Neural network layers primer

Part of Sebastian Bodenstein's presentation at Wolfram U

http://www.wolfram.com/broadcast/video.php?c=442&v=2173

Orlando Machine Learning and Data Science meetup

Deep learning series, part 2

Anton Antonov

2018-06-02

What Are Neural Nets?

- **Modern term**: differentiable programming
- Based on the Introduction to Neural Nets tutorial

1. Layers

- A layer is the simplest component of a network. Create a layer:

In[167]:=
```
elem = ElementwiseLayer[Tanh]
```

Out[167]=
```
ElementwiseLayer[ Function: Tanh
                  Output: tensor ]
```

- Layers **only** act on numeric tensors:

In[168]:=
```
elem@{1, 2, 3}
N@Tanh@{1, 2, 3}
```

Out[168]=
```
{0.761594, 0.964028, 0.995055}
```

Out[169]=
```
{0.761594, 0.964028, 0.995055}
```

- Layers are differentiable. Differentiability is a key property that allows for the efficient training of nets, which we will see later:

In[170]:=
```
elem[{1, 2, 3}, NetPortGradient["Input"]]
```

Out[170]=
```
{0.419974, 0.0706508, 0.009866}
```

In[171]:=
```
D[Tanh[x], x] /. x -> {1., 2., 3.}
```

Out[171]=
```
{0.419974, 0.0706508, 0.00986604}
```

What Are Neural Nets?

1. Layers (continued)

- They can run on both NVIDIA GPUs and CPUs:

```
In[172]:= elem[{1, 2, 3}, TargetDevice → "GPU"]
```

··· ElementwiseLayer: TargetDevice -> GPU could not be used; your system does not appear to have an NVIDIA GPU.

```
Out[172]= $Failed
```

- They do shape inference:

```
In[173]:= ElementwiseLayer[Tanh, "Input" → {4, 32}]

Out[173]= ElementwiseLayer[ Function: Tanh
                            Output: matrix (size: 4 × 32) ]
```

- Certain layers have learnable parameters
  - without this, no learning would be possible!

```
In[174]:= dot = LinearLayer[3, "Input" → 2]

Out[174]= LinearLayer[ uninitialized  Input: vector (size: 2)
                                      Output: vector (size: 3) ]
```

Initialize the parameters in the layer:

```
In[175]:= dot2 = NetInitialize@dot

Out[175]= LinearLayer[ Input: vector (size: 2)
                       Output: vector (size: 3) ]
```

```
In[176]:= NetExtract[dot2, "Weights"]

Out[176]= {{-0.110853, -0.720296}, {-0.178274, 0.974589}, {0.00243454, 0.928205}}
```

What Are Neural Nets?

1. Layers (continued)

So far, we have seen layers that have exactly one input. Some layers have more than one input. For example, MeanSquaredLossLayer compares two arrays, called the *input* and the *target*, and produces a single number that represents Mean[$(input - target)$^2].

In[177]:=
```
msloss = MeanSquaredLossLayer[]
```

Out[177]=
```
MeanSquaredLossLayer[ Input: tensor
                      Target: tensor ]
```

The inputs of the layer are named and must be supplied in an association when the net is applied:

In[178]:=
```
msloss[<|"Input" → {1, 2, 3}, "Target" → {4, 0, 4}|>]
```

Out[178]=
```
4.66667
```

The full list of available layers is:

