$$h^{(t)} = \tanh\left(W_{hx} \cdot x^{(t)} + W_{hh} \cdot h^{(t-1)} + b_h\right) \quad (1)$$

$$p^{(t)} = W_{ph} \cdot h^{(t)} + b_p \quad (2)$$

$$\hat{y}(t) = \text{softmax}\left(p^{(t)}\right) \qquad K = 10 \Rightarrow \text{Digits}$$

$$\mathcal{L} = -\sum_{k=1}^{K} y_k \cdot \log\left(\hat{y}_k\right)$$

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W_{ph}} = \underbrace{\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{y}_k^{(T)}}}_{①} \cdot \underbrace{\frac{\partial \hat{y}_i^{(T)}}{\partial p_j^{(T)}}}_{②} \cdot \underbrace{\frac{\partial p^{(T)}}{\partial W_{ph}^{(T)}}}_{③}$$

① $$\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{y}^{(T)}} = -\sum_{k=1}^{K} \frac{y_k}{\hat{y}^{(T)}}$$

② $$\frac{\partial \hat{y}_i^{(T)}}{\partial p_j^{(T)}} = \hat{y} \cdot \left(\delta_{ij} - y_j\right)$$

↑ Kronecker-Delta: 1 for $i = j$, 0 for $i \neq j$

③ $$\frac{\partial p^{(T)}}{\partial W_{ph}^{(T)}} = h^{(T)}$$

$$\Rightarrow \frac{\partial \mathcal{L}^{(T)}}{\partial W_{ph}} = -\sum_{k=1}^{K} \frac{y_k}{\hat{y}_k^{(T)}} \cdot \hat{y}_k^{(T)} \left(\delta_{kj} - \hat{y}_j\right) \cdot h^{(T)}$$

$$\Leftrightarrow \underbrace{\sum_{k=1}^{K} -y_k \cdot \delta_{kj}}_{\text{if } k=j \Rightarrow -1,\ \text{else } 0} \cdot h^{(T)} + \underbrace{\sum_{k=1}^{K} y_k}_{=1} \cdot \hat{y}_j \cdot h^{(T)}$$

$$\Leftrightarrow -y_k \cdot h^{(T)} + \hat{y}_j \cdot h^{(T)}$$

$$\Leftrightarrow \underline{h^{(T)} \cdot \left(\hat{y}_j - y_k\right)}$$

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W_{hh}} = \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{y}_k^{(T)}} \cdot \frac{\partial y}{\partial p^{(T)}} \cdot \frac{\partial p^{(T)}}{\partial h^{(T)}} \cdot \frac{\partial h^{(T)}}{\partial W_{hh}}$$

① ② ④ ⑤

We already have ① and ②

④ $\frac{\partial p^{(T)}}{\partial h^{(T)}} = W_{ph}$

Substitute Function to apply chain rule

$$\Rightarrow f(T) = W_{hx} \cdot x^{(T)} + W_{hh} \cdot h^{(T-1)} + b_h$$

$$\Rightarrow h(T) = \tanh(f(T))$$

$$\Rightarrow$$ ⑤ $\frac{\partial h^{(T)}}{\partial W_{hh}} = \frac{\partial h^{(T)}}{\partial f^{(T)}} \cdot \frac{\partial f^{(T)}}{\partial W_{hh}}$ ⑥ ⑦

⑥ $\frac{\partial h^{(T)}}{\partial f^{(T)}} = 1 - \tanh^2(f(T)) = 1 - h(T)^2$

⑦ $\frac{\partial f^{(T)}}{\partial W_{hh}} = h^{(T-1)}$

Put everything Together

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W_{hh}} = \sum_{t=1}^{T} \sum_{k=1}^{K} \left(\frac{-y_k}{\hat{y}_k}\right) \cdot \hat{y}_k \cdot (\delta_{kj} - \hat{y}_j) \cdot W_{ph} \cdot (1 - h^{(T)2}) \cdot h^{(T-1)}$$

Apply the same Transformations as above

$$\Rightarrow \sum_{t=1}^{T} (\hat{y} - y) \cdot W_{ph} \cdot (1 - h^{(T)2}) \cdot h^{(T-1)}$$

The first gradient depends only on the current Timestep, while the second Gradient also depends on former Timesteps

The second derivative has the problem for longer Timesteps, that the derivative chain will get very long. ⇒ Vanishing Gradients

