






COMS 4776: Neural Networks and Deep Learning

Tutorial: RNNs

Amogh Inamdar

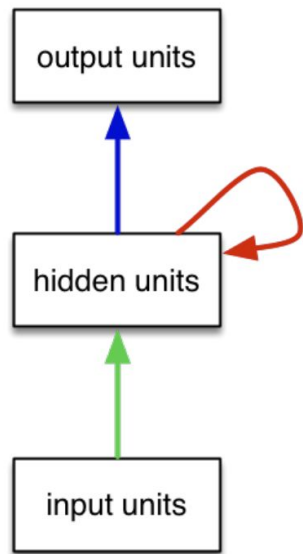
ai2442@columbia.edu

Recap: Why do we care?

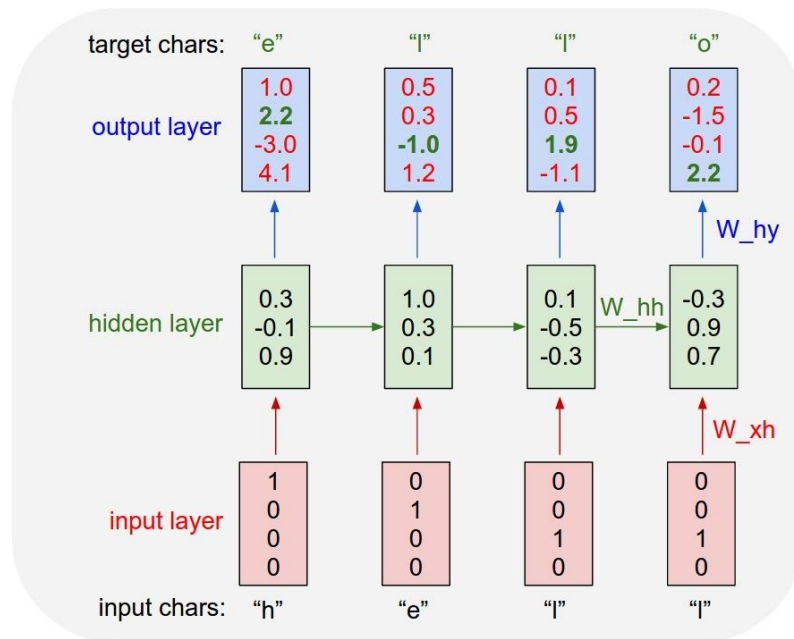
Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation		→	
Sentiment classification	"There is nothing to like in this movie."	→	
DNA sequence analysis	AGCCCCTGTGAGGAAGTAG	→	AGCCCCTGTGAGGAAGTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger .

Auto-Regression: future states can be predicted from the existing trajectory of a sequence.

