In [2]:

```python
# Aim: Get LSTM to learn addition mod n
# Sequence to Sequence modelling
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
```

In [3]:

```python
# mock test of overriding as would be expected in recurrent structure
# thus overwriting works!
# pytorch keeps versions of stuff
mock_in = torch.tensor(1.0, requires_grad=True)
mock_rec = torch.tensor(2.0, requires_grad=True)
mock_rec2 = torch.tensor(2.0, requires_grad=True)
mock_in = mock_in * mock_rec
mock_in = mock_in * mock_rec
mock_in = mock_in * mock_rec
mock_in.backward()
print(mock_rec._grad)

# if you do not detach here (e.g some sort of multiheaded loss) you get error as graph got garbaged when you backprop
mock_cpy = mock_in.detach()
mock_cpy = mock_cpy * mock_rec2
mock_cpy.backward()

# trying to access mock_cpy's grad gives error
# trying to
print(mock_rec2._grad) # makes sense, doesnt include anything from mock_in
```

```
tensor(12.)
tensor(8.)
```

In [4]:

```python
class PrefixSumDataset(Dataset):

    def __init__(self, n, length, num_samples):

        self.X = np.random.randint(0,n,size=(num_samples,length))
        self.Y = np.cumsum(self.X, axis=1) % n
        self.n = n
    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        # onehot encode everything, onehot requires long vector
        return F.one_hot(torch.tensor(self.X[idx], dtype=torch.long), num_classes=self.n),\
    F.one_hot(torch.tensor(self.Y[idx], dtype=torch.long), num_classes=self.n)
```

In [5]:

```python
# testing
PS = PrefixSumDataset(3, 5,10)
PS.__getitem__(1)
```

Out[5]:

```
(tensor([[1, 0, 0],
         [1, 0, 0],
         [0, 1, 0],
         [1, 0, 0],
         [0, 0, 1]]),
 tensor([[1, 0, 0],
         [1, 0, 0],
         [0, 1, 0],
         [0, 1, 0],
         [1, 0, 0]]))
```

In [90]:

```python
"""
Demo of LSTMs

Can't do teacher forcing as output isn't fed back into input
Will train as is
"""
class SeqModel(nn.Module):

    def __init__(self, n, hidden_dim):
        """     initH, initC ->
        data -> LSTM -> outputLayer, hidden, cell
        outputLayer -> (dropout) -> decoder -> price prediction
        """
        super(SeqModel, self).__init__()

        # if num_layers = 1 then dropout is not used, it is not applied to final output layer
        self.LSTM = nn.LSTM(input_size = n, hidden_size=hidden_dim, num_layers = 1,batch_first=True, dropout=0)
        self.decoder = nn.Sequential(torch.nn.Dropout(p=0.1),
                                     nn.Linear(hidden_dim,n),
                                     nn.Softmax())
        self.init_hidden = torch.zeros((1,hidden_dim), dtype=torch.float)
        self.init_cell = torch.zeros((1,hidden_dim), dtype=torch.float)

    def forward(self, x):
        # x is a one hot encoded array to prefix sum shape (*, len, n)
        if len(x.shape) == 2: # no batch
            lstm_outputs, (final_hidden, final_cell) = self.LSTM(x.float(), (
                self.init_hidden, self.init_cell
            ))
        else:
            lstm_outputs, (final_hidden, final_cell) = self.LSTM(x.float(), ( # broadcast to batch
                self.init_hidden.repeat(1,len(x),1), self.init_cell.repeat(1,len(x),1)
            ))
        return self.decoder(lstm_outputs)

"""
Fold operator style model

Can do teacher forcing

Bit different to NAR in that we dont work in latent space

In NAR it would be
new value -> encode, concat to previous latent (representing prev sum) -> new latent -> decode -> new sum
and we can still do teacher forcing by taking correct prev sum and encoding it as latent
"""
class AggregatorModel(nn.Module):

    def __init__(self, n, hidden_dim):
        """     initH, initC ->
        prev_sum , cur_val -> concat -> 2 layer NN -> new_sum

        new_sum is fed back into prev_sum
        """
        super(AggregatorModel, self).__init__()

        # if num_layers = 1 then dropout is not used, it is not applied to final output layer
        self.processor = nn.Sequential(nn.Linear(2*n, hidden_dim),
                                       nn.ReLU(),
                                       nn.Dropout(p=0.1),
                                       nn.Linear(hidden_dim, n),
                                       nn.Softmax())

    def forward(self, cur_val, prev_sum):
        # x is a one hot encoded array to prefix sum shape (*, len, n)
        return self.processor(torch.concat((cur_val, prev_sum), dim=-1))
```

