CPSC 420 Homework #4 Fall 2023 (written + coding)

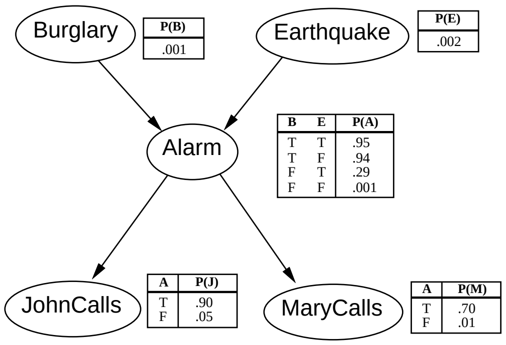
Total: 100 pts

Yoonsuck Choe

October 29, 2023

1 Uncertainty and Probabilistic Reasoning

Problem 1, Written (5 pts): Given the belief network as shown below, calculate the joint probability $P(JohnCalls, \neg MaryCalls, Alarm, \neg Earthquake, Burglary)$.



P(B)
.001

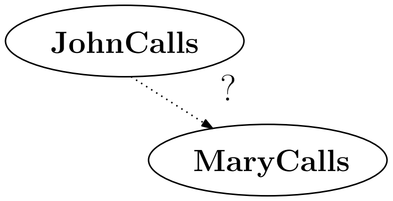
P(E)
.002

B	E	P(A)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J)
T	.90
F	.05

A	P(M)
T	.70
F	.01

Problem 2, Written (5 pts): Consider the belief network construction algorithm. (1) If the node order is “JohnCalls”, “Marycalls”, ..., and “JohnCalls” is already added to the beilef network, should “JohnCalls” point to “MaryCalls” when “MaryCalls” is newly added? (2) Explain by referring to certain probabilities that need to be compared in this case, and whether these probabilities are equal or not.



Problem 3, Written (5 pts): Consider random variables $Earthquake, Burglary$, and $Alarm$. Using these three nodes, give an example of “intercausal inference”. Explain why. Hint: Consider the cause and effect.

2 Learning

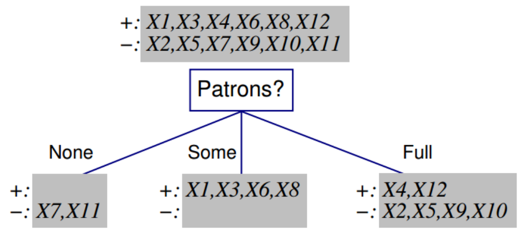
2.1 Decision Tree Learning

Consider the following set of examples where given attributes A through E, you want to make a decision "Target".

Example# / Attributes	A	B	C	D	E	Target
1	1	1	1	0	0	N
2	1	1	0	1	1	N
3	0	1	1	1	1	N
4	0	1	0	1	0	N
5	1	1	1	0	1	N
6	0	0	1	1	1	Y
7	0	1	1	0	0	Y
8	0	1	0	0	1	Y
9	1	1	1	1	0	N
10	0	0	1	0	0	Y
11	1	0	1	1	1	Y
12	0	1	0	1	1	N
13	1	0	0	0	0	Y
14	1	1	0	1	0	N

Table 1: Simple Decision Task

Problem 4, Written (6 pts): For three attributes A, B, and D, draw a depth-one decision tree (one attribute tested) with the attribute as the root node. Use the following style. Note that all the attributes are binary attributes, so you will only have two branches.



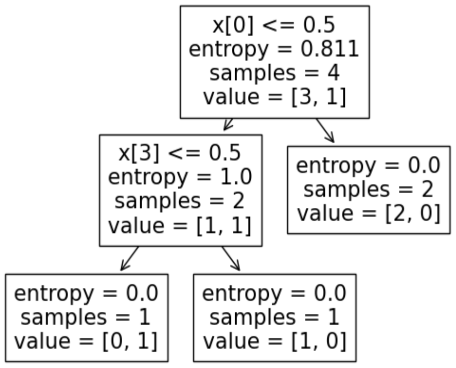
Problem 5, Written (14 pts): For the data in Table 1, (1) For each of the three attributes A, B, and D, calculate the information gain by hand. Use the result from problem 3. (2) Based on the information gains, which attribute would you choose to test first?

