RNNs/GRUs/LSTMs as LM

Pavlos Protopapas

Outline

- Recap: Language Models
- Recap: Neural Networks as Language Models
- Bidirectional two-layer LSTM Language Model
- Embeddings and how we use them

Outline

- **Recap: Language Models**
- Recap: Neural Networks as Language Models
- Bidirectional two-layer LSTM Language Model
- Embeddings and how we use them

Recap: Language Models

A Language Model predicts the next word x_{t+1} in a sentence based on previous words.

Words

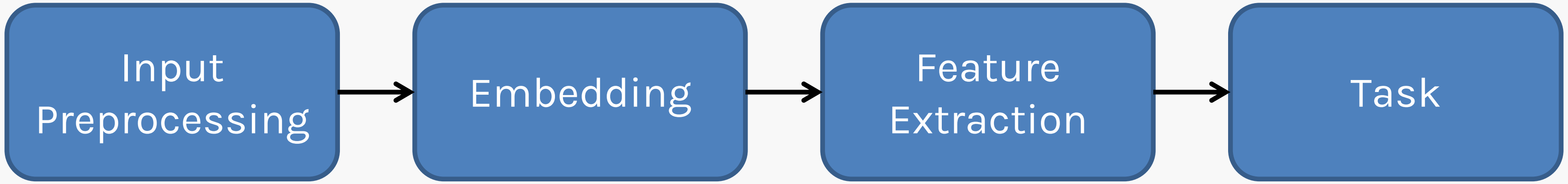
$$P(x_{t+1} | x_t, x_{t-1}, x_{t-2}, \ldots, x_0; \Theta)$$

Remember that in any model, the learned parameters always condition the model.

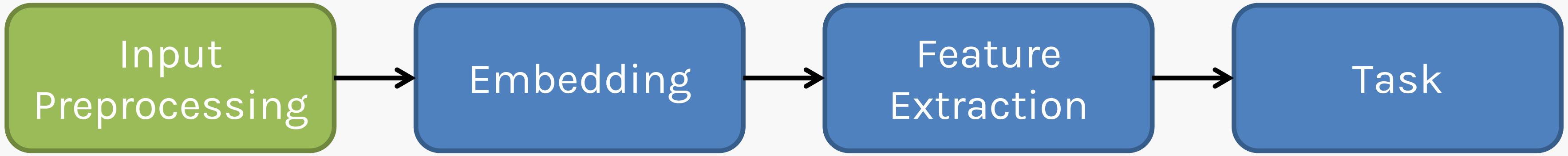
Recap: Language Models

So far, we have been talking about embeddings, neural networks and language models, but without the complete picture.

We can split the language model pipeline into four parts:



Recap: Language Models



Input:

The entire collection of data is the corpus in which we train on. The corpus can be a small dataset such as IMDB, or an enormous such as Wikipedia.

From that corpus we can define a vocabulary, which is composed of the unique words in the corpus. It will influence the complexity of our language model.

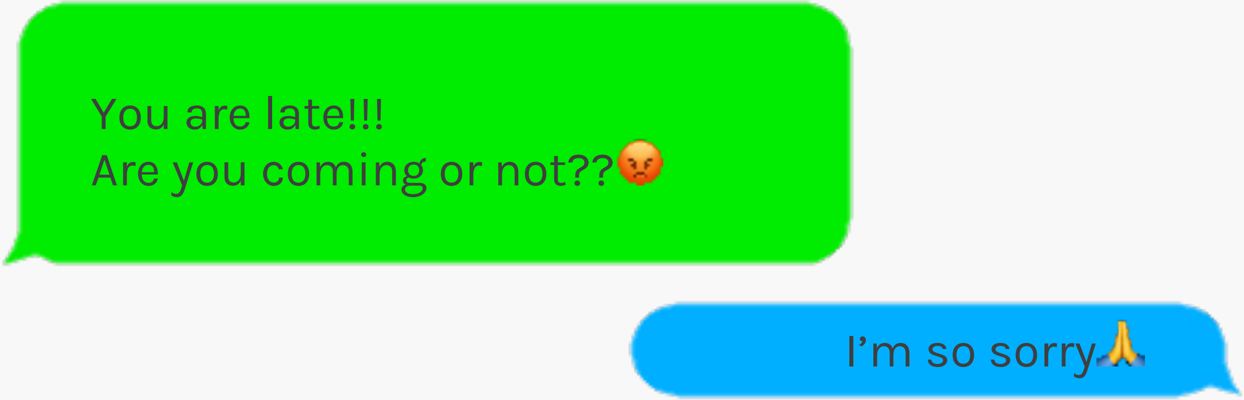
Recap: Language Models

Input Preprocessing → Embedding → Feature Extraction → Task

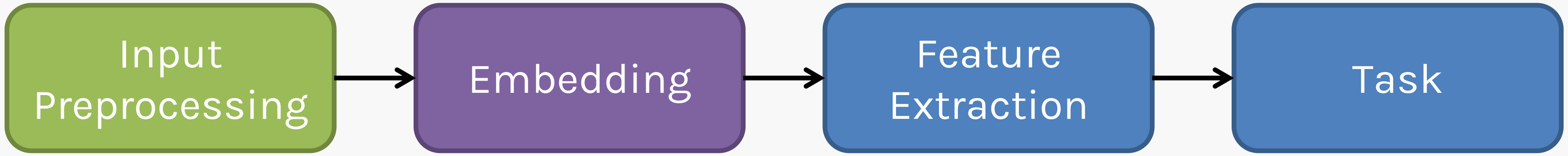
Preprocessing:

The data generally is collected from sources such as the internet. It comes as it is, with punctuation signs, grammatical mistakes and even emojis.

In general, language models often use preprocessed text, without uppercase letters, and where punctuation signs and non-text data are removed.



Recap: Language Models

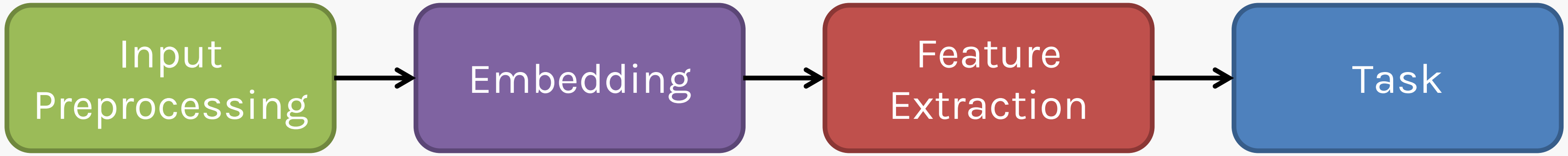


Embedding:

This stage transforms the data to be fed to the network. The embedding layer transforms every word into a fixed length vector with arbitrary dimension.

One-hot-encoding or word2vec are some examples.

Recap: Language Models

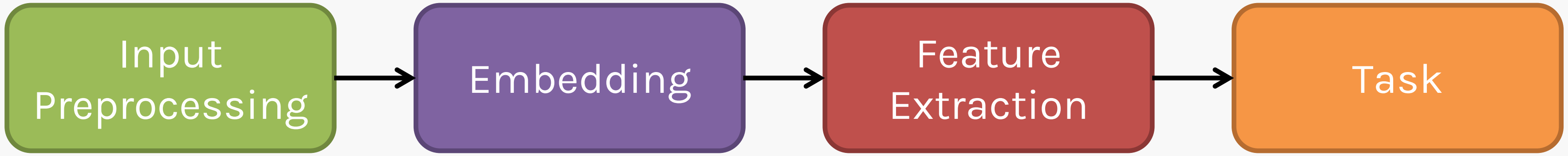


Feature extraction:

This stage is the model. Usually a neural network.

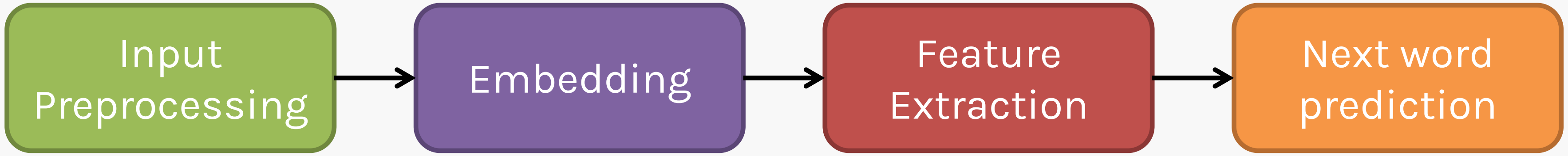
From the sequence of embeddings, we generate a new representation which will be used in the next stage. Its job is to capture as much contextual information as possible.

Recap: Language Models

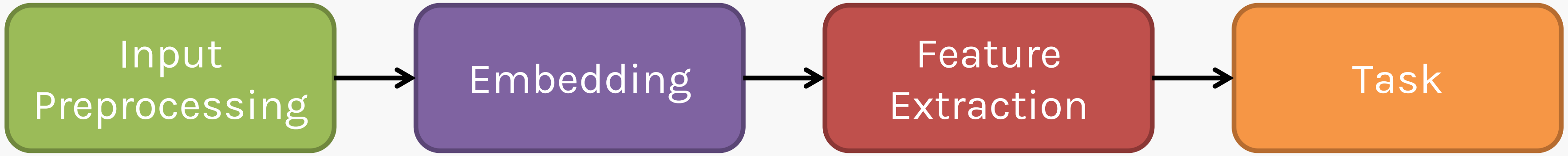


Task: The task of the language model is to predict the next word in the sentence.