In [94]:

```python
from tqdm import tqdm

# train aggregator model
def trainBatch(batchX, batchY, mModel, mLoss_fn, mOptimizer, parameters_dict):
    n = batchX.shape[-1]
    l = batchX.shape[-2]
    batchSize = batchX.shape[0]
    teacher_forced = parameters_dict['teacher_forced']
    trunc_size = parameters_dict['trunc_size']
    mOptimizer.zero_grad()

    loss = 0

    if teacher_forced:
        # feed in correct 'prev' sums
        for i in range(l):
            true_prev_sum = torch.zeros((batchSize, n), dtype=torch.float) if i == 0 else batchY[:,i-1,:].float()
            preds = mModel(true_prev_sum, batchX[:,i,:])
            #print(preds, batchY[:,i,:])
            loss += mLoss_fn(preds, batchY[:,i,:].float())
    else:
        # dont feed in correct 'prev' sums
        prev_sum = torch.zeros((batchSize, n), dtype=torch.float)
        for i in range(l):
            if i%trunc_size == 0:
                prev_sum = prev_sum.detach()
            #print(preds, batchY[:,i,:])
            prev_sum = mModel(batchX[:,i,:], prev_sum)
            loss += mLoss_fn(prev_sum.squeeze(), batchY[:,i,:].squeeze().float())

    loss.backward()
    mOptimizer.step()
    return loss.item()

def trainBatchLSTM(batchX, batchY, mModel, mLoss_fn, mOptimizer, parameters_dict):
    # standard training protocol
    # no teacher forcing no batching
    pred = mModel(batchX)
    loss = mLoss_fn(pred, torch.squeeze(batchY).float())
    mOptimizer.zero_grad()
    loss.backward()
    mOptimizer.step()
    return loss.item()

def trainloop(train_dataset, mModel, mLoss_fn, mOptimizer, prob_teacher_force, trunc_size=5):
    # execute one train pass over data
    # return total training loss
    mDataLoader = DataLoader(train_dataset, batch_size = 20, shuffle=True)
    total_loss = 0
    use_lstm = False
    if mModel.__class__.__name__ == 'SeqModel':
        use_lstm = True

    for batchNum, (X,y) in tqdm(enumerate(mDataLoader)):
        if use_lstm:
            total_loss += trainBatchLSTM(X,y, mModel, mLoss_fn, mOptimizer, {})
        else: #AggModel
            use_teacher_force = np.random.random() < prob_teacher_force
            parameters_dict = {'teacher_forced':use_teacher_force, 'trunc_size':trunc_size}
            total_loss += trainBatch(X,y, mModel, mLoss_fn, mOptimizer, parameters_dict)

    return total_loss

def evaluate_testset(test_dataset, model):
    # returns accuracy and plots confusion matrix
    test_loader = DataLoader(test_dataset, batch_size = test_dataset.__len__()) # vectorise
    [(_,(testX, testy))] = [c for c in enumerate(test_loader)]
    predictions = torch.argmax(model(testX),-1)
    print(f"Test Accuracy: {torch.sum(predictions == testy)}/{ test_dataset.__len__()} correct")
    sns.heatmap(confusion_matrix(testy, predictions))

def evaluate_testset(test_dataset, mModel, print_result = True):
    # returns accuracy and plots confusion matrix
    with torch.inference_mode():
        mModel.eval() # turn off dropout
        test_loader = DataLoader(test_dataset, batch_size = test_dataset.__len__()) # vectorise
        [(_,(testX, testy))] = [c for c in enumerate(test_loader)]
        n = testX.shape[-1]

        correct = 0
        total = len(testy.flatten())//testy.shape[-1] # total number of tests

        use_lstm = False
        if mModel.__class__.__name__ == 'SeqModel':
            use_lstm = True

        if use_lstm:
            predictions = torch.argmax(mModel(testX), -1)
            actual = torch.argmax(testy,-1)
            correct = torch.sum(predictions == actual)
        else:
            prev_sum = torch.zeros((test_dataset.__len__(), n), dtype=torch.float) #probabilities
            for i in range(testy.shape[1]):
                prev_sum = mModel(testX[:,i,:], prev_sum)
                predictions = torch.argmax(prev_sum, -1)
                actual = torch.argmax(testy[:,i,:],-1)
                #print(predictions.shape, actual.shape)
                correct += torch.sum(predictions == actual)

        # I wonder if later values are harder to predict
        if print_result:
            print(f"Evaluate on test set: {correct} out of {total} values correct")
        mModel.train() # turn dropout back on
        return correct.item()/total
```

