

# Lecture 12:

# Language and Sequence Modeling

Iurii Efimov



**girafe**  
**ai**

# Outline

1. RNN intuitions
2. Language models
3. Memory concept: LSTM
4. RNN as encoder for sequential data
5. Vanishing gradient

# RNNs generating...



Shakespeare

PANDARUS:

Alas, I think he shall be come approached and the day  
When little grain would be attain'd into being never fed,  
And who is but a chain and subjects of his death,  
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,  
Breaking and strongly should be buried, when I perish  
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and  
my fair nues begun out of the fact, to be conveyed,  
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

Algebraic Geometry  
(Latex)

*Proof.* Omitted. □

**Lemma 0.1.** *Let  $\mathcal{C}$  be a set of the construction.*

*Let  $\mathcal{C}$  be a gerber covering. Let  $\mathcal{F}$  be a quasi-coherent sheaves of  $\mathcal{O}$ -modules. We have to show that*

$$\mathcal{O}_{\mathcal{X}} = \mathcal{O}_X(\mathcal{L})$$

*Proof.* This is an algebraic space with the composition of sheaves  $\mathcal{F}$  on  $X_{\text{étale}}$  we have

$$\mathcal{O}_X(\mathcal{F}) = \{ \text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F}) \}$$

where  $\mathcal{G}$  defines an isomorphism  $\mathcal{F} \rightarrow \mathcal{F}$  of  $\mathcal{O}$ -modules. □

**Lemma 0.2.** *This is an integer  $\mathbb{Z}$  is injective.*

*Proof.* See Spaces, Lemma ?? □

**Lemma 0.3.** *Let  $S$  be a scheme. Let  $X$  be a scheme and  $X$  is an affine open covering. Let  $\mathcal{U} \subset X$  be a canonical and locally of finite type. Let  $X$  be a scheme. Let  $X$  be a scheme which is equal to the formal complex.*

*The following to the construction of the lemma follows.*

*Let  $X$  be a scheme. Let  $X$  be a scheme covering. Let*

$$b: X \rightarrow Y' \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

*be a morphism of algebraic spaces over  $S$  and  $Y$ .*

*Proof.* Let  $X$  be a nonzero scheme of  $X$ . Let  $X$  be an algebraic space. Let  $\mathcal{F}$  be a quasi-coherent sheaf of  $\mathcal{O}_X$ -modules. The following are equivalent

- (1)  $\mathcal{F}$  is an algebraic space over  $S$ .
- (2) If  $X$  is an affine open covering.

Consider a common structure on  $X$  and  $X$  the functor  $\mathcal{O}_X(\mathcal{U})$  which is locally of finite type. □

Linux kernel (source code)

```
/*  
 * If this error is set, we will need anything right after that BSD.  
 */  
  
static void action_new_function(struct s_stat_info *wb)  
{  
    unsigned long flags;  
    int lel_idx_bit = e->edd, *sys & -((unsigned long) *FIRST_COMPAT);  
    buf[0] = 0x7fffffff & (bit << 4);  
    min(inc, slist->bytes);  
    printk(KERN_WARNING "Memory allocated %02x/%02x, "  
           "original MLL instead\n"),  
           min(min(multi_run - s->len, max) * num_data_in),  
           frame_pos, sz + first_seg);  
    div_u64_w(val, inb_p);  
    spin_unlock(&disk->queue_lock);  
    mutex_unlock(&s->sock->mutex);  
    mutex_unlock(&func->mutex);  
    return disassemble(info->pending_bh);  
}
```

Source: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>



*Proof.* Omitted. □

**Lemma 0.1.** *Let  $\mathcal{C}$  be a set of the construction.*

*Let  $\mathcal{C}$  be a gerber covering. Let  $\mathcal{F}$  be a quasi-coherent sheaves of  $\mathcal{O}$ -modules. We have to show that*

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

*Proof.* This is an algebraic space with the composition of sheaves  $\mathcal{F}$  on  $X_{\text{étale}}$  we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where  $\mathcal{G}$  defines an isomorphism  $\mathcal{F} \rightarrow \mathcal{F}$  of  $\mathcal{O}$ -modules. □

**Lemma 0.2.** *This is an integer  $Z$  is injective.*

*Proof.* See Spaces, Lemma ?? □

**Lemma 0.3.** *Let  $S$  be a scheme. Let  $X$  be a scheme and  $X$  is an affine open covering. Let  $\mathcal{U} \subset \mathcal{X}$  be a canonical and locally of finite type. Let  $X$  be a scheme. Let  $X$  be a scheme which is equal to the formal complex.*

*The following to the construction of the lemma follows.*

*Let  $X$  be a scheme. Let  $X$  be a scheme covering. Let*

$$b: X \rightarrow Y' \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

*be a morphism of algebraic spaces over  $S$  and  $Y$ .*

*Proof.* Let  $X$  be a nonzero scheme of  $X$ . Let  $X$  be an algebraic space. Let  $\mathcal{F}$  be a quasi-coherent sheaf of  $\mathcal{O}_X$ -modules. The following are equivalent

- (1)  $\mathcal{F}$  is an algebraic space over  $S$ .
- (2) If  $X$  is an affine open covering.

Consider a common structure on  $X$  and  $X$  the functor  $\mathcal{O}_X(U)$  which is locally of finite type. □

This since  $\mathcal{F} \in \mathcal{F}$  and  $x \in \mathcal{G}$  the diagram

$$\begin{array}{ccc} S & \xrightarrow{\quad} & \\ \downarrow & & \downarrow \\ \xi & \xrightarrow{\quad} & \mathcal{O}_{X'} \\ \text{gor}_s & \uparrow & \searrow \\ & \alpha' & \\ & \downarrow & \\ & \alpha' & \rightarrow \alpha \end{array} \quad \begin{array}{c} X \\ \downarrow \\ \text{Mor}_{\text{Sets}} \text{d}(\mathcal{O}_{X_{X/\mathcal{F}}}, \mathcal{G}) \end{array}$$

is a limit. Then  $\mathcal{G}$  is a finite type and assume  $S$  is a flat and  $\mathcal{F}$  and  $\mathcal{G}$  is a finite type  $f_*$ . This is of finite type diagrams, and

- the composition of  $\mathcal{G}$  is a regular sequence,
- $\mathcal{O}_{X'}$  is a sheaf of rings.

□

*Proof.* We have see that  $X = \text{Spec}(R)$  and  $\mathcal{F}$  is a finite type representable by algebraic space. The property  $\mathcal{F}$  is a finite morphism of algebraic stacks. Then the cohomology of  $X$  is an open neighbourhood of  $U$ . □

*Proof.* This is clear that  $\mathcal{G}$  is a finite presentation, see Lemmas ??.

A reduced above we conclude that  $U$  is an open covering of  $\mathcal{C}$ . The functor  $\mathcal{F}$  is a “field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_x \longrightarrow \mathcal{O}_{X_{\text{étale}}}^{-1} \longrightarrow \mathcal{O}_{X_{\lambda}}^{-1} \mathcal{O}_{X_{\lambda}}(\mathcal{O}_{X_{\eta}}^{\vee})$$

is an isomorphism of covering of  $\mathcal{O}_{X_{\lambda}}$ . If  $\mathcal{F}$  is the unique element of  $\mathcal{F}$  such that  $X$  is an isomorphism.

The property  $\mathcal{F}$  is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme  $\mathcal{O}_X$ -algebra with  $\mathcal{F}$  are opens of finite type over  $S$ . If  $\mathcal{F}$  is a scheme theoretic image points. □

If  $\mathcal{F}$  is a finite direct sum  $\mathcal{O}_{X_{\lambda}}$  is a closed immersion, see Lemma ??.

This is a sequence of  $\mathcal{F}$  is a similar morphism.



```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

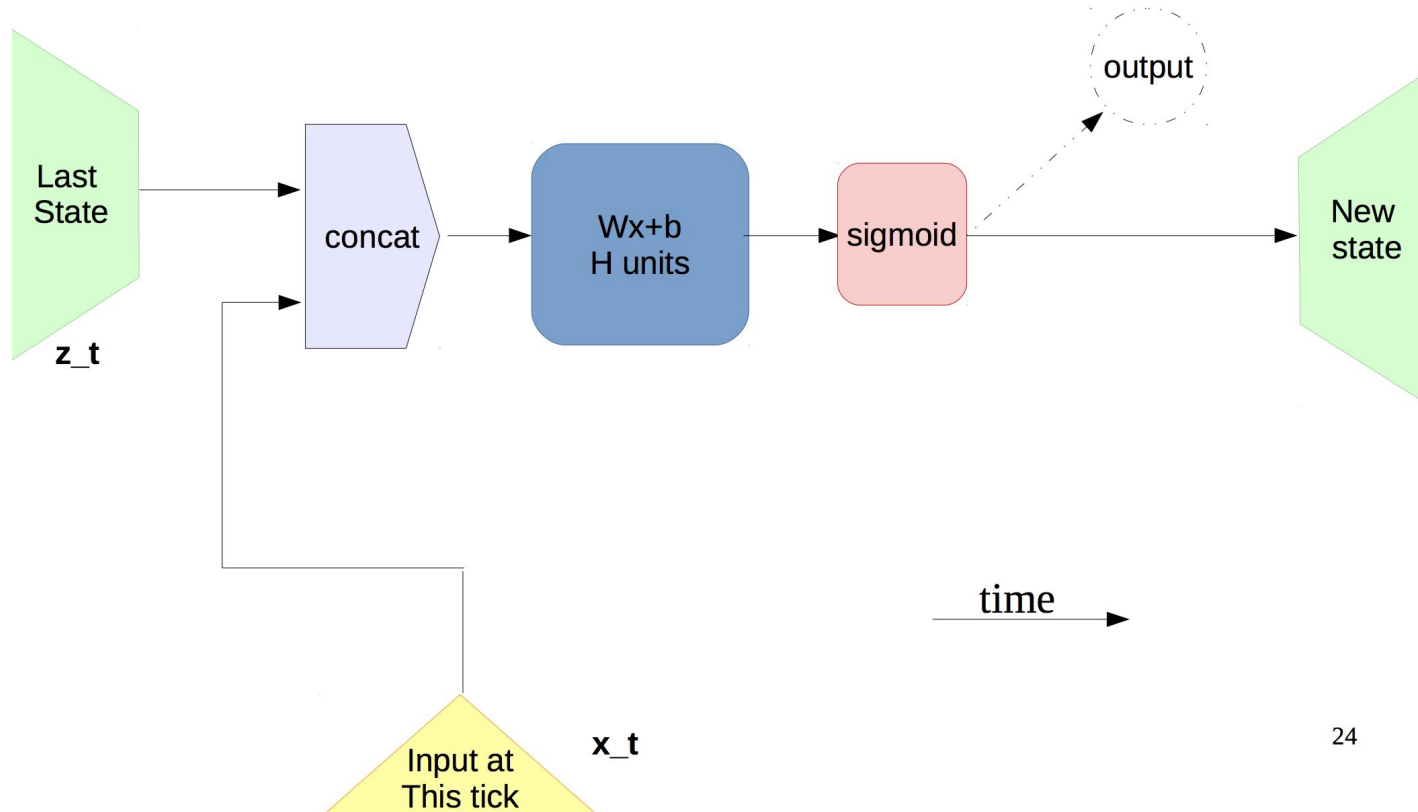
#define REG_PG      vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)      (func)

#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %!esp, %0, %3" : : "r" (0)); \
    if (__type & DO_READ)

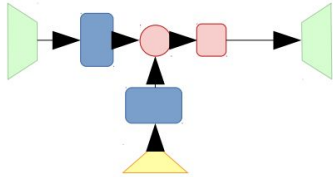
static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
    PUT_PARAM_RAID(2, sel) = get_state_state();
    set_pid_sum((unsigned long)state, current_state_str(),
        (unsigned long)-1->lr_full; low;
}
}
```

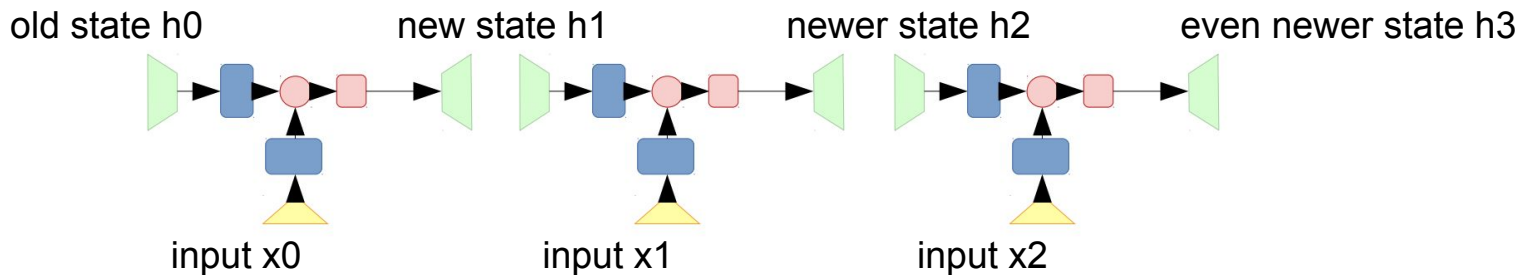
# Recurrent neural network



# Recurrent neural network



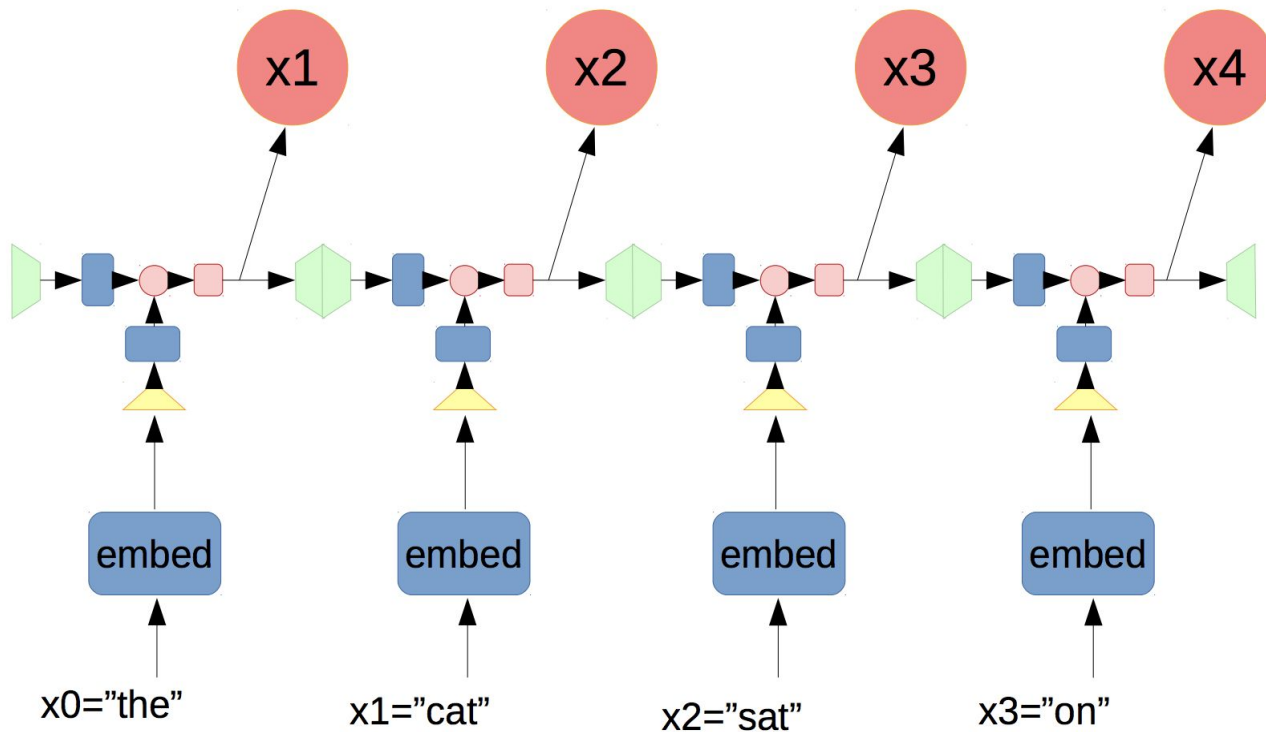
# Recurrent neural network



We use same weight matrices for all steps



# Recurrent neural network





# Recurrent neural network: with formulas

$$h_0 = \bar{0}$$

$$h_1 = \sigma(\langle W_{\text{hid}}[h_0, x_0] \rangle + b)$$

$$h_2 = \sigma(\langle W_{\text{hid}}[h_1, x_1] \rangle + b) = \sigma(\langle W_{\text{hid}}[\sigma(\langle W_{\text{hid}}[h_0, x_0] \rangle + b, x_1)] \rangle + b)$$

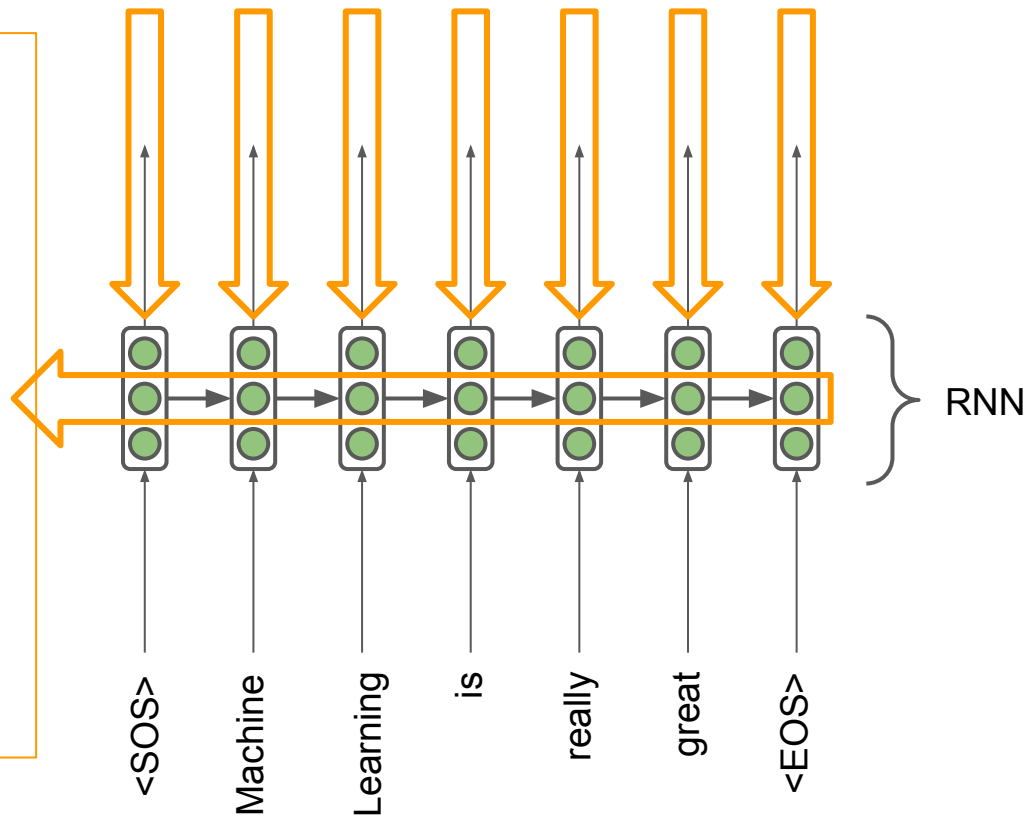
$$h_{i+1} = \sigma(\langle W_{\text{hid}}[h_i, x_i] \rangle + b)$$

$$P(x_{i+1}) = \text{softmax}(\langle W_{\text{out}}, h_i \rangle + b_{\text{out}})$$

# How to train it?

Loss

(e.g. Negative  
log-likelihood)



# RNNs generating...



Shakespeare

PANDARUS:

Alas, I think he shall be come approached and the day  
When little grain would be attain'd into being never fed,  
And who is but a chain and subjects of his death,  
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,  
Breaking and strongly should be buried, when I perish  
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and  
my fair nues begun out of the fact, to be conveyed,  
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

Algebraic Geometry  
(Latex)

*Proof.* Omitted. □

**Lemma 0.1.** *Let  $\mathcal{C}$  be a set of the construction.*

*Let  $\mathcal{C}$  be a gerber covering. Let  $\mathcal{F}$  be a quasi-coherent sheaves of  $\mathcal{O}$ -modules. We have to show that*

$$\mathcal{O}_{\mathcal{X}} = \mathcal{O}_X(\mathcal{L})$$

*Proof.* This is an algebraic space with the composition of sheaves  $\mathcal{F}$  on  $X_{\text{étale}}$  we have

$$\mathcal{O}_X(\mathcal{F}) = \{ \text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F}) \}$$

where  $\mathcal{G}$  defines an isomorphism  $\mathcal{F} \rightarrow \mathcal{F}$  of  $\mathcal{O}$ -modules. □

**Lemma 0.2.** *This is an integer  $\mathbb{Z}$  is injective.*

*Proof.* See Spaces, Lemma ?? □

**Lemma 0.3.** *Let  $S$  be a scheme. Let  $X$  be a scheme and  $X$  is an affine open covering. Let  $\mathcal{U} \subset X$  be a canonical and locally of finite type. Let  $X$  be a scheme. Let  $X$  be a scheme which is equal to the formal complex.*

*The following to the construction of the lemma follows.*

*Let  $X$  be a scheme. Let  $X$  be a scheme covering. Let*

$$b: X \rightarrow Y' \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

*be a morphism of algebraic spaces over  $S$  and  $Y$ .*

*Proof.* Let  $X$  be a nonzero scheme of  $X$ . Let  $X$  be an algebraic space. Let  $\mathcal{F}$  be a quasi-coherent sheaf of  $\mathcal{O}_X$ -modules. The following are equivalent

- (1)  $\mathcal{F}$  is an algebraic space over  $S$ .
- (2) If  $X$  is an affine open covering.

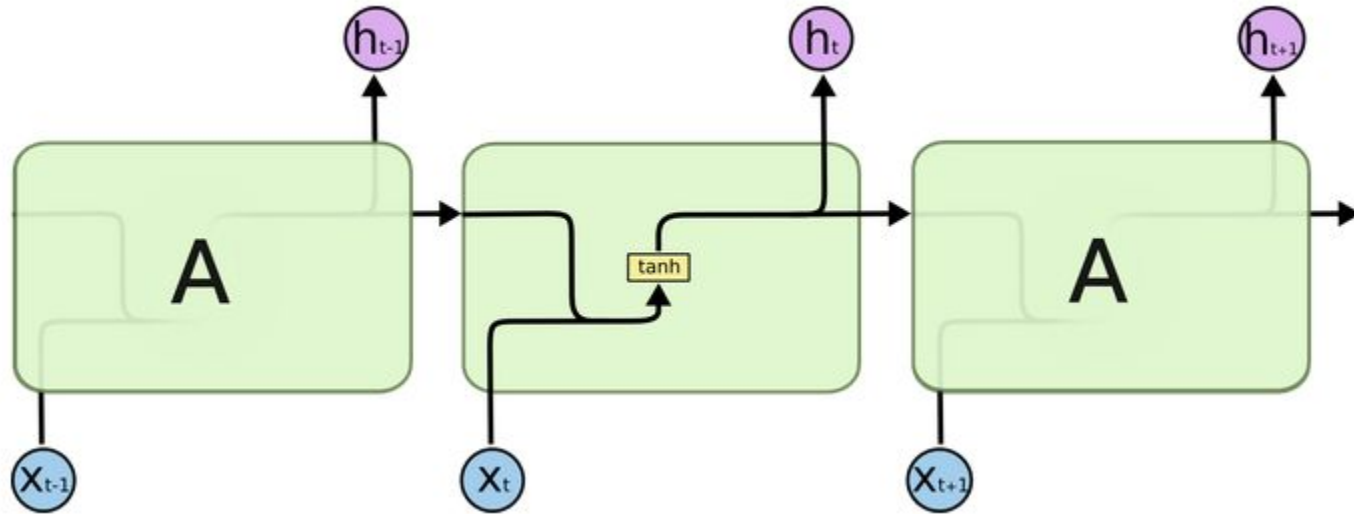
Consider a common structure on  $X$  and  $X$  the functor  $\mathcal{O}_X(\mathcal{U})$  which is locally of finite type. □

Linux kernel (source code)

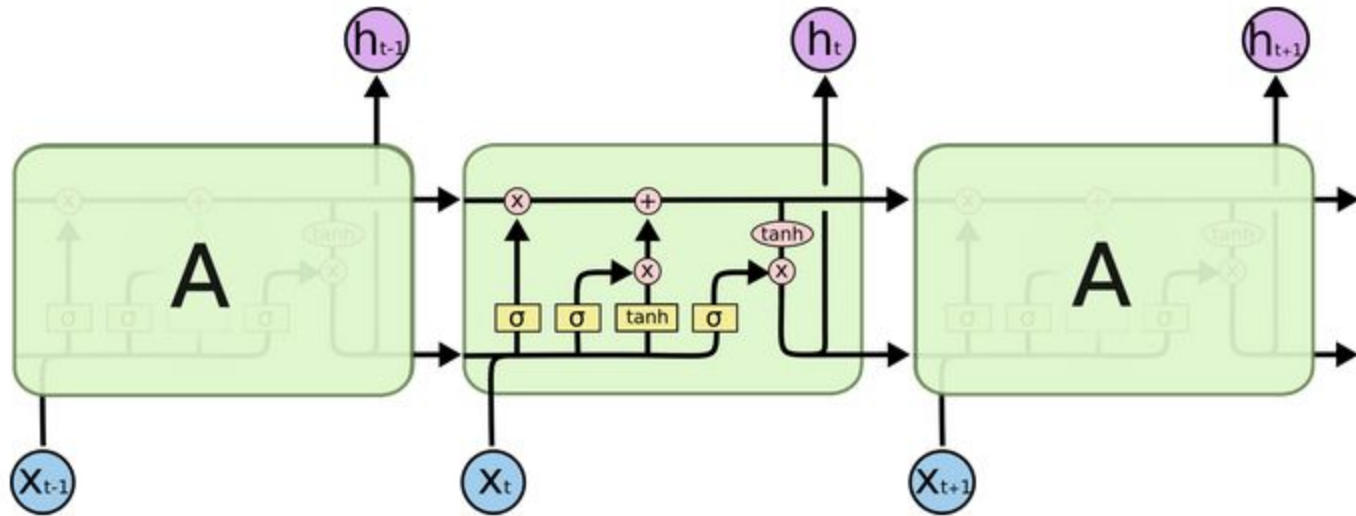
```
/*  
 * If this error is set, we will need anything right after that BSD.  
 */  
  
static void action_new_function(struct s_stat_info *wb)  
{  
    unsigned long flags;  
    int lel_idx_bit = e->edd, *sys & -((unsigned long) *FIRST_COMPAT);  
    buf[0] = 0x7fffffff & (bit << 4);  
    min(inc, slist->bytes);  
    printk(KERN_WARNING "Memory allocated %02x/%02x, "  
           "original MLL instead\n"),  
           min(min(multi_run - s->len, max) * num_data_in),  
           frame_pos, sz + first_seg);  
    div_u64_w(val, inb_p);  
    spin_unlock(&disk->queue_lock);  
    mutex_unlock(&s->sock->mutex);  
    mutex_unlock(&func->mutex);  
    return disassemble(info->pending_bh);  
}
```

Source: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

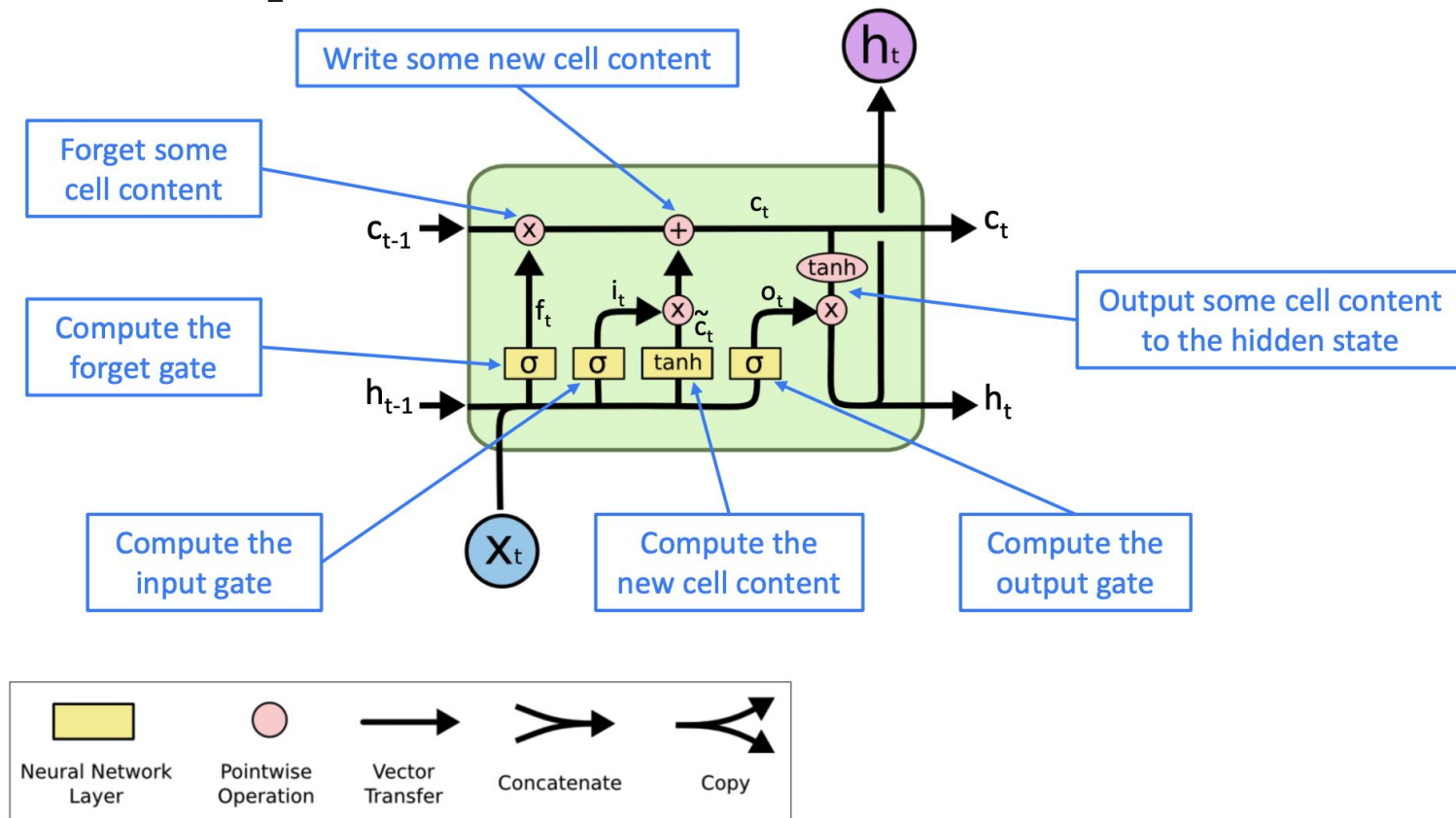
# Vanilla RNN



# LSTM



# LSTM: quick overview



# LSTM: quick overview



**Forget gate:** controls what is kept vs forgotten, from previous cell state

**Input gate:** controls what parts of the new cell content are written to cell

**Output gate:** controls what parts of cell are output to hidden state

**New cell content:** this is the new content to be written to the cell

**Cell state:** erase (“forget”) some content from last cell state, and write (“input”) some new cell content

**Hidden state:** read (“output”) some content from the cell

**Sigmoid function:** all gate values are between 0 and 1

$$f^{(t)} = \sigma \left( W_f h^{(t-1)} + U_f x^{(t)} + b_f \right)$$

$$i^{(t)} = \sigma \left( W_i h^{(t-1)} + U_i x^{(t)} + b_i \right)$$

$$o^{(t)} = \sigma \left( W_o h^{(t-1)} + U_o x^{(t)} + b_o \right)$$

$$\tilde{c}^{(t)} = \tanh \left( W_c h^{(t-1)} + U_c x^{(t)} + b_c \right)$$

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

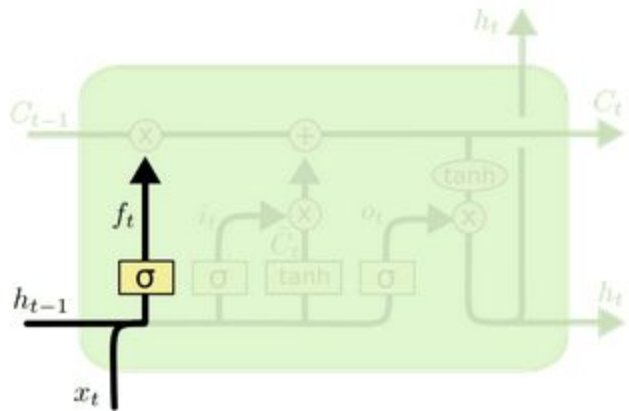
$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

All these are vectors of same length  $n$

Gates are applied using element-wise product

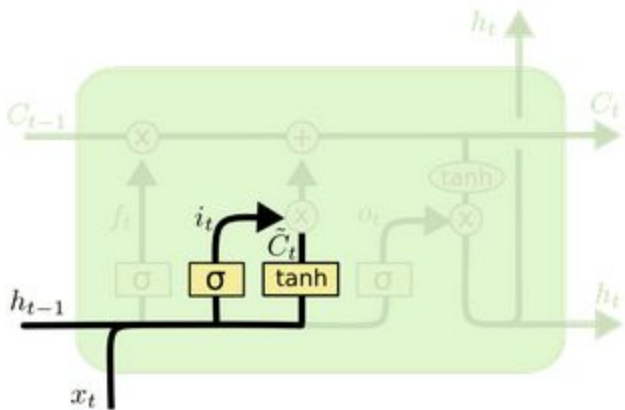


# LSTM: quick overview



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

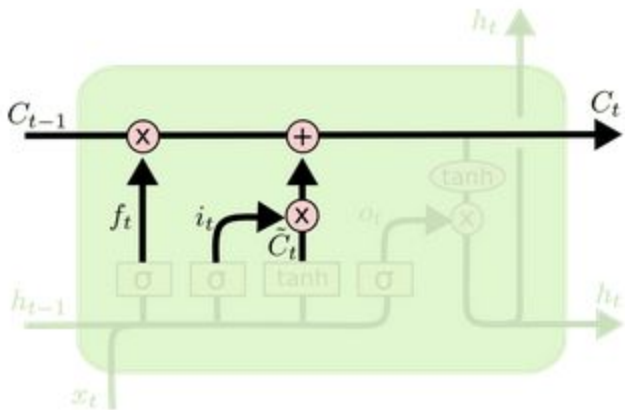
# LSTM: quick overview



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

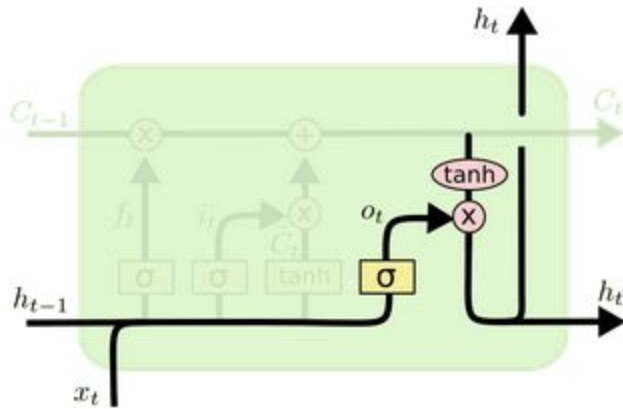
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

# LSTM: quick overview



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

# LSTM: quick overview



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$



# LSTM: with formulas

**Forget gate:** controls what is kept vs forgotten, from previous cell state

**Input gate:** controls what parts of the new cell content are written to cell

**Output gate:** controls what parts of cell are output to hidden state

**New cell content:** this is the new content to be written to the cell

**Cell state:** erase (“forget”) some content from last cell state, and write (“input”) some new cell content

**Hidden state:** read (“output”) some content from the cell

**Sigmoid function:** all gate values are between 0 and 1

$$f^{(t)} = \sigma \left( W_f h^{(t-1)} + U_f x^{(t)} + b_f \right)$$

$$i^{(t)} = \sigma \left( W_i h^{(t-1)} + U_i x^{(t)} + b_i \right)$$

$$o^{(t)} = \sigma \left( W_o h^{(t-1)} + U_o x^{(t)} + b_o \right)$$

$$\tilde{c}^{(t)} = \tanh \left( W_c h^{(t-1)} + U_c x^{(t)} + b_c \right)$$

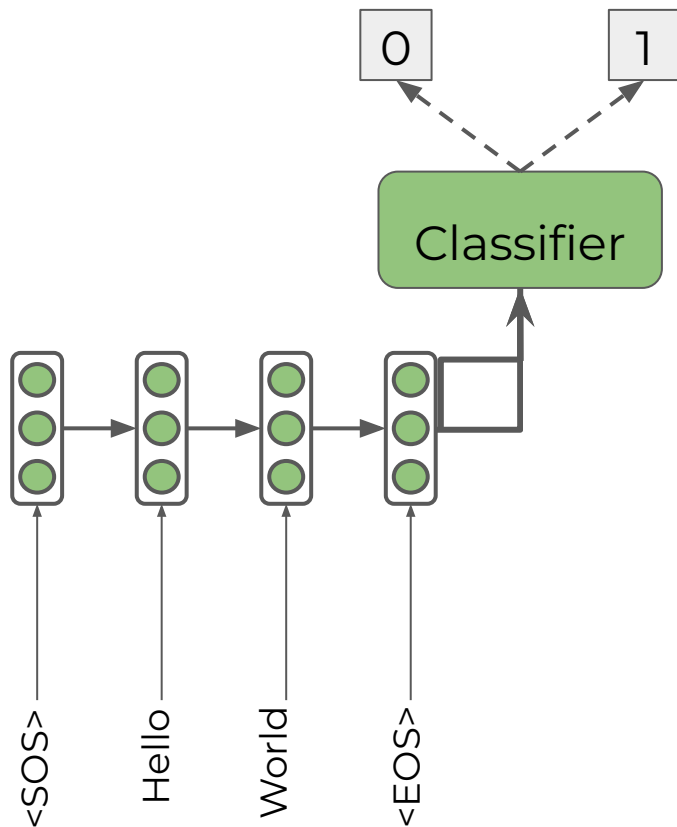
$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

All these are vectors of same length  $n$

Gates are applied using element-wise product

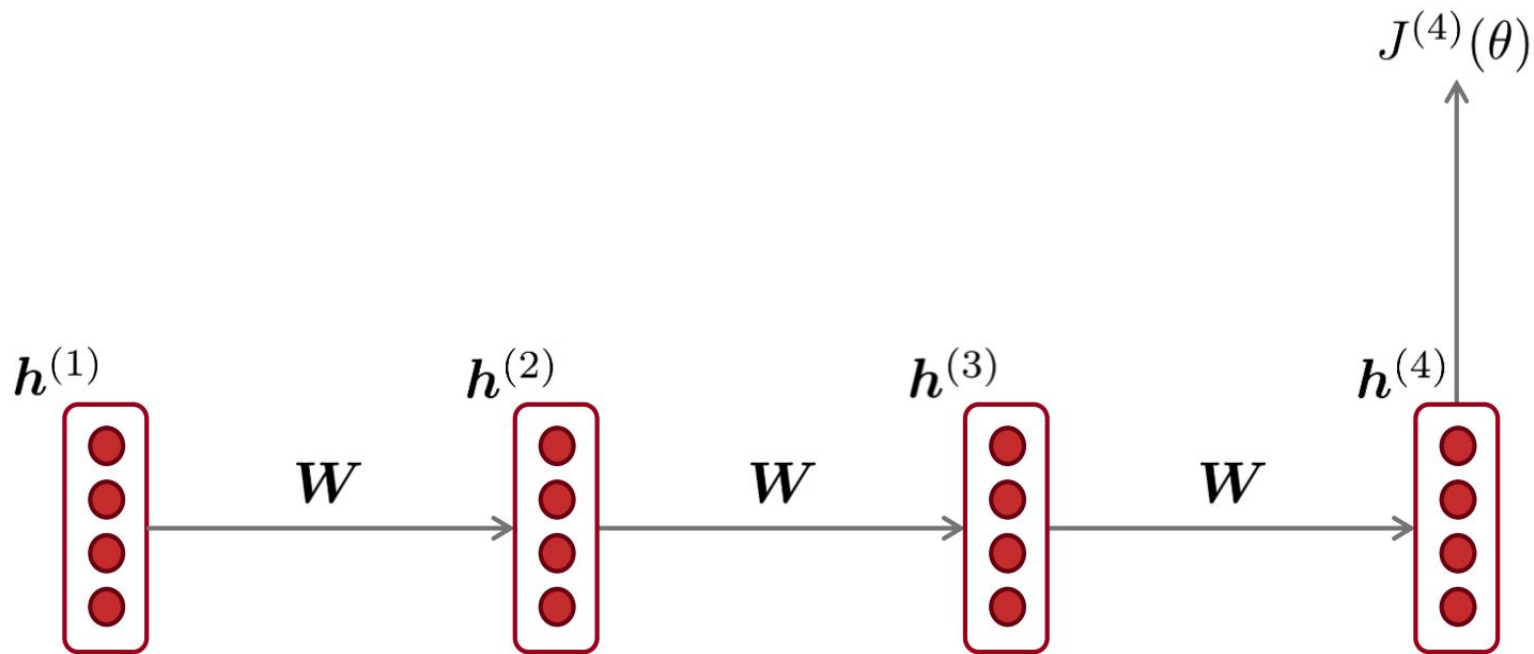
# RNN as encoder for sequential data



RNNs can be used to encode an input sequence in a fixed size vector.

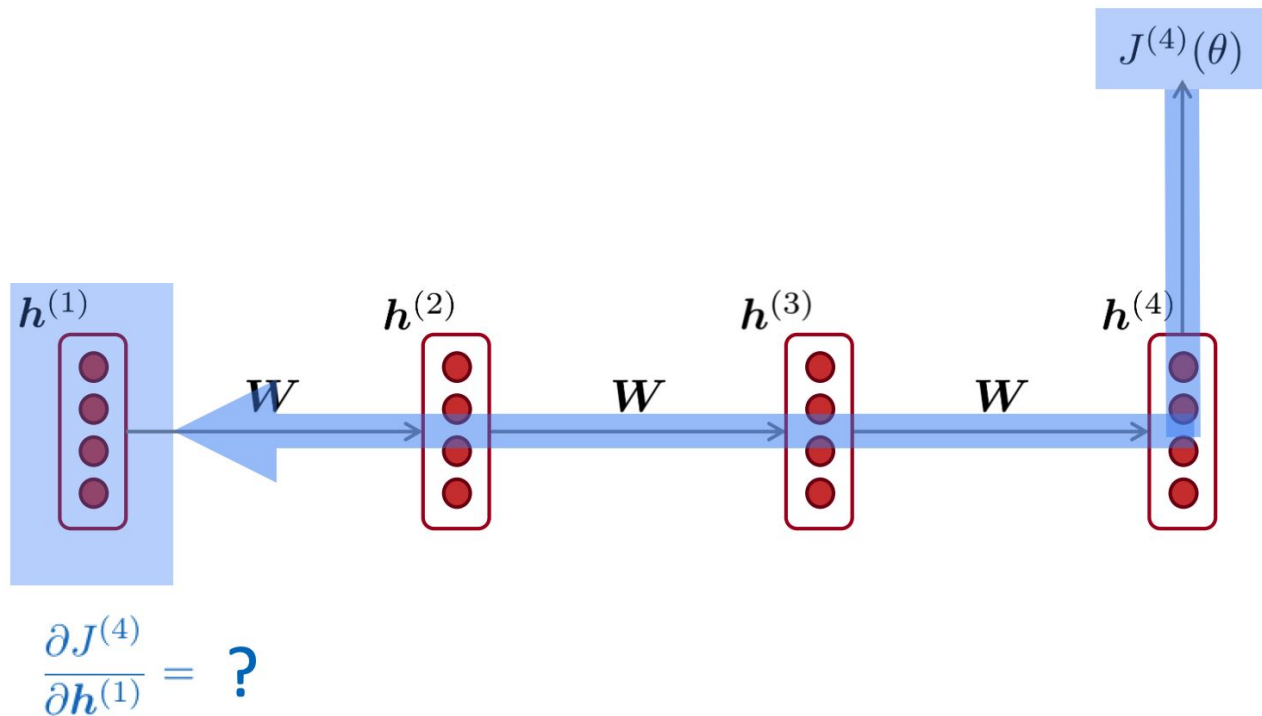
This vector can be treated as a representation of input sequence.

# Vanishing gradient problem





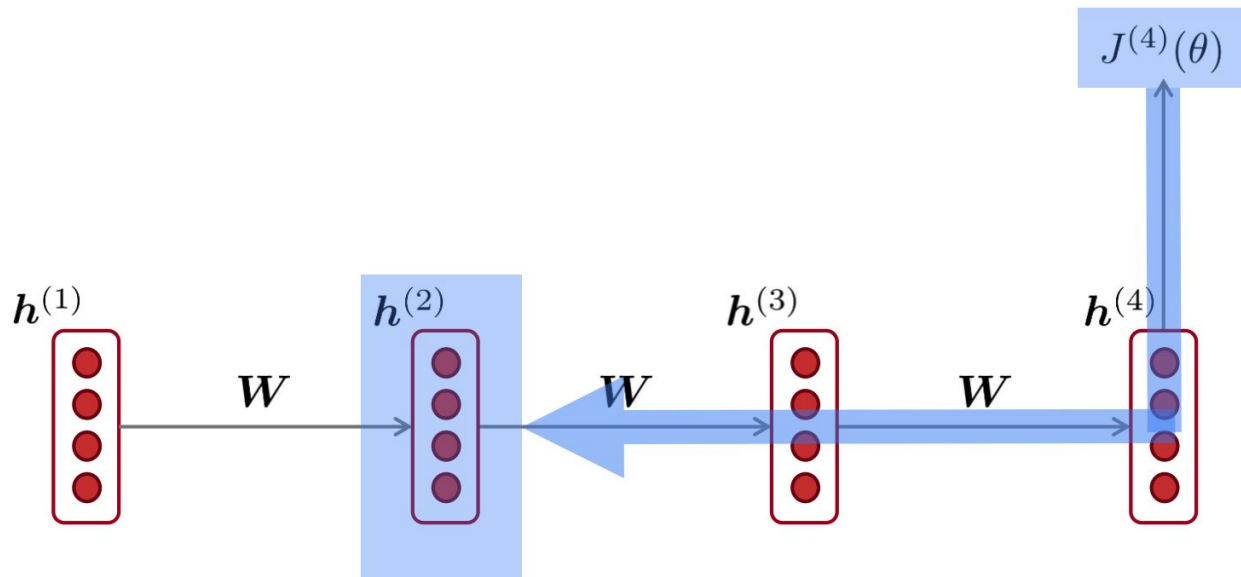
# Vanishing gradient problem







# Vanishing gradient problem

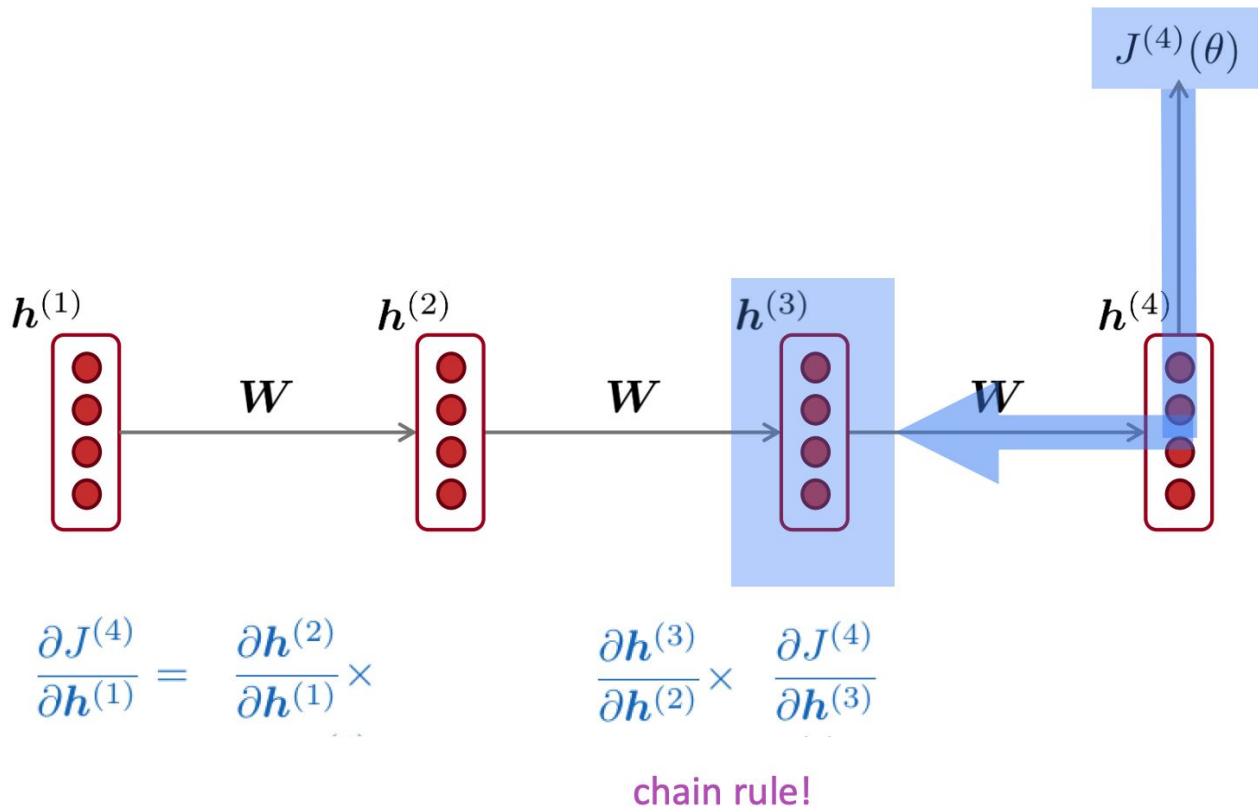


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial J^{(4)}}{\partial h^{(2)}}$$

chain rule!

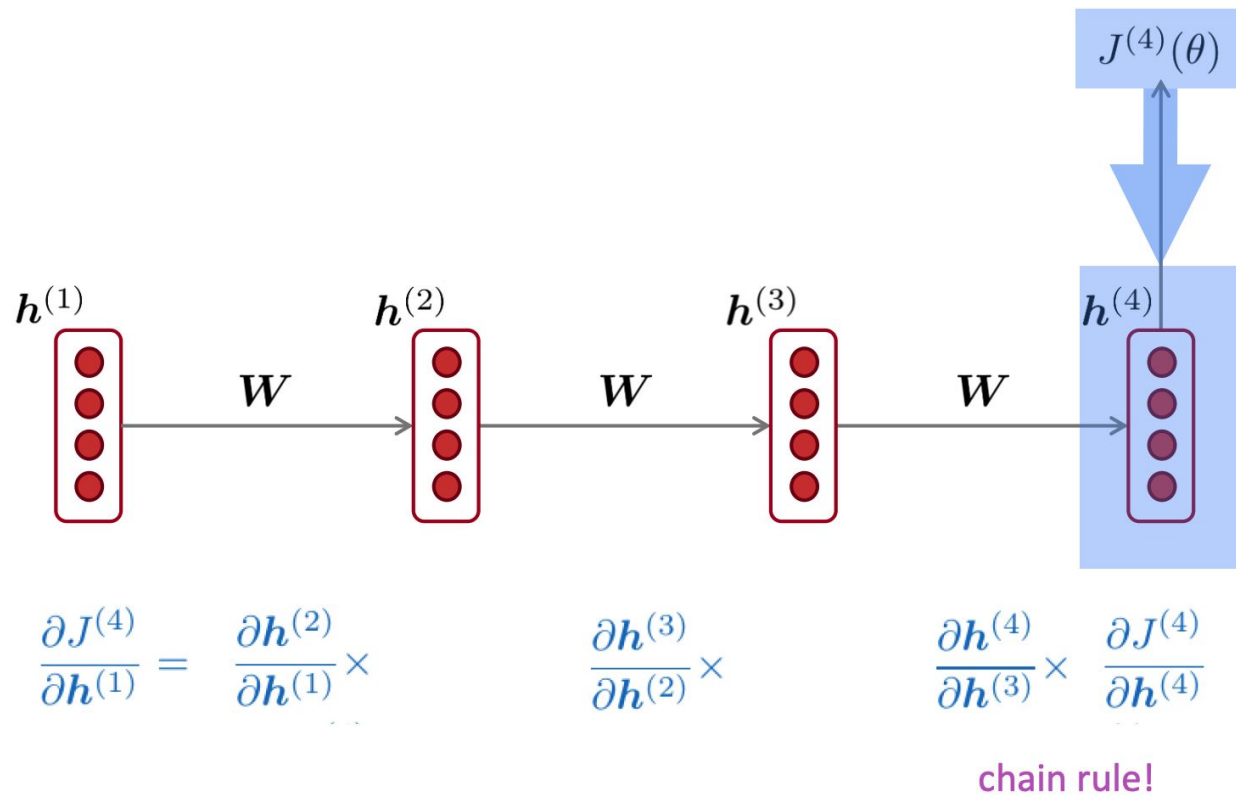


# Vanishing gradient problem





# Vanishing gradient problem

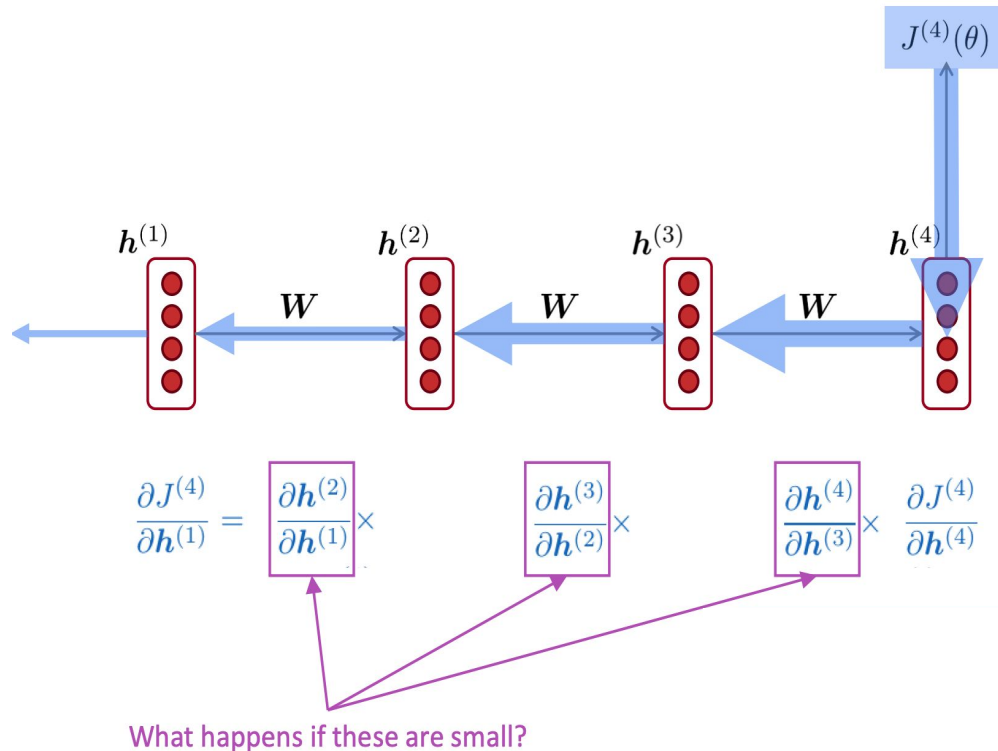




# Vanishing gradient problem

Vanishing gradient problem:

When the derivatives are small, the gradient signal gets smaller and smaller as it backpropagates further



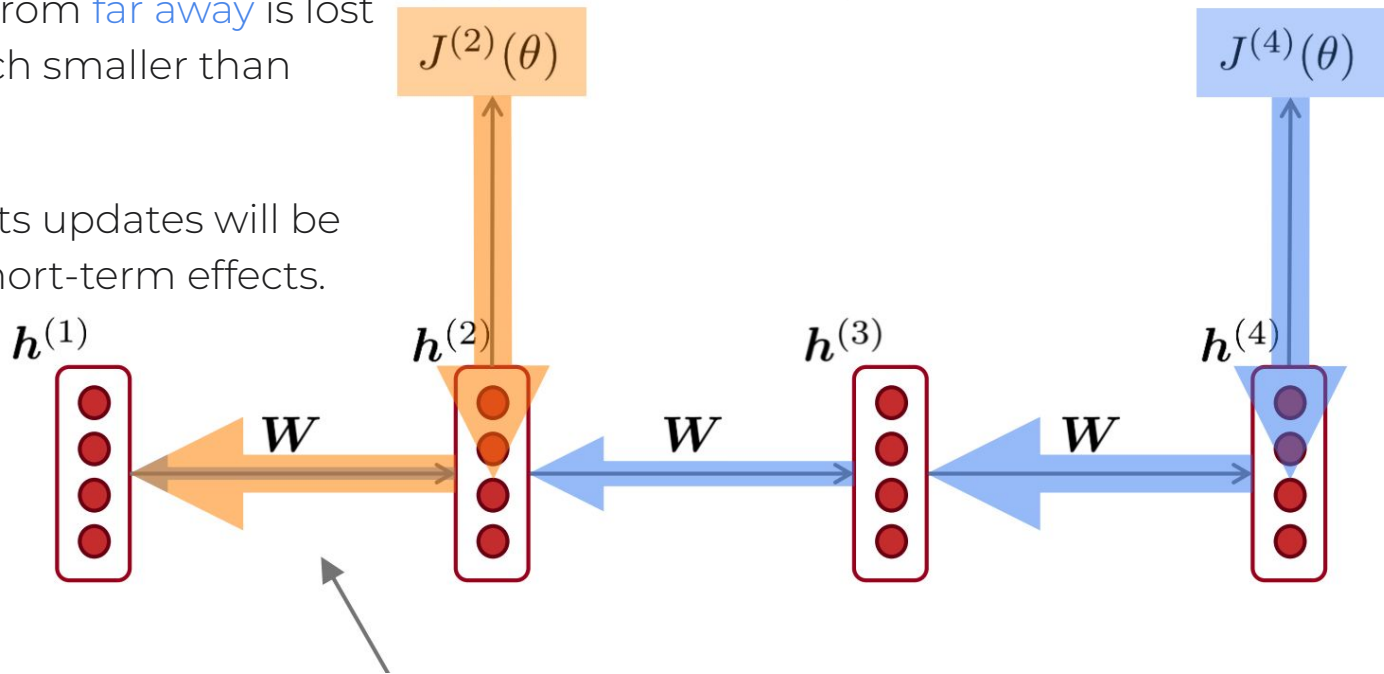
More info: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013  
<http://proceedings.mlr.press/v28/pascanu13.pdf>



# Vanishing gradient problem

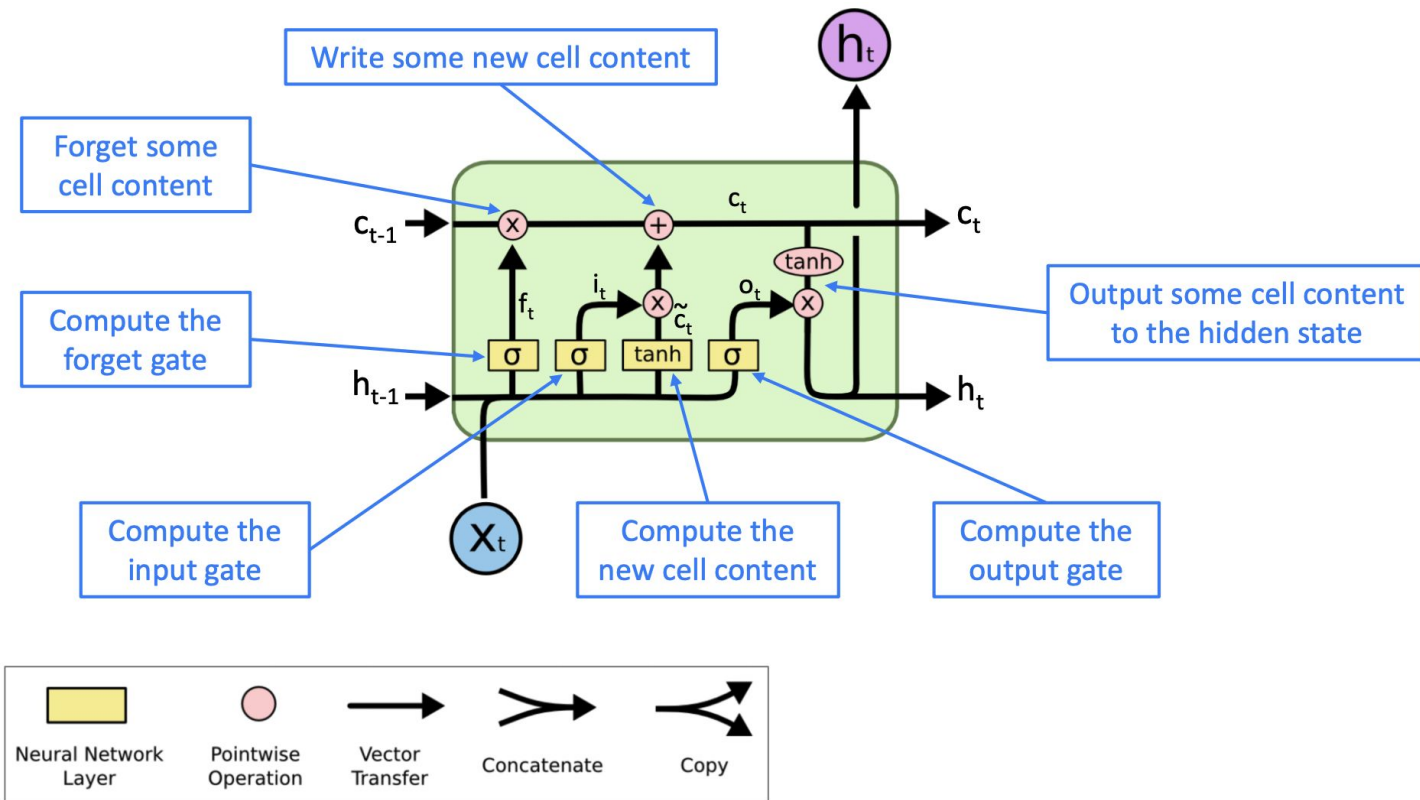
Gradient signal from **far away** is lost because it's much smaller than from **close-by**.

So model weights updates will be based only on short-term effects.





# Vanishing gradient: LSTM





# Vanishing gradient: LSTM

**Forget gate:** controls what is kept vs forgotten, from previous cell state

**Input gate:** controls what parts of the new cell content are written to cell

**Output gate:** controls what parts of cell are output to hidden state

**New cell content:** this is the new content to be written to the cell

**Cell state:** erase (“forget”) some content from last cell state, and write (“input”) some new cell content

**Hidden state:** read (“output”) some content from the cell

**Sigmoid function:** all gate values are between 0 and 1

$$f^{(t)} = \sigma \left( W_f h^{(t-1)} + U_f x^{(t)} + b_f \right)$$

$$i^{(t)} = \sigma \left( W_i h^{(t-1)} + U_i x^{(t)} + b_i \right)$$

$$o^{(t)} = \sigma \left( W_o h^{(t-1)} + U_o x^{(t)} + b_o \right)$$

$$\tilde{c}^{(t)} = \tanh \left( W_c h^{(t-1)} + U_c x^{(t)} + b_c \right)$$

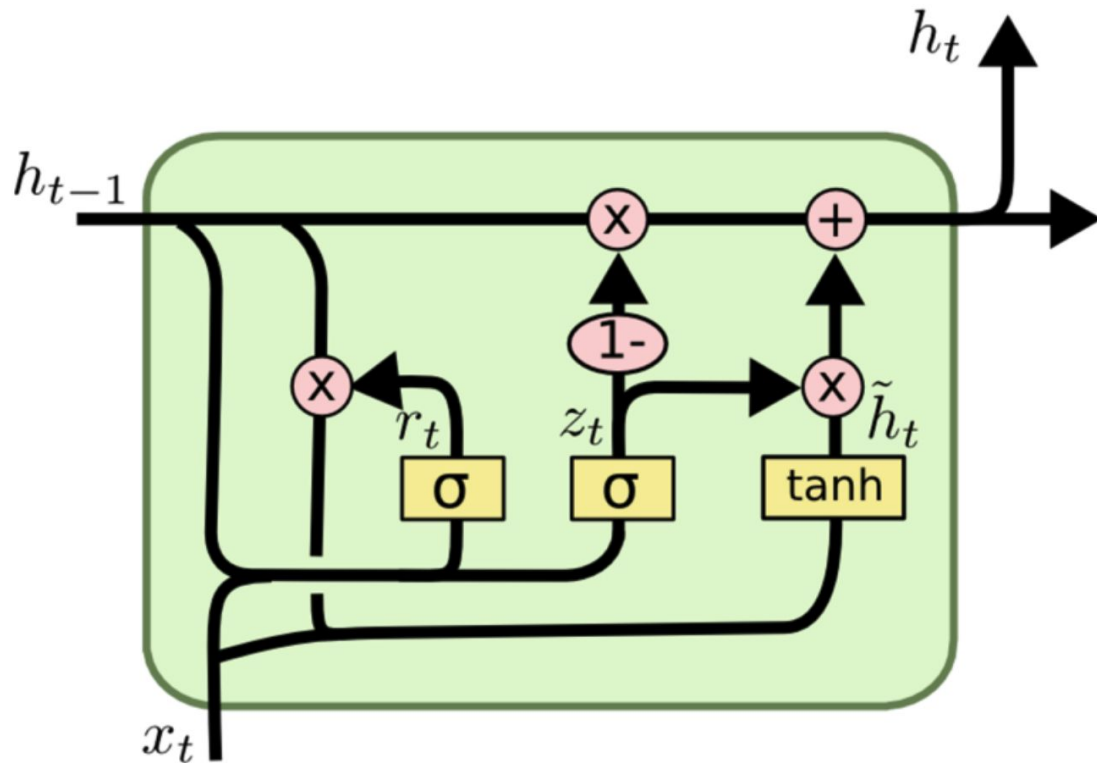
$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

Gates are applied using element-wise product

All these are vectors of same length  $n$

# Vanishing gradient: GDL







# Vanishing gradient: GRU

**Update gate:** controls what parts of hidden state are updated vs preserved

$$\mathbf{u}^{(t)} = \sigma \left( \mathbf{W}_u \mathbf{h}^{(t-1)} + \mathbf{U}_u \mathbf{x}^{(t)} + \mathbf{b}_u \right)$$

**Reset gate:** controls what parts of previous hidden state are used to compute new content

$$\mathbf{r}^{(t)} = \sigma \left( \mathbf{W}_r \mathbf{h}^{(t-1)} + \mathbf{U}_r \mathbf{x}^{(t)} + \mathbf{b}_r \right)$$

**New hidden state content:** reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

$$\tilde{\mathbf{h}}^{(t)} = \tanh \left( \mathbf{W}_h (\mathbf{r}^{(t)} \circ \mathbf{h}^{(t-1)}) + \mathbf{U}_h \mathbf{x}^{(t)} + \mathbf{b}_h \right)$$

$$\mathbf{h}^{(t)} = (1 - \mathbf{u}^{(t)}) \circ \mathbf{h}^{(t-1)} + \mathbf{u}^{(t)} \circ \tilde{\mathbf{h}}^{(t)}$$

**Hidden state:** update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

**How does this solve vanishing gradient?**

Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

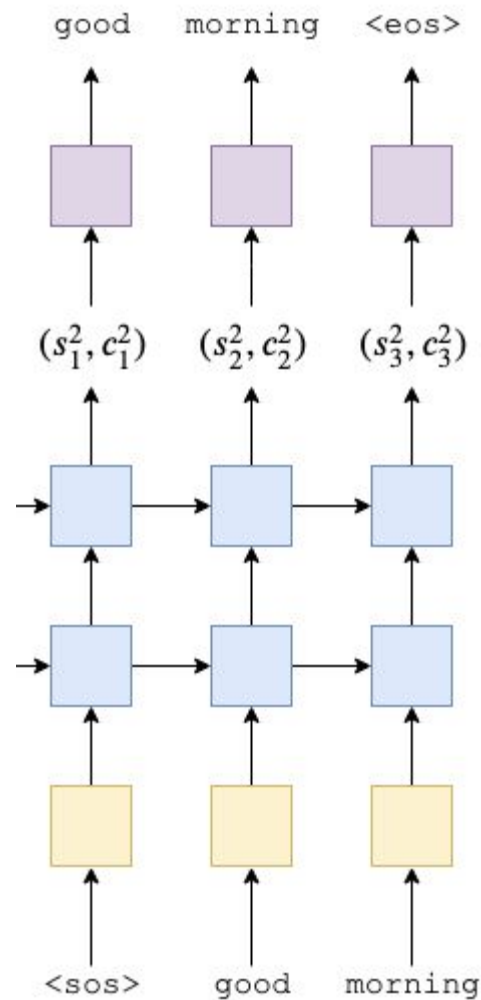
# Vanishing gradient: LSTM vs GRU



- LSTM and GRU are both great
  - GRU is quicker to compute and has fewer parameters than LSTM
  - There is no conclusive evidence that one consistently performs better than the other
  - LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)

# Outro

- RNN is a great choice for data with sequential structure
- Multi-layer RNN can also be of great use
- **Rule of thumb:** start with LSTM, but switch to GRU if you want something more efficient



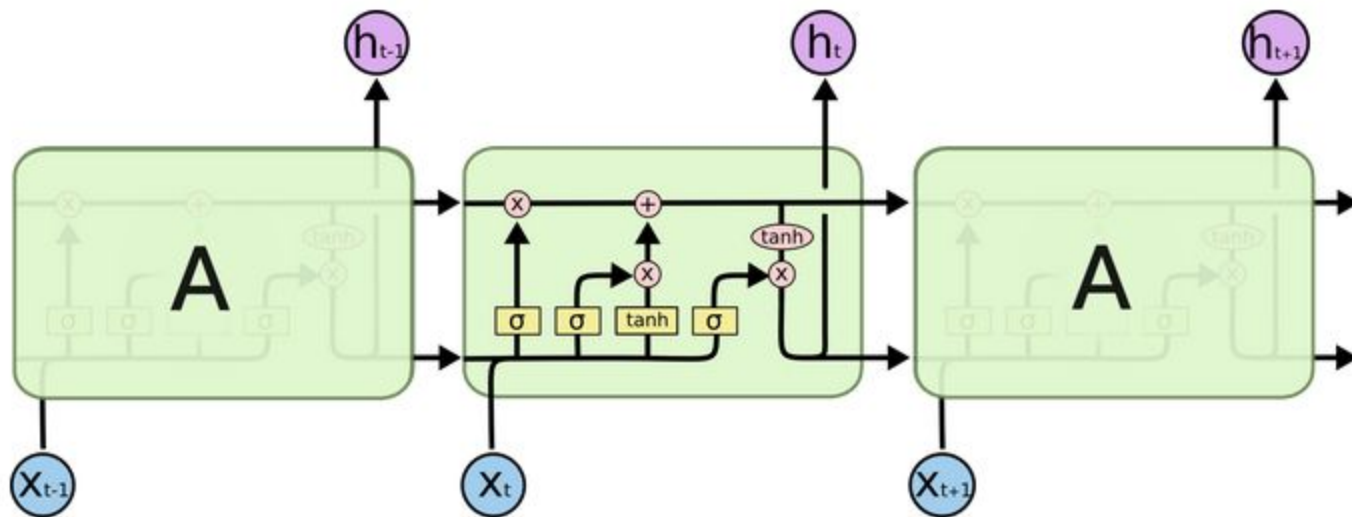
# Q & A



That's all. Feel free to ask any questions.

RNNs, we are coming. Time to generate some names!

# Recap: LSTM





# Exploding gradient problem

- If the gradient becomes too big, then the SGD update step becomes too big:
- This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)
- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

$$\theta^{new} = \theta^{old} - \underbrace{\alpha}_{\text{learning rate}} \underbrace{\nabla_{\theta} J(\theta)}_{\text{gradient}}$$



# Exploding gradient solution

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

---

**Algorithm 1** Pseudo-code for norm clipping

---

```
 $\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$   
if  $\|\hat{\mathbf{g}}\| \geq \textit{threshold}$  then  
     $\hat{\mathbf{g}} \leftarrow \frac{\textit{threshold}}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$   
end if
```

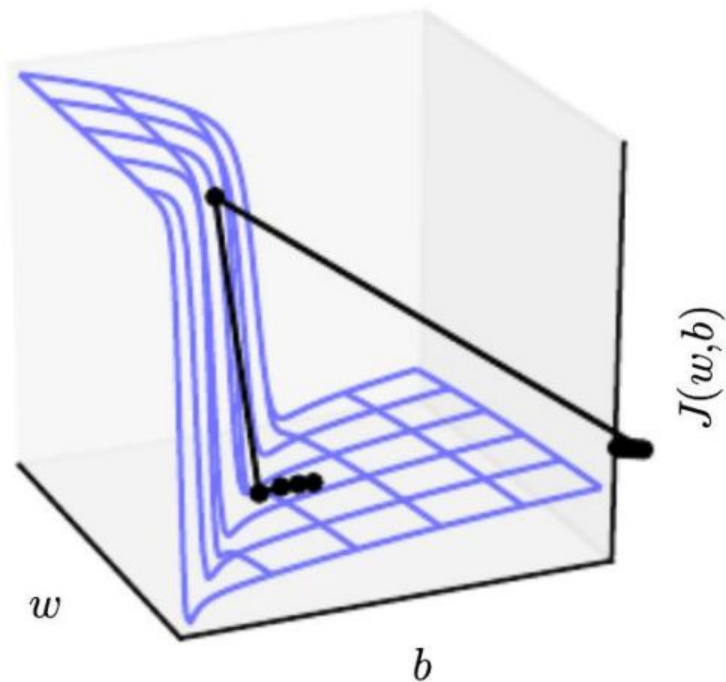
---

- Intuition: take a step in the same direction, but a smaller step

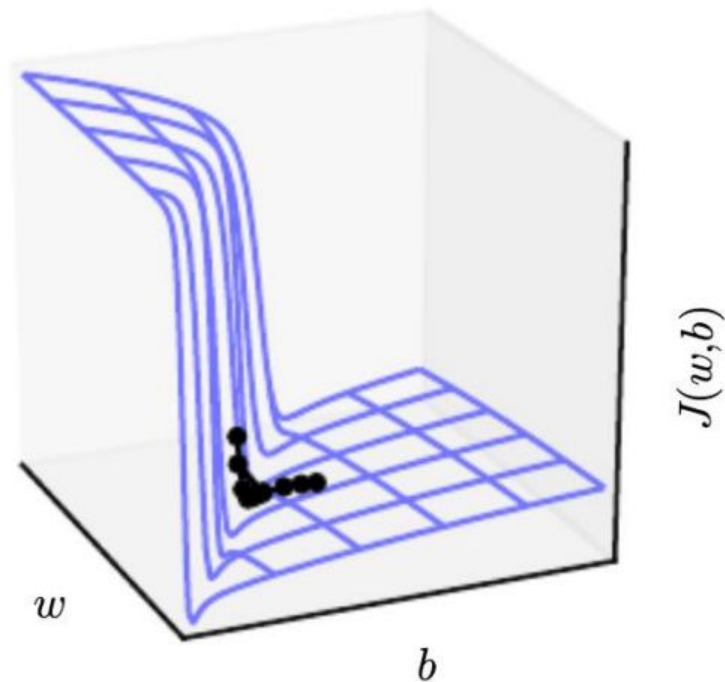
# Exploding gradient solution



Without clipping



With clipping





# Vanishing gradient: LSTM vs GRU



- LSTM and GRU are both great
  - GRU is quicker to compute and has fewer parameters than LSTM
  - There is no conclusive evidence that one consistently performs better than the other
  - LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)

**Rule of thumb:** start with LSTM, but switch to GRU if you want something more efficient



# Vanishing gradient in non-RNN

Vanishing gradient is present in all deep neural network architectures.

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution(but not actually for that problem):** dense connections (just like in DenseNet)

## Conclusion:

Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]. Gradients magnitude drops exponentially with connection length.

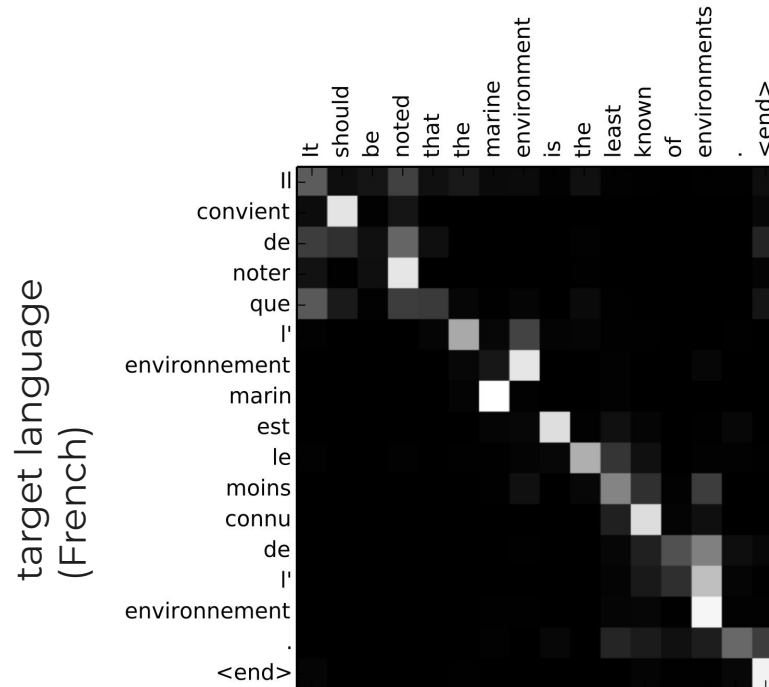
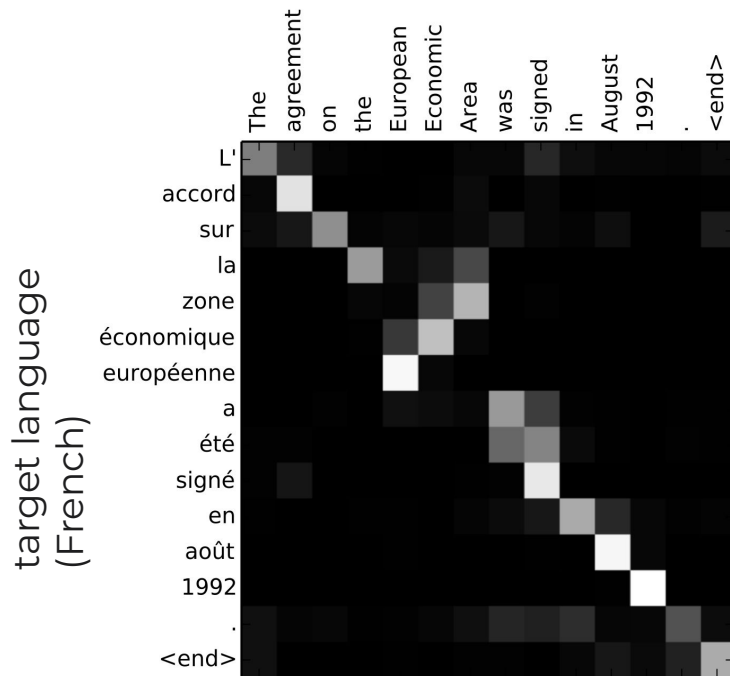
Source: "Learning Long-Term Dependencies with Gradient Descent is Difficult", Bengio et al. 1994, <http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf>

# Attention maps in translation



source language (English)

source language (English)





# Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural network architectures.

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution:** direct (or skip-) connections (just like in ResNet)

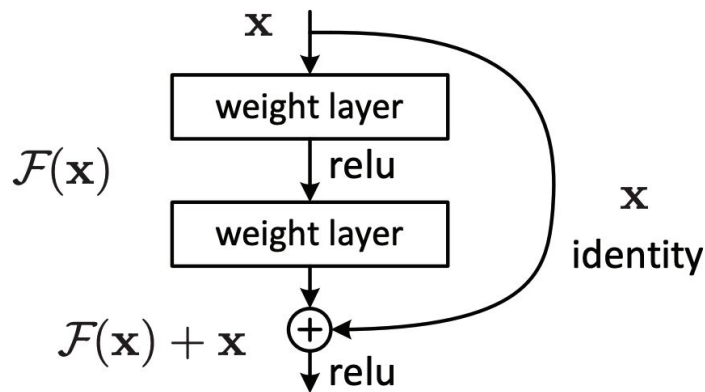


Figure 2. Residual learning: a building block.

Source: "Deep Residual Learning for Image Recognition", He et al, 2015.

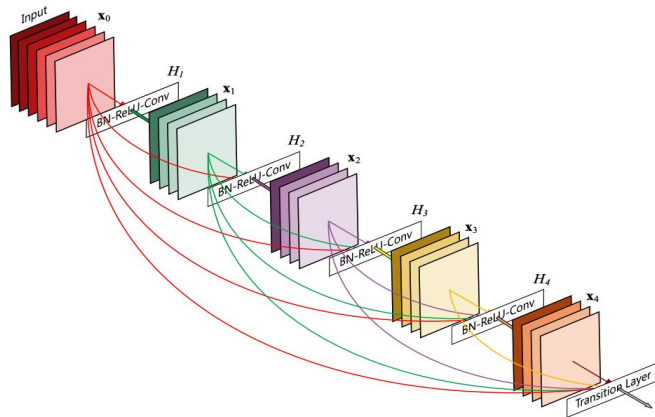
<https://arxiv.org/pdf/1512.03385.pdf>



# Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural network architectures.

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution:** dense connections (just like in DenseNet)

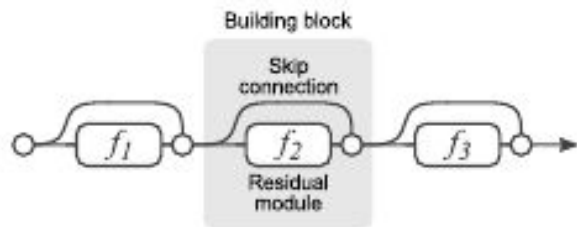


Source: "Densely Connected Convolutional Networks", Huang et al, 2017  
<https://arxiv.org/pdf/1608.06993.pdf>



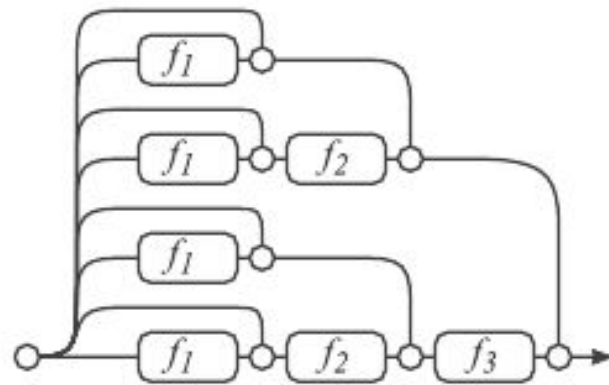
# Another view on ResNets and vanishing gradient

“Residual Networks Behave Like Ensembles of Relatively Shallow Networks”



(a) Conventional 3-block residual network

=



(b) Unraveled view of (a)

# Revise



1. RNN intuitions
2. Language models
3. Memory concept: LSTM
4. RNN as encoder for sequential data
5. Vanishing gradient

# Thanks for attention!

Questions?



**girafe**  
**ai**

