Assignment 2

Jacob Taylor Cassady

Deep Learning

March 17, 2020

# RNN Dimensionality

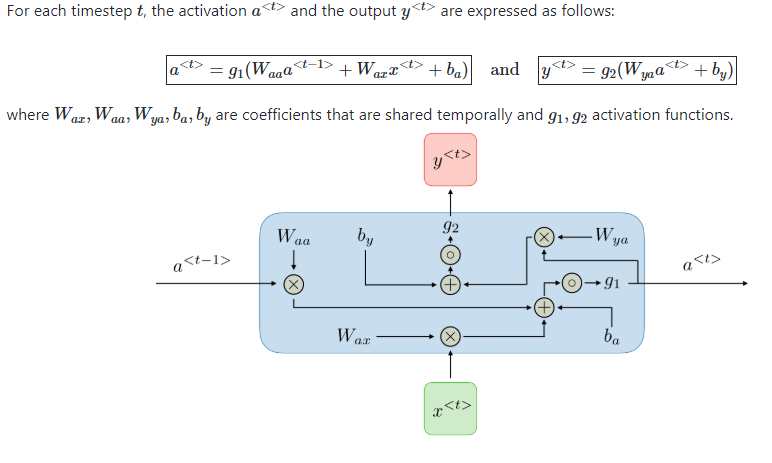


Figure : Model of Recurrent Neural Network (RNN) Layer taken from <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>

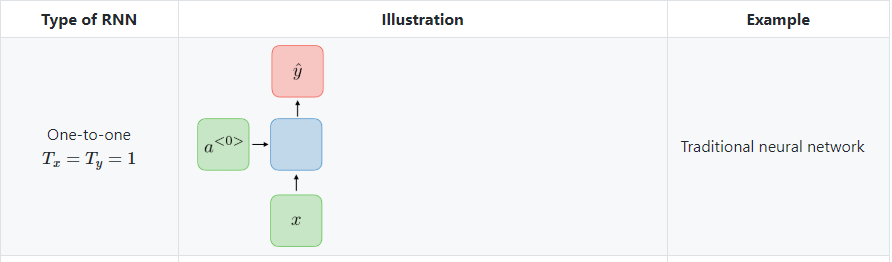


Figure : Model of Single Layer RNN Network taken from <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>

## How many Dimensions must the inputs of an RNN layer have?

Table : RNN Input Dimensions

|  |  |  |
| --- | --- | --- |
| Input | Dimension | Description |
| x(t) | 2D tensor | Input of neuron |
| a(t-1) | 1D tensor | Activations of previous layer or initial state |

## What does each dimension represent?

Table : RNN Input Dimension Representations

|  |  |
| --- | --- |
| Input | Dimension Representation |
| x(t) or input | (timesteps, input\_features) |
| a(t-1) or previous state | (input\_features,) |

## What about its output?

There are two outputs from a traditional RNN layer. They are described in the table below.

Table : RNN Output Dimension Descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| Output | Dimension | Dimension Representation | Description |
| y(t) | 2D tensor | (timesteps, output\_features) | Output of Neuron |
| a(t) | 1D tensor | (output\_features,) | Activations for next RNN layer |

# Consider a CNN composed of three convolutional layers, each with 3x3 kernels, a stride of 2, and some padding. The lowest layer outputs 100 feature maps, the middle one outputs 200, and the top one outputs 400. The input images are RGB images of 200x300 pixels.

## What is the total number of parameters in the CNN?

Table 4: Trainable Parameters Count

|  |  |
| --- | --- |
| Layer | Trainable Parameters |
| Layer\_1 | (3\*3\*3 + 1) \* 100 =2,800 |
| Layer\_2 | (3\*3\*100 + 1) \* 200 = 180,200 |
| Layer\_3 | (3\*3\*200 +1) \* 400 = 720,400 |
| TOTAL | **903,400** |

## If we are using 32-bit floats, at least how much RAM with this network require when making a prediction for a single instance?

Table : Layer Feature Map Sizes

|  |  |
| --- | --- |
| Layer | Feature map Size |
| Layer\_1 | 100x150 |
| Layer\_2 | 50x75 |
| Layer\_3 | 25x38 |

Table : Layer Feature map Memory Sizes

|  |  |
| --- | --- |
| Layer | Feature Map Memory Size |
| Layer\_1 | 4x100x150x100 = 6 MB |
| Layer\_2 | 4x50x75x200 = 2.9 MB |
| Layer\_3 | 4x25x38x400 = 1.4 MB |
| TOTAL | 6 + 2.9 = 8.9 MB |

Total\_Memory = 8.9 MB + (903,400\*4) = 17.8 MB

## What about when training on a mini-batch of 50 images?

The network would require the same amount of memory, 17.8 MB, when training on a mini-batch of 50 images.

# Use transfer learning for large image classification, going through these steps:

## Create a training set containing at least 100 images per class. For example, you could classify your own pictures based on the location (beach, mountain, city, etc.), or alternatively you can use an existing dataset.

The dataset I am using is an image dataset full of American Sign Language (ASL) representations. There are 6 classes representing the first 6 digits in ASL. In total there are 200 images for each class. My initial goal was to use the COVID chest X-ray dataset but there are no where near 100 images for each class just yet.

## Split the data into a training set, validation set, and a test set.

Note: I am reusing some code from a prior online machine learning course I took on coursera in 2018. I will make sure the label the functions I am borrowing like the one below.





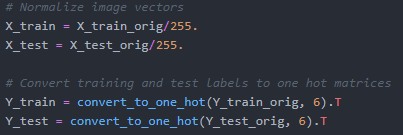
\* I realize the code for splitting the test set into test and validation sets is far from optimal. I was having issues with the shape of my data after this operation and choose a quick fix. \*

Table : Dataset Shapes

|  |  |  |
| --- | --- | --- |
| Dataset | Data Type | Shape |
| Train | Features | (1080, 64, 64, 3) |
| Test | Features | (60, 64, 64, 3) |
| Validation | Features | (60, 64, 64, 3) |
| Train | Targets | (1080, 6) |
| Test | Targets | (60, 6) |
| Validation | Targets | (60, 6) |

## Build the Input Pipeline, including the appropriate preprocessing operations, and optionally add data augmentation.

The only preprocessing done was dividing all pixel values by 255 and converting all class enumerations to one-hot encoding.



## Fine-tune a pretrained model on this dataset



### Training Results

A close up of a map

Description automatically generated

### Test Results