In[179]:=
```
? *Layer
```

▼ System`

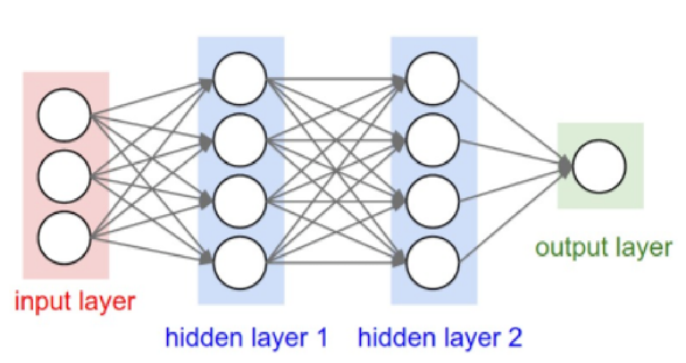
AggregationLayer	DeconvolutionLayer	LongShortTermMemoryLayer	SequenceRestLayer
AppendLayer	DotLayer	MeanAbsoluteLossLayer	SequenceReverseLayer
BasicRecurrentLayer	DotPlusLayer	MeanSquaredLossLayer	SoftmaxLayer
BatchNormalizationLayer	DropoutLayer	PaddingLayer	SpatialTransformationLayer
CatenateLayer	ElementwiseLayer	PartLayer	SummationLayer
ConstantArrayLayer	EmbeddingLayer	PoolingLayer	ThreadingLayer
ConstantPlusLayer	FlattenLayer	ReplicateLayer	TotalLayer
ConstantTimesLayer	GatedRecurrentLayer	ReshapeLayer	TransposeLayer
ContrastiveLossLayer	ImageAugmentationLayer	ResizeLayer	UnitVectorLayer
ConvolutionLayer	InstanceNormalizationLayer	SequenceAttentionLayer	
CrossEntropyLossLayer	LinearLayer	SequenceLastLayer	
CTCLossLayer	LocalResponseNormalizationLayer	SequenceMostLayer	

What Are Neural Nets?

1. Layers: What do they do?

The simplest learnable layer is the **LinearLayer**:

In[180]:=



- Taken from Convolution Networks by Stanford CS231

This is just:

In[181]:=
```
linear[data_, weight_, bias_] := Dot[weight, data] + bias
```

Comparing this to a **LinearLayer**:

In[182]:=
```
layer = NetInitialize@LinearLayer[2, "Input" → 3]
layer[{2, 10, 3}]
```

Out[182]= LinearLayer[

Parameters
Output dimensions: 2

Arrays
Weights: matrix (size: 2 × 3)
Biases: vector (size: 2)

Ports
Input: vector (size: 3)
Output: vector (size: 2)

]

Out[183]= {4.2519, 5.59917}

In[184]:=
```
linear[{2, 10, 3}, NetExtract[layer, "Weights"], NetExtract[layer, "Biases"]]
```

Out[184]= {4.2519, 5.59917}

What Are Neural Nets?

Single neural net layers are generally not useful by themselves. We usually need to combine multiple layers together to do something interesting.

- The simplest container is a chain

2. Chain Containers

- Chain together two operations:

In[185]:=
```
net = NetChain[{ElementwiseLayer[Tanh], ElementwiseLayer[LogisticSigmoid]}]
```

Out[185]= NetChain[Input port: tensor; Output port: tensor; Number of layers: 2]

- Equivalent to:

In[186]:=
```
f[x_] := LogisticSigmoid@Tanh@x
```

- Equivalent on data:

In[187]:=
```
data = {1., 2., 3.};
net@data
f@data
```

Out[188]= {0.6817, 0.723927, 0.730085}

Out[189]= {0.6817, 0.723927, 0.730085}

What Are Neural Nets?

3. Graph Containers

NetChain does not allow a net to take more than one input, so we need to use **NetGraph** to build the training network.

- Create a **NetGraph**:

```
In[190]:= net = NetGraph[{ElementwiseLayer[Tanh], ElementwiseLayer[LogisticSigmoid], TotalLayer[]}, {NetPort["Input1"] → 1, NetPort["Input2"] → 2, {1, 2} → 3}]

Out[190]= NetGraph[ Number of inputs: 2
                    Output port: tensor
                    Number of layers: 3 ]
```

- Equivalent to:

```
In[191]:= func = (Tanh@#Input1 + LogisticSigmoid@#Input2) &;
```

- Evaluate on data:

```
In[192]:= data = <|"Input1" -> {0.1, -2.4}, "Input2" → {-1.2, 3.4}|>;
          net@data
          func@data

Out[193]= {0.331143, -0.0159703}

Out[194]= {0.331143, -0.0159703}
```

- As all of the layers are differentiable, so is the container:

```
In[195]:= net[data, NetPortGradient["Input1"]]

Out[195]= {0.990066, 0.0323837}
```

What Are Neural Nets?

