ML2 Coursework 2

nh693

March 22, 2024

1 Introduction : 5

1.1 a) Specify in mathematical terms the desired criteria for this problem (e.g., how do you express in mathematical terms that "the resulting image C should look almost visually indistinguishable from B "?)

The simplified objective is to conceal an image A into another image B to produce the composite image C. This however demands a few key concepts. We demand that there is an concealling function $f : (A, B) \to C$ which takes in the secret message/image, A, composes it with the cover image, B, and gives the resultant image C. Further, a deconstruction function, D, s.t. $D(C) = U$ where U is known as the unveiled image. For the perfect system, we would hope that $U = A$ and $D = f^{-1}$ so that we may write, $C = f(A, B)$, $U = D(C) = D(f(A, B)) = f^{-1} \circ f(A, B) = A$. However, we may never have equality but instead we may aim to come as close as possible to the equality such that we can say $A \approx U$ due to the inherent loss of accuracy during the concealing and decoding methods of the system once we optimise the loss/cost functions of the system.

As just mentioned, we seek to construct loss functions in order to minimise them and obtain a starting point for our problem. The first loss function of which can be thought of as the loss associated with the Visual Similarity (of B and C), which may take the form of the Mean Squared Error, MSE, (helpful for Neural Network, NN, later due to the generally convex nature of the MSE loss),

$$L_{ViusalSimilarity}(B, C) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (B_{ij} - C_{ij})^2 \tag{1}$$

Where $A, B \in \mathbb{R}^{n \times m}$ and $n, m \in \mathbb{N}$. Similarly, we define, the reconstruction Loss,

$$L_{reconstruction}(C, U) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (C_{ij} - U_{ij})^2 \tag{2}$$

Where $C, U \in \mathbb{R}^{n \times m}$ and $n, m \in \mathbb{N}$.

Hence we may consider an objective function that can be formulated as the minimisation of the linear combination of the two loss functions.

$$z^* = \min\{\mu L_{VisualSimilar}(B, C) + \lambda L_{reconstruction}(A, D(C))\} \tag{3}$$

where $\mu, \lambda \in \mathbb{R}^+$ are balancing hyper parameters to be determined at a later stage depending on the importance whether we demand a more concealed image or a more accurate answer.

For the resulting image C to be visually indistinguishable from B we seek to minimise equation 1.

1.2 b) Suitability of Deep Learning

Deep Learning is very suitable for the following reasons: Convoluted Neural Networks (CNN's), are fantastic at identifying and hence subsequently extracting complex features within data sets such as images as highlighted by the classic example of LeNet-5 to more advanced multi classification CNN's. Further, Deep Learning (DL) models are highly effective at optimising complex functions (like loss functions defined above) with the use of backpropagation and gradient descent methods. Finally, with adequate training, hyper parameter selection, optimisation strategies (such as momentum based learning) and Regularisation techniques, Deep Learning models can adapt well to unseen data making them capable and robust to many different types of input data A or B, and suitable for real world application if created correctly.

2 Data Handling : 10

2.1 a) Data Specifications

Trivially, the dimension of A must not be larger than the dimension of B $dim(A) \leq dim(B)$, otherwise we have more data to encrypt than space to encrypt it to! There are ways to increase the dimensions (blowing up the image) to account for this but

this must be applied before the network is handling the data. This implies that the dimension of B upper bounds the maximum data to be concealed by A. Resizing, otherwise known as standardising, A and B to ensure consistency and efficiency in the models as well as working around the previous problem of $dim(B)$ being too small. Resizing A and B to fixed dimensions of common sizes (128x128 for example), should ensure adequate resolution and hence dimension allowance of A. Moreover, data types of .jpeg or other lossy compression types are an issue and may cause data to be inadvertently lost, hence, sticking to lossless compression data types such as .png is advised. It is important to note that image C must obey many of the above mentioned comments to ensure no data is lost of our concealed data A. During the CNN process, use of padding and pooling are common practice however care must again be taken to not inadvertently lose data of A with pooling procedures (see 3.2 below for more). Rather padding is advised for resizing to not lose any data and therefore not compromising the quality of the concealed image.

2.2 b) Org. Data for Training, Validation and Testing

A suitable ratio of the data should be used for each set, such as 70:15:15 Training:Validation:Test. This is to ensure that there is a suitable portion of the data available to teach the model how to conceal image A into image B, tune hyper parameters (such as the learning rate, number of epochs, regularisation parameters, optimiser, dropout rate or even the architecture of the CNN), and evaluating the model's performance on unseen data with the test set.

Additionally, Data Augmentation such as rotation, scaling or colour adjustments to increase the robustness of the model and prevent over fitting should be considered when loading the training data to improve the robustness and ability to perform in the real world cases whilst countering overfitting.