Example: Next word prediction



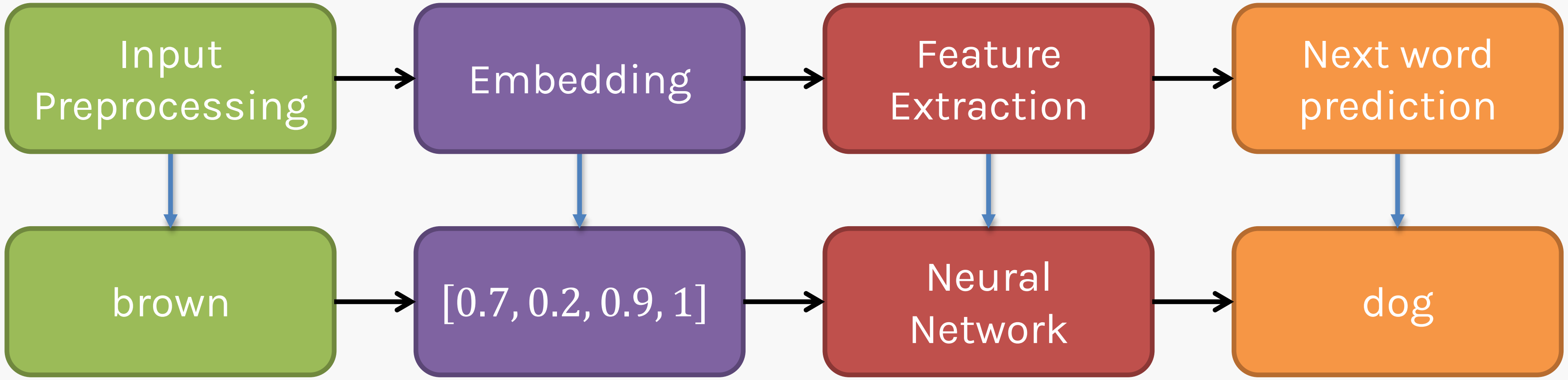
Recap: Language Models



Task: The task of the language model is to predict the next word in the sentence.

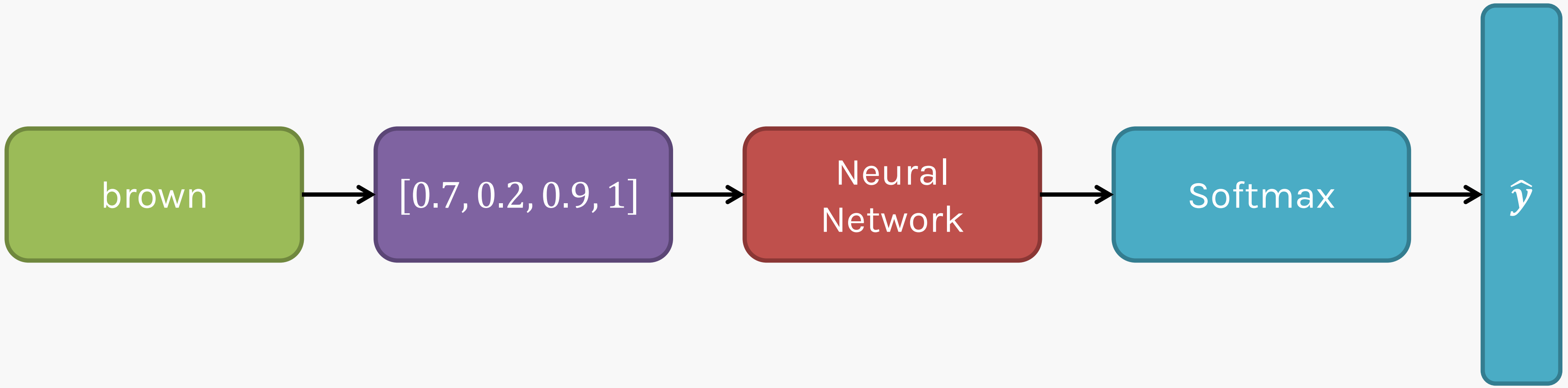
Example: Next word prediction

“The brown dog”



Recap: Language Models

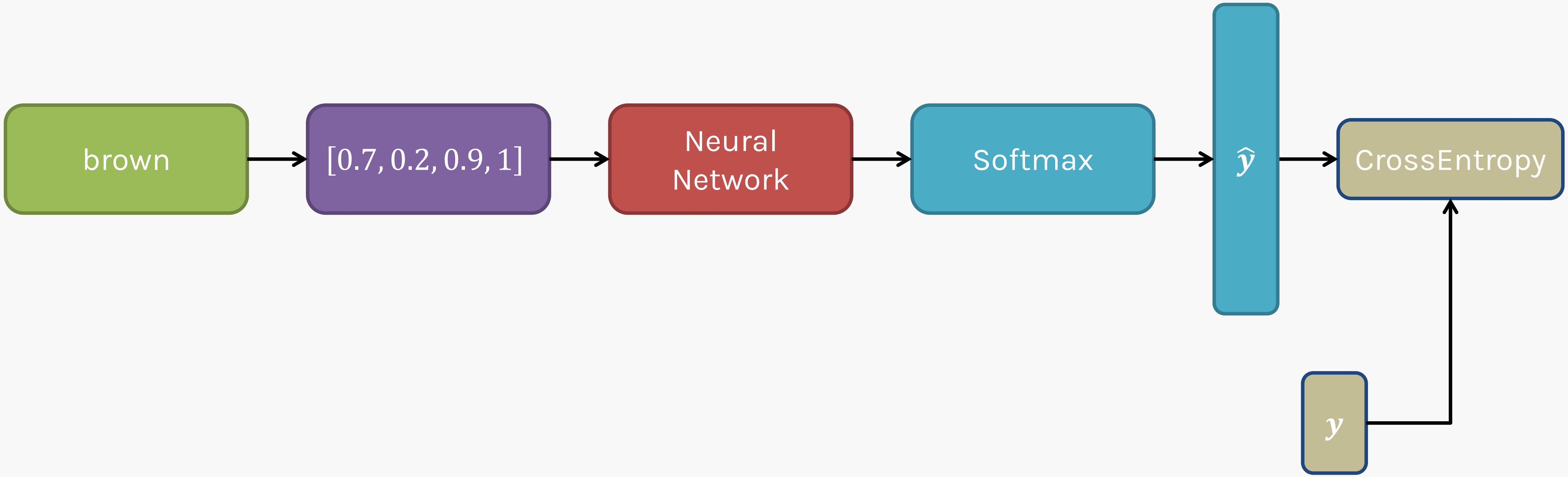
The prediction is softmax-normalized, and its values represent the probability of each word.



Recap: Language Models

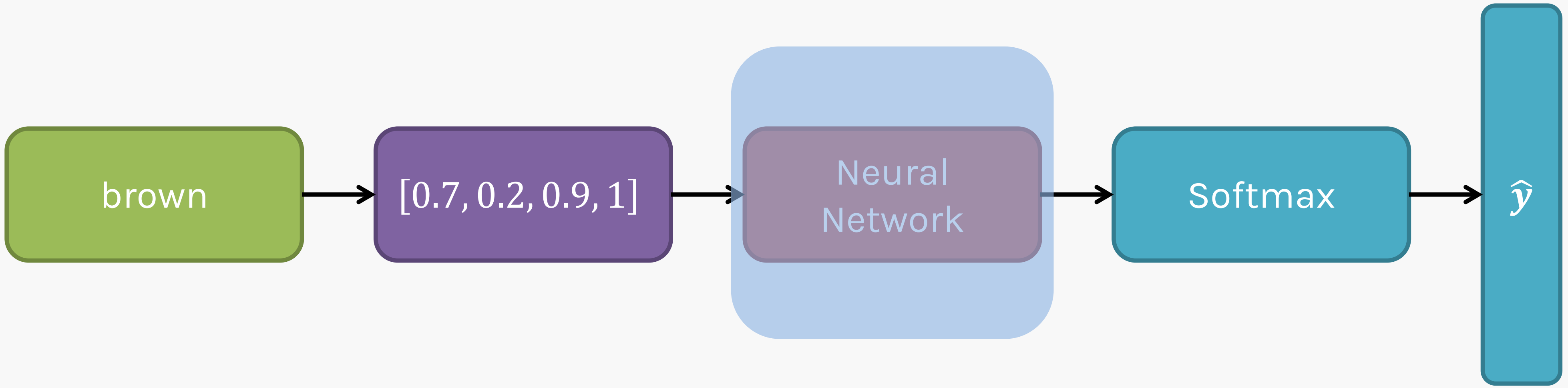
To train the model, we need to compare our normalized prediction with the ground truth $\boldsymbol{y}$, using the Cross-Entropy loss.

In general, $\boldsymbol{y}$ is one-hot encoded.



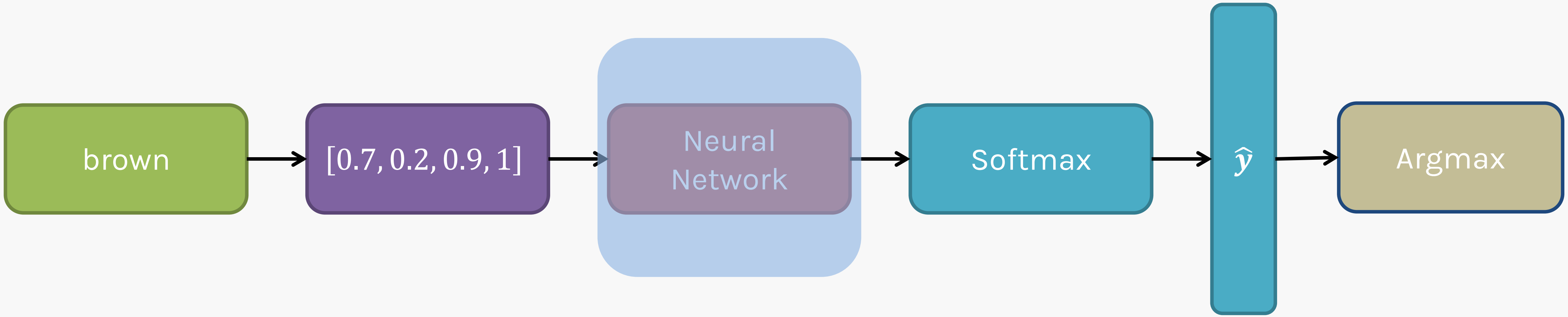
Recap: Language Models

Once the model is trained, we freeze the learned weights of the network. They will not change anymore.



