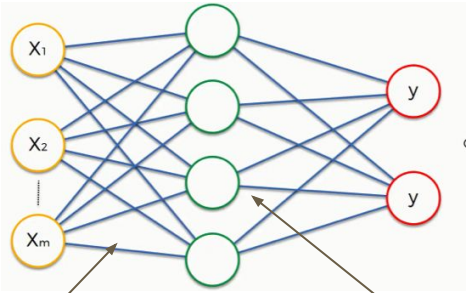
Deep Learning

6. Recurrent Neural Network

RECAP

Fully connected network



$$a = \sigma(W_1 x + b_1)$$

$$p = \sigma(W_2 a + b_2)$$

non-linearity

ConvNet

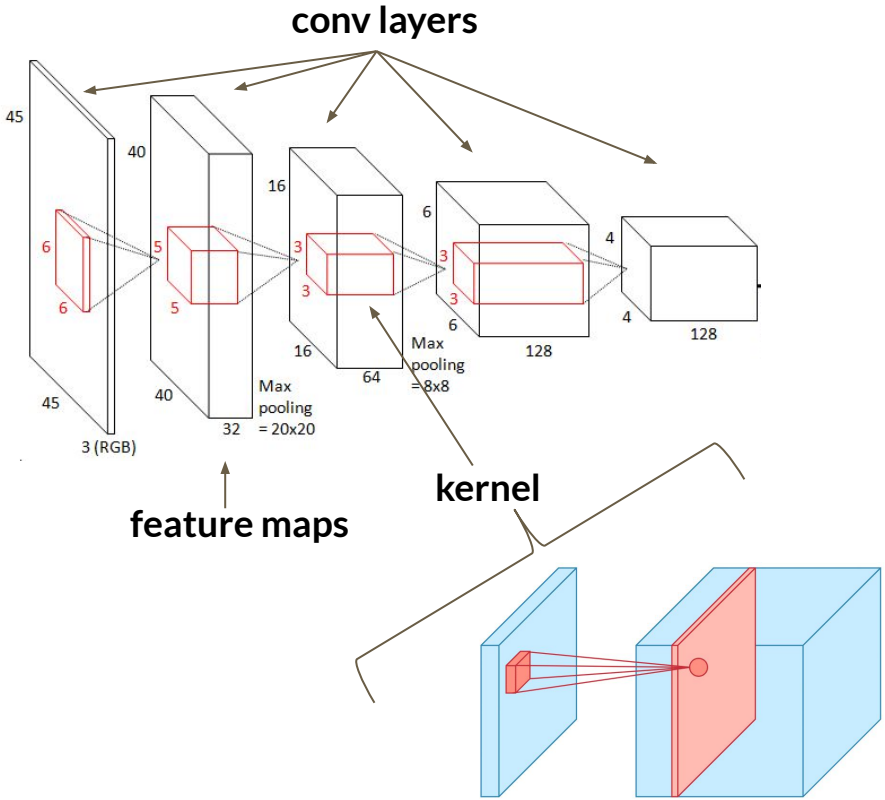
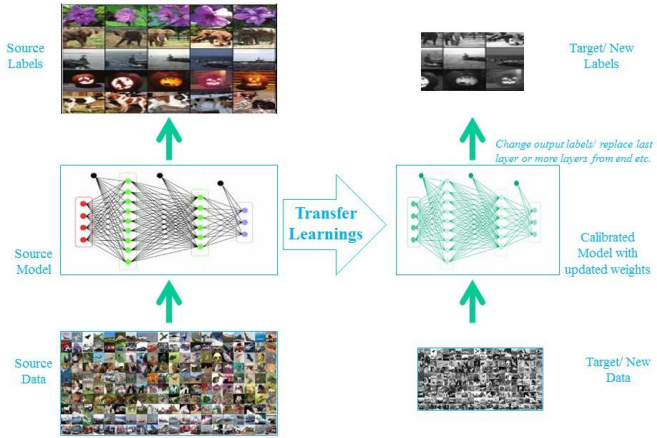