Recap: Recurrent Neural Networks



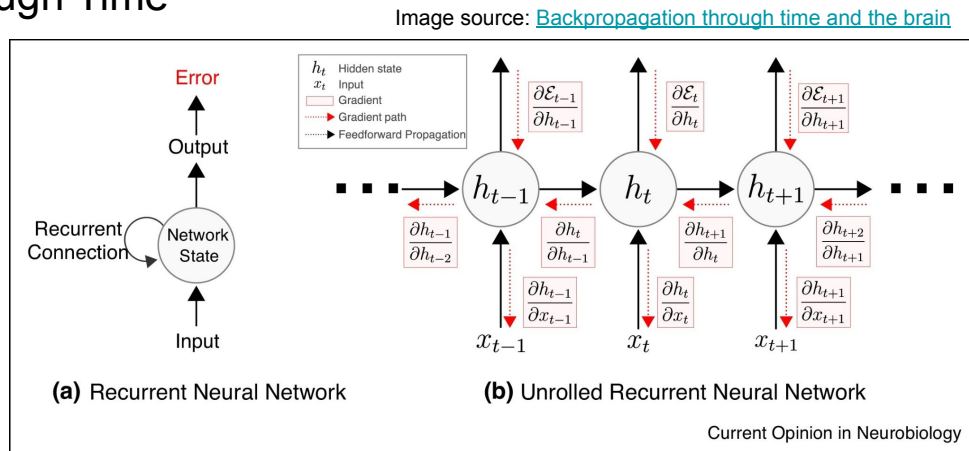
Folded



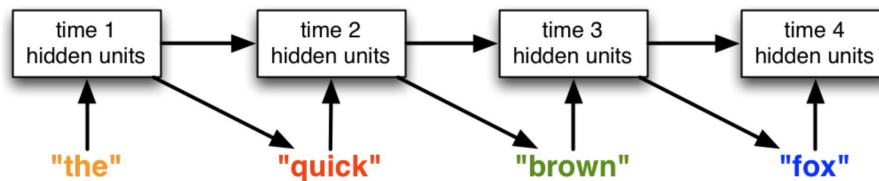
Unfolded

Recap: RNN Training

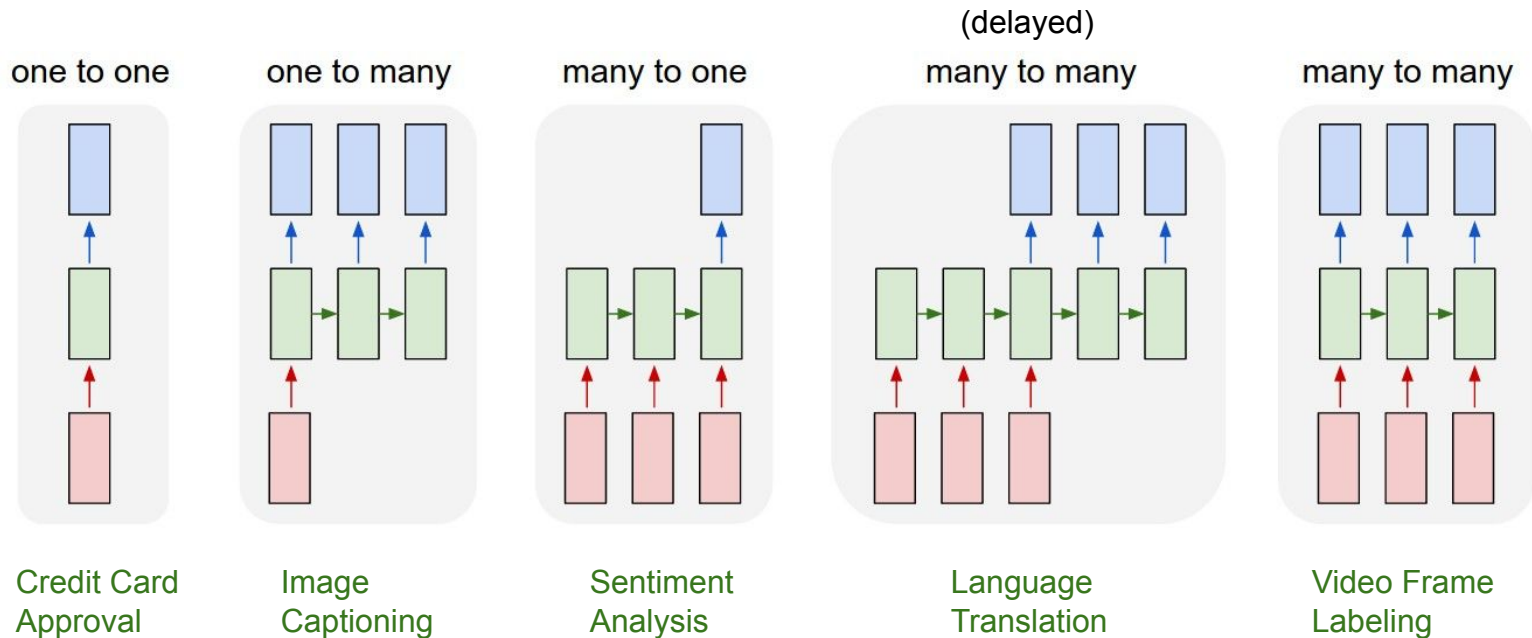
Backpropagation Through Time



Teacher Forcing



Input-Output Architectures



Limitations of Vanilla RNNs

- Vanishing and exploding gradients
 - Can be solved by clipping, but this leads to biased updates.
 - Another approach: limit sequence lengths (e.g., a sliding window). Drawbacks?
- All dependencies (that matter) are assumed to be from left-to-right (causal)
 - Is this always true? Can future inputs re-contextualize past inputs?
- Simple model of sequence memory
 - Older inputs are harder to memorize regardless of importance.
 - No 'memory consolidation' or focus on multiple scopes of past context.

