ML System Design

1. Clarify Requirements
What is the goal? Any secondary goal?
e.g. for CTR - maximizing the number of clicks is the primary goal. A secondary goal might be the quality of the ads/content
Ask questions about the scale of the system - how many users, how much content?
2. How the ML system fits into the overall product backend
Think/draw a very simple diagram with input/output line between system backend and ML system
3. Data Related Activites
Data Explore - whats the dataset looks like?
Understand different features and their relationship with the target
- Is the data balanced? If not do you need oversampling/undersampling?
- Is there a missing value (not an issue for tree-based models)
- Is there an unexpected value for one/more data columns? How do you know if its a typo etc. and decide to ignore?
Feature Importance - partial dependency plot, SHAP values, dataschool video (reference)
(ML Pipeline: Data Ingestion) Think of Data ingestion services/storage
(ML Pipeline: Data Preparation) Feature Engineering - encoding categorical features, embedding generation etc.
(ML Pipeline - Data Segregation) Data split - train set, validation set, test set
4. Model Related Activities
(ML Pipeline - Model Train and Evaluation) Build a simple model (XGBoost or NN)
- How to select a model? Assuming its a Neural Network
1. NLP/Sequence Model
- start: LSTM with 2 hidden layers
- see if 3 layers help,
- improve: check if Attention based model can help
2. Image Models - (Don't care right now)
3. Other
- start: Fully connected NN with 2 hidden layers
- Improve: problem specific
(ML Pipeline - Model Train and Evaluation) What are the different hyperparameters (HPO) in the model that you chose and why?
(ML Pipeline - Model Train and Evaluation) Once the simple model is built, do a bias-variance tradeoff, it will give you an idea of overfitting vs underfitting and based on whether overfit or underfit, you need different approaches to make you model better.
Draw the ML pipeline (reference #3)

ML System Design (cont)

Model Debug (reference #1)
Model Deployment (reference#3)
(ML Pipeline: Performance Monitoring) Metrics
AUC, F1, MSE, Accuracy, NDCG for ranking problems etc.
When to use which metrics?
5.Scaling

Binary Search: Time - O(log n), Space - O(1)

```
def search(nums, target):
                start = 0
               end = len(nums) - 1
               mid = 0

                while start <= end:

                       mid = (start + end) // 2

                       # If x is greater, ignore
left half
                       if nums[mid] < target:
                                start = mid + 1

                       # If x is smaller, ignore
right half
                       elif nums[mid] > target:
                               end = mid - 1

                       # means x is present at mid
                        else:
                                return mid

               # If we reach here, then the
element was not present
                return -1
```

DFS for In-order Tree Traversal

DFS for In-order Tree Traversal (cont)

```
>         break

   print()
```

DFS for Graphs

```
def DFS(self,s):
# prints all vertices in DFS manner from a given
source.
# Initially mark all vertices as not visited
        visited = [False for i in range( sel f.V)]

       # Create a stack for DFS
        stack = []

       # Push the current source node.
        sta ck.a pp end(s)

        while (len(s tack)):
               # Pop a vertex from stack and print
it
               s = stack[-1]
                sta ck.p op()

               # Stack may contain same vertex
twice. So
               # we need to print the popped item
only
               # if it is not visited.
               if (not visite d[s]):
                        pri nt( s,end=' ')
                        vis ited[s] = True

               # Get all adjacent vertices of the
popped vertex s
               # If a adjacent has not been
visited, then push it
               # to the stack.
               for node in self.a dj[s]:
                       if (not visite d[n ode]):
                                sta ck.a pp end -
(node)
```

```
def inOrder(root):
        current = root
        stack = []

        while True:
               if current is not None:
                        sta ck.a pp end (cu rrent)

                        current = curren t.left
                eli f(s tack):
                        current = stack.p op()
                        pri nt( cur ren t.data,
end=" ")
                        current = curren t.right

                else:
```