Recap: Language Models

Our predicted word is the one with the maximum probability.

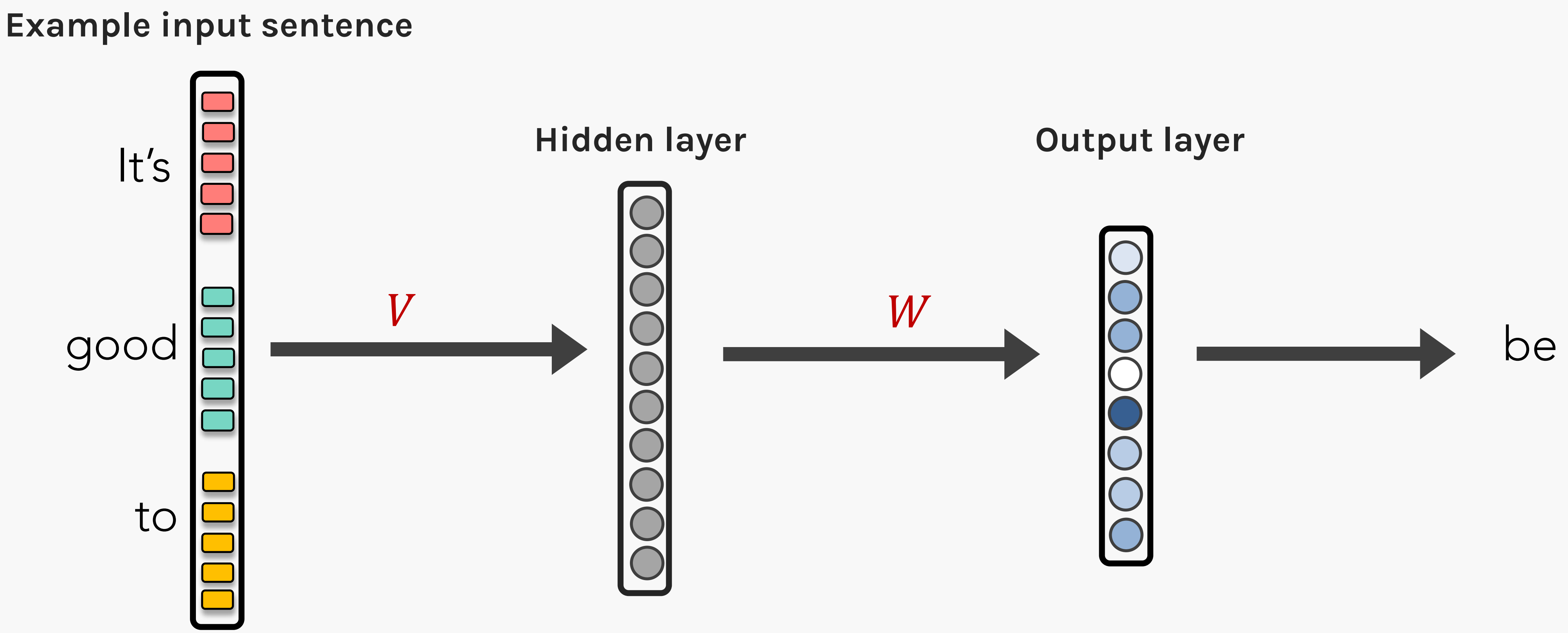


Outline

- Recap: Language Models
- **Recap: Neural Networks as Language Models**
- Bidirectional two-layer LSTM Language Model
- Embeddings and how we use them

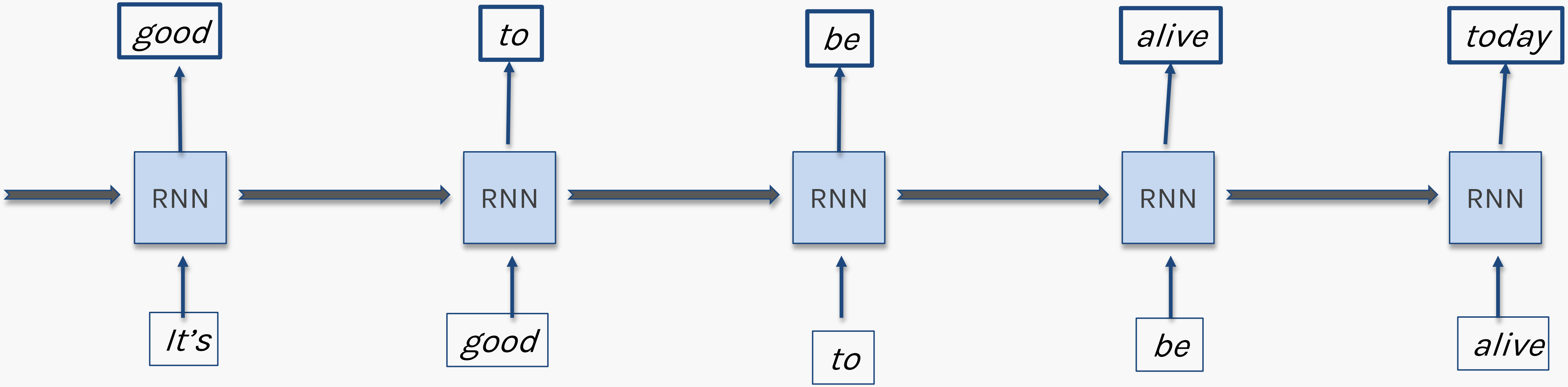
Recap: Neural Networks as Language Models

We have seen a FFCC network as a LM in previous lectures.



Recap: Neural Networks as Language Models

We have used an RNN to process text:

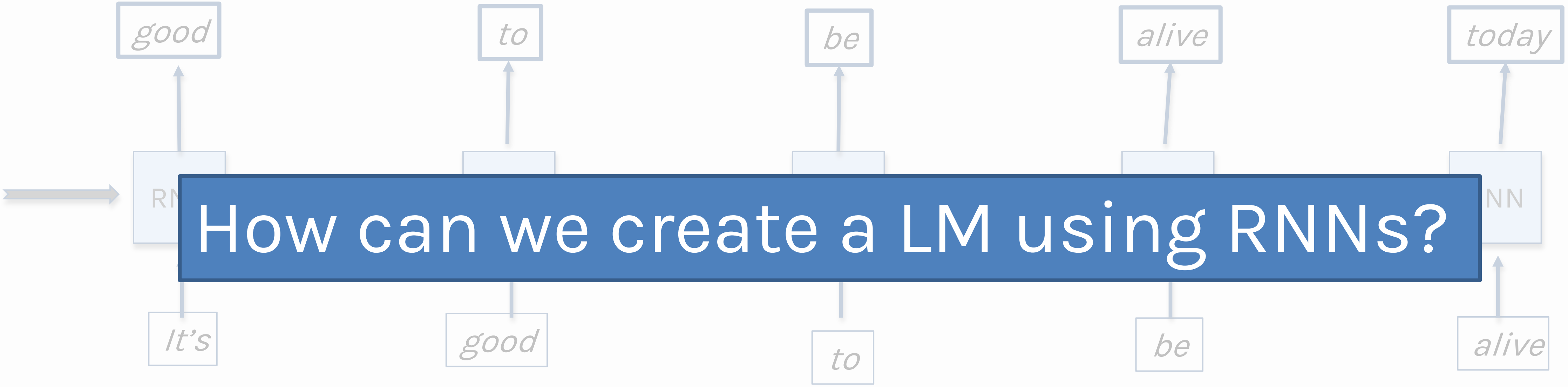


The key advantage of RNN is that they have a memory and and be unrolled to variable length sentences.

In RNN, we use each hidden state to predict the corresponding word.

Recap: Neural Networks as Language Models

We have used an RNN to process text:



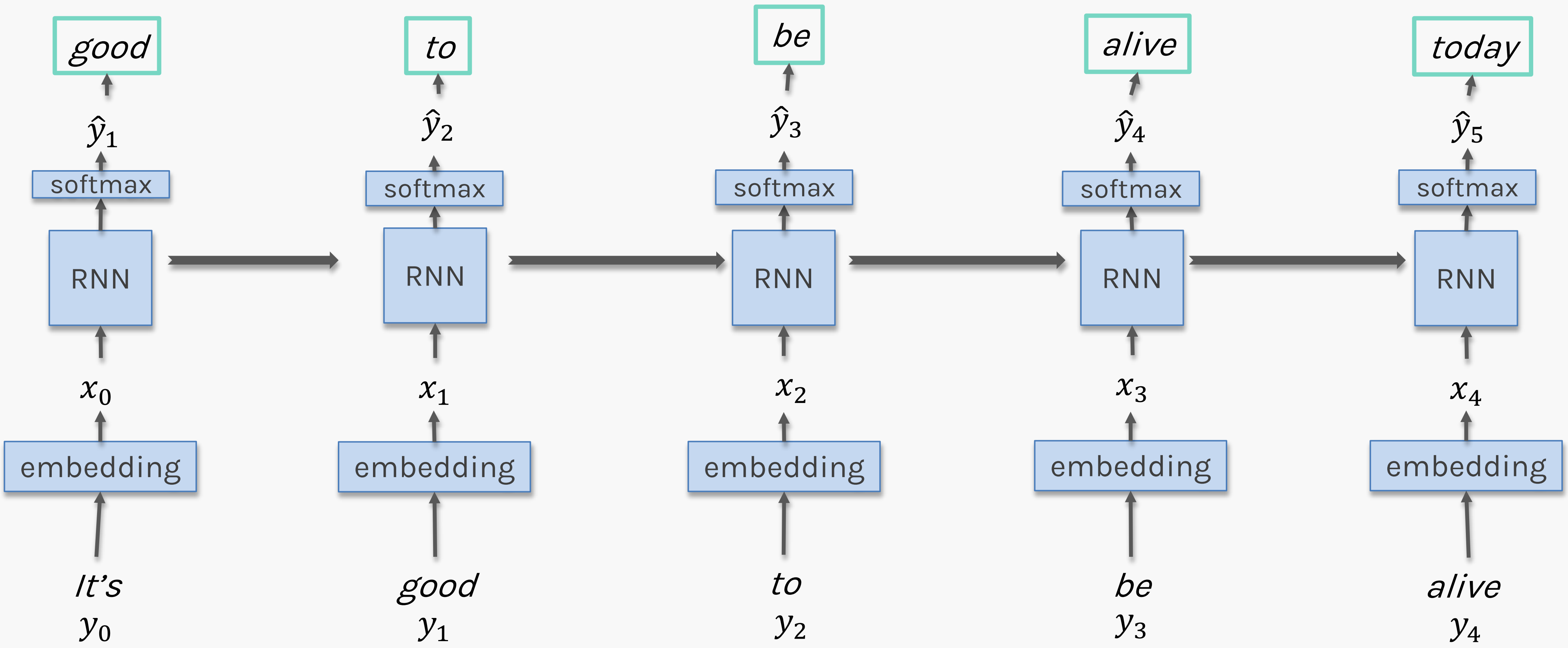
How can we create a LM using RNNs?

The key advantage of RNN is that they have a memory and and be unrolled to variable length sentences.

In RNN, we use each hidden state to predict the corresponding word.

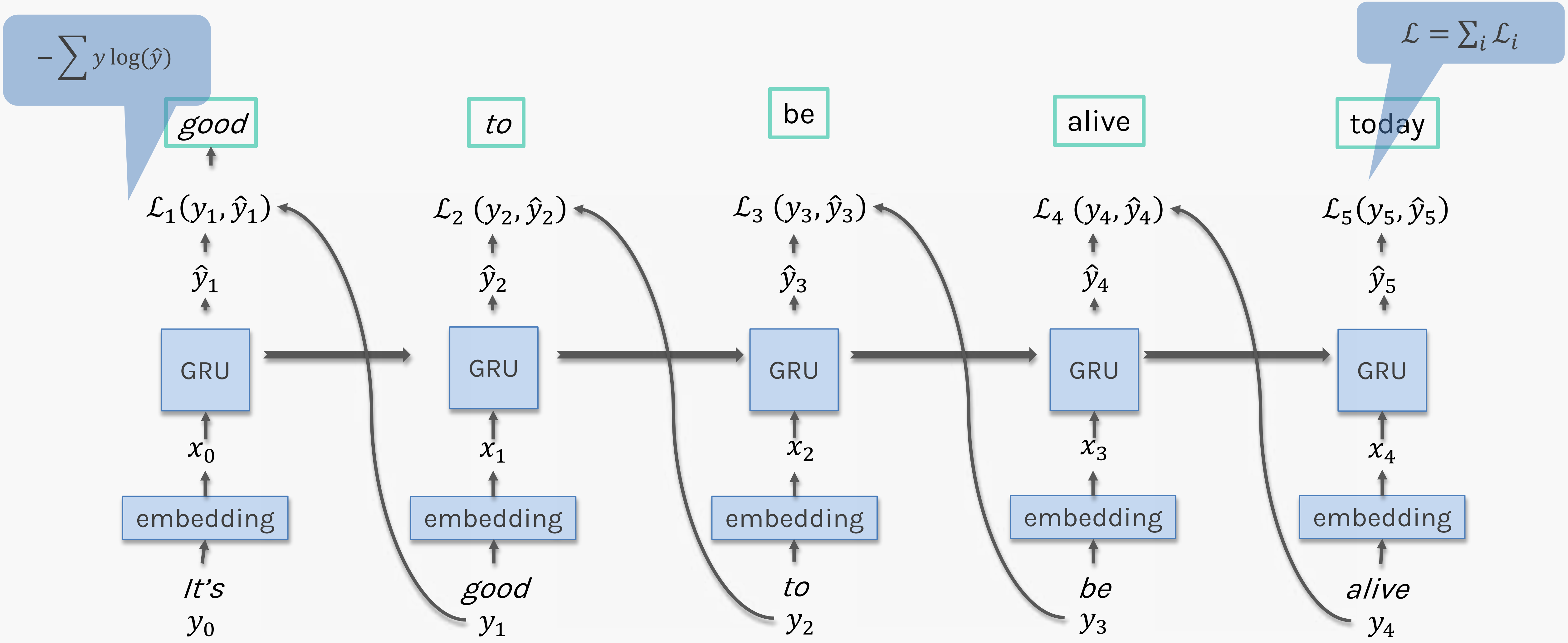
Recap: Neural Networks as Language Models

Using an RNN/GRU/LSTM as an LM will look like this:



Recap: Neural Networks as Language Models

Using an RNN/GRU/LSTM as an LM will look like this:

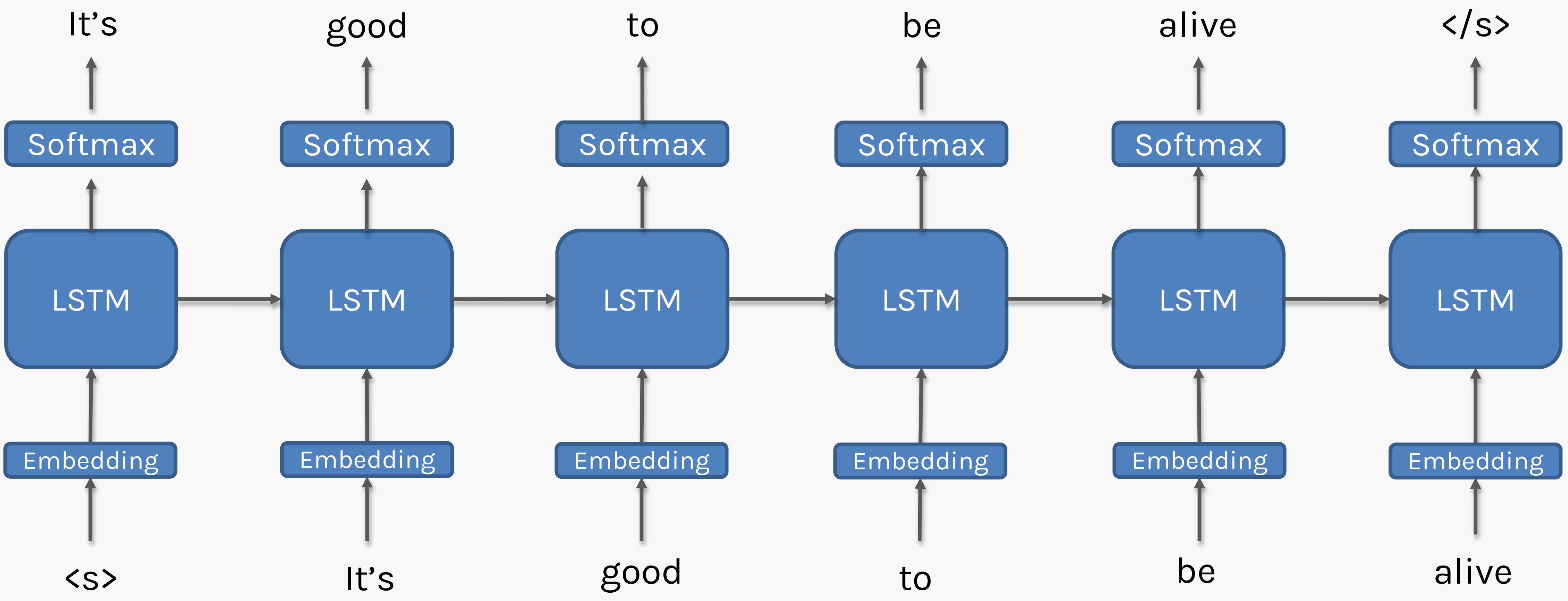


Outline

- Recap: Language Models
- Recap: Neural Networks as Language Models
- **Bidirectional two-layer LSTM Language Model**
- Embeddings and how we use them

