UiO : Department of Informatics
University of Oslo

INF 5400 Machine learning for image classification

Lecture 11: Recurrent neural networks

March 27 , 2019

Tollef Jahren

Outline

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNN’s
- Training Recurrent Neural Networks
- Simple examples
- Advanced models
- Advanced examples

About today

Goals:

- Understand how to build a recurrent neural network
- Understand why long range dependencies is a challenge for recurrent neural networks

Readings

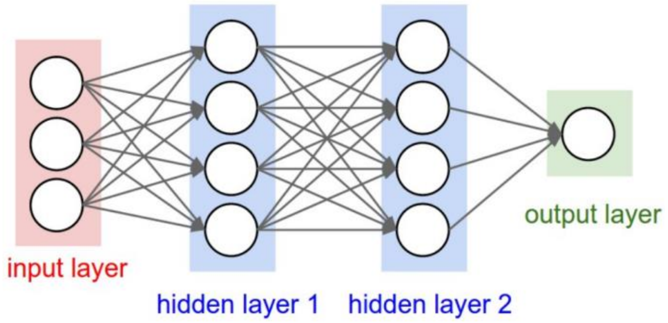
- Video:
  - cs231n: Lecture 10 | Recurrent Neural Networks
- Read:
  - The Unreasonable Effectiveness of Recurrent Neural Networks
- **Optional:**
  - Video: cs224d L8: Recurrent Neural Networks and Language Models
  - Video: cs224d L9: Machine Translation and Advanced Recurrent LSTMs and GRUs

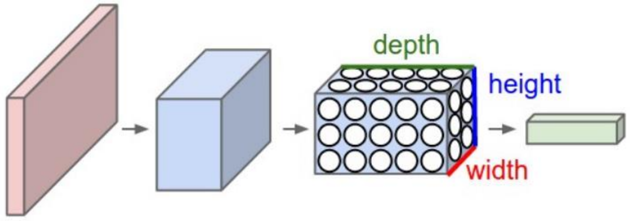
Progress

- **Overview (Feed forward and convolution neural networks)**
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNN’s
- Training Recurrent Neural Networks
- Simple examples
- Advanced models
- Advanced examples

Overview

- **What have we learnt so far?**
  - Fully connected neural network
  - Convolutional neural network
- **What worked well with these?**
  - Fully connected neural network works well when the input data is structured
  - Convolutional neural network works well on images
- **What does not work well?**
  - Processing data with unknown length



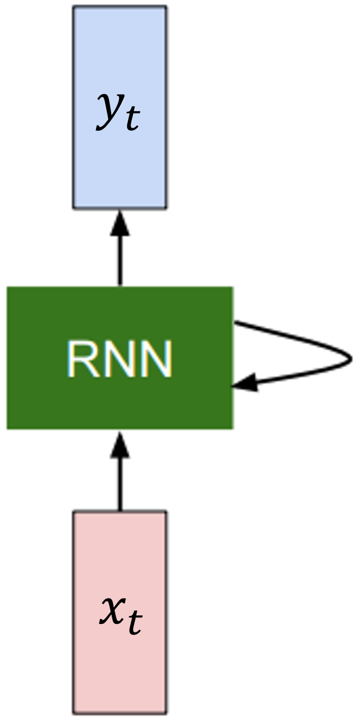


Progress

- Overview (Feed forward and convolution neural networks)
- **Vanilla Recurrent Neural Network (RNN)**
- Input-output structure of RNN’s
- Training Recurrent Neural Networks
- Simple examples
- Advanced models
- Advanced examples

Recurrent Neural Network (RNN)

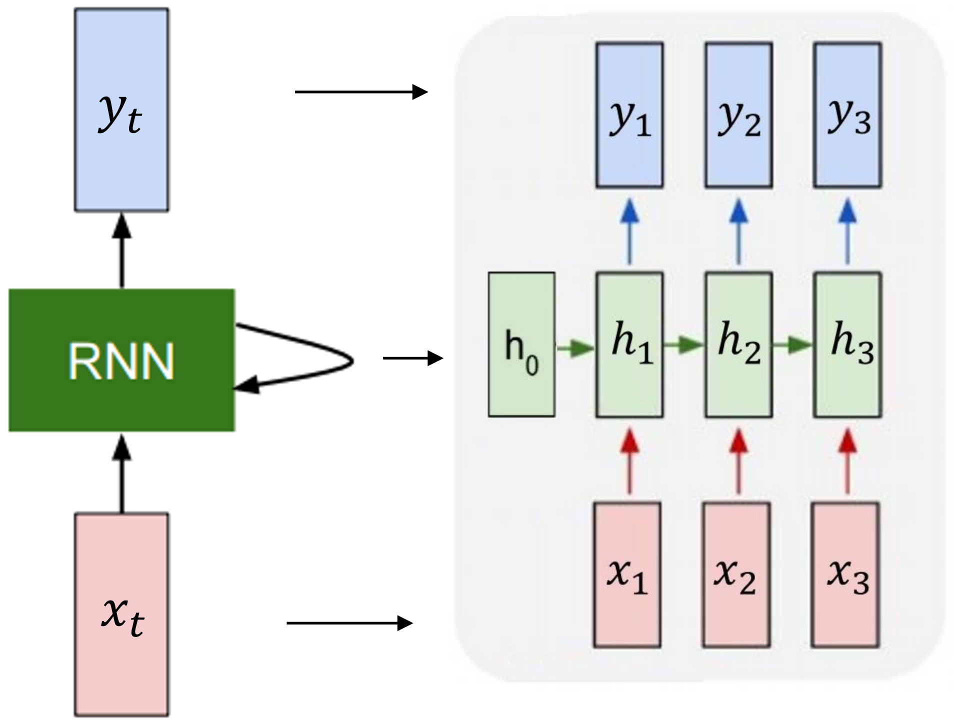
- Takes a new input
- Manipulate the state
- Reuse weights
- Gives a new output



Recurrent Neural Network (RNN)

- Unrolled view

- x_t : The input vector:
- h_t: The hidden state of the RNN
- y_t : The output vector:



Recurrent Neural Network (RNN)

- Recurrence formula:
  - Input vector: x_t
  - Hidden state vector: h_t

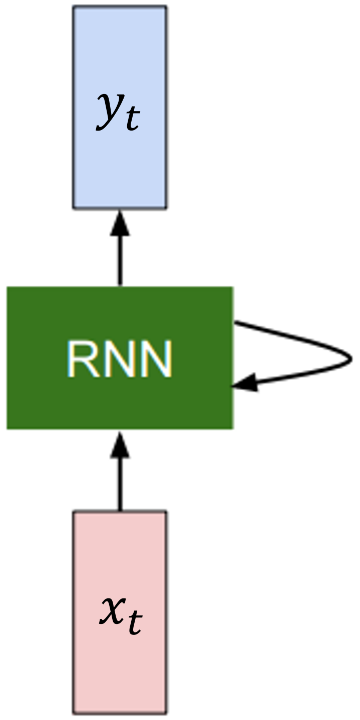
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function
with parameters W

old state

input vector at
some time step



Note: We use the same function and parameters for very “time” step

(Vanilla) Recurrent Neural Network

- Input vector: x_t
- Hidden state vector: h_t
- Output vector: y_t
- Weight matrices: W_{hh}, W_{hx}, W_{hy}

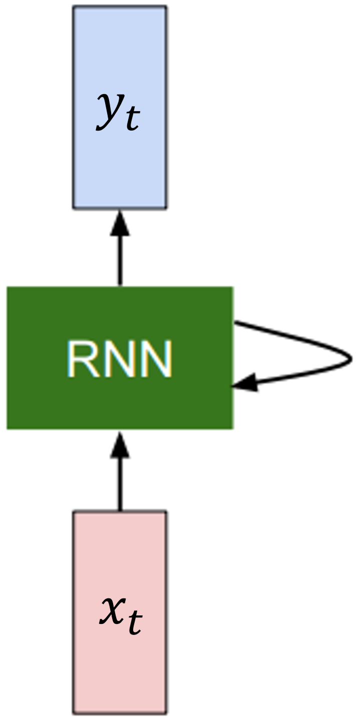
General form:

$$h_t = f_W(h_{t-1}, x_t)$$

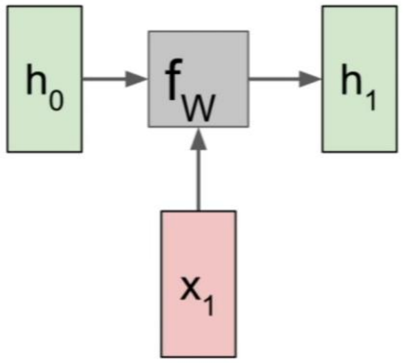
Vanilla RNN:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t + b)$$

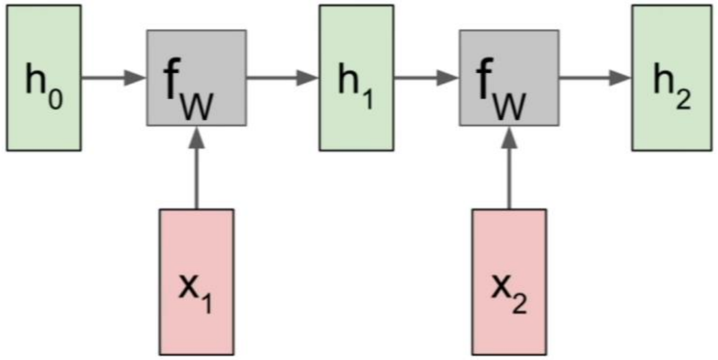
$$y_t = W_{hy}h_t$$



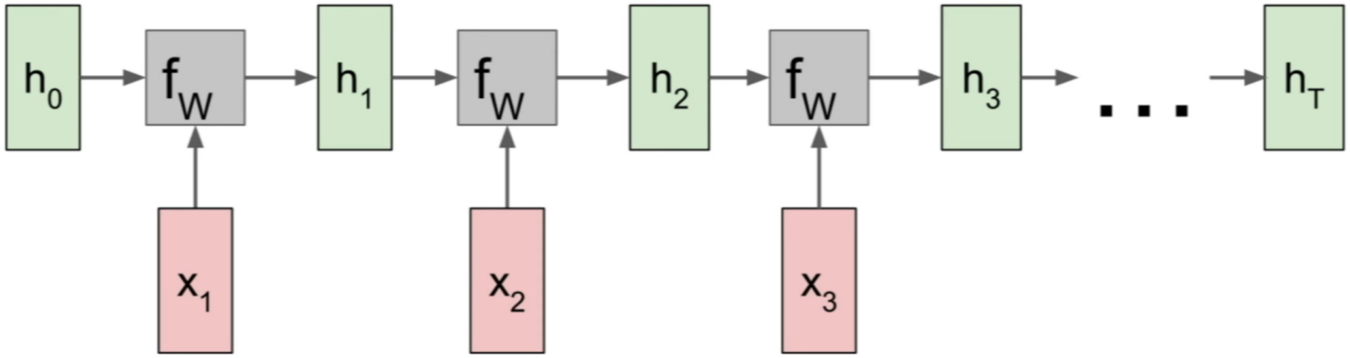
RNN: Computational Graph



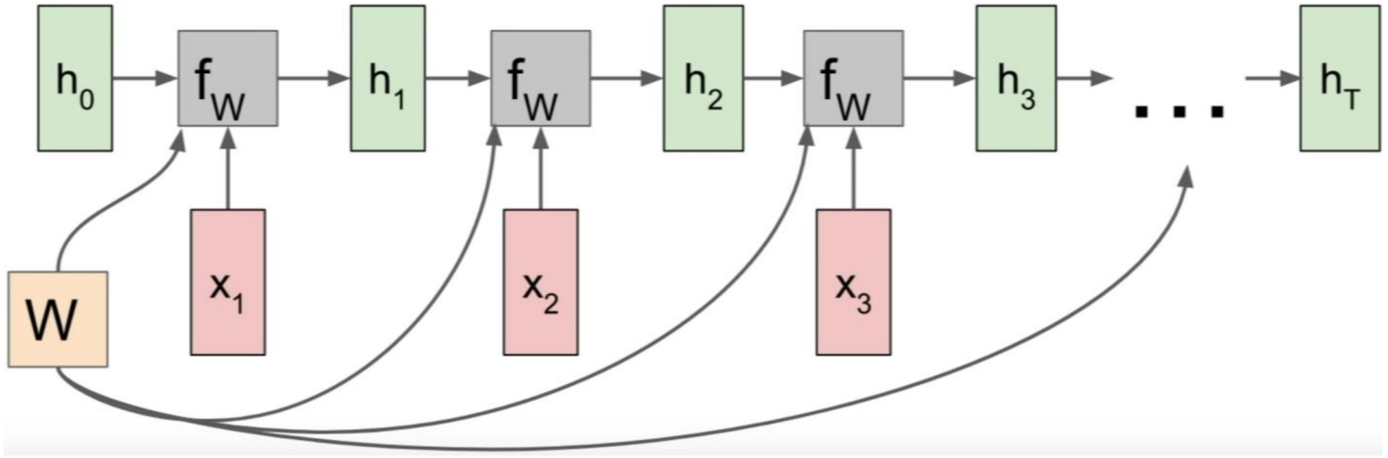
RNN: Computational Graph



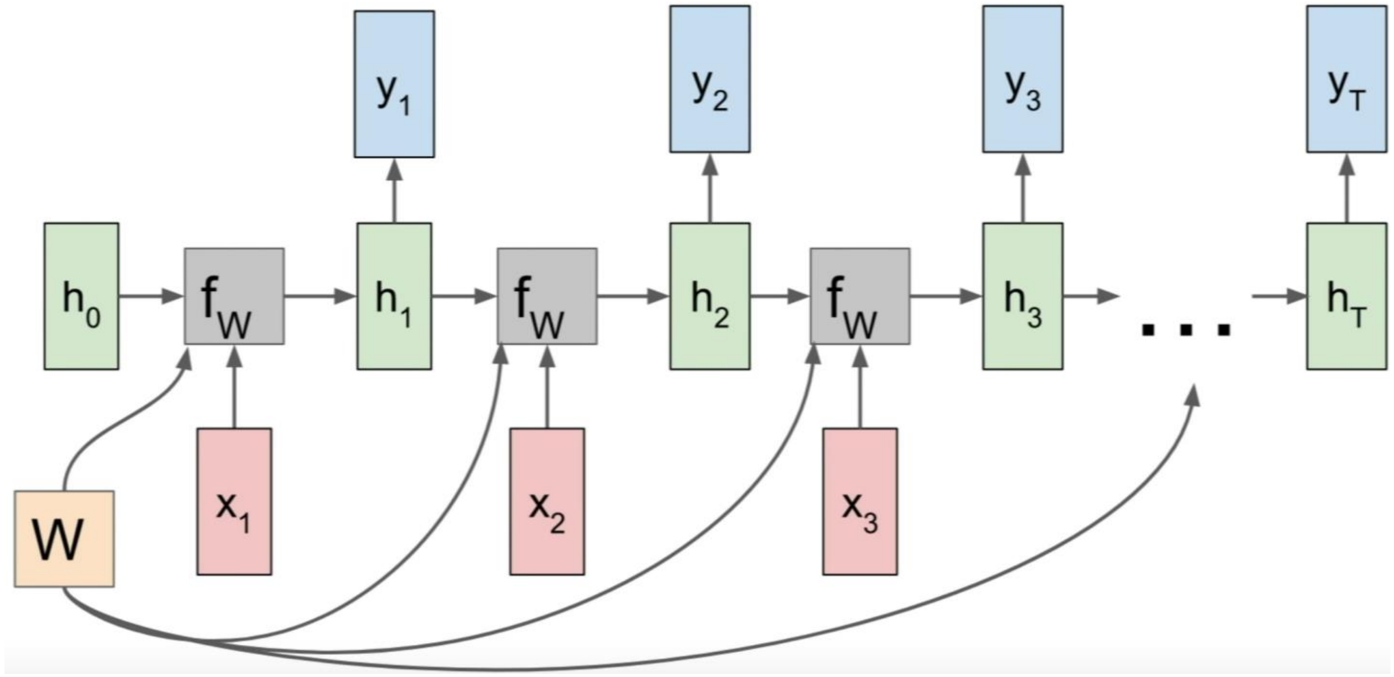
RNN: Computational Graph



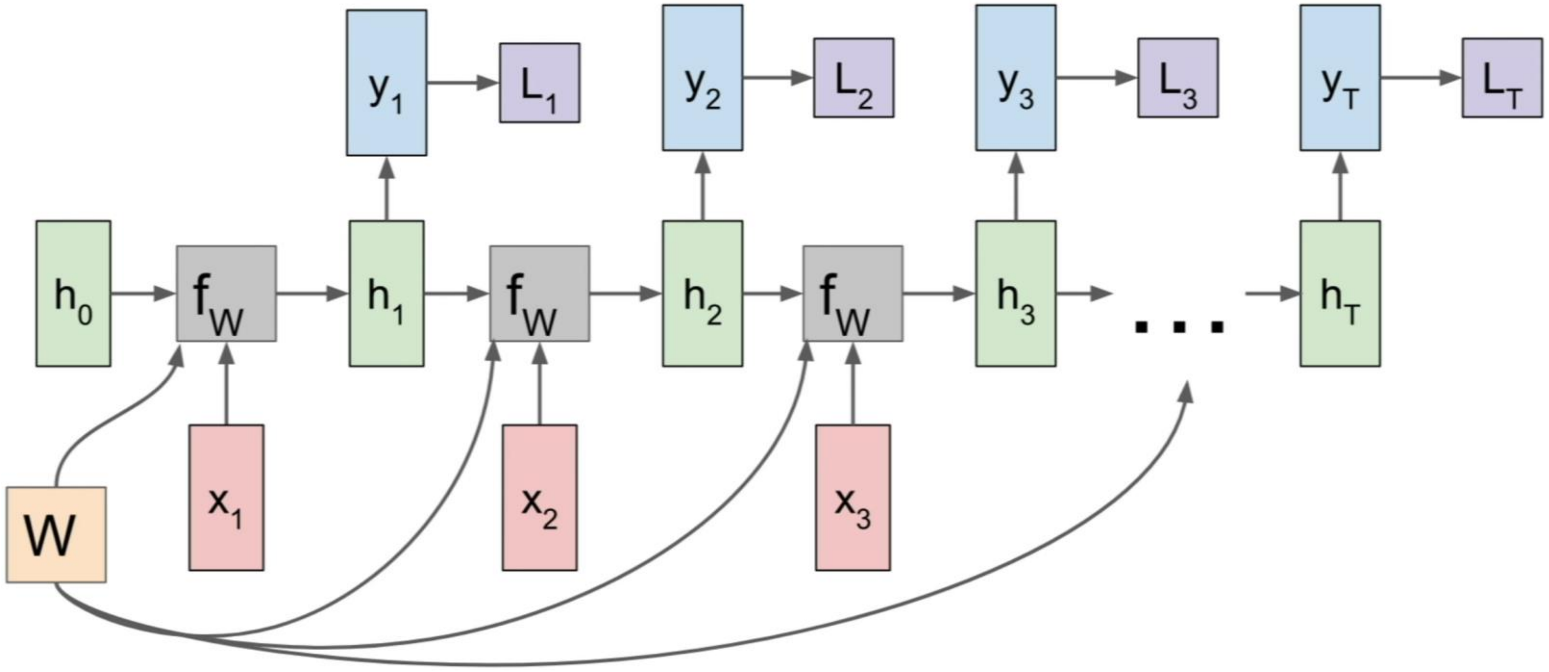
RNN: Computational Graph



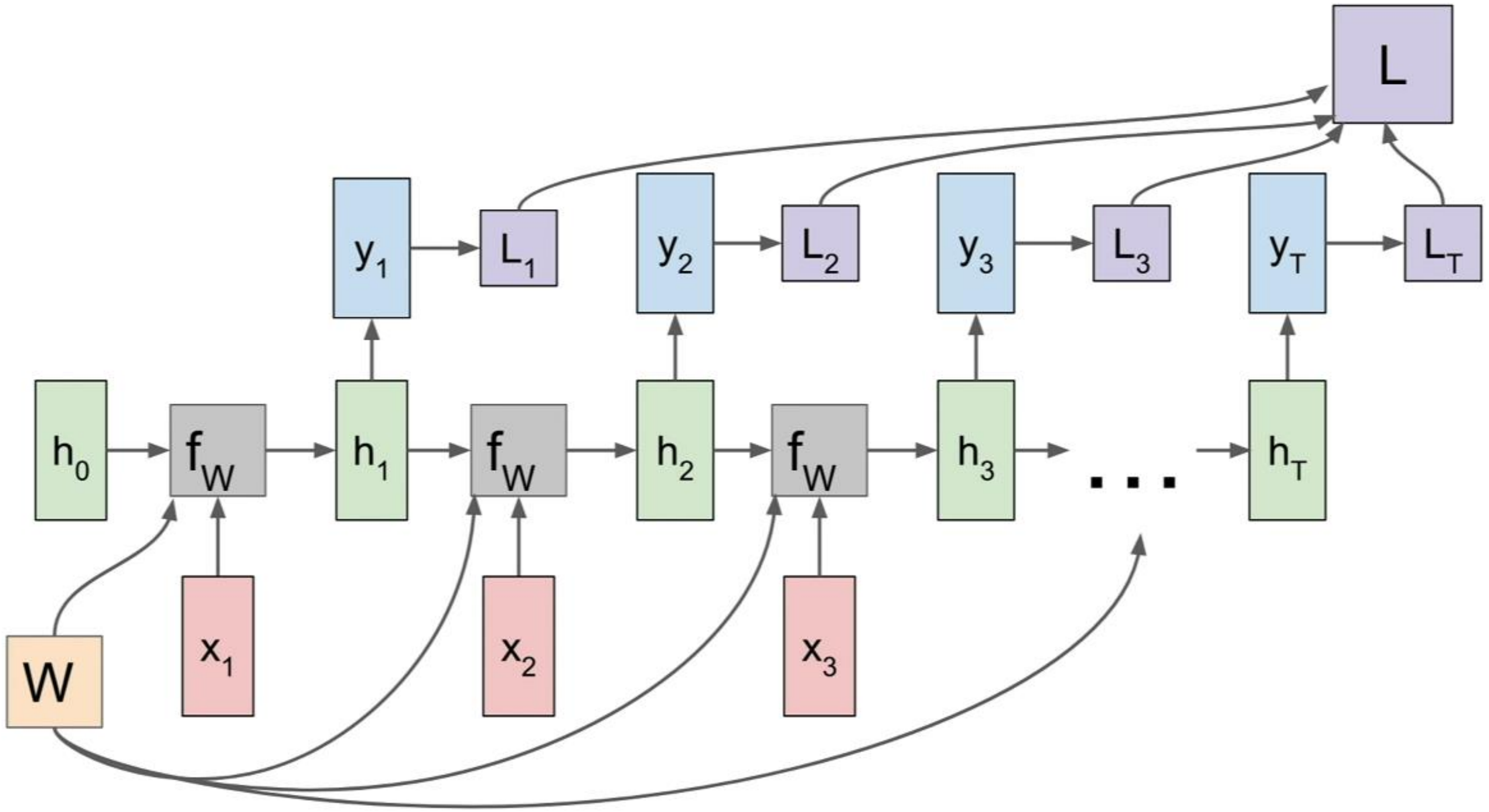
RNN: Computational Graph



RNN: Computational Graph

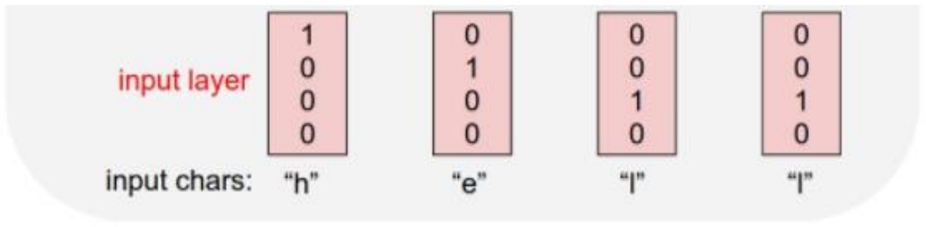


RNN: Computational Graph



Example: Language model

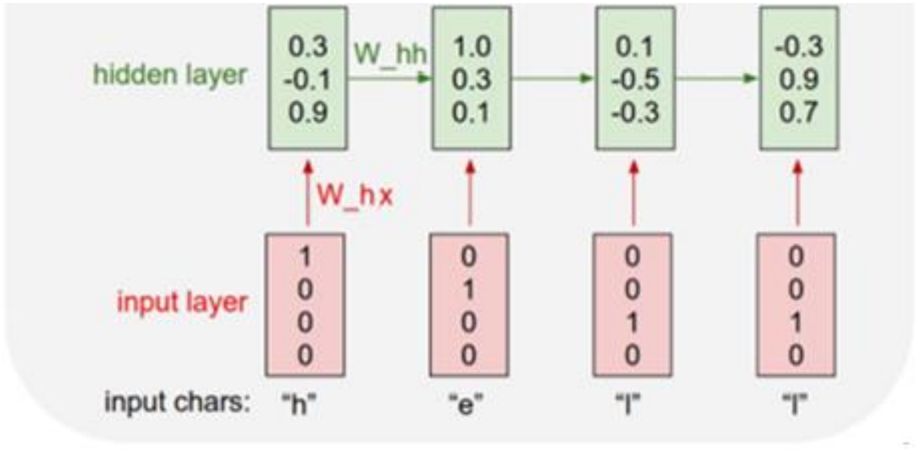
- Task: Predicting the next character
- Training sequence: “hello”
- Vocabulary:
  - [h, e, l, o]
- Encoding: Onehot



RNN: Predicting the next character

- Task: Predicting the next character
- Training sequence: “hello”
- Vocabulary:
  - [h, e, l, o]
- Encoding: Onehot
- Model:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t)$$

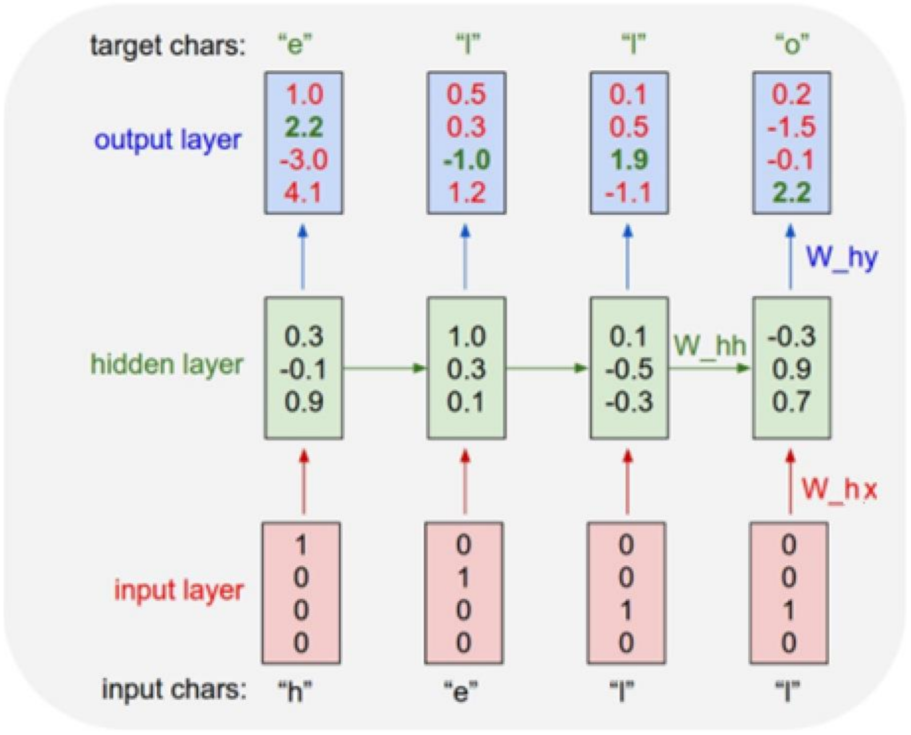


RNN: Predicting the next character

- Task: Predicting the next character
- Training sequence: “hello”
- Vocabulary:
  - [h, e, l, o]
- Encoding: Onehot
- Model:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t + b)$$

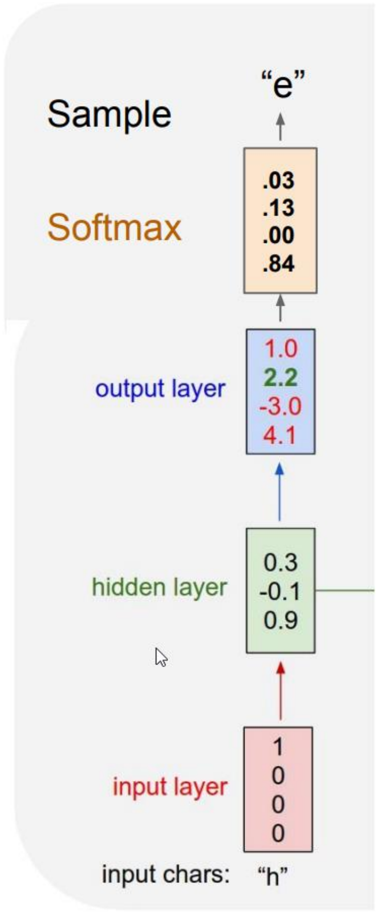
$$y_t = W_{hy}h_t$$



RNN: Predicting the next character

- Vocabulary:
  - [h, e, l, o]

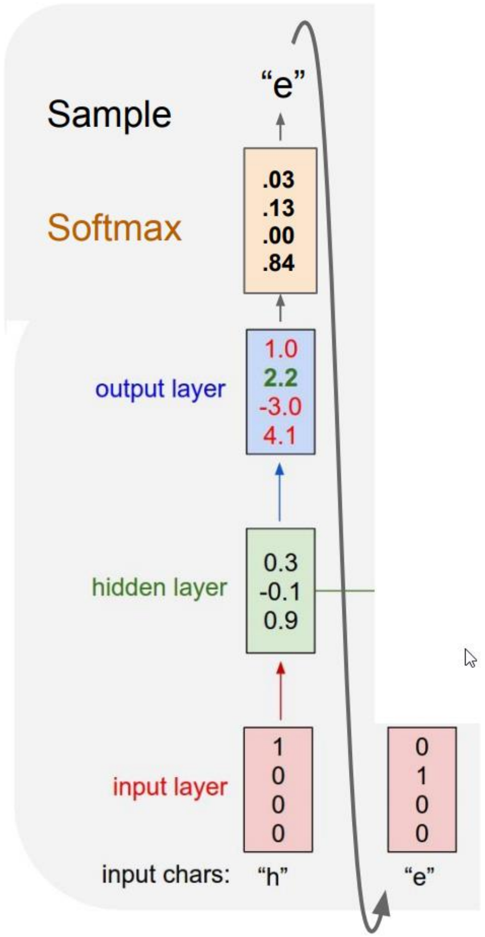
- At test time, we can sample from the model to synthesis new text.



RNN: Predicting the next character

- Vocabulary:
  - [h, e, l, o]

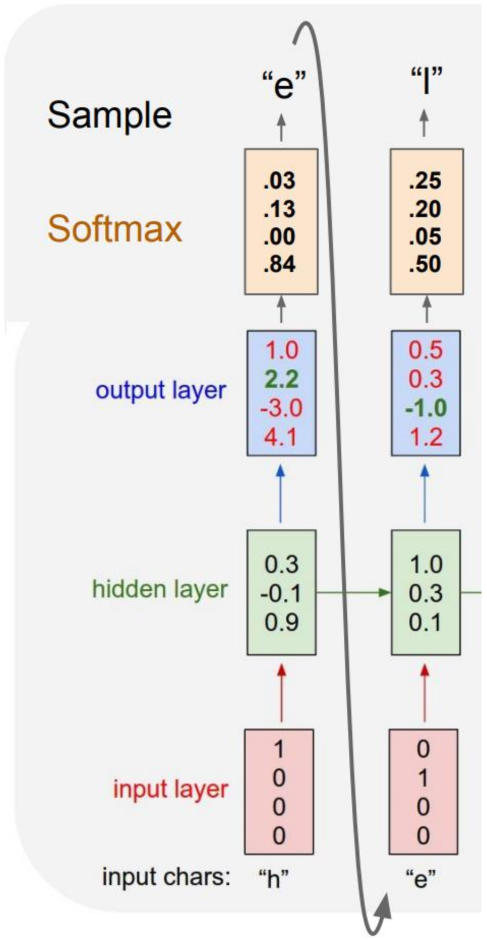
- At test time, we can sample from the model to synthesis new text.



RNN: Predicting the next character

- Vocabulary:
  - [h, e, l, o]

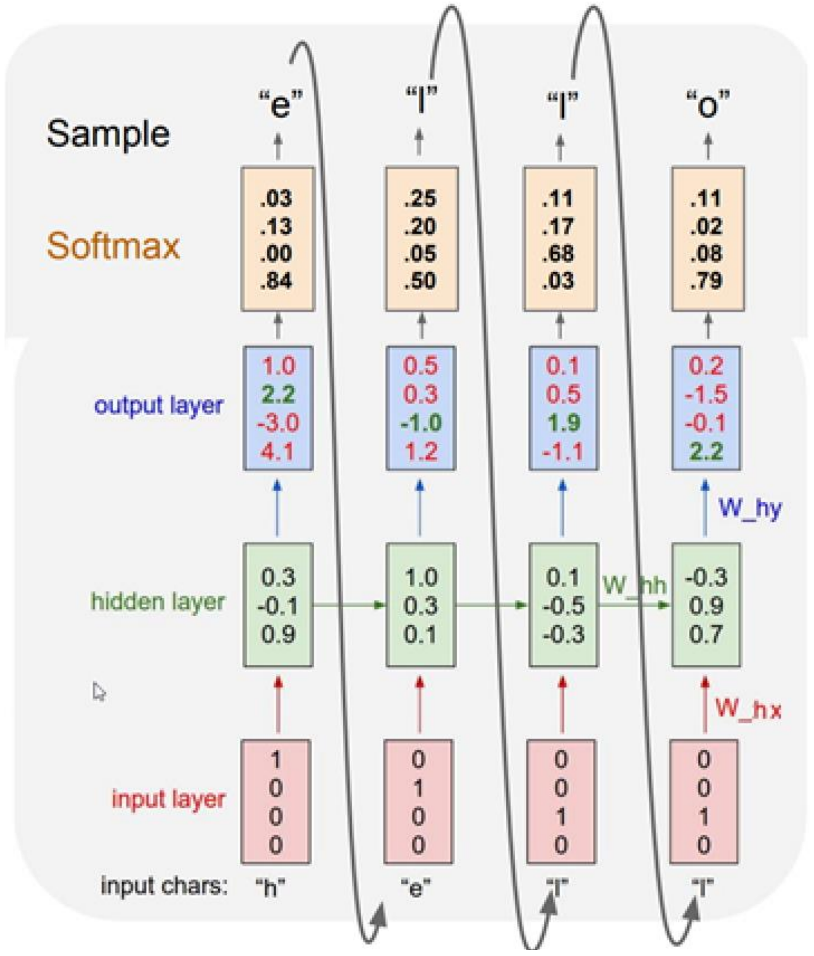
- At test time, we can sample from the model to synthesis new text.



RNN: Predicting the next character

- Vocabulary:
  - [h, e, l, o]

- At test time, we can sample from the model to synthesis new text.



Progress

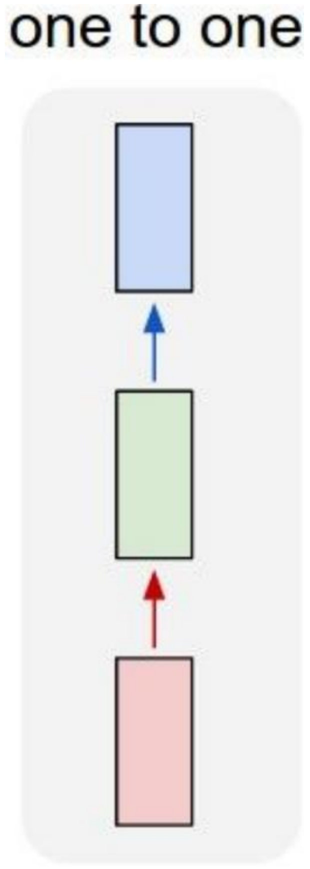
- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- **Input-output structure of RNN's**
- Training Recurrent Neural Networks
- Simple examples
- Advanced models
- Advanced examples

Input-output structure of RNN’s

- One-to-one
- one-to-many
- many-to-one
- Many-to-many
- many-to-many (encoder-decoder)

RNN: One-to-one

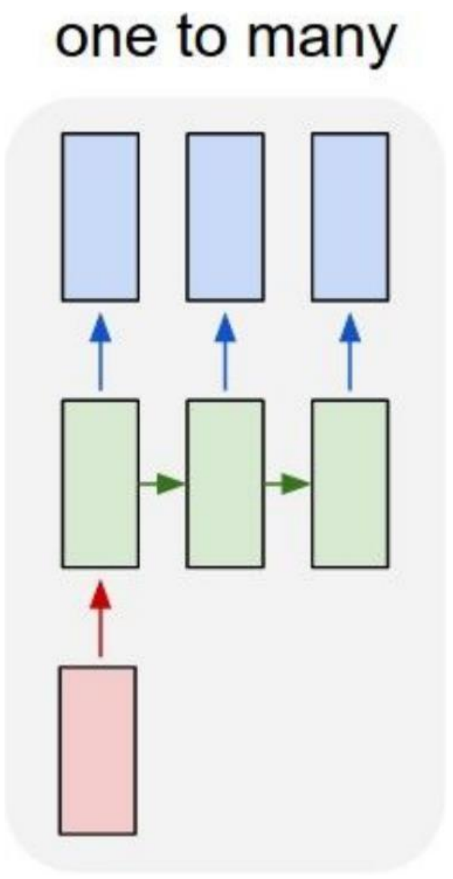
Normal feed forward neural network



RNN: One-to-many

Task:

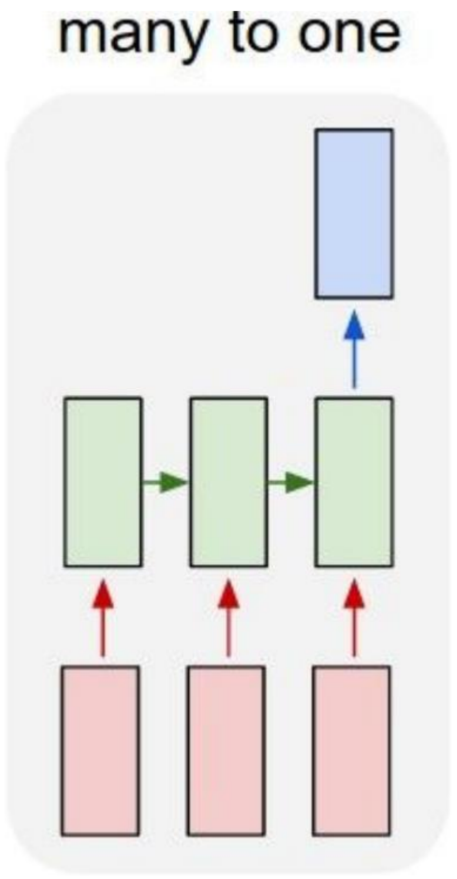
- Image Captioning
  - (Image → Sequence of words)



RNN: Many-to-one

Tasks:

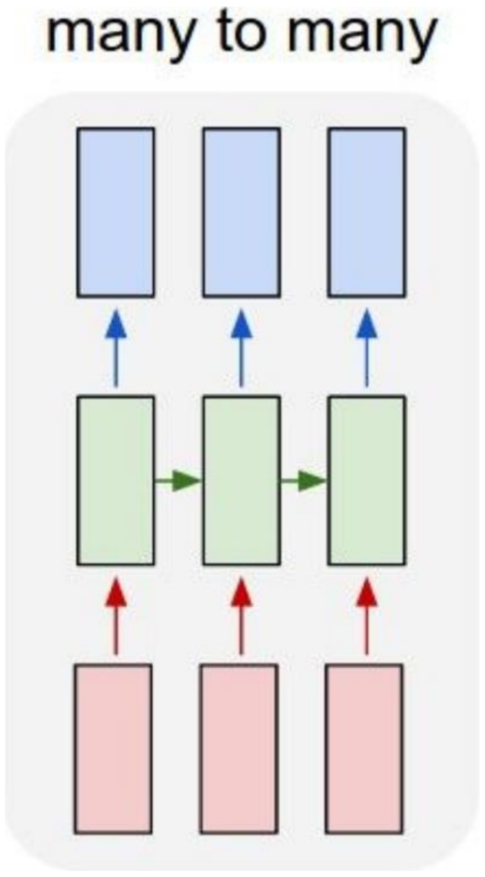
- Sentiment classification
- Video classification



RNN: Many-to-many

Task:

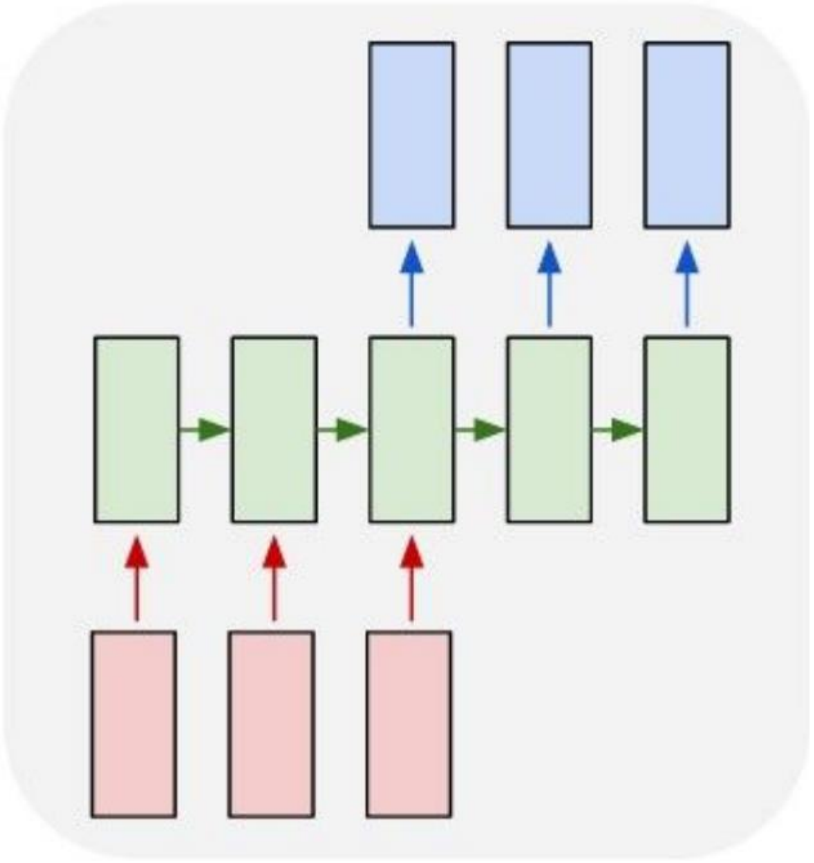
- Video classification on frame level



RNN: Many-to-many (encoder-decoder)

Task:

- Machine Translation
  (English → French)



Progress

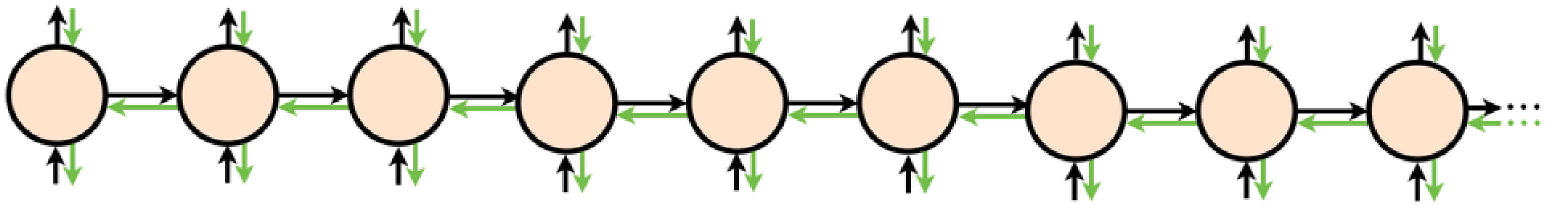
- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNN’s
- **Training Recurrent Neural Networks**
- Simple examples
- Advanced models
- Advanced examples

Training Recurrent Neural Networks

- The challenge in training a recurrent neural network is to preserve long range dependencies.

- Vanilla recurrent neural network
  - $h_t = f(W_{hh}h_{t-1} + W_{hx}x_t + b)$
  - $f = relu()$ can easily cause exploding values and/or gradients
  - $f = \tanh()$ can easily cause vanishing gradients, difficult to remember more than ~7 steps.

- Finite memory available



Exploding or vanishing gradients

- $\tanh()$ solves the exploding value problem
- $\tanh()$ does NOT solve the exploding gradient problem, think of a scalar input and a scalar hidden state.

$$h_t = tanh(W_{hh}h_{t-1} + W_{hx}x_t + b)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = [1 - \tanh^2(W_{hh}h_{t-1} + W_{hx}x_t + b)] \cdot W_{hh}$$

The gradient can explode/vanish exponentially in time (steps)

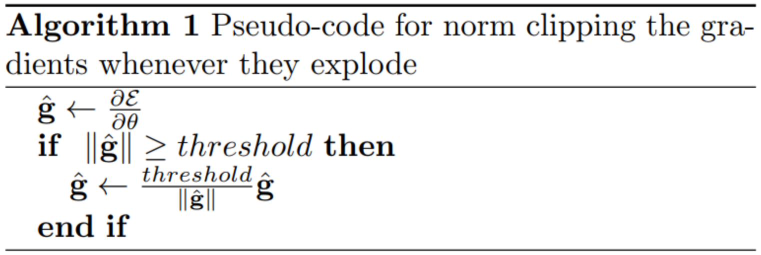
- If $|W_{hh}| < 1$, vanishing gradients
- If $|W_{hh}| > 1$, exploding gradients

Exploding or vanishing gradients

- Conditions for vanishing and exploding gradients for the weight matrix, W_{hh}:
  - If largest singular value > 1: Exploding gradients
  - If largest singular value < 1: Vanishing gradients

Gradient clipping

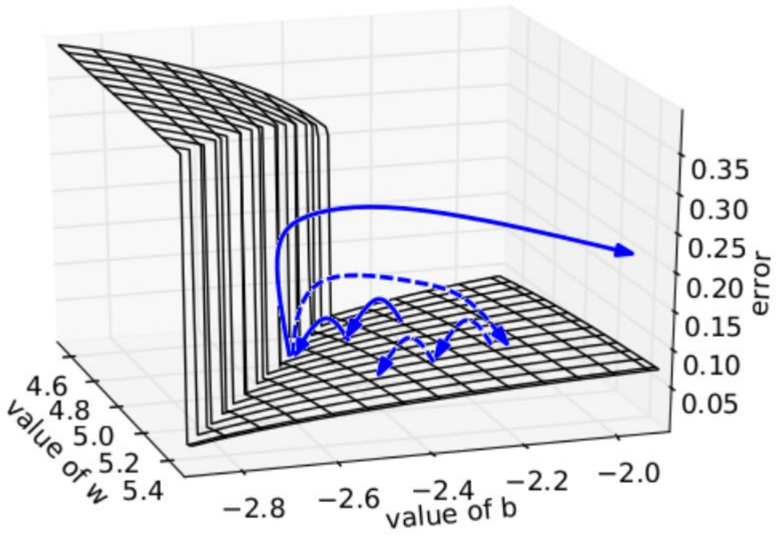
- To avoid exploding gradients, a simple trick is to set a threshold of the norm of the gradient matrix, $\frac{\partial h_t}{\partial h_{t-1}}$.



Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

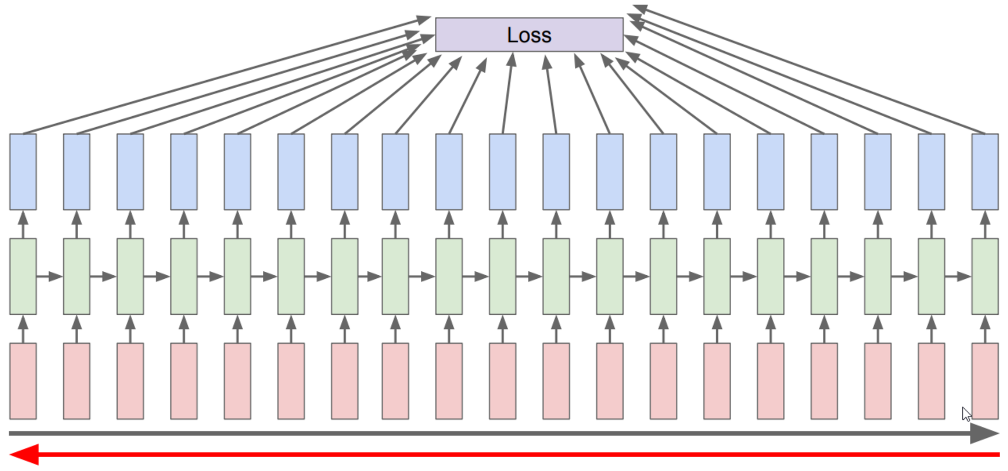
```
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ then
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if
```

- Error surface of a single hidden unit RNN
- High curvature walls
- Solid lines: Standard gradient descent trajectories
- Dashed lines gradient rescaled to fixes size



Vanishing gradients - “less of a problem”

- In contrast to feed-forward networks, RNNs will not stop learning in spite of vanishing gradients.
- The network gets “fresh” inputs each step, so the weights will be updated.
- The challenge is to learning long range dependencies. This can be improved using more advanced architectures.
- Output at time step t is mostly effected by the states with to time t.



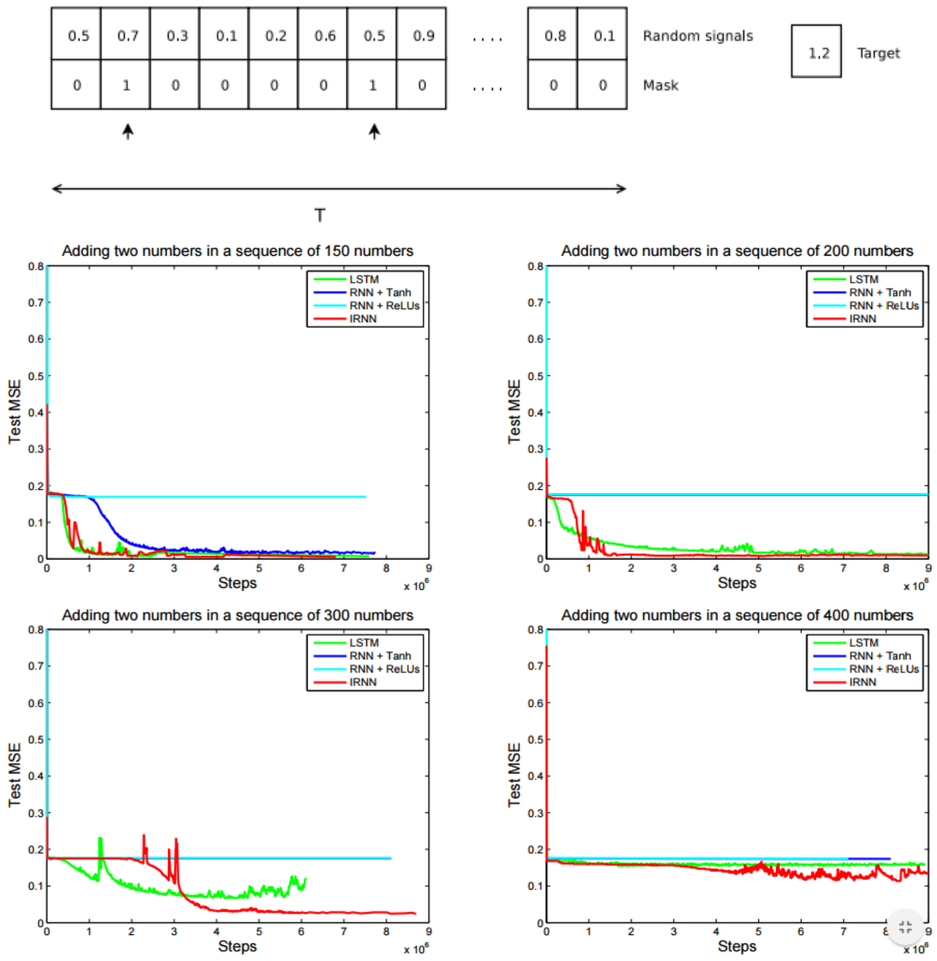
Initialization of the weights

- Potential initialization schemes for the transition matrices, W_{hh}:
  - Activation (tanh): $Var[W_l] = \frac{2}{n_l + n_{l+1}}$
  - Activation (tanh): $Var[W_l] = \frac{1}{n_l}$
  - Activation (relu): Orthogonal matrix
  - Activation (relu): Identity matrix

- Initial state, h_0:
  - Initialize to zeros or random
  - **Note:** We can learn the initial state by backpropagate into it.

Some initialization schemes can help RNNs

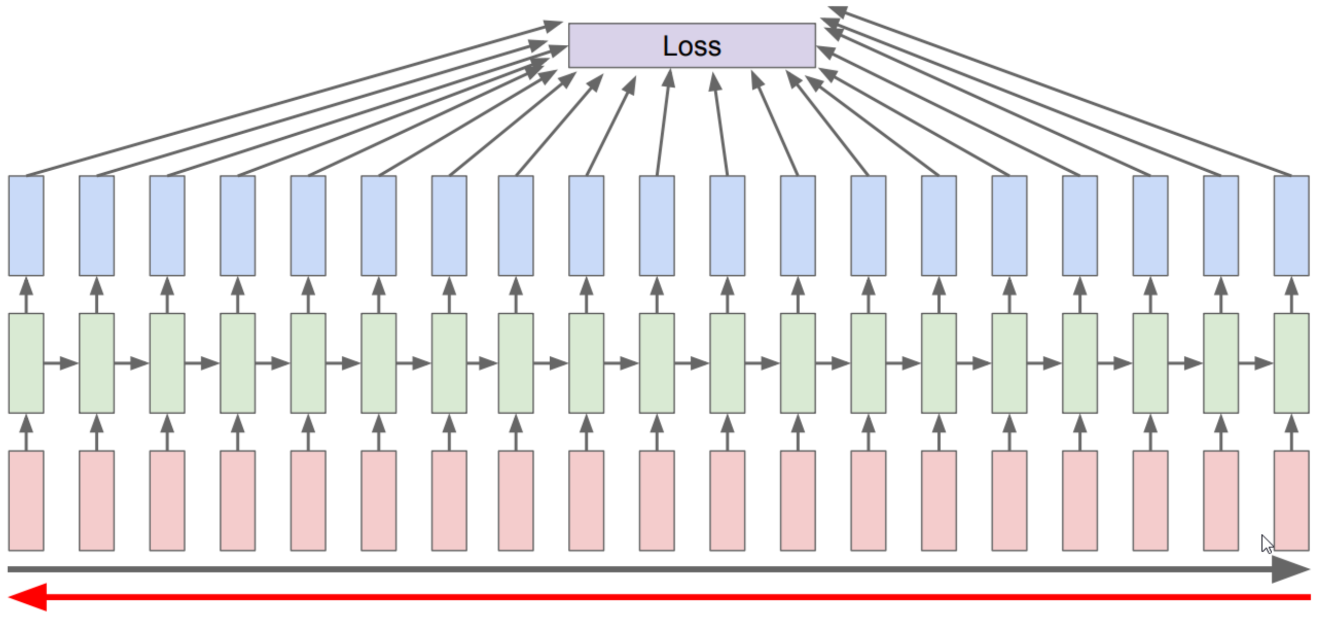
- Identity initialization of state transition matrix, W_{hh}:
  - Initially no state change can be a good start
  - Avoid exploding/vanishing gradients
  - May be able to use ReLu
  - The output should be the sum of the two values which have “mask” value of 1.



A Simple Way to Initialize Recurrent Networks of Rectified Linear Units

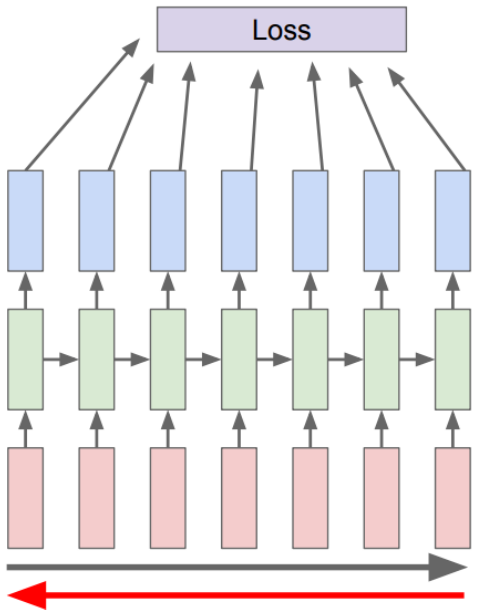
Backpropagation through time (BPTT)

- Training of a recurrent neural network is done using backpropagation and gradient descent algorithms
  - To calculate gradients you have to keep your inputs in memory until you get the backpropagating gradient
  - Then what if you are reading a book of 100 million characters?



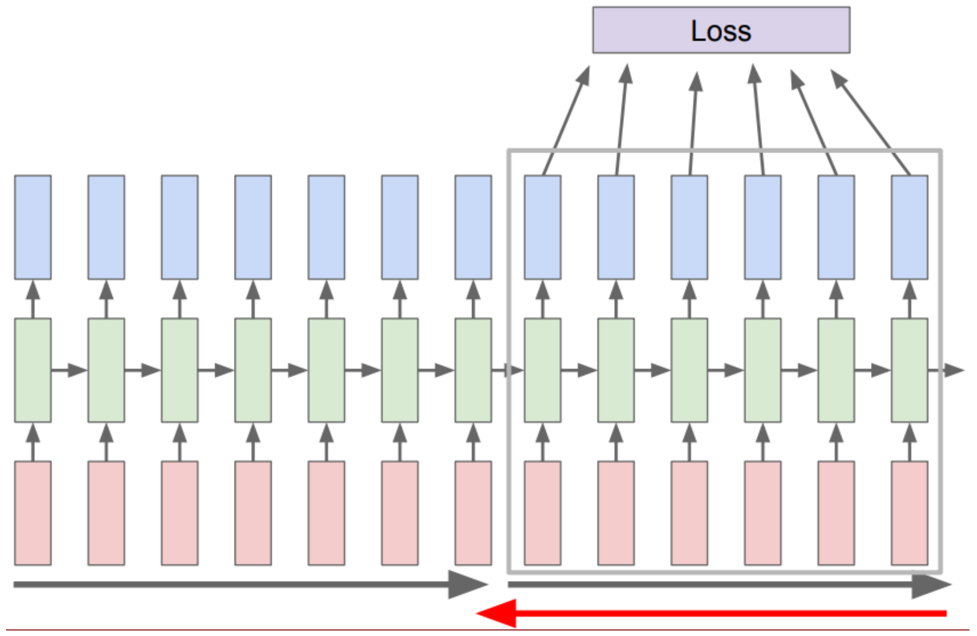
Truncated Backpropagation thought time

- The solution is that you stop at some point and update the weight as if you were done.



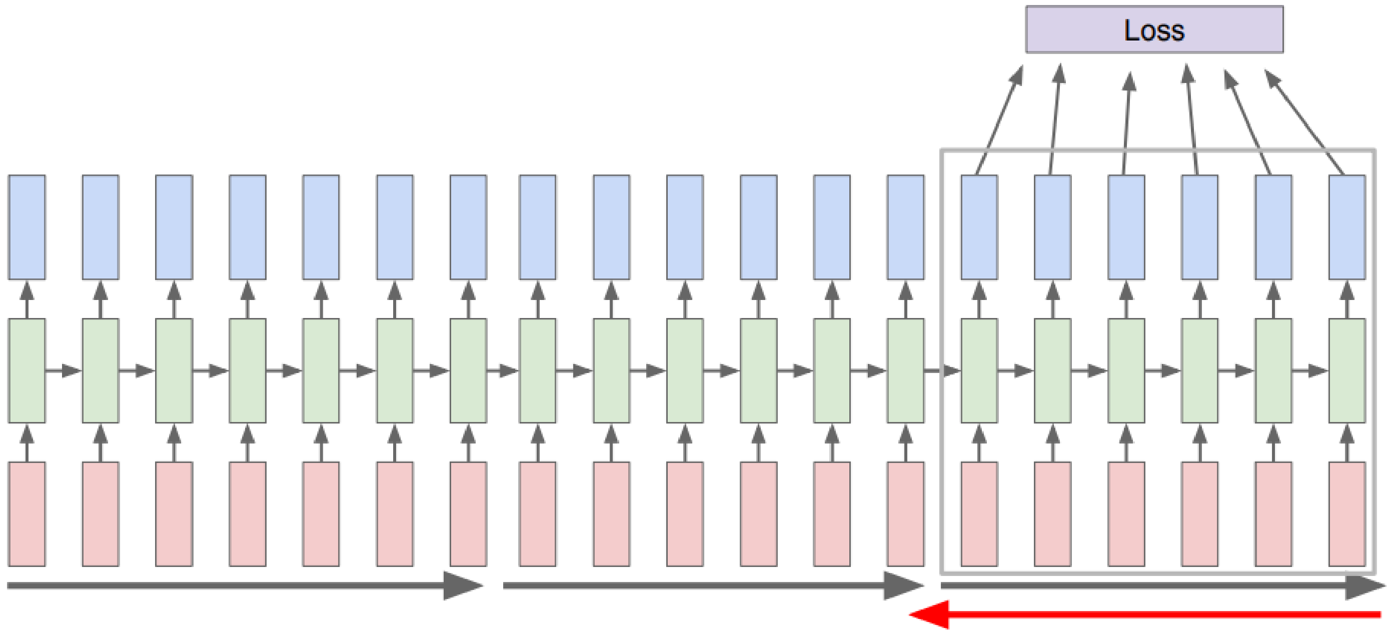
Truncated Backpropagation thought time

- The solution is that you stop at some point and update the weight as if you were done.



Truncated Backpropagation thought time

- The solution is that you stop at some point and update the weight as if you were done.



Truncated Backpropagation thought time

- Advantages:
  - Reducing the memory requirement
  - Faster parameter updates

- Disadvantage:
  - Not able to capture longer dependencies then the truncated length.

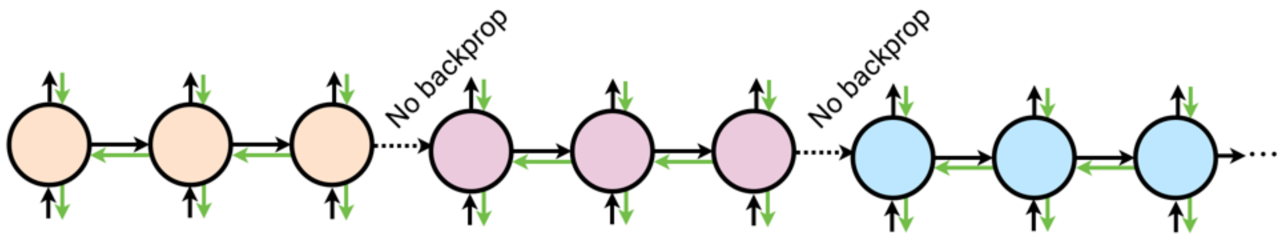
What effect do you get from stopping the gradients?

- You are not guarantied to remember interactions longer than the number of steps.

- Say a text call a person Anna then much later refer to her as “she”

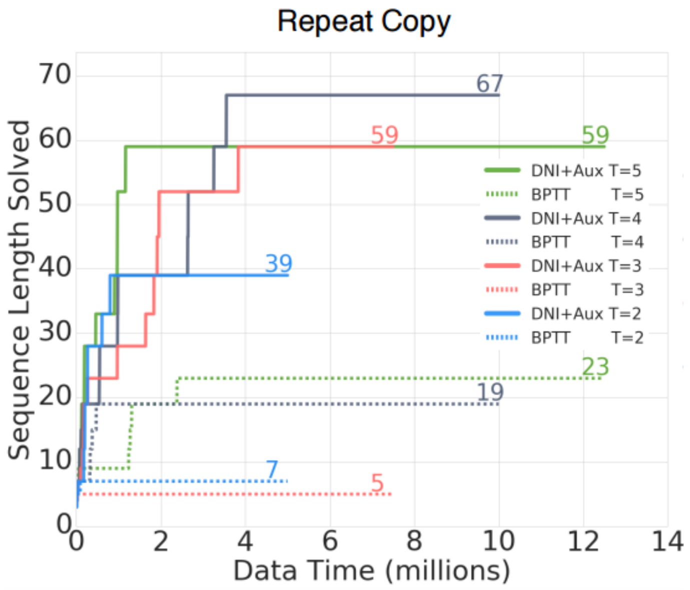
- A model with a lot of steps could directly learn to remember Anna.

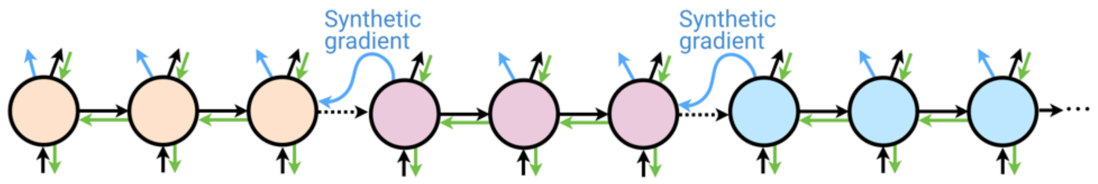
- RNNs can in test time remember longer than the number of steps:
  - You may have learned from earlier cases where “name” and “he/she” were much closer
  - The model knows that it should save gender of the name



A fun fix! Synthetic gradients

- Train a network to predict the gradients you would get if all steps were connected
- You can learn that if you see Anna in the state, you should expect gradients "asking for Anna" later
- This is of course not perfect since you don't have all the coming information





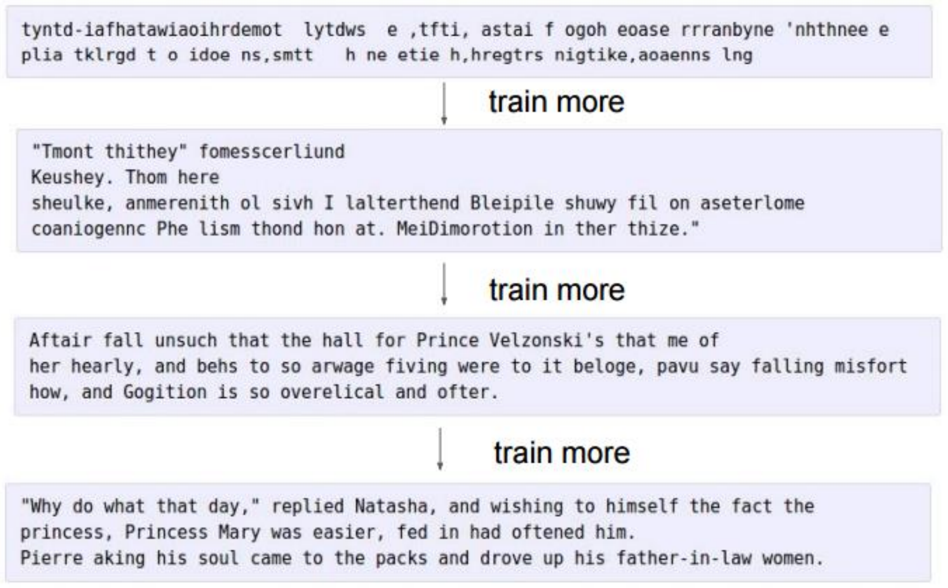
[Decoupled Neural Interfaces using Synthetic Gradients]

Progress

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNN’s
- Training Recurrent Neural Networks
- **Simple examples**
- Advanced models
- Advanced examples

Inputting Shakespeare

- Input a lot of text
- Try to predict the next character
- Learning spelling, quotes and punctuation.



```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

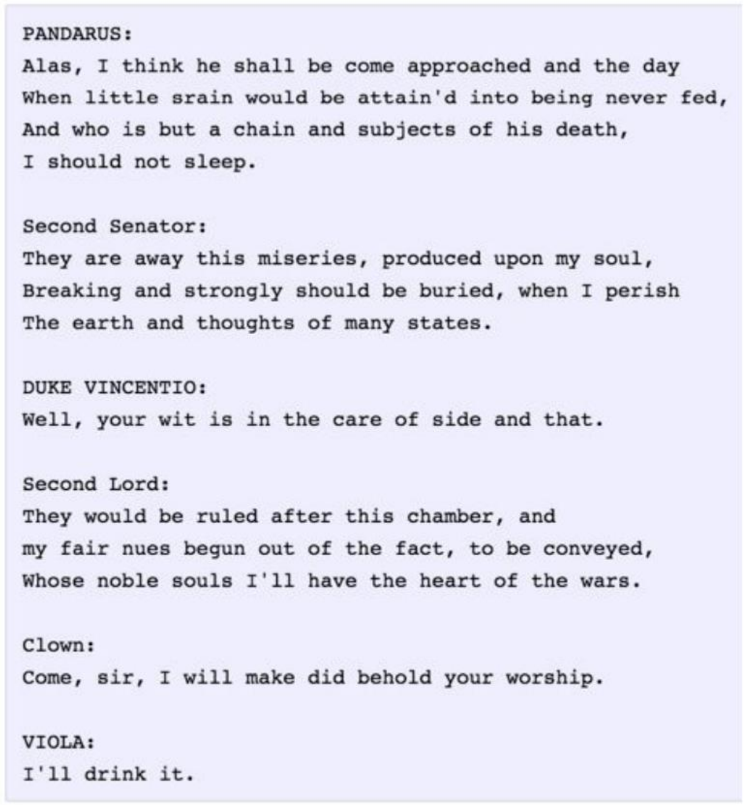
```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

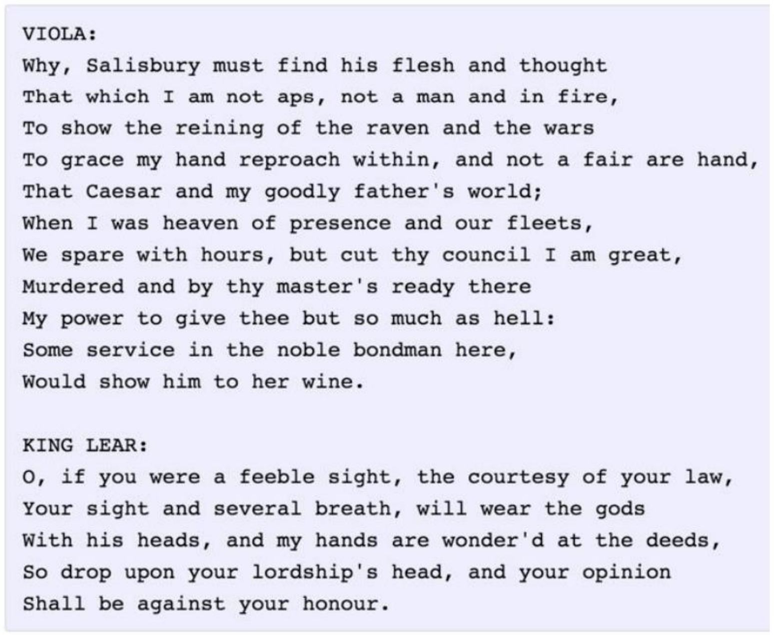
```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

https://gist.github.com/karpathy/d4dee566867f8291f086

Inputting Shakespeare



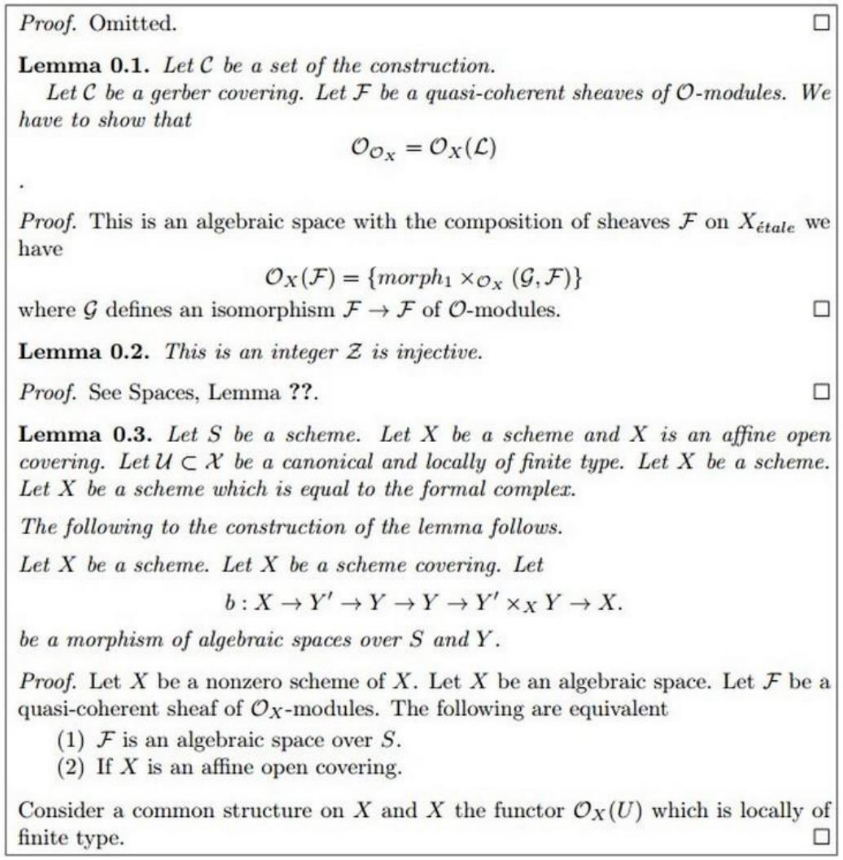
```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

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.
```



```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

https://gist.github.com/karpathy/d4dee566867f8291f086

Inputting 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

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

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

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

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

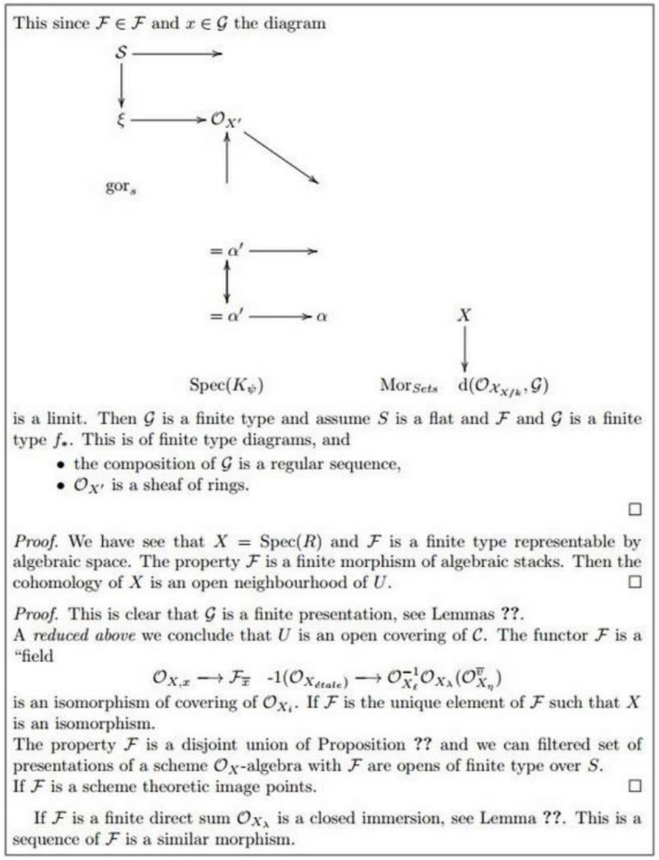
$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

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

$$\begin{tikzcd}
\mathcal{S} \arrow[r] \arrow[d] & {} & & & \\
\xi \arrow[r] & \mathcal{O}_{X'} \arrow[dr] & & & \\
\mathrm{gor}_s & {} \arrow[u] & {} & & \\
& =\alpha' \arrow[r] \arrow[d, leftrightarrow] & {} & & \\
& =\alpha' \arrow[r] & \alpha & & X \arrow[d] \\
& \mathrm{Spec}(K_\psi) & & \mathrm{Mor}_{Sets} & \mathrm{d}(\mathcal{O}_{X_{X/k}}, \mathcal{G})
\end{tikzcd}$$

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 $X = \mathrm{Spec}(R)$ 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. □

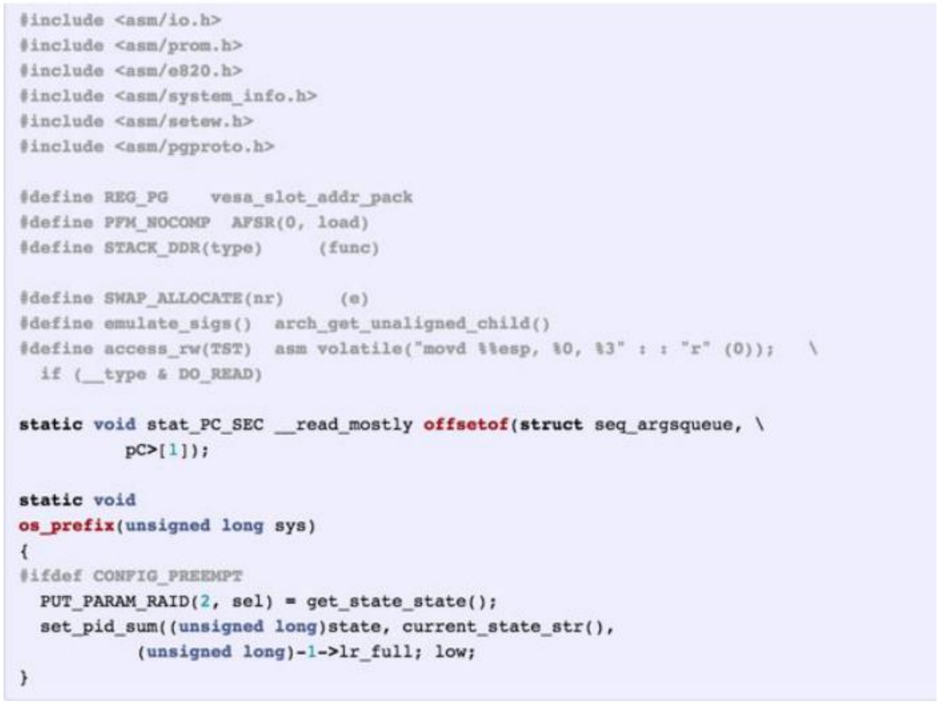
Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

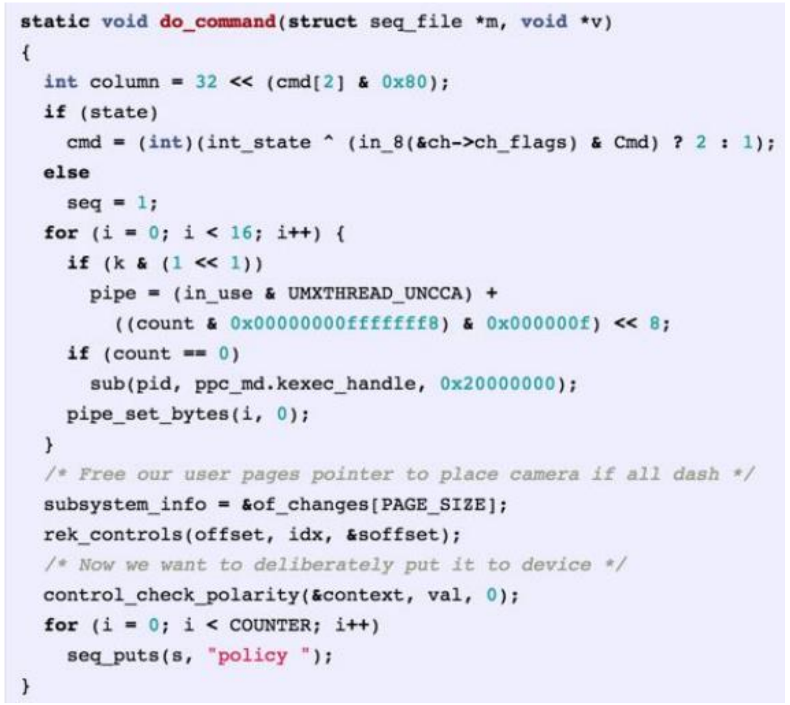
is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ 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.

Inputting C code (linux kernel)



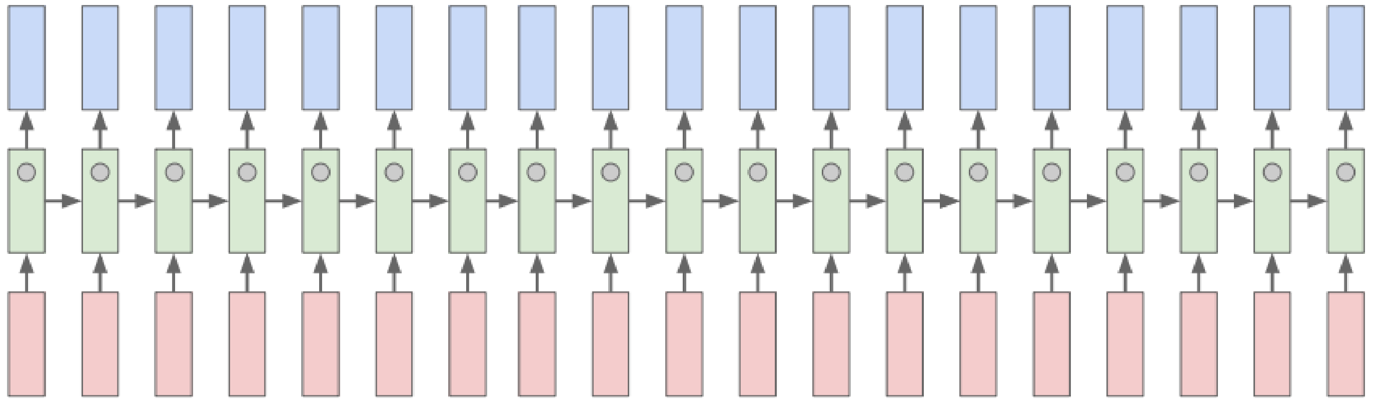
```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#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;
}
```

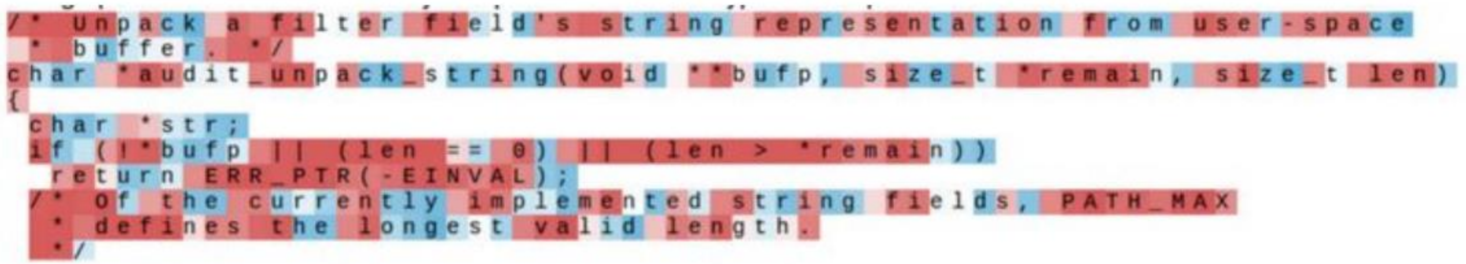


```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Interpreting cell activations

- Semantic meaning of the hidden state vector

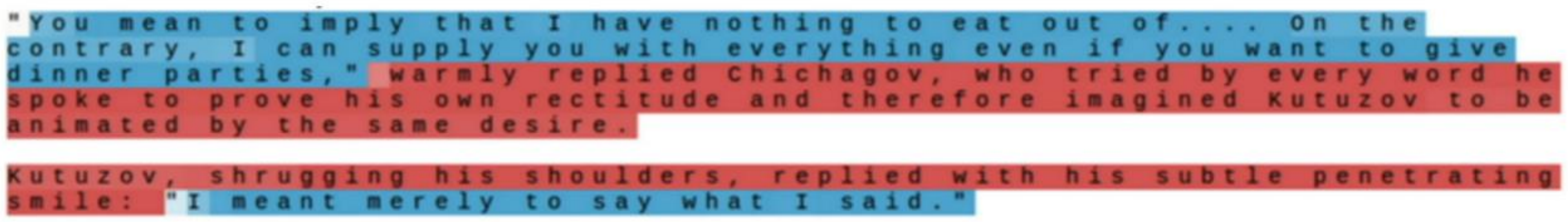




```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
```

Visualizing and Understanding Recurrent Networks

Interpreting cell activations



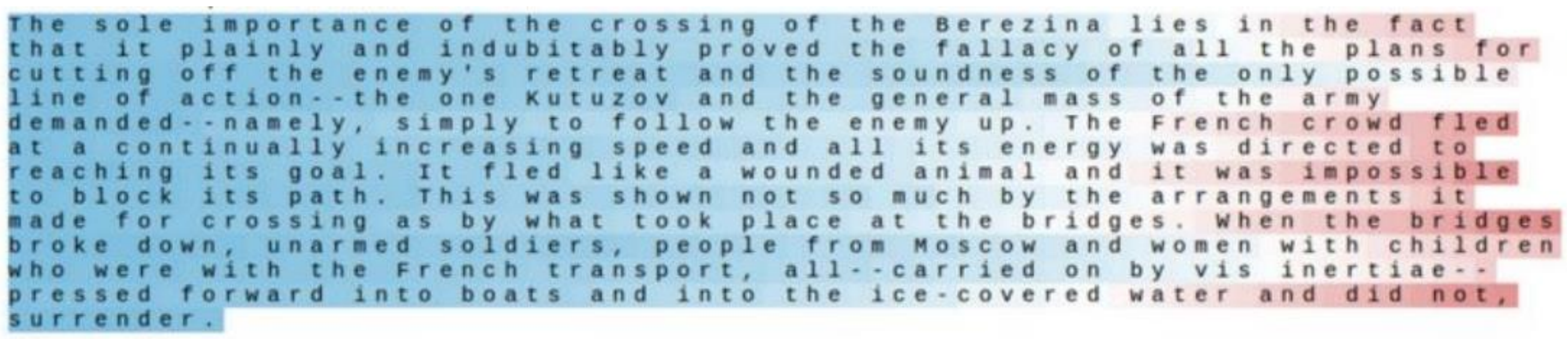
"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

quote detection cell

Visualizing and Understanding Recurrent Networks

Interpreting cell activations

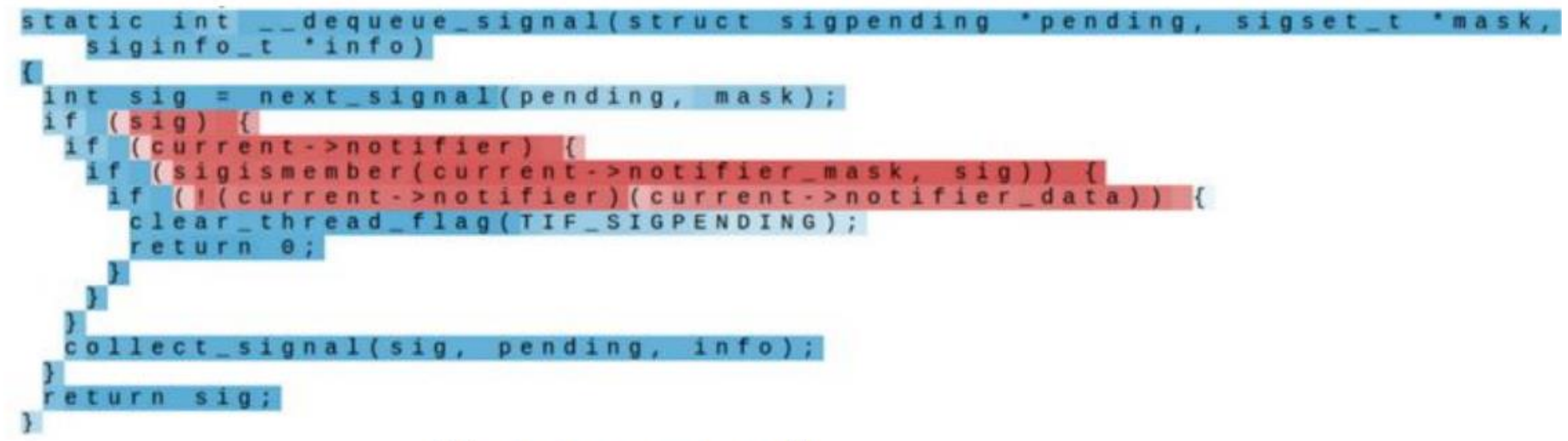


The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.

line length tracking cell

Visualizing and Understanding Recurrent Networks

Interpreting cell activations

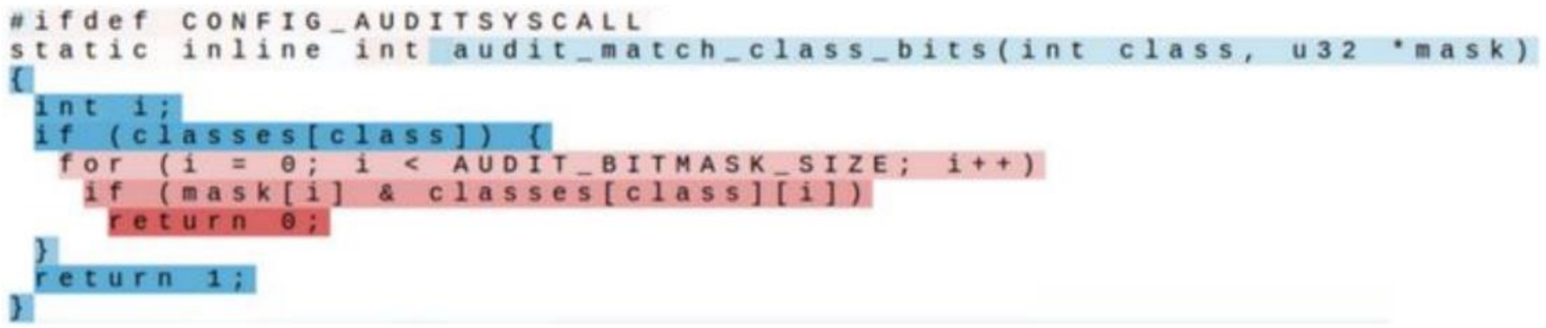


```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
   siginfo_t *info)
{
 int sig = next_signal(pending, mask);
 if (sig) {
  if (current->notifier) {
   if (sigismember(current->notifier_mask, sig)) {
    if (!(current->notifier)(current->notifier_data)) {
     clear_thread_flag(TIF_SIGPENDING);
     return 0;
    }
   }
  }
  collect_signal(sig, pending, info);
 }
 return sig;
}
```

if statement cell

Visualizing and Understanding Recurrent Networks

Interpreting cell activations



```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
 int i;
 if (classes[class]) {
  for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
   if (mask[i] & classes[class][i])
    return 0;
 }
 return 1;
}
```

code depth cell

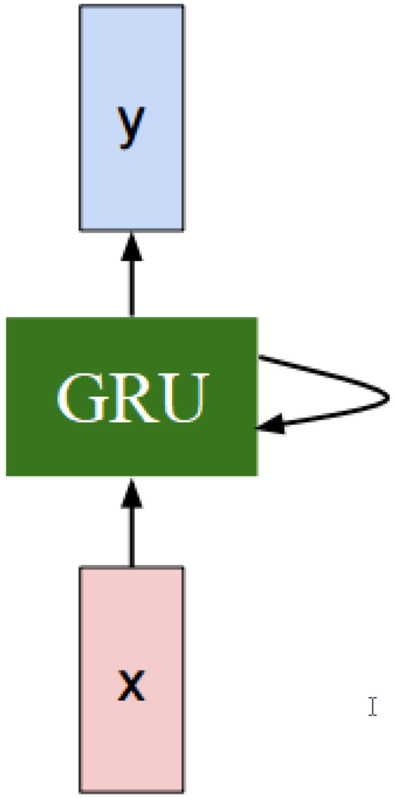
Visualizing and Understanding Recurrent Networks

Progress

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNN’s
- Training Recurrent Neural Networks
- Simple examples
- **Advanced models**
- Advanced examples

Gated Recurrent Unit (GRU)

- A more advanced recurrent unit
- Uses gates to control information flow
- Used to improve long term dependencies



Gated Recurrent Unit (GRU)

GRU has the ability to adding and removing to the state, not “transforming the state” only.

Vanilla RNN

- Cell state $h_t = tanh(W^{hx}x_t + W^{hh}h_{t-1} + b)$

GRU

- Update gate $\Gamma^u = \sigma(W^u x_t + U^u h_{t-1} + b^u)$
- Reset gate $\Gamma^r = \sigma(W^r x_t + U^r h_{t-1} + b^r)$
- Candidate cell state $\tilde{h}_t = tanh(Wx_t + U(\Gamma^r \circ h_{t-1}) + b)$
- Final cell state $h_t = \Gamma^u \circ h_{t-1} + (1 - \Gamma^u) \circ \tilde{h}_t$

With Γ^r as ones and Γ^u as zeros, GRU → Vanilla RNN

Gated Recurrent Unit (GRU)

Summary:

- The update gate , Γ^u, can easily be close to 1 due to the sigmoid activation function. Copying the previous hidden state will prevent vanishing gradients.
- With $\Gamma^u = 0$ and $\Gamma^r = 0$ the GRU unit can forget the past.
- In a standard RNN, we are transforming the hidden state regardless of how useful the input is.

• Update gate	$\Gamma^u = \sigma(W^u x_t + U^u h_{t-1} + b^u)$
• Reset gate	$\Gamma^r = \sigma(W^r x_t + U^r h_{t-1} + b^r)$
• Candidate cell state	$\tilde{h}_t = tanh(W x_t + U(\Gamma^r \circ h_{t-1}) + b)$
• Final cell state	$h_t = \Gamma^u \circ h_{t-1} + (1 - \Gamma^u) \circ \tilde{h}_t$

Long Short Term Memory (LSTM) (for reference only)

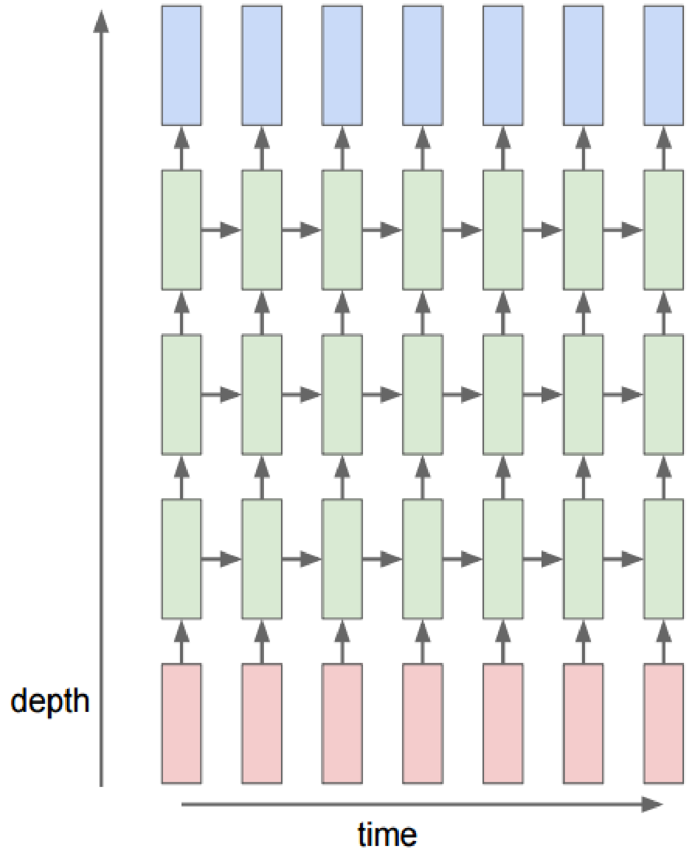
GRU		LSTM	
Update gate	$\Gamma^u = \sigma(W^u x_t + U^u h_{t-1} + b^u)$	Forget gate	$\Gamma^f = \sigma(W^f x_t + U^f h_{t-1} + b^f)$
Reset gate	$\Gamma^r = \sigma(W^r x_t + U^r h_{t-1} + b^r)$	Input gate	$\Gamma^i = \sigma(W^i x_t + U^i h_{t-1} + b^i)$
		Output gate	$\Gamma^o = \sigma(W^o x_t + U^o h_{t-1} + b^o)$
Candidate cell state	$\tilde{h}_t = tanh(W x_t + U(\Gamma^r \circ h_{t-1}) + b)$	Potential cell memory	$\tilde{c}_t = tanh(W^c x_t + U^c h_{t-1} + b^c)$
Final cell state	$h_t = \Gamma^u \circ h_{t-1} + (1 - \Gamma^u) \circ \tilde{h}_t$	Final cell memory	$c_t = \Gamma^f \circ c_{t-1} + \Gamma^i \circ \tilde{c}_t$
		Final cell state	$h_t = \Gamma^o \circ \tanh(c_t)$

LSTM training

- One can initialize the biases of the forget gate such that the values in the forget gate are close to one.

Multi-layer Recurrent Neural Networks

- Multi-layer RNNs can be used to enhance model complexity
- Similar as for feed forward neural networks, stacking layers creates higher level feature representation
- Normally, 2 or 3 layer deep, not as deep as conv nets
- More complex relationships in time



Bidirectional recurrent neural network

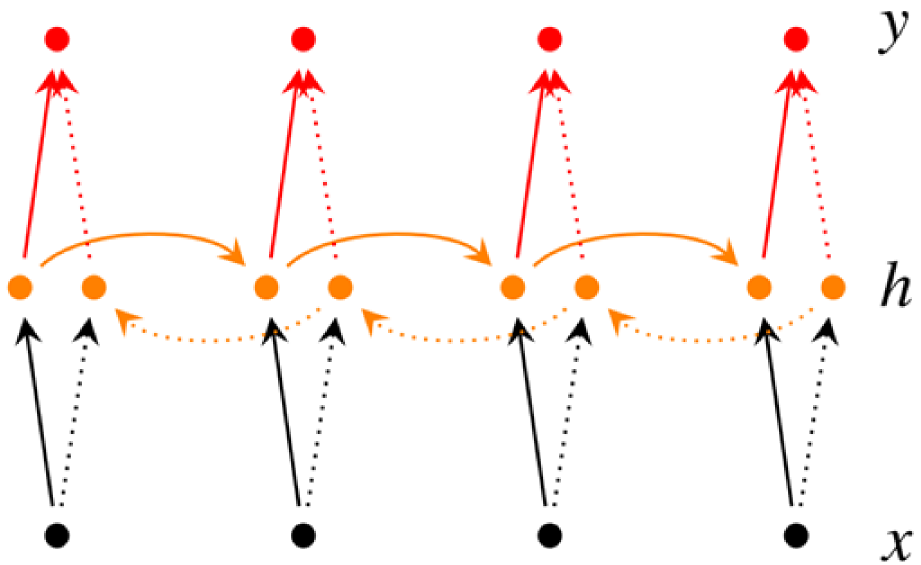
- The blocks can be vanilla, LSTM and GRU recurrent units
- Real time vs post processing

$$\overrightarrow{h}_t = f(\overrightarrow{W}_{hx}x_t + \overrightarrow{W}_{hh}\overrightarrow{h}_{t-1})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}_{hx}x_t + \overleftarrow{W}_{hh}\overleftarrow{h}_{t+1})$$

$$h_t = [\overrightarrow{h}_t, \overleftarrow{h}_t]$$

$$y_t = g(W_{hy}h_t + b)$$



Bidirectional recurrent neural network

- Example:
  - **Task:** Is the word part of a person's name?
  - **Text 1:** He said, "Teddy bear are on sale!"
  - **Text 2:** He said, "Teddy Roosevelt was a great President!"

$$\overrightarrow{h}_t = f\left(\overrightarrow{W}_{hx} x_t + \overrightarrow{W}_{hh} \overrightarrow{h}_{t-1}\right)$$

$$\overleftarrow{h}_t = f\left(\overleftarrow{W}_{hx} x_t + \overleftarrow{W}_{hh} \overleftarrow{h}_{t+1}\right)$$

$$h_t = \left[\overrightarrow{h}_t, \overleftarrow{h}_t\right]$$

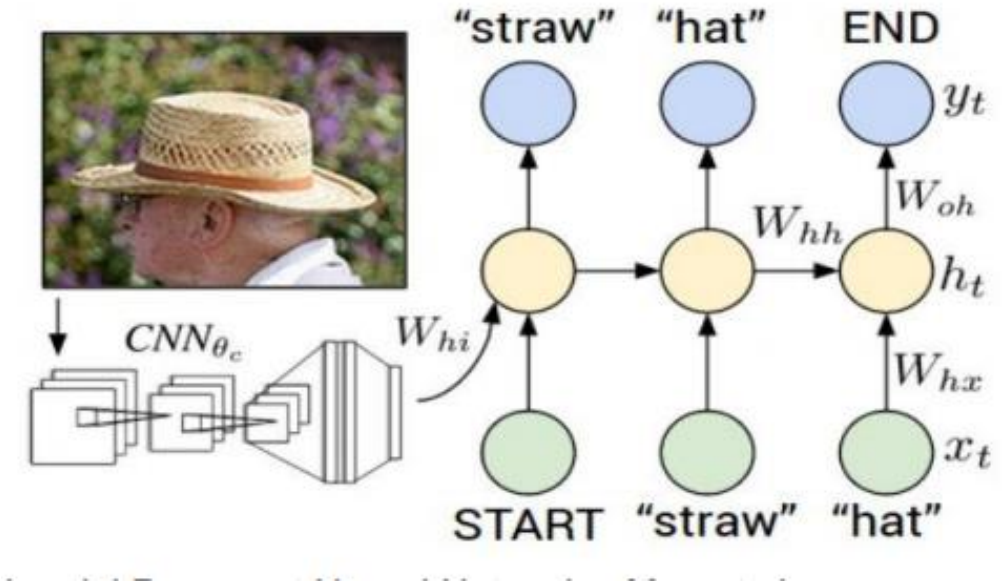
$$y_t = g(W_{hy} h_t + b)$$

Progress

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNN’s
- Training Recurrent Neural Networks
- Simple examples
- Advanced models
- **Advanced examples**

Image Captioning

- Combining text RNN with image CNN
- Learning RNN to interpret top features from CNN



Deep Visual-Semantic Alignments for Generating Image Descriptions

Image Captioning

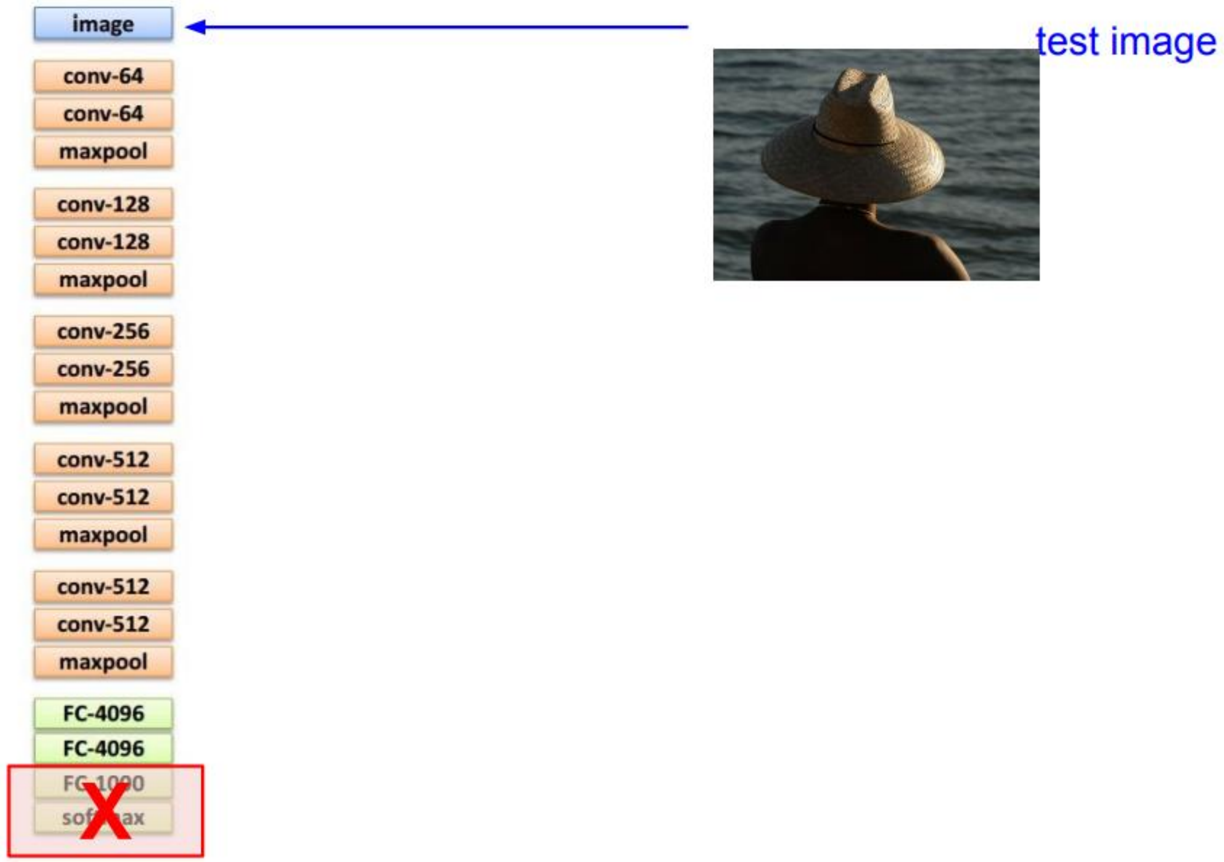


Image Captioning

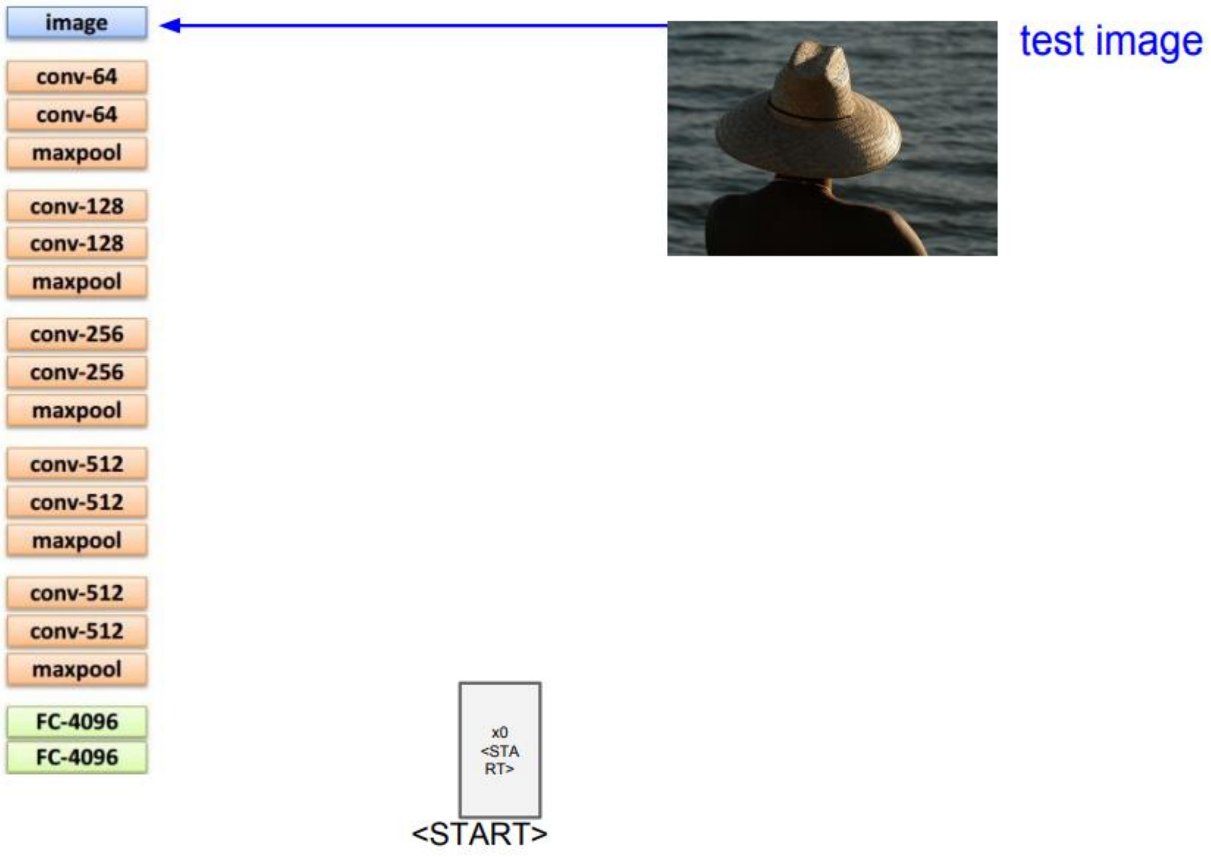
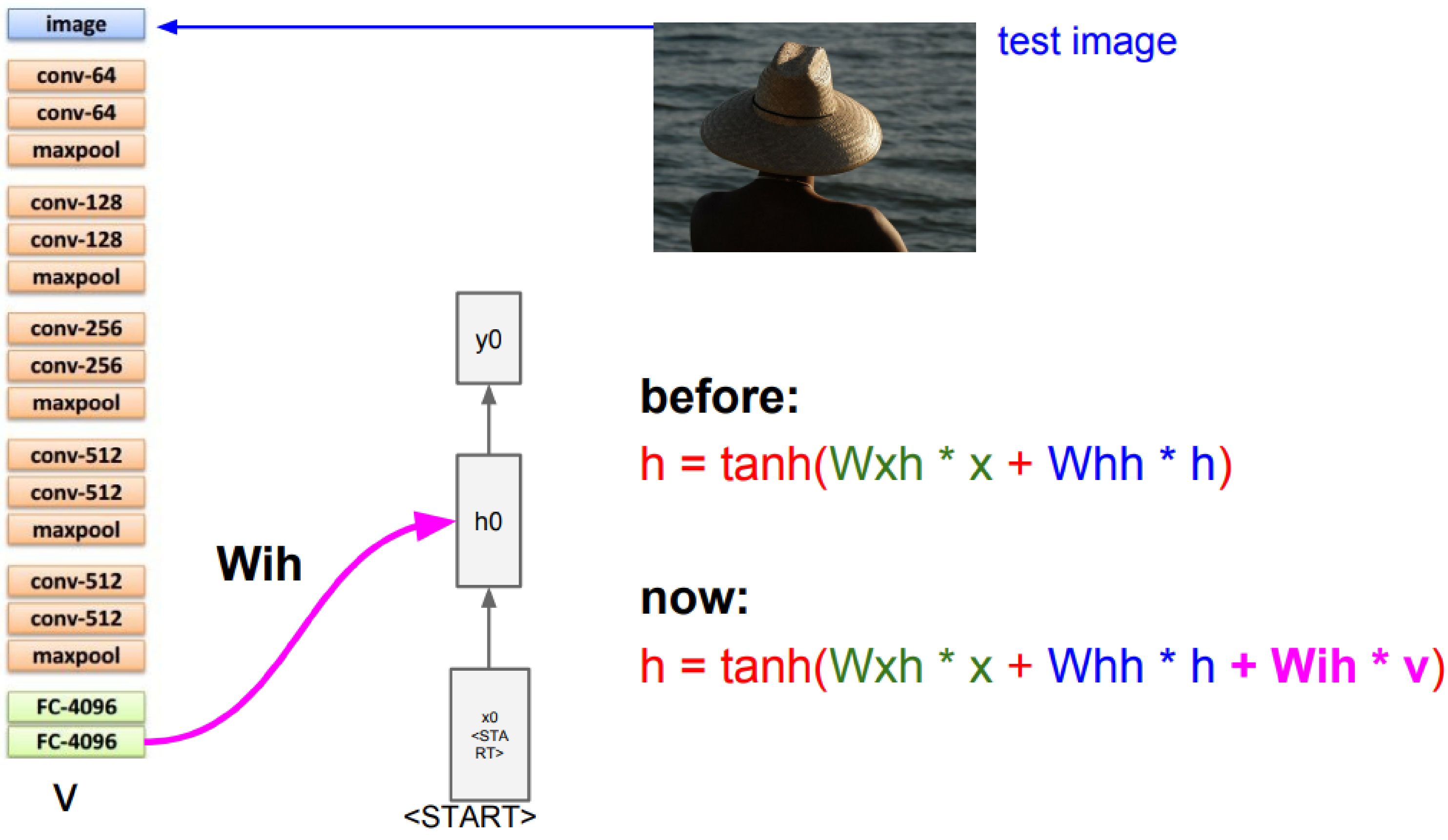


Image Captioning



before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih** * **v**)

Image Captioning

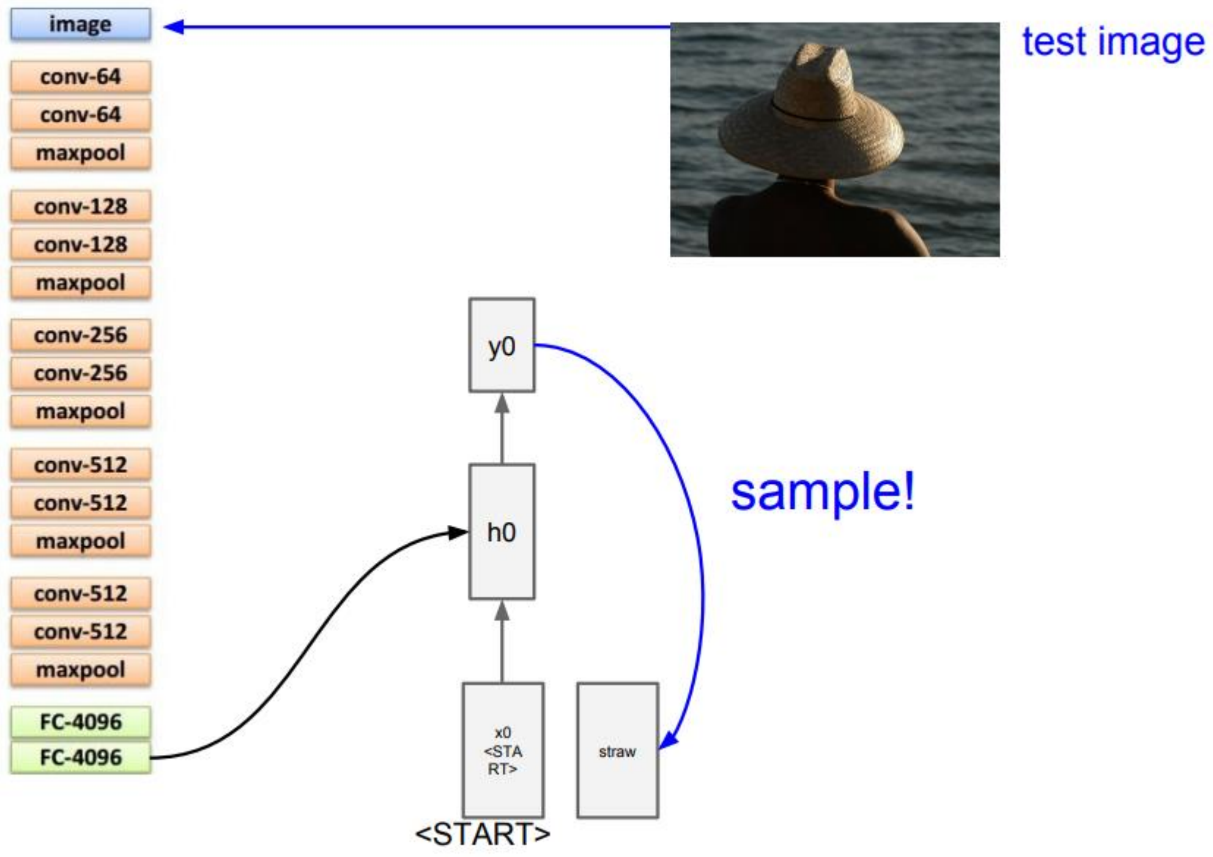


Image Captioning

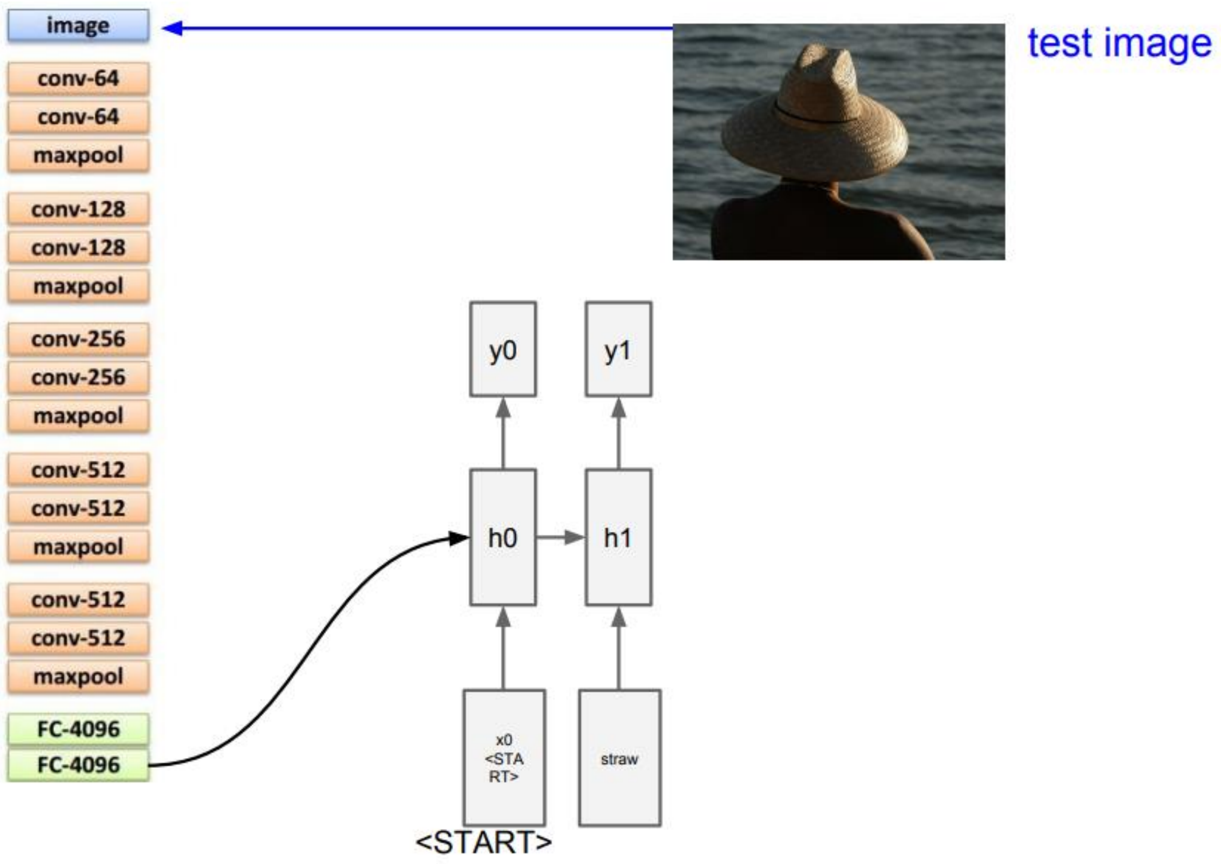


Image Captioning

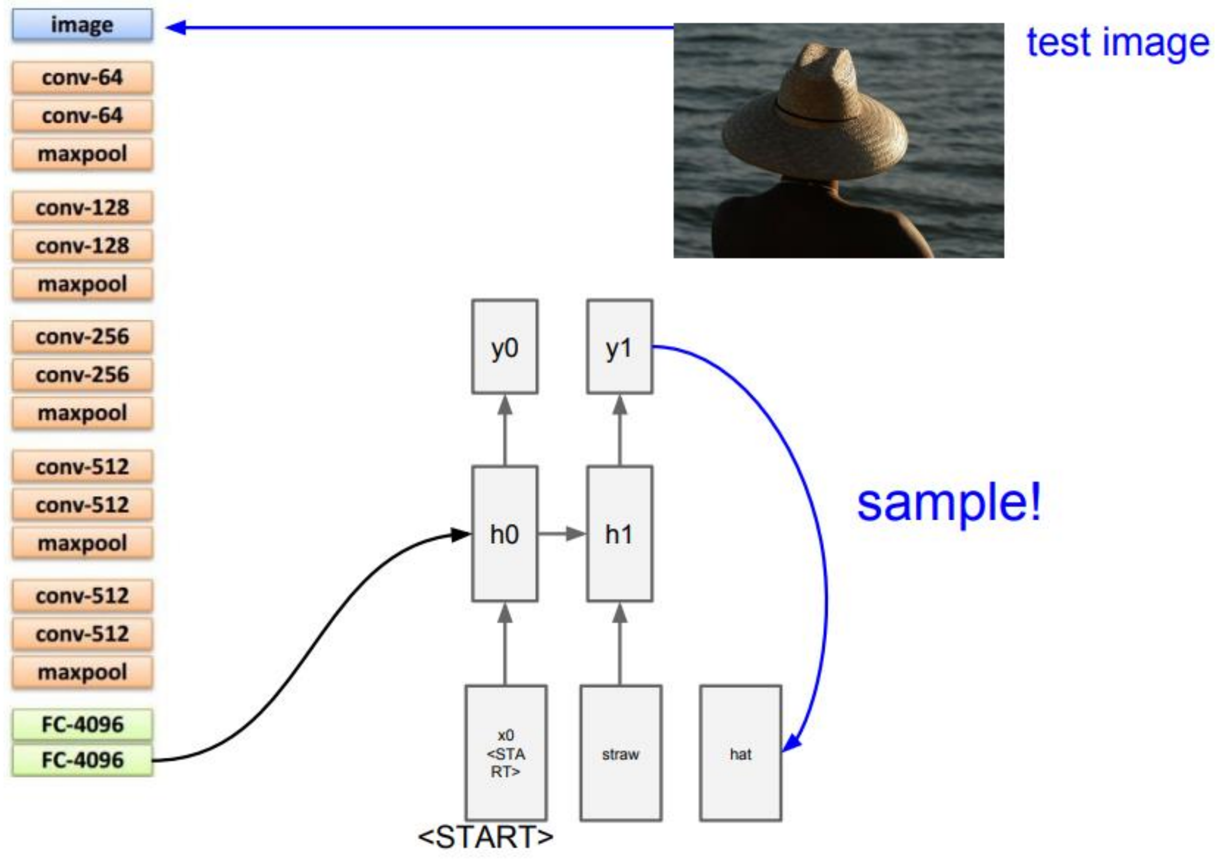


Image Captioning

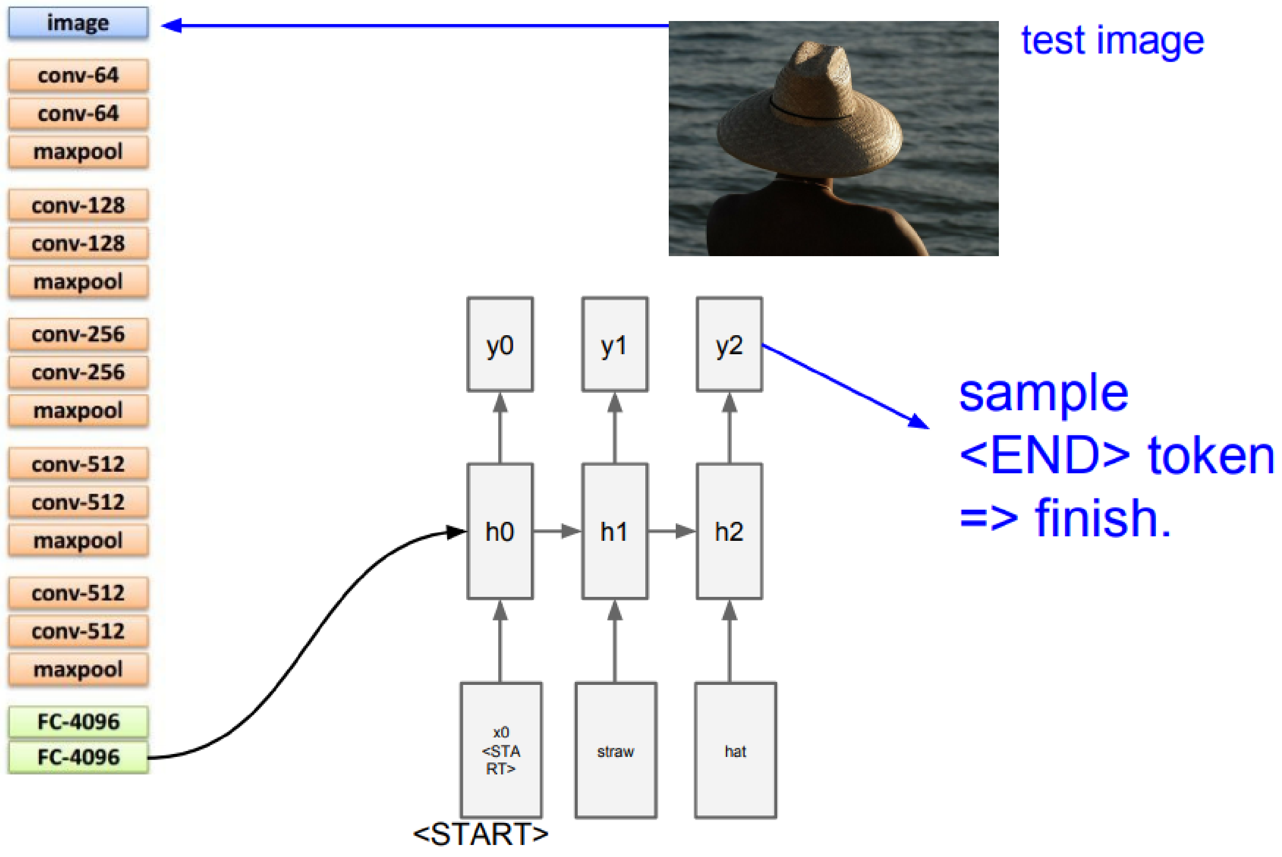


Image Captioning: Results

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

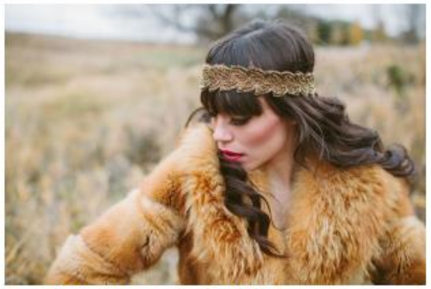
A tennis player in action on the court

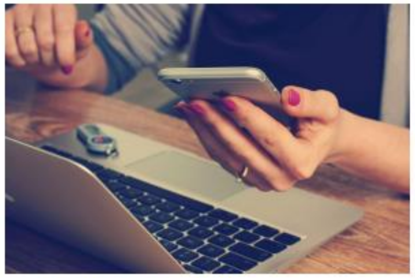
Two giraffes standing in a grassy field

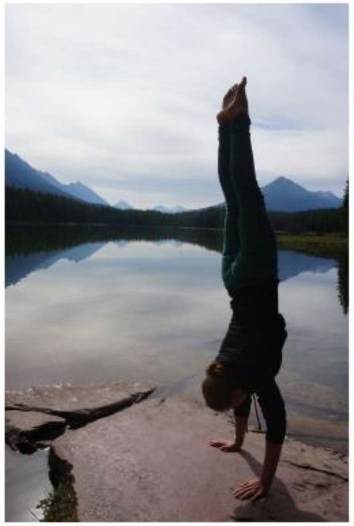
A man riding a dirt bike on a dirt track

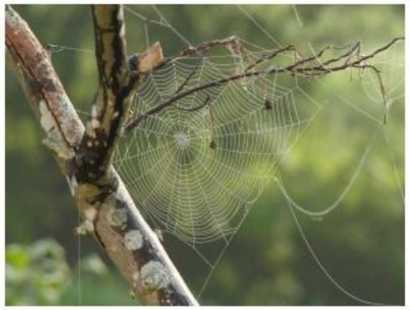
NeuralTalk2

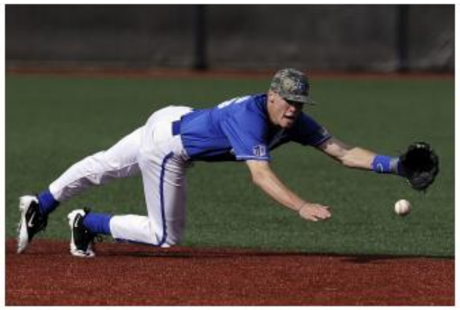
Image Captioning: Failures


A woman is holding a cat in her hand


A person holding a computer mouse on a desk


A woman standing on a beach holding a surfboard

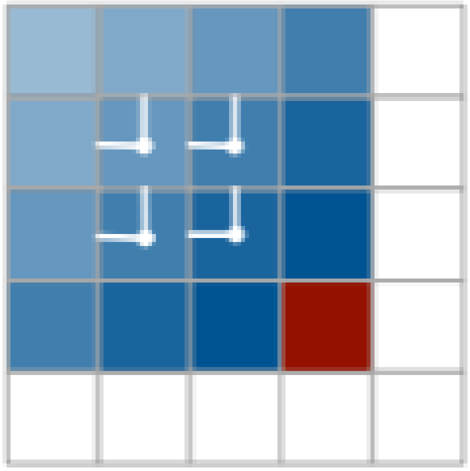

A bird is perched on a tree branch

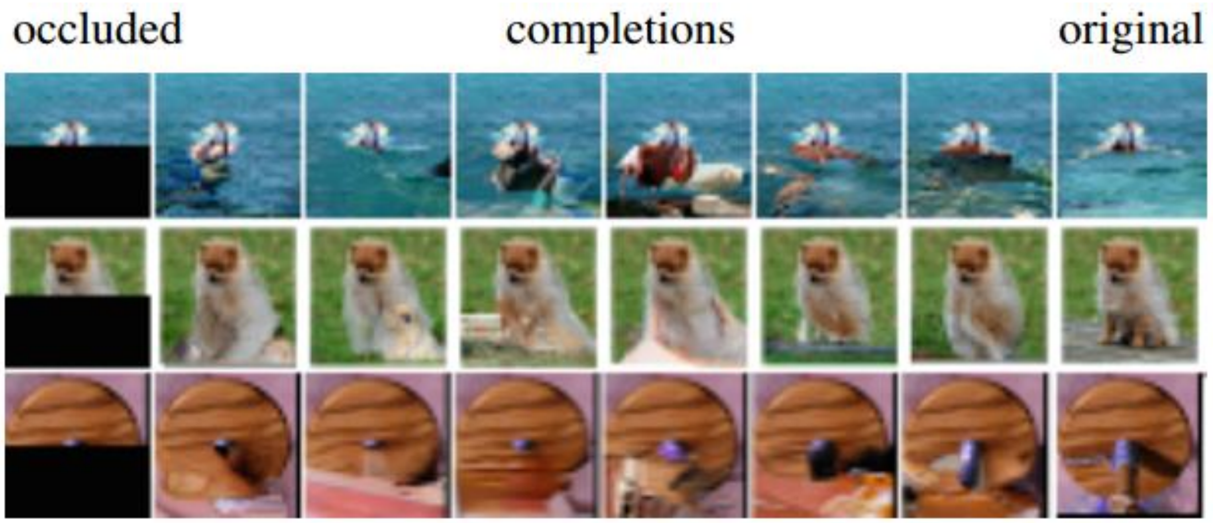

A man in a baseball uniform throwing a ball

NeuralTalk2

Generating images

- Input top and left pixel to predict a new pixel.
- Fill in holes.





Machine translation

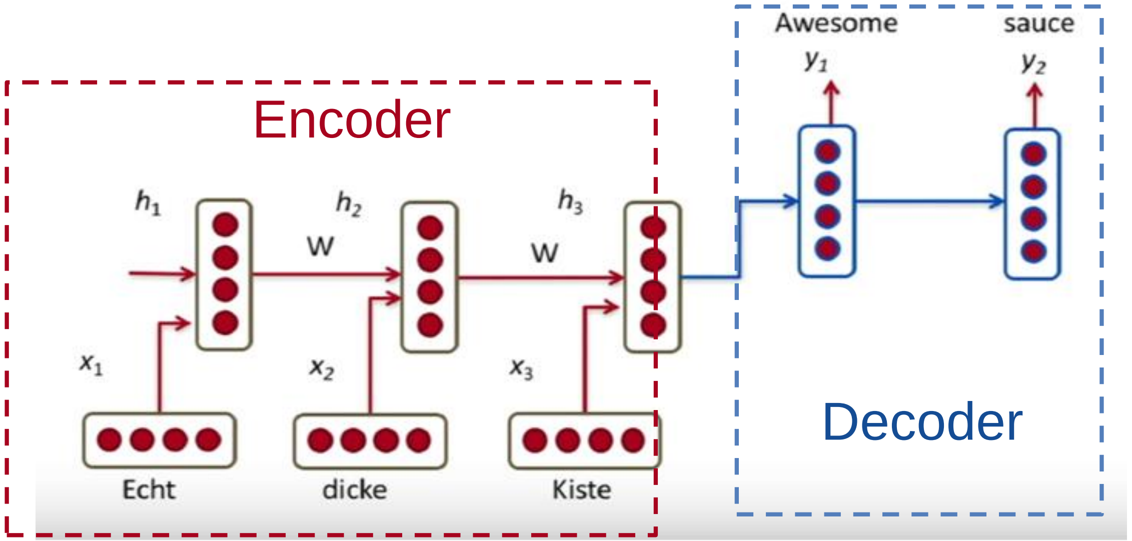
- Example:
  - German to English:
  - “Echt dicke Kiste” to “Awesome sauce”

- Tokenize each word/character, e.g. as a onehot vector.
- <EOS> token
- Can use cross-entropy loss for every timestep

- Character model vs vocabulary model:
  - Pros character model :
    - Much smaller class size (input/ouput vector length)
    - You don’t need to worry about unknown words
  - Pros vocabulary model
    - Shorter sequences, easier to capture long term dependencies
    - Less computationally expensive to train

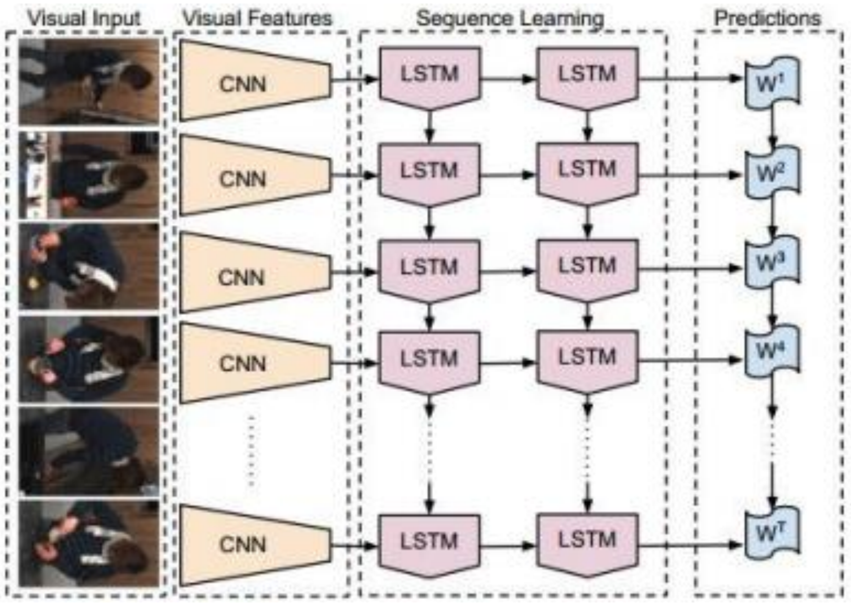
Machine translation

- RNN: Many-to-many (encoder-decoder)
- Tips which may help:
  - Different W for the encoder and the decoder
  - Concatenate the last hidden state of the encoder to all hidden state in the decoder
  - In the decoder, pass in y_n as x_{n+1}
  - Reverse the order:
    - ABC $\rightarrow$ XY
    - CBA $\rightarrow$ XY



CNN + RNN

- With RNN on top of CNN features, you capture a larger time horizon



Long-term Recurrent Convolutional Networks for Visual Recognition and Description

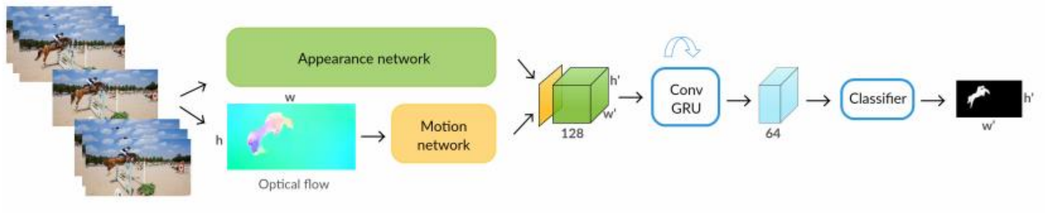
A network can be both convolutional and recurrent

- By simply changing the matrix multiplications to convolutions

$$
\begin{aligned}
\mathbf{z}_t &= \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1}),\\
\mathbf{r}_t &= \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1}),\\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1}))\\
\mathbf{h}_t &= (1-\mathbf{z}_t)\mathbf{h}_{t-1} + \mathbf{z}_t \tilde{\mathbf{h}}_t,
\end{aligned}
$$

$$\downarrow$$

$$
\begin{aligned}
\mathbf{z}_t^l &= \sigma(\mathbf{W}_z^l * \mathbf{x}_t^l + \mathbf{U}_z^l * \mathbf{h}_{t-1}^l),\\
\mathbf{r}_t^l &= \sigma(\mathbf{W}_r^l * \mathbf{x}_t^l + \mathbf{U}_r^l * \mathbf{h}_{t-1}^l),\\
\tilde{\mathbf{h}}_t^l &= \tanh(\mathbf{W}^l * \mathbf{x}_t^l + \mathbf{U} * (\mathbf{r}_t^l \odot \mathbf{h}_{t-1}^l))\\
\mathbf{h}_t^l &= (1-\mathbf{z}_t^l)\mathbf{h}_{t-1}^l + \mathbf{z}_t^l \tilde{\mathbf{h}}_t^l,
\end{aligned}
$$



Learning Video Object Segmentation with Visual Memory