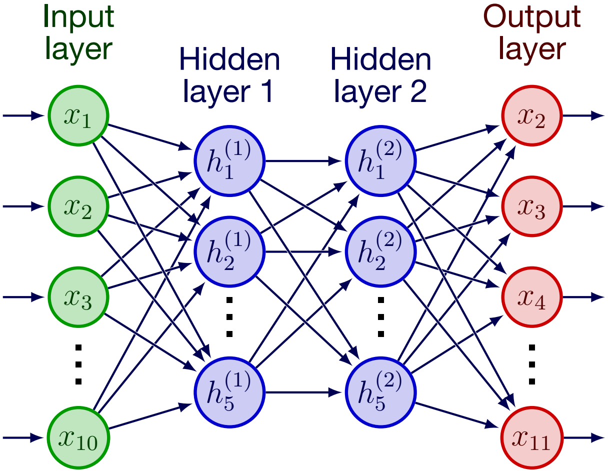
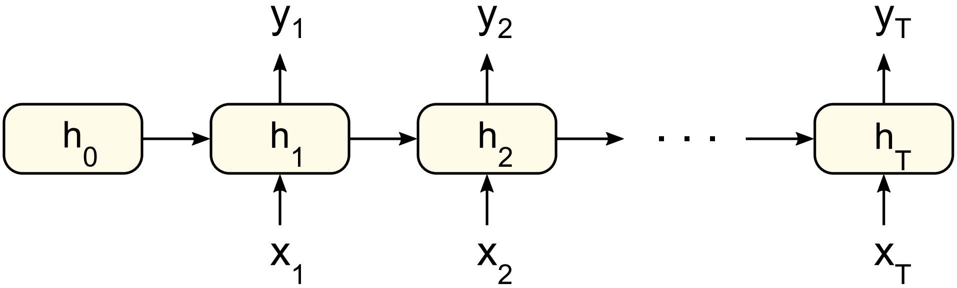
Recurrent Neural Network

One-step forecast



Problem: Past values depend on future values (e.g. x_2 depends on x_{10})

Recurrent Neural Network

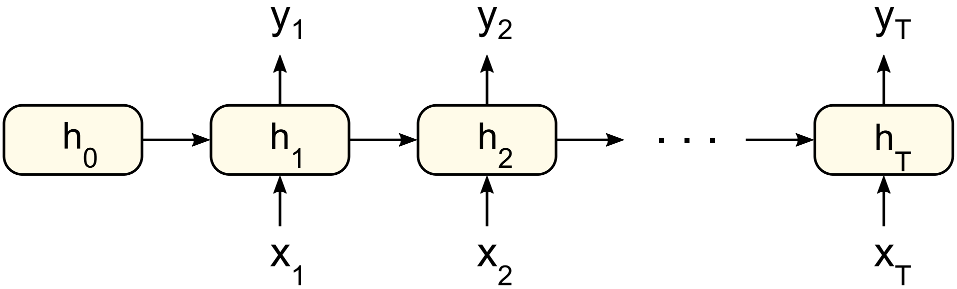


Inputs: $x_1, x_2, \ldots, x_T$

Hidden states: $h_1, h_2, \ldots, h_T$

Outputs: $y_1, y_2, \ldots, y_T$

Recurrent Neural Network



$$h_t = \tanh(W_{xh} x_t + W_{hh} h_{t-1})$$
$$y_t = W_{hy} h_t.$$

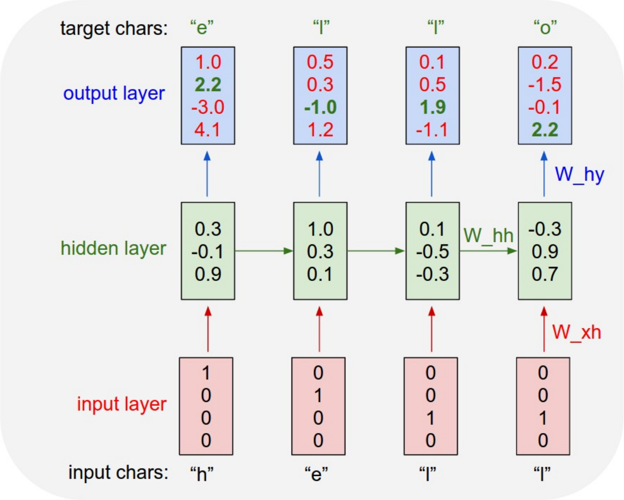
RNN Initialization in PyTorch

an RNN with 10 input features, 20 hidden state dimensions, 2 layers, and batch-first ordering

```
rnn_layer = nn.RNN(input_size=10, hidden_size=20,
                   num_layers=2, batch_first=True)
```

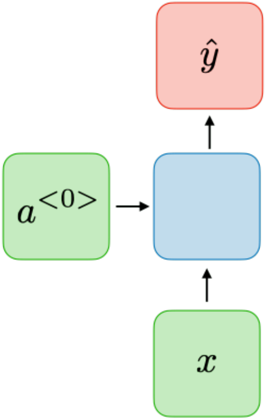
Character-level language model

- Goal: predicting the next character.
- Use the next characters as the target.



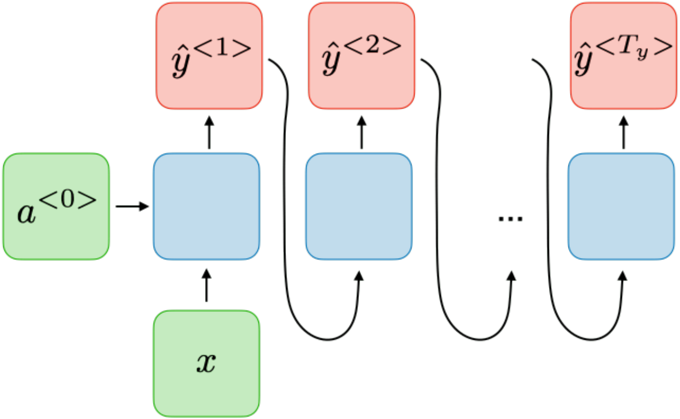
Types of RNN

One-to-one ($T_x = T_y = 1$)



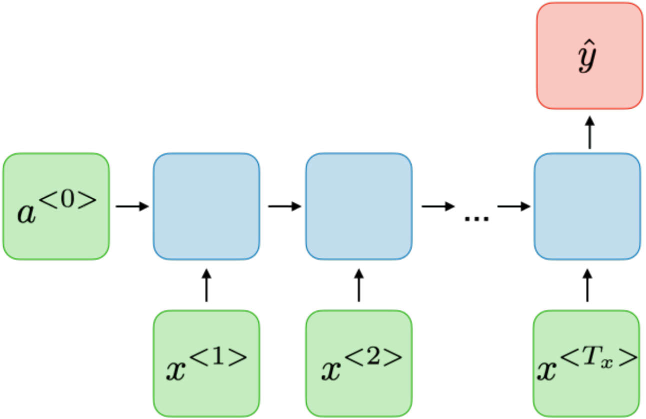
Types of RNN

One-to-many $(T_x = 1, T_y > 1)$



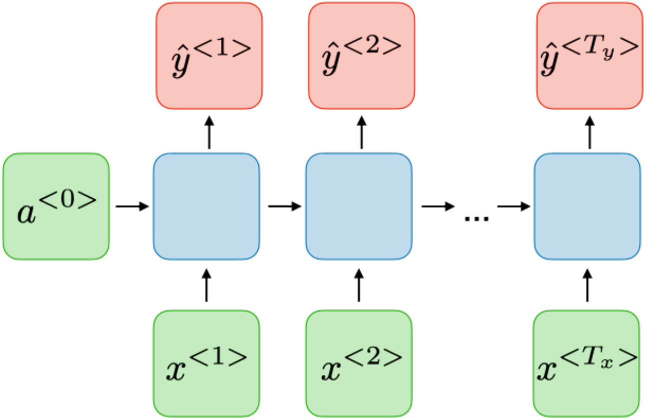
Types of RNN

Many-to-one $(T_x > 1, T_y = 1)$



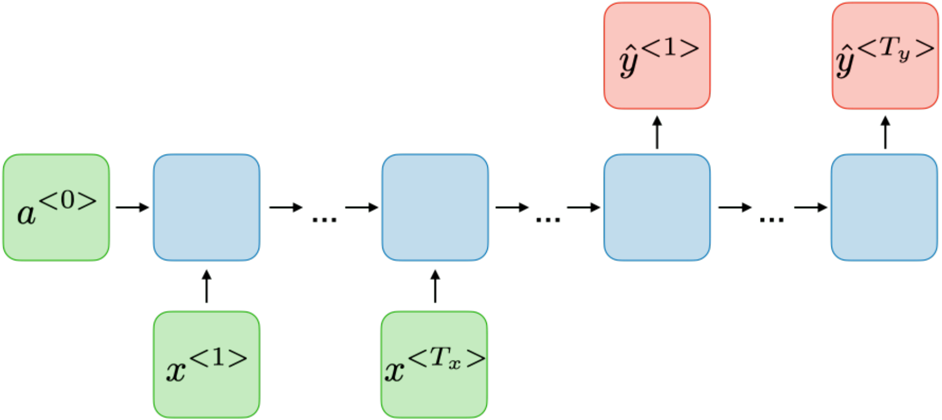
Types of RNN

Many-to-many ($T_x = T_y$)

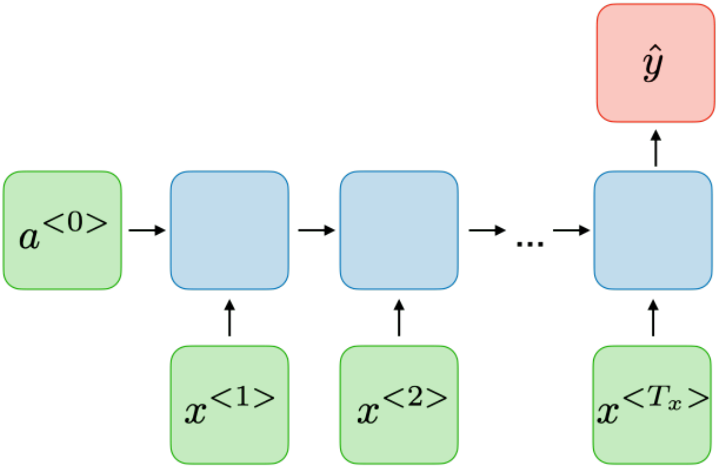


Types of RNN

Many-to-many $(T_x \neq T_y)$

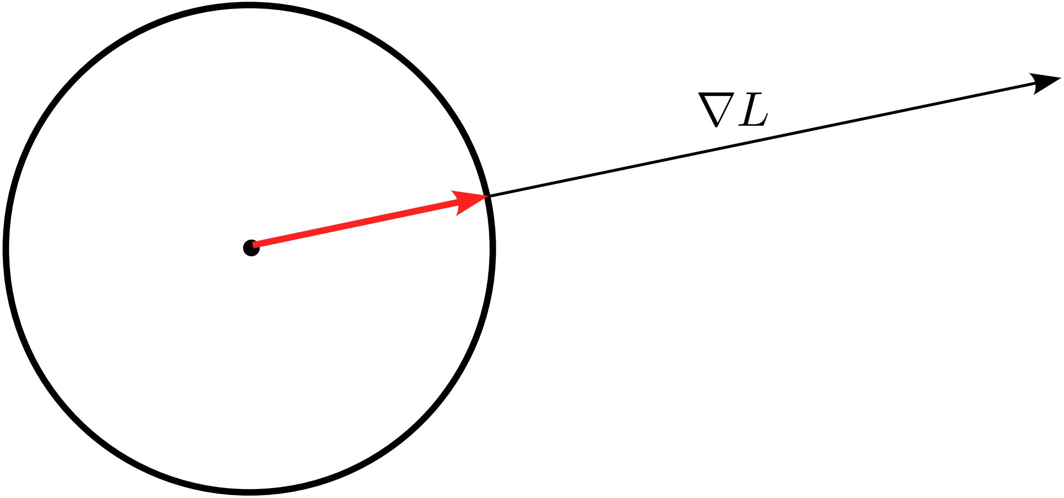


Recurrent Neural Network



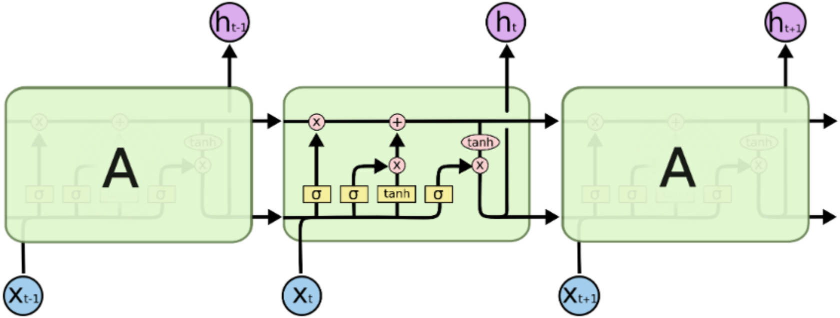
However, there’s gradient vanishing/exploding problem.

Gradient clipping

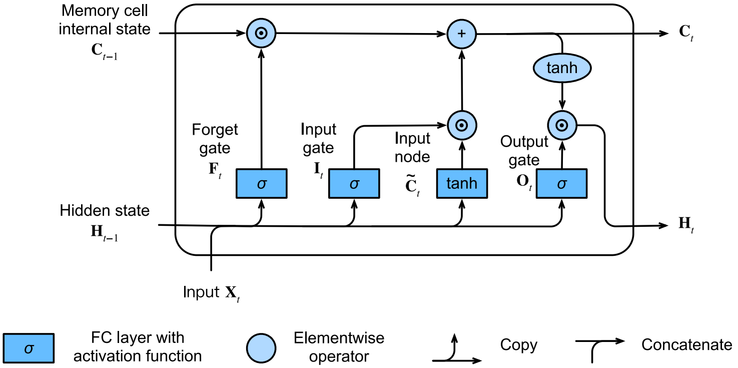


solves gradient exploding

Long-short term memory (LSTM)



Long-short term memory (LSTM)

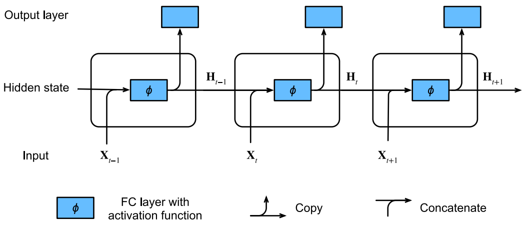

Memory cell internal state $\mathbf{C}_{t-1}$
Hidden state $\mathbf{H}_{t-1}$
Input $\mathbf{X}_t$
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Input node $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
σ
tanh
σ
tanh
+
$\mathbf{C}_t$
$\mathbf{H}_t$
σ
FC layer with activation function
Elementwise operator
Copy
Concatenate

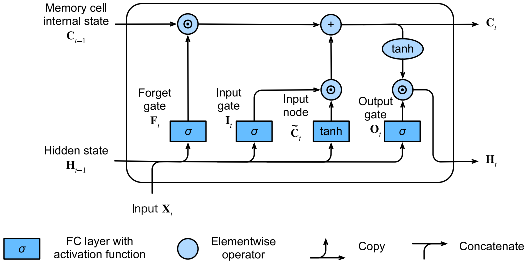
- **Hidden state** h_t and **Cell state** c_t
- h_t is also the output.

LSTM Layer in PyTorch

```
lstm_layer = nn.LSTM(input_size=10, hidden_size=20,
                     num_layers=2, batch_first=True)
```

RNN vs LSTM


Output layer
Hidden state
Input
ϕ
$\mathbf{H}_{t-1}$
$\mathbf{H}_{t}$
$\mathbf{H}_{t+1}$
$\mathbf{X}_{t-1}$
$\mathbf{X}_{t}$
$\mathbf{X}_{t+1}$
FC layer with activation function
Copy
Concatenate


Memory cell internal state $\mathbf{C}_{t-1}$
$\mathbf{C}_{t}$
tanh
Forget gate $\mathbf{F}_{t}$
Input gate $\mathbf{I}_{t}$
Input node $\tilde{\mathbf{C}}_{t}$
Output gate $\mathbf{O}_{t}$
σ
tanh
+
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_{t}$
Input $\mathbf{X}_{t}$
FC layer with activation function
Elementwise operator
Copy
Concatenate

Text generation with LSTM

Example: Generate a paragraph with 400 characters from an input of 12 characters

Data: Text corpus from Wikipedia, preprocessed into 12 initial characters + 388 next characters

Prediction: Predict the paragraph from the initial characters: “The quick br”

Embedding Layer in PyTorch

Transform from 1000-dimensional one-hot-encoding input into a 50-dimensional vector.

```
embedding_layer = nn.Embedding(num_embeddings=1000,
                               embedding_dim=50)
```

Simple LSTM Model for Text Generation

```python
class SimpleLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, num_layers):
        super(SimpleLSTM, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x, hidden):
        x = self.embedding(x)
        out, hidden = self.lstm(x, hidden)
        out = self.fc(out)
        return out, hidden
```

Text generation with LSTM

100 iterations

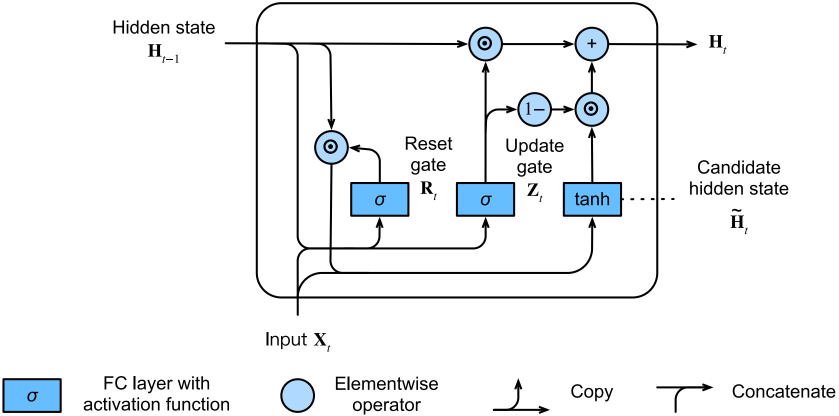
The quick br ypqznwrt lmji vbjfr lswmpz jqir nkfld awzmr cxpk vnz jqtr awvn lsj...

Text generation

2000 iterations

The quick brown fox jumps the lazy over the moon bright stars with sky running but slowly path field...

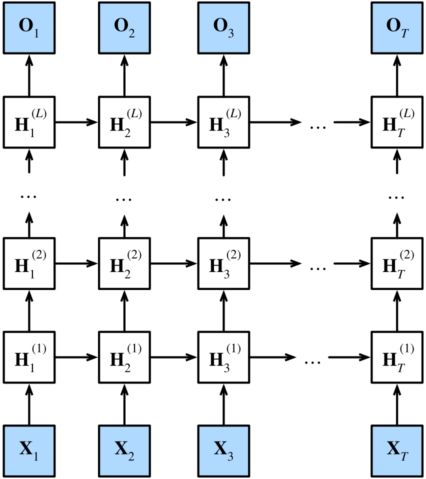
Gated Recurrent Unit (GRU)



GRU Layer in PyTorch

```
gru_layer = nn.GRU(input_size=10, hidden_size=20,
                   num_layers=2, batch_first=True)
```

Deep RNN



$\mathbf{O}_1$
$\mathbf{O}_2$
$\mathbf{O}_3$
$\mathbf{O}_T$
$\mathbf{H}_1^{(L)}$
$\mathbf{H}_2^{(L)}$
$\mathbf{H}_3^{(L)}$
…
$\mathbf{H}_T^{(L)}$
…
…
…
…
$\mathbf{H}_1^{(2)}$
$\mathbf{H}_2^{(2)}$
$\mathbf{H}_3^{(2)}$
…
$\mathbf{H}_T^{(2)}$
$\mathbf{H}_1^{(1)}$
$\mathbf{H}_2^{(1)}$
$\mathbf{H}_3^{(1)}$
…
$\mathbf{H}_T^{(1)}$
$\mathbf{X}_1$
$\mathbf{X}_2$
$\mathbf{X}_3$
$\mathbf{X}_T$

Filling in the blank

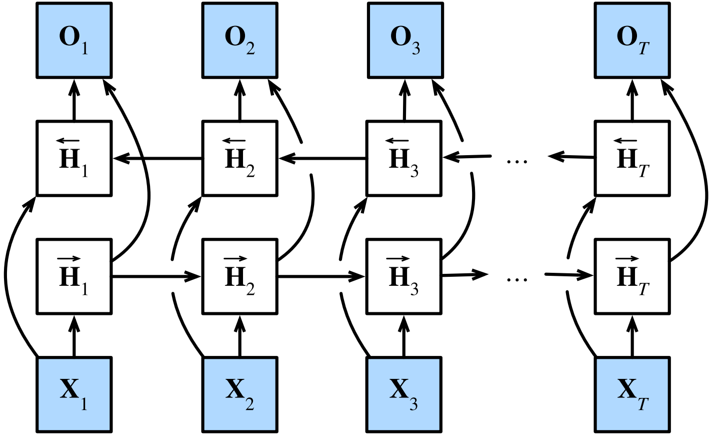
I am _____.

I am _____ hungry at all.

I am _____ hungry, and I can eat a horse.

The missing word heavily depends on the words that come after

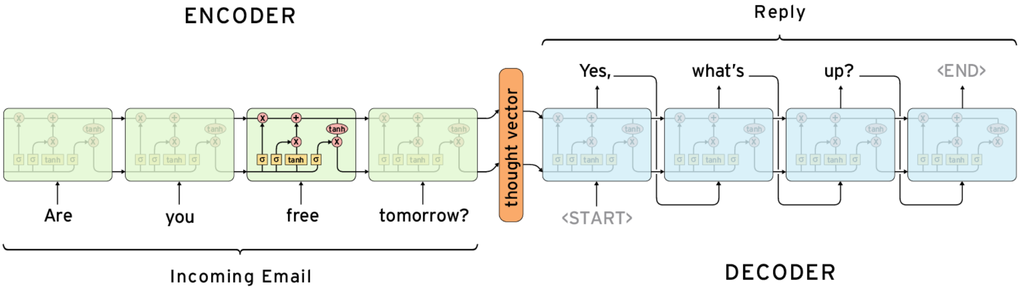
Bidirectional RNN


$\mathbf{O}_1$
$\mathbf{O}_2$
$\mathbf{O}_3$
$\mathbf{O}_T$
$\overleftarrow{\mathbf{H}}_1$
$\overleftarrow{\mathbf{H}}_2$
$\overleftarrow{\mathbf{H}}_3$
…
$\overleftarrow{\mathbf{H}}_T$
$\overrightarrow{\mathbf{H}}_1$
$\overrightarrow{\mathbf{H}}_2$
$\overrightarrow{\mathbf{H}}_3$
…
$\overrightarrow{\mathbf{H}}_T$
$\mathbf{X}_1$
$\mathbf{X}_2$
$\mathbf{X}_3$
$\mathbf{X}_T$

Bidirectional RNN Layer in PyTorch

```
rnn_layer = nn.RNN(input_size=10, hidden_size=20,
                   num_layers=2, bidirectional=True, ba
```

Encoder-Decoder Seq2Seq



Sentence padding

Suppose RNN encoder has 7 hidden units, RNN decoder has 6 hidden units

How are we going to split the following sentences?

$$\begin{aligned} \texttt{input} &= \texttt{'hello, how are you'} \\ \texttt{output} &= \texttt{'i am fine'} \end{aligned}$$

Sentence padding

Suppose RNN encoder has 7 hidden units, RNN decoder has 6 hidden units

How are we going to split the following sentences?

```
 input = 'hello, how are you'
output = 'i am fine'
```

```
encoder_input = ['hello','how','are','you','<EOS>','<PAD>','<PAD>']
decoder_input = ['<START>','i','am','fine','<EOS>','<PAD>']
       output = ['i','am','fine','<EOS>','<PAD>','<PAD>']
```