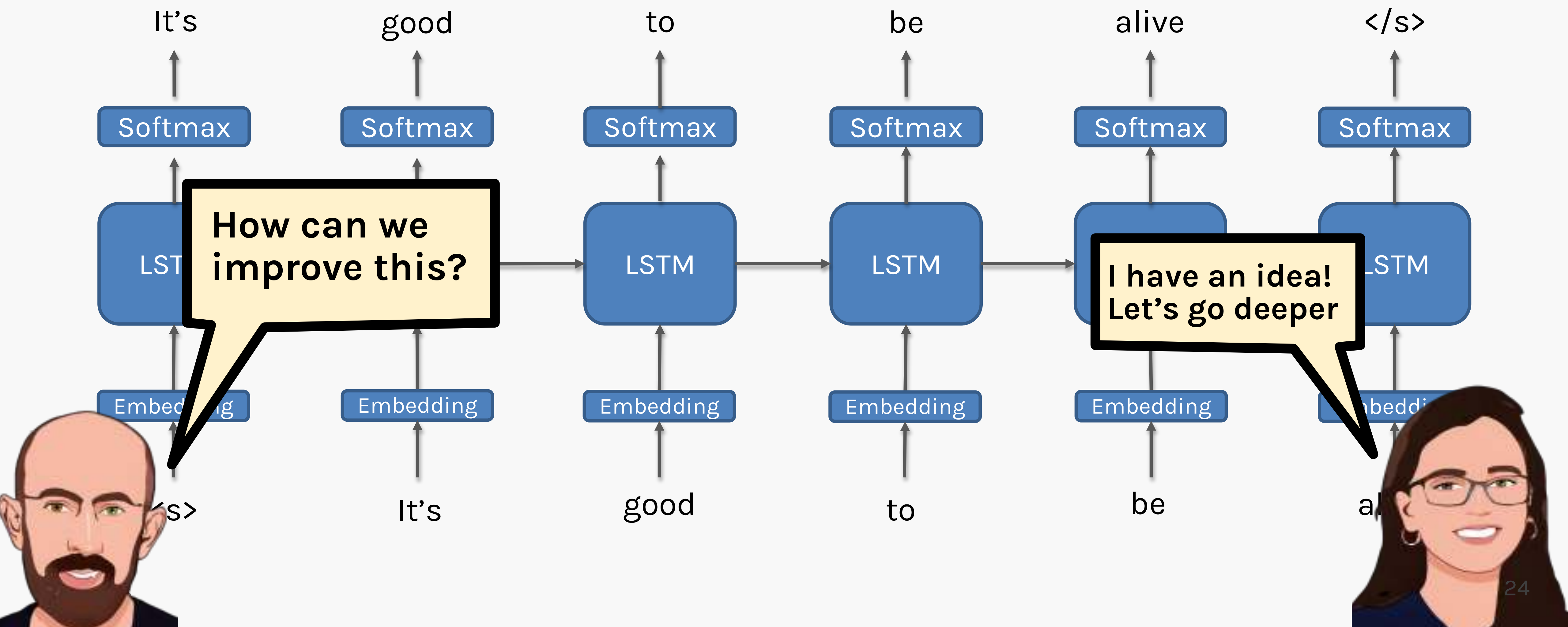
Bidirectional two-layer LSTM Language Model

To create the LM embeddings we could use an LSTM network.



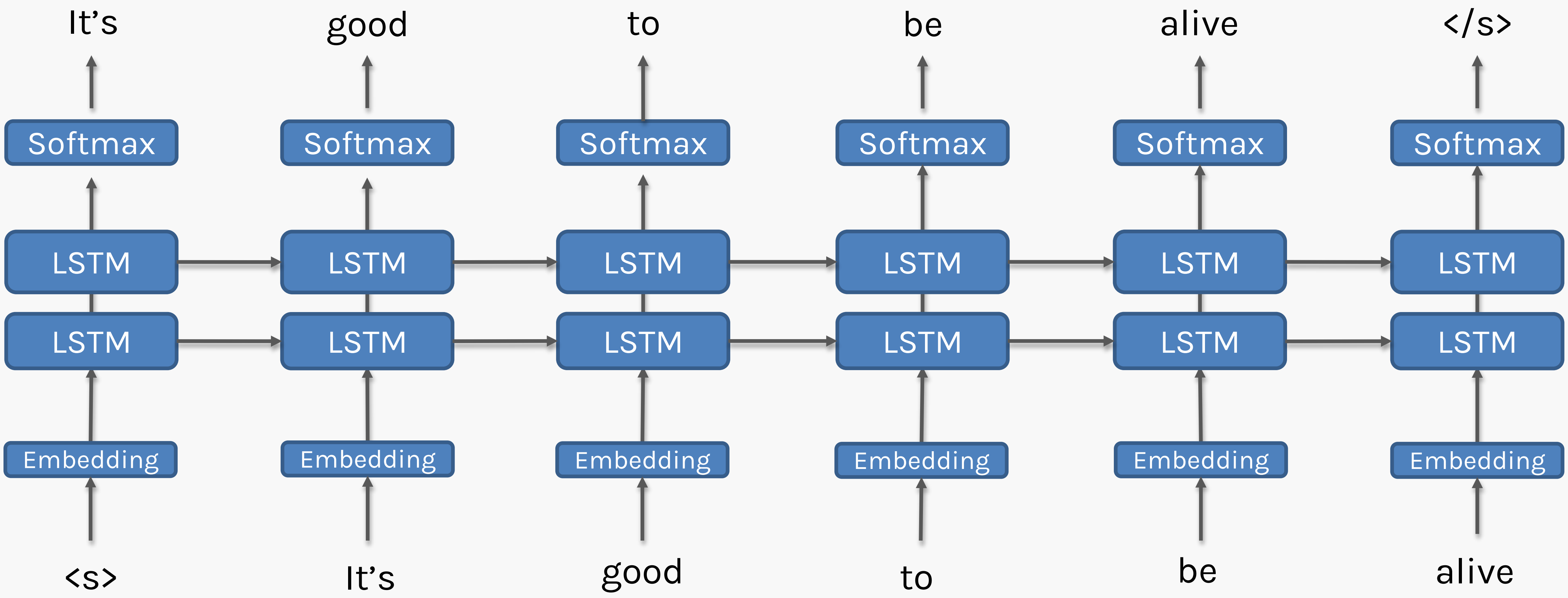
Bidirectional two-layer LSTM Language Model

To create the LM embeddings we could use an LSTM network.



Bidirectional two-layer LSTM Language Model

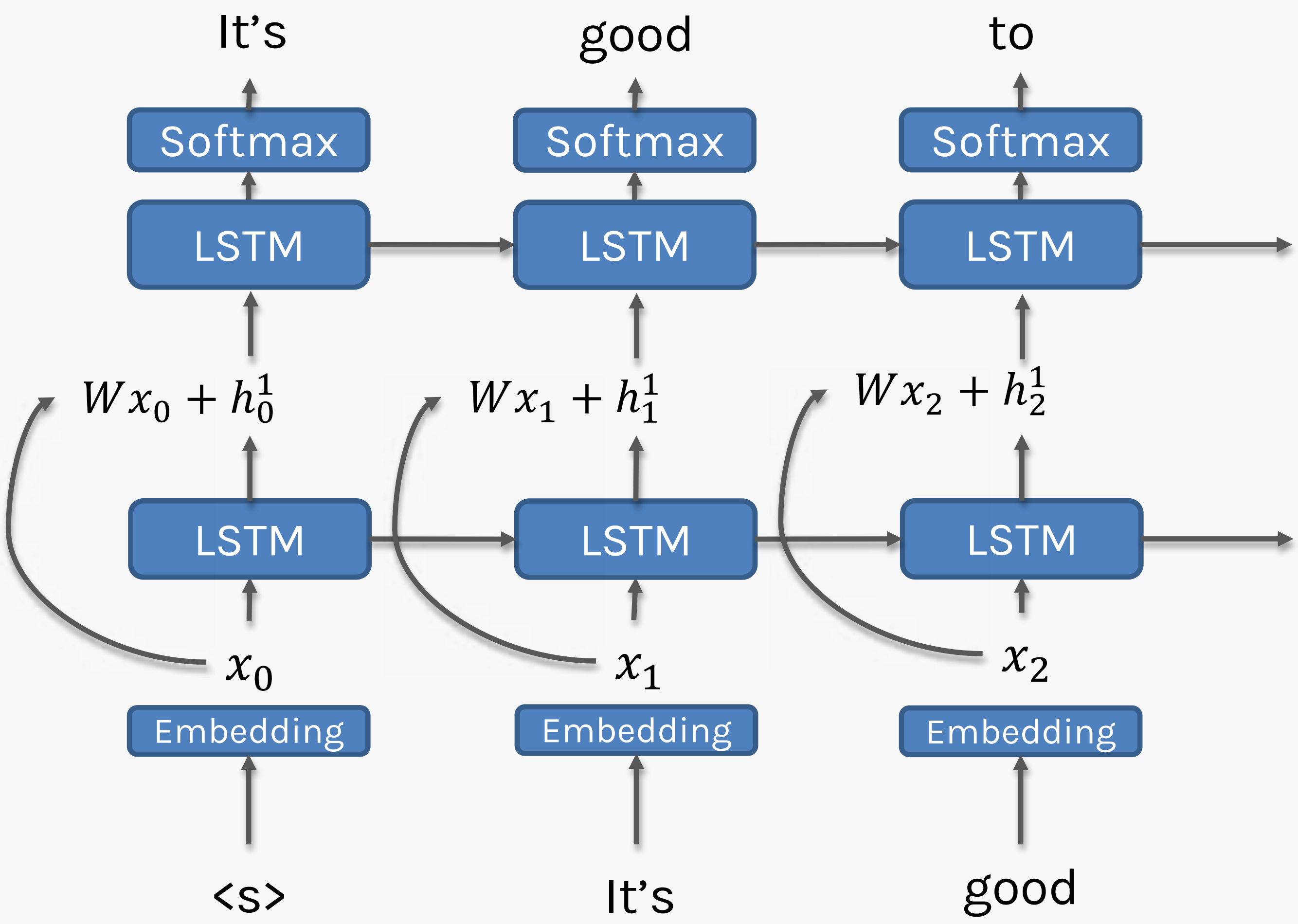
The model can be improved by adding another recurrent layer.



However, if we want to go even deeper, we will face optimization issues.

Bidirectional two-layer LSTM Language Model

We can add a residual connection between the LSTM layers.



The output h_t^1 is the sum of the output of the LSTM and the input **x**.

$$h_t^1 = LSTM_1(x_t) + Wx_t$$

W is a linear transformation to adjust the dimensions of the input.

Bidirectional two-layer LSTM Language Model

The semantic information is not always contained in the past.

Some words in the future might be important to understand the information in the present.