4. Containers Continued

- Containers behave exactly like normal layers!
  - differentiable, run on GPUs, etc
- containers can be nested, as they are just like normal layers:

In[196]:=
```
NetChain[{NetChain[{LinearLayer[]}]}]
```

Out[196]=
```
NetChain[ uninitialized  Input port:        tensor
                         Output port:       tensor
                         Number of layers:  1 ]
```

- models in the Repository are almost all some form of container:

In[197]:=
```
NetModel["AdaIN-Style Trained on MS-COCO and Painter by Numbers Data"]
```

Out[197]=
```
NetGraph[ Number of inputs:  2
          Output port:       image
          Number of layers:  4 ]
```

What Are Neural Nets?

5. NetEncoders

Fundamentally, because they must be *differentiable*, neural net layers operate on numeric tensors. However, we often want to train and use nets on other data, such as images, audio, text, etc. To do this, we can use a **NetEncoder** to translate this data to numeric tensors.

Create an image **NetEncoder** that produces a 1 × 12 × 12 tensor:

```
In[198]:= imageenc = NetEncoder[{"Image", {12, 12}, "ColorSpace" → "Grayscale"}]
```

Out[198]= NetEncoder[

Type:	Image
Image size:	{12, 12}
Color space:	Grayscale
Color channels:	1
Mean image:	None
Variance image:	None
Output:	3–tensor (size: 1 × 12 × 12)

]

Apply the encoder to an image:

```
In[199]:= imageenc[
```

```
]
```

```
Out[199]= {{{1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.}, {1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.}, {1., 1., 1., 1., 0., 0., 0., 0., 0., 1., 1., 1.}, {1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 1., 1.},
   {1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1., 1.}, {1., 1., 1., 1., 0., 0., 0., 0., 0., 1., 1., 1.}, {1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 1.}, {1., 1., 0., 1., 1., 1., 1., 1., 0., 0., 1., 1.},
   {1., 1., 0., 0., 1., 1., 0., 0., 0., 1., 1., 1.}, {1., 1., 1., 1., 0., 0., 0., 0., 1., 1., 1., 1.}, {1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.}, {1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.}}}
```

Can also be applied to files

```
In[200]:= f = File@FindFile["ExampleData/coneflower.jpg"]

Out[200]= File[ /Applications/Mathematica.app/Contents/Documentation/English/System/ExampleData/coneflower.jpg » ]
```

```
In[331]:= Short[imageenc[f]]

Out[331]//Short= {{{0.898039, 0.890196, 0.870588, ≪6≫, 0.658824, 1., 0.611765}, ≪10≫, {0.917647, ≪10≫, ≪19≫}}}
```

- Allows for out-of-core learning on image and audio files!
  - See the tutorial Training on Large Datasets for more
- A large collection of encoders are available for different datatypes
  - “Audio”
  - "Characters"

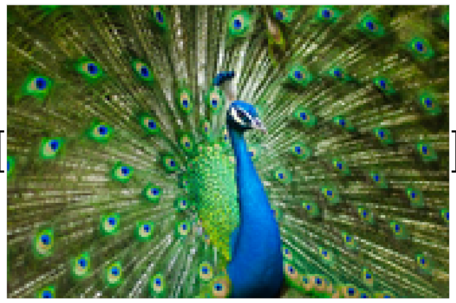
What Are Neural Nets?

5. NetEncoders

Encoders are what allows trained models to be used directly on the type of interest:

In[202]:=
```
net = NetModel["Inception V3 Trained on ImageNet Competition Data"]
```

Out[202]=
```
NetChain[ Input port: image
          Output port: class
          Number of layers: 33 ]
```

In[203]:=
```
net[
```

```
]
```

Out[203]= peacock

In[204]:=
```
NetExtract[net, "Input"]
```

Out[204]= NetEncoder[

Type:	Image
Image size:	{299, 299}
Color space:	RGB
Color channels:	3
Mean image:	{0.5, 0.5, 0.5}
Variance image:	None
Output:	3–tensor (size: 3 × 299 × 299)

]

What Are Neural Nets?

