AIE1902: Exploring Language Models with AI

Xiaopeng Li
CUHK(SZ)
SAI

February 2, 2026

Outline

From Word2Vec to Neural Language Models

Recurrent Neural Network (RNN)

Long Short-Term Memory (LSTM) (Bonus)

Decoding

Recap: Word2Vec

Goal: construct plausible sentences, sequences with $p(y_1, \ldots, y_T)$ high

CBOW: use neighboring words to predict the center word

Skip-Gram: use the center word to predict neighboring words

They are essentially using the most special neural network!

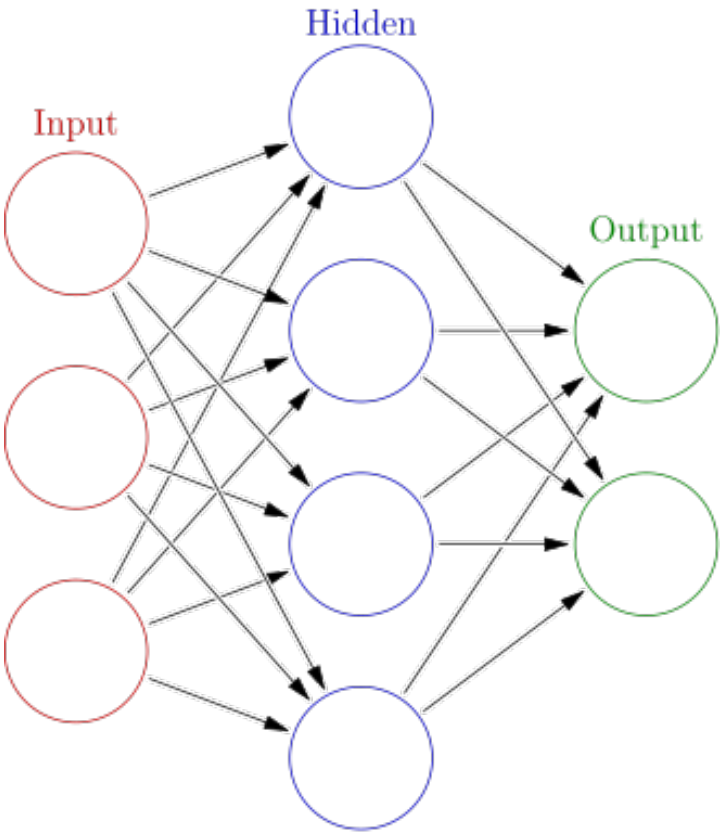
$$x \mapsto \text{softmax}(BA^\top x)$$

CBOW and Skip-Gram uses the same network but different data:

- CBOW, $x = \frac{1}{2m} \sum_{i=-m, i\neq 0}^{m} e_{w_{t+i}}$ and $y = e_{w_t}$
- Skip-Gram, $x = e_{w_t}$ and $y = e_{w_{t+i}}$ for $i = -m, \ldots, m$, $i \neq 0$.

Basics of Neural Network

Multilayer Perceptron (MLP):



Input layer: $x \in \mathbb{R}^{|\mathcal{V}|}$;

Hidden layer: $h = A^\top x \in \mathbb{R}^d$;

Output layer: $s = Bh \in \mathbb{R}^{|\mathcal{V}|}$;

Softmax: at the end of the output layer, we usually apply softmax to obtain probability.

Arrow: arrow from input i to hidden j means $\times A_{ij}$.

Computation Inside Neuron

Neuron: a circle in the picture represents one scalar component.

- each neuron in the input layer means x_j
- each neuron in the hidden layer computes: summation, bias, activation,

$$h_i = \sigma\left(\sum_j w_{ij}x_j + b_i\right).$$

Activation: component-wise operation, add non-linearity,

- linear activation $\sigma(x) = x$, simplest;
- sigmoid activation $\sigma(x) = \frac{1}{1+e^{-x}}$, nonlinear;
- ReLU activation $\sigma(x) = \max\{x, 0\}$;
- ...