Batch Normalization

```
activation_map_sample1 = np.array([
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]
], dtype= np.f lo at32)

activa tio n_m ap_ sample2 = np.array([
       [1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]
], dtype= np.f lo at32)

activa tio n_m ap_ sample3 = np.array([
       [9, 8, 7],
       [6, 5, 4],
       [34, 2, 1]
], dtype= np.f lo at32)
activa tio n_m ean_bn = np.mea n([ act iva tio -
n_m ap_ sam ple1, activa tio n_m ap_ sam ple2,
activa tio n_m ap_ sam ple3], axis=0)

#get standard deviation across different samples in
batch for each activation
activa tio n_s td_bn = np.std ([a cti vat ion _ma -
p_s ample1, activa tio n_m ap_ sam ple2, activa -
tio n_m ap_ sam ple3], axis=0)

activa tio n_m ap_ sam ple1_bn = (activ ati on_ -
map _sa mple1 - activa tio n_m ean_bn) / activa -
tio n_s td_bn
```

BCE

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log (1 - \hat{Y}_i))$$

Linked List

```
class Node:

       # Constr uctor to initialize the node
object
       def __init __( self, data):
                sel f.data = data
                sel f.next = None



class Linked List:

       # Function to initialize head
       def __init __( self):
                sel f.head = None

       # Function to reverse the linked list
       def revers e(s elf):
               prev = None
                current = self.head
                whi le( current is not None):
                       next = curren t.next
                        cur ren t.next = prev
                       prev = current
                        current = next
                sel f.head = prev

       # Function to insert a new node at the
beginning
       def push(self, new_data):
                new _node = Node(n ew_ data)
                new _no de.next = self.head
                sel f.head = new_node
def delete N(head, position):
       temp = head
       prev = head
       for i in range(0, position):
               if i == 0 and position == 1:
                       head = head.next
```

Linked List (cont)

```
>        else:
          if i == position-1 and temp is not None:
              prev.next = temp.next
          else:
              prev = temp

              # Position was greater than
              # number of nodes in the list
              if prev is None:
                  break
              temp = temp.next
    return head
    def search(self, x):

        # Initialize current to head
        current = self.head

        # loop till current not equal to None
        while current != None:
            if current.data == x:
                return True # data found

            current = current.next

        return False # Data Not found
    def getCount(self):
        temp = self.head # Initialise temp
        count = 0 # Initialise count

        # Loop while end of linked list is not reached
        while (temp):
            count += 1
            temp = temp.next
        return count

    # Utility function to print the LinkedList
    def printList(self):
```

Linked List (cont)

```
>        temp = self.head
        while(temp):
            print(temp.data, end=" ")
            temp = temp.next


# Driver code
llist = LinkedList()
llist.push(20)
llist.push(4)
llist.push(15)
llist.push(85)
```

Floor/Ceil

```
3//2 # floor == 1
-(-3//2) # ceil == 2
```

BFS for Level-order Tree Traversal

```
class TreeNode:
def __init __( self, key):
self.data = key
self.left = None
self.right = None
def printL eve lOr der (root):
        if not root:
                  return []


         queue = []
         que ue.a pp end (root)

         whi le( len (queue) > 0):
                  pri nt( que ue[ 0].d ata)
                node = queue.p op(0)
                if node.left is not None:
                          que ue.a pp end (no de.l -
eft)
```

BFS for Level-order Tree Traversal (cont)

```
>       if node.right is not None:
         queue.append(node.right)
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
printLevelOrder(root)
```

BFS for Graphs

```
def bfs(r,c):
       q = collec tio ns.d eque()
        q.a ppe nd( (r,c))
        vis ite d.a dd( (r,c))
        nei ghbors = [[-1,0 ],[ 0,- 1], [1, 0], -
[0,1]]
       area = 1
        while q:
               row, col = q.popl eft()
               for n in neighbors:
                        curInd = (row + n[0], col +
n[1])
                      if (curInd[0] >= 0) and
(curInd[0] < rows) and (curInd[1] >= 0) and
(curInd[1] < cols):
                                 if grid[c urI nd[ -
0]] [cu rIn d[1]] and (curInd not in visited):
                                       q.a ppe -
nd( curInd)
                                       vis ite -
d.a dd( curInd)
                                      area = area
+ 1
        return area
```