If the activations get very big, there can also happen ⇒ Exploding Gradients

```python
class VanillaRNN(nn.Module):

    def __init__(
        self, seq_length, input_dim, num_hidden, num_classes, batch_size, device="cpu"
    ):
        super(VanillaRNN, self).__init__()
        self.seq_length = seq_length
        self.num_hidden = num_hidden
        self.num_classes = num_classes
        self.device = device
        self.batch_size = batch_size

        self.W_hx = torch.nn.Parameter(torch.randn(num_hidden, input_dim))
        self.W_hh = torch.nn.Parameter(torch.randn(num_hidden, num_hidden))
        self.W_ph = torch.nn.Parameter(torch.randn(num_classes, num_hidden))
        self.b_h = torch.nn.Parameter(torch.randn(num_hidden))
        self.b_p = torch.nn.Parameter(torch.randn(num_classes))

    def forward(self, x):
        h = torch.zeros(self.batch_size, self.num_hidden).to(self.device)
        for t in range(self.seq_length):
            x_t = x[:, t:t+1] # preserve 2 dimensions
            # Calculate the output as described in equation (1)
            h = torch.tanh(x_t @ self.W_hx.t() + h @ self.W_hh.t() + self.b_h)
        # Calculate the digit prediction for the last step
        output = h @ self.W_ph.t() + self.b_p
        return output
```

```python
class LSTM(nn.Module):

    def __init__(
        self, seq_length, input_dim, num_hidden, num_classes, batch_size, device="cpu"
    ):
        super(LSTM, self).__init__()
        self.seq_length = seq_length
        self.input_dim = input_dim
        self.num_hidden = num_hidden
        self.num_classes = num_classes
        self.device = device
        self.batch_size = batch_size

        # Input modulation gate
        self.W_gx = torch.nn.Parameter(torch.randn(num_hidden, input_dim))
        self.W_gh = torch.nn.Parameter(torch.randn(num_hidden, num_hidden))
        self.b_g = torch.nn.Parameter(torch.randn(num_hidden))
        # Input gate
        self.W_ix = torch.nn.Parameter(torch.randn(num_hidden, input_dim))
        self.W_ih = torch.nn.Parameter(torch.randn(num_hidden, num_hidden))
        self.b_i = torch.nn.Parameter(torch.randn(num_hidden))
        # Forget gate
        self.W_fx = torch.nn.Parameter(torch.randn(num_hidden, input_dim))
        self.W_fh = torch.nn.Parameter(torch.randn(num_hidden, num_hidden))
        self.b_f = torch.nn.Parameter(torch.randn(num_hidden))
        # Output gate
        self.W_ox = torch.nn.Parameter(torch.randn(num_hidden, input_dim))
        self.W_oh = torch.nn.Parameter(torch.randn(num_hidden, num_hidden))
        self.b_o = torch.nn.Parameter(torch.randn(num_hidden))
        # Linear
        self.W_ph = torch.nn.Parameter(torch.randn(num_classes, num_hidden))
        self.b_p = torch.nn.Parameter(torch.randn(num_classes))

    def forward(self, x):
        c = torch.zeros(self.batch_size, self.num_hidden).to(self.device) # Cell state
        h = torch.zeros(self.batch_size, self.num_hidden).to(self.device)

        for t in range(self.seq_length):
            x_t = x[:, t:t+1] # preserve 2 dimensions

            # Input modulation gate
            g = torch.tanh(x_t @ self.W_gx.t() + h @ self.W_gh.t() + self.b_g)
            # Input gate
            i = torch.sigmoid(x_t @ self.W_ix.t() + h @ self.W_ih.t() + self.b_i)
            # Forget gate
            f = torch.sigmoid(x_t @ self.W_fx.t() + h @ self.W_fh.t() + self.b_f)
            # Output gate
            o = torch.sigmoid(x_t @ self.W_ox.t() + h @ self.W_oh.t() + self.b_o)
            # Calculate next cell state
            c = g * i + c * f
            # Calculate next hidden state
            h = (torch.tanh(c) * o).to(self.device)
            # Calculate output
            p = torch.einsum('ch,bh->bc',self.W_ph, h) + self.b_p

        output = p

        return output
```

```python
def train(config):
    assert config.model_type in ("RNN", "LSTM")
    print("Chosen model type:", config.model_type)
    # Initialize the device which to run the model on
    device = torch.device(config.device)

    # Initialize the model that we are going to use
    model = VanillaRNN(
        config.input_length,
        config.input_dim,
        config.num_hidden,
        config.num_classes,
        config.batch_size,
        device) if config.model_type == "RNN" else LSTM(
        config.input_length,
        config.input_dim,
        config.num_hidden,
        config.num_classes,
        config.batch_size,
        device
        )
    model = model.to(device)

    # Initialize the dataset and data loader (note the +1)
    dataset = PalindromeDataset(config.input_length + 1)
    data_loader = DataLoader(dataset, config.batch_size, num_workers=1)

    # Setup the loss and optimizer
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.RMSprop(model.parameters(), lr=config.learning_rate)
    accuracies = []
    for step, (batch_inputs, batch_targets) in enumerate(data_loader):

        # Only for time measurement of step through network
        t1 = time.time()

        batch_inputs = batch_inputs.to(device)
        batch_targets = batch_targets.to(device)
        batch_predictions = model(batch_inputs)

        ###########################################################################
        # QUESTION: what happens here and why?
        ###########################################################################
        torch.nn.utils.clip_grad_norm(model.parameters(), max_norm=config.max_norm)
        ###########################################################################
        # Clips gradients to prevent exploding gradients
        # => Ensuring stable training.
        # max_norm limits gradient magnitude: values higher than the max_norm will turn into that value
```

```
    optimizer.zero_grad()

    loss = criterion(batch_predictions, batch_targets)
    accuracy = torch.mean((torch.argmax(batch_predictions, dim=1) == batch_targets).float())
    accuracies.append(accuracy.cpu().detach().numpy())

    # Backward pass
    loss.backward()
    optimizer.step()

    # Just for time measurement
    t2 = time.time()
    examples_per_second = config.batch_size / float(t2 - t1)

    if step % 1000 == 0:
        print(
            "[{}] Train Step {:04d}/{:04d}, Batch Size = {}, Examples/Sec = {:.2f}, "
            "Accuracy = {:.2f}, Loss = {:.3f}".format(
                datetime.now().strftime("%Y-%m-%d %H:%M"),
                step,
                config.train_steps,
                config.batch_size,
                examples_per_second,
                accuracy,
                loss,
            )
        )

    if step == config.train_steps:
        # If you receive a PyTorch data-loader error, check this bug report:
        # https://github.com/pytorch/pytorch/pull/9655
        break

print("Done training.")
return model, accuracies


################################################################################
################################################################################
```

notebook

June 27, 2024

[ ]:
```
import torch
import torch.nn as nn
import numpy as np
from part1.vanilla_rnn import VanillaRNN
from part1.train import train

import matplotlib.pyplot as plt
```

0.0.1 Task 1.3 Testing out RNN

[ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[ ]:
```
# Define a configuration object, matching the structure expected by `train`
class Config:
    def __init__(self,
                 model_type="RNN",
                 input_length=10,
                 input_dim=1,
                 num_classes=10,
                 num_hidden=128,
                 batch_size=128,
                 learning_rate=0.001,
                 train_steps=10000,
                 max_norm = 10.0,
                 device="cpu"
                 ):
        self.model_type = model_type
        self.input_length = input_length
        self.input_dim = input_dim
        self.num_classes = num_classes
        self.num_hidden = num_hidden
        self.batch_size = batch_size
        self.learning_rate = learning_rate
        self.train_steps = train_steps
        self.max_norm = max_norm
        self.device = device
```