6. NetDecoders

A net will always output a numeric tensor. But for a task like classification, one wants class-labels as output. A **NetDecoder** is a mechanism for returning non-numeric tensors from nets.

In[205]:=
```
dec = NetDecoder[{"Class", {"dog", "cat"}}]
```

Out[205]=
```
NetDecoder[ Type: Class
            Labels: {dog, cat}
            Input depth: 1
            Dimensions: 2 ]
```

This decoder will interpret a vector of probabilities over classes as a class label:

In[206]:=
```
dec[{0.1, 0.9}]
```

Out[206]=
```
cat
```

The probabilities can also be obtained:

In[207]:=
```
dec[{0.1, 0.9}, "Probabilities"]
```

Out[207]=
```
<|dog → 0.1, cat → 0.9|>
```

What Are Neural Nets?

6. NetDecoders

A decoder can be attached to the output of a layer or container:

In[208]:=
```
soft = SoftmaxLayer["Output" -> NetDecoder[{"Class", {"dog", "cat"}}]]
```

Out[208]= SoftmaxLayer[Level: −1 Output: class]

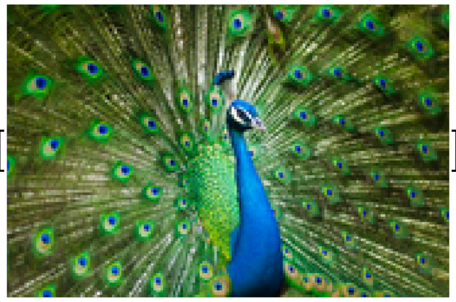
In[209]:=
```
soft[{44, 41}, "Probabilities"]
```

Out[209]= ⟨| dog → 0.952574, cat → 0.0474259 |⟩

This mechanism allows pre-trained nets to output class-labels:

In[210]:=
```
net = NetModel["Inception V3 Trained on ImageNet Competition Data"]
```

Out[210]= NetChain[Input port: image Output port: class Number of layers: 33]

In[211]:= net[]

Out[211]= peacock

In[212]:=
```
NetExtract[net, "Output"]
```

Out[212]= NetDecoder[Type: Class Labels: {other, kit fox, ≪998≫, dumbbell} Input depth: 1 Dimensions: 1001]

What Are Neural Nets?

7. Training

- To train a net, it must have one output, the loss
- Training involves finding parameters to minimize the loss

A very simple example:

```
In[213]:= data = {{1} → {1.9}, {2} → {4.1}, {3} → {6.0}, {4} → {8.1}}
Out[213]= {{1} → {1.9}, {2} → {4.1}, {3} → {6.}, {4} → {8.1}}
```

```
In[214]:= net = NetInitialize@NetGraph[{LinearLayer[1], MeanSquaredLossLayer[]}, {1 → 2}, "Input" → {1}]
Out[214]= NetGraph[ Number of inputs: 2
                    Loss port: real
                    Number of layers: 2 ]
```

Evaluating this net:

```
In[215]:= net[<|"Input" → {2}, "Target" → {4.1}|>]
Out[215]= 42.0742
```

What Are Neural Nets?

7. Training

Train the net:

In[216]:=
```
trainednet = NetTrain[net, data]
```

Out[216]=
```
NetGraph[ Number of inputs: 2
          Loss port: real
          Number of layers: 2 ]
```

The output is now much smaller:

In[217]:=
```
trainednet[<|"Input" → {2}, "Target" → {4.1}|>]
```

Out[217]=
```
0.00999998
```

Training has changed the parameters:

In[218]:=
```
NetExtract[net, {1, "Weights"}]
NetExtract[trainednet, {1, "Weights"}]
```

Out[218]=
```
{{-1.19323}}
```

Out[219]=
```
{{2.05}}
```