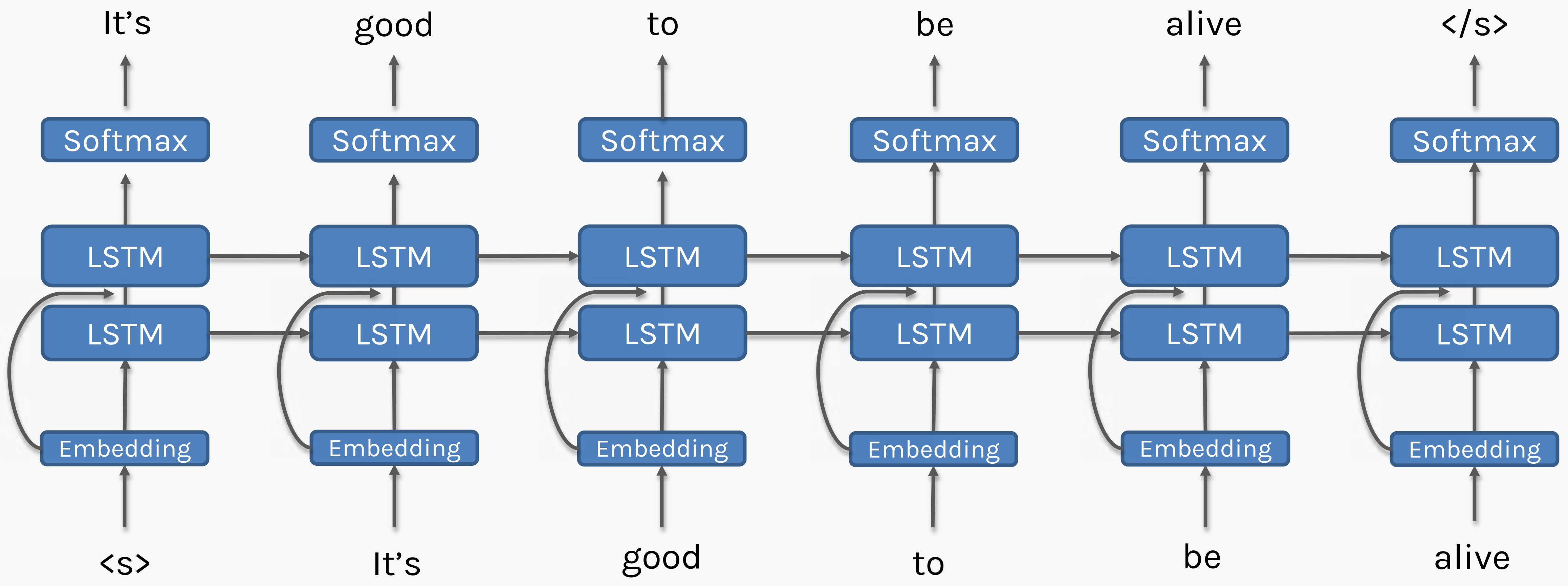
That is why we listen until the other person stops talking!

We can improve the model using another LSTM going in the opposite direction.

We predict the previous word in the sentence, starting from the end.

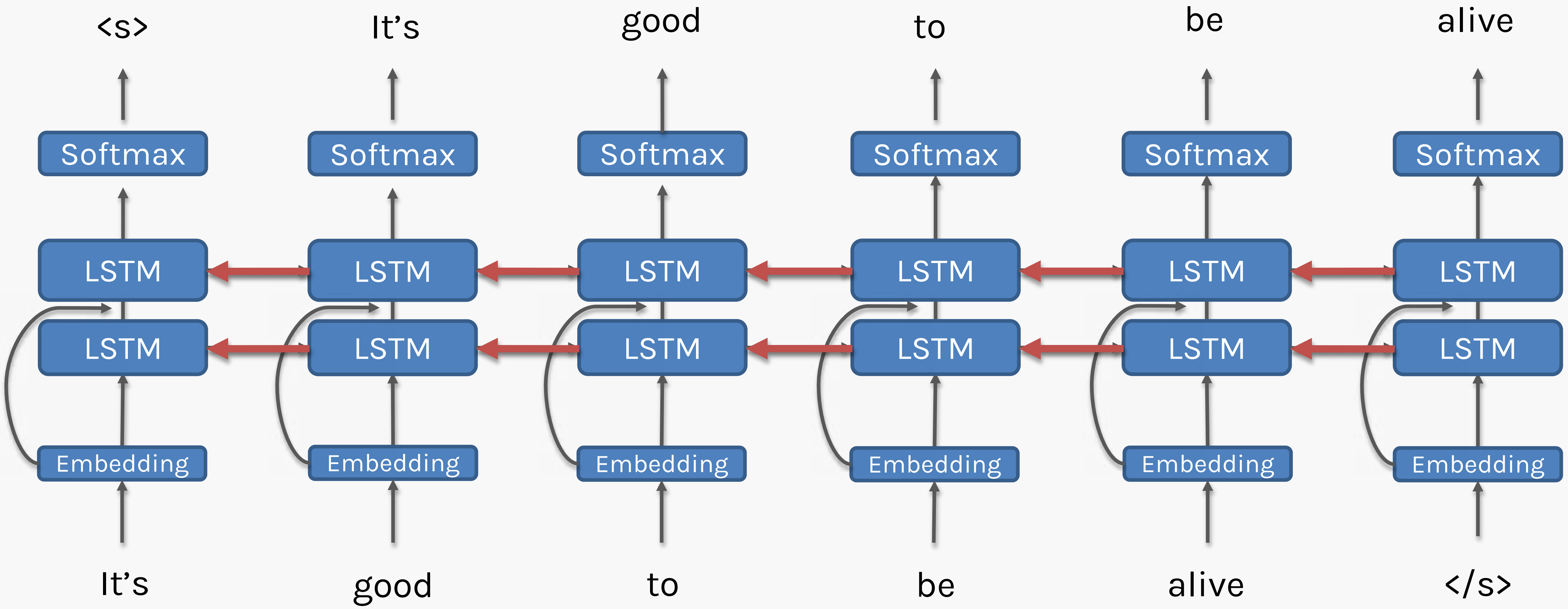
Bidirectional two-layer LSTM Language Model

We replicate the same structure…



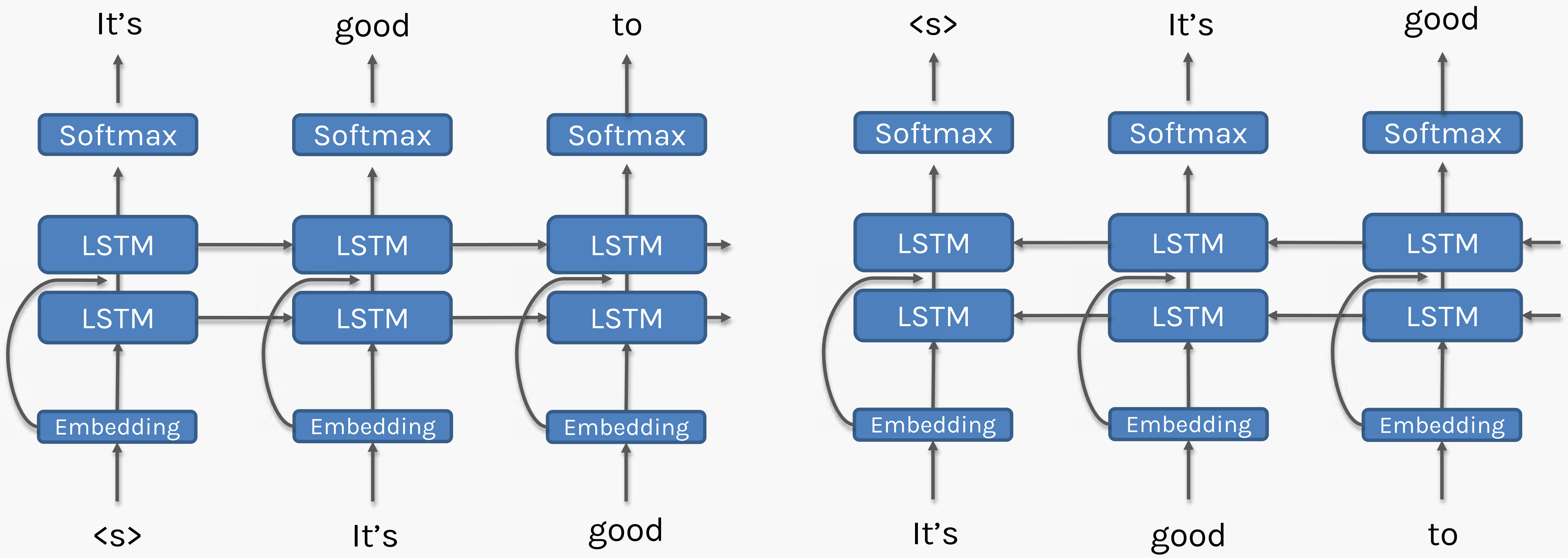
Bidirectional two-layer LSTM Language Model

We replicate the same structure but in the opposite direction!

