Q1.

```
import tensorflow as tf
data = pd.read_csv('train.csv')
a = tf.placeholder(tf.float32, shape=[None, 16])
b = tf.placeholder(tf.float32, shape=[None, 3])
phi = { 'alpha': tf.Variable(tf.random_normal([16, 8])),
'beta': tf.Variable(tf.random_normal([8, 3])) }
omega = { 'alpha': tf.Variable(tf.random_normal([8])),
'beta': tf.Variable(tf.random_normal([3])) }
tl = tf.add(tf.matmul(x, phi['alpha']), omega['alpha'])
tl = tf.nn.relu(tl)
fl = tf.matmul(tl, phi['beta']) + omega['beta'])
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(fl, b)) izer(learning rate=0.5).min
optimizer = tf.train.AdamOptimizer(learning_rate=0.5).minimize(cost)
init = tf.initialize_all_variables()
```

Refer to the partial code of an ANN implementation using Tensorflow and answer following questions:

a) Which activation function is used in the output layer? Why do you think this particular activation function was chosen? What difference would it make if we used sigmoid function instead?

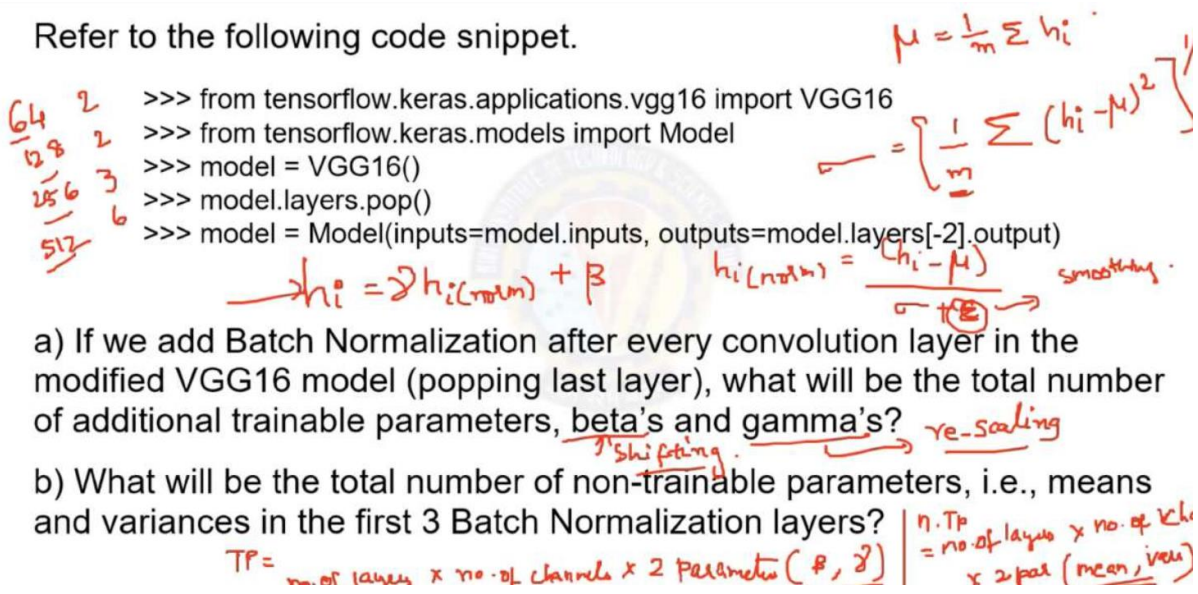
b) Calculate the output values assuming the input vector to the output layer to be [2,0,1,0], weight matrix to be [[0.2,0.3,0.2,0.1], [0.1,0.2,0.5,0.1], [0.2,0.1,0.1,0.1]] and bias vector to be [0.3,0.1,0.1]

Solution:

a) Softmax is used in the output layer. The sum of output values from all the output nodes would be 1 so it is likely that the required output is probability distribution of a given variable. Sigmoid could also be used but the output values will not sum to 1.

b) X.W^T+b = [0.9, 0.8, 0.6]; softmax values = [0.38, 0.34, 0.28]

Q2.



Answer

a) 8448

b) 512 (2*64*2 + 128*2)

Conv layer Channels	No. of layers	Tp		Parameters
64	2	2*64*2 = 256	2*64*2 = 256	512
128	2	2*128*2 = 512	2*128*2 = 512	1,024
256	3	3*256*2 = 1,536	3*256*2 = 1,536	3,072
512	6	6*512*2 = 6,144	6*512*2 = 6,144	12,288
		8,448	8,448	16,896

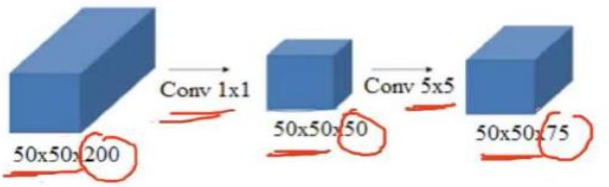
Q3.

In the following figure, 1x1 operators are used first to process the same 50x50 images of depth 200, and first output 50x50 images of depth 50, and then 5x5 operators are used to output 50x50 images of depth 75.

a) What is the padding size used in the first step and padding size in the last step?

b) How many multiplication operations are needed here?

ii.



Answer

a) For 1x1 convolution, padding size = 0.

For 5x5 convolution, padding size = 2.

b) For 1x1 convolution,

# of multiplication operations = 50x50x50x200 = 2.5x107

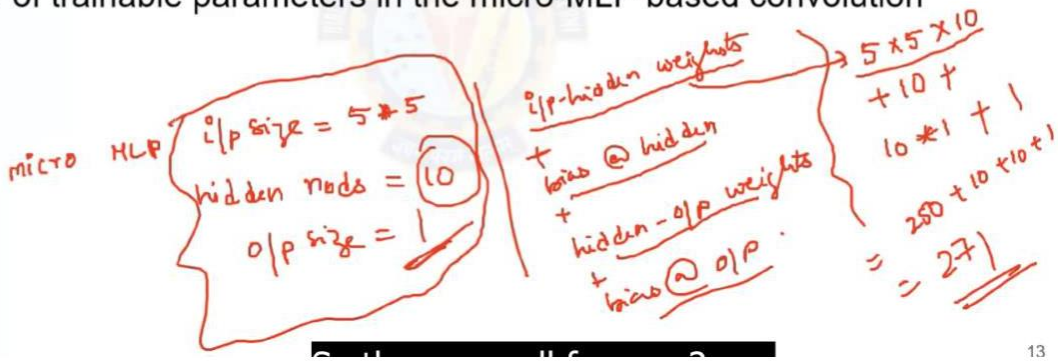
For 5x5 convolution,

# of multiplication operations = 50x50x5x5x50x75 = 234375000

So, total # of operations = 236,875,000

Question

A network-in-network architecture is used to classify gray-scale images of size 64x64 into 100 classes. A micro-MLP with a single hidden layer of 10 nodes is used in the first layer instead of convolution filters of size 5x5 to generate an output feature map of depth (channel) 1. What will be the number of trainable parameters in the micro-MLP based convolution layer?



Answer

Micro-MLP input size: 5*5, Hidden nodes 10, output size 1

Total # of trainable parameters

= 5*5*10 (input-hidden weights) + 10 (bias @ hidden) + 10*1 (hidden-to-output nodes) +1 (bias @ output)

= 271

Question

Which of the following networks is (are) more suited architecturally for classifying images of varied size (e.g., the input image database has images of size 64x64, 96x96, 128x96, etc.)? No image rescaling is permitted.

Answer

Networks that use global average pooling generate feature vectors whose size is independent of input data, after flattening. Resnet, NiN, Inception all use global average pooling.

Question

Consider an LSTM network with one hidden layer of 20 nodes used for predicting the next word in one hot encoded representation in its output . No bias is used in any of the nodes. The corpus is of length 1000 words and there are 100 unique words. Assume a 10 dimensional word embedding module outside of the LSTM network, whose output is fed to the word predictor LSTM network. What will be the total number of trainable weights in the LSTM network?

i/p size = 10
hidden layer = 20
o/p = 100
4 * o/p-dim (inp dim + o/p dim)

4 * (n + m + 1) * m
4 * 2 * (20 + 10) * 20 = 4800
i/p to LSTM
hidden to o/p = 20 * 100 = 2000
=> 6800

Answer

Input size to the network: 10

Number of hidden nodes: 20

Number of output nodes: 100

Total weights from input to LSTM hidden nodes = 4*2*(20+10)*20 = 4800

Total weights from LSTM hidden nodes to output nodes = 20*100 = 2000

Total Weights = 6800

Question

In a network in network architecture, one hot output encoding is used to classify 100 classes of objects. The last convolution layer generates 7x7x128 features maps (depth=128). Assuming the fully connected subnetwork has one hidden layer with 50 nodes, what will be the total number of trainable weights (excluding biases) in the fully connected subnetwork (from the last convolution layer to the output layer)?

Answer

NiN uses global averaging pooling.

After flattening, feature vector size = 128 hitting the FC layer

Total # of weights in FC layer = 128*50+50*100=11400

Question:

Consider a vanilla RNN network with one hidden layer of 20 nodes used for predicting the next word in a text corpus. No bias is used in any of the nodes. The corpus is of length 1000 words and there are 50 unique words. Assume a 10 dimensional word embedding module outside of the RNN network, whose output is fed to the word predictor RNN network. One hot encoding is used in the output. What will be the total number of trainable parameters in the RNN network?

Answer

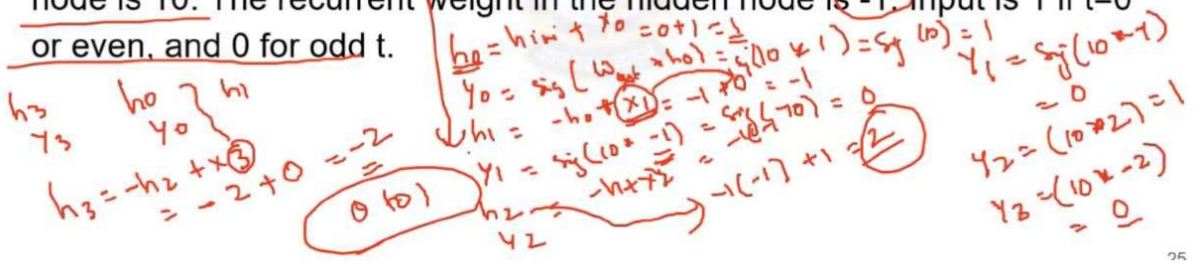
Input-to-hidden weights = (20+10)*20

Hidden-to-output weights = 20*50

Total weights = 1600

Question

Consider a RNN where the hidden state equation is h(t+1) = W h(t) + x (t+1) . Calculate and plot the output y(t) of a single hidden node and single input/output RNN, where activation function used in hidden and output nodes are, respectively, linear and sigmoid. Assume that the initial hidden state is 0, and the input sequence is of even length. Assume also that all biases are 0. Weight from input to hidden node is 1 and hidden to output node is 10. The recurrent weight in the hidden node is -1. Input is 1 if t=0 or even, and 0 for odd t.



Answer

h0 = h(initial) + x0 = 0 + 1 = 1

y(0) = sig(wout * h0) = sig(10 * 1) = sig(10) = 1

h1 = −h0 + x1 = −1 + 0 = −1

y(1) = sig(wout * h0) = sig(10 * −1) = sig(−10) = 0

h2 = −h1 + x2 = −1(−1) + 1 = 2

y(1) = sig(wout * h0) = sig(10 * 2) = sig(20) = 1

h3 = −h2 + x3 = −2 + 0 = 2 = −2

y(1) = sig(wout * h0) = sig(10 * −2) = sig(−20) = 0

The value of y will be close to 1 when t is even and close to 0 when t is odd. The value will oscillate from 1 to 0 and 0 to 1 for each time stamp starting from t=0. The graph will oscillate between 0 and 1.

Question: Explain the working of the code snippet given below.

```
model = keras. models. Sequential()
model. add (keras.layers. Embedding (input_dim = 20000, output_dim = 128))
model.add (keras.layers.LSTM (128, dropout =0.2))
model.add (keras.layers. Dense (1, activation='sigmoid'))
model. summary ()
```

Answer:

This code snippet creates a sequential neural network model using the Keras API with the following architecture:

1. **Embedding Layer**:
   - The first layer added to the model is an Embedding layer. This layer is used for text processing tasks and is typically applied to sequences of integers representing words.

   - Parameters:
     - `input_dim`: Specifies the size of the vocabulary, i.e., the maximum integer index + 1. In this case, it's set to 20000, indicating that the model expects input sequences composed of integers ranging from 0 to 19999.
     - `output_dim`: Specifies the dimension of the embedding vector for each word. In this case, it's set to 128, meaning each word will be represented as a vector of length 128.

2. **LSTM Layer**:
   - The second layer added is an LSTM (Long Short-Term Memory) layer. LSTM is a type of recurrent neural network (RNN) architecture that is well-suited for sequence prediction problems, especially in natural language processing tasks.

   - Parameters:
     - `128`: Specifies the number of units (or neurons) in the LSTM layer.
     - `dropout`: Specifies the dropout rate, which is a regularization technique to prevent overfitting by randomly dropping a fraction of input units during training. Here, it's set to 0.2, indicating that 20% of the input units will be randomly dropped during training.

3. **Dense Layer**:
   - The final layer added is a Dense layer. This is a fully connected layer where each neuron is connected to every neuron in the previous layer.

   - Parameters:
     - `1`: Specifies the number of units in the Dense layer, which is 1 in this case. Since the task involves binary classification (sigmoid activation function is used), a single unit is sufficient to produce the binary output.
     - `activation='sigmoid'`: Specifies the activation function to be used in the Dense layer. The sigmoid activation function is chosen here, which squashes the output values between 0 and 1, making it suitable for binary classification tasks where the output represents probabilities.

4. **Model Summary**:
   - Finally, the `summary()` method is called on the model to display a summary of the model architecture, including the type of each layer, output shape, and number of parameters.

Question

For an image classification problem which classifies images into two categories, 128 by 128 pixel colour images is fed through four convolution layers with 32, 64, 128, 256 kernels of 5 by 5 respectively with max pooling for each layer. Then the tensors are fed through two layers of fully connected neurons with 1024 and 512 neurons.

(a) Draw a CNN for the above.

(b) Write Tensorflow-Keras code snippet for the above.

(c) Compute the number of parameters learned in each convolution and maxpooling layers.

Answer:

Sure, let's tackle each part one by one:

(a) Here's a rough sketch of the CNN architecture described:

Input (128x128x3)
|
Conv2D (32 filters, 5x5)
|
MaxPooling2D
|
Conv2D (64 filters, 5x5)
|
MaxPooling2D
|
Conv2D (128 filters, 5x5)
|
MaxPooling2D
|
Conv2D (256 filters, 5x5)
|
MaxPooling2D
|
Flatten
|
Dense (1024 neurons)
|
Dense (512 neurons)
|
Output (2 classes)

(b) Here's the TensorFlow-Keras code snippet for the described architecture:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()
```

```
# Convolutional layers
model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(128, 128, 3)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(256, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Fully connected layers
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(512, activation='relu'))

# Output layer
model.add(Dense(2, activation='softmax'))  # Assuming binary classification

model.summary()
```

(c) Let's calculate the number of parameters learned in each convolution and maxpooling layers:

1. **Convolutional Layer 1**:
   - Parameters learned:
     - Number of filters: 32
     - Filter size: 5x5
     - Input channels: 3 (RGB)
   - Total parameters: (5 * 5 * 3 + 1) * 32 = 2432

2. **MaxPooling Layer 1**:
   - MaxPooling layers do not have any trainable parameters. They only perform downsampling.

3. **Convolutional Layer 2**:
   - Parameters learned:
     - Number of filters: 64
     - Filter size: 5x5
   - Total parameters: (5 * 5 * 32 + 1) * 64 = 51264

4. **MaxPooling Layer 2:**
- No trainable parameters.

5. **Convolutional Layer 3:**
- Parameters learned:
- Number of filters: 128
- Filter size: 5x5
- Total parameters: (5 * 5 * 64 + 1) * 128 = 204928

6. **MaxPooling Layer 3:**
- No trainable parameters.

7. **Convolutional Layer 4:**
- Parameters learned:
- Number of filters: 256
- Filter size: 5x5
- Total parameters: (5 * 5 * 128 + 1) * 256 = 819456

8. **MaxPooling Layer 4:**
- No trainable parameters.

In summary, we calculated the number of parameters learned in each convolutional layer, while maxpooling layers don't have any parameters as they perform downsampling only.