In [8]:

```python
N = 3
seq_len = 20
num_train = 5000
num_test = 1000
PS_train = PrefixSumDataset(N, seq_len, num_train)
PS_test = PrefixSumDataset(N, seq_len, num_test)
```

In [107…

```python
optimizer_fn = torch.optim.Adam
LR = 1e-3
L2REG = 1e-6
model_agg = AggregatorModel(N, 20) # hidden dim size 20
teacher_probability = 0.5
loss_fn = nn.BCELoss()
```

In [108…

```python
optimizer_agg = optimizer_fn(model_agg.parameters(), lr=LR, weight_decay = L2REG)
epochs = 8

for i in range(epochs):
    print(f"Iteration {i}:")
    print(f"training loss: {trainloop(PS_train, model_agg,loss_fn, optimizer_agg, teacher_probability)}")
    evaluate_testset(PS_test, model_agg)
```

```
Iteration 0:
250it [00:04, 57.49it/s]
training loss: 3070.762818336487
Evaluate on test set: 7445 out of 20000 values correct
Iteration 1:
250it [00:04, 59.82it/s]
training loss: 2609.641504764557
Evaluate on test set: 7442 out of 20000 values correct
Iteration 2:
250it [00:04, 60.70it/s]
training loss: 2247.430208683014
Evaluate on test set: 9431 out of 20000 values correct
Iteration 3:
250it [00:03, 62.94it/s]
training loss: 2112.8823947906494
Evaluate on test set: 12509 out of 20000 values correct
Iteration 4:
250it [00:04, 58.32it/s]
training loss: 1909.3351097106934
Evaluate on test set: 19639 out of 20000 values correct
Iteration 5:
250it [00:03, 62.53it/s]
training loss: 1535.3752093315125
Evaluate on test set: 20000 out of 20000 values correct
Iteration 6:
250it [00:04, 62.05it/s]
training loss: 1213.1252872943878
Evaluate on test set: 20000 out of 20000 values correct
Iteration 7:
250it [00:04, 62.22it/s]
training loss: 1277.7330566644669
Evaluate on test set: 20000 out of 20000 values correct
```

In [109…

```python
with torch.inference_mode(): #cuz I got dropout going
    model_agg.eval()
    for i in range(N):
        for j in range(N):
            V1 = F.one_hot(torch.tensor(i, dtype=torch.long), num_classes=N).float()
            V2 = F.one_hot(torch.tensor(j, dtype=torch.long), num_classes=N).float()
            print(model_agg(V1, V2), i,j)
    model_agg.train()
```

```
tensor([0.9745, 0.0105, 0.0150]) 0 0
tensor([0.0315, 0.9433, 0.0252]) 0 1
tensor([0.0346, 0.0640, 0.9014]) 0 2
tensor([0.0343, 0.9385, 0.0272]) 1 0
tensor([0.0321, 0.0438, 0.9240]) 1 1
tensor([0.8599, 0.0960, 0.0441]) 1 2
tensor([0.0277, 0.0205, 0.9519]) 2 0
tensor([0.8865, 0.0278, 0.0857]) 2 1
tensor([0.0292, 0.9043, 0.0665]) 2 2
```