[ ]:
```python
# Create models for palindromes with increasing length
print("Training RNN with Palindrome length of 5")
config5 = Config(input_length=5, device=device)
model5, accuracies5 = train(config5)

print("Training RNN with Palindrome length of 6")
config6 = Config(input_length=6, device=device)
model6, accuracies6 = train(config6)

print("Training RNN with Palindrome length of 7")
config7 = Config(input_length=7, device=device)
model7, accuracies7 = train(config7)

print("Training RNN with Palindrome length of 8")
config8 = Config(input_length=8, device=device)
model8, accuracies8 = train(config8)

print("Training RNN with Palindrome length of 9")
config9 = Config(input_length=9, device=device)
model9, accuracies9 = train(config9)

print("Training RNN with Palindrome length of 10")
config10 = Config(input_length=10, device=device)
model10, accuracies10 = train(config10)
```

```
Training RNN with Palindrome length of 5
Chosen model type: RNN

p:\Masterstudium\SS24\GenAIVisSynth\tutorial\9tutorial\part1\train.py:82:
UserWarning: torch.nn.utils.clip_grad_norm is now deprecated in favor of
torch.nn.utils.clip_grad_norm_.
  torch.nn.utils.clip_grad_norm(model.parameters(), max_norm=config.max_norm)

[2024-06-27 21:59] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
920.61, Accuracy = 0.19, Loss = 12.561
[2024-06-27 21:59] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
32008.04, Accuracy = 0.87, Loss = 0.318
[2024-06-27 21:59] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
25595.75, Accuracy = 1.00, Loss = 0.028
[2024-06-27 21:59] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
43415.08, Accuracy = 0.99, Loss = 0.026
[2024-06-27 21:59] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
42673.15, Accuracy = 0.93, Loss = 0.257
[2024-06-27 21:59] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
32004.23, Accuracy = 1.00, Loss = 0.013
[2024-06-27 21:59] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
32023.32, Accuracy = 0.97, Loss = 0.064
[2024-06-27 22:00] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
31981.35, Accuracy = 1.00, Loss = 0.002
```

```
[2024-06-27 22:00] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
42652.81, Accuracy = 1.00, Loss = 0.001
[2024-06-27 22:00] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
31947.09, Accuracy = 1.00, Loss = 0.004
[2024-06-27 22:00] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
31998.50, Accuracy = 1.00, Loss = 0.001
Done training.
Training RNN with Palindrome length of 6
Chosen model type: RNN
[2024-06-27 22:00] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
25603.08, Accuracy = 0.05, Loss = 16.808
[2024-06-27 22:00] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
25610.40, Accuracy = 0.62, Loss = 1.606
[2024-06-27 22:00] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
21226.07, Accuracy = 0.72, Loss = 1.179
[2024-06-27 22:00] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
32002.32, Accuracy = 0.87, Loss = 0.428
[2024-06-27 22:00] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
21349.30, Accuracy = 0.92, Loss = 0.203
[2024-06-27 22:00] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
25616.51, Accuracy = 1.00, Loss = 0.010
[2024-06-27 22:00] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
21319.63, Accuracy = 0.99, Loss = 0.061
[2024-06-27 22:01] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
32025.23, Accuracy = 0.91, Loss = 0.190
[2024-06-27 22:01] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
42652.81, Accuracy = 0.98, Loss = 0.032
[2024-06-27 22:01] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
32000.41, Accuracy = 0.95, Loss = 0.160
[2024-06-27 22:01] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
21334.03, Accuracy = 0.99, Loss = 0.018
Done training.
Training RNN with Palindrome length of 7
Chosen model type: RNN
[2024-06-27 22:01] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
21276.54, Accuracy = 0.12, Loss = 15.023
[2024-06-27 22:01] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
21465.39, Accuracy = 0.42, Loss = 4.749
[2024-06-27 22:01] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
32082.64, Accuracy = 0.40, Loss = 3.038
[2024-06-27 22:01] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
25606.74, Accuracy = 0.51, Loss = 1.698
[2024-06-27 22:01] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
32021.41, Accuracy = 0.60, Loss = 1.007
[2024-06-27 22:01] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
18291.40, Accuracy = 0.74, Loss = 0.511
[2024-06-27 22:01] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
32269.69, Accuracy = 0.77, Loss = 0.573
```

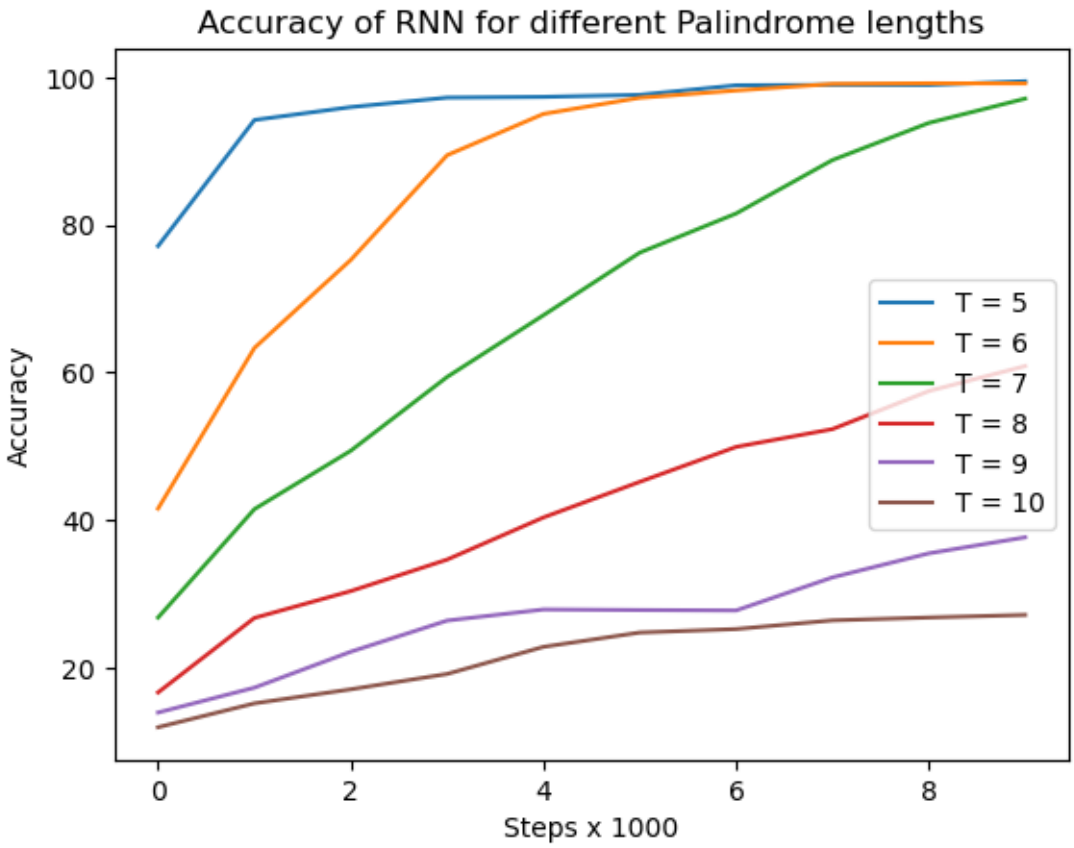
```
[2024-06-27 22:02] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
21329.79, Accuracy = 0.70, Loss = 0.700
[2024-06-27 22:02] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
32000.41, Accuracy = 0.95, Loss = 0.166
[2024-06-27 22:02] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
32008.04, Accuracy = 0.95, Loss = 0.118
[2024-06-27 22:02] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
24125.78, Accuracy = 0.98, Loss = 0.034
Done training.
Training RNN with Palindrome length of 8
Chosen model type: RNN
[2024-06-27 22:02] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
25626.30, Accuracy = 0.10, Loss = 15.064
[2024-06-27 22:02] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
21336.58, Accuracy = 0.27, Loss = 6.125
[2024-06-27 22:02] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
21340.82, Accuracy = 0.27, Loss = 3.887
[2024-06-27 22:02] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
21449.95, Accuracy = 0.34, Loss = 2.294
[2024-06-27 22:02] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
32240.63, Accuracy = 0.32, Loss = 1.909
[2024-06-27 22:02] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
21327.25, Accuracy = 0.36, Loss = 1.549
[2024-06-27 22:03] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
32015.68, Accuracy = 0.49, Loss = 1.351
[2024-06-27 22:03] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
25991.04, Accuracy = 0.52, Loss = 1.173
[2024-06-27 22:03] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
25588.43, Accuracy = 0.53, Loss = 1.054
[2024-06-27 22:03] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
25450.15, Accuracy = 0.53, Loss = 1.017
[2024-06-27 22:03] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
21332.34, Accuracy = 0.60, Loss = 0.930
Done training.
Training RNN with Palindrome length of 9
Chosen model type: RNN
[2024-06-27 22:03] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
18281.44, Accuracy = 0.11, Loss = 15.018
[2024-06-27 22:03] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
21330.64, Accuracy = 0.16, Loss = 7.571
[2024-06-27 22:03] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
25605.52, Accuracy = 0.23, Loss = 3.884
[2024-06-27 22:03] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
25407.99, Accuracy = 0.23, Loss = 2.621
[2024-06-27 22:04] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
32296.87, Accuracy = 0.26, Loss = 2.193
[2024-06-27 22:04] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
32396.27, Accuracy = 0.28, Loss = 1.948
```