Extensions: Bidirectional RNNs

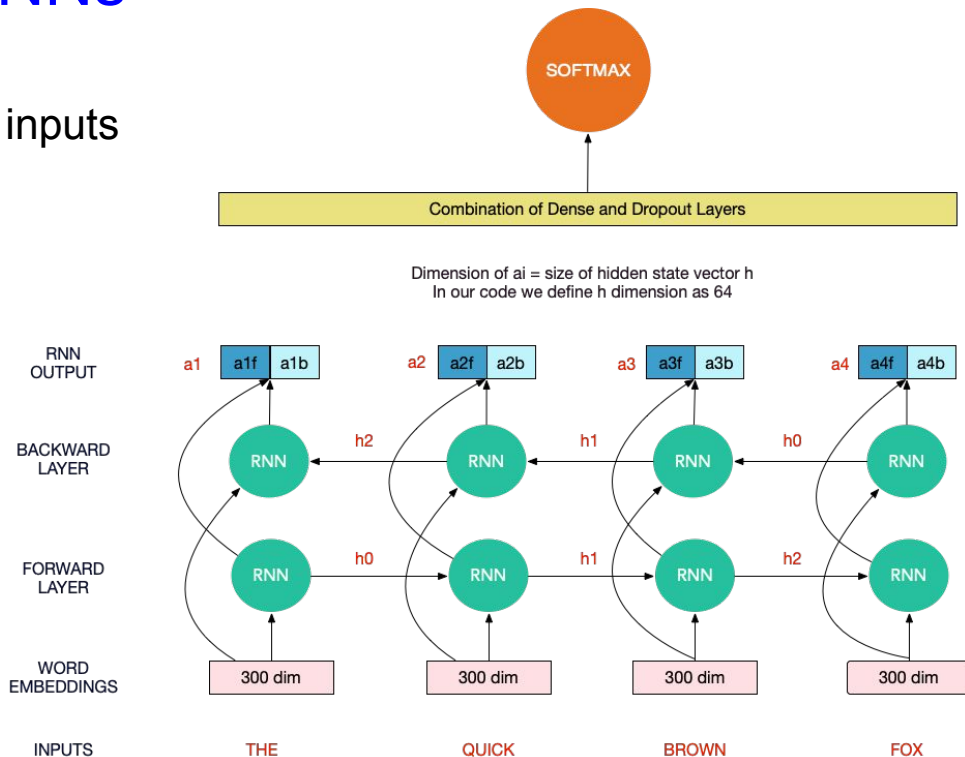
Learning how a past input relates to future inputs could lead to better representations.

When looking backwards can help

- Bad handwriting: is that a 7 or a T?
- Contextual semantics in language:
 - *He said, "Teddy bears are on sale!"*
 - *He said, "Teddy Roosevelt was a great President."*

Drawbacks?

- Need to see the full sequence up front!
- More compute: 2x forward/backward passes.



Extensions: Deep RNNs

Stack hidden layers between inputs and outputs: this is NN~~DL~~ after all!

Benefits

- Standard deep learning idea.
- Intuitively: shallower layers learn local associations; deeper layers for long-term structure.

Drawbacks

- Mo' params, mo' (compute and complexity) problems.
- Tough to train: hyperparameter search, gradient dynamics, etc.

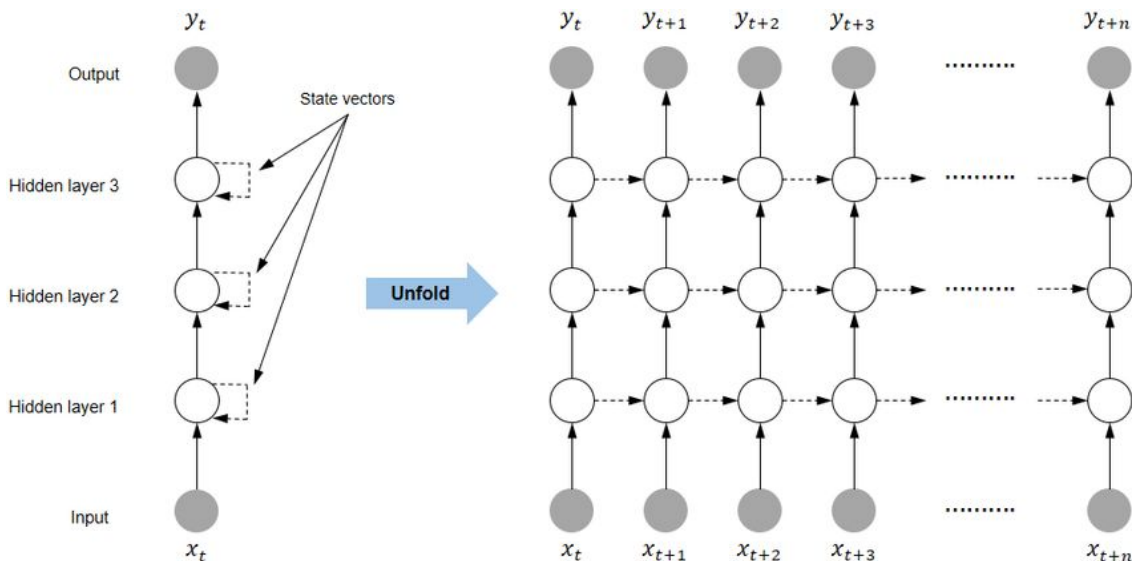


Image source: [Moradi & Samwald: ResearchGate](#)

