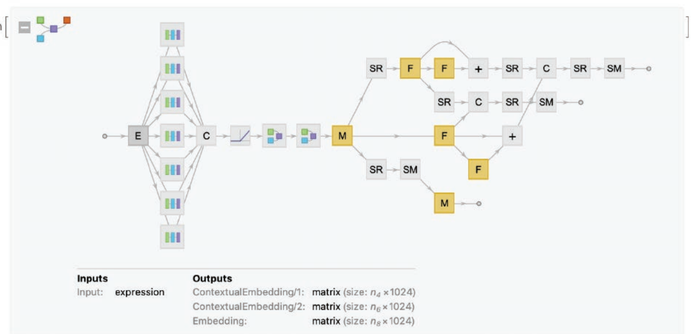
Deep Learning in Mathematica

Taliesin Beynon, Sebastian Bodenstein

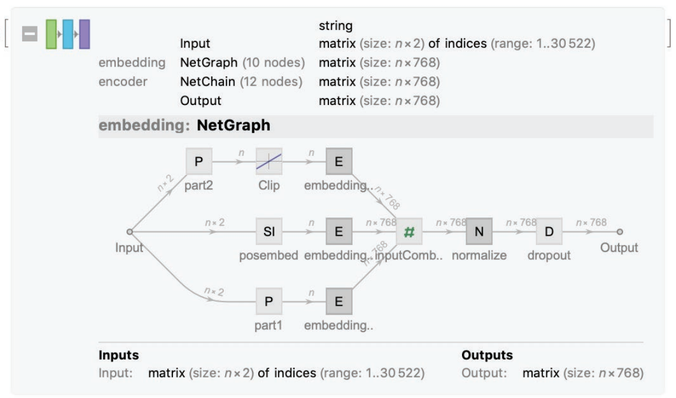
Interactive exploration of network structure

```
NetModel["ELMo Contextual Word Representations Trained on 1B Word Benchmark"]
```

NetGraph[]

Inputs		Outputs	
Input:	expression	ContextualEmbedding/1:	matrix (size: $n_4 \times 1024$)
		ContextualEmbedding/2:	matrix (size: $n_6 \times 1024$)
		Embedding:	matrix (size: $n_8 \times 1024$)

```
NetModel["BERT Trained on BookCorpus and English Wikipedia Data"]
```

NetChain[

		string
	Input	matrix (size: $n \times 2$) of indices (range: 1..30 522)
embedding	NetGraph (10 nodes)	matrix (size: $n \times 768$)
encoder	NetChain (12 nodes)	matrix (size: $n \times 768$)
	Output	matrix (size: $n \times 768$)

embedding: NetGraph

Inputs		Outputs	
Input:	matrix (size: $n \times 2$) of indices (range: 1..30 522)	Output:	matrix (size: $n \times 768$)

Programmatic access to hundreds of models via NetModel

```
NetModel["ResNet-101
```

"ResNet-101 Trained on Augmented CASIA-WebFace Data"

"ResNet-101 Trained on ImageNet Competition Data"

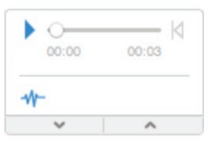
"ResNet-101 Trained on YFCC100m Geotagged Data"

Curated pretrained nets with rich examples (inspired TF-Hub)

https://resources.wolframcloud.com/NeuralNetRepository/

Record an audio sample and transcribe it:

```
record = AudioCapture[]
```




```
NetModel["Deep Speech 2 Trained on Baidu English Data"][record]
{t, h, i, s,  , i, s,  , a,  , t, e, s, t,  , r, e, c, o, r, d, i, n, g}
```

Foundation for neural APIs in Mathematica

```
FindTextualAnswer[WikipediaData["Paris"],
 "How many people live in Paris?"]
```

```
2,229,621
```

```
ImageContents[  ]
```

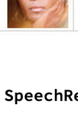
Image	Concept	BoundingBox
	elephant	Rectangle[{433
	elephant	Rectangle[{336
	zebra	Rectangle[{35.!
	elephant	Rectangle[{160
	zebra	Rectangle[{3.4

```
ImageRestyle[  ,  ]
```



```
TextContents[
 "The flag of Italy is green, white and red. Since 1861,
    the capital is Rome, which also serves as the
    capital of the Lazio region. With 2,872,800
    residents in 1,285 km2 (496.1 sq mi)",
 Automatic, {"HighlightedSnippet", "Interpretation"}]
```