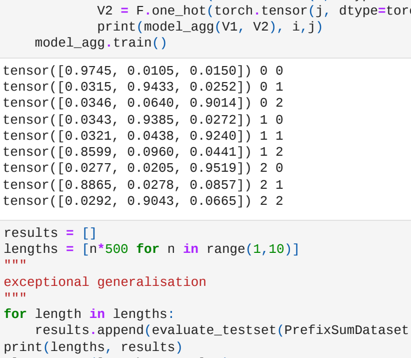
In [111…

```python
results = []
lengths = [n*500 for n in range(1,10)]
"""
exceptional generalisation
"""
for length in lengths:
    results.append(evaluate_testset(PrefixSumDataset(N, length, num_test), model_agg, False))
print(lengths, results)
plt.scatter(lengths, results)
```

```
[500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500] [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

Out[111]:

```
<matplotlib.collections.PathCollection at 0x156c38397c0>
```



In [18]:

```python
#torch.save(model_agg.state_dict(), 'model_aggregator_1.pth')
```

Out[18]:

```
<All keys matched successfully>
```

In [10]:

```python
model_agg.load_state_dict(torch.load("model_aggregator_1.pth"))
```

Out[10]:

```
<All keys matched successfully>
```

In [22]:

```python
# no teacher forcing
# it gets there pretty quick but a bit slower than with teacher forcing
model_agg_no_tf = AggregatorModel(N, 20)
optimizer_agg_no_tf = optimizer_fn(model_agg_no_tf.parameters(), lr=LR, weight_decay = L2REG)
epochs = 15

for i in range(epochs):
    print(f"Iteration {i}:")
    print(f"training loss: {trainloop(PS_train, model_agg_no_tf,loss_fn, optimizer_agg_no_tf, 0)}")
    evaluate_testset(PS_test,model_agg_no_tf)
```

```
Iteration 0:
250it [00:04, 50.15it/s]
training loss: 3179.372380256653
Evaluate on test set: 7375 out of 20000 values correct
Iteration 1:
250it [00:09, 27.45it/s]
training loss: 3163.3683528900146
Evaluate on test set: 7375 out of 20000 values correct
Iteration 2:
250it [00:04, 51.75it/s]
training loss: 3139.0282850265503
Evaluate on test set: 7494 out of 20000 values correct
Iteration 3:
250it [00:07, 35.13it/s]
training loss: 3104.0846633911133
Evaluate on test set: 7526 out of 20000 values correct
Iteration 4:
250it [00:04, 56.33it/s]
training loss: 3059.462224006653
Evaluate on test set: 7760 out of 20000 values correct
Iteration 5:
250it [00:05, 46.28it/s]
training loss: 3017.513150215149
Evaluate on test set: 8408 out of 20000 values correct
Iteration 6:
250it [00:05, 42.78it/s]
training loss: 2979.7695808410645
Evaluate on test set: 8927 out of 20000 values correct
Iteration 7:
250it [00:05, 49.35it/s]
training loss: 2936.7332525253296
Evaluate on test set: 9381 out of 20000 values correct
Iteration 8:
250it [00:04, 55.87it/s]
training loss: 2827.7859449386597
Evaluate on test set: 14113 out of 20000 values correct
Iteration 9:
250it [00:04, 51.80it/s]
training loss: 2497.061755657196
Evaluate on test set: 19997 out of 20000 values correct
Iteration 10:
250it [00:05, 46.23it/s]
training loss: 1976.3370780944824
Evaluate on test set: 20000 out of 20000 values correct
Iteration 11:
250it [00:04, 51.53it/s]
training loss: 1544.2978539466858
Evaluate on test set: 20000 out of 20000 values correct
Iteration 12:
250it [00:04, 54.53it/s]
training loss: 1376.1043405532837
Evaluate on test set: 20000 out of 20000 values correct
Iteration 13:
250it [00:04, 56.36it/s]
training loss: 1292.2302622795105
Evaluate on test set: 20000 out of 20000 values correct
Iteration 14:
250it [00:04, 55.38it/s]
training loss: 1159.765312075615
Evaluate on test set: 20000 out of 20000 values correct
```

In [72]:

```python
PS_train_long = PrefixSumDataset(N, 500, 1000)
```

In [78]:

```python
# does no truncation still work if we have very long sequences?
model_agg_no_tf_no_truncate = AggregatorModel(N, 20)
optimizer_agg_no_tf_no_truncate = optimizer_fn(model_agg_no_tf_no_truncate.parameters(), lr=3e-3, weight_decay = 0)
epochs = 10

for i in range(epochs):
    print(f"Iteration {i}:")
    print(f"training loss: {trainloop(PS_train_long , model_agg_no_tf_no_truncate,loss_fn, optimizer_agg_no_tf_no_truncate, 0, 1000)}")
    evaluate_testset(PS_test,model_agg_no_tf_no_truncate)
```

```
Iteration 0:
50it [00:22,  2.21it/s]
training loss: 15926.431915283203
Evaluate on test set: 7089 out of 20000 values correct
Iteration 1:
50it [00:18,  2.76it/s]
training loss: 15915.798034667969
Evaluate on test set: 7421 out of 20000 values correct
Iteration 2:
50it [00:16,  2.98it/s]
training loss: 15914.5537109375
Evaluate on test set: 7420 out of 20000 values correct
Iteration 3:
50it [00:19,  2.56it/s]
training loss: 15913.419342041016
Evaluate on test set: 7297 out of 20000 values correct
Iteration 4:
50it [00:17,  2.79it/s]
training loss: 15913.136352539062
Evaluate on test set: 7258 out of 20000 values correct
Iteration 5:
50it [00:17,  2.88it/s]
training loss: 15912.357330322266
Evaluate on test set: 7286 out of 20000 values correct
Iteration 6:
50it [00:18,  2.76it/s]
training loss: 15912.246948242188
Evaluate on test set: 7071 out of 20000 values correct
Iteration 7:
50it [00:16,  3.05it/s]
training loss: 15911.882049560547
Evaluate on test set: 7184 out of 20000 values correct
Iteration 8:
50it [00:16,  2.97it/s]
training loss: 15911.733001708984
Evaluate on test set: 7421 out of 20000 values correct
Iteration 9:
50it [00:16,  3.12it/s]
training loss: 15911.049011230469
Evaluate on test set: 7421 out of 20000 values correct
```

In [79]:

```python
# how about with truncated gradients
# works about same
model_agg_no_tf_truncate_long = AggregatorModel(N, 20)
optimizer_agg_no_tf_truncate_long = optimizer_fn(model_agg_no_tf_truncate_long.parameters(), lr=3e-3, weight_decay = 0)
epochs = 10

for i in range(epochs):
    print(f"Iteration {i}:")
    print(f"training loss: {trainloop(PS_train_long , model_agg_no_tf_truncate_long,loss_fn, optimizer_agg_no_tf_truncate_long, 0, 50)}")
    evaluate_testset(PS_test,model_agg_no_tf_truncate_long)
```

```
Iteration 0:
50it [00:16,  3.09it/s]
training loss: 15938.273132324219
Evaluate on test set: 7103 out of 20000 values correct
Iteration 1:
50it [00:16,  3.06it/s]
training loss: 15915.856872558594
Evaluate on test set: 7057 out of 20000 values correct
Iteration 2:
50it [00:16,  3.03it/s]
training loss: 15912.405517578125
Evaluate on test set: 7600 out of 20000 values correct
Iteration 3:
50it [00:16,  3.09it/s]
training loss: 15911.503845214844
Evaluate on test set: 7451 out of 20000 values correct
Iteration 4:
50it [00:16,  3.07it/s]
training loss: 15908.551361083984
Evaluate on test set: 7421 out of 20000 values correct
Iteration 5:
50it [00:16,  3.06it/s]
training loss: 15906.133361816406
Evaluate on test set: 7307 out of 20000 values correct
Iteration 6:
50it [00:16,  3.12it/s]
training loss: 15904.23648071289
Evaluate on test set: 7421 out of 20000 values correct
Iteration 7:
50it [00:16,  3.03it/s]
training loss: 15901.664581298828
Evaluate on test set: 7642 out of 20000 values correct
Iteration 8:
50it [00:16,  3.11it/s]
training loss: 15898.991394042969
Evaluate on test set: 7534 out of 20000 values correct
Iteration 9:
50it [00:17,  2.92it/s]
training loss: 15896.304748535156
Evaluate on test set: 7536 out of 20000 values correct
```

In [104…