LSTM Networks

Image source: [Bidirectional LSTM - GeeksforGeeks](#)

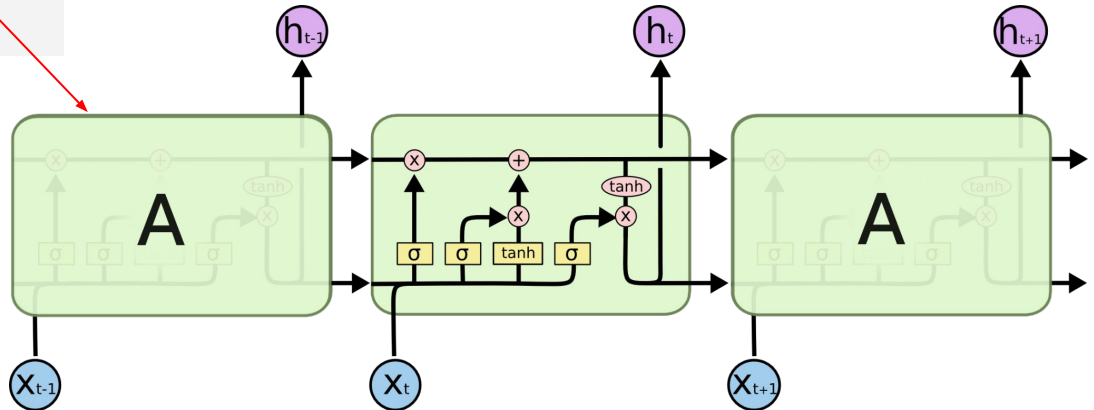
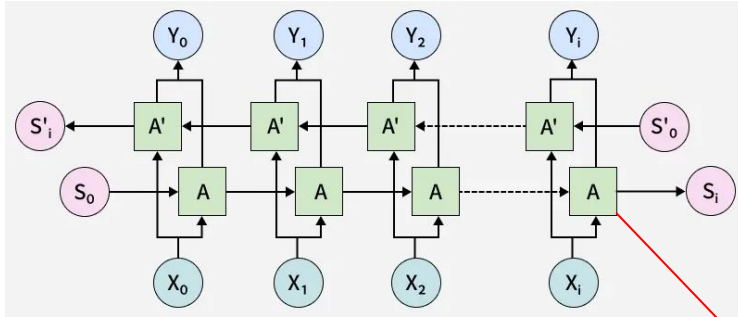
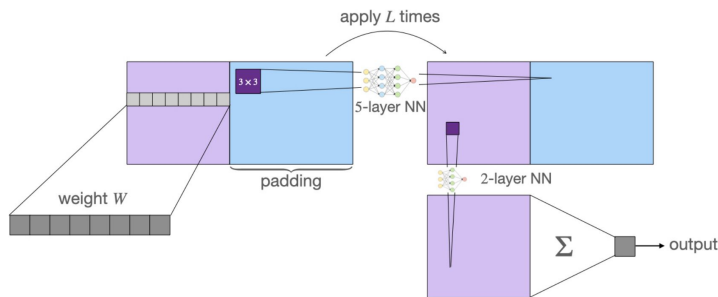


Image source: [LSTM Networks -- colah's blog](#)

Recurrence in Other Architectures

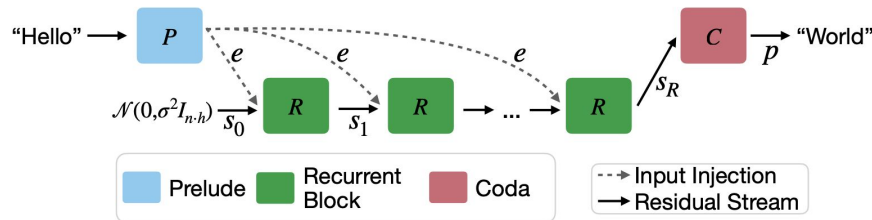
A variety of neural network architectures can be made recurrent for sequential learning.