HighlightedSnippet	Interpretation
flag of Italy is green, white and red.	Italy
flag of Italy is green, white and red.	
of Italy is green, white and red. Since 1861,	
is green, white and red. Since 1861, the	
white and red. Since 1861, the capital is	1861
1861, the capital is Rome, which also serves	Rome, Lazio, Italy

Image	Age	Gender	Emotion
	48	Male	neutral
	47	Male	happiness
	54	Male	fear
	35	Male	neutral
	42	Male	neutral
	67	Male	anger
	44	Female	neutral

```
SpeechRecognize[AudioRecord[]]
```

```
hello computer can you hear me
```

Elegant, flexible live reporting

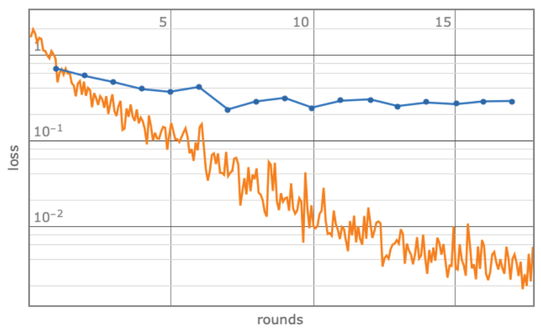
```
NetTrain["LeNet", "MNISTSample",
 ValidationSet → Scaled[0.1]]
```

Training Progress

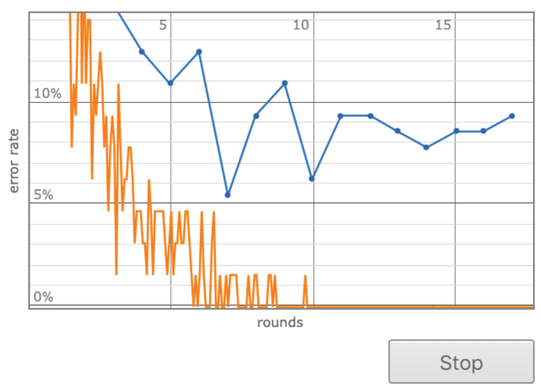
progress	42% (round 19/44)		
time	11s elapsed, 15s left, 2196 examples/s		
method	ADAM optimizer, batch size 64, CPU		
	current	round	validation
loss	3.5×10^{-3}	3.27×10^{-3}	2.9×10^{-1}
error	0%	0%	9.38%

training set — validation set

loss



error rate

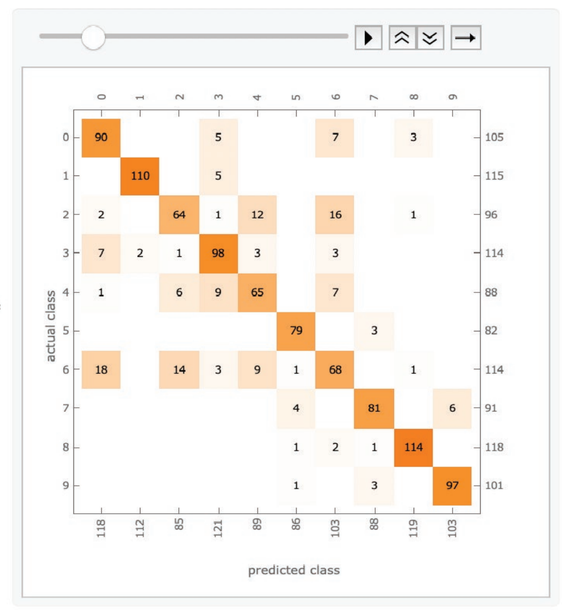


Stop

Measure arbitrary net properties during training

```
results = NetTrain["LeNet", "FashionMNISTSample", All,
  TrainingProgressMeasurements → "ConfusionMatrixPlot"]
ListAnimate[results["RoundMeasurementsLists",
  "ConfusionMatrixPlot"]]
```

Out[2]=

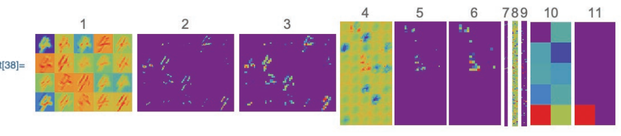


Powerful queries of internal network state (RNNs, gradients, activations)