Softmax Layer and Loss

Softmax: from scores to probabilities

- Network output before softmax is a score (or logit) vector
- Softmax converts to a probability distribution while keeping order:

$$p_k = \frac{e^{s_k}}{\sum_{j=1}^{|\mathcal{V}|} e^{s_j}}, \quad \sum_{k=1}^{|\mathcal{V}|} p_k = 1.$$

Loss: cross-entropy for word prediction

- True word y is one-hot; predicted distribution is p.
- Cross-entropy loss:

$$\mathcal{L}(p, y) = -\sum_{k=1}^{|\mathcal{V}|} y_k \log p_k = -\log p_{\text{true}}.$$

- Larger loss if model gives low probability to the correct word.

Training loop

Parameters: $\theta = \{A, B\}$ (network weights);

Iterative Training Process:

- Forward (propagation). $h = A^\top x$, $s = Bh$, $p = \text{softmax}(s)$.
- Loss. $\mathcal{L} = -\log p_{\text{true}}$.
- Backward (propagation). Compute gradients $\nabla_\theta \mathcal{L}$.
- Update. $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$.

Why Backpropagation? It is an efficient way to compute gradients of functions with layered structure. Details of backpropagation is not required for this course.

Limitation of Word2Vec

Word2Vec can predict a word using a center or window.

But language modeling goal is:

$$p(w_{1:T}) = \prod_{t=1}^{T} p(w_t | w_{1:t-1}).$$

Discrepancy:

- Word2Vec $\approx$ learns good word vectors;
- LM $\approx$ assigns probability to whole sentences.

We need a model that processes words in order and keeps information from the past!

Fixed-Window Context is Not Enough

Word Order: the set of words matters more than the order, e.g.,

$$\texttt{man bites dog} \quad \neq \quad \texttt{dog bites man}$$

CBOW can produce similar $x = \frac{1}{2m} \sum e_{w_{t+i}}$.

Long-range dependencies: Some predictions require words far away

$$\texttt{The book that you gave me yesterday } \underline{\texttt{is}} \ldots$$

To predict is vs are, we must remember the subject **book**.

Conclusion: We need a model that is sensitive to sequence order with a memory that carries information across time steps:

- Recurrent Neural Network (RNN), or
- Long Short-Term Memory (LSTM)

Outline

More Data, More Efficient

One way get training data is to use chunks of contiguous text $x = (x_1, x_2, \ldots, x_T)$ and predict the shifted version $y = (x_2, x_3, \ldots, x_{T+1})$.