More on the topic of preventing overfitting is the bias variance trade off topic that is brought to light by including more diverse data to ensure a well generalised model and ensures strong performance across types of data and scenarios, generally shuffling the data before splitting into subsets is a safe way to ensure there is no bias in the different sets to aid the above points.

As already mentioned in 2.a), prepossessing of the data such as standardisation of the data as well as normalising the pixel values of all the data to a uniform range, is highly important. This further backs up the point that care must be taken to ensure data images are suitably formatted for the CNN in order not to lose critical information via resizing issues mentioned above in 2.a).

3 Model : 18

3.1 a) Deep Learning Architecture

For the concealing task at hand, as mentioned CNNs are most suitable due to brilliantly processing image data and handling complex patterns. Per layer in the CNN, we are able to detect subtle features in the cover image, B, which is essential for concealing the secret image, A. This key point is how we would aim to maintain the difference between B and C for the reverse process of extracting A. Precisely, the hierarchical feature extraction capabilities of CNNs make them most suited for the task at hand. This ability to recognise spacial relationships within images is not present in feed forward neural networks and hence provides a unique selling point to CNNs. Furthermore, while handling high-dimensional image data CNNs are more efficient than their fully connected feed forward N.N. counterparts. This efficiency is stemming from the reduction in parameters required in CNNs over FFNNs, with the capabilities of parameter sharing and sparse connectivity.

Precisely, there are three reasons CNN's should outperform other NN's on this task. Due to the Sparse connectivity allowing for drastically less parameters to be used implying a much more efficient system. Parameter sharing allowing for the same parameter to be used more than once effectively only learning one set of parameters at each location allowing for a further efficient system. Finally, translation invariance, which allows for translation of the input also allowing the activations of the network to be translated in the same way which once again reduce memory requirements of the system. Reduction in memory requirements and efficiency should be highly valued in this scenario as image data can easily have very high dimensions with high quality images implying large data structures in our NN's very quickly.

3.2 b) Spec. of the CNN Architecture

Depth and Width are very important to ensure we capture all the intricate details and efficiency of the network, however it can be hard to know the exact amount of layers or filters at first and are considered hyper parameters to be potentially changed in the validation set stage. However, without knowing the complexity of the data, starting with approximately 4 layers in depth should provide sufficient to be able to capture complex patterns in the data (necessary for the concealing and decoding part of the process).

The layers should certainly contain convolutional layers as discussed, however we should be very limited in using pooling layers in order to limit the amount of loss (reduction in resolution) of the concealed image and hence inadvertently reduce the accuracy in the unveiled image, U. Opting for 1 pooling layer prioritises the detailed data required for reconstruction over efficiency in this case however, therefore we should look to make our network more efficient in other aspects (discussed below).

On a related note, having a few dozen filters that double per layer starting with 16 and ending with 128 should enable the network to extract all the relevant features, essential for constructing an accurate U, without overwhelming the network with

parameters. The hope is that this conservative amount of layers should strike a balance between complexity and efficiency whilst still having sufficient accuracy.

It is important to remember that this CNN may not typically require a fully connected layer at the end due to the output being an image rather than a classification problem. The output layer must be the same dimension as the input image assuming $dim(A) = dim(B)$, and hence this final touch is critical to ensure we don't inadvertently mismatch data but present out extracted data correctly.

Filters (kernel) in the convolutional layers are key to detect specific types of features at various locations in the data which is very important for the feature extraction and concealing process. Importantly, the smaller the filter the more detailed the features are. Therefore, without knowing how complex A is in terms of features, we may seek smaller filters (3x3) to safeguard ourselves from missing detailed data about the image hence relating to higher potential accuracy in U.

Closely linked, the stride of the network should be set to 1 for assumption of complex A, in order to apply the filter every 1 pixel over. This ensures no loss of feature detection, however this does increase computational complexity along with small filter size, however given the task at hand I believe it is paramount not to lose data on A hence setting stride to 1 and having small filters is key.

As already mentioned, padding should be used to ensure same output dimensions without compromising on loss of data with other methods. In terms of activation functions, ReLU is sufficient due to its efficiency at large computational scale against its competitors which is present due to the above choices of structure. Pooling layers should be kept over a small window size (2x2) for the same logic as filters by not wanting to lose much data about A, but also reducing the dimension of the feature maps contributing to the efficiency of the model.

In all, these choices should aim to balance intricacies of the features from A while aiming to promote efficiency in ways to retain as much data as possible. However as mentioned, further adjustments may be needed of the hyperparameters in the validation stage to improve accuracy or efficiency of the networks.