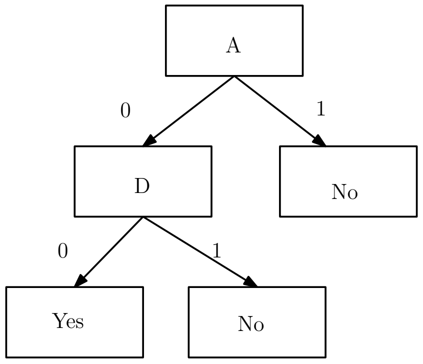
Problem 6, Programming (15 pts) - use `hw4.ipynb` Write a short program (in Python) to compute the information gain given the number of positive and negative samples prior to testing with an attribute, and the number of positive and negative samples after testing with the attribute for each possible value of the attribute. For example, the call would be `infogain([6, 6], [[0, 2], [4, 0], [2, 4]])` for the Patrons attribute shown in the figure above in problem 3.

Use your program to compute the information gain for each of the three attributes A, B, and D in the Table 1.

Note: You have to check and handle cases where $\log_2(0)$ appears and treat it as 0.

Problem 7, Programming (10 pts) - use `hw4.ipynb` (1) Using the decision tree package in scikit-learn, write a Python code to train a decision tree on the data in Table 1. (2) Based on the results, draw a decision tree by hand, mapping back `X[0]` to `A`, `X[1]` to `B`, etc. so that it is more easily readable by a human (see below, and read the comment in the provided code). You don't need to program this. Just draw it by hand. (3) Test two novel inputs (inputs that do not appear in Table 1), and report the trained decision tree's response.





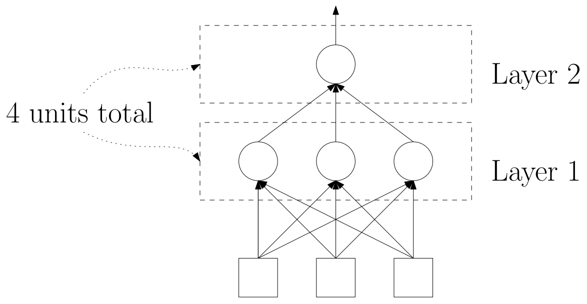
2.2 Neural Networks

Problem 8, Programming (25 pts) - use `hw4.ipynb` (1) Write a perceptron program, from scratch, in python (this is a simple program, so the core part would be roughly less than 20 lines of code). Test it for AND, OR, and XOR functions. Write a plotting function to show the decision boundary (see example code). (2) For each function, run multiple training sessions with different random initial weights. Report the results, including where the final decision boundary was for each session. (Note: XOR would not terminate, so stop it after a few steps.) (3) For the AND and OR problems, is it possible to have the perceptron make zero error without training? Assume the weights are randomly initialized. Explain why or why not. Observing the decision bounadaries throughout the training session can give you some insight.

* You don't need to store the data in a file and load it. Just define the data matrix in your code.

Problem 9, Written (5 pts): Show how we can implement the XOR function linking up three perceptron units. You don't need to code anything. Graphically illustrate your idea, both (1) neural network topology, and (2) decision boundary of each unit in the input space.

* Note: do not confuse the number of layers and the number of units. For example, 1 hidden layer with 3 units and 1 output layer with 1 unit will have a total of 4 perceptron units in 2 layers. See figure below.



* You can use `http://playground.tensorflow.org` to get some insights. You may use a screen-shot of the proper configuration as your answer.

Problem 10, Written Exercise (no credit):

Given $E(w) = \frac{1}{4}w(w+2)(w+1)(w-2)$, derive a gradient descent learning rule to adjust w, where α is the learning rate. Basically, you need to find:

$$\Delta w = -\alpha \frac{dE}{dw}$$

For example, if $E(w) = w^2$, then $\frac{dE}{dw} = -2w$, thus $\Delta w = \alpha w$. Since $\frac{dE}{dw}$ is a function of w (let's call this function $dEdw(w) = -2w$), you can plug in a value given the current w. When $w = 1.3$, $dEdw(w) = dEdw(1.3) = -2.6$. So, if $\alpha = 0.1$, $\Delta w = -\alpha \times dEdw(w) = -0.1 \times (-2.6) = 2.6$.