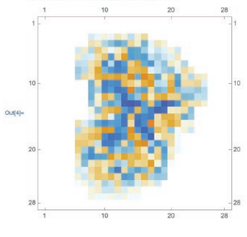
```
activations = lenet[  , NetPort[All, "Output"]];
```

```
KeyValueMap[ArrayPlot[ ··· ] &, activations] // Row
```

Out[38]=




```
inputGradient = lenetTraining[
  <|"Input" -> , "Output" → 3|>,
  NetPortGradient["Input"]];
MatrixPlot[inputGradient[[1]]]
```



```
model = NetModel[
  "Wolfram English Character-Level
    Language Model V1"];
```

```
NestList[
  NetStateObject[model],
  "Hello, why are ", 100] // StringJoin
```

```
Hello, why are you so strange?"

"I don't know what you want to say to you."

"I will not be able to see him as I d
```

Extremely easy to use

```
trained = NetTrain[
   NetChain[{LinearLayer[], LogisticSigmoid}],
   {1 → False, 2 → False, 3 → True, 4 → True}]
```

NetChain[

Input port:	real
Output port:	boolean
Number of layers:	2

```
trained[3.5]
```

```
True
```

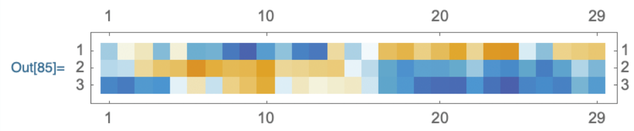
Fully automatic handling of variable-length arrays

```
gru = NetInitialize@NetChain[{
    UnitVectorLayer[],
    ConvolutionLayer[10, 3, Interleaving → True],
    GatedRecurrentLayer[3]},
   "Input" → "Characters"]
```

NetChain[

		string
	Input	vector of n_1 indices (range: 1..97)
1	UnitVectorLayer	matrix (size: $n_1 \times 97$)
2	ConvolutionLayer	matrix (size: $n_2 \times 10$)
3	GatedRecurrentLayer	matrix (size: $n_2 \times 3$)
	Output	matrix (size: $n_2 \times 3$)

```
gru["i am a variable length sequence"] // Transpose //
 MatrixPlot
```



Flexible functional abstractions

```
AttentionLayer["Key" → {"m", 2},
 "Value" → {"m", 3}, "Query" → {"n", 1}]
```

AttentionLayer[uninitialized

Parameters	
Scoring net:	NetGraph[...]
Mask:	None
Score rescaling:	None
Ports	
Key:	matrix (size: $m \times 2$)
Value:	matrix (size: $m \times 3$)
Query:	matrix (size: $n \times 1$)
Output:	matrix (size: $n \times 3$)

```
NetBidirectionalOperator[
  LongShortTermMemoryLayer[3, "Input" → {"Varying", 2}]
 ]
```

NetBidirectionalOperator[uninitialized

Forward net:	LongShortTermMemoryLayer[...]
Backward net:	LongShortTermMemoryLayer[...]

NetMapOperator — define a network that maps over a sequence

NetMapThreadOperator — define a network that maps over multiple sequences

NetFoldOperator — define a recurrent network that folds in elements of a sequence

NetPairEmbeddingOperator ▪ **NetNestOperator** ▪ **NetBidirectionalOperator**

Principled design philosophy

Avoid low-level details

- No batch dimension during net definition
- Variable length dimensions treated as abstract symbols:
  RNN operates on a n×20 matrix
  padding for batches of unequal n done under the hood
- GPU training via a single option TargetDevice -> “GPU”

Nets should appear and act like functions

- NetEncode/Decoder translate audio, text, images <-> arrays
- Makes net completely self-contained, exportable, deployable

Nets should be stateless objects (language is immutable)

- NetStateObject encapsulates states for e.g. sequence generation
- Nets can be safely and easily composed inside one another

Extensive built-in documentation

- Hundreds of examples
- Tutorials for Computer Vision, Sequence Learning, Unsupervised
  Out-of-core training, Mixture Density Nets, Style Transfer, etc.

Clear, specific error messages

- E.g. “incompatible ranks for output of layer 1, LinearLayer[{5},..],
  and input to layer 2, LongShortTermMemoryLayer[5,..];
  a vector is not compatible with a matrix, respectively.”