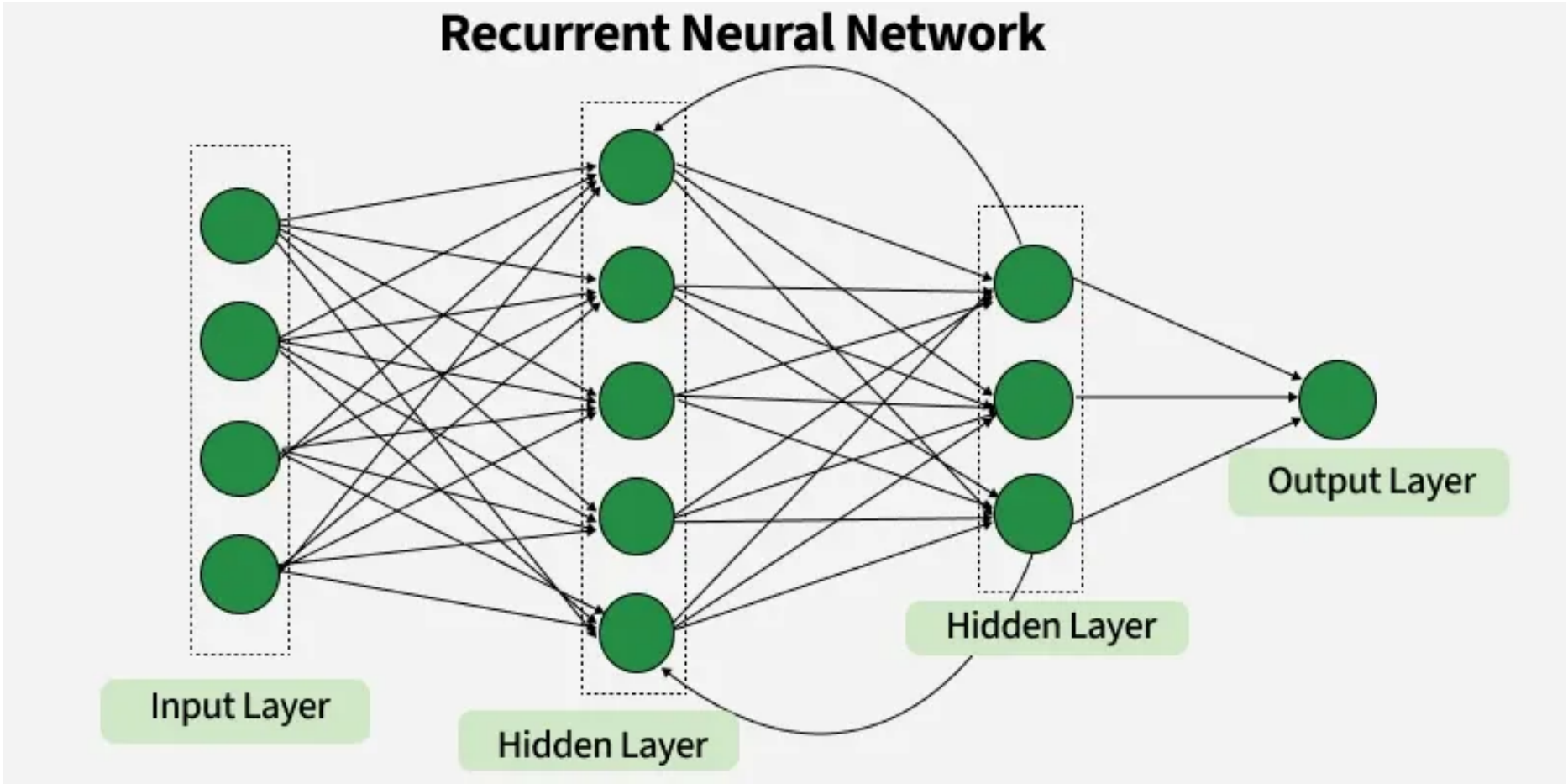
Example: `START The man walks END`

- $x = (\texttt{START}, \texttt{"The"}, \texttt{"man"}, \texttt{"walks"})$,
- $y = (\texttt{"The"}, \texttt{"man"}, \texttt{"walks"}, \texttt{END})$.