```python
# lstms do well training on size 20 samples
model_lstm = SeqModel(N, 20)
optimizer_lstm = optimizer_fn(model_lstm.parameters(), lr=LR, weight_decay = L2REG)
epochs = 8

for i in range(epochs):
    print(f"Iteration {i}:")
    print(f"training loss: {trainloop(PS_train , model_lstm,loss_fn, optimizer_lstm, 0,0)}")
    evaluate_testset(PS_test,model_lstm)
```

```
Iteration 0:
250it [00:05, 48.63it/s]
training loss: 258.072349190712
Evaluate on test set: 7375 out of 20000 values correct
Iteration 1:
250it [00:05, 49.38it/s]
training loss: 257.70398807525635
Evaluate on test set: 7383 out of 20000 values correct
Iteration 2:
250it [00:04, 53.25it/s]
training loss: 254.9025342464447
Evaluate on test set: 7897 out of 20000 values correct
Iteration 3:
250it [00:05, 44.66it/s]
training loss: 251.5894876718521
Evaluate on test set: 9133 out of 20000 values correct
Iteration 4:
250it [00:05, 47.54it/s]
training loss: 234.26939922571182
Evaluate on test set: 19999 out of 20000 values correct
Iteration 5:
250it [00:04, 52.19it/s]
training loss: 177.63114911317825
Evaluate on test set: 20000 out of 20000 values correct
Iteration 6:
250it [00:04, 50.86it/s]
training loss: 172.67809599637985
Evaluate on test set: 20000 out of 20000 values correct
Iteration 7:
250it [00:04, 51.66it/s]
training loss: 171.2624163031578
Evaluate on test set: 20000 out of 20000 values correct
```

In [ ]:

```python
evaluate_testset(PrefixSumDataset(N, 2000, 1000),model_lstm)
```

In [103…

```python
# lstms do well training on size 20 samples
# but it cant learn on more than that
model_lstm2 = SeqModel(N, 20)
optimizer_lstm2 = optimizer_fn(model_lstm2.parameters(), lr=1e-2, weight_decay = 0)
epochs = 8

for i in range(epochs):
    print(f"Iteration {i}:")
    print(f"training loss: {trainloop(PS_train_long , model_lstm2,loss_fn, optimizer_lstm2, 0,0)}")
    evaluate_testset(PS_test,model_lstm2)
```

```
Iteration 0:
50it [00:16,  3.10it/s]
training loss: 51.641745924949646
Evaluate on test set: 6739 out of 20000 values correct
Iteration 1:
50it [00:18,  2.66it/s]
training loss: 51.64139652252197
Evaluate on test set: 6744 out of 20000 values correct
Iteration 2:
50it [00:21,  2.36it/s]
training loss: 51.64139246940613
Evaluate on test set: 6756 out of 20000 values correct
Iteration 3:
50it [00:19,  2.60it/s]
training loss: 51.64142048358917
Evaluate on test set: 6757 out of 20000 values correct
Iteration 4:
50it [00:18,  2.68it/s]
training loss: 51.641589760780334
Evaluate on test set: 6757 out of 20000 values correct
Iteration 5:
50it [00:16,  3.11it/s]
training loss: 51.64180362224579
Evaluate on test set: 6756 out of 20000 values correct
Iteration 6:
50it [00:16,  3.04it/s]
training loss: 51.64198422431946
Evaluate on test set: 6759 out of 20000 values correct
Iteration 7:
50it [00:19,  2.59it/s]
training loss: 51.64186716079712
Evaluate on test set: 6759 out of 20000 values correct
```

In [ ]:

In [ ]: