	LinearLayer	vector (size: 10)
8	SoftmaxLayer	vector (size: 10)
	Output	class

]

Testing

Predict results for the test data:

In[57]:=
```
clRes = tNet /@ testData⟦All, 1⟧;
Short[clRes]
```

Out[58]//Short= {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ≪9958≫, 9, 4, 9, 9, 9, 9, 9, 9, 4, 9, 4, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9}

This computes a classifier measurements object:

In[59]:=
```
cm = ClassifierMeasurements[tNet, trainingData]
```

Out[59]= ClassifierMeasurementsObject[
Classifier: Net
Number of test examples: 60 000
Number of classes: 10
Accuracy: 0.95591667 ± 0.00084
Data not in notebook; Store now »
]

Here is the accuracy:

In[60]:=
```
cm["Accuracy"]
```

Out[60]= 0.955917

Here are the precision, recall, specificity, false positive rate metrics:

In[61]:=
```
ms = {"Precision", "Recall", "Specificity", "FalsePositiveRate"};
Transpose[Dataset[AssociationThread[ms -> cm[ms]]]]
```

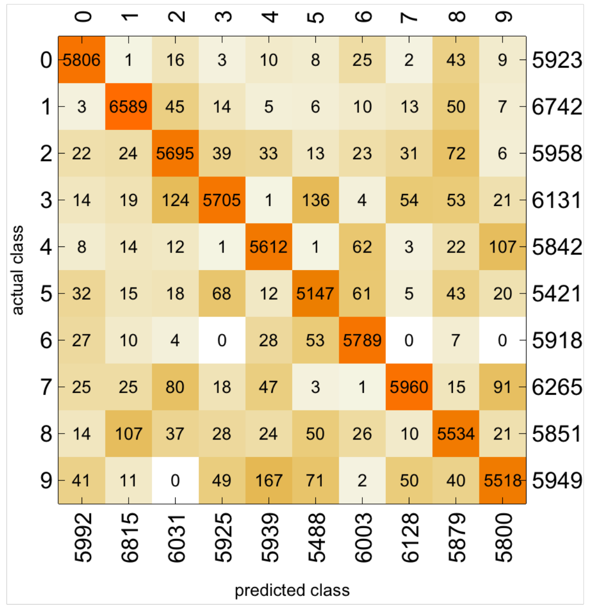
Out[62]=

	Precision	Recall	Specificity	FalsePositiveRate
0	0.968959	0.980246	0.99656	0.00343954
1	0.966838	0.977306	0.995757	0.00424349
2	0.944288	0.955858	0.993783	0.00621739
3	0.962869	0.930517	0.995916	0.00408398
4	0.94494	0.96063	0.993962	0.00603789
5	0.937864	0.949456	0.993752	0.00624782
6	0.964351	0.978202	0.996043	0.00395695
7	0.972585	0.951317	0.996874	0.00312645
8	0.941317	0.945821	0.993629	0.00637131
9	0.951379	0.927551	0.994783	0.00521729

Here is a confusion matrix plot:

In[63]:=
```
cm["ConfusionMatrixPlot"]
```

Out[63]=



Here are the top confused digits:

In[64]:=
```
cm["TopConfusions"]
```

Out[64]= {3 → 5, 4 → 9, 7 → 9, 2 → 8, 4 → 6, 5 → 6, 3 → 7, 3 → 8, 1 → 8, 1 → 2}