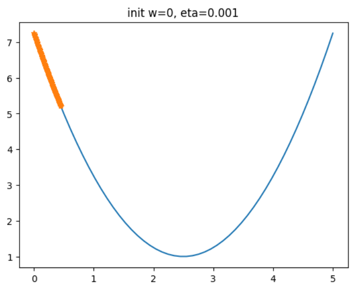
Problem 11, Programming Exercise (no credit) - use `hw4.ipynb`

Write a short program to simulate the gradient descent steps in the previous problem: $E(w) = \frac{1}{4}w(w+2)(w+1)(w-2)$. Given any arbitraty initial w value between -3.0 and 3.0 it is guaranteed to find the local minima. Plug in the initial w value to the derived function $\frac{dE}{dw}$ to compute Δw, and use the updated w value plug it in to $\frac{dE}{dw}$ to compute Δw, etc. Plot the w values over each iteration, overlaid over the plot for the function $E(w)$. Show the plots in the range of w values from -3.0 to 3.0.

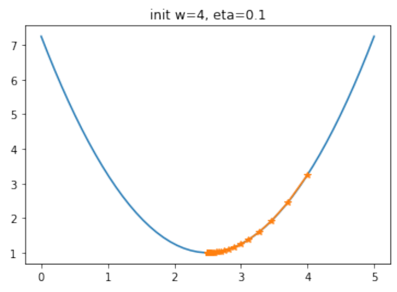
1. Suppose dEdw(w) is the derivative function.
2. Given initial_w = 2.0, alpha=0.01
3. DeltaW = -0.01 * dEdw(2.0).
4. new_w = 2.0 + DeltaW = 2.0 - 0.01 * dEdw(2.0).
5. next DeltaW = -0.01 * dEdw(new_w)
6. ...

- Experiment with different learning rates α between 0.01 and 0.1 with initial w values between -2.5 and 2.5. Plot it.
- Show two initial w values that lead to the global minimum. Plot it.
- Show two initial w values that lead to the local minumum (that is not a global minumum). Plot it.

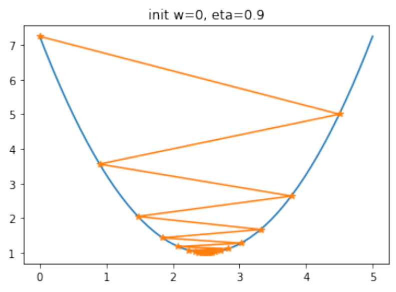
Here are some typical behaviors, when $E(w) = (w - 2.5)^2$.



α too low



α about right



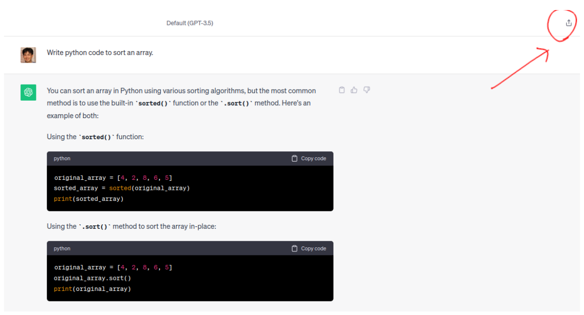
α too high

3 Deep Learning

Problem 12, Programming (10 pts) - use `hw4.ipynb`

For this problem, code and run everything in Colab and submit your results at the end of the provided ipynb file.

Show all your prompts: Include these as a comment at the beginning of your code. Include the link to your ChatGPT session at the beginning (see below). Do this after you've obtained the final functioning code. For example, `https://chat.openai.com/share/0e575a05-ae67-4d0d-a76d-1e95b8b29714`



Write python code to sort an array.

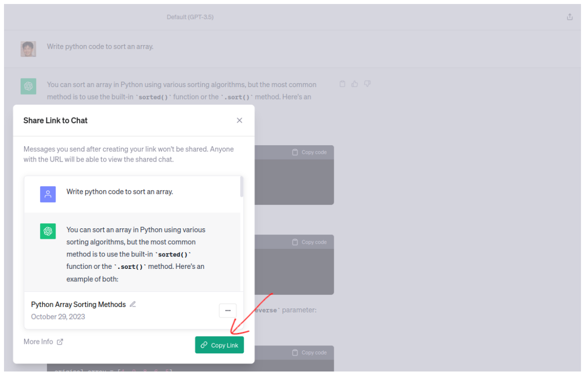
You can sort an array in Python using various sorting algorithms, but the most common method is to use the built-in `sorted()` function or the `.sort()` method. Here's an example of both:

Using the `sorted()` function:

```python
original_array = [4, 2, 8, 6, 5]
sorted_array = sorted(original_array)
print(sorted_array)
```

Using the `.sort()` method to sort the array in-place:

```python
original_array = [4, 2, 8, 6, 5]
original_array.sort()
print(original_array)
```



Share Link to Chat

Messages you send after creating your link won't be shared. Anyone with the URL will be able to view the shared chat.