Softmax Reg PyTorch

```
import time
from torchv ision import datasets
from torchv ision import transforms
from torch.u ti ls.data import DataLoader
import torch.n n.f un ctional as F
import torch
device = torch.d ev ice ("cu da: 0" if torch.c -
ud a.i s_a vai lable() else " cpu ")
# Hyperp ara meters
random _seed = 123
learni ng_rate = 0.1
num_epochs = 25
batch_size = 256
# Archit ecture
num_fe atures = 784
num_cl asses = 10
###### ### ### ### ### ### #####
### MNIST DATASET
###### ### ### ### ### ### #####
train_ dataset = datase ts.M NI ST( roo t=' data',
                                                -
            tra in= True,
                                                -
            tra nsf orm =tr ans for ms.T oT ens -
or(),
                                                -
            dow nlo ad= True)
test_d ataset = datase ts.M NI ST( roo t=' data',
                                                -
          tra in= False,
                                                -
          tra nsf orm =tr ans for ms.T oT ens or())
train_ loader = DataLo ade r(d ata set =tr ain -
_da taset,
                                                -
  bat ch_ siz e=b atc h_size,
                                                -
  shu ffl e=True)
test_l oader = DataLo ade r(d ata set =te st_ dat -
aset,
                                                -
bat ch_ siz e=b atc h_size,
```

Softmax Reg PyTorch (cont)

```
>                       shuffle=False)
# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape) #NCHW
    print('Image label dimensions:', labels.shape)
    break
####### MODEL ######
class SoftmaxRegression(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super(SoftmaxRegression, self).__init__()
        self.linear = torch.nn.Linear(num_features, num_classes)

        self.linear.weight.detach().zero_()
        self.linear.bias.detach().zero_()

    def forward(self, x):
        logits = self.linear(x)
        probas = F.softmax(logits, dim=1)
        return logits, probas
class MLP(torch.nn.Module):
    def __init__(self, num_features, num_hidden, num_classes):
        super().__init__()

        self.num_classes = num_classes

        ### 1st hidden layer
        self.linear_1 = torch.nn.Linear(num_features, num_hidden)
        self.linear_1.weight.detach().normal_(0.0, 0.1)
        self.linear_1.bias.detach().zero_()
        ### Output layer
        self.linear_out = torch.nn.Linear(num_hidden, num_classes)
        self.linear_out.weight.detach().normal_(0.0, 0.1)
        self.linear_out.bias.detach().zero_()

    def forward(self, x):
```

Softmax Reg PyTorch (cont)

```
>       out = self.linear_1(x)
        out = torch.sigmoid(out)
        logits = self.linear_out(out)
        #probas = torch.softmax(logits, dim=1)
        return logits#, probas

#################################
### Model Initialization
#################################

torch.manual_seed(RANDOM_SEED)
model = MLP(num_features=28*28,
            num_hidden=100,
            num_classes=10)
model = SoftmaxRegression(num_features=num_features,
                          num_classes=num_classes)
model.to(device)
############################
### COST AND OPTIMIZER
############################
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
torch.manual_seed(random_seed)
def compute_accuracy(model, data_loader):
    correct_pred, num_examples = 0, 0

    for features, targets in data_loader:
        features = features.view(-1, 28*28).to(device)
        targets = targets.to(device)
        logits, probas = model(features)
        _, predicted_labels = torch.max(probas, 1)
        num_examples += targets.size(0)
        correct_pred += (predicted_labels == targets).sum()

    return correct_pred.float() / num_examples * 100
```

Softmax Reg PyTorch (cont)

```
> start_time = time.time()
epoch_costs = []
for epoch in range(num_epochs):
    avg_cost = 0.
    for batch_idx, (features, targets) in enumerate(train_loader):

        features = features.view(-1, 28*28).to(device)
        targets = targets.to(device)

        ### FORWARD AND BACK PROP
        logits, probas = model(features)

        # note that the PyTorch implementation of
        # CrossEntropyLoss works with logits, not
        # probabilities
        cost = F.cross_entropy(logits, targets)
        optimizer.zero_grad()
        cost.backward()
        avg_cost += cost

        ### UPDATE MODEL PARAMETERS
        optimizer.step()

        ### LOGGING
        if not batch_idx % 50:
            print ('Epoch: %03d/%03d | Batch %03d/%03d | Cost: %.4f'
                   %(epoch+1, num_epochs, batch_idx,
                     len(train_dataset)//batch_size, cost))

    with torch.set_grad_enabled(False):
        avg_cost = avg_cost/len(train_dataset)
        epoch_costs.append(avg_cost)
        print('Epoch: %03d/%03d training accuracy: %.2f%%' % (
              epoch+1, num_epochs,
              compute_accuracy(model, train_loader)))
        print('Time elapsed: %.2f min' % ((time.time() - start_time)/60))
```