CS 335 & CS 337

Adarsh Raj - 190050004

Assignment-5

November 2021

Question 1 - Directed Graphical Models

1.1 - D-Separation

For the given Directed graphical model shown in figure below, we have conditional independence queries:

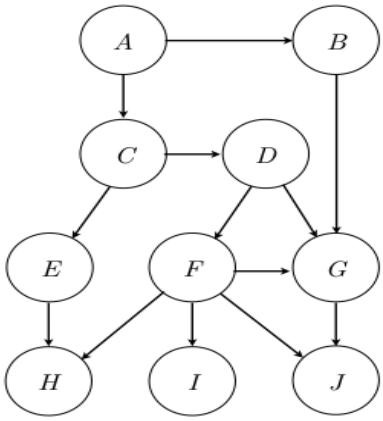

Figure 1: Directed Graphical Model

(a) **C⊥B: No**

For this case, path `(C-A-B)` is not blocked, as `A` is a **tail to tail** node, and is not in the conditional set. Hence, the answer is false.

(b) **C⊥B | A: Yes**

For this case, all paths are blocked as follows: path `(C-A-B)` is blocked by `A` as it is **tail to tail** and in conditional set, `G` and `J` are blocking for paths `(C-D-G-B)`, `(C-D-F-G-B)`, `(C-D-F-J-G-B)` and other paths containing them, as they are **head to head** node and are not in conditional set. Other paths from `C` to `B`, which includes `H`, is blocked by `H` as it is **head to head** and not in conditional set.

(c) **C⊥B | A, J: No**

For this case, path `(C-D-F-J-G-B)` is not blocked, as for this path `J` is not blocking as it is a **head to head** node and is in the conditional set. Other nodes are also non blocking as they are not in conditional set and are **head to tail** nodes. Hence, the answer is false.

(d) **C⊥B | A, J, D: Yes**

For this case, all paths are blocked as follows: earlier path `(C-D-F-J-G-B)` was not blocked, but including `D` in the conditional set makes it blocked, as `D` is **head to tail** node. Other paths containing `D` are also blocked due to same reason. Path `(C-A-B)` is blocked by `A` as it is **tail to tail** and in conditional set. For the other paths from `C` to `B`, `H` is blocking as it is **head to head** and not in conditional set.

(e) **C⊥G: No**

For this case, path `(C-D-G)` is not blocked, as `D` is a **head to tail** node and is not in conditional set. Hence, the answer is false.

(f) **C⊥G | B: No**

For this case, path `(C-D-G)` is not blocked, as `D` is a **head to tail** node and is not in conditional set. Hence, the answer is false.

(g) **C⊥G | B, D: Yes**

For this case, all the paths are blocked as follows: path which includes `H` is blocking as it is **head to head** and not in conditional set, path which includes `B` is blocking as it is **head to tail** and in the conditional set, path which includes `D` like `(C-D-G)` (which was non blocked earlier) is also blocked now, as `D` (**head to tail**) is in the conditional set.

(h) **C⊥G | B, D, H: No**

For this case, path `(C-E-H-F-G)` is not blocked, as for this path `H` is not blocking as it is a **head to head** node and is in the conditional set. Other nodes are also non blocking as they are not in conditional set and are **head to tail** `(E,F)` or **tail to tail** `(F)` nodes. Hence, the answer is false.

(i) **C⊥G | B, D, H, E: Yes**

For this case, all the paths are blocked as follows: path which includes `E` is blocking as it is **head to tail** and is in the conditional set, path which includes `B` is blocking as it is **head to tail** and in the conditional set, path which includes `D` like `(C-D-G)` is also blocked now, as `D` (**head to tail**) is in the conditional set. Since, to reach `G`, one of the above nodes must be encountered. Hence, all paths are blocked.

(j) **B⊥I | J: No**

For this case, path `(B-G-J-F-I)` is not blocked, as for this path `J` is not blocking as it is a **head to head** node and is in the conditional set. Other nodes are also non blocking as they are not in conditional set and are **head to tail** `(G)` or **tail to tail** `(F)` nodes. Hence, the answer is false.

1.2 - Probability Distribution

The expression for the joint probability $P(A, B, C, D, E, F, G, H, I, J)$ in 10 terms can be written as probabilities of all nodes given some nodes. This is as follows:

$$P(A, B, C, D, E, F, G, H, I, J) = \prod_{i=1}^{10} P(Node_i \mid parent(Node_i))$$

Now the joint probability (factorisation) can be expressed as the product of following terms, where the form is $P(Node \mid Parents(Node))$:

$P(A), P(B \mid A), P(C \mid A), P(D \mid C), P(E \mid C), P(F \mid D), P(G \mid B, D, F), P(H \mid E, F), P(I \mid F), P(J \mid F, G)$

1.3 - Parameters to learn probability distribution

Assuming that all the nodes in the DGM represent Bernoulli Random variables. i.e. each node takes value 1 with probability θ and value 0 with probability $1 - \theta$.