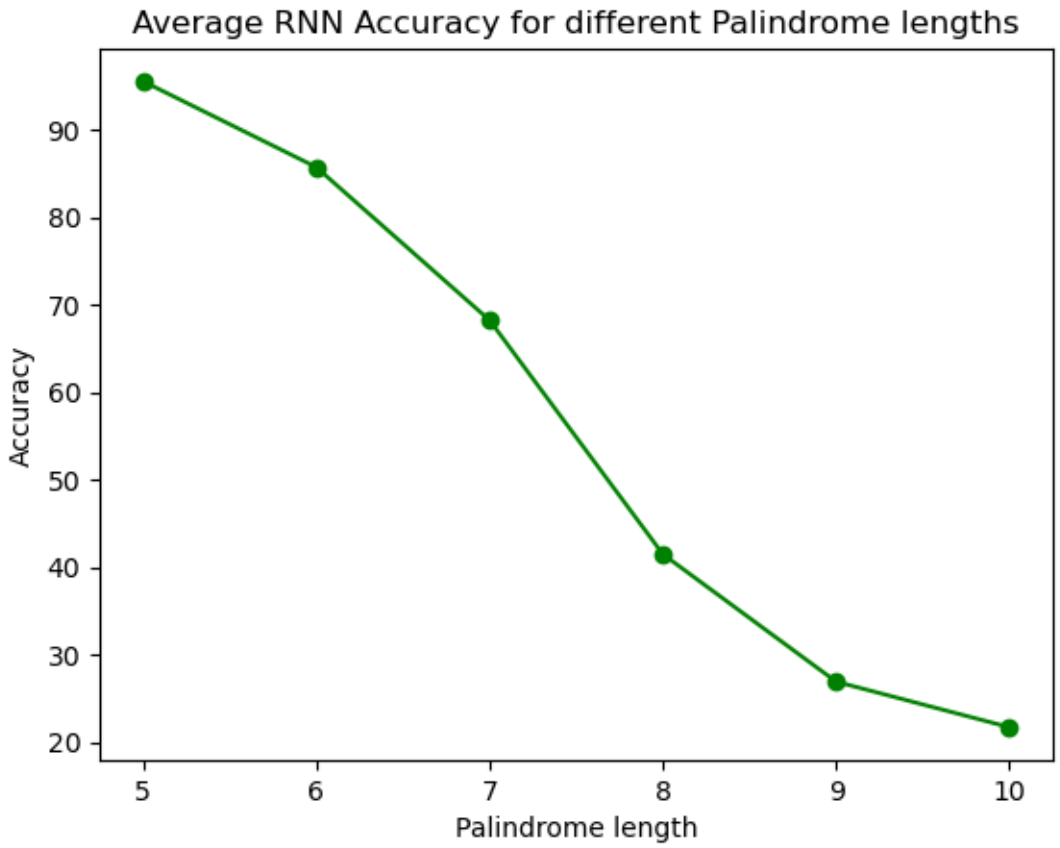
```
[2024-06-27 22:04] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
25603.08, Accuracy = 0.26, Loss = 1.938
[2024-06-27 22:04] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
18287.66, Accuracy = 0.33, Loss = 1.667
[2024-06-27 22:04] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
18281.44, Accuracy = 0.35, Loss = 1.675
[2024-06-27 22:04] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
25589.65, Accuracy = 0.30, Loss = 1.659
[2024-06-27 22:04] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
25601.86, Accuracy = 0.35, Loss = 1.394
Done training.
Training RNN with Palindrome length of 10
Chosen model type: RNN
[2024-06-27 22:04] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
21334.03, Accuracy = 0.13, Loss = 17.432
[2024-06-27 22:04] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
25594.53, Accuracy = 0.17, Loss = 7.254
[2024-06-27 22:04] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
21344.21, Accuracy = 0.15, Loss = 4.830
[2024-06-27 22:04] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
21279.91, Accuracy = 0.23, Loss = 2.705
[2024-06-27 22:05] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
21284.13, Accuracy = 0.17, Loss = 2.448
[2024-06-27 22:05] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
18210.13, Accuracy = 0.20, Loss = 2.164
[2024-06-27 22:05] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
15996.87, Accuracy = 0.26, Loss = 1.941
[2024-06-27 22:05] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
25759.09, Accuracy = 0.34, Loss = 1.870
[2024-06-27 22:05] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
25605.52, Accuracy = 0.21, Loss = 2.043
[2024-06-27 22:05] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
21447.38, Accuracy = 0.30, Loss = 1.904
[2024-06-27 22:05] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
25447.74, Accuracy = 0.34, Loss = 1.735
Done training.
```

[ ]:
```python
def average_accuracies(accuracies, training_steps):
  step_size = training_steps // 10
  range = np.arange(0, len(accuracies)-step_size, step_size)
  accuracies_avg = [np.mean(accuracies[i:i+step_size])*100 for i in range]
  return accuracies_avg
```

[ ]:
```python
accuracies5_avg = average_accuracies(accuracies5, config5.train_steps)
accuracies6_avg = average_accuracies(accuracies6, config6.train_steps)
accuracies7_avg = average_accuracies(accuracies7, config7.train_steps)
accuracies8_avg = average_accuracies(accuracies8, config8.train_steps)
```

```
accuracies9_avg = average_accuracies(accuracies9, config9.train_steps)
accuracies10_avg = average_accuracies(accuracies10, config10.train_steps)

plt.plot(accuracies5_avg, label="T = 5")
plt.plot(accuracies6_avg, label="T = 6")
plt.plot(accuracies7_avg, label="T = 7")
plt.plot(accuracies8_avg, label="T = 8")
plt.plot(accuracies9_avg, label="T = 9")
plt.plot(accuracies10_avg, label="T = 10")
plt.xlabel(f"Steps x 1000")
plt.ylabel("Accuracy")
plt.title("Accuracy of RNN for different Palindrome lengths")
plt.legend()
plt.show()

# Plot accuracy x palindrome length plot
palindrome_accuracies = [
  np.mean(accuracies5)*100,
  np.mean(accuracies6)*100,
  np.mean(accuracies7)*100,
  np.mean(accuracies8)*100,
  np.mean(accuracies9)*100,
  np.mean(accuracies10)*100
  ]

plt.plot(range(5,11), palindrome_accuracies, "-go")
plt.xlabel("Palindrome length")
plt.ylabel("Accuracy")
plt.title("Average RNN Accuracy for different Palindrome lengths")
plt.show()
```


Accuracy of RNN for different Palindrome lengths
Accuracy
100
80
60
40
20
0
2
4
6
8
Steps x 1000
T = 5
T = 6
T = 7
T = 8
T = 9
T = 10



One can clearly see, that the RNN performs worse and worse with longer Palindrome lengths. An interesting finding is that it seems to perform better for uneven lengths than for even lengths. For example is the accuracy of T=7 with ~ 80% higher than the accuracy of T=6 with ~ 70% This might also be due to the model not yet completely converging.