Image from https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection

Image from http://benanne.github.io/images

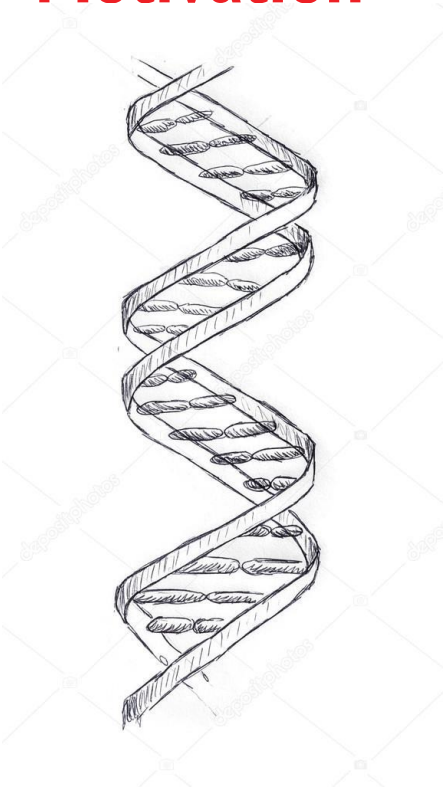
RECAP

Transfer / Self-Supervision



Photos from: http://deeplearning.net/tutorial/fcn_2D_segm.html; https://medium.com/@wilburdes/semantic-segmentation-using-fully-convolutional-neural-networks-86e45336f99b

Motivation



- Until now we have:
  - FCN: unstructured data (left)
  - CNN: localized data (right)

- BUT sometimes we have **sequential data**
  - Input is a sequence
  - Output is a sequence

Text generation

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

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

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

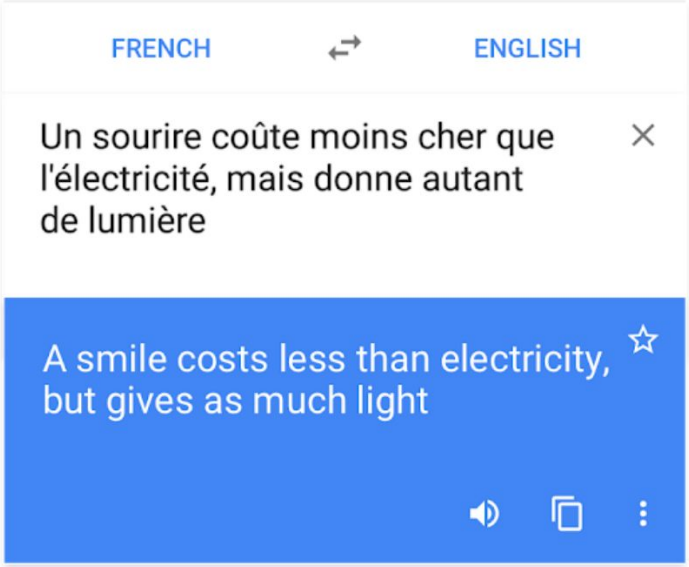
(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

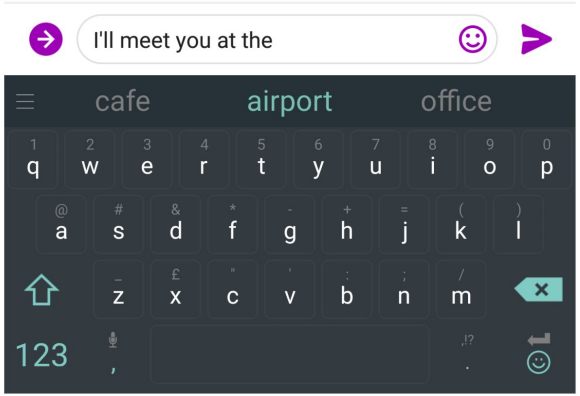
Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

```
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.
```

A. Karpathy and J. Johnson

Machine Translation

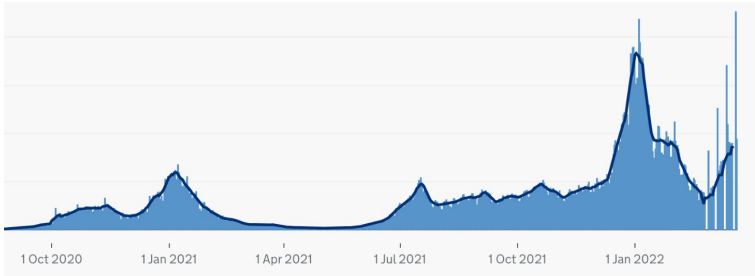


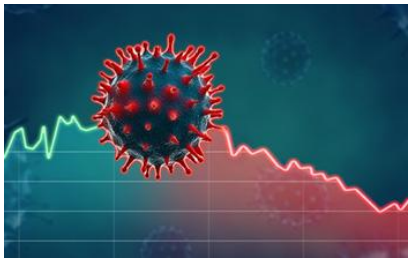


Wu, Yonghui et al. “Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.”

Time series prediction

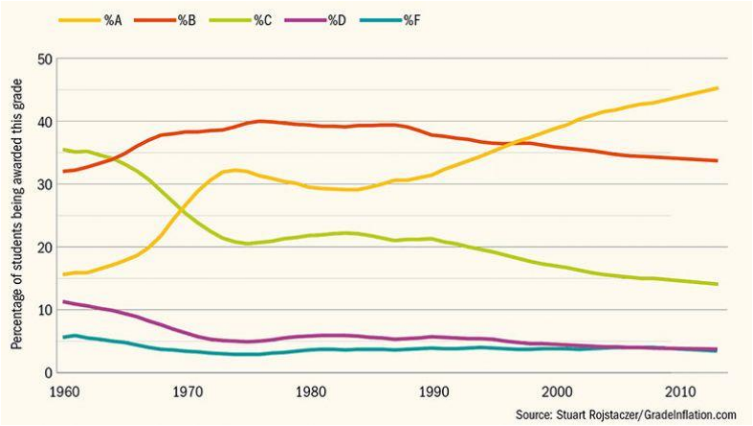
Number of covid cases in UK





ArunKumar et al, Forecasting of COVID-19 using deep layer Recurrent Neural Networks (RNNs) with Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) cells

Percentage of students being awarded each grade



Patil et al, Effective Deep Learning Model to Predict Student Grade Point Averages, ICCIC 2017

Graphics: https://coronavirus.data.gov.uk/details/cases https://coronavirus.data.gov.uk/ Image: https://nouvelles.umontreal.ca/en/article/2020/05/01/far-more-cases-than-officially-reported/; https://www.timeshighereducation.com/features/grade-inflation-worldwide-trend

Learning to execute

Input:

```
f=4692
for x in range(4):f-=1664
j=1443
for x in range(8):j+=f
d=j
for x in range(11):d-=4699
print(d).
```

Target: -65958.

Input:

```
f=136315;
h=(f+37592);
g=418652;
print((g-(h+234728))).
```

Target: 10017.

Zaremba, et al. "Learning to execute." arXiv preprint arXiv:1410.4615 (2014)

Captioning

O. Vinyals et al. " Show and Tell: A Neural Image Caption Generator"(2015).

I. Duta, A. Nicolicioiu, V. Bogolin, M. Leordeanu . "Mining for meaning: from vision to language through multiple networks consensus"(2018).

Video classification:

EAT

SCORE

LONG JUMP

Video classification:

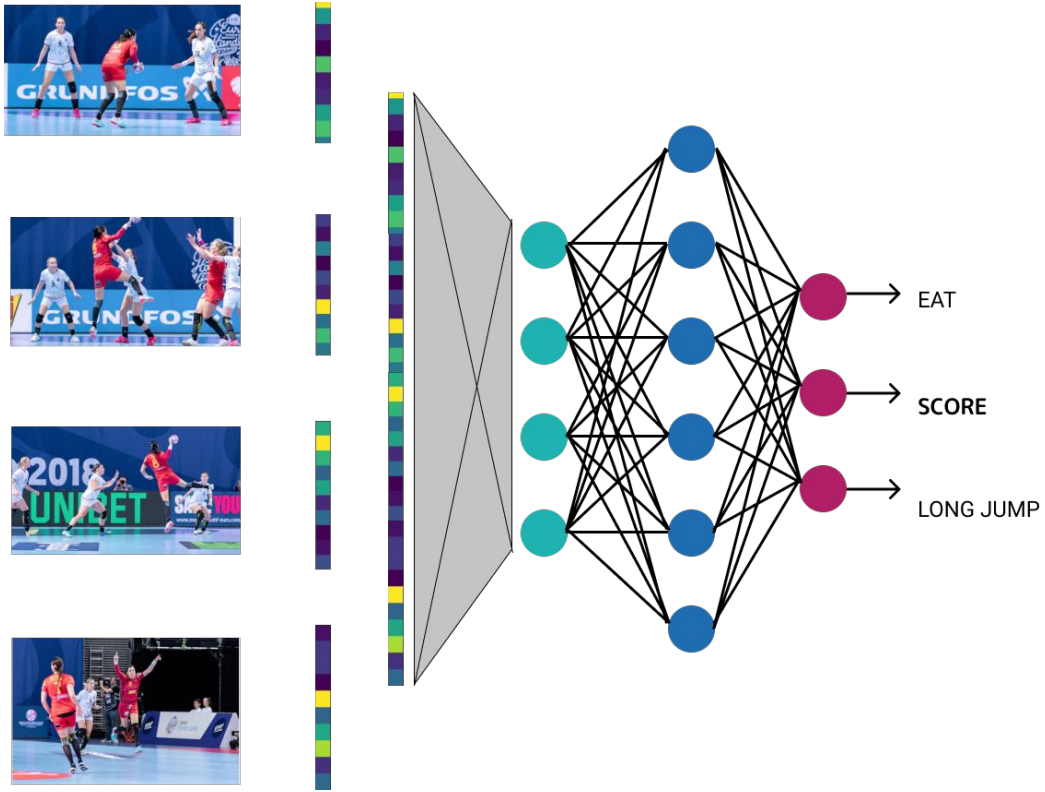
EAT

SCORE

LONG JUMP

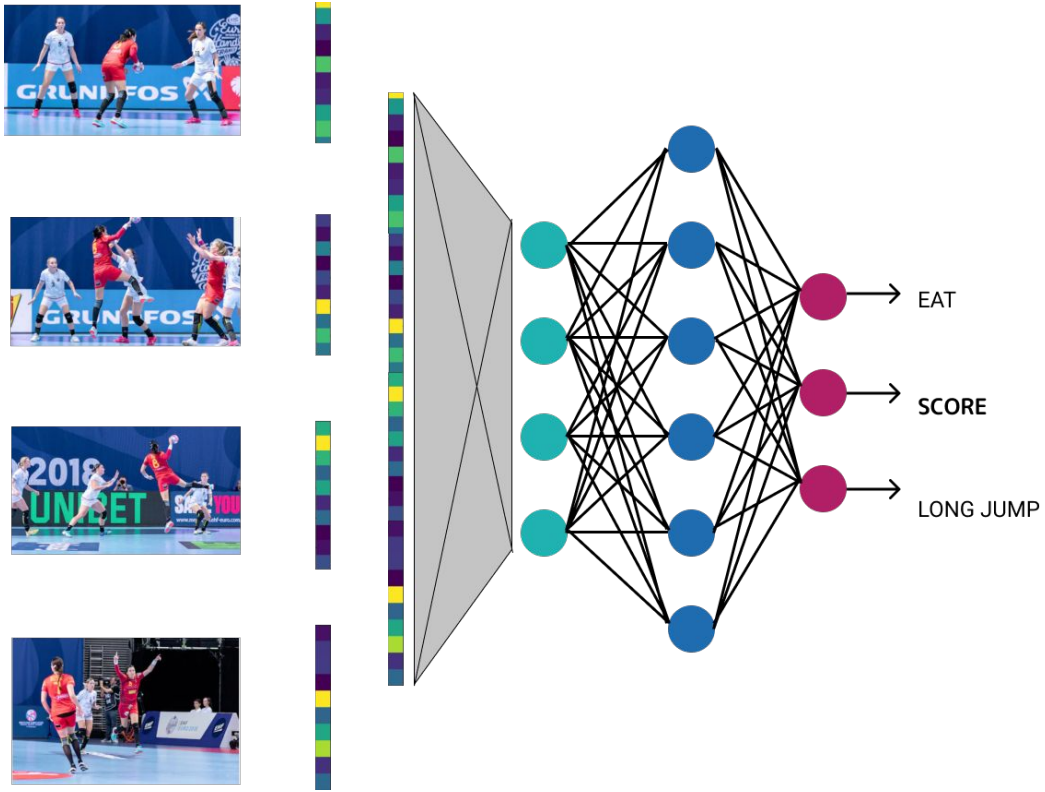
- for each frame extract features using **CNN**
- concatenate all the features in a **huge** video representation vector

Video classification:



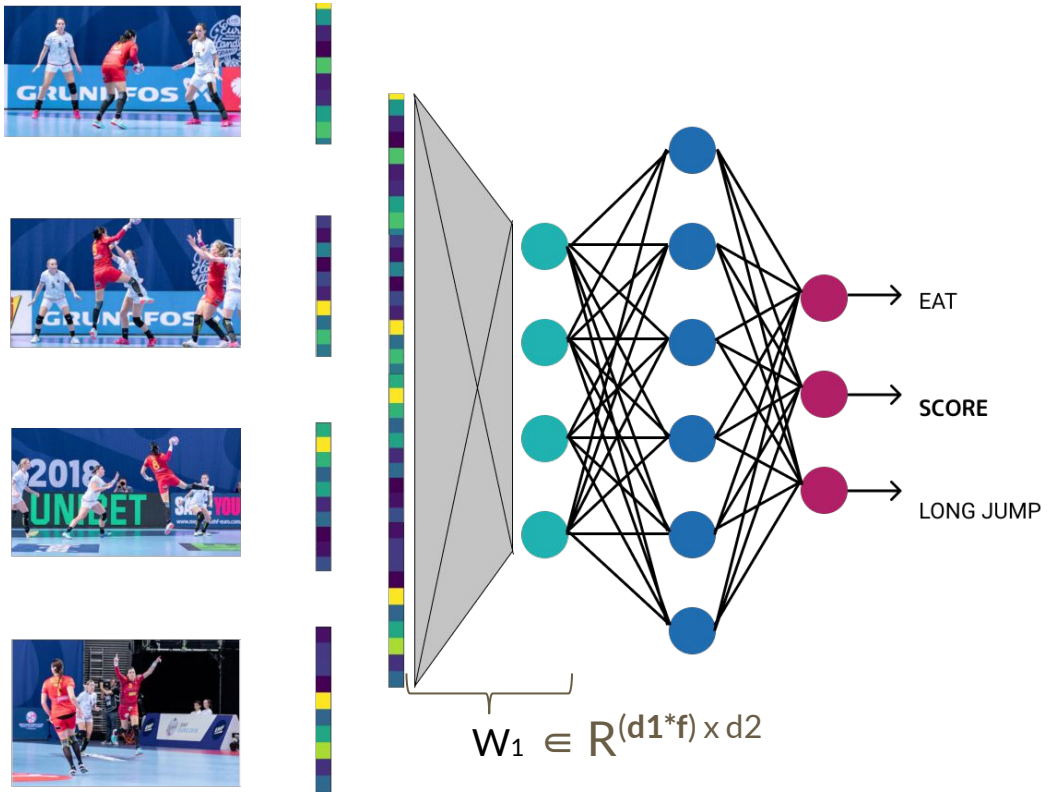
- for each frame extract features using **CNN**
- concatenate all the features in a **huge** video representation vector
- process the video representation using **fully connected layers**

Video classification:



Why not?

Video classification:



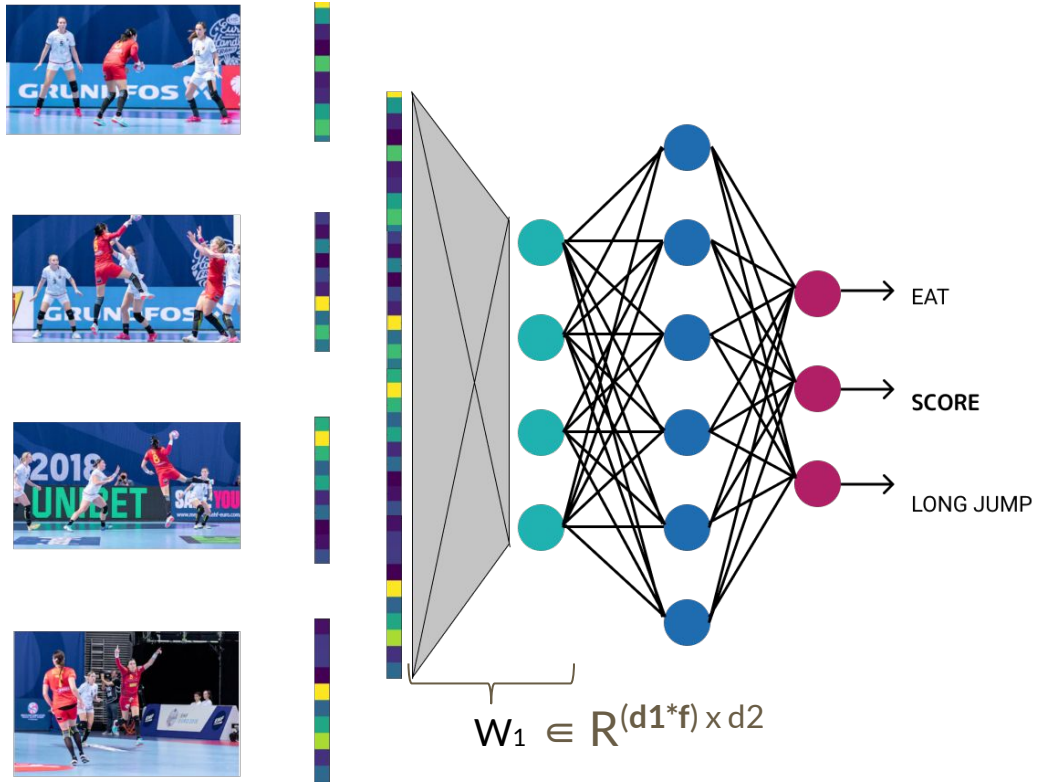
Why not?

Video classification:

?

- EAT
- **SCORE**
- LONG JUMP