For each probability, the number of parameters required will be at least $2^{Parents}$ for each node as the node is dependent on the parent θ also, and possible combinations will be given by the expression $2^{number\ of\ parents}$. Now, also given in the problem $I = F$, hence, 0 parameters will be required to learn I, given F. Also, since H is OR of E and F, 0 parameters will be required to learn $P(H|E, F)$.

Summing up all the parameters, we get number of parameters to be at least **23** for learning the joint probability. Hence, the minimum number of parameters is **23**.

Probability Distribution	**# Parameters**
$P(A)$	1
$P(B \mid A)$	$2^1 = 2$
$P(C \mid A)$	$2^1 = 2$
$P(D \mid C)$	$2^1 = 2$
$P(E \mid C)$	$2^1 = 2$
$P(F \mid D)$	$2^1 = 2$
$P(G \mid B, D, F)$	$2^3 = 8$
$P(H \mid E, F)$	0
$P(I \mid F)$	0
$P(J \mid F, G)$	$2^2 = 4$
$\mathbf{P(A, B, C, D, E, F, G, H, I, J)}$	**23**

Question 2 - CNN Theory Questions

Task 1 - Detecting Image Containing Single Object

Given, a trained a CNN-based network to distinguish between images based on the kind of vehicle that the image contains. Also, the output layer of our network consists of a soft-max layer that can classify an image into one of N vehicular categories.

For the given images in 3 columns of Figure 3, the different components of the trained CNN based network would extract in the intermediate layers to help distinguish between images containing different vehicles, are as follows:

- **Convolution Layers:** A convolution kernel converts all the pixels in its receptive field into a single value. For example, if we would apply a convolution to an image, we will be decreasing the image size as well as bringing all the information in the field together into a single pixel.

  The **early convolutional layers** of the trained CNN model extract general or low-level features. The choice of kernel size determines the size of these features. Hence, it can be said that these layers have localised perspective over the image and hence are helpful in detecting vertical and horizontal edges, colors, gradient orientation, etc.

  With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset. Hence, the **later convolutional layers** gives a high level perspective and they detect larger and relatively more detailed features like patterns, shapes, textures, etc. Also, the layers at the end of CNN may detect even broader features like the parts of vehicles.

  For each image in the Figure 3, the convolution layers deeper in the network, closer to the output layer, may detect parts such as wheels, handles, etc of the motorbike and windshield, doors, roof, etc of the car. The earlier layers will detect the color pattern, gradients, edges of the car and motorbike, etc. An edge pattern shaping like a car can be distinguised from other vehicles which helps in classfication.

- **Pooling Layers:** The Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is **useful for extracting dominant features which are rotational and positional invariant**, thus maintaining the process of effectively training of the model. Pooling layer operates on each feature map independently. Since, these layers downsample the feature vectors, overfitting is reduced overall.

- **Fully Connected Layer:** A Fully-Connected layer is used for learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The fully connected layer provides CNN model the ability to mix signals, since every single neuron has a connection to every single one in the next layer, now there is a flow of information between each input dimension (pixel location) and each output class, thus the decision is based truly on the whole image. Hence, overall it summarises the interaction between different filter kernel responses.

Why CNN correct results even when the location or size of an object in the image changes?

Due to the CNN property of **Translational Invariance**. This property makes the CNN invariant to translation, which means that if we translate the inputs the CNN will still be able to detect the class to which the input belongs. Translational Invariance is a result of the pooling operation. Hnece, for the first image, where car is on top-left, then the output(kernel response) will also be significant in the top-left of the output tensor, similarly, for bottom right, it will be in the bottom right of the output tensor.

CNN also produces correct results for images of different sizes due to inherent properties which are as follows:

- The input size at all because once the kernel and step sizes are described, the convolution at each layer can generate appropriate dimension outputs according to the corresponding inputs.
- We can use Global Average Pooling or Global Max Pooling to reduce the feature maps. This leads to the output dimensionality being independent of the spatial size of the feature maps. In case of classification, we can then proceed to use a fully connected layer on top to get the classes.

