Feb 14, 2024

AI6126 – Advanced Computer Vision Assignment # 1

Reinelle Jan Bugnot
G2304329L

Question 1: Consider a multivariate regression problem in which we predict the height of an individual in meters and their weight in kilos from some data **x**. Here, the units take quite different values. What problems do you see this causing? Propose two solutions to these problems.

The distribution of the values for height (in meters) and weight (in kilos) differ significantly. For example, typical height values may range from 1.5 to 2 meters while weight values range from 50 to 100 kilos. This scale disparity between the two target variables in this multi-target multivariate regression problem can cause numerical instability during model training, leading to **slower convergence** of the optimization algorithm. This is because the scale disparity disproportionately influences the loss function during training, making the model prioritize minimizing the errors in the prediction of the *weight* target variable since it is of much larger scale that the *height* variable, and therefore, has greater numerical impact in the calculation of the loss value (e.g. MSE).

Two solutions I propose is (1) to apply **normalization** or **standardization** across the target variables value to equalize the value distributions of the two target variables, or (2) **decouple the prediction model** by training two separate models to predict *height* and *weight* using the same training data. This approach allows the model to independently learn the relationships between the input features and the respective target variable.

Question 2: A convolutional network was trained using stochastic gradient descent with a learning rate of 0.01 and a batch size of 100 on a training dataset of 4,000 examples for 100,000 steps. How many epochs was the network trained for?

A training pipeline with 4,000 training examples on an SGD optimization algorithm with a batch size of 100, means there are 40 batches per epoch. So if there are 100,000 steps in total, this indicates that the model was trained for **2,500 epochs.**

Question 3: A network with the type of each layer and the corresponding output shape is given in Figure Q3.

```
----------------------------------------------------------------
        Layer (type)               Output Shape
================================================================
            Conv2d-1            [-1, 6, 28, 28]
              ReLU-2            [-1, 6, 28, 28]
         MaxPool2d-3            [-1, 6, 14, 14]
            Conv2d-4           [-1, 16, 10, 10]
              ReLU-5           [-1, 16, 10, 10]
         MaxPool2d-6             [-1, 16, 5, 5]
            Conv2d-7            [-1, 120, 1, 1]
              ReLU-8            [-1, 120, 1, 1]
            Linear-9                  [-1, 84]
             ReLU-10                  [-1, 84]
           Linear-11                  [-1, 10]
       LogSoftmax-12                  [-1, 10]
================================================================
```

Figure Q3

The input has a shape of 1x32x32. The output shape of each layer is provided as [<ignore>, output channels, height, width]. For instance, at layer 'Conv2d-1', the output shape is [6, 28, 28], i.e., six feature maps of spatial size 28x28. Each conv filter and neuron of linear layer has a bias term and stride = 1. No padding is assumed.

Calculate the number of parameters for each layer and finally the total number of parameters of this network.

The table below summarizes the calculation of the number of parameters for each layer in the provided network.

TABLE I. Calculation of Number of Parameters per Layer

Input Size: [-1, 1, 32, 32]

Layer Name	Layer Shape	Intermediate Values	Total Parameters (P)
Conv2d-1	[-1, 6, 28,28]	# of Filters (F): 6 *Kernel Size (K): 5x5 Input Channels (c): 1 Bias (b): 6	$P = F * K * c + b$ = 6 * (5 * 5) * 1 + 6 = **156**
ReLU-2	[-1, 6, 28, 28]	No parameters	No parameters
MaxPool2d-3	[-1, 6, 14, 14]	No parameters	No parameters
Conv2d-4	[-1, 16, 10, 10]	# of Filters (F): 16 Kernel Size (K): 5x5 Input Channels (c): 6 Bias (b): 16	$P = F * K * c + b$ = 16 * (5 * 5) * 6 + 16 = **2,416**
ReLU-5	[-1, 16, 10, 10]	No parameters	No parameters
MaxPool2d-6	[-1, 16, 5, 5]	No parameters	No parameters
Conv2d-7	[-1, 120, 1, 1]	# of Filters (F): 120 Kernel Size (K): 5x5 Input Channels (c): 16 Bias (b): 120	$P = F * K * c + b$ = 120 * (5 * 5) * 16 + 120 = **48,120**

