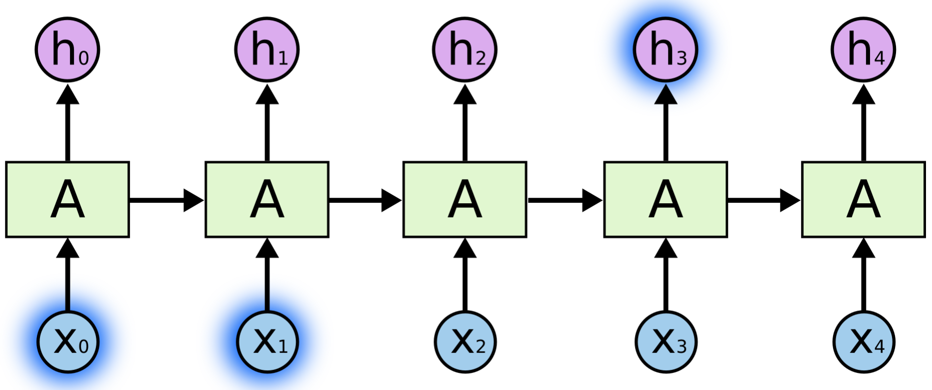
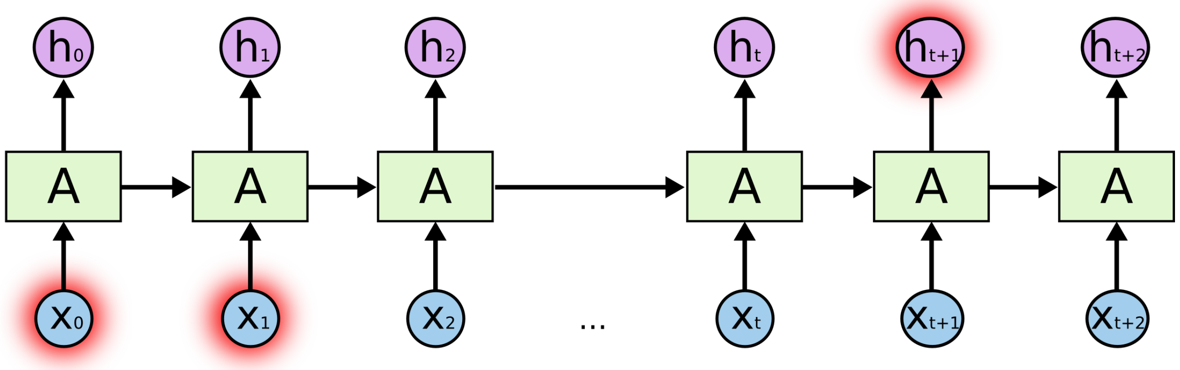
LSTM and GRU

1. The Problem of Long-Term Dependencies

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in **"the clouds are in the [ ],"** we don't need any further context – it's pretty obvious the next word is going to be **sky**. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.

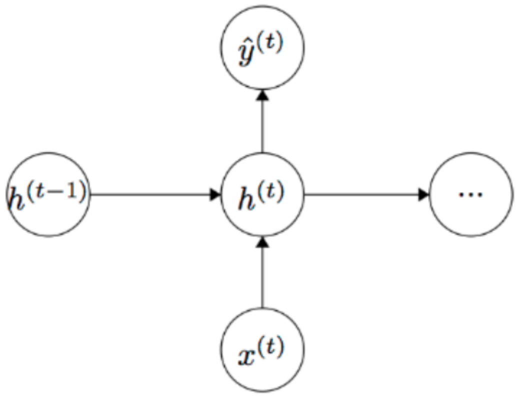


But there are also cases where we need more context. Consider trying to predict the last word in the text **"I grew up in France... I speak fluent [ ]."** Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of **France**, from further back.



In theory, RNNs are absolutely capable of handling such "long-term dependencies." A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don't seem to be able to learn them. RNN is likely to correctly predict **sky** than **French** in the examples above.

2. Vanishing Gradient & Gradient Explosion Problems



The graph above illustrates the input and output of a single neuron of a RNN where

$$h_t = \sigma(W^{hh}h_{t-1} + W^{hx}x_{[t]})$$
$$\hat{y} = softmax(W^{(S)}h_t)$$

and the same weights $W^{(hh)}$ and $W^{(hx)}$ are applied repeatedly at each timestep.

In theory RNN is able to capture long distance dependency. However in practice, its facing two primary challenges: **Gradient Explosion Problem** and **Vanishing Gradient Problem**.

Let's consider a simple situation for illustration purpose where the activation function is simply an identity transformation:

$$h_t := W^{hh}h_{t-1} + W^{xh}x_t$$

In error backpropagation, we know that the partial derivative of loss ℓ with respect to the hidden state h_t at time-step t is $\frac{\partial \ell}{\partial \boldsymbol{h}_t}$. Applying the chain rule, we can compute the partial derivative of loss ℓ with respect to the hidden state h_0 at the first time-step 0:

$$\frac{\partial \ell}{\partial h_0} = (\frac{\partial h_t}{\partial h_0})^T \frac{\partial \ell}{\partial h_t}$$

where

$$\frac{\partial h_t}{\partial h_0} = \prod_{i=1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=1}^{t} W_{hh} = (W_{hh})^t$$

This shows that in backpropagation, we are cumulatively multiplying the weight matrix W_{hh}:

$$\begin{aligned}\frac{\partial \ell}{\partial h_0} &= (\frac{\partial h_t}{\partial h_0})^T \frac{\partial \ell}{\partial h_t} \\ &= ((W_{hh})^t)^T \frac{\partial \ell}{\partial h_t}\end{aligned}$$

We can do a SVD on the weight matrix W_{hh}:

$$W_{hh} = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

where r is the rank of matrix W_{hh}.

Therefore

$$\frac{\partial h_t}{\partial h_0} = W_{hh}^t = U\Sigma^t V^T = \sum_{i=1}^{r} \sigma_i^t u_i v_i^T$$

and

$$\begin{aligned}\frac{\partial \ell}{\partial h_0} &= (W_{hh}^t)^T \frac{\partial \ell}{\partial h_t} \\ &= (\sum_{i=1}^{r} \sigma_i^t u_i v_i^T)^T \frac{\partial \ell}{\partial h_t} \\ &= \sum_{i=1}^{r} \sigma_i^t v_i u_i^T \frac{\partial \ell}{\partial h_t}\end{aligned}$$

When t is very large, the result of the partial derivative depends on whether W_{hh}'s largest eigenvalue σ_1 is larger than 1 or smaller than 1:

$$\begin{aligned}1.01^{365} &= 37.8 \\ 0.99^{365} &= 0.03 \\ 1.02^{365} &= 1377.4 \\ 0.98^{365} &= 0.0006\end{aligned}$$

- **Gradient Explosion Problem**: when $\sigma_1 > 1, \lim\limits_{t\to\infty} \sigma_1^t = \infty$

$$\frac{\partial \ell}{\partial \boldsymbol{h}_0} = \sum_{i=1}^{r} \sigma_i^t \boldsymbol{v}_i \boldsymbol{u}_i^\top \frac{\partial \ell}{\partial \boldsymbol{h}_t} \approx \infty \cdot \boldsymbol{v}_1 \boldsymbol{u}_1^\top \frac{\partial \ell}{\partial \boldsymbol{h}_t} = \infty$$

- **Vanishing Gradient Problem**: when $\sigma_1 < 1, \lim\limits_{t\to\infty} \sigma_1^t = 0$

$$\frac{\partial \ell}{\partial \boldsymbol{h}_0} = \sum_{i=1}^{r} \sigma_i^t \boldsymbol{v}_i \boldsymbol{u}_i^\top \frac{\partial \ell}{\partial \boldsymbol{h}_t} \approx 0 \cdot \boldsymbol{v}_1 \boldsymbol{u}_1^\top \frac{\partial \ell}{\partial \boldsymbol{h}_t} = 0$$

Note that **Gradient Explosion Problem** is a computation issue while **Vanishing Gradient Problem** causes memory loss from early time steps (long-term dependency).

3. Solution to the Exploding & Vanishing Gradients

3.1 Solution to Gradient Explosion Problem

To solve the problem of exploding gradients, Thomas Mikolov first introduced a simple heuristic solution called `gradient clipping`. That is, whenever the gradient exceeds a certain threshold, it is set back to the threshold.

```
$\hat{g} \leftarrow \frac{\partial E}{\partial W}$
if $||\hat{g}|| \geqslant$ threshold then
    $\hat{g} \leftarrow \frac{\text{threshold}}{||\hat{g}||}\hat{g}$
end if
```

The second technique is to use **ReLU** instead of sigmoid function. The derivative for the ReLU is either 0 or 1. This way, gradients would flow through the neurons whose derivative is 1 without getting attenuated while propagating back through time-steps.

3.2 Solution to Vanishing Gradient Problem

Vanishing Gradient Problem is like a problem of sending grain to the front line during war. The food delivery team also has to consume food themselves. When the supply point is too far from the front line, the food will be depleted halfway before it is delivered. In RNN, the gradient (partial derivative) is the grain, and the gradient is gradually consumed as it moves forward.

LSTM and GRU control the information flow in the RNN through a gate mechanism, which is used to alleviate the disappearance of the gradient. For example, when we read a product review

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. In only ate half of it but will definitely be buying again!

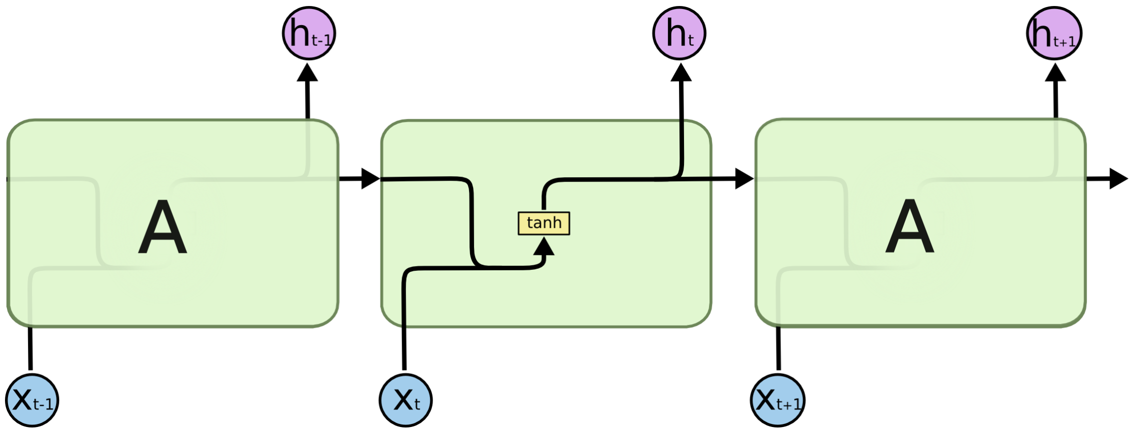
We will pay more attention to some words or phrases in the review:

Amazing! This box of cereal gave me a **perfectly balanced breakfast**, as all things should be. In only ate half of it but will **definitely be buying again!**

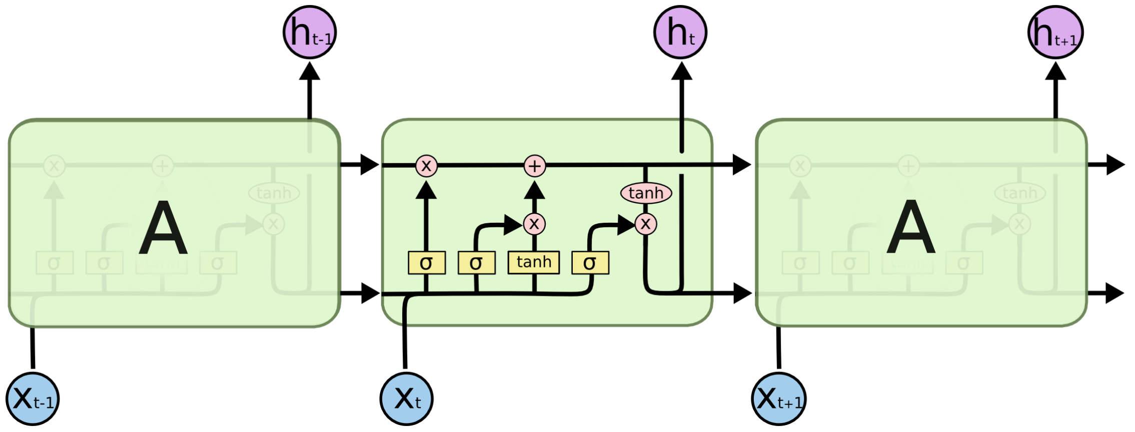
LSTM and GRU will selectively ignore some words so that they won't be involved in the information propagation through the hidden layers. Only relevant, important information will be propagated.

4. (Long Short Term Memory networks) LSTM

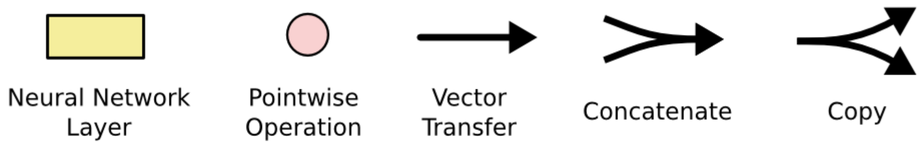
All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



Don't worry about the details of what's going on. We'll walk through the LSTM diagram step by step later. For now, let's just try to get comfortable with the notation we'll be using.



- Each line carries an entire vector, from the output of one node to the inputs of others
- The pink circles represent pointwise operations, like vector addition
- The yellow boxes are learned neural network layers
- Lines merging denote concatenation

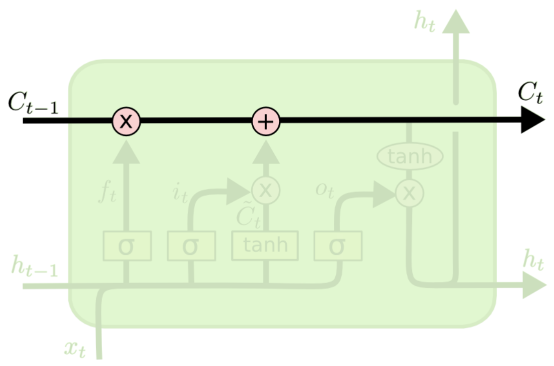
- A line forking denote its content being copied and the copies going to different locations

4.1 The Core Idea Behind LSTMs

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

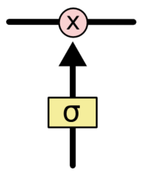
The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

Note that the cell state C_t only carries information through time-steps. There is another state, the hidden state h_t, that also carries information and act as output of the time-steps. You will see below that h_t interacts with the input at each time-step x_t to decide the output of the gates but C_t doesn't.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



The **sigmoid layer** outputs numbers between [0, 1], describing how much of each component should be let through. A value of 0 means "let nothing through," while a value of 1 means "let everything through!"

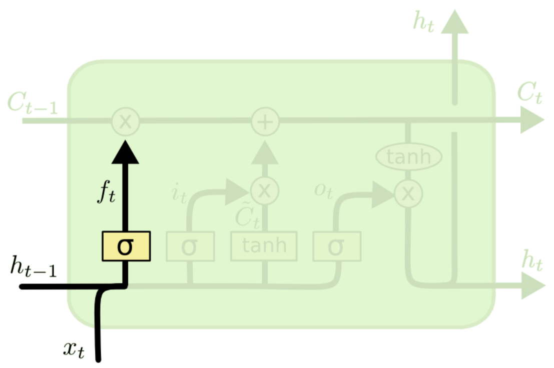
An LSTM has three of these **sigmoid** gates, to protect and control the cell state.

4.2 Step-by-Step LSTM Walk Through

4.2.1 Step 1: Decide what to forget or remember from old cell state C_{t-1}

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a **sigmoid** layer called the "forget gate layer." It looks at the hidden layer output from the last time-step h_{t-1} and input at current time-step x_t, and outputs a number between 0 and 1 for each element in the cell state from the last time-step C_{t-1}. A 1 represents "completely keep this element" while a 0 represents "completely get rid of this element." Note that it's usually a number between 0 - 1, meaning some information of the last time-step cell state is kept and some is forgotten.

In later step, we will perform elementwise multiplication between f_t and C_{t-1}. This is how the sigmoid layer is acting as a "gate" and how it impacts the information passed from C_{t-1} to the current time-step.

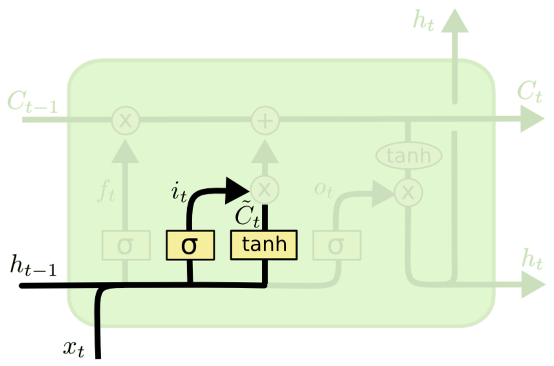


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

4.2.2 Step 2: Decide what to forget or remember from new input x_t and old hidden state h_{t-1}

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a **sigmoid** layer called the "input gate layer" decides which values from the input at the current time-step x_t we'll allow to impact the cell state. The output of the sigmoid layer is denoted as i_t. Next, a **tanh** layer creates a vector of new candidate values $\tilde{C}_t$ from the input at the current time-step x_t (and possibly some information from h_{t-1}), that could be added to the cell state. Note that the **tanh** function squashes the data into values between [-1, 1] which does the normalization job. In the next step, we'll combine these two to create an update to the state.



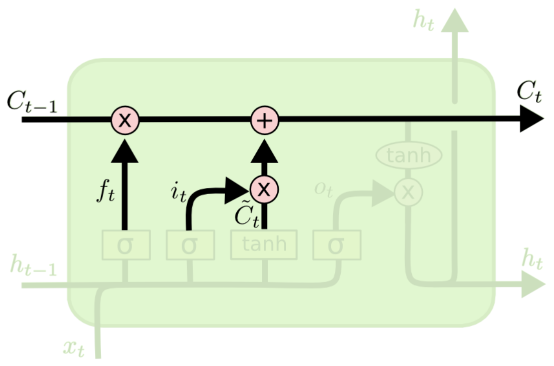
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

4.2.3 Step 3: Update the old cell state C_{t-1}

It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value. Note that below "*" is elementwise multiplication.

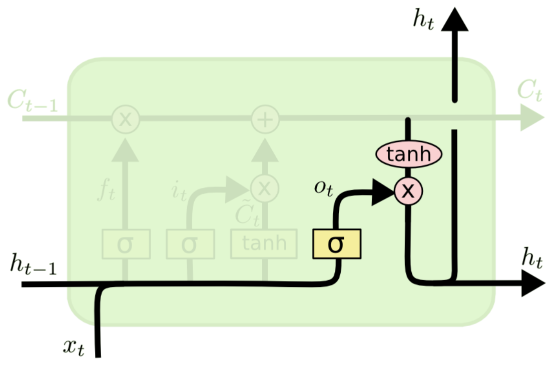


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.

4.2.4 Step 4: Decide what to output at the current time-step: h_t

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a **sigmoid** layer which decides what parts of the cell state we're going to output. Then, we put the cell state C_t through **tanh** (to push the values to be between −1 and 1) and multiply it by the output of the **sigmoid** gate, so that we only output the parts we decided to. h_t is the output at the current time-step and is also the hidden state at the current time-step. Note that below "*" is elementwise multiplication.

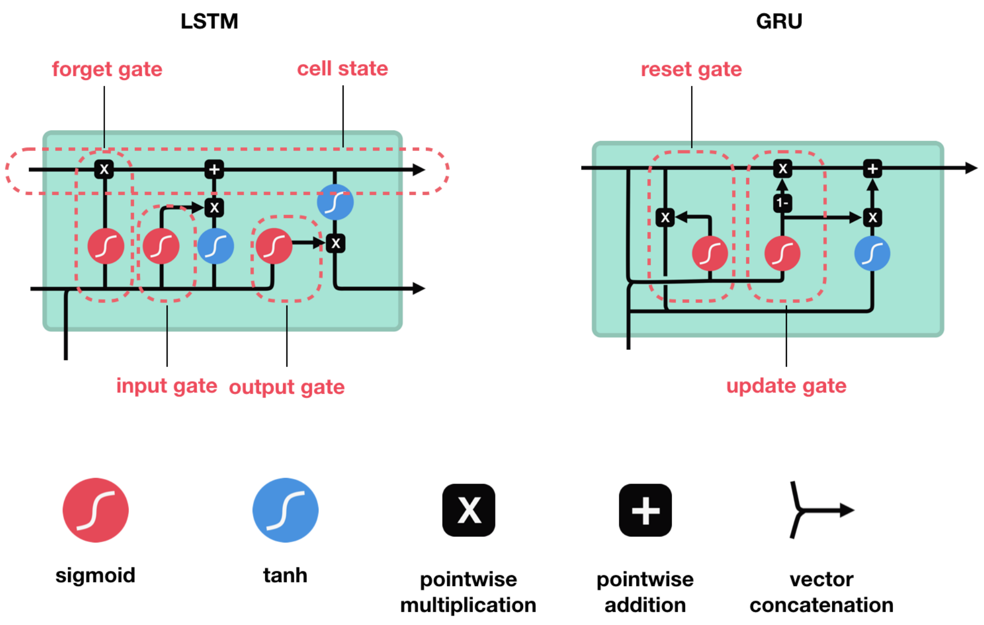


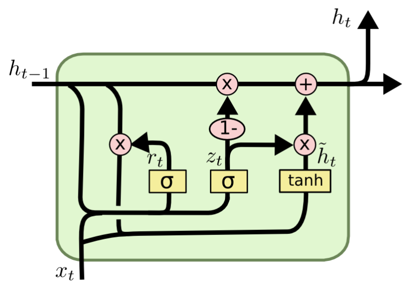
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

5. Gated Recurrent Unit (GRU)

The GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM. **GRU's got rid of the cell state and used the hidden state to transfer information**. It also only has two gates, a reset gate and update gate.

- The **update gate** z_t acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.
- The **reset gate** r_t is another gate is used to decide how much past information to forget.





$$\begin{aligned} z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \\ r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \\ \tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \end{aligned}$$

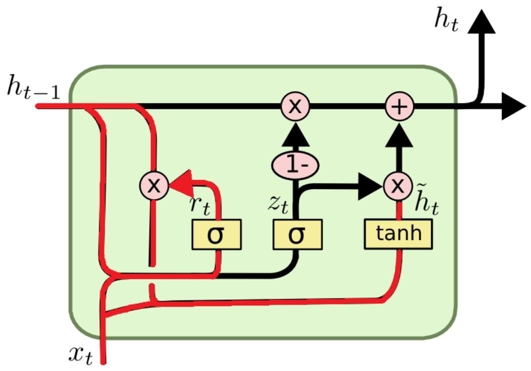
Both the update gate z_t and the reset gate r_t are decided by the combination of the hidden state from the last time-step h_{t-1} and the input of the current time step x_t.

5.1 Step 1: Decide what to forget or remember from new input x_t

Firstly, we compute the result of the reset gate by $r_t = \sigma(W_z \cdot [h_{t-1}, x_t])$. r_t is used to "reset" the hidden state of the last time-step h_{t-1} by performing elementwise multiplication: $h'_{t-1} = h_{t-1} * r_t$. Note that in the graph above, h_{t-1} flows downwards (second vertical path).

Then we perform vector concatenation of “reset" hidden state h'_{t-1} and input at current time-step x_t. And then the result will go through a **tanh layer** so that the values of the vector is squashed to [-1, 1]. We use $\tilde{h}_t$ to represent the vector here.

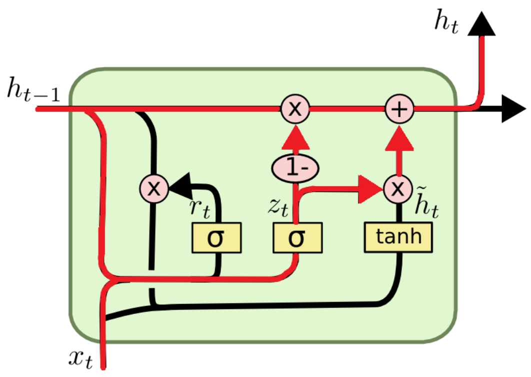
$\tilde{h}_t$ is essentially the combination of **some** information (controlled by reset gate r_t) from the hidden state of the last time-step h_{t-1} and **all** information of the input at the current time-step x_t.



5.2 Step 2: Decide what to output at the current time-step: h_t, by combining some info from h_{t-1} and some info from x_t

At this step, firstly we compute the update gate by $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$. Through the sigmoid function, the values of the elements in z_t are between [0, 1]. 1 means completely keep the information of the corresponding element while 0 means completely forget the information of the corresponding element.

We then compute the new hidden state $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$



- $(1 - z_t) * h_{t-1}$: $(1 - z_t)$ has values between [0, 1]. It acts as a gate to selectively "forget" or "keep" information from the hidden state from the last time-step h_{t-1}. It is similar to the "forget gate" in LSTM. Essentially, $(1 - z_t) * h_{t-1}$ contains **some** information of h_{t-1}.
- $z_t * \tilde{h}_t$: Recall that $\tilde{h}_t$ is essentially the combination of **some** information from the hidden state of the last time-step h_{t-1} and **all** information of the input at the current time-step x_t. The elementwise multiplication here is mainly to selectively "forget" or "keep" information from the input at the current time-step x_t, because $\tilde{h}_t$ contains all information of x_t. Essentially, $z_t * \tilde{h}_t$ contains **some** information of h_{t-1} and **some** information of x_t.
- Intuitively, $(1 - z_t)$ and z_t have some kind of balance and connection. If $(1 - z_t)$ is close to 1, it will keep more information from the hidden state of the last time-step h_{t-1}. In the meantime, z_t will be close to 0, which will keep less information from the input of the current time-step x_t.

References

1. 三次简化一张图：一招理解LSTM/GRU门控机制, 张皓, available from: https://zhuanlan.zhihu.com/p/28297161
2. Understanding LSTM Networks, Christopher Olah, available from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
3. 人人都能看懂的GRU, 陈诚, available from: https://zhuanlan.zhihu.com/p/32481747