DeepLearning

A substantial effort was put into Deep Learning for Maple 2021, including offering a variety of new specialty forms of neural networks.

```
> with(DeepLearning):
```

New commands

These commands are new for Maple 2021:

- BidirectionalLayer
- ConvolutionLayer
- DenseLayer
- DropoutLayer
- EmbeddingLayer
- FlattenLayer
- GatedRecurrentUnitLayer
- GetEagerExecution
- LongShortTermMemoryLayer
- MaxPoolLayer
- Sequential
- SetEagerExecution

New types of neural networks

The addition of Layer objects offers an easy mechanism for building specialized types of neural networks, including the following:

- **Convolutional neural networks**: using ConvolutionLayer. These networks are often used for tasks such as image and video recognition.
- **Recurrent neural networks**: using GatedRecurrentUnitLayer or LongShortTermMemoryLayer. These networks are often used for text processing or classification.

The new Sequential command allows one or more Layers to be stacked together, feeding the output from one layer into the next as input. This allows you to build sophisticated special-purpose neural networks by composing layers of different types.

Both Layer objects and the composite models created by the Sequential command are examples of Model objects. A Model can be fed new data to generate predictions and can also be saved to a file in the standard HDF5 file format.

Examples

Sequential Model: The Pima Diabetes Dataset

In this example we build a **Sequential** model with **DenseLayers** to model the Pima diabetes dataset, generating predictions about whether individuals will be diagnosed with a diabetes based on other diagnostic measurements included in the dataset.

This dataset is originally from the United States National Institute of Diabetes and Digestive and Kidney Diseases. All patients here are females at least 21 years old of Pima Indian heritage.

Source:
Pima Indian Diabetes Database, https://www.kaggle.com/uciml/pima-indians-diabetes-database. Data originally published in:

- Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In Proceedings of the Symposium on Computer Applications and Medical Care (pp. 261--265). IEEE Computer Society Press.

We first load the data into a DataFrame:

> $dataset := Import(\ \text{"datasets/pima-epidemiology-diabetes.csv"}, base = datadir\ )$

$dataset :=$

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31.0	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0.	0.232	54	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

768 x 9 DataFrame

As before, we divide the dataset into training and test data. We are attempting to predict the last column, *Outcome*.

> $training\_data, test\_data := dataset\left[\ 1..600,..\right], dataset\left[\ 601..-1,..\right]$

training_data, test_data :=

	Pregnancies	*Glucose*	*BloodPressure*	*SkinThickness*	*Insulin*	*BMI*	*DiabetesPedigreeFunction*	*Age*	*Outcome*
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31.0	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0.	0.232	54	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

600 x 9 DataFrame ,

	Pregnancies	*Glucose*	*BloodPressure*	*SkinThickness*	*Insulin*	*BMI*	*DiabetesPedigreeFunction*	*Age*	*Outcome*
601	1	108	88	19	0	27.1	0.400	24	0
602	6	96	0	0	0	23.7	0.190	28	0
603	1	124	74	36	0	27.8	0.100	30	0
604	7	150	78	29	126	35.2	0.692	54	1
605	4	183	0	0	0	28.4	0.212	36	1
606	1	124	60	32	0	35.8	0.514	21	0
607	1	181	78	42	293	40.0	1.258	22	1
608	1	92	62	25	41	19.5	0.482	25	0
609	0	152	82	39	272	41.5	0.270	27	0
610	1	111	62	13	182	24.0	0.138	23	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

168 x 9 DataFrame

We define a neural network using the Sequential command which stacks one or more neural network **layers**.

```
> model := Sequential([ DenseLayer(12, activation="relu"), DenseLayer(8, activation
  ="relu"), DenseLayer(1, activation="sigmoid") ])
```

model := [*DeepLearning Model*
<tensorflow.python.keras.engine.sequential.Sequential object at 0x138244bb0>]

```
> model:-Compile( loss="binary_crossentropy", optimizer="adam", metrics
  =[ "accuracy" ] )
```

Python:—None

We can train the data easily with the **Fit** command:

```
> model:-Fit( training_data[ .., 1..8 ], training_data[Outcome], epochs=150, batchsize
  =10 )
```

"<Python object: <tensorflow.python.keras.callbacks.History object at 0x13838f430>>"

We now have a trained model whose accuracy we can test against the test data or

against any new data. The accuracy is not especially high but nevertheless useful for predictive purposes.

```
> model:-Evaluate( test_data[ .., 1 ..8 ], test_data[Outcome] )
```

$$\{\text{"loss"} = 0.613269329071045, \text{"accuracy"} = 0.690476179122925\}$$

To get a sense of what how this model behaves on individuals within the dataset, we can take a slice from the test data and compare the observed vs. predicted outcome.

```
> predictions := model:-Predict( test_data[ 760 ..768, 1 ..8 ] )
```

$$predictions := \begin{bmatrix} 0.222275912761688 \\ 0.132090508937836 \\ 0.561888575553894 \\ 0.109222382307053 \\ 0.300420016050339 \\ 0.234794706106186 \\ 0.226543396711349 \\ 0.162764191627502 \\ 0.144395038485527 \end{bmatrix}$$

```
> test_data[ 760 ..768, Outcome ], predictions
```

$$\begin{bmatrix} 760 & 1 \\ 761 & 0 \\ 762 & 1 \\ 763 & 0 \\ 764 & 0 \\ 765 & 0 \\ 766 & 0 \\ 767 & 1 \\ 768 & 0 \end{bmatrix}, \begin{bmatrix} 0.222275912761688 \\ 0.132090508937836 \\ 0.561888575553894 \\ 0.109222382307053 \\ 0.300420016050339 \\ 0.234794706106186 \\ 0.226543396711349 \\ 0.162764191627502 \\ 0.144395038485527 \end{bmatrix}$$