[Recurrent Convolutional Neural Networks Learn Succinct Learning Algorithms](#)



Recurrent CNNs for algorithm learning

[Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach](#)



Recurrent transformers for math reasoning

Code Demo

Tweet Sentiment Classification with an RNN

An RNN in PyTorch

```
class TweetRNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(TweetRNN, self).__init__()
        self.emb = nn.Embedding.from_pretrained(glove.vectors)
        self.hidden_size = hidden_size
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        # Look up the embedding
        x = self.emb(x)
        # Forward propagate the RNN
        out, _ = self.rnn(x)
        # Pass the output of the last time step to the classifier
        out = self.fc(out[:, -1, :])
        return out

model = TweetRNN(50, 50, 2)
```

Imports

```
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from torch.utils.data import Dataset, DataLoader
```

Data Preprocessing

```
df = pd.read_csv(url, delimiter='\\t', header=None, names=['label', 'text'])

def preprocess_text(text):
    return text.lower().split()

df['text'] = df['text'].apply(preprocess_text)
df = df[['text', 'label']]

le = LabelEncoder()
df['label'] = le.fit_transform(df['label'])

train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)
```

Why?

Data Preprocessing

```
vocab = set([word for phrase in df['text'] for word in phrase])  
word_to_idx = {word: idx for idx, word in enumerate(vocab, 1)}
```

```
def encode_phrase(phrase):  
    return [word_to_idx[word] for word in phrase]
```

Why?

```
train_data['text'] = train_data['text'].apply(encode_phrase)  
test_data['text'] = test_data['text'].apply(encode_phrase)
```

```
max_length = max(df['text'].apply(len))
```

```
def pad_sequence(seq, max_length):  
    return seq + [0] * (max_length - len(seq))
```

Why?

```
train_data['text'] = train_data['text'].apply(lambda x: pad_sequence(x, max_length))  
test_data['text'] = test_data['text'].apply(lambda x: pad_sequence(x, max_length))
```

Dataset Construction

```
class SentimentDataset(Dataset):
    def __init__(self, data):
        self.texts = data['text'].values
        self.labels = data['label'].values

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
        return torch.tensor(text, dtype=torch.long), torch.tensor(label, dtype=torch.long)

train_dataset = SentimentDataset(train_data)
test_dataset = SentimentDataset(test_data)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```


Define the RNN

```
class SentimentRNN(nn.Module):
    def __init__(self, vocab_size, embed_size, hidden_size, output_size):
        super(SentimentRNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.rnn = nn.RNN(embed_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.embedding(x)
        h0 = torch.zeros(1, x.size(0), hidden_size).to(x.device)
        out, _ = self.rnn(x, h0)
        out = self.fc(out[:, -1, :])
        return out
```

The PyTorch RNN module also returns the final hidden state. When is this useful?

```
vocab_size = len(vocab) + 1
embed_size = 128
hidden_size = 128
output_size = 2
model = SentimentRNN(vocab_size, embed_size, hidden_size, output_size)
```

Train the Network

```
criterion = nn.CrossEntropyLoss()  
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
num_epochs = 10  
for epoch in range(num_epochs):  
    model.train()  
    epoch_loss = 0  
    for texts, labels in train_loader:  
        outputs = model(texts)  
        loss = criterion(outputs, labels)  
  
        optimizer.zero_grad()  
        loss.backward()  
        optimizer.step()  
  
    epoch_loss += loss.item()
```

```
print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss / len(train_loader):.4f}')
```

It Works!

```
Epoch [1/10], Loss: 0.4059  
Epoch [2/10], Loss: 0.4014  
Epoch [3/10], Loss: 0.3965  
Epoch [4/10], Loss: 0.3950  
Epoch [5/10], Loss: 0.3950  
Epoch [6/10], Loss: 0.3969  
Epoch [7/10], Loss: 0.3959  
Epoch [8/10], Loss: 0.3972  
Epoch [9/10], Loss: 0.3975  
Epoch [10/10], Loss: 0.3964
```

Evaluate the Network

```
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for texts, labels in test_loader:
        outputs = model(texts)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Accuracy: {accuracy:.2f}%')
```

Accuracy: 86.64%

Hurray!!!

```
# Example usage
text = "Hated the movie!" # Replace with the text you want to classify
vocab = {} # Provide your vocabulary mapping (word -> index)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

predicted_label = predict_sentiment(text, model, vocab, device)
print(f"Predicted Label: {predicted_label}")
```

Predicted Label: 0

Thank You