Foundations for Natural Language Processing

Recurrent Neural Networks (RNNs): Classification and Language Modeling

Ivan Titov

(with graphics/materials from Elena Voita)

School of informatics

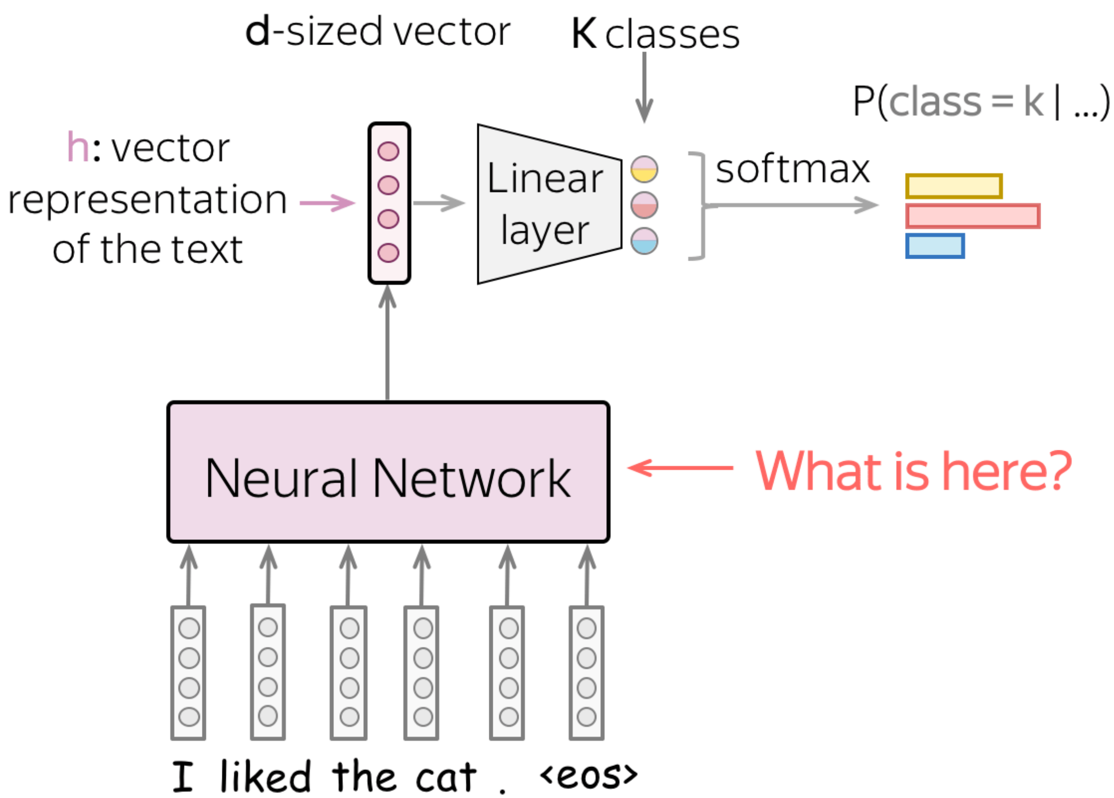
Summary for Neural Text Classification (so far)

We considered

- Generalization of logistic regression
- Easy to integrate embeddings, estimated on unlabeled text
- BoW models

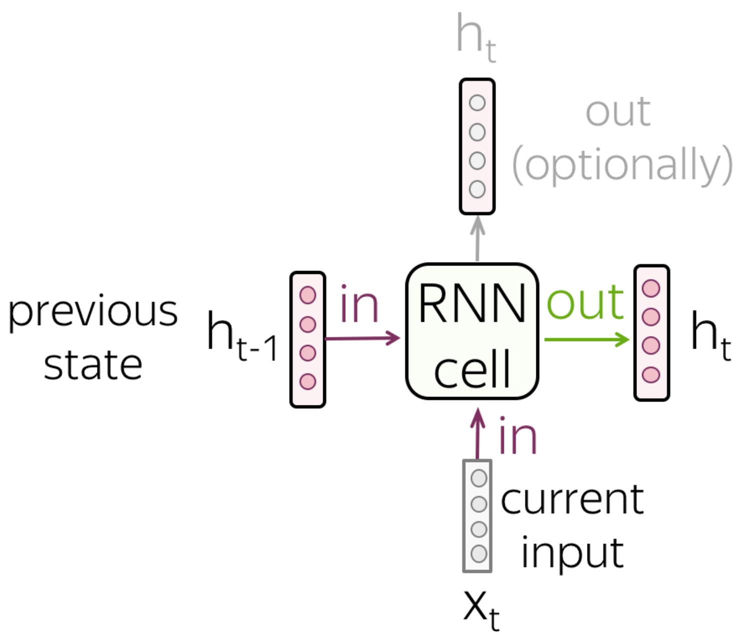
Now - Recurrent Neural Networks (RNNs)
First for text classification, and then for language modeling / generation

Recap: NNs for text classification

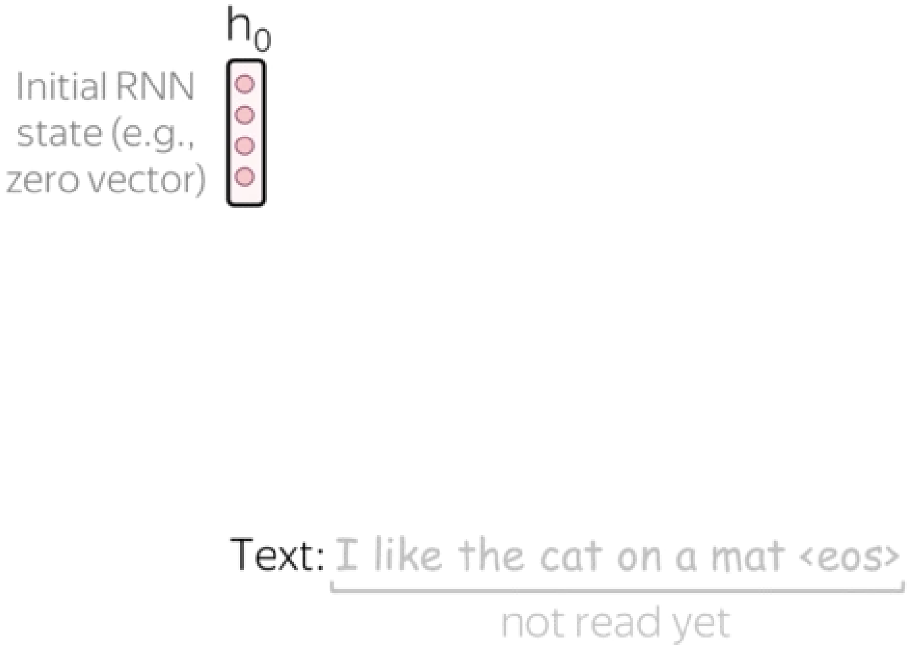


We will finish up with classification today by introducing a class of models which will be particularly useful for text generation (RNNs)

Recurrent Neural Networks: RNN cell



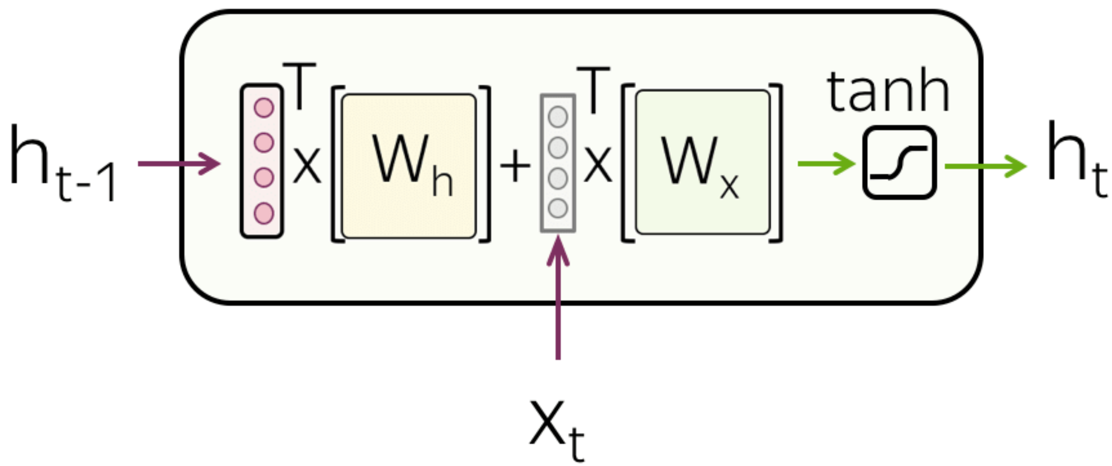
RNN reads a sequence of tokens



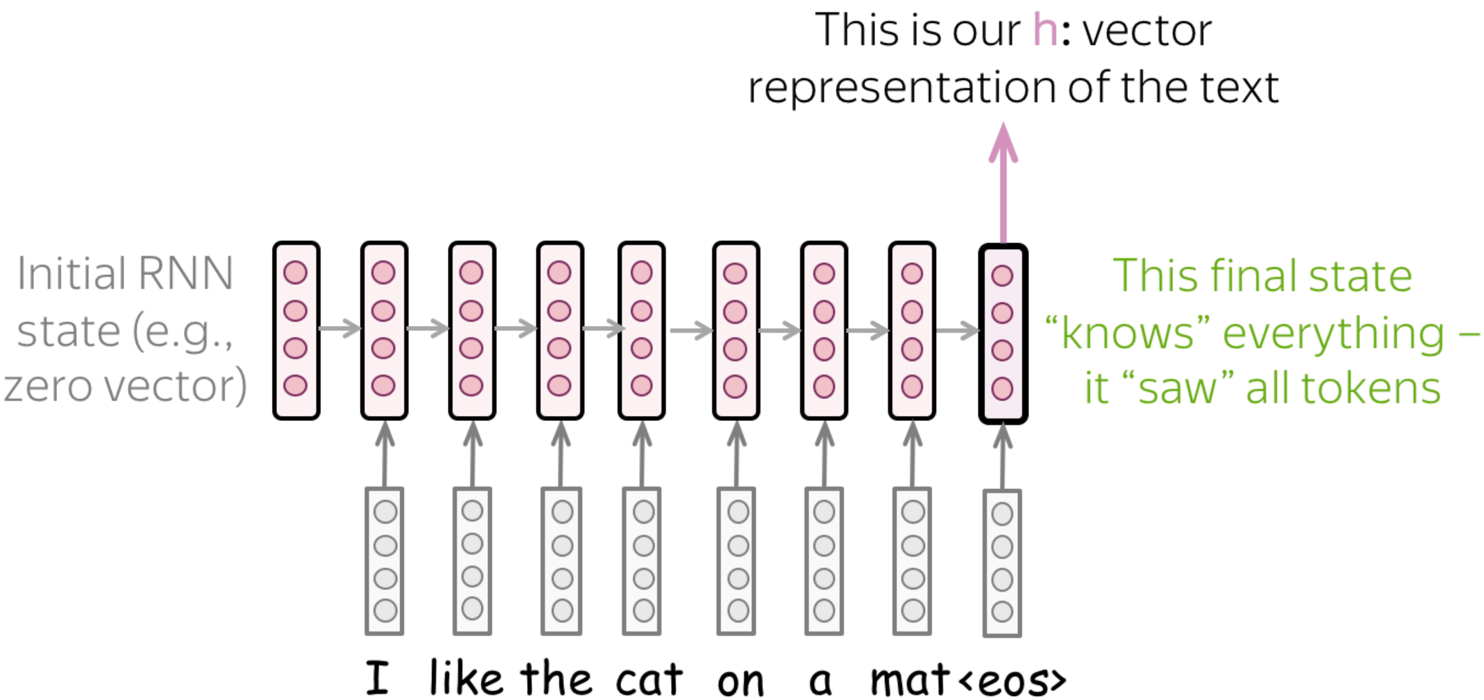
(video, not visible in pdf)