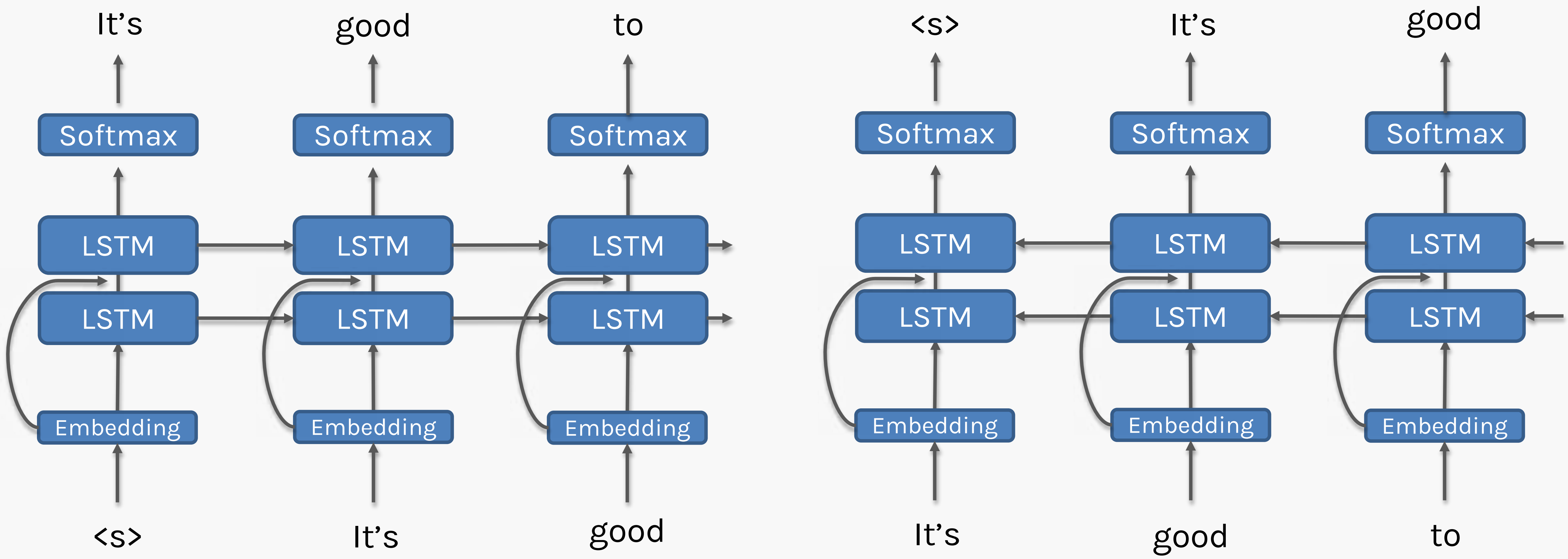


Bidirectional two-layer LSTM Language Model

The final model has two networks:



Bidirectional two-layer LSTM Language Model



Bidirectional two-layer LSTM Language Model

We optimize **both** networks at the same time. For each word, we optimize both RNNs jointly:

$$L = -\sum y \log(\hat{y}_{right}) - \sum y \log(\hat{y}_{left})$$

Each direction of the LSTM predict the same word, but from different directions.

We will revisit this part later.

Bidirectional two-layer LSTM Language Model

What do we do?

What is the final y?

No inference – just the training for the embedding

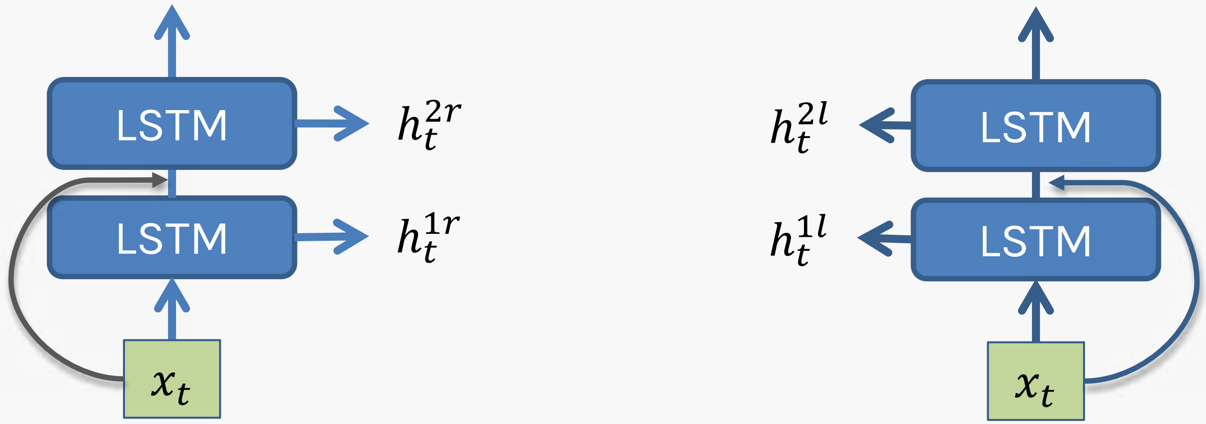
Outline

- Recap: Language Models
- Recap: Neural Networks as Language Models
- Bidirectional two-layer LSTM Language Model
- **Embeddings and how we use them**

Embeddings

These are the famous “**E**mbeddings from **L**anguage **Mo**dels”

$$E_t = \{x_t, h_t^{1l}, h_t^{2l}, h_t^{1r}, h_t^{2r}\}$$



Embeddings

We concatenate the embeddings at each recurrent level,

$$h_t^1 = \{h_t^{1l}, h_t^{1r}\}$$
$$h_t^2 = \{h_t^{2l}, h_t^{2r}\}$$

And we rename the word embeddings as

$$h_t^0 = x_t$$

And the new embedding takes the form:

$$\boxed{E_t = \{h_t^0, h_t^1, h_t^2\}}$$

Dimensions:

h_t^0: same as r and x, e.g. 64

h_t^1: same as $h_t^0 \rightarrow 64$

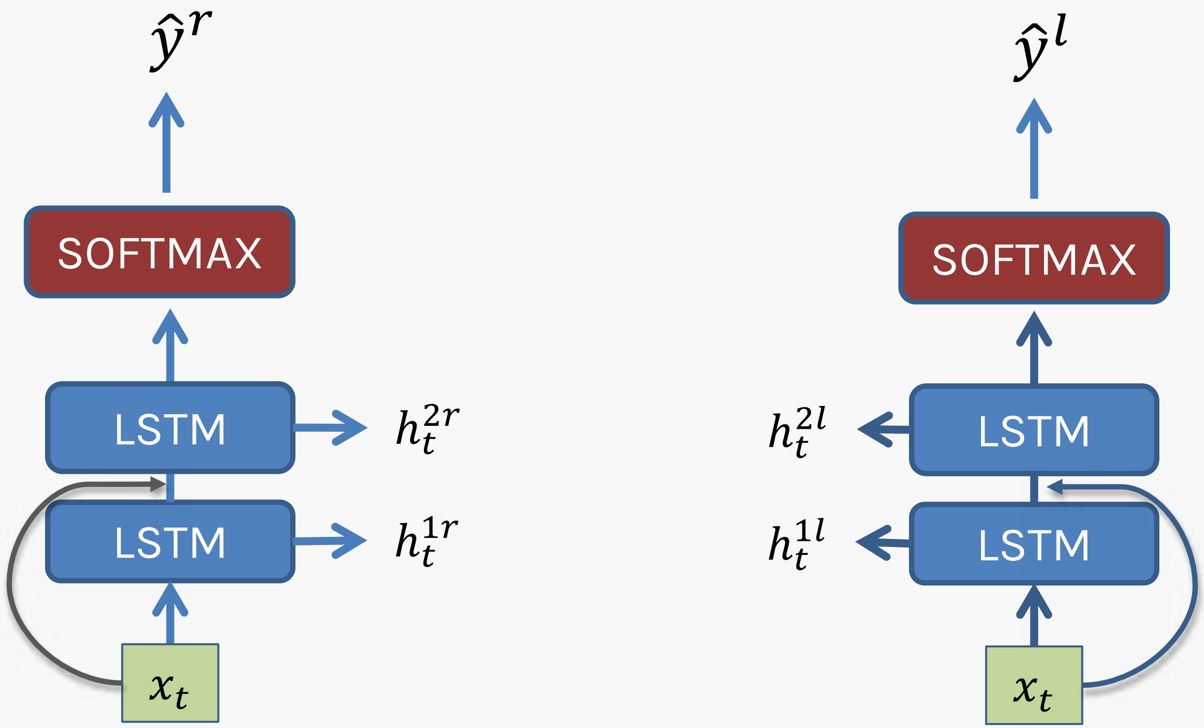
$h_t^{1l} \rightarrow 32$, $h_t^{1r} \rightarrow 32$

h_t^2 : same as $h_t^0 \rightarrow 64$

$h_t^{2l} \rightarrow 32$, $h_t^{2r} \rightarrow 32$

Embeddings training

$$L = -\sum y \log(\hat{y}^r) - \sum y \log(\hat{y}^l)$$



Embeddings

The new embedding takes the form:

$$\boxed{E_t = \{h_t^0, h_t^1, h_t^2\}}$$

Once trained, we freeze them and use it to extract the multi-level representations, $E_t = \{h_t^0, h_t^1, h_t^2\}$.

For each sentence we can create a context-dependent embedding.

LM embeddings: How to use

Now we are ready to replace word2vec.

Why are we so eager to replace something that works?

I can play guitar pretty well.

They went to play in the park.

Romeo and Juliet is a tragic play.

Embeddings: How to use

I can play guitar pretty well.

They went to play in the park.

Romeo and Juliet is a tragic play.

We must learn all the different meanings

Word2vec is context-independent.

Which means that the same word used in different contexts will have the same embedding.

Embeddings: How to use

We now want to use these embeddings as an input.

Does this mean we have to change the model entirely?

Embeddings: How to use

We now want to use these embeddings as an input.

Does this mean we have to change the model entirely?

Not at all!

Embeddings: How to use

We use the embeddings as our **sauce**.

We concatenate the embeddings E_t to the ones we use to feed the model for our other task.