Task 2 - Detecting Separated Objects

Given, images that contain a combination of multiple vehicles but well separated from each other within the same image. We can modify the CNN model as follows to accomodate this:

- **Boundary Box/ Grid Addition**: The image can be divided into multiple rectangular boxes or grids corresponding to different regions. We except to map different objects to different boxes or grids, because of the non overlapping property mentioned in the problem statement.

- **Output Layer:** The output layer must change to accommodate the above grid mapping, which is `softmax` for each grid, thus classifying objects in grid individually. This network thus divides the image into regions and predicts

bounding boxes and probabilities of objects for each grid. These bounding boxes are weighted by the predicted probabilities.

- **Post Processing:** Since, multiple grid classifications will be present in the final classification of the output layer. We must do post processing steps like Intersection Over Union or Non Maximal Suppression to make the bounding box one, and this can be easily done as the objects will be well separated.
- **Limitations:** If the object doesn't fit in a single rectangle, then it makes non-trivial for classification.

- We can't use any global processing layers like Global Average Pooling Layer as there are multiple objects in the global image and we want to classify the separated objects which belong to different classes.

Task 3 - Detecting Overlapping Objects

To detect overlapping vehicles within the same image, i,e occlusion, we can use the same idea as above with very little modifications, which are as follows:

- **Boundary Box/ Grid Modification**: The image can be divided into rectangular grids same as before. But, now multiple objects can be present in a single grid, which we can checked by adding a **sigmoid** layer corresponding to each grid which will provide the probabilities of the objects which may be present in the grid. Hence, these bounding boxes are weighted by the predicted probabilities. This is similar to the **YOLO** architecture, which uses similar schemes for occlusion prevention.

- **Output Layer:** The output of above can be fed to the softmax layer which will be used for grid mapping. After this post-processing steps like Intersection and non-maximal suppression can be applied similar to the above subpart.

- **Limitations:** The limitation with this method is that in case we have multiple objects of the same category in a region, then they'll be counted only once.

Question 3 - Feed Forward Neural Networks

Lab

Implementation of a feed forward network to predict digit from a image (using MNIST dataset) and flower type from flower dataset given in the `data` folder. In MNIST dataset each element in `(28x28)` 2D array and in flowers dataset each element is a vector of length `2048`.

All the functions `forward`, `backward` for all the layers were implemented. Also, different activation functions and loss functions along with their derivatives were implemented. Then, `fit` function was implemented using `SGD` (Stochastic Gradient Descent), with `epochs=50` and `lr=0.01`. Following networks were trained for MNIST and Flowers Dataset respectively and the activation function used is `tanh` along with `cross entropy loss`.

The code can be found in file `assignment_5.ipynb`.

```
network = [
    FlattenLayer(input_shape=(28, 28)),
    FCLayer(28*28, 12),
    ActivationLayer(tanh, tanh_prime),
    FCLayer(12, 10),
    SoftmaxLayer(10)
] # This creates feed forward
```

Figure 2: Neural Network for MNIST Dataset

```
network = [
    FlattenLayer(input_shape=(2048, 1)),
    FCLayer(2048, 18),
    ActivationLayer(tanh, tanh_prime),
    FCLayer(18, 5),
    SoftmaxLayer(5)
] # This creates feed forward
```

Figure 3: Neural Network for Flowers Dataset

The `fit` function was used to train the model and model was saved for both datasets with names `weights_mnist.pkl` and `weights_flowers.pkl` and are stored in the `models` directory. The saved models are called in predict function and are tested on test data. Following accuracies were obtained for a random seed for spliting the datasets into training and validation. The print statement is self explanatory regarding which accuracy is being calculated.

```
: yt = predict(mx_tr, 'mnist')
  acc = np.mean(yt == my_tr)
  print(f"Accuracy on MNIST Training set: {acc}")

  yt = predict(mx_val, 'mnist')
  acc = np.mean(yt == my_val)
  print(f"Accuracy on MNIST Validation set: {acc}")

  yt = predict(fx_tr, 'flowers')
  acc = np.mean(yt == fy_tr)
  print(f"Accuracy on FLOWERS Training set: {acc}")

  yt = predict(fx_val, 'flowers')
  acc = np.mean(yt == fy_val)
  print(f"Accuracy on FLOWERS Validation set: {acc}")
```

```
Accuracy on MNIST Training set: 0.9425208333333334
Accuracy on MNIST Validation set: 0.9394166666666667
Accuracy on FLOWERS Training set: 0.9510221465076661
Accuracy on FLOWERS Validation set: 0.9472789115646258
```

Figure 4: Accuracies on Datasets of MNIST and Flowers using saved weights