Vanilla RNN

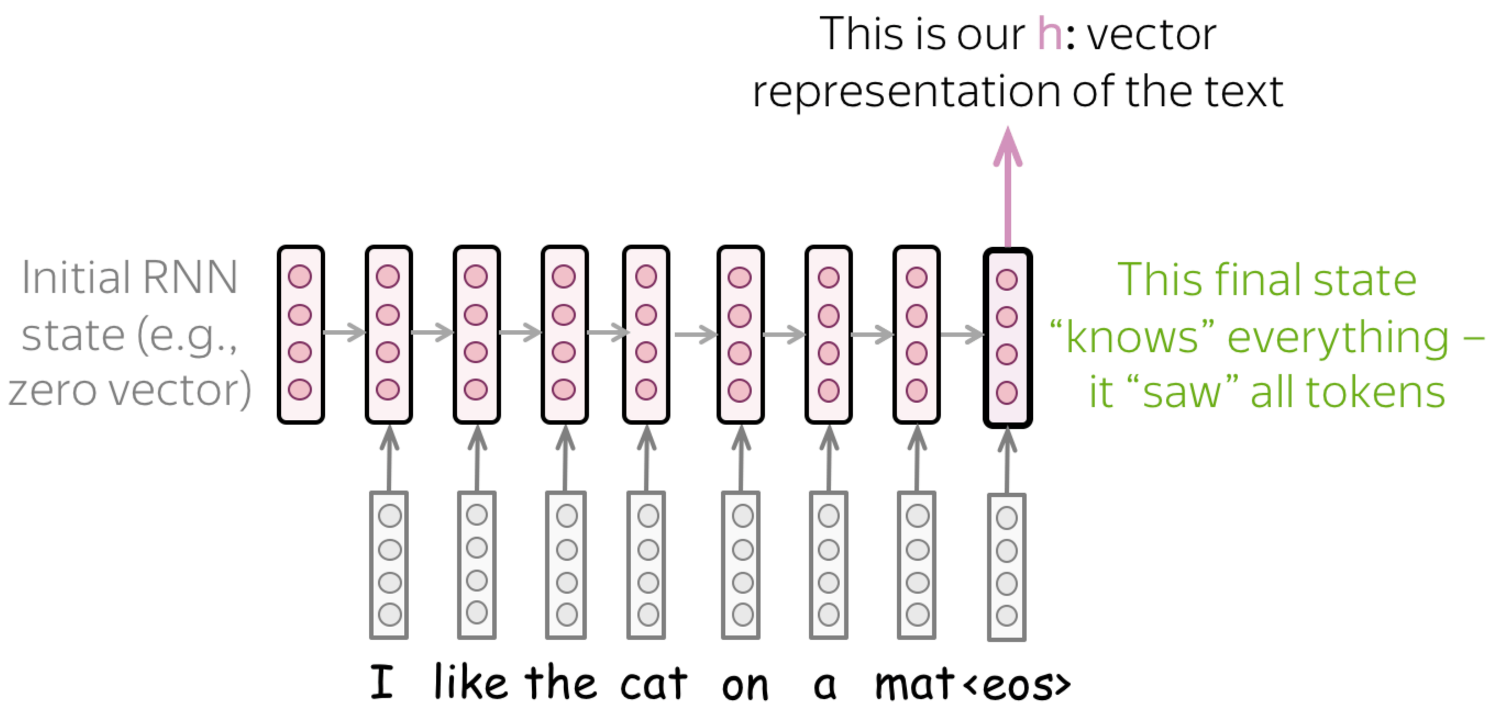
$$h_t = \tanh(h_{t-1} W_h + x_t W_x)$$



Text representation with RNN



Text representation with RNN

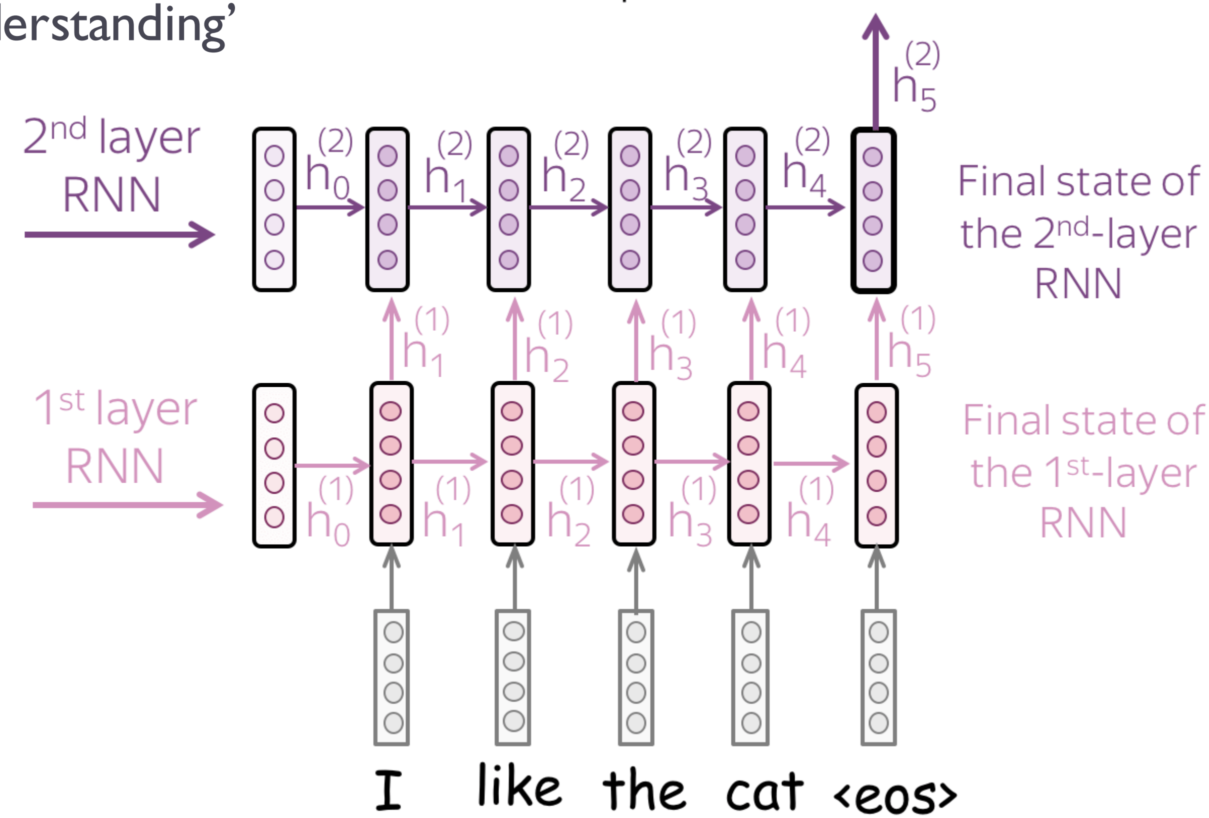


The architecture may not be expressive enough, how can we make the model more powerful?

Text representation with multi-layer RNN

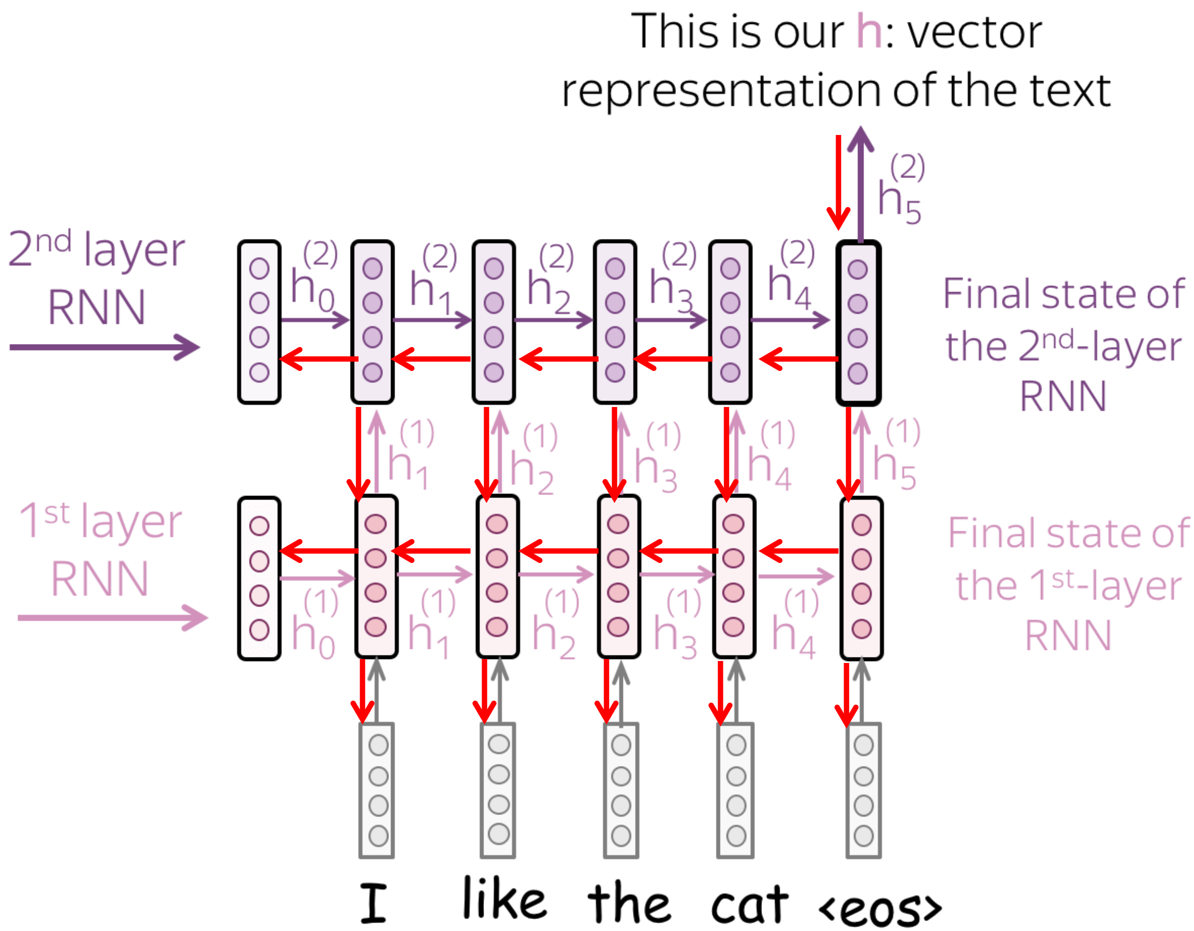
Better at capturing different levels of abstraction of representations; crucial for hard tasks requiring 'deep understanding'

This is our h: vector representation of the text



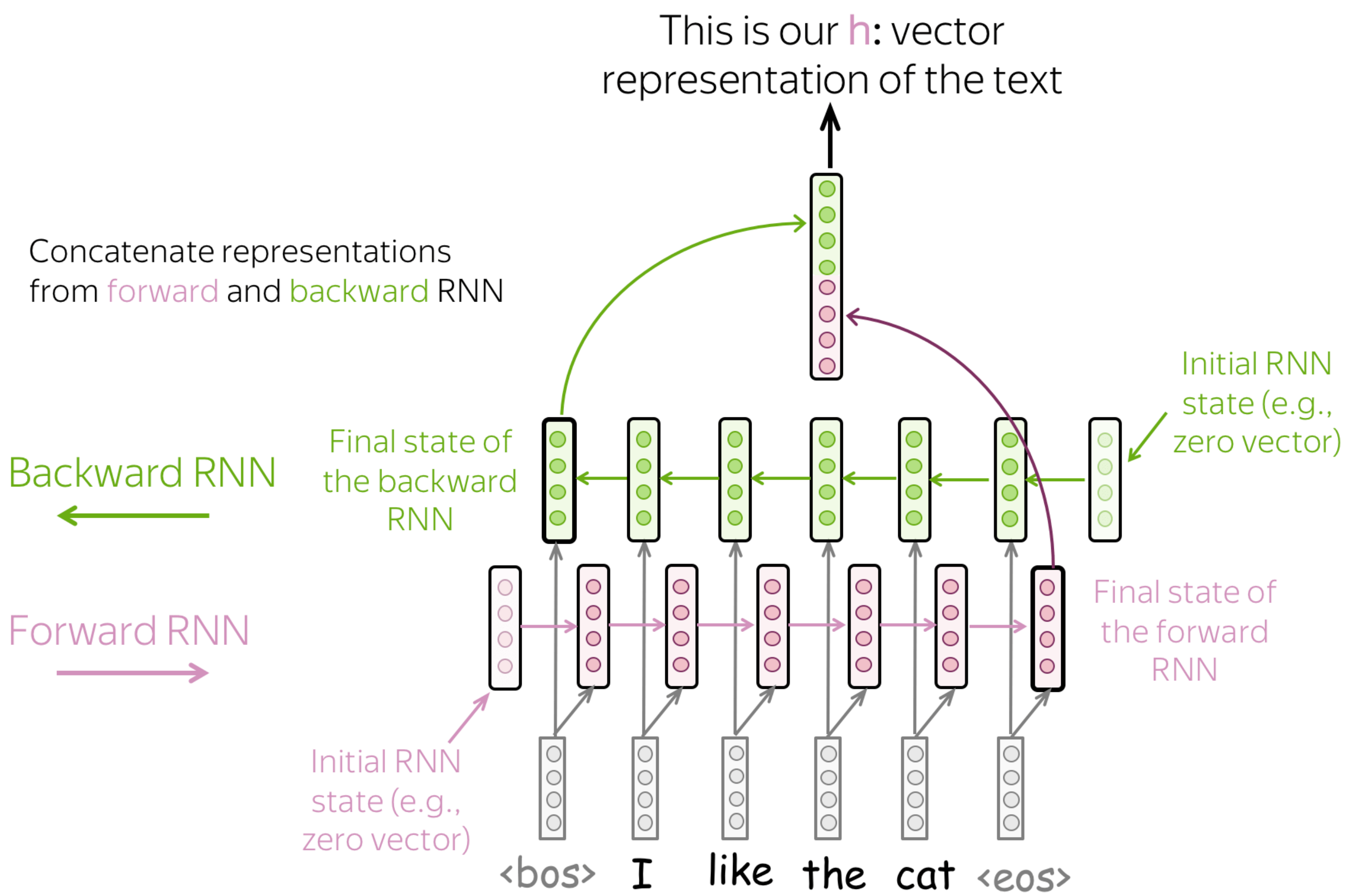
Is there a problem with passing only the last state as *h*?

Text representation with multi-layer RNN



Models learn by backpropagation, it takes many steps to propagate to the very beginning of the sentence; the model will not learn to reliably encode early parts of the sentence (long context), how can we address it?

Text representation with bidirectional RNN

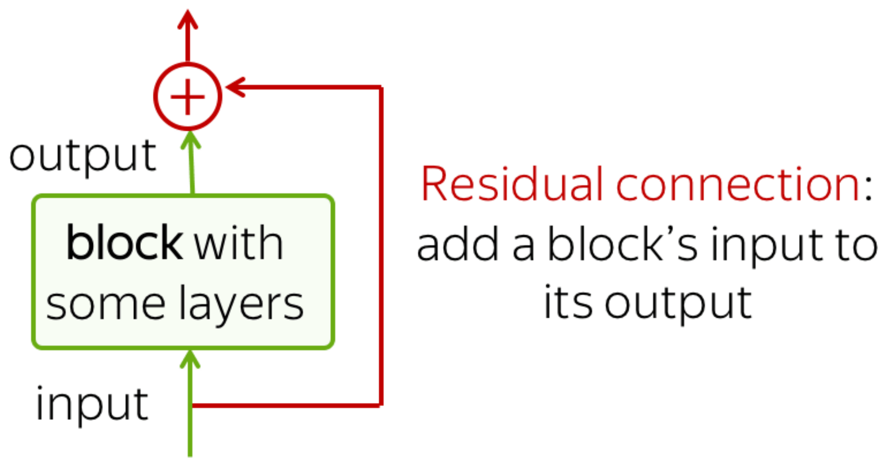


Stacking many layers

Unfortunately, when stacking a lot of layers, you can have a problem with propagating gradients from top to bottom through a deep network.

To mitigate, this we use Residual or Highway connections

Residual connections:

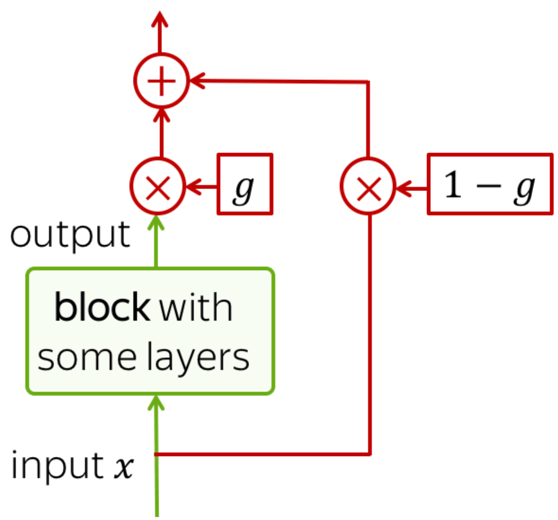


Stacking many layers

Unfortunately, when stacking a lot of layers, you can have a problem also with propagating gradients from top to bottom through a deep network (we will analyze it for RNNs).

To mitigate, this we use Residual or Highway connections

Highway connections:



Highway connection: gated sum of a block's input and output

$$g = \sigma(Wx + b)$$

Gate: because of σ, its values are in (0, 1)

Logistic sigmoid:

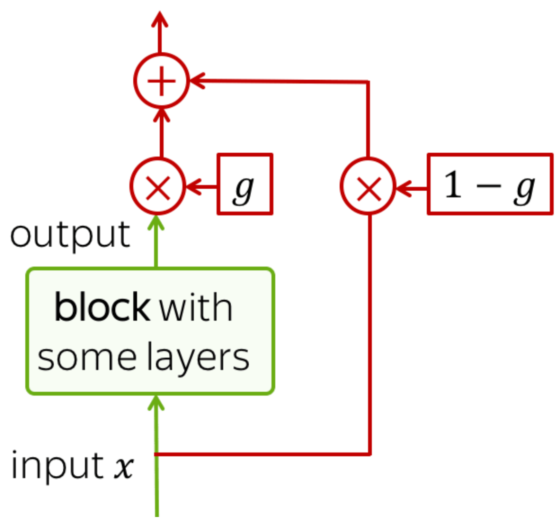
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Stacking many layers

Unfortunately, when stacking a lot of layers, you can have a problem also with propagating gradients from top to bottom through a deep network.

To mitigate, this we use Residual or Highway connections

Highway connections:



Highway connection: gated sum of a block's input and output

$$g = \sigma(Wx + b)$$

Gate: because of σ, its values are in (0, 1)

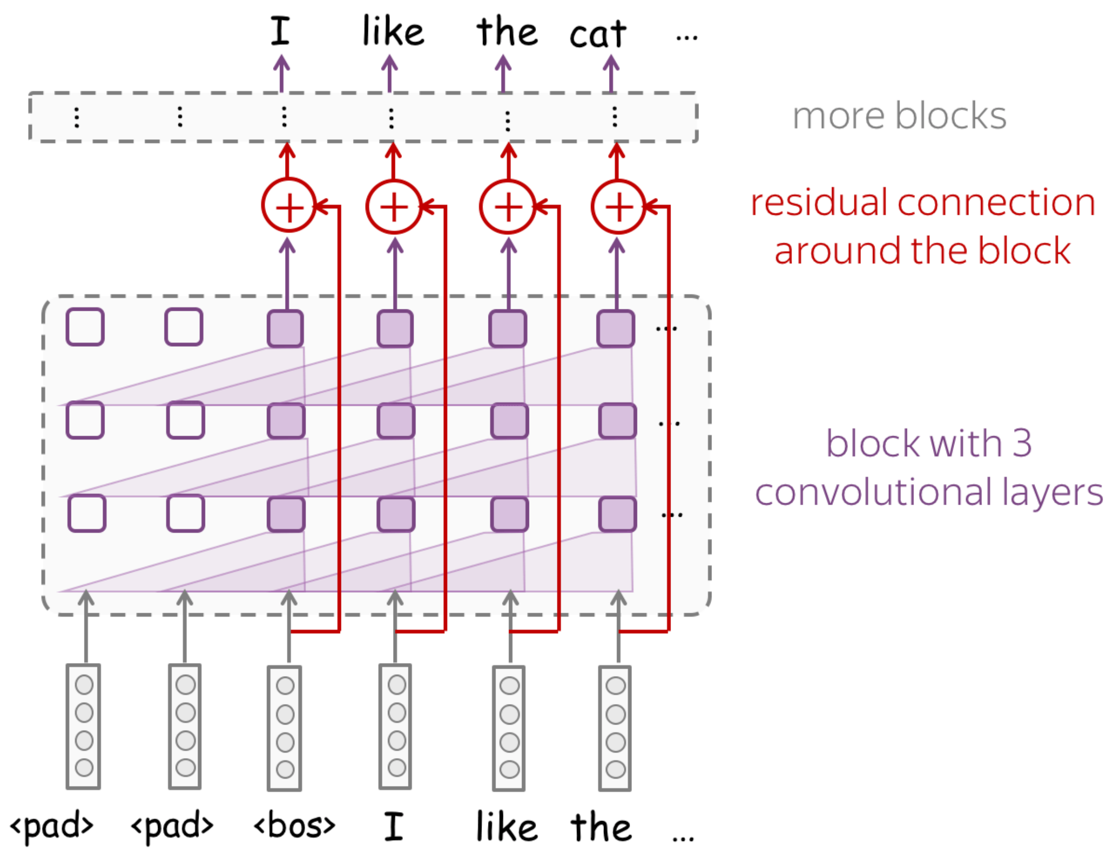
Logistic sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

If you apply the idea of highway connections in horizontal direction (along sequence), you will obtain popular (stronger) RNN varieties, such as LSTMs

Multilayer models with residual connections

This is an example of multilayer Convolutional Neural Network but it would look the same with multilayer / bi-directional RNNs



Having residual connection is necessary for training large-language models typically powered with Transformers (will talk about them in a couple of lectures)

Summary on classification

We have not talked about CNNs in FNLP (due to the storm) but you may have seen (or will see them in other classes); they are / were very popular in computer vision

- Naïve Bayes

  very fast to train, robust, makes overly strong assumptions

- Logistic regression

  still easy to train, requires strong features, fewer assumptions

- Feedfordward neural network (neural BoW)

  still easy to train, few assumptions, breaks sentence into words

- ~~Convolutional Neural Networks~~

  ~~still easy to train, fewer assumptions, but still (~) breaks sentence into ngrams~~

- Recurrent Neural Networks

  does not break a sentence into pieces, but the information is carried within the sentence through a vector

Language modeling

Recall, the language model assigns the probability to a sequence of words $y_1, y_2, \ldots, y_n$, relying on the chain rule:

$$P(y_1, y_2, \ldots, y_n) = P(y_1) \cdot P(y_2|y_1) \cdot P(y_3|y_1, y_2) \cdot \cdots \cdot P(y_n|y_1, \ldots, y_{n-1})$$

$$= \prod_{t=1}^{n} P(y_t|y_{<t})$$

How do we compute $P(y_t|y_{<t})$?

Ngram models made independence assumptions, basically breaking the sequence into smaller subsequences for estimation

Samples from ngram models: any issues?

```
hahn , director of the christian " love and
compassion " was designed as a result of any form ,
in the transaction is active in the stuva grill .
_eos_
```

```
pupils from eastern europe , africa , saudi arabia
' s church , yearn for such an open structure of
tables several times on monday 14 september 2003 ,
his flesh when i was curious to know and also to
find what they are constructed with a speeding
arrow . _eos_
```

Samples from ngram models: any issues?

```
hahn , director of the christian " love and
compassion " was designed as a result of any form ,
in the transaction is active in the stuva grill .
_eos_
```

```
pupils from eastern europe , africa , saudi arabia
' s church , yearn for such an open structure of
tables several times on monday 14 september 2003 ,
his flesh when i was curious to know and also to
find what they are constructed with a speeding
arrow . _eos_
```

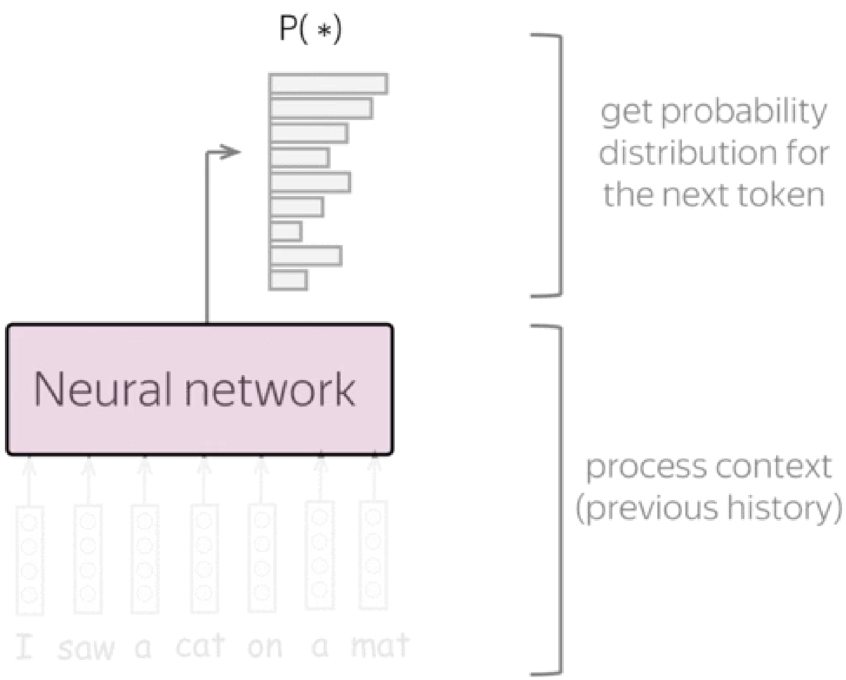
Ngram models clearly struggle with capturing longer context

NNs (e.g., RNNs) will let us model text without making explicit independence assumptions

Intuition

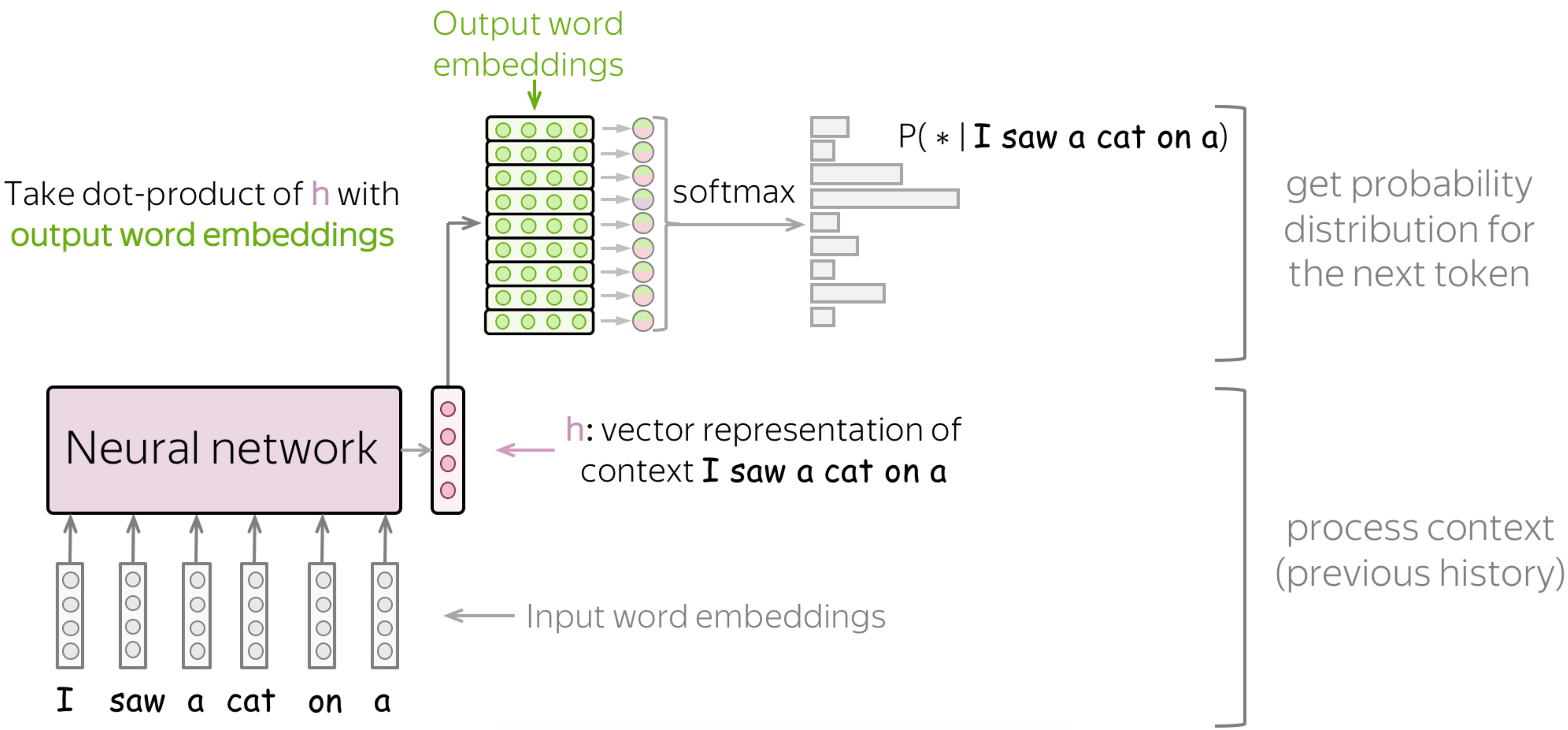
Neural language models has to:

1. *Produce a representation of the prefix*
2. *Generate a probability distribution over the next token*



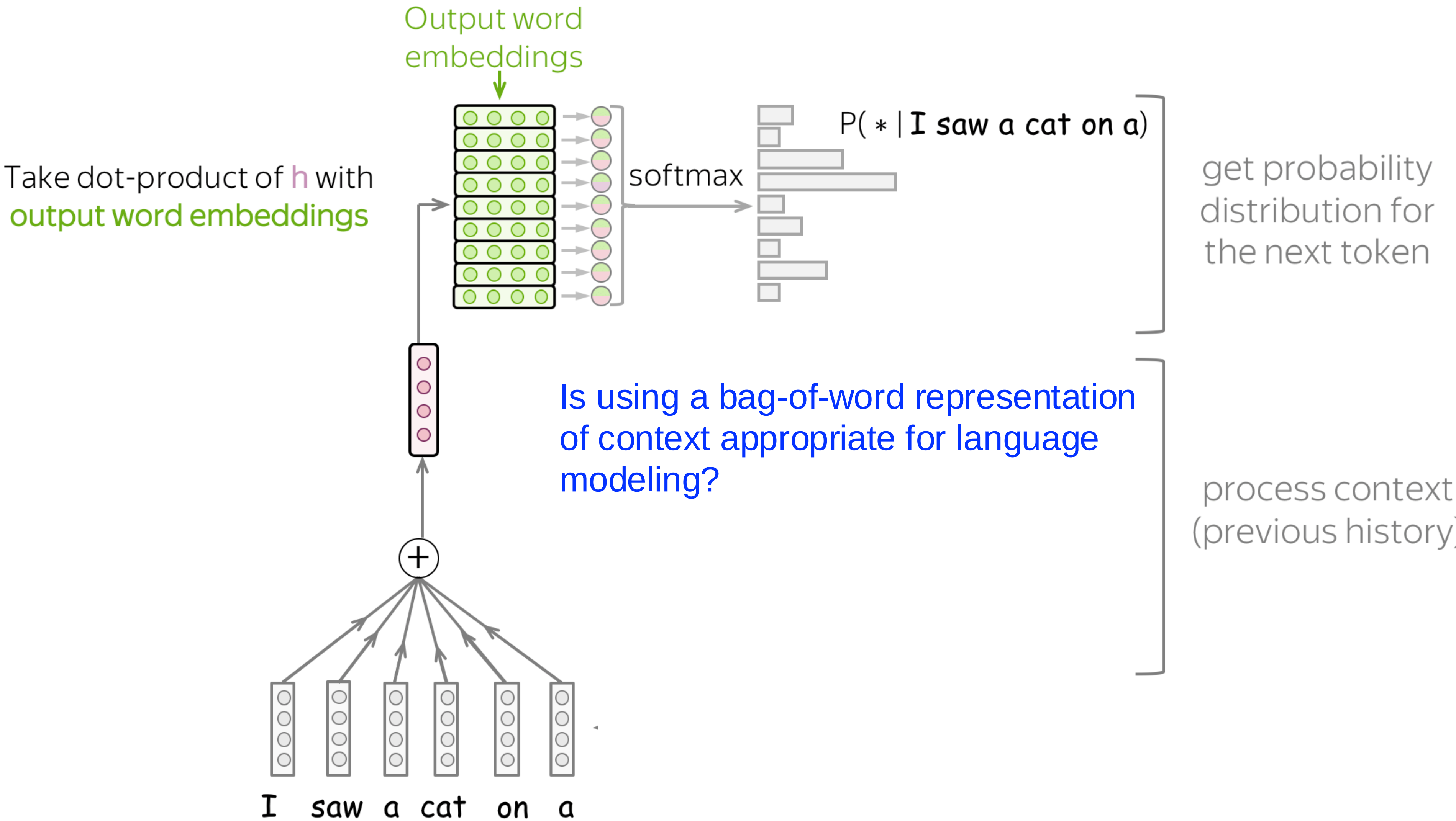
Predicting a word, given a prefix, is just a classification problem!

High-level intuition for a language model



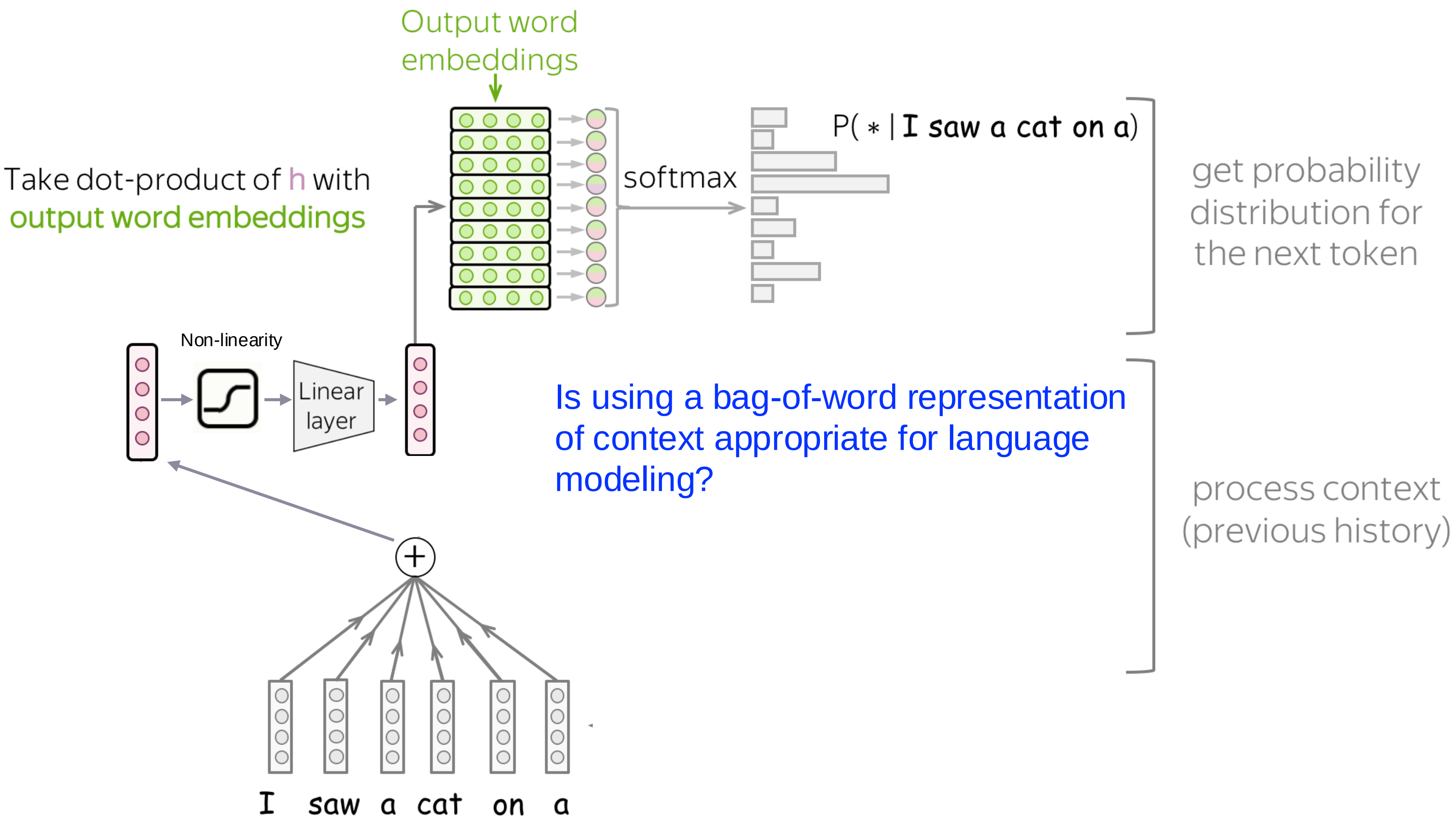
$$p(y_t|y_{<t}) = \frac{exp(h_t^T e_{y_t})}{\sum_{w \in V} exp(h_t^T e_w)}$$

What neural models to use for language modeling?



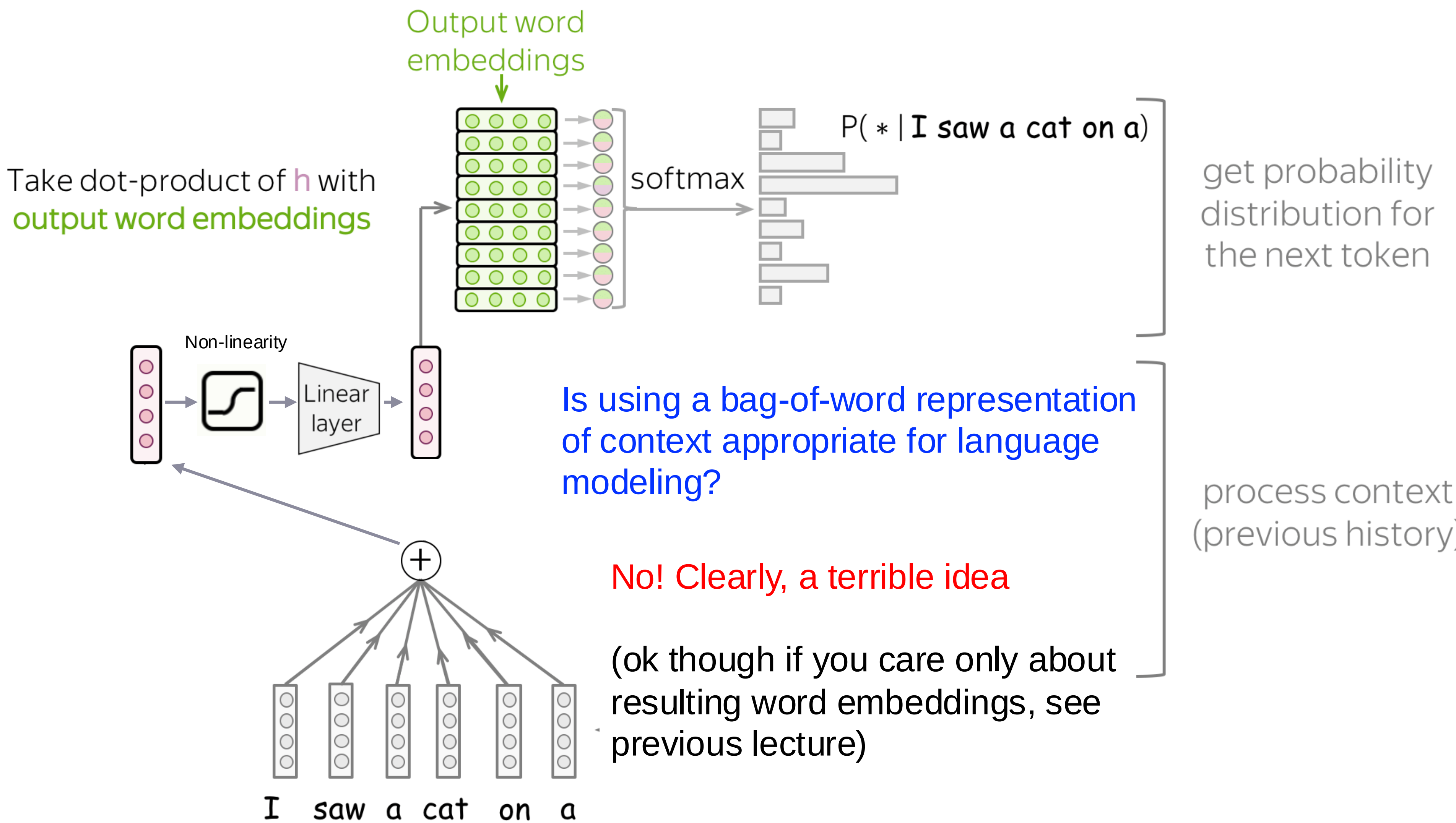
Is using a bag-of-word representation of context appropriate for language modeling?

What neural models to use for language modeling?



Is using a bag-of-word representation of context appropriate for language modeling?

What neural models to use for language modeling?

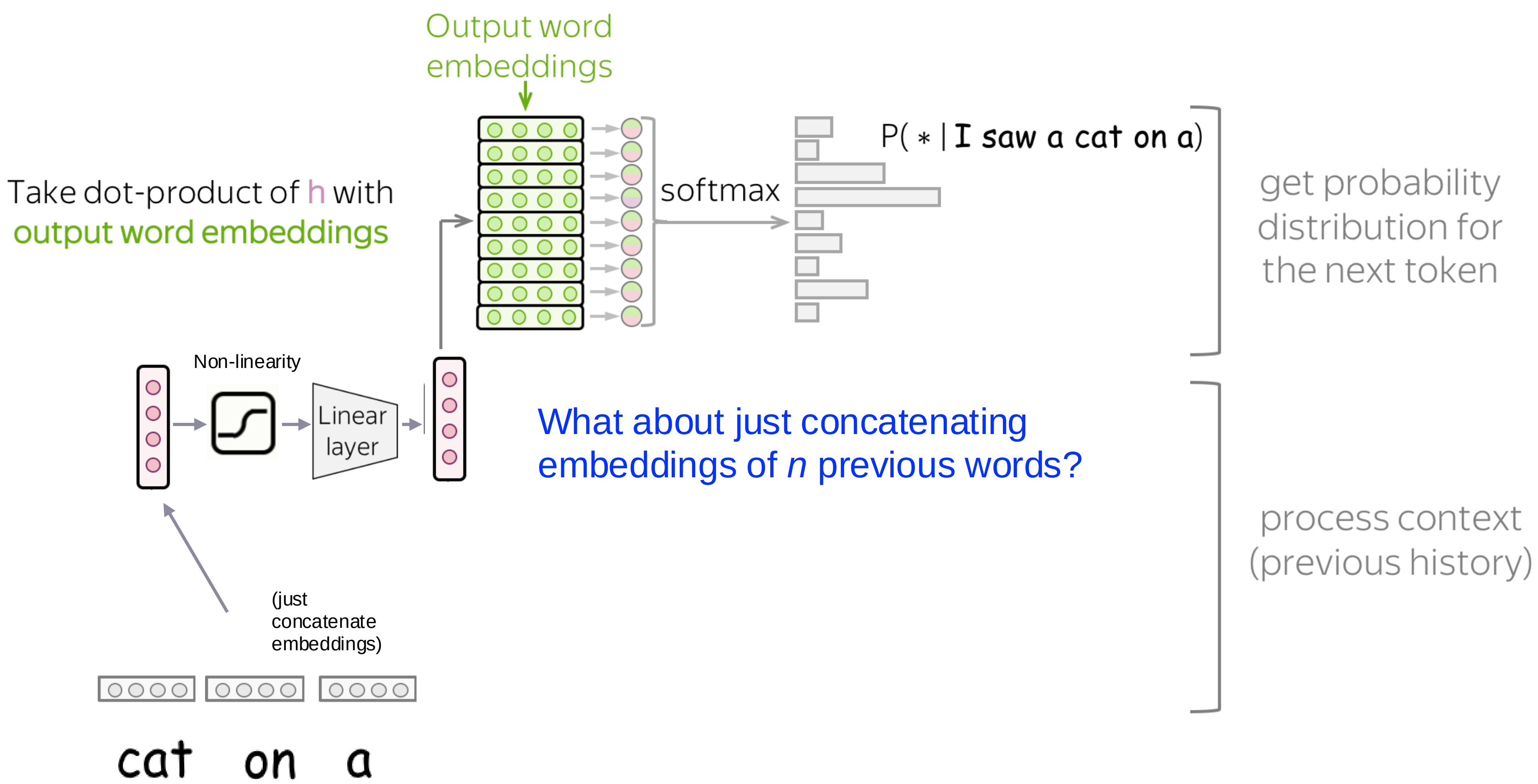


Is using a bag-of-word representation of context appropriate for language modeling?

No! Clearly, a terrible idea

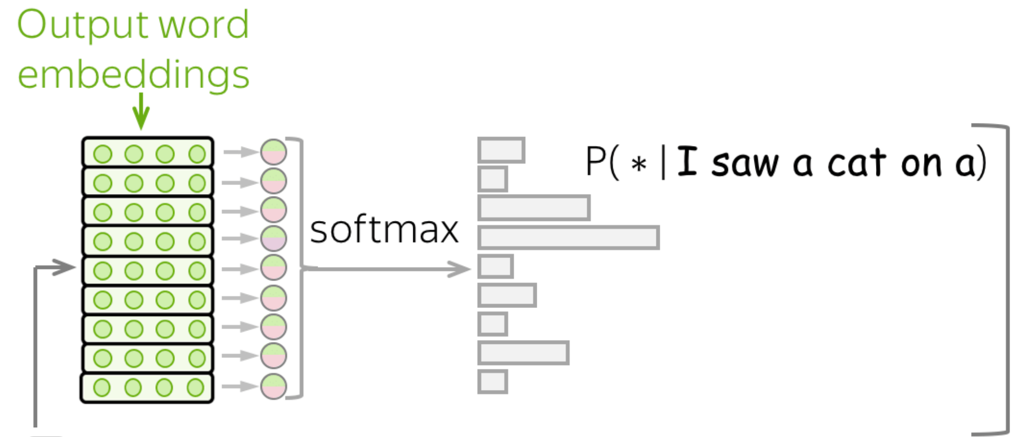
(ok though if you care only about resulting word embeddings, see previous lecture)

What neural models to use for language modeling?



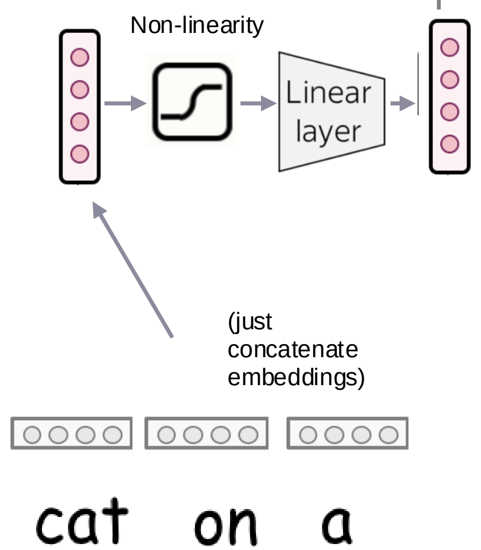
What about just concatenating embeddings of *n* previous words?

What neural models to use for language modeling?



Take dot-product of h with output word embeddings

get probability distribution for the next token



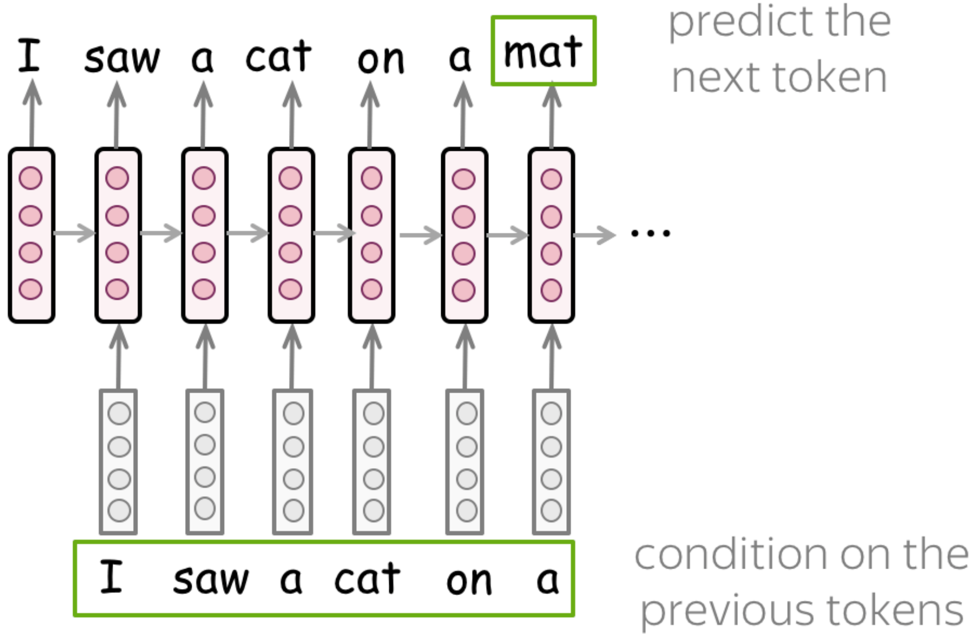
What about just concatenating embeddings of *n* previous words?

process context (previous history)

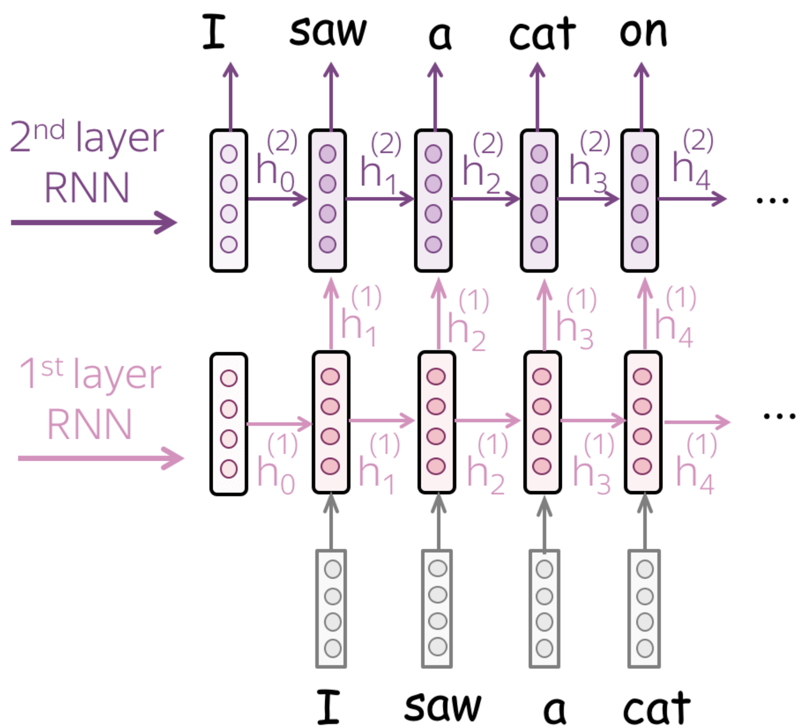
It a classic neural language models (aka feedforward neural LM, aka neural ngram LM)

As with ‘classic’ ngram LMs, it make a conditional independence assumption and cannot capture long context. RNN model addresses it!

RNN language model



Multi-layer RNN language model



Training the language model

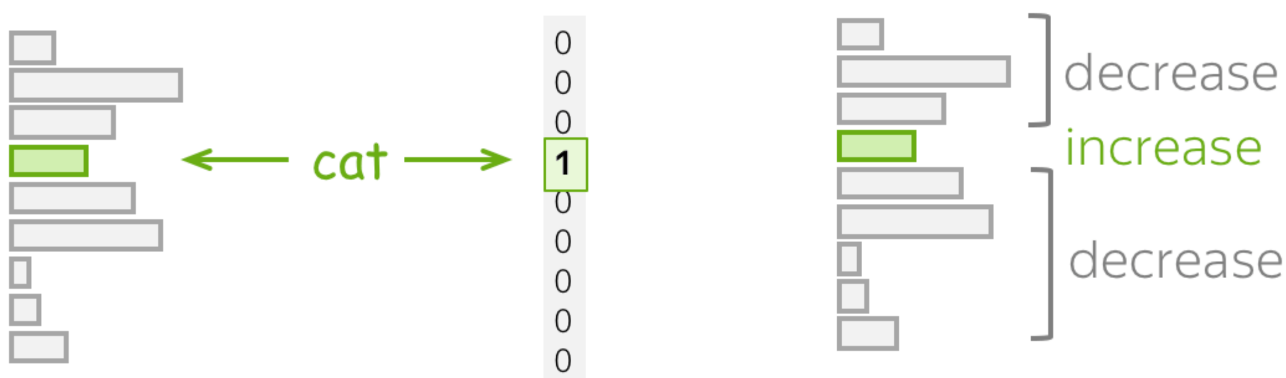
Training is done in a very much the same way as we train a classifier!

$$Loss = -\log(p(y_t|y_{<t}))$$

we want the model
to predict this

Training example: **I saw a** cat on a mat <eos>

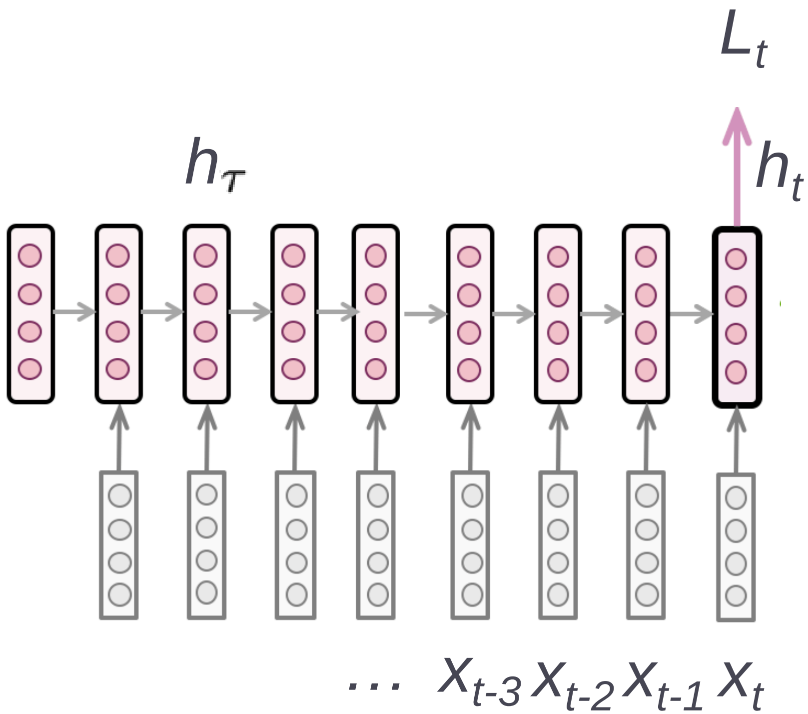
Model prediction: p(* | **I saw a**)   Target   Loss = -log (p(cat)) → min



Training for one sentence with RNN LM

(video, not visible in pdf)

Learning RNNs: vanishing gradient problem



RNN model is trained by back-propagating through the computation

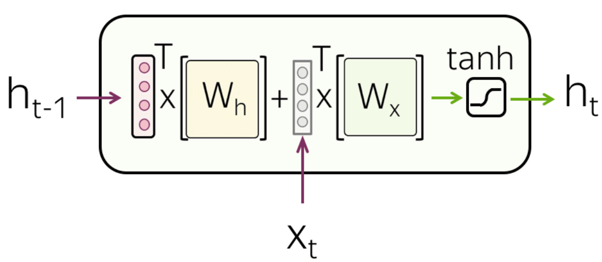
$$\frac{dL_t}{dW_h} = \sum_{\tau=0}^{t} \frac{dL_t}{dh_t} \cdot \frac{dh_t}{dh_\tau} \cdot \frac{dh_\tau}{dW_h}$$

W_h is *t* times in the computation, as it is used on every step of RNN

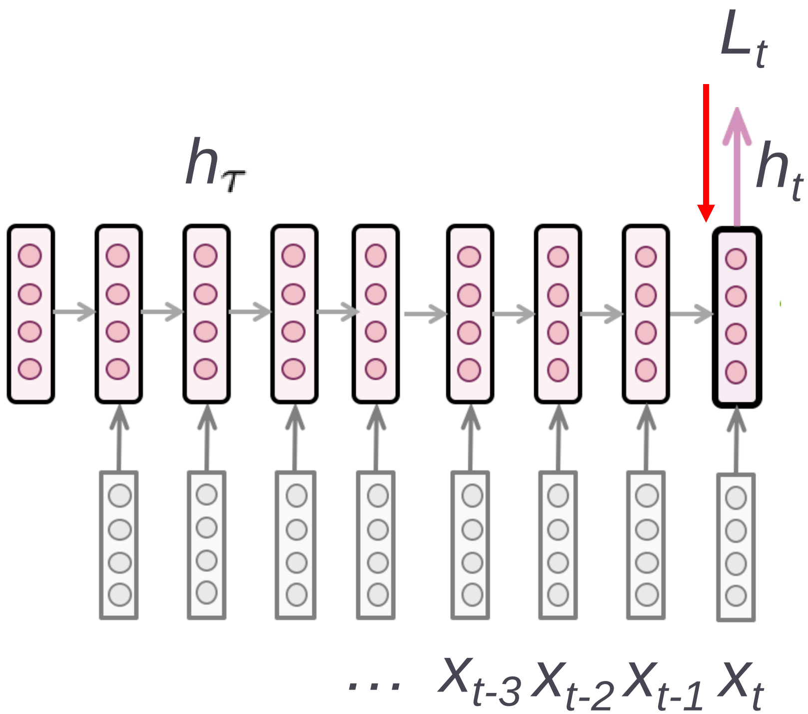
Each term in the sum corresponds to the “use” of W_h at step τ

For the model to carry information across distances, we cannot ignore terms with $\tau \ll t$

$h_t = \tanh(h_{t-1} W_h + x_t W_x)$



Learning RNNs: vanishing gradient problem

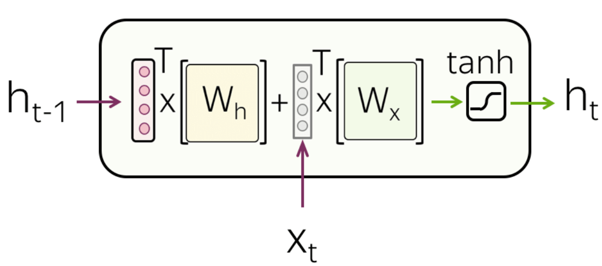


RNN model is trained by back-propagating through the computation

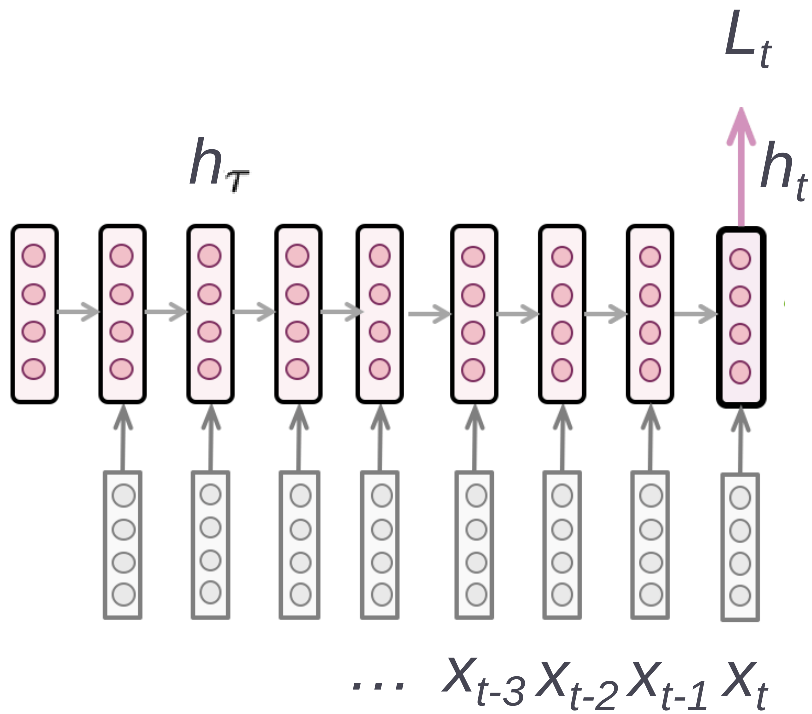
$$\frac{dL_t}{dW_h} = \sum_{\tau=0}^{t} \boxed{\frac{dL_t}{dh_t}} \cdot \frac{dh_t}{dh_\tau} \cdot \frac{dh_\tau}{dW_h}$$

This the same gradient as in logistic regression, not a problem

$h_t = \tanh(h_{t-1}W_h + x_t W_x)$



Learning RNNs: vanishing gradient problem

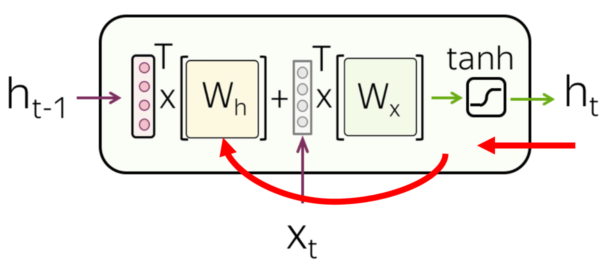


RNN model is trained by back-propagating through the computation

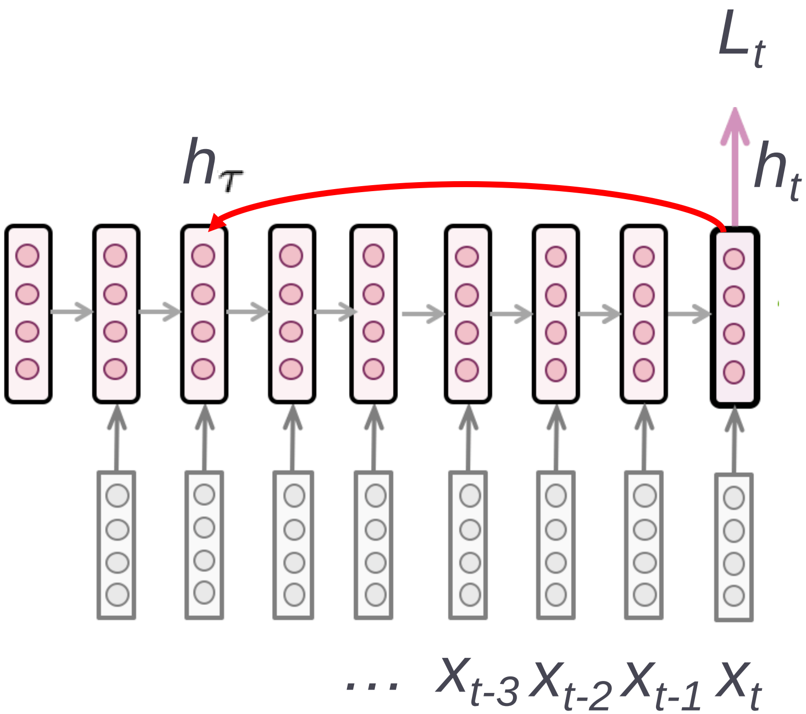
$$\frac{dL_t}{dW_h} = \sum_{\tau=0}^{t} \frac{dL_t}{dh_t} \cdot \frac{dh_t}{dh_\tau} \cdot \boxed{\frac{dh_\tau}{dW_h}}$$

This is just a back-prop through one RNN unit

$$h_t = \tanh(h_{t-1}W_h + x_t W_x)$$



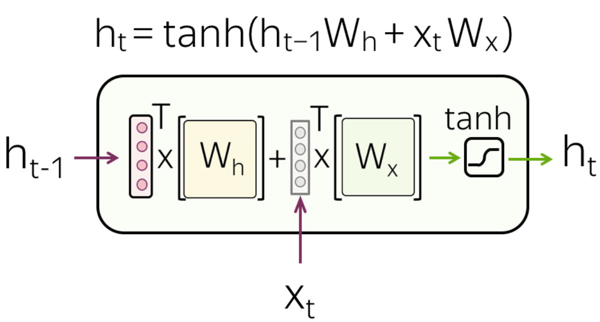
Learning RNNs: vanishing gradient problem



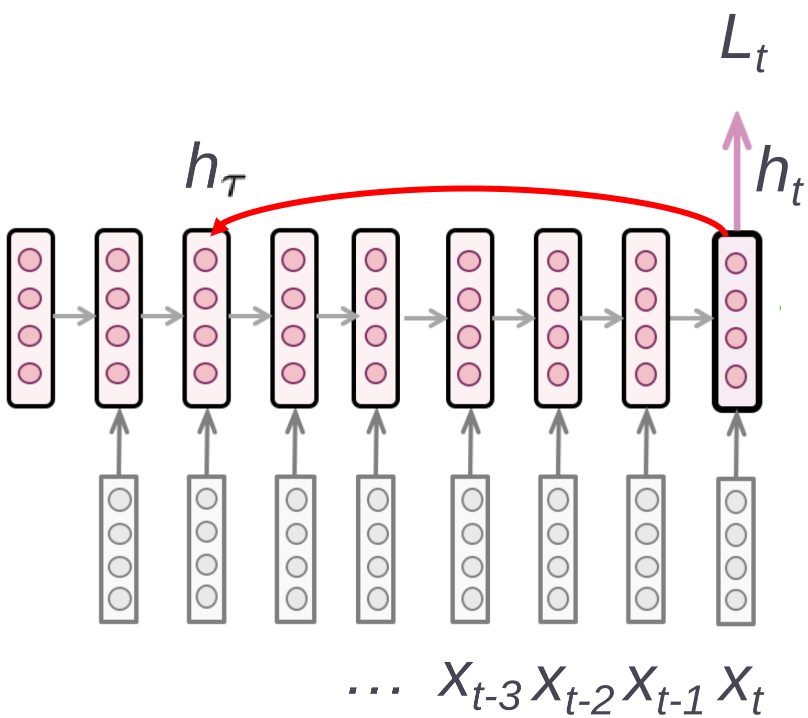
RNN model is trained by back-propagating through the computation

$$\frac{dL_t}{dW_h} = \sum_{\tau=0}^{t} \frac{dL_t}{dh_t} \cdot \boxed{\frac{dh_t}{dh_\tau}} \cdot \frac{dh_\tau}{dW_h}$$

This is the term which causes the problem



Learning RNNs: vanishing gradient problem



RNN model is trained by back-propagating through the computation

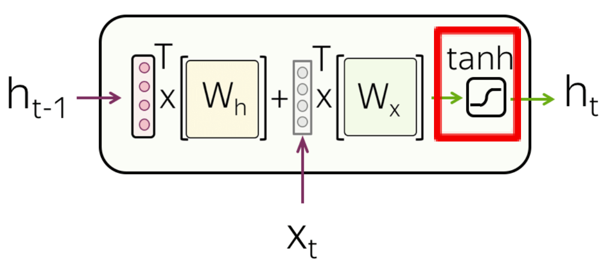
$$\frac{dL_t}{dW_h} = \sum_{\tau=0}^{t} \frac{dL_t}{dh_t} \cdot \boxed{\frac{dh_t}{dh_\tau}} \cdot \frac{dh_\tau}{dW_h}$$

For RNN with tanh activations:

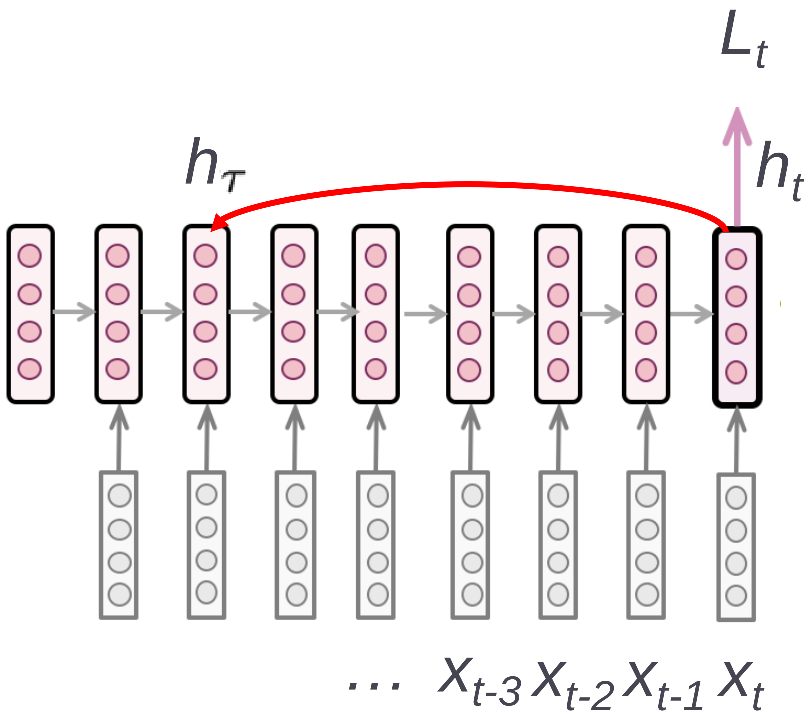
$$\frac{dh_t}{dh_\tau} = \prod_{k=\tau}^{t-1} \boxed{\mathrm{diag}(1 - h_{k+1}^2)} W_h$$

This just a gradient of the *tanh* activation function; let's not worry about it

$h_t = \tanh(h_{t-1} W_h + x_t W_x)$



Learning RNNs: vanishing gradient problem



RNN model is trained by back-propagating through the computation

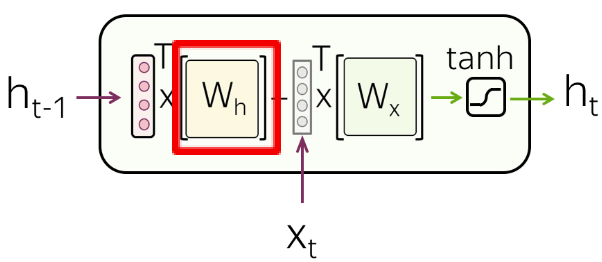
$$\frac{dL_t}{dW_h} = \sum_{\tau=0}^{t} \frac{dL_t}{dh_t} \cdot \boxed{\frac{dh_t}{dh_\tau}} \cdot \frac{dh_\tau}{dW_h}$$

For RNN with tanh activations:

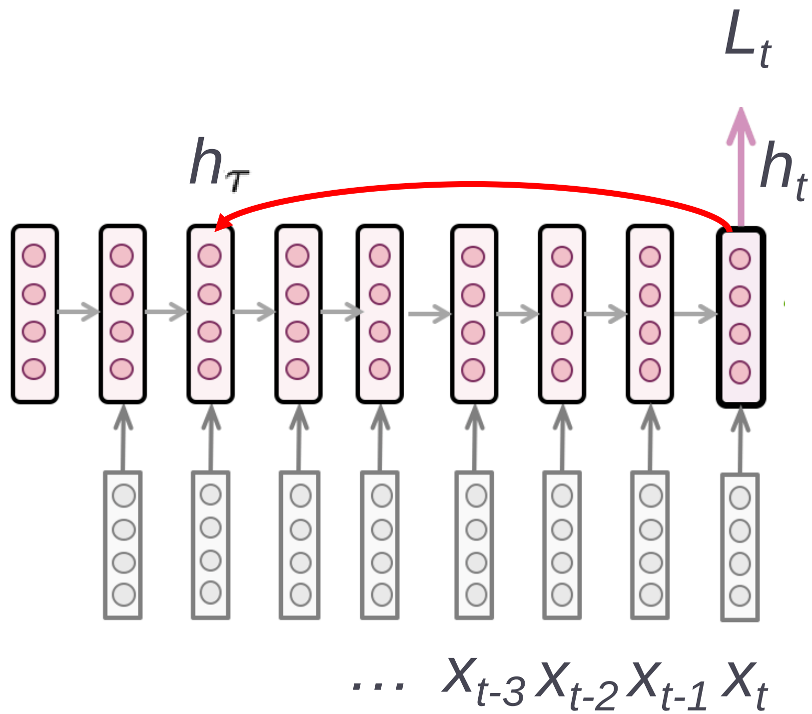
$$\frac{dh_t}{dh_\tau} = \prod_{k=\tau}^{t-1} \mathrm{diag}(1 - h_{k+1}^2)\boxed{W_h}$$

This the gradient of the linear operation, and it is the troublemaker but how come?

$h_t = \tanh(h_{t-1}W_h + x_t W_x)$



Learning RNNs: vanishing gradient problem

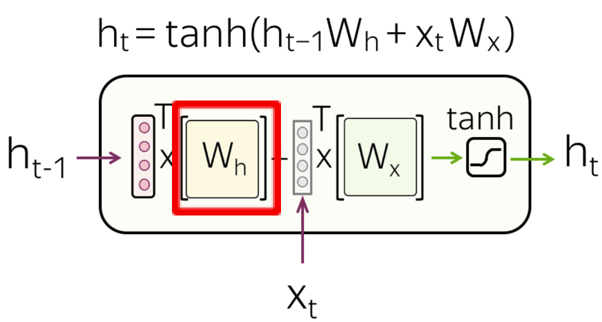


RNN model is trained by back-propagating through the computation

$$\frac{dL_t}{dW_h} = \sum_{\tau=0}^{t} \frac{dL_t}{dh_t} \cdot \boxed{\frac{dh_t}{dh_\tau}} \cdot \frac{dh_\tau}{dW_h}$$

For RNN with tanh activations:

$$\frac{dh_t}{dh_\tau} = \left(\prod_{k=\tau}^{t-1} \mathrm{diag}(1-h_{k+1}^2)\right) W_h^{\boxed{(t-\tau)}}$$

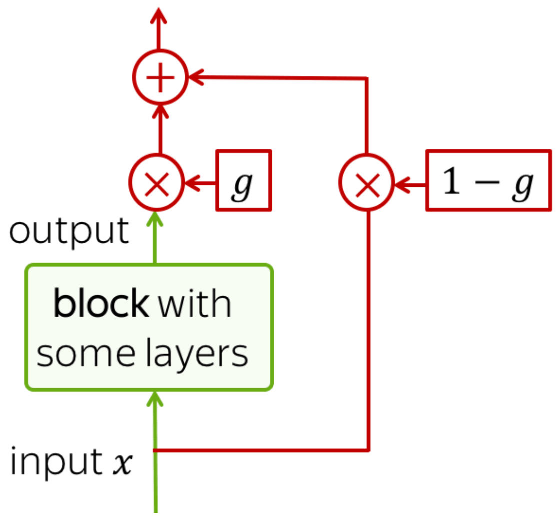


This repetitive multiplication in BP results either in gradient explosion (which can be addressed) or vanishing (which is not easy to address)

The reason for the problem is the repetitive multiplication by W_h on the forward pass

LSTMs (and GRUs)

Recall, highway connections:



<u>Intuitively</u>, LSTMs and GRUs add highway connections in the temporal ('horizontal') dimension; now the information can also be propagated through weighted edition, so $\frac{dh_t}{dh_\tau}$ won't vanish (go to zero)

RNNs vs Ngram models

Ngram language model

- relies on a short prefix, to get a distribution over next tokens
- explicit independence assumption (can't use context outside of the ngram window)
- smoothing is necessary

Feedforward neural language models (and CNN language models)

- again, relies on a short prefix, to get a distribution over next tokens
- again, explicit independence assumption
- but smoothing is not necessary

RNN language model

- 'compresses' the past into a state, used to compute the distribution over next tokens
- no independence assumptions; the gradient descent learns to compress the past
- all the information is carried through hidden states (but hard to carry it across long distances) [LSTMs/GRUs make it easier to learn to carry but the bottleneck is still there]