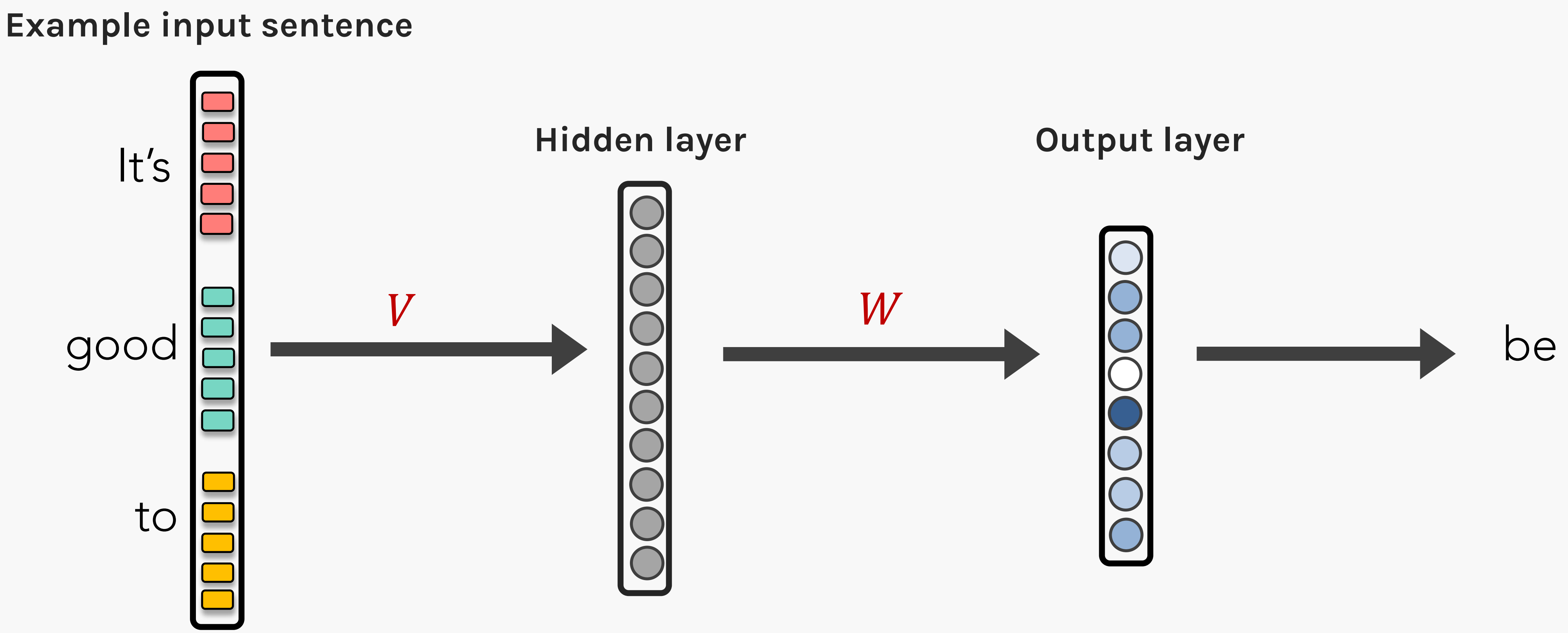
$$e'_t = \{e_t, E_t\}$$

Any other embedding

LM embeddings

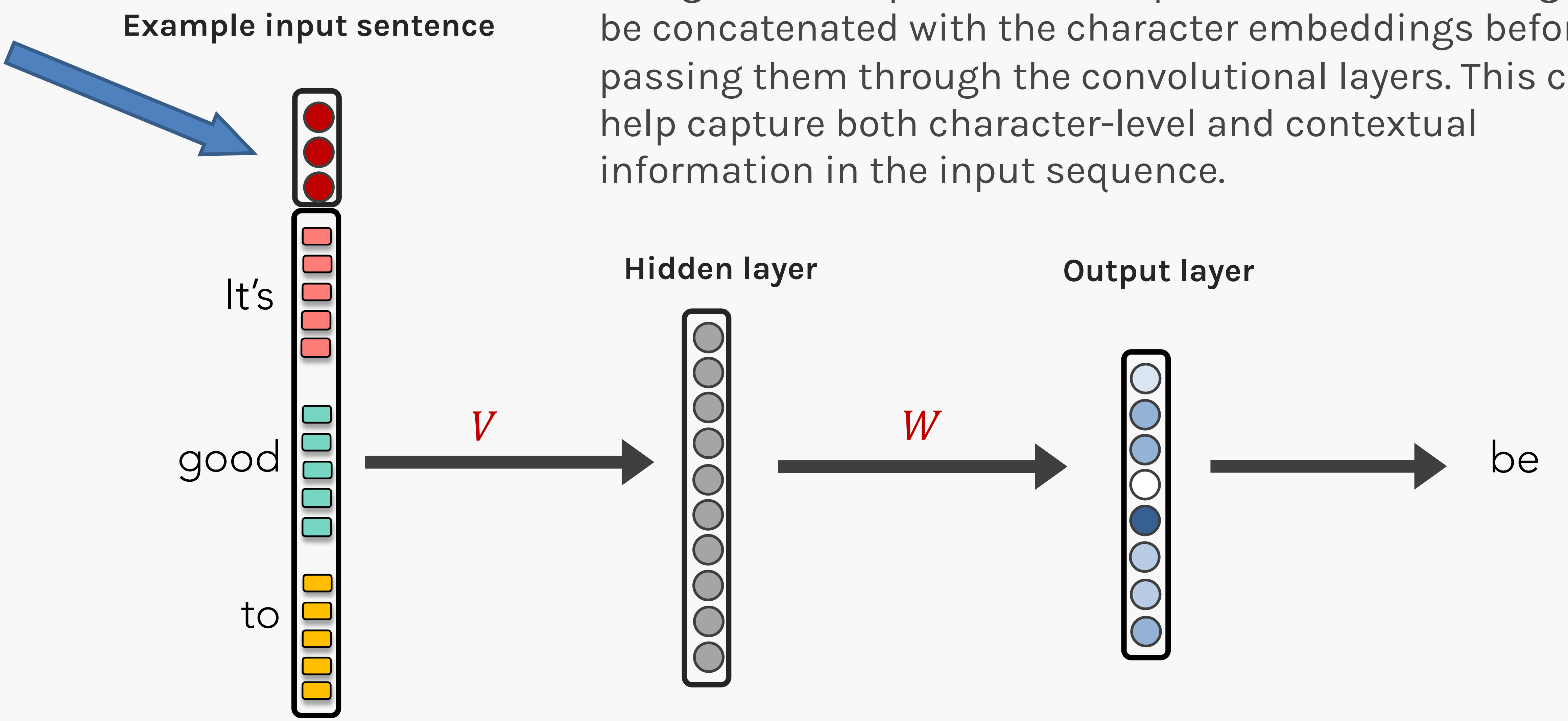
Embeddings: How to use

Going back to a previous example,



Embeddings: How to use

Going back to a previous example, the LM embeddings can be concatenated with the character embeddings before passing them through the convolutional layers. This can help capture both character-level and contextual information in the input sequence.



Embeddings: How to use

We use the embeddings as our secret sauce.

We concatenate our embeddings to the ones we use to feed the model for our other task.

$$e'_t = \{e_t, E_t\}$$

Instead of using a pre-made sauce, we can prepare our own.

Embeddings: **Secret Sauce**

If we want a tastier meal, we can choose the proportions between salt, pepper and garlic.

And the overall quantity to include them in the recipe.

The same way we do with the condiments of a meal. Our secret sauce give us the flexibility to fine-tune and adjust the embeddings for our specific needs.

Embeddings: Secret Sauce

There are 3 s values:
s_o, s_1, s_2
and one γ

The secret sauce of LM embedding is the ability to adapt the embeddings to each task as

$$\mathbf{LLE_t} = \gamma \sum_{j=0}^{L} s_j \boldsymbol{h}_t^j$$

$$\boldsymbol{h}_t^j = \{h_t^0, h_t^1, h_t^2\}$$
$$h_t^1 = \{h_t^{1l}, h_t^{1r}\}$$
$$h_t^2 = \{h_t^{2l}, h_t^{2r}\}$$

The scalar $\boldsymbol{\gamma}$ and the vector $\boldsymbol{s}$ are trained by the model. The value of $\boldsymbol{s}$ is softmax normalized.

The model can choose where to pay more attention, and how much importance to give to the LM embeddings.

Embeddings: Secret Sauce

The secret sauce of LM is the ability to adapt the embeddings to each task as

$$\mathbf{LLE}_{\mathrm{t}} = \gamma \sum_{j=0}^{L} s_j \boldsymbol{h}_t^j$$

The scalar $\boldsymbol{\gamma}$ and the vector $\boldsymbol{s}$ are trained by the model. The value of $\boldsymbol{s}$ is **softmax** normalized.

In our secret sauce, we can choose $\boldsymbol{\gamma}$ or how much sauce to put in our dish.

And $\boldsymbol{s}$ or the proportion of spices.

Embeddings: Secret Sauce

If our task benefits from the character level embeddings more than higher-level ones it can assign $s_0 = 1$ and not even look at the other parts of the LM.

$$\mathbf{LLE}_{\mathrm{t}} = \gamma(1 \cdot \boldsymbol{h}_t^0 + 0 \cdot \boldsymbol{h}_t^1 + 0 \cdot \boldsymbol{h}_t^2)$$
$$= \gamma \boldsymbol{h}_t^0$$

Embeddings: Secret Sauce

If another task benefits from the character higher level embeddings , it can assign $s_1 = 0.3$ and $s_2 = 0.7$.

$$\mathbf{LLE}_\mathrm{t} = \gamma(0 \cdot \boldsymbol{h}_t^0 + 0.3 \cdot \boldsymbol{h}_t^1 + 0.7 \cdot \boldsymbol{h}_t^2)$$
$$= \gamma(0.3 \cdot \boldsymbol{h}_t^1 + 0.7 \cdot \boldsymbol{h}_t^2)$$

Without looking at the character-level representation.

Embeddings: Secret Sauce

The secret sauce enables the model a higher degree of flexibility which improves almost all the models in any task.