ReLU-8	[-1, 120, 1, 1]	No parameters	No parameters
Linear-9	[-1, 84]	**Input Neurons (i): 120 Output Neurons (o): 84 Bias (b): 84	$P = i * o + b$ = 120 * 84 + 84 = **10,164**
ReLU-10	[-1, 84]	No parameters	No parameters
Linear-11	[-1, 10]	Input Neurons (i): 84 Output Neurons (o): 10 Bias (b): 10	$P = i * o + b$ = 84 * 10 + 10 = **850**
LogSoftax-12	[-1, 10]	No parameters	No parameters

*****The Kernel size for the convolutional layers was calculated using the following formula [1]:

$$N_{out} = \frac{N_{in} - K + 2P}{S} + 1 \quad (1)$$

where: N_{out} = Size of output feature map

N_{in} = Size of input image / feature map

K = Size of Kernel / Filter

P = Size of Padding

S = # of Strides

****** This value is taken directly from the total number of neurons or output nodes from the previous layer.

Total Number of Parameters: 156 + 2,416 + 48,120 + 10,164 + 850 = **61,706**

Question 4: Figure Q4 depicts a block that consists of three convolutional layers. The input volume has a size of 256 × 32 × 32 and the second layer has 32 convolution filters each with a size of 64 × 3 × 3, stride = 1 and padding = 1.

Provide the values of n_1, d_1, F_1, n_2, d_2, and F_2 to form a valid block. Explain your design.

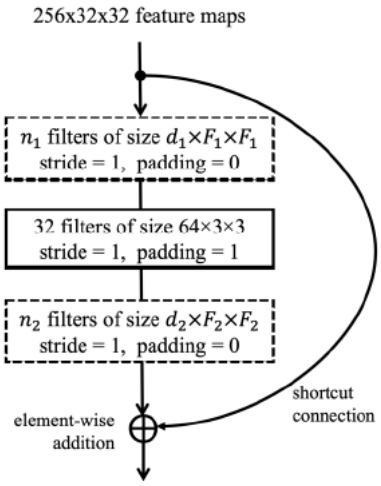

Figure Q4

To for a valid block, the output volume of each layer must match the input volume of the succeeding layer, or the shortcut connection. Based on the diagram, I can set the following variables to the following values to simplify later calculations:

$$\boldsymbol{d_1 = 256}$$

$$\boldsymbol{n_1 = 64}$$

$$\boldsymbol{d_2 = 32}$$

$$\boldsymbol{n_2 = 256}$$

By matching the depth d_k of the filter volumes with the depth of the preceding layer's output volume, we ensure that each filter's output is with a depth of 1; and therefore the depth of the layer's output volume is determined by the number of filters n_k.

Given these values and using Eq. (1), we can derive three equations with 4 unknowns:

$$32 = O_2 - F_2 + 1 \tag{2}$$

$$O_2 = O_1 - 3 + 2 + 1 \tag{3}$$

$$O_1 = 32 - F_1 + 1 \tag{4}$$

where:

O_1 – Output dimension of the 1st Layer
O_2 – Output dimension of the 2nd Layer
O_3 – Output dimension of the 3rd Layer

We have 4 unknowns and 3 equations. We can set F_1 as a parameter and select a value based on our design intuition. Note however that our selection is bounded by the following constraints:

$$F_2 > 0 \tag{5}$$

$$F_1 > 0 \tag{6}$$

From Eq. (5),

$$F_2 > 0$$

From Eq. (2),

$$F_2 = (O_2 - 32 + 1) > 0$$

From Eq. (3),

$$F_2 = (O_1 - 3 + 2 + 1) - 32 + 1 > 0$$

From Eq. (4),

$$F_2 = (32 - F_1 + 1) - 3 + 2 + 1 - 32 + 1 > 0$$

Simplifying,

$$F_2 = 2 - F_1 > 0$$

Which gives us,

$$F_2 = 2 - F_1 \quad (7)$$

$$F_1 < 2 \quad (8)$$

The only valid value we can select for F_1 such that Eq. (8) and Eq. (6) holds true is $\boldsymbol{F_1 = 1}$. And hence, $\boldsymbol{F_2 = 1}$.

.

Question 5: Can batch normalization have a regularization effect on the training of deep learning models? Explain your answer with reference to how batch normalization is applied.

Although not originally intended, Batch Normalization was observed to have a regularizing effect on the training of deep learning models. During the training of a neural network, the activations of subsequent layers may change as preceding layers are updated. Batch normalization essentially standardizes the inputs to a layer for each mini-batch using the mean and standard deviation of the mini-batch. After normalization, batch normalization introduces learnable parameters (scale and shift) for each activation dimension. These parameters allow the model to adaptively scale and shift the normalized activations, providing the network with flexibility in representation. This is demonstrated in the equations shown in the figure below. In effect, Batch Normalization reduces internal covariate shift, leading to faster model convergence.

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$ Per-channel mean, shape is $1 \times D$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$ Per-channel variance, shape is $1 \times D$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$ Normalize the values, shape is $N \times D$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ **Scale and shift** the **normalized** values to add flexibility, output shape is $N \times D$

Learnable parameters: γ and β, shape is $1 \times D$

Fig. 1. Math behind Batch Normalization [1]

Each mini-batch used for training are sampled randomly from the entire training dataset. This means the mean and standard deviation of each mini-batch are relatively randomized values that depend on how the mini-batch was sampled. Because of this,

noise is introduced to each layer through Batch Normalization. **This noise can act as a regularizer** by making the activations of the hidden layers less deterministic, preventing overfitting and improving generalization in the long run.

Question 6: How is batch normalization utilized in ResNet architectures? Discuss its placement.

Batch Normalization is generally important (if not presently required) for training extremely deep neural networks because of its capacity to eliminate internal covariate shift, which happens when the distribution of output values from earlier layers change as the weights are updated. This change in distribution affects the training process because it forces higher layers to adapt to this drift, slowing down learning [2]. Applying Batch Normalization also allows for the use of higher learning rates which helps the model to converge faster. Typically in ResNets, the Batch Normalization layer is placed after each convolutional layer and before the activation functions, as shown in the Figure below.

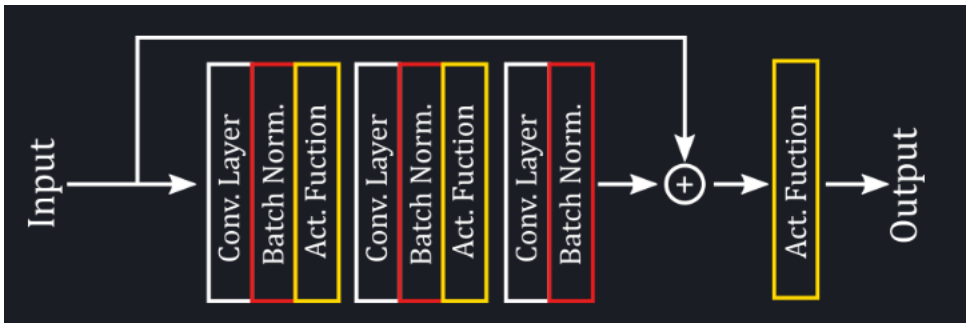

Fig. 2. ResNet Block. The BN layer is placed after CONV layers and before the activation function [3].

Question 7: The 'Net' class in Tutorial 1a 'MNIST-2D Classification' defines a simple convolutional neural network. Redesign the class, including the constructor (__init__) and the forward pass (forward), so that it implements the following layers. You just need provide the code snippet that define the class in your answer. Provide the output returned by the print(model) too.

```
# 1. A valid convolution with kernel size 5, 1 input channel and 10 output channels
# 2. A max pooling operation over a 2x2 area
# 3. A ReLU
# 4. A valid convolution with kernel size 5, 10 input channels and 20 output channels
# 5. A 2D Dropout layer
# 6. A max pooling operation over a 2x2 area
# 7. A ReLU
# 8. A flattening operation
# 9. A fully connected layer mapping from (whatever dimensions we are at -- find out
using .shape) to 50
# 10. A ReLU
# 11. A fully connected layer mapping from 50 to 10 dimensions
# 12. A SoftMax function.
```

The Modified class constructor code snippet is shown below.

```
from os import X_OK

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.drop = nn.Dropout2d()
        self.fc = nn.Linear(50, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.max_pool2d(x,2)
        x = F.relu(x)

        x = self.conv2(x)
        x = self.drop(x)
        x = F.max_pool2d(x,2)
        x = F.relu(x)

        x = x.flatten(1)

        x = nn.Linear(x.shape[1], 50)(x)
        x = F.relu(x)
        x = self.fc(x)
        x = F.log_softmax(x)

        return x
```

```
print(model)
```

```
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (drop): Dropout2d(p=0.5, inplace=False)
  (fc): Linear(in_features=50, out_features=10, bias=True)
)
```

Fig. 3. Code snippet for the modified PyTorch Class.

References

[1] Chen-Change Loy, *Convolutional Neural Network II*, Week – 3 Lecture notes, Nanyang Technological University, Singapore.

[2] Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ArXiv:1502.03167 [Cs]. http://arxiv.org/abs/1502.03167

[3] Batch Normalization and ResNets (Figure)