[ ]:
```
torch.save(model5.state_dict(), 'model5_RNN.pth')
torch.save(model6.state_dict(), 'model6_RNN.pth')
torch.save(model7.state_dict(), 'model7_RNN.pth')
torch.save(model8.state_dict(), 'model8_RNN.pth')
torch.save(model9.state_dict(), 'model9_RNN.pth')
torch.save(model10.state_dict(), 'model10_RNN.pth')
```

0.0.2 Task 1.4 Benefits of momentum and adaptive learning rate

momentum: Momentum makes it harder for a weight update to change the direction and instead drives it in the general direction of the last couple of gradients. This makes jittering towards the local optima less strong and thus makes the model converge faster.

adaptive learning rate: The adaptive learning rate allows the model to always learn effectively, no matter if the model is currently in a very flat target space or very steep target space. If the gradients are very small, then the learning rate will be automatically increase, which will make

sure that the model continues to learn. If the gradients get very big, then the learning rate will automatically decrease, to avoid overshooting the local optima within 1 update step. This also adds to a faster convergence and more stable training.

1 Task 2: LSTMs

1.0.1 Task 1.5 Theory behind LSTMs

Subtask a input modulation gate g(t): The purpose of the input modulation gate is to apply a non-linear transformation (tanh) to the current input x(t) and h(t-1). The output of g(t) is a vector of candidates to be added to the cell state. The nonlinearity is important, so that the network is able to learn

input gate i(t): The purpose of the input gate is to decide what and how much of x(t) and h(t-1) should be added to the cell state. For that it employs the sigmoid function which scales the input from 0 to 1 thus deciding how much of each value in g(t) should be added to the cell state

forget gate f(t): The purpose of the forget gate is to decide what parts and how much of the cell state we should forget. For that they use a sigmoid function to introduce nonlinearity and scale the output between 0 and 1 for the forget gate. This especially makes sense, since the output of the forget gate will be multiplied elementwise with the cell state thus deciding for each of the elements of the cell state, how relevant they still are

output gate o(t): The purpose of the output gate is to decide what part of the cell state should be part of the output h(t). For that it again uses the sigmoid function on the input x(t) and h(t-1). The input and last hidden state influence what parts of the cell state should be transported to the next hidden state

Subtask b

- T = sequence length => no effect on number of weights
- d = feature dimensionality (input dim)
- n = number of units (hidden dim)
- m = batch size => no effect on number of weights
- p = output dimensionality

x = d

h = n

Parameters:

- W_gx = n * d
- W_gh = n * n
- b_g = n
- W_ix = n * d
- W_ih = n * n
- b_i = n
- W_fx = n * d

- W_fh = n * n
- b_f = n
- W_ox = n * d
- W_oh = n * n
- b_o = n
- W_ph = p * n
- b_p = p

Parameter formula:

$$4 \cdot (n \cdot d + n \cdot n + n) + p \cdot n + p$$

1.0.2 Task 1.6 Implement LSTM network

=> View lstm.py

[ ]:
```
print("Training LSTM with Palindrome length of 5")
config5_lstm = Config(model_type="LSTM", input_length=5, device=device)
model5_lstm, accuracies5_lstm = train(config5_lstm)

print("Training LSTM with Palindrome length of 6")
config6_lstm = Config(model_type="LSTM", input_length=6, device=device)
model6_lstm, accuracies6_lstm = train(config6_lstm)

print("Training LSTM with Palindrome length of 7")
config7_lstm = Config(model_type="LSTM", input_length=7, device=device)
model7_lstm, accuracies7_lstm = train(config7_lstm)

print("Training LSTM with Palindrome length of 8")
config8_lstm = Config(model_type="LSTM", input_length=8, device=device)
model8_lstm, accuracies8_lstm = train(config8_lstm)

print("Training LSTM with Palindrome length of 9")
config9_lstm = Config(model_type="LSTM", input_length=9, device=device)
model9_lstm, accuracies9_lstm = train(config9_lstm)

print("Training LSTM with Palindrome length of 10")
config10_lstm = Config(model_type="LSTM", input_length=10, device=device)
model10_lstm, accuracies10_lstm = train(config10_lstm)
```

```
Training LSTM with Palindrome length of 5
Chosen model type: LSTM

p:\Masterstudium\SS24\GenAIVisSynth\tutorial\9tutorial\part1\train.py:82:
UserWarning: torch.nn.utils.clip_grad_norm is now deprecated in favor of
```

```
torch.nn.utils.clip_grad_norm_.
  torch.nn.utils.clip_grad_norm(model.parameters(), max_norm=config.max_norm)
[2024-06-27 22:07] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
4929.67, Accuracy = 0.16, Loss = 8.463
[2024-06-27 22:08] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
7529.43, Accuracy = 1.00, Loss = 0.005
[2024-06-27 22:08] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
12801.23, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:08] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
7999.75, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:08] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
12799.71, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:08] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
12798.49, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:08] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
12840.42, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:09] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
14222.88, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:09] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
12761.98, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:09] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
11603.75, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:09] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
12840.73, Accuracy = 1.00, Loss = 0.000
Done training.
Training LSTM with Palindrome length of 6
Chosen model type: LSTM
[2024-06-27 22:09] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
11634.69, Accuracy = 0.10, Loss = 7.324
[2024-06-27 22:09] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
10659.18, Accuracy = 0.98, Loss = 0.075
[2024-06-27 22:10] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
11663.25, Accuracy = 1.00, Loss = 0.002
[2024-06-27 22:10] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
11708.78, Accuracy = 1.00, Loss = 0.001
[2024-06-27 22:10] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
10691.87, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:10] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
11666.54, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:10] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
11637.21, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:11] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
11008.45, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:11] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
12812.84, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:11] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
10673.17, Accuracy = 1.00, Loss = 0.000
```

```
[2024-06-27 22:11] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
12795.74, Accuracy = 1.00, Loss = 0.000
Done training.
Training LSTM with Palindrome length of 7
Chosen model type: LSTM
[2024-06-27 22:11] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
9145.08, Accuracy = 0.12, Loss = 7.574
[2024-06-27 22:11] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
10667.65, Accuracy = 0.83, Loss = 0.385
[2024-06-27 22:12] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
8000.46, Accuracy = 1.00, Loss = 0.052
[2024-06-27 22:12] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
10692.51, Accuracy = 0.99, Loss = 0.012
[2024-06-27 22:12] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
9163.50, Accuracy = 1.00, Loss = 0.004
[2024-06-27 22:12] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
10644.39, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:13] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
9844.88, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:13] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
9850.30, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:13] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
9845.78, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:13] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
9852.83, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:13] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
10692.94, Accuracy = 1.00, Loss = 0.000
Done training.
Training LSTM with Palindrome length of 8
Chosen model type: LSTM
[2024-06-27 22:14] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
8506.37, Accuracy = 0.11, Loss = 7.062
[2024-06-27 22:14] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
9119.14, Accuracy = 0.75, Loss = 0.548
[2024-06-27 22:14] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
9140.41, Accuracy = 0.99, Loss = 0.077
[2024-06-27 22:14] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
9867.50, Accuracy = 1.00, Loss = 0.005
[2024-06-27 22:14] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
9142.74, Accuracy = 1.00, Loss = 0.004
[2024-06-27 22:15] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
9162.25, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:15] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
8528.53, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:15] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
9846.87, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:15] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
9851.57, Accuracy = 1.00, Loss = 0.000
```