3.3 Provide a Python code snippet for the PyTorch class implementing the forward method of the neural network you devised.

```python
import numpy as np
import torch
import torch.nn as nn

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        #convolusions
        self.conv1 = nn.Conv2d( in_channels=3, out_channels = 16, kernel_size =3 , padding =  1)
        self.conv2 = nn.Conv2d( in_channels=16, out_channels = 32, kernel_size =3 , padding =  1)
        self.conv3 = nn.Conv2d( in_channels=32, out_channels = 64, kernel_size =3 , padding =  1)
        self.conv4 = nn.Conv2d( in_channels=64, out_channels = 128, kernel_size =3 , padding =  1)

        #pool
        self.pool = nn.MaxPool2d(kernel_size=2,stride= 2)
        #relu
        self.relu = nn.ReLU()
        #output to account for the rgb and dimension
        self.output_conv = nn.Conv2d(128, 3, kernel_size=1, stride=1) #assuming 128x128x3 dim

    def forward(self,x):

        x = self.relu(self.conv1(x))
        x = self.pool(self.relu(self.conv2(x))) #only use of pooling
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.output_conv(x)

        return(x)

```

See figure 3.3.

4 Training : 22

4.1 (a) Specify the loss (and cost) function and the learnable parameters, explaining why you chose this loss (for example, why do you expect to work well for this problem?) and which favourable mathematical properties it has. [5]

As already defined in subsection 1.1, the loss function would take the form of equation 1 and the cost function would take the form of the objective function shown in equation 3. The loss function takes the form of the Mean Squared Error of the two pixel-wise differences between B and C for equation 1 and for C and U for equation 2. The choice is supported by the differentiablity of the loss function which is crucial during backpropagation which is also a convex function meaning there should be only one local minima which is also the global minima. As our objective is to conceal A as best as possible but resolve U as best as possible, we seek to find a middle ground of the already defined equation 3 where λ, μ are (hyper)parameters to be found/altered depending on what process may take precedence. On the topic of parameters, the CNN would have many learnable parameters of weights and biases of the layers of the CNN. These parameters are to be found and adjusted during the training phase of the process and are representing the importance of different features from the data in order to piece it together once again in the unveiled image U.

4.2 b) Specify which optimisation algorithm you will use for the training phase, choosing among the algorithms covered during the course, and explain why it is amenable to the loss function you devised. Discuss parameter choices related to the algorithm of your choice (e.g., how do you select the learning rate and how do you initialise the learnable parameters)

Given the problem at hand, a suitable choice for the optimisation could be Adam (Adaptive movement estimation) which strikes a dynamic way to balance robustness and efficiency. The adam optimiser is efficient for problems with large datasets and parameters as it handles sparse gradients efficiently which comes into play effectively given the high dimensions at play with image data and sparseness of deep layered CNNs. One of the key advantages of this optimiser is the ability to compute adaptive learning rates for each parameter in the system which should aid the balance of bias and variance which is crucial for an optimal result without overfitting or underfitting. Another note is that Adam is implementing a momentum based component which may aid in accelerating the convergence in the right direction, even though our cost and loss functions are convex this is still a smart move as the momentum based component of Adam will aid the convergence when the gradient is small as it approaches the minima, reaching the optimal solution faster. The only downsides to this choice of algorithm is that it is not the least computationally expensive when compared to the likes of stochastic gradient descent, which reduces efficiency of the model, especially for very large data sets. Especially since the loss/cost function is convex there could be reason to use a less computationally expensive model but with the benefits of adaptive learning rates I believe this is the correct implementation. Adam initial parameter values of learning rate $= 0.001$ and $\beta_1, \beta_2 \in [0, 1)$ such as the standard $\beta_1 = 0.9, \beta_2 = 0.99$ should need no further tuning.

4.3 c) In addressing the above points, consider the various challenges training poses and discuss if any of the possible solutions we covered in the course (e.g., regularisation, early stopping, data augmentation) would be sensible and/or necessary within the deep learning strategy you are devising.

There are certainly challenges encountered in the training stage of the process, primarily to counter overfitting (the model performing well on the training but not as well on the validation or test data) but not exclusively. Regularisation techniques such as L2 regularisation (weight decay) would aid in preventing overfitting as the training procedure continues by helping to penalize large weightings, hence simplifying the model and reducing overfitting, improving accuracy in U. Care must be taken not to have too large of a decay however due to the use of ReLU once the weight is zero (or less than) relu will not revive this weight. Just as important is Early Stopping, this process involves monitoring the models performance on the validation set and stopping the training process as the performance starts to degrade. This further aids in the ability of the model to respond to data outside of the training set as can be expected with real world use. Finally data augmentation such as including rotated training set examples, color adjusted or scaled to further improve the models ability in the real world examples.