Write python code to sort an array.

You can sort an array in Python using various sorting algorithms, but the most common method is to use the built-in `sorted()` function or the `.sort()` method. Here's an example of both:

Python Array Sorting Methods
October 29, 2023

DO NOT use the homework problem's text itself as the prompt. Write your own prompt.

(1) In the first code cell, use ChatGPT to generate code for a Convolutional Neural Network (CNN) variant "LeNet" (Yann LeCun's original model), and train it on the MNIST hand-written data set. Use 2 conv layers. Make the receptive field size to be 5×5. Use 25 channels for each conv layer (i.e., you will have 25 kernels). Experiment with different learning parameters, such as the number of epochs, batchsize, etc. You should get a pretty good outcome (above 99% accuracy).

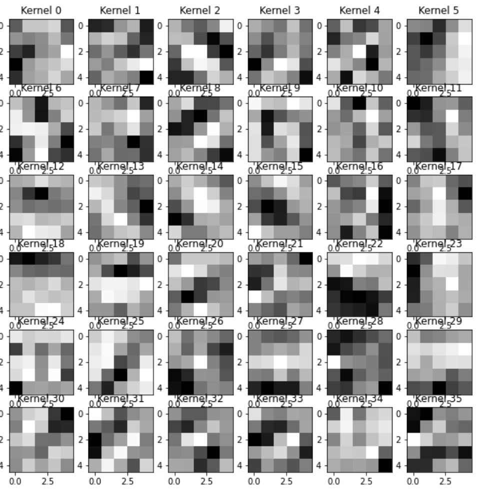
(2) In the CNN code generated, you will see a section like below. Explain what each layer does, in your own words. Do not use ChatGPT for this. Include this as a comment in the code. Note: this code is in tensorflow. If your code is generated for Pytorch or some other framework, you will get something similar.

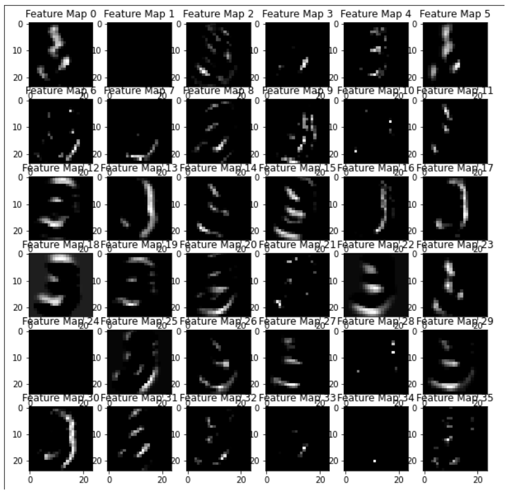
```
# This line sets up ... <-- your explanation
model = Sequential()

# This line creates XX many ... with such and such ... and .. . The input is set up as ...
model.add(Conv2D(25, (5, 5), activation='relu', input_shape=(w, h, ch)))

# ...
model.add(MaxPooling2D(2, 2))
...
model.add(Flatten())
...
model.add(Dense(units=84, activation='relu'))
model.add(Dense(units=10, activation = 'softmax'))
```

(3) In the second code cell, use ChatGPT to plot the convolution kernels from the first conv layer. Use the "gray" colormap. See below for an example. Here, there were 25 channels, which were plotted in a 5×5 arrangement.

(4) In the third code cell, use ChatGPT to plot the featuremaps from first conv layer. Pick an input in the

category “3” and show the plot. Use the “gray” colormap. See below for an example. Here, there were 25 channels, which were plotted in a 5×5 arrangement.


(a) Receptive fields (conv kernel)


(b) Featuremap