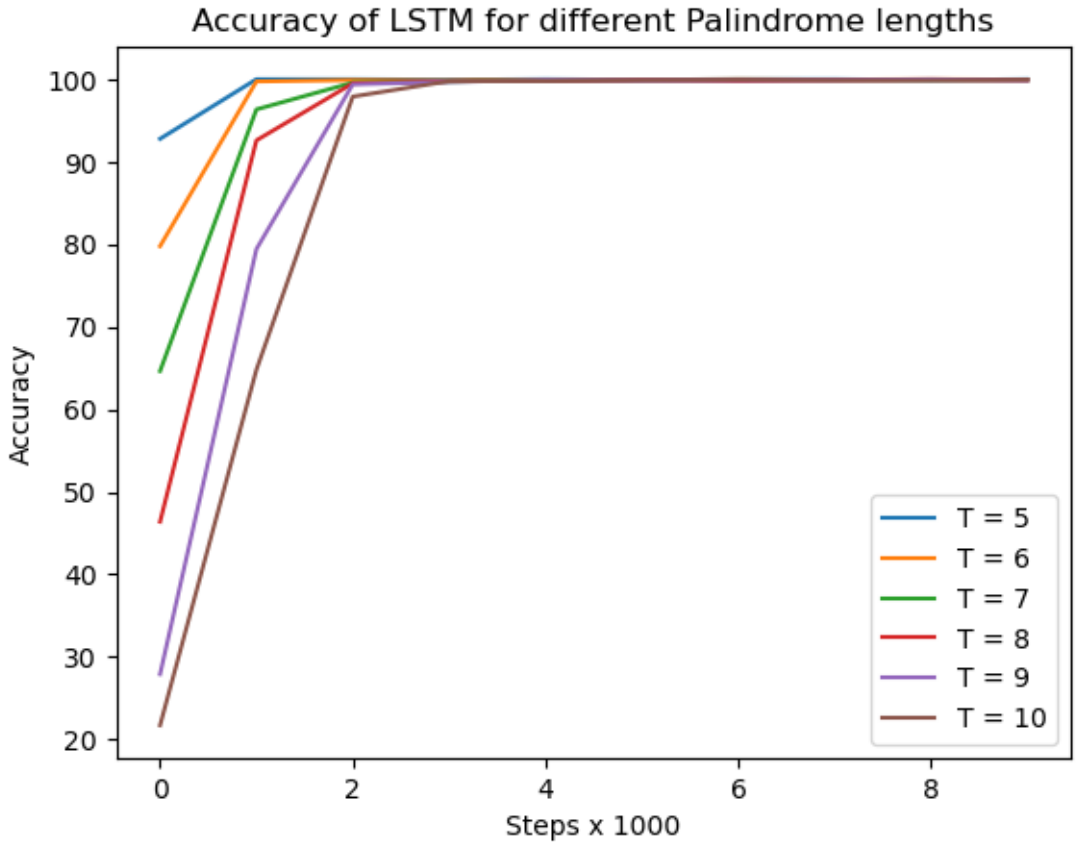
```
[2024-06-27 22:16] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
9143.68, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:16] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
7110.97, Accuracy = 1.00, Loss = 0.000
Done training.
Training LSTM with Palindrome length of 9
Chosen model type: LSTM
[2024-06-27 22:16] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
8528.94, Accuracy = 0.09, Loss = 8.242
[2024-06-27 22:16] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
8534.90, Accuracy = 0.51, Loss = 1.184
[2024-06-27 22:17] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
8536.53, Accuracy = 0.96, Loss = 0.115
[2024-06-27 22:17] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
7995.34, Accuracy = 1.00, Loss = 0.003
[2024-06-27 22:17] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
8533.27, Accuracy = 1.00, Loss = 0.002
[2024-06-27 22:17] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
9161.46, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:18] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
7101.28, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:18] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
9143.36, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:18] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
8533.68, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:18] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
9847.23, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:19] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
9117.74, Accuracy = 1.00, Loss = 0.000
Done training.
Training LSTM with Palindrome length of 10
Chosen model type: LSTM
[2024-06-27 22:19] Train Step 0000/10000, Batch Size = 128, Examples/Sec =
8019.10, Accuracy = 0.09, Loss = 9.400
[2024-06-27 22:19] Train Step 1000/10000, Batch Size = 128, Examples/Sec =
7997.00, Accuracy = 0.39, Loss = 1.869
[2024-06-27 22:19] Train Step 2000/10000, Batch Size = 128, Examples/Sec =
8533.14, Accuracy = 0.91, Loss = 0.314
[2024-06-27 22:20] Train Step 3000/10000, Batch Size = 128, Examples/Sec =
8533.41, Accuracy = 1.00, Loss = 0.018
[2024-06-27 22:20] Train Step 4000/10000, Batch Size = 128, Examples/Sec =
7998.79, Accuracy = 1.00, Loss = 0.002
[2024-06-27 22:20] Train Step 5000/10000, Batch Size = 128, Examples/Sec =
7999.86, Accuracy = 1.00, Loss = 0.001
[2024-06-27 22:20] Train Step 6000/10000, Batch Size = 128, Examples/Sec =
7999.75, Accuracy = 0.83, Loss = 0.685
[2024-06-27 22:21] Train Step 7000/10000, Batch Size = 128, Examples/Sec =
7999.98, Accuracy = 1.00, Loss = 0.000
```

