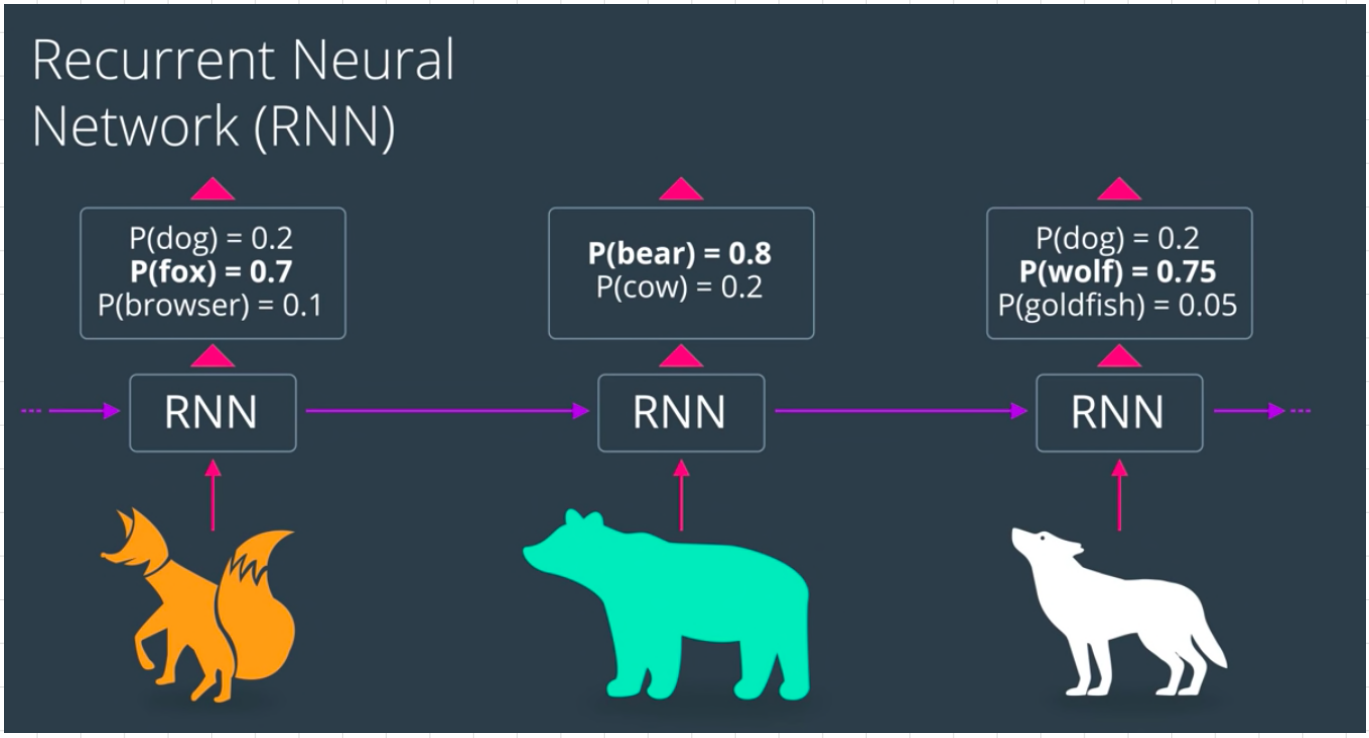
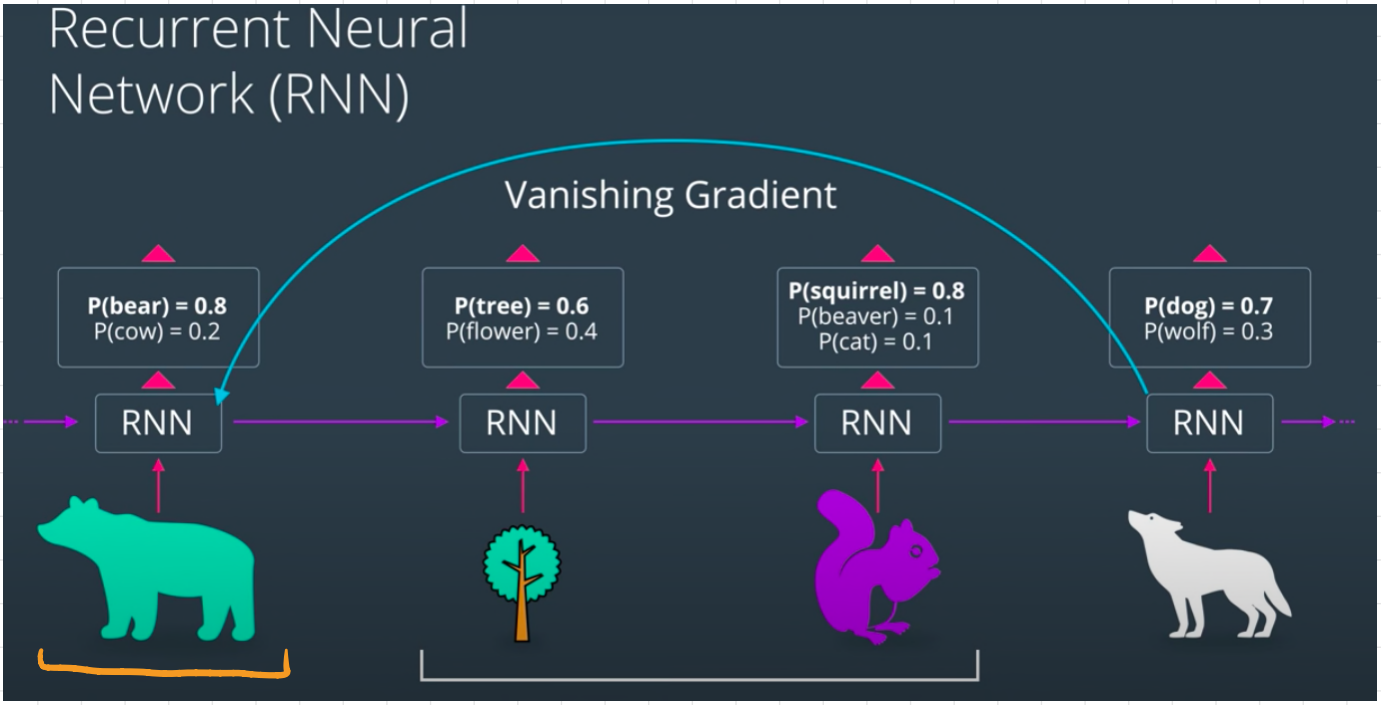
LSTM

LSTM vs. RNN

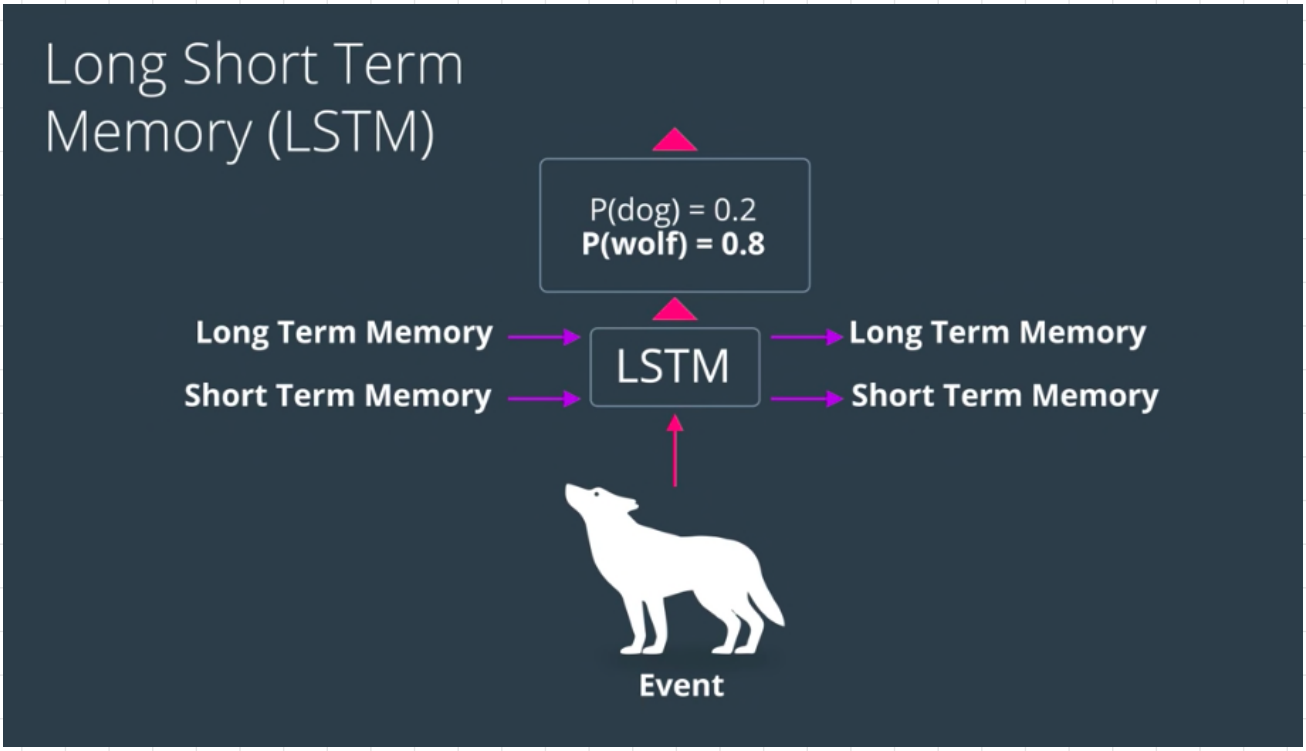


Use output from previous as an input to hint

However, info can come from all the way back, and memory is short-term.

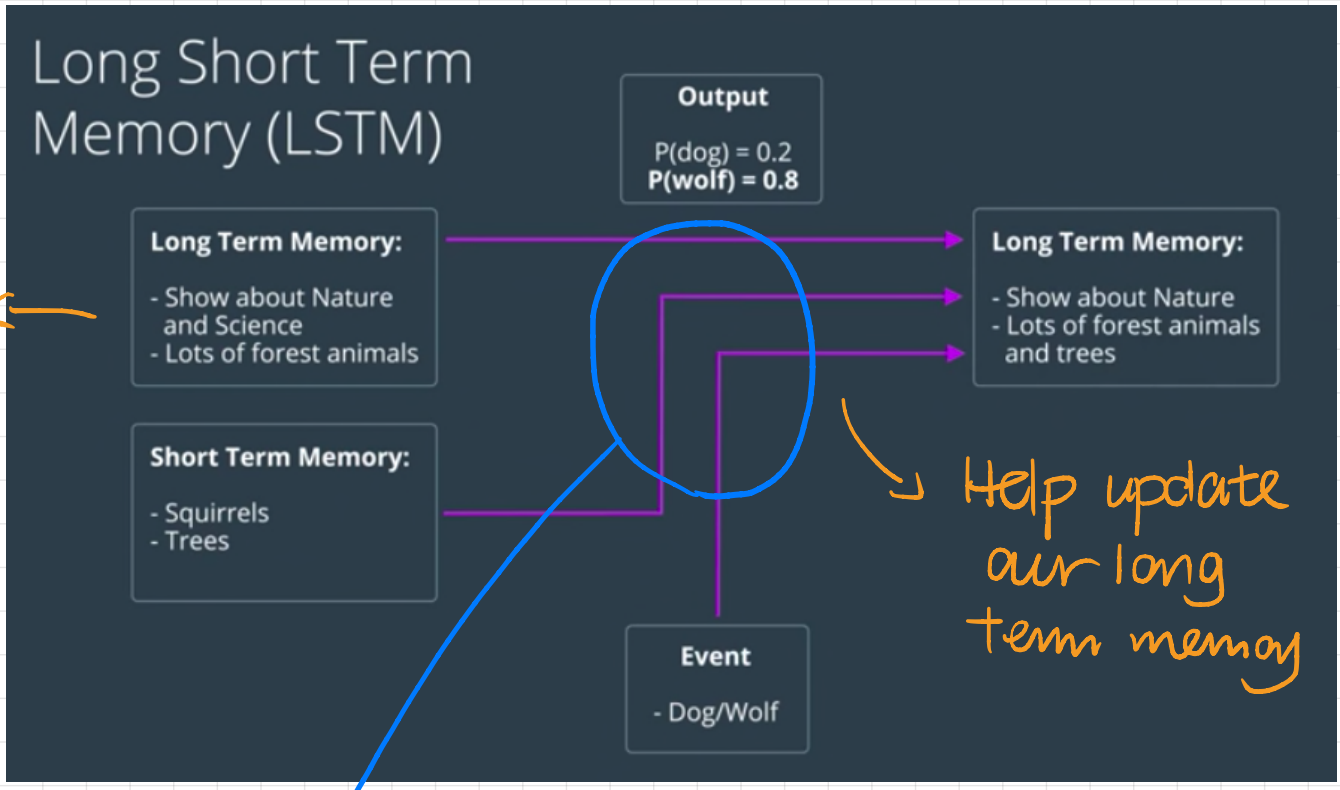


Info we care



LSTM preserves long term memory!

LSTM Basics

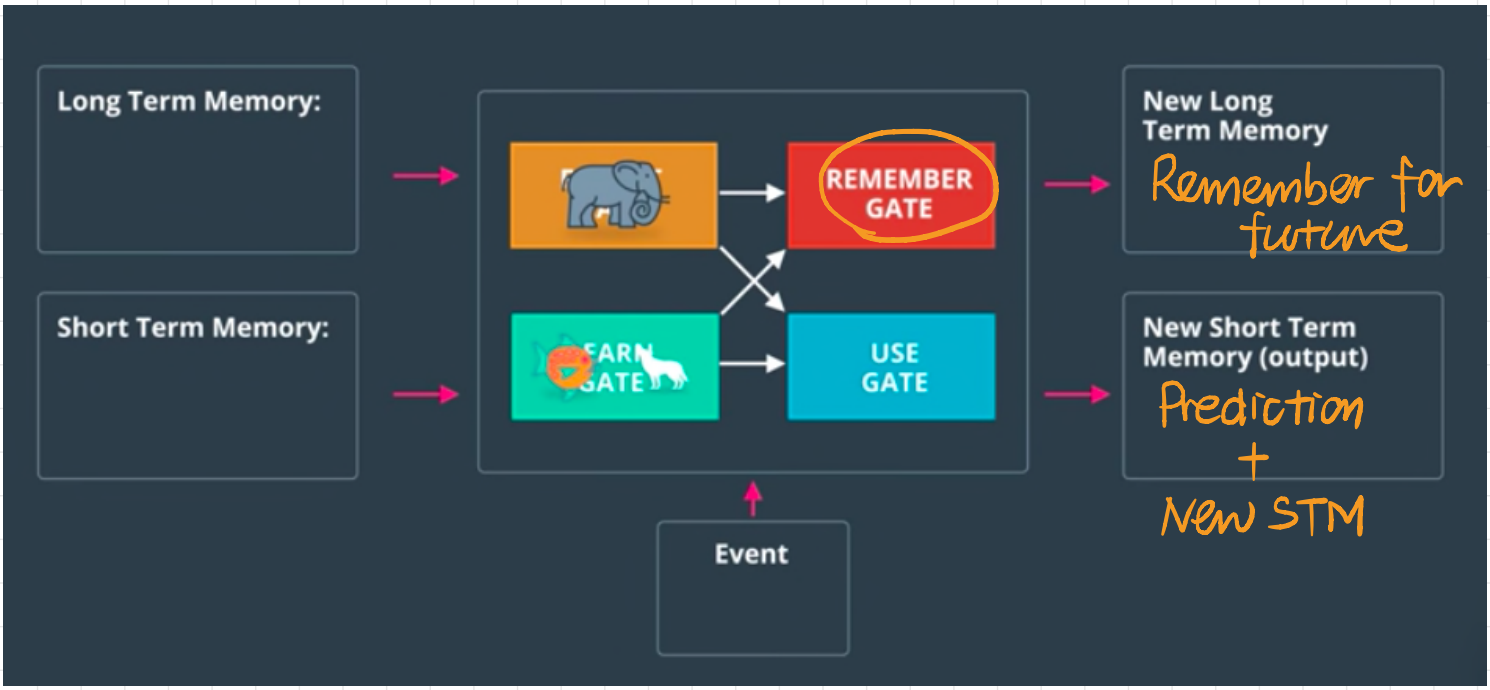


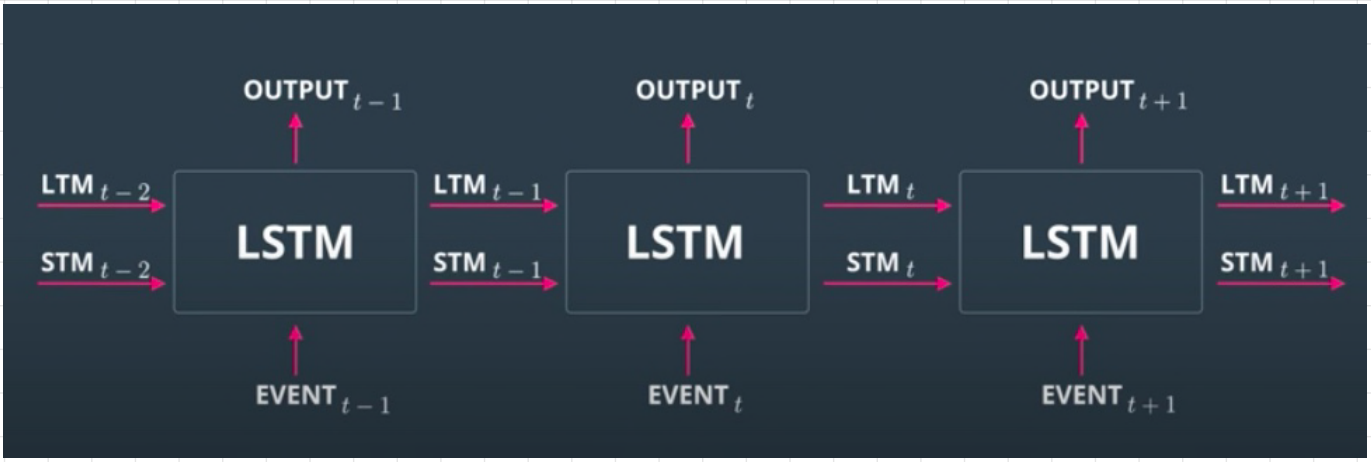
hint wolf

Help update our long term memory

Also update short term memory:
- Wolves
- squirrels

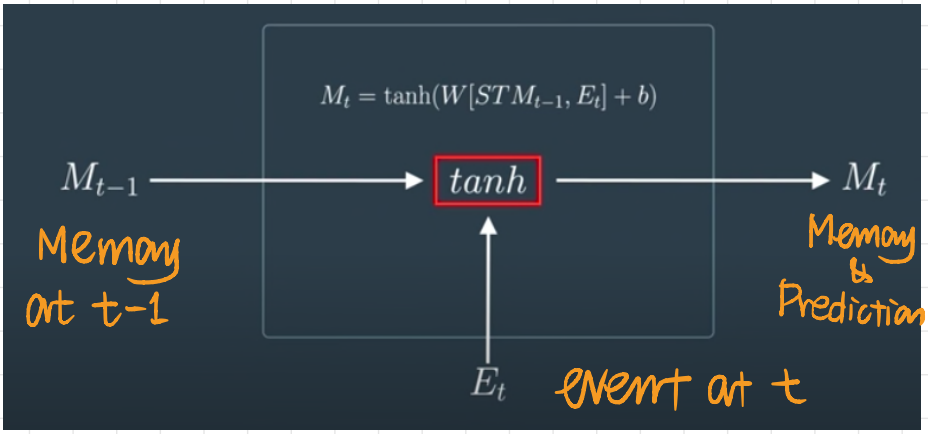
no trees



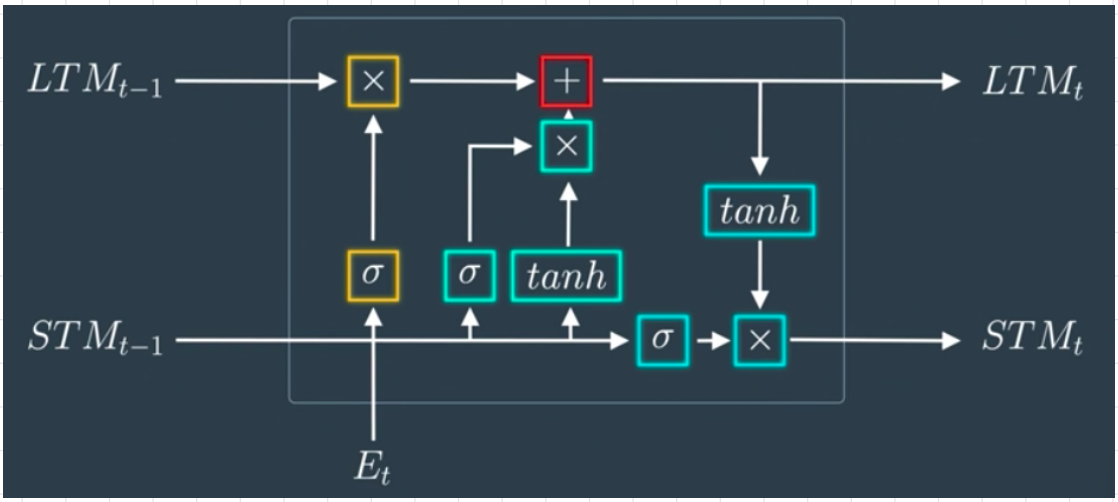


Architecture of LSTM

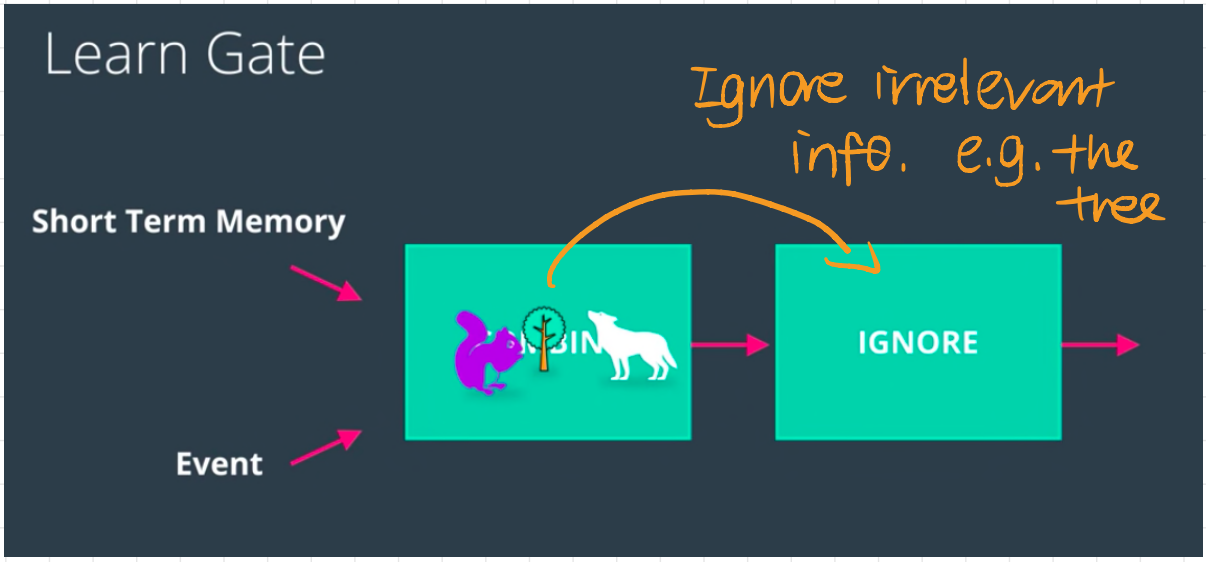
RNN Review

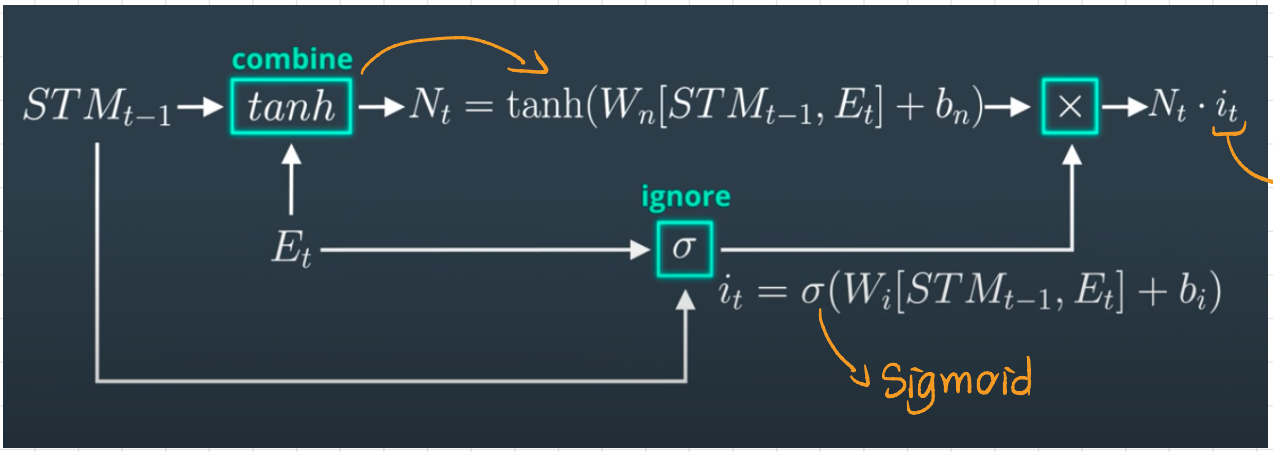


LSTM

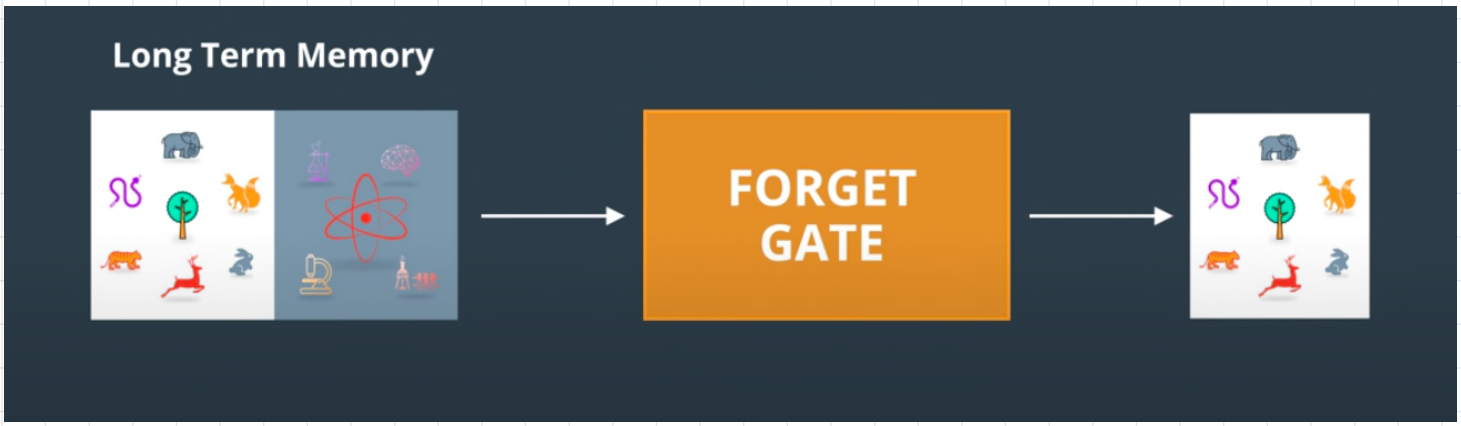


The Learn Gate

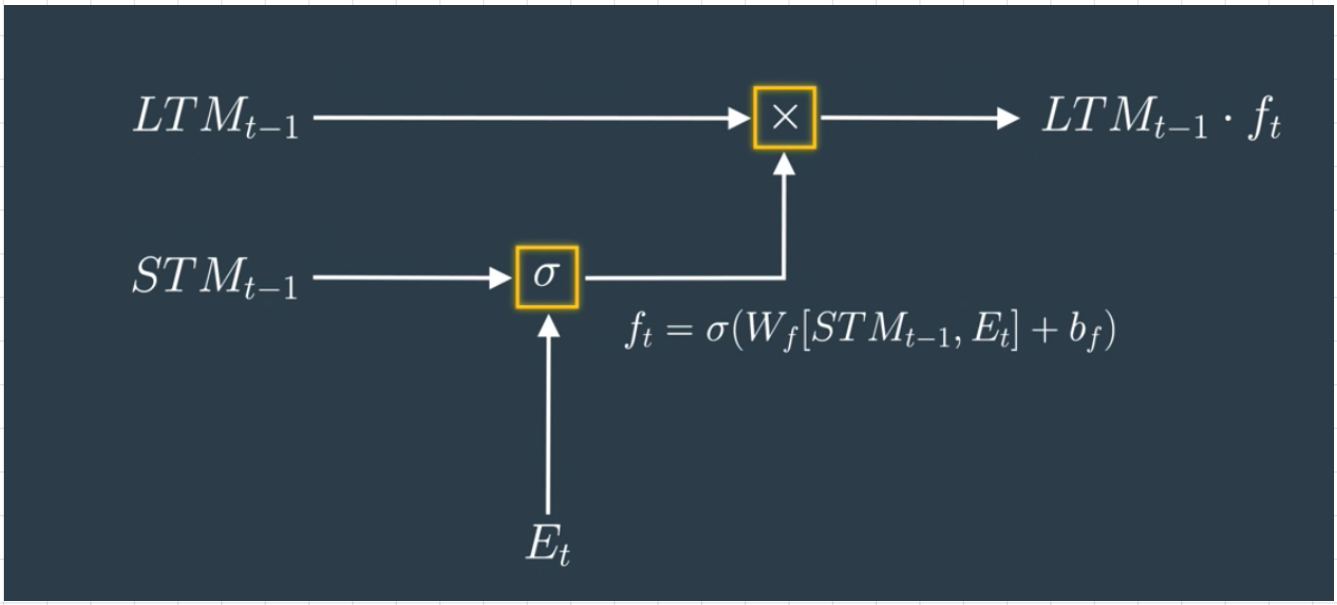




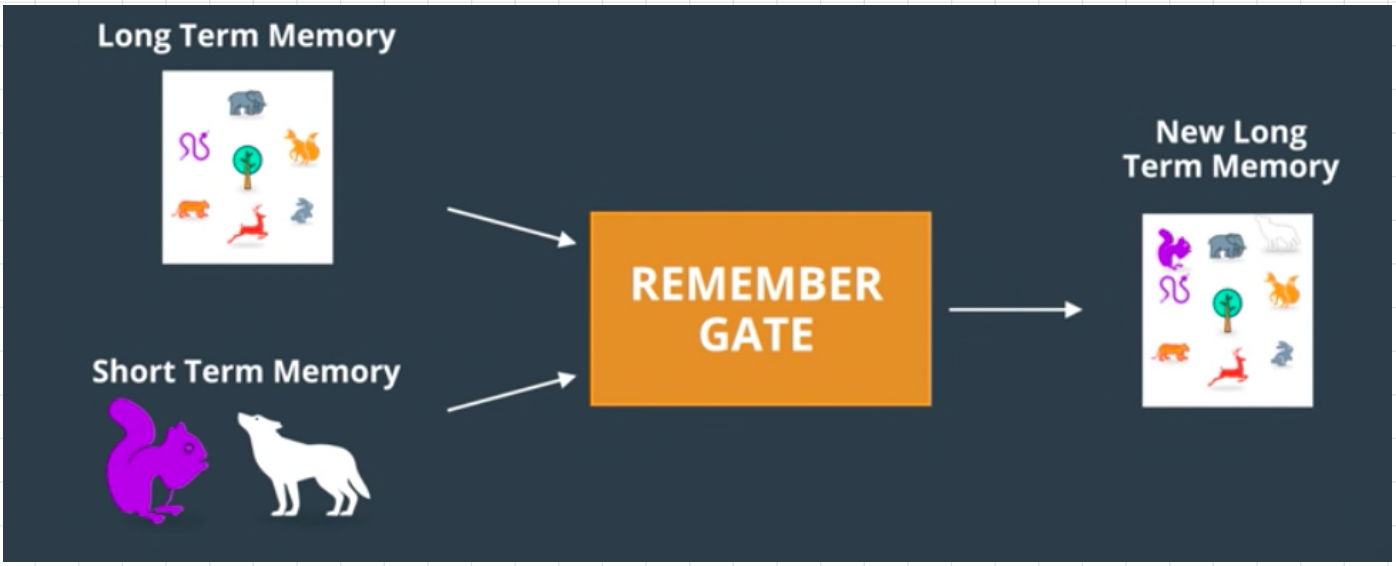
The Forget Gate

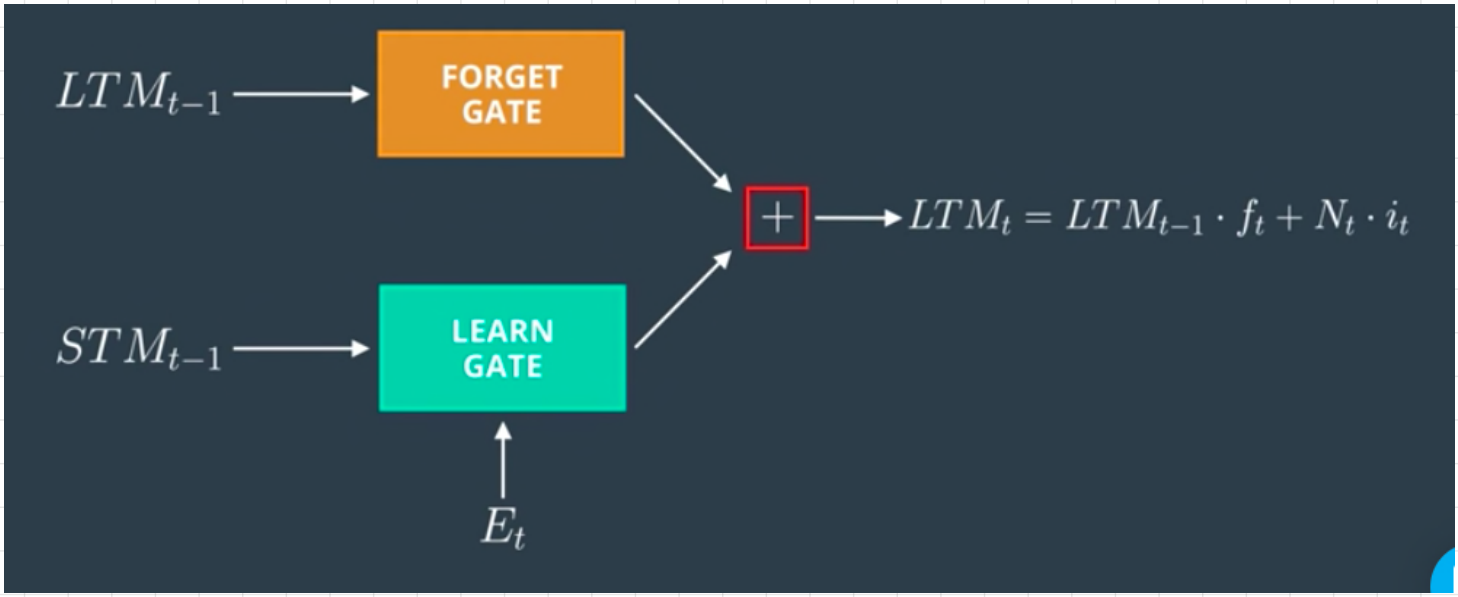


E.g. watching a nature show, so forget about science when making prediction.

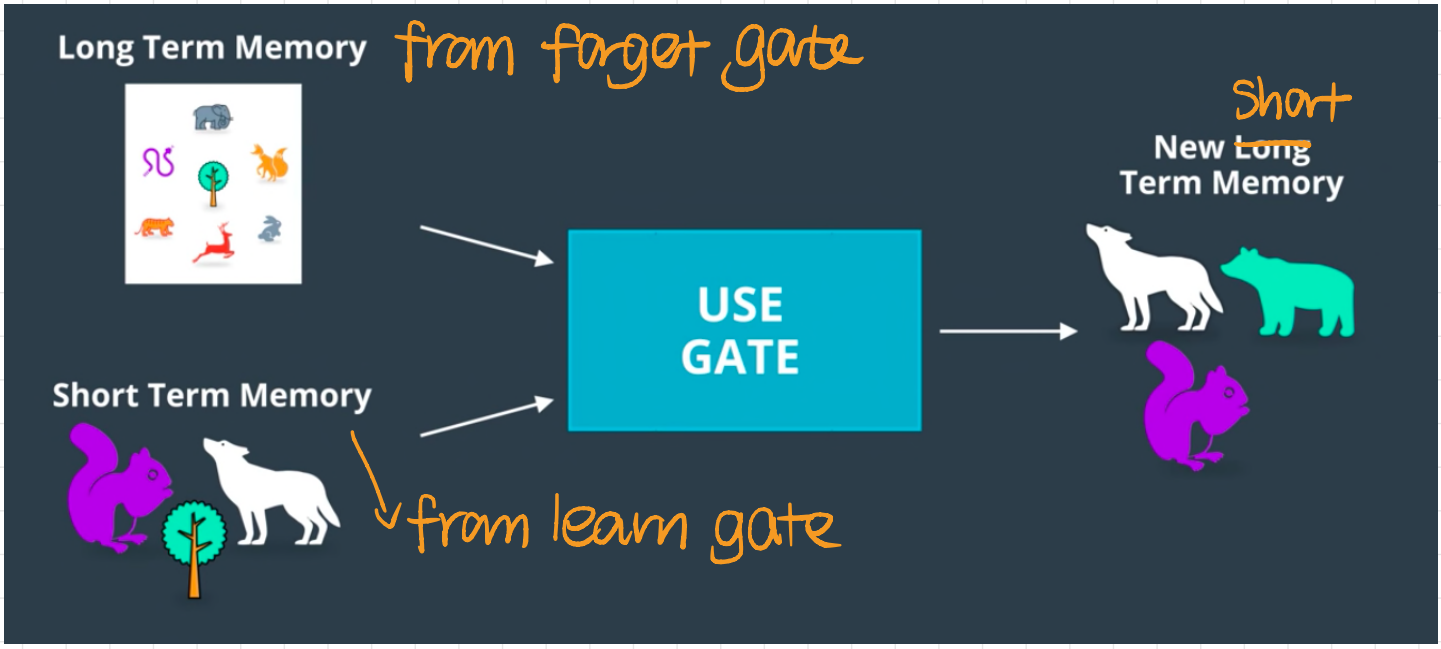


The Remember Gate





The Use Gate



Taking only the useful info from both.

$$U_t = \tanh(W_u LTM_{t-1} \cdot f_t + b_u)$$

LTM_{t-1} → FORGET GATE → $tanh$

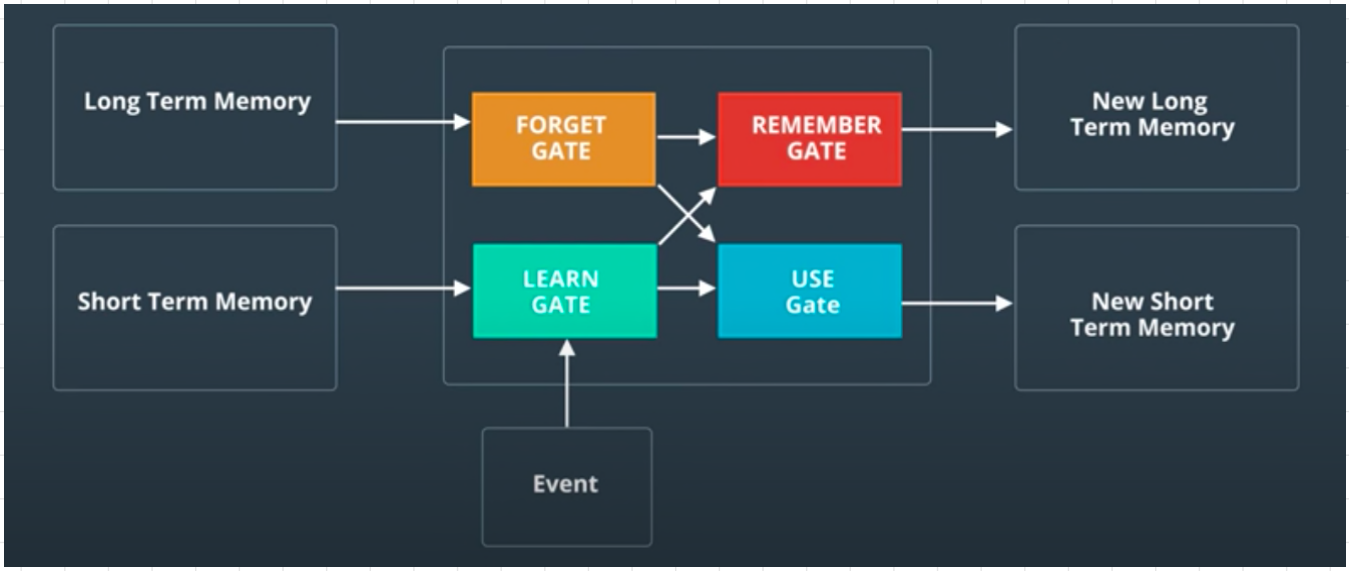
STM_{t-1} → σ

$\times$ → OUTPUT $STM_t = U_t \cdot V_t$

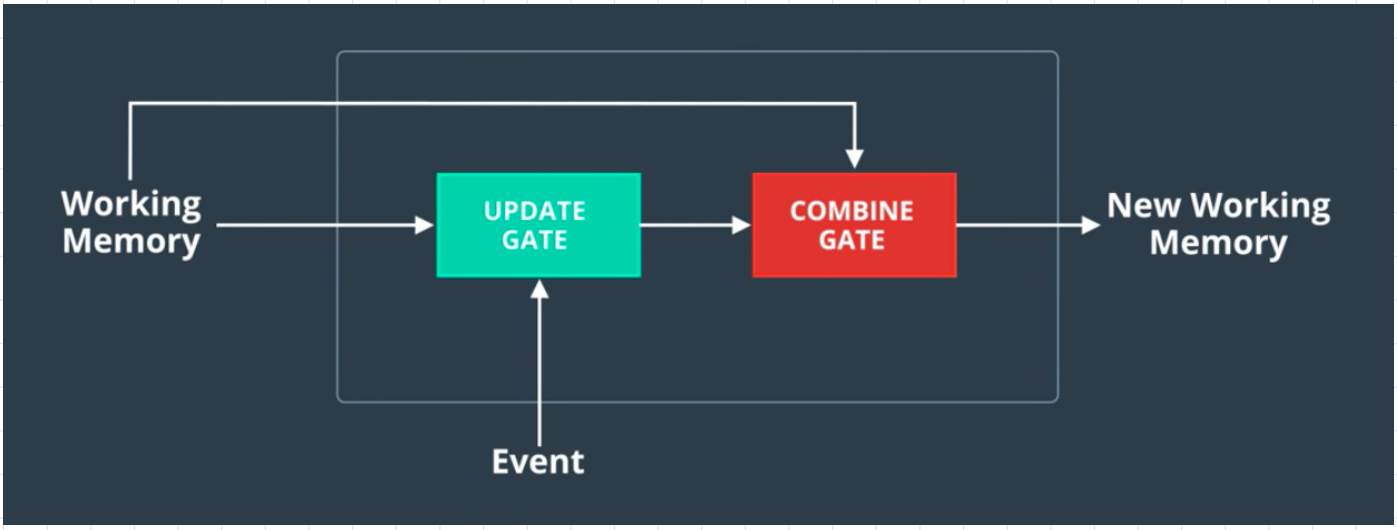
$$V_t = \sigma(W_v[STM_{t-1}, E_t] + b_v)$$

E_t

Put it together



GRU



Peephole connections

Monitoring Validation Loss vs. Training Loss

If you're somewhat new to Machine Learning or Neural Networks it can take a bit of expertise to get good models. The most important quantity to keep track of is the difference between your training loss (printed during training) and the validation loss (printed once in a while when the RNN is run on the validation data (by default every 1000 iterations)). In particular:

- If your training loss is much lower than validation loss then this means the network might be **overfitting**. Solutions to this are to decrease your network size, or to increase dropout. For example you could try dropout of 0.5 and so on.
- If your training/validation loss are about equal then your model is **underfitting**. Increase the size of your model (either number of layers or the raw number of neurons per layer)

Approximate number of parameters

The two most important parameters that control the model are `n_hidden` and `n_layers`. I would advise that you always use `n_layers` of either 2/3. The `n_hidden` can be adjusted based on how much data you have. The two important quantities to keep track of here are:

- The number of parameters in your model. This is printed when you start training.
- The size of your dataset. 1MB file is approximately 1 million characters.

These two should be about the same order of magnitude. It's a little tricky to tell. Here are some examples:

- I have a 100MB dataset and I'm using the default parameter settings (which currently print 150K parameters). My data size is significantly larger (100 mil >> 0.15 mil), so I expect to heavily underfit. I am thinking I can comfortably afford to make `n_hidden` larger.
- I have a 10MB dataset and running a 10 million parameter model. I'm slightly nervous and I'm carefully monitoring my validation loss. If it's larger than my training loss then I may want to try to increase dropout a bit and see if that helps the validation loss.

Best models strategy

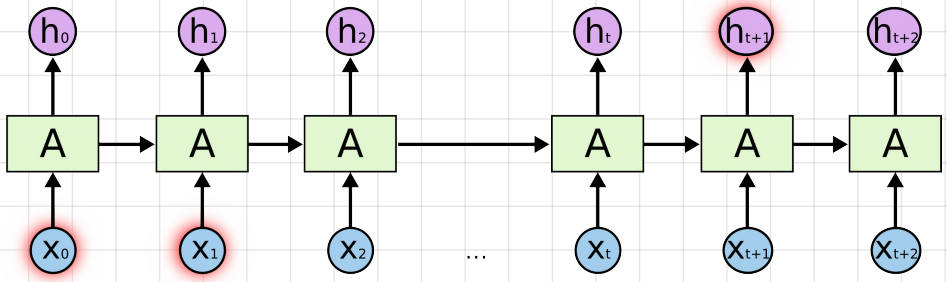
The winning strategy to obtaining very good models (if you have the compute time) is to always err on making the network larger (as large as you're willing to wait for it to compute) and then try different dropout values (between 0,1). Whatever model has the best validation performance (the loss, written in the checkpoint filename, low is good) is the one you should use in the end.

It is very common in deep learning to run many different models with many different hyperparameter settings, and in the end take whatever checkpoint gave the best validation performance.

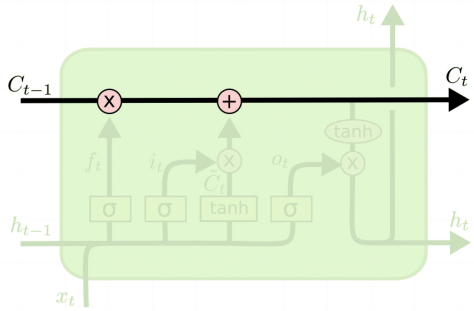
By the way, the size of your training and validation splits are also parameters. Make sure you have a decent amount of data in your validation set or otherwise the validation performance will be noisy and not very informative.

My own research about LSTM

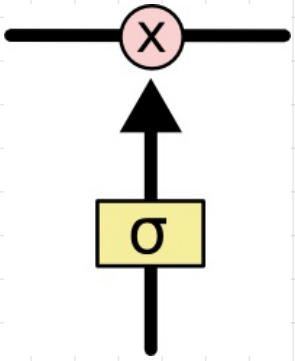
RNN: incapable of handling long-term memory



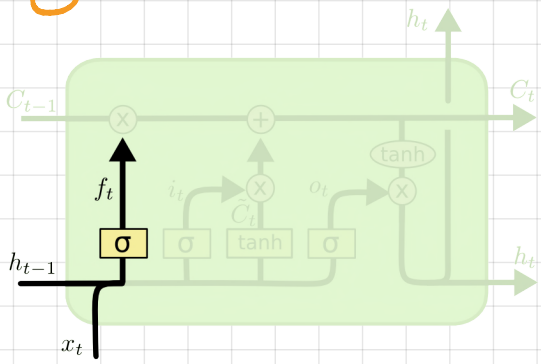
Cell state



Sigmoid gate.



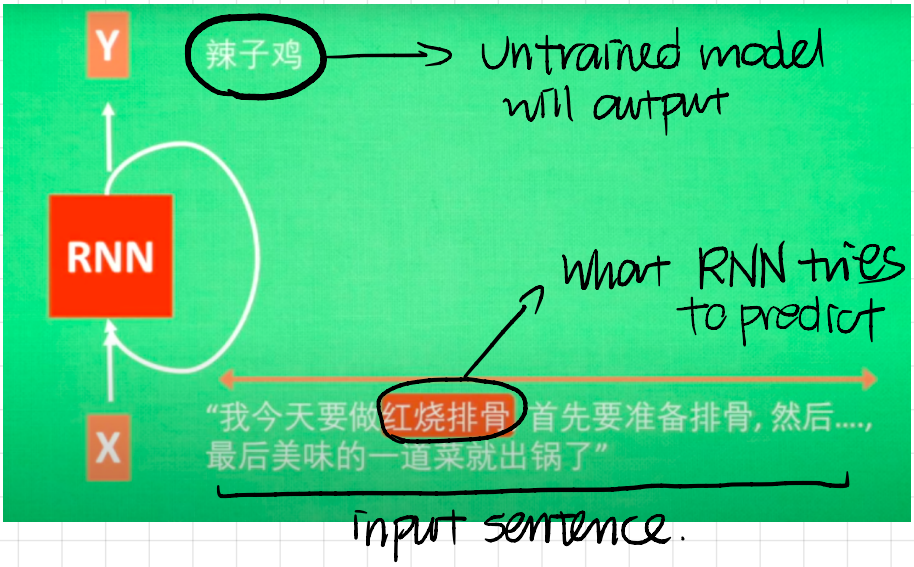
The Forget Gate



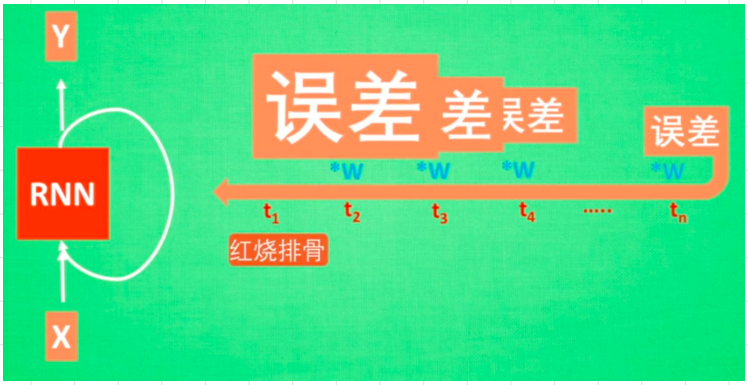
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

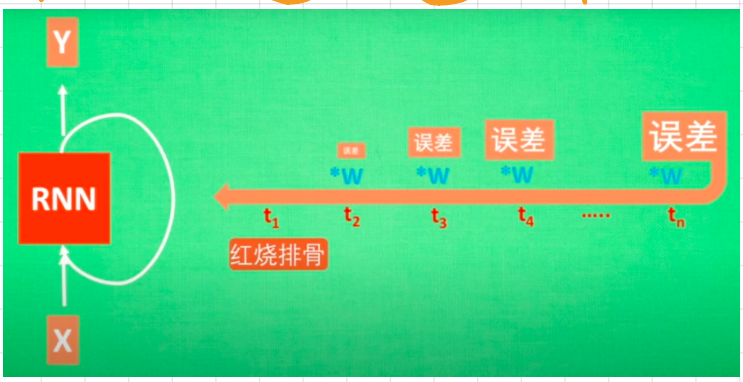
The point:

RNN has short memory.



Why RNN is not good at processing long sequence:

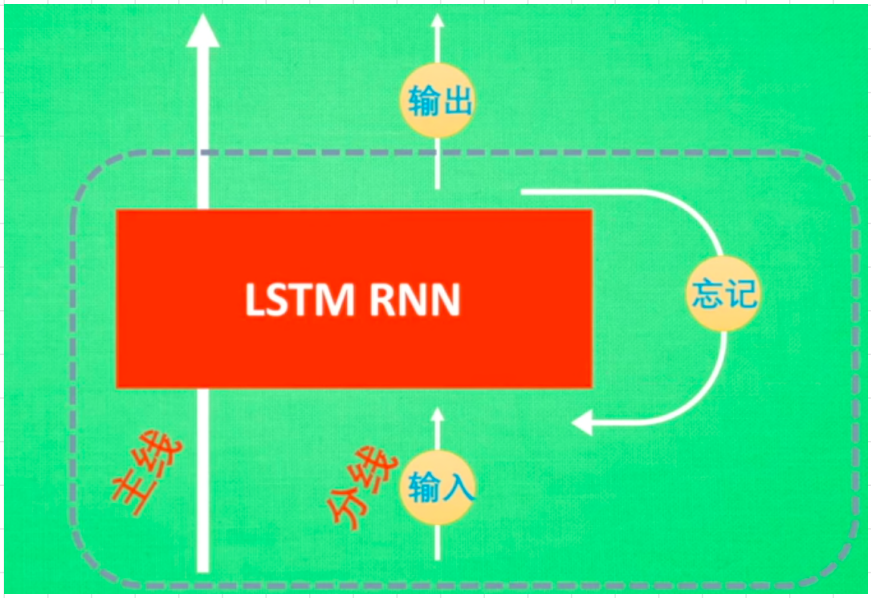




The reason lies in **Gradient Exploding** & **Vanishing Gradient**

The weight is multiplied by the gradient in backprop

Architecture:



主线：控制全局的记忆

输入：如果分线对结果重要，即写入主线

忘记：若分线改变了对主线的想法，一部分主线会被忘记

Regarding Hyperparameters

Embedding

vocab_size too large, so we use an embed layer to reduce the dimensionality.

init shape (vocab size, embed_dim)

≡ hidden dim

Time step = seq length.

# of hidden units ≡ # of LSTM cells, inputs of each timestep goes through each unit

- E.g. A sentence of 10 words. feature = words
  then timestep = 10, hidden_dim is of choice.

Output

- Shape: (batch size, seq length, output size)
  - e.g. 1 batch = 1 sentence (of 10 words)
    Each word goes through 15 LSTM units
    shape, output: (5, 10, 15)