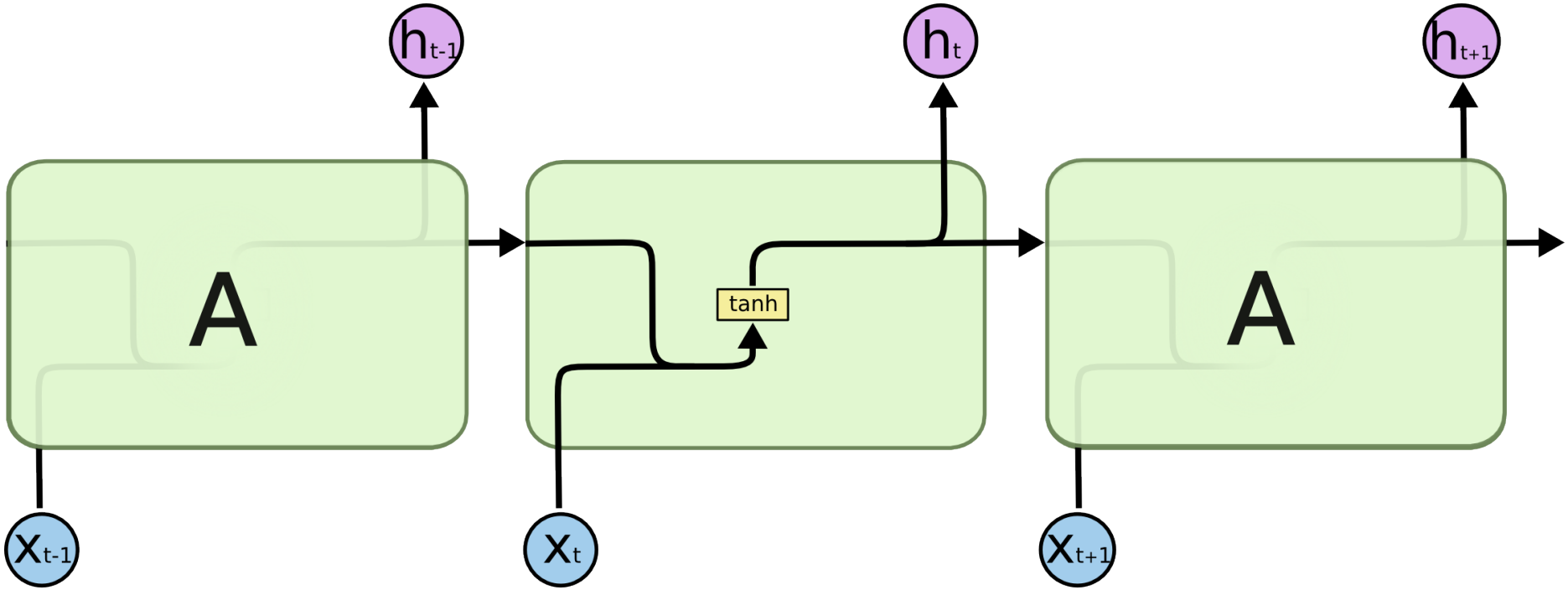
Goal: our new model is trained to do

- predict `"The"` from $(\texttt{START})$;
- predict `"man"` from $(\texttt{START}, \texttt{"The"})$;
- predict `"walks"` from $(\texttt{START}, \texttt{"The"}, \texttt{"man"})$.
- ...

RNN


Recurrent Neural Network
Input Layer
Hidden Layer
Hidden Layer
Output Layer

Unrolled Recurrent Block



Notice:

- This picture represents only one layer.
- Each green block is doing identical computation.
- Each block can contain many neurons.

RNN: Recursions

Each time step of an RNN related h_t to h_{t-1} and x_t.

- At each time step, we pass in x_t (the current word).
- Additionally, also use the previous h_{t-1} (a past history).

Recursion Formula:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b_h), \quad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Prediction: at each time step, $\hat{y}_t = \text{sofmax}(W_y h_t + b_y)$.

Loss: at each time step, we have a Cross-Entropy (CE) loss $\mathcal{L}_t(\hat{y}_t, y_t)$.

The full loss per training pair $((x_1, \ldots, x_T), (y_1 = x_2, \ldots, y_T = x_{T+1}))$ is an average of the loss across all time steps $\frac{1}{T}\sum_{t=1}^{T} \mathcal{L}_t(\hat{y}_t, y_t)$.

Problems?

As the data gets longer, an RNN can no longer propagate dependencies between things far in the future and elements earlier on. Why?

Example: $W_h = a$, $W_x = b$ and $b_h = c$ are all scalars, with linear activation, then we have recursion $h_t = ah_{t-1} + bx_t + c$. Also, $W_y = d$ and $b_y = 0$ are scalars, then $\mathcal{L}_t(\hat{y}_t, y_t) = (\hat{y}_t - y_t)^2$.

Gradient: what is $\frac{\partial \mathcal{L}_1}{\partial a}$? use chain rule:

$$\frac{\partial \mathcal{L}_1}{\partial a} = \frac{\partial \mathcal{L}_1}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial h_1} \frac{\partial h_1}{\partial a} = 2(\hat{y}_1 - y_1)dh_0.$$

Similarly,

$$\frac{\partial \mathcal{L}_2}{\partial a} = 2(\hat{y}_1 - y_1)dh_1 + 2(\hat{y}_1 - y_1)dah_0.$$

RNN: Gradients Issues

Continue doing this for $t = 3, 4$, we obtain

$$\begin{aligned}\frac{\partial \mathcal{L}_4}{\partial a} = 2(\hat{y}_4 - y_4)da^3 h_0 + 2(\hat{y}_4 - y_4)da^2 h_1 \\ + 2(\hat{y}_4 - y_4)dah_2 + 2(\hat{y}_4 - y_4)dh_3\end{aligned}$$

If a is small, we have that the term measuring the dependency between the current time step and the one in the past is less and less.

- This is called the vanishing gradient problem.

If a is too large, we'll have an infinite gradient, another problem.

- This is called the exploding gradient problem.

RNN: Solving Gradient Issues

The exploding gradient problem can be solved by gradient clipping

$$g^{\text{clip}} = g \cdot \min\left(1, \frac{c}{\|g\|}\right).$$

The vanishing gradient problem might be solved by other ways

- Residual connections
- Xavier Initialization (some sort of smart initialization)
- Momentum based optimization algorithm (Adam)

None of the above are principled, they "might" work.