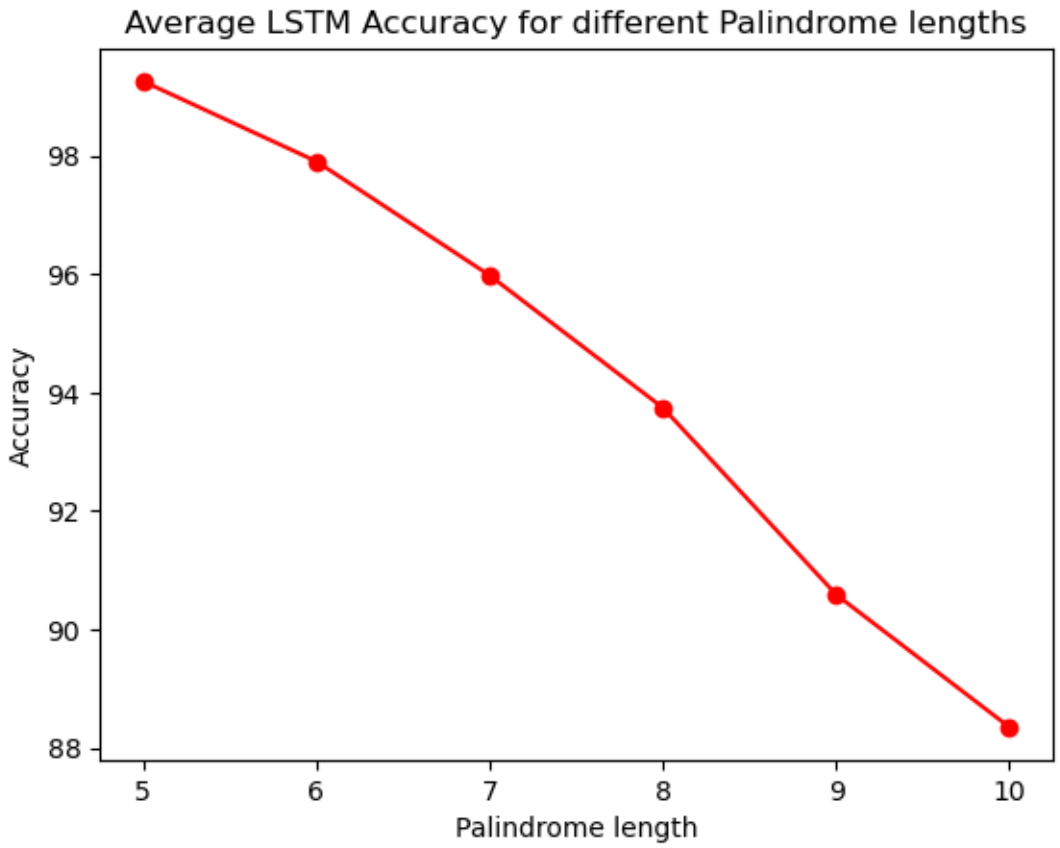
```
[2024-06-27 22:21] Train Step 8000/10000, Batch Size = 128, Examples/Sec =
7522.99, Accuracy = 1.00, Loss = 0.000
[2024-06-27 22:21] Train Step 9000/10000, Batch Size = 128, Examples/Sec =
8015.39, Accuracy = 1.00, Loss = 0.001
[2024-06-27 22:22] Train Step 10000/10000, Batch Size = 128, Examples/Sec =
8404.63, Accuracy = 1.00, Loss = 0.000
Done training.
```

[ ]:
```
torch.save(model5_lstm.state_dict(), 'model5_LSTM.pth')
torch.save(model6_lstm.state_dict(), 'model6_LSTM.pth')
torch.save(model7_lstm.state_dict(), 'model7_LSTM.pth')
torch.save(model8_lstm.state_dict(), 'model8_LSTM.pth')
torch.save(model9_lstm.state_dict(), 'model9_LSTM.pth')
torch.save(model10_lstm.state_dict(), 'model10_LSTM.pth')
```

[ ]:
```
accuracies5_lstm_avg = average_accuracies(accuracies5_lstm, config5_lstm.
 ↪train_steps)
accuracies6_lstm_avg = average_accuracies(accuracies6_lstm, config6_lstm.
 ↪train_steps)
accuracies7_lstm_avg = average_accuracies(accuracies7_lstm, config7_lstm.
 ↪train_steps)
accuracies8_lstm_avg = average_accuracies(accuracies8_lstm, config8_lstm.
 ↪train_steps)
accuracies9_lstm_avg = average_accuracies(accuracies9_lstm, config9_lstm.
 ↪train_steps)
accuracies10_lstm_avg = average_accuracies(accuracies10_lstm, config10_lstm.
 ↪train_steps)

plt.plot(accuracies5_lstm_avg, label="T = 5")
plt.plot(accuracies6_lstm_avg, label="T = 6")
plt.plot(accuracies7_lstm_avg, label="T = 7")
plt.plot(accuracies8_lstm_avg, label="T = 8")
plt.plot(accuracies9_lstm_avg, label="T = 9")
plt.plot(accuracies10_lstm_avg, label="T = 10")
plt.xlabel(f"Steps x 1000")
plt.ylabel("Accuracy")
plt.title("Accuracy of LSTM for different Palindrome lengths")
plt.legend()
plt.show()

# Plot accuracy x palindrome length plot
palindrome_lstm_accuracies = [
  np.mean(accuracies5_lstm)*100,
  np.mean(accuracies6_lstm)*100,
  np.mean(accuracies7_lstm)*100,
  np.mean(accuracies8_lstm)*100,
  np.mean(accuracies9_lstm)*100,
```

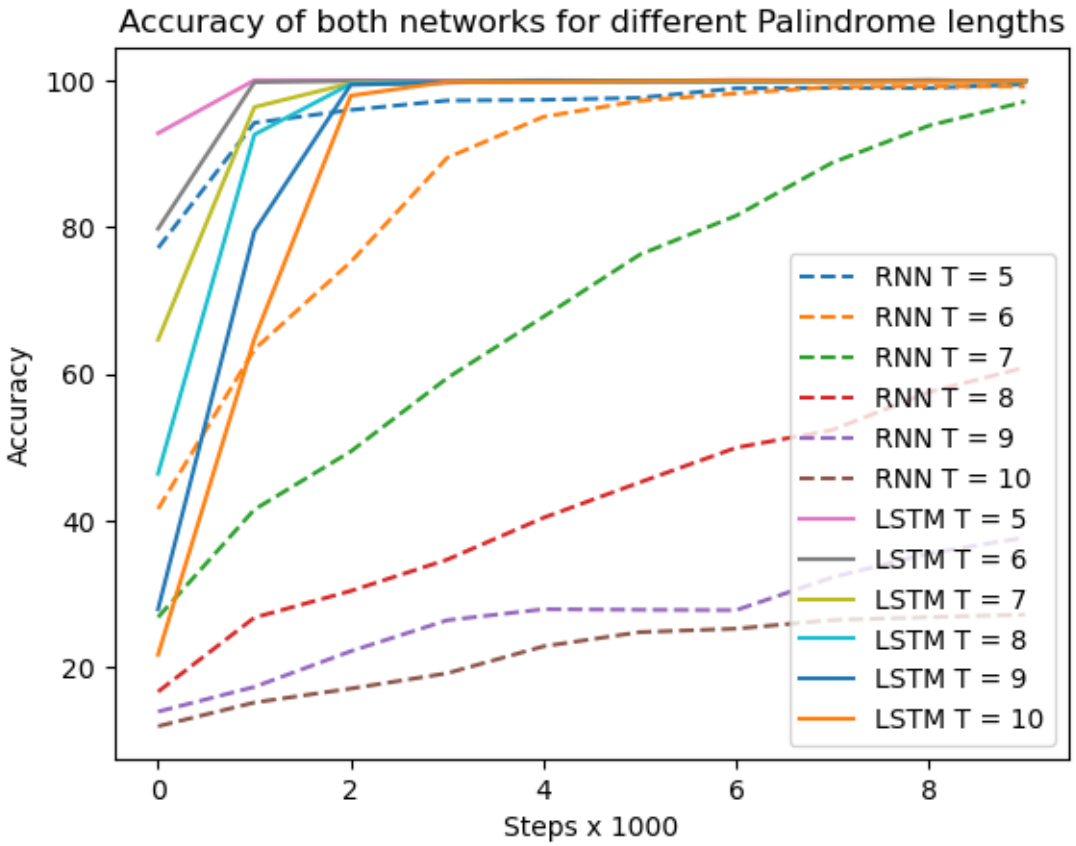
```
  np.mean(accuracies10_lstm)*100
  ]

plt.plot(range(5,11), palindrome_lstm_accuracies, "-ro")
plt.xlabel("Palindrome length")
plt.ylabel("Accuracy")
plt.title("Average LSTM Accuracy for different Palindrome lengths")
plt.show()
```

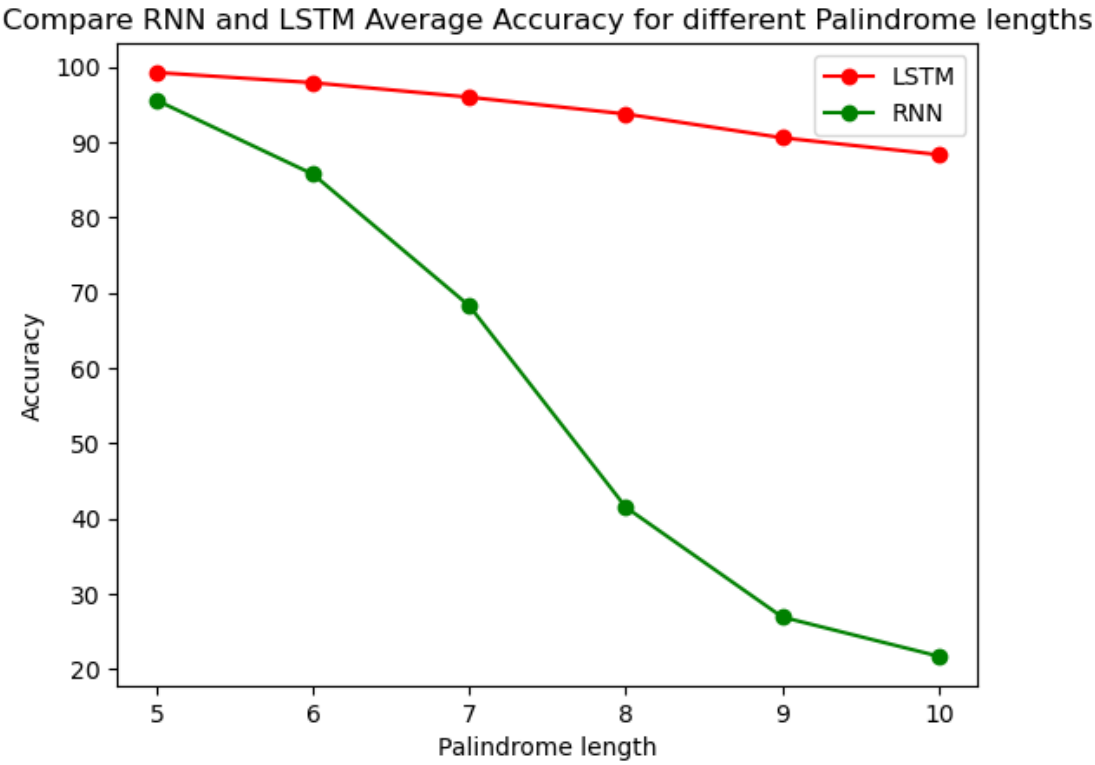




[ ]:
```
plt.plot(accuracies5_avg, '--', label="RNN T = 5")
plt.plot(accuracies6_avg, '--', label="RNN T = 6")
plt.plot(accuracies7_avg, '--', clip_on=True, label="RNN T = 7")
plt.plot(accuracies8_avg, '--', label="RNN T = 8")
plt.plot(accuracies9_avg, '--', label="RNN T = 9")
plt.plot(accuracies10_avg, '--', label="RNN T = 10")
plt.plot(accuracies5_lstm_avg, label="LSTM T = 5")
plt.plot(accuracies6_lstm_avg, label="LSTM T = 6")
plt.plot(accuracies7_lstm_avg, label="LSTM T = 7")
plt.plot(accuracies8_lstm_avg, label="LSTM T = 8")
plt.plot(accuracies9_lstm_avg, label="LSTM T = 9")
plt.plot(accuracies10_lstm_avg, label="LSTM T = 10")
plt.xlabel(f"Steps x 1000")
plt.ylabel("Accuracy")
plt.title("Accuracy of both networks for different Palindrome lengths")
plt.legend()
plt.show()

plt.plot(range(5,11), palindrome_lstm_accuracies, "-ro", label="LSTM")
plt.plot(range(5,11), palindrome_accuracies, "-go", label="RNN")
```

```
plt.xlabel("Palindrome length")
plt.ylabel("Accuracy")
plt.title("Compare RNN and LSTM Average Accuracy for different Palindrome␣
  ↪lengths")
plt.legend()
plt.show()
```





1.0.3 Comparison RNN LSTM

- RNN trains way faster than LSTM, since it has way less parameters
- RNN performs around the same good as LSTM for Palindrome lengths of T=5, but then starts to fall off while the LSTM continues to perform well.

Reasons

- The RNN fails to store the information in the hidden state for increasing palindrome length due to vanishing gradient.
- Thanks to the cell state, the LSTM can pass information from early layers very easily to late neurons. Since for this test, the first digit is crucial for the last digit, the LSTM will perform especially good here.