Assignment 2

Jacob Taylor Cassady

Deep Learning

March 17, 2020

1 RNN Dimensionality

For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

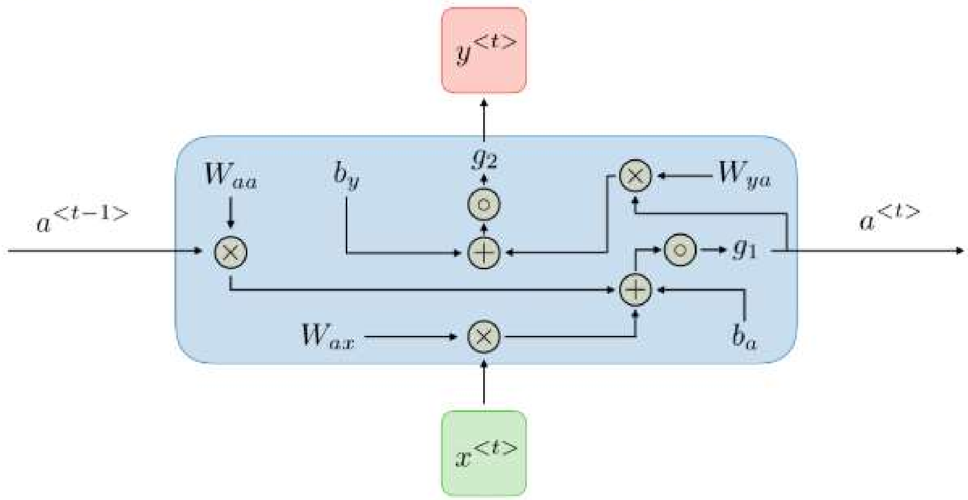


Figure 1: Model of Recurrent Neural Network (RNN) Layer taken from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

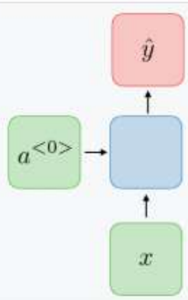
Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network

Figure 2: Model of Single Layer RNN Network taken from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

1.1 HOW MANY DIMENSIONS MUST THE INPUTS OF AN RNN LAYER HAVE?

Table 1: 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

1.2 WHAT DOES EACH DIMENSION REPRESENT?

Table 2: RNN Input Dimension Representations

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

1.3 WHAT ABOUT ITS OUTPUT?

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

Table 3: 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

2 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.

2.1 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**

2.2 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 5: Layer Feature Map Sizes

Layer	Feature map Size
Layer_1	100x150
Layer_2	50x75
Layer_3	25x38

Table 6: 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

2.3 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.

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

3.1 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.

3.2 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.

```python
def load_dataset(relative_directory_path):
    """
    Created By: Jacob Taylor Cassady
    Last Updated: 2/7/2018
    Objective: Load in practice data from example tensorflow model.

    Arguments:
    None

    Returns:
    train_set_x_orig -- A NumPy array of (currently) 1080 training images of shape (64,64,3).  Total nparray shape of (1080,64,64,3)
    train_set_y_orig -- A NumPy array of (currently) 1080 training targets.  Total nparray shape of (1, 1080) [After reshape]
    test_set_x_orig -- A NumPy array of (currently) 200 test images of shape (64,64,3).  Total nparray shape of (120,64,64,3)
    test_set_y_orig -- A NumPy array of (currently) 200 test targets.  Total nparray shape of (1,120) [After reshape]
    classes -- A NumPy array of (currently) 6 classes. (0-5)
    """

    train_dataset = h5py.File(relative_directory_path + 'train_signs.h5', 'r')
    train_set_x_orig = np.array(train_dataset["train_set_x"][:])
    train_set_y_orig = np.array(train_dataset["train_set_y"][:])

    test_dataset = h5py.File(relative_directory_path + 'test_signs.h5', 'r')
    test_set_x_orig = np.array(test_dataset["test_set_x"][:])
    test_set_y_orig = np.array(test_dataset["test_set_y"][:])

    classes = np.array(test_dataset['list_classes'][:])

    train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig, classes
```

```python
def prepare_data():
    relative_directory_path = ".." + os.path.sep + "data" + os.path.sep
    print("\nLoading data from relative directory path:", relative_directory_path)
    X_train_orig, Y_train_orig, X_test_orig, Y_test_orig, classes = load_dataset(relative_directory_path)

    # Normalize image vectors
    X_train = X_train_orig/255.
    X_test = X_test_orig/255.

    # Convert training and test labels to one hot matrices
    Y_train = convert_to_one_hot(Y_train_orig, 6).T
    Y_test = convert_to_one_hot(Y_test_orig, 6).T

    # Split test set into test and valdiation sets
    test_set = np.array(list(zip(X_test, Y_test)))
    test = False

    random.shuffle(test_set)
    X_test = list([])
    Y_test = list([])
    X_validation = list([])
    Y_validation = list([])

    for feature, target in test_set:
        if test:
            X_test.append(feature)
            Y_test.append(target)
            test = False
        else:
            X_validation.append(feature)
            Y_validation.append(target)
            test = True

    return X_train, Y_train, np.array(X_test), np.array(Y_test), np.array(X_validation), np.array(Y_validation)
```

* 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 7: 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)

3.3 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.

```
# Normalize image vectors
X_train = X_train_orig/255.
X_test = X_test_orig/255.

# Convert training and test labels to one hot matrices
Y_train = convert_to_one_hot(Y_train_orig, 6).T
Y_test = convert_to_one_hot(Y_test_orig, 6).T
```

3.4 FINE-TUNE A PRETRAINED MODEL ON THIS DATASET

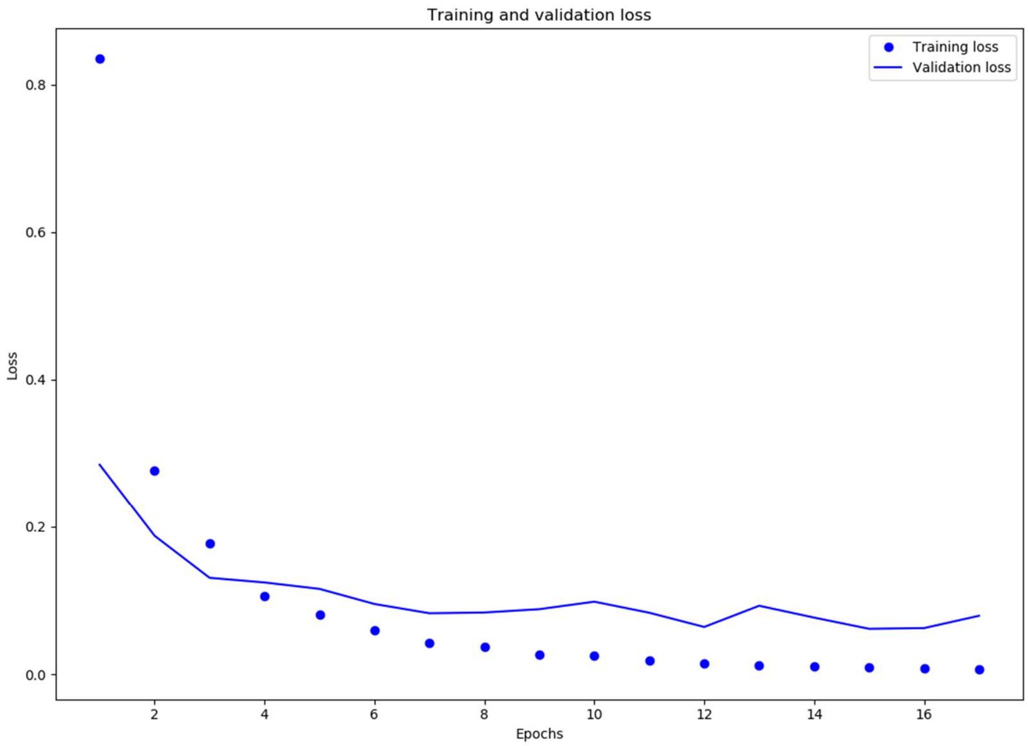
```
if __name__ == "__main__":
    # Retrieve data
    X_train, Y_train, X_test, Y_test, X_validation, Y_validation = prepare_data()

    # Train Model
    model = models.Sequential()
    conv_base = VGG16(weights='imagenet', include_top=False, input_shape=(64, 64, 3))
    conv_base.trainable = False
    model.add(conv_base)
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dense(6, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
    training_history = model.fit(X_train, Y_train, epochs=20, batch_size=32, validation_data=(X_validation, Y_validation))

    # Display Training history graphs
    history_dict = training_history.history
    training_loss_values = history_dict['loss']
    validation_loss_values = history_dict['val_loss']
    epochs = range(1, len(training_loss_values) + 1)
    plt.plot(epochs, training_loss_values, 'bo', label='Training loss')
    plt.plot(epochs, validation_loss_values, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    # Evaluate Model On Test Data
    predictions = model.evaluate(X_test, Y_test)
    print("Test Loss = " + str(predictions[0]))
    print("Test accuracy = " + str(predictions[1]))
```

3.4.1 Training Results



3.4.2 Test Results

```
Test Loss = 0.032941333452860516
Test accuracy = 0.9833333492279053
```