In practice, vanilla RNNs do not perform as good as they theoretically can due to the vanishing gradient problem.

- More complicated (LSTM or Transformers) models are used.

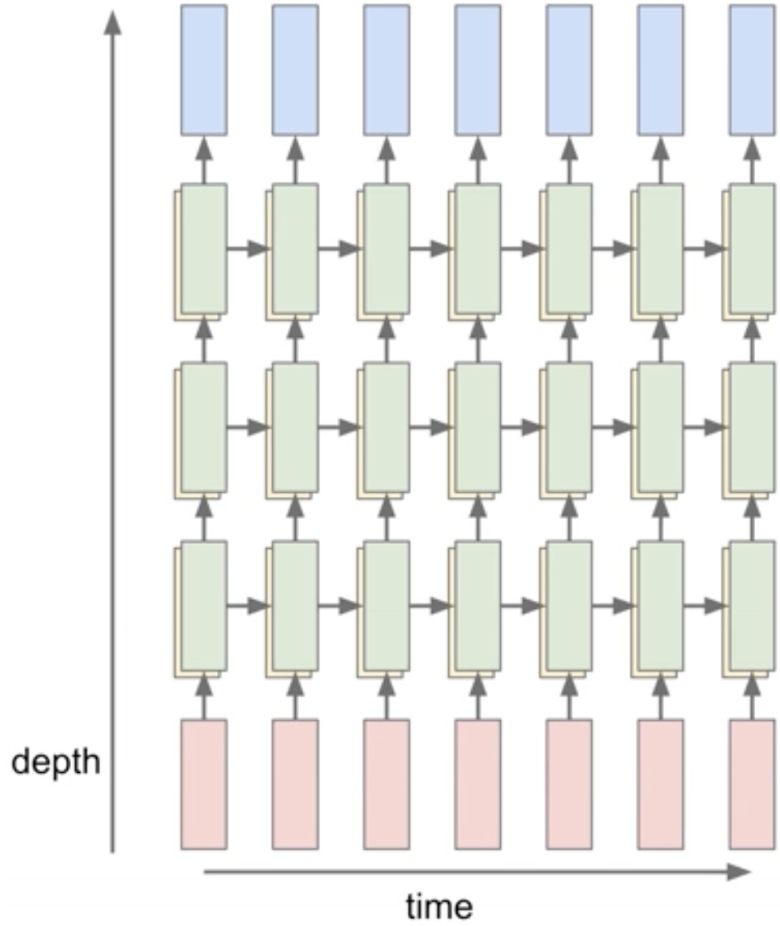
Extension (bonus)

Truncated BP: when computing gradients, doing back propagation in RNN models can be expensive since you need to go all the way back to $t = 0 \implies$ only back to $t = T - k$.

Bidirectional RNN: Some tasks need both past and future words to understand a word.

Multilayer RNN:

- more layers $\rightarrow$ vertical
- more time $\rightarrow$ horizontal



Outline

LSTM

Motivation: RNN suffer from vanishing gradient problems

Because of this, people have tried to come up with variants that resolve this issue, and LSTM (1997) are one of the most popular variants.

Empirically, an LSTM has been shown to have a memory of about 100 time steps, whereas a RNN has about just 7.

LSTM Dynamics

Each time step not only has h_t, but now there is a cell state c_t that is carried forward.

Recursion:

$$(f_t, i_t, o_t, g_t) = W_h h_{t-1} + W_x x_t + b$$

Cell State: key new feature

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot \tanh(g_t).$$

- $\sigma(f_t)$: how much old memory you want to keep
- $\sigma(i_t)$: how much new info you want to write

This cell state is a combination of old information and new information.

Hidden State:

$$h_t = \sigma(o_t) \odot \tanh(c_t).$$

LSTM Dynamics

The matrix $W_x = [W_{fx}, W_{ix}, W_{ox}, W_{gx}] \in \mathbb{R}^{d_h \times 4d_h}$ and $b = [b_f, b_i, b_o, b_g] \in \mathbb{R}^{4d_h}$, and $h_0, c_0 \in \mathbb{R}^{d_h}$.

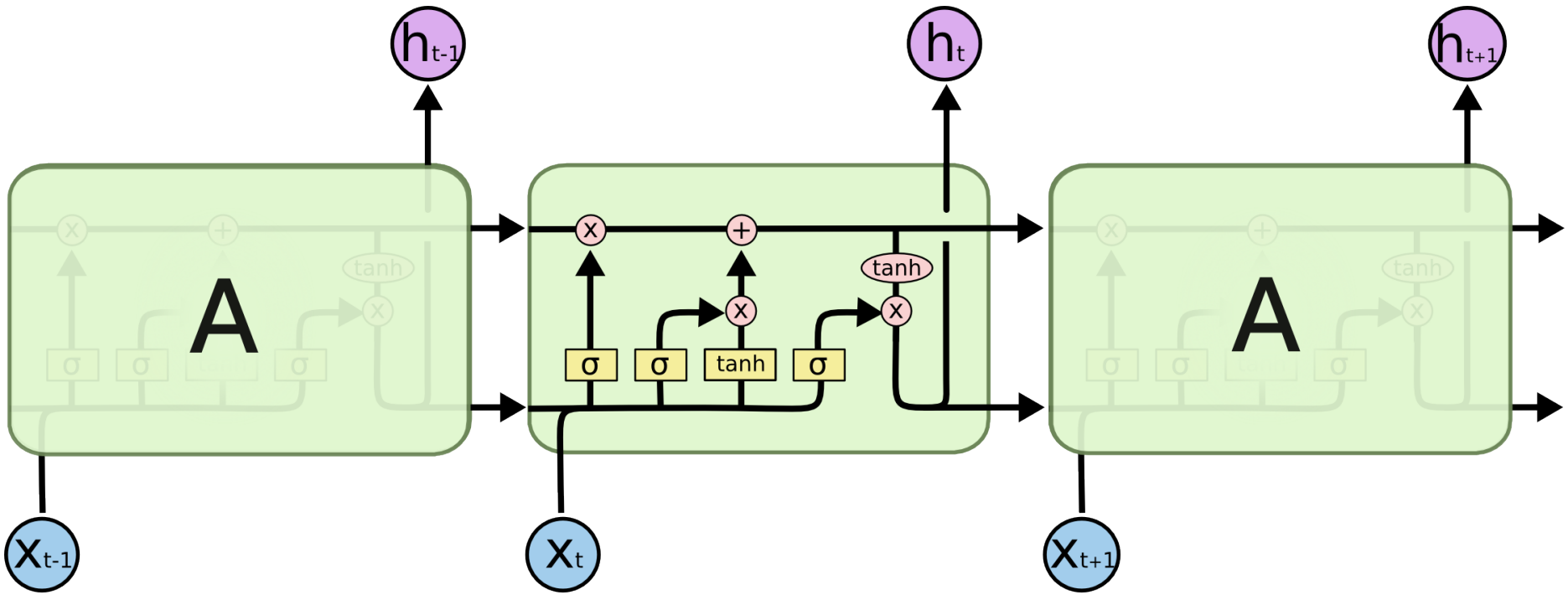
$\odot$ is the Hadamard product, or entrywise product: $[A \odot B] = A_{ij}B_{ij}$.

σ is sigmoid function.

The big idea of LSTM networks is that the added ct allows the networks to remember things

Benefit: There is less chance of vanishing gradient phenomenon and a current loss update can use information from far out in the past!

LSTM



Outline

Greedy Decoding

Decoding: How do we generate text using the model?

Usually, we start with a context like `"The man walks"` or a special start token `"START"` and begin decoding with the model.

Greedy decoding: always pick the highest probability next word and use that as your next word.

Once picked, add the word to your context and repeat until some terminal condition happens.

Greedy Decoding Example

Suppose the context is "The man walks".

Get $\arg\max_y p(y|\text{"The"}, \text{"man"}, \text{"walks"})$ and suppose this is "down".

Now the context is "The man walks down".

Get $\arg\max_y p(y|\text{"The"}, \text{"man"}, \text{"walks"}, \text{"down"})$ and suppose this is "the".

Etc. Repeat this until the context is done or until a special end "END" token is generated.

Greedy is not Optimal

Remember we really want to solve: $\max_y p(y_1, \ldots, y_T)$.

Greedy, generally, will not give the optimal solution because it is too short-sighted.

Also, human don't always pick the word with the next highest probability to generate the things you say.

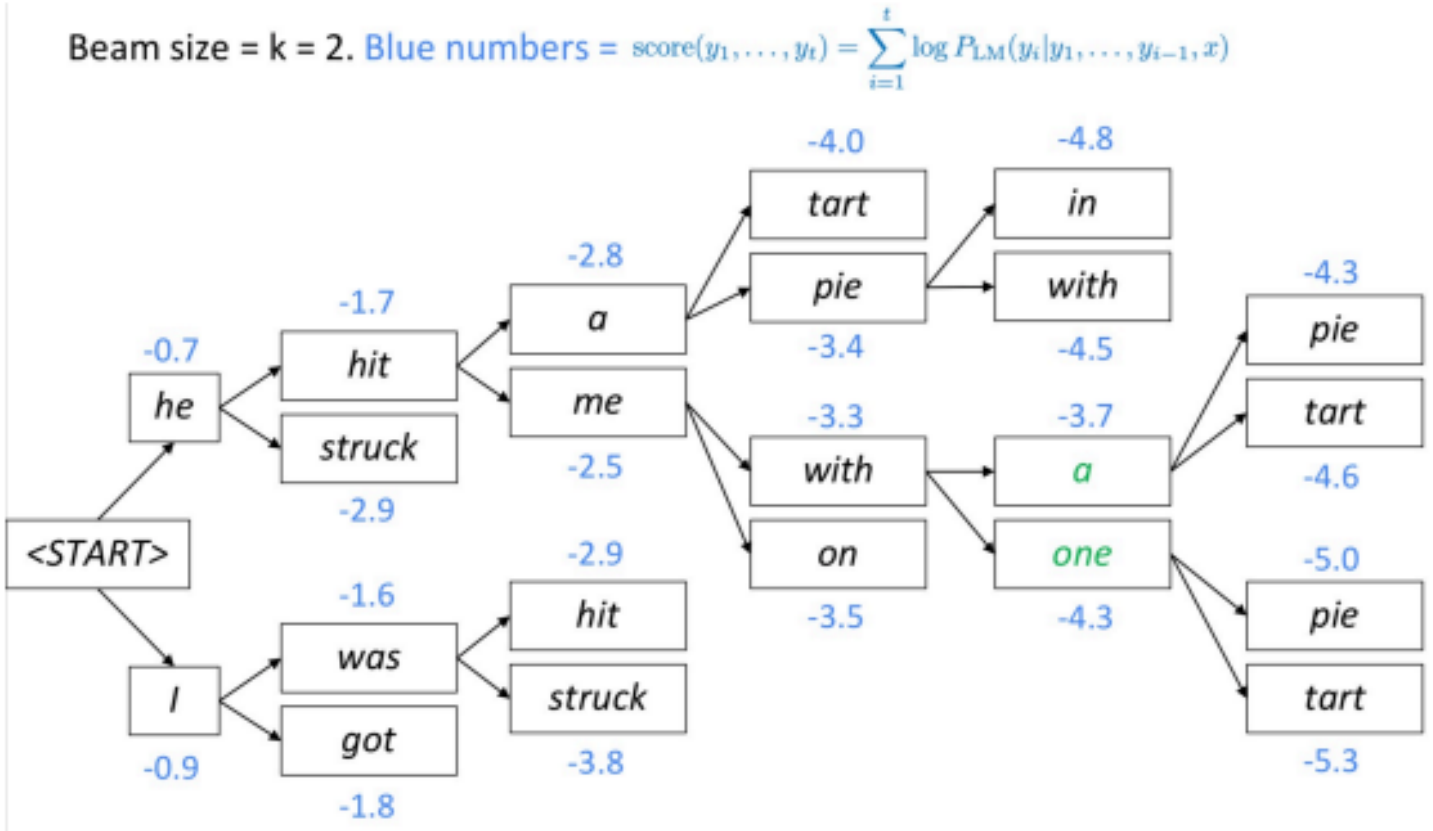
Sometimes, you make a lower probability choice but in the long run this leads to higher probability choices overall.

Another idea: Beam Search.

Beam Search Decoding

Instead of picking the next highest probability word to add to your context, store a beam of size k as your set of candidates.

Example: $k = 2$,



Thank you for your attendance!

Questions?

Next week (Lab session):

- Use Pytorch to train a RNN/LSTM on WikiText2 data set.
- Data loading.
- Model setup.
- Model training.
- Model evaluation.