In all, these processes are key to ensure optimal performance in real world application and inclusion of data augmentation further enhances the non biassness due to the further inclusion of different data of the training data as mentioned earlier.

```python
import torch.optim as optim
# define our cnn class
model = CNN()
criterion = nn.MSELoss()  # define mse use
#adam optimiser, beta values and learning rate as defined in text, last arg is for l2 reg.
optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.99),weight_decay=1e-5)
nEpochs = 10 # total number of epochs
EpochEpsilon = 3 # number of epochs to wait before calling early stop if no improvement
#training loop begins here
for epoch in range(nEpochs):
    model.train()  # train mode on at start of each epoch
    for inputs, labels in trainLoader:  # looping through train data
        optimizer.zero_grad()  # remember to clear old gradients every loop
        outputs = model(inputs)
        loss = criterion(outputs, labels)  # calc mse loss
        loss.backward()  # backprop
        optimizer.step()  # update model params

        # early stopping based on validation loss
        with torch.no_grad():
            model.eval()  # validation set mode
            valLoss = 0
            for valInputs, valLabels in valLoader:  # going through all val data like training
                valOutputs = model(valInputs)
                valLoss += criterion(valOutputs, valLabels).item()
            valLoss /= len(valLoader)

            # save model if val loss improved
            if valLoss < bestValLoss:
                bestValLoss = valLoss
                torch.save(model.state_dict(), 'TheBestModel.pth')  # saving model state

            # stop early if no improvement
            if epoch - bestEpoch > EpochEpsilon:
                print("Stopping early, no improvemnt in val loss.")
                break

# post training, load best model
model.load_state_dict(torch.load('TheBestModel.pth'))
```

Figure 1:

4.4 d) Provide a Python code snippet implementing the loop over epochs for training your network using PyTorch. This should include the call to the cost function, back-propagation, and optimiser operations. There is no need to include code for printing or plotting results.

See figure 1

5 Assessment of Results : 5

5.1 a) Explain when you can consider the testing phase satisfactory. [2]

The assessment of when we may consider the testing phase satisfactory is mainly down to the accuracy of U is suitably low, or to use our predefined definition, when equation 2 the visual reconstruction can be suitably low such that there is sufficient accuracy.

More generally we should consider the linear combination of the two loss functions as the "objective function" as already mentioned in equation 3. The reason for this is that we not only care about the unveiled image but also minimising the loss function shown in equation 1.

Hence, once more we may generally define some ϵ such that

$$z^* = \mu L_{VisualSimilar}(B, C) + \lambda L_{reconstruction}(A, D(C)) < \epsilon : \epsilon, \mu, \lambda \in \mathbb{R}^+ \tag{4}$$

To take this further we could apply another neural network with a classification output to determine how often our system can identify the unveiled image as being the hidden image A after passing through our original system. However, this method may have more errors to account for and is potentially not as simple as the above threshold proposal.

5.2 b) Discuss which quality metrics you would use in view of the learning strategy you devised

Building on the previous answer, it seems logical to assume the metric that would be of most significance would be the linear combination of the two loss functions shown in equation 4. Hence monitoring the cost function as said in the previous answer and ensuring the value of the objective function is less than the predefined ϵ threshold value is paramount to ensure effectiveness for the process.

However, more fundamentally, as stated throughout the article here we care greatly about the accuracy of U, and hence the loss function of just equation 2 should be held highly to ensure we have suitable pixel-wise reconstruction and hence a well working system to our problem defined in our introduction.

Pushing forward with the idea of using a classification neural network to feed our unveiled image into to determine the accuracy of U after passing our original system could be doubled down on and used as a high quality metric, however this comes with great care as we must ensure the classifier has unrivalled accuracy on test set of the original CNN to ensure it is worth of being a high quality metric. Once more the proposed classifier still may open doors to more complex issues to do with the whole new classifier network and should be used only if you could guarantee certainty for classification for all image types of A and B in the original CNN.

Thinking in a more real world application, the ability for the model to perform in the real world is highly important and has a significant weighting, hence purposefully introducing new data unseen from the training and validation sets that the model was trained on is a valid way to test whether the model is robust enough to handle real world complexity with unseen data and would be testament to the data auxiliary, early stopping and other overfitting procedures implemented in the model to enable suitable real world application.

In summary, the quality metrics could be anything from reliable lost function values of the reconstruction loss eq 2 being below a threshold or the objective function (cost) being below a threshold eq 3, to more real world applicable methods of testing the model on unseen data, and perhaps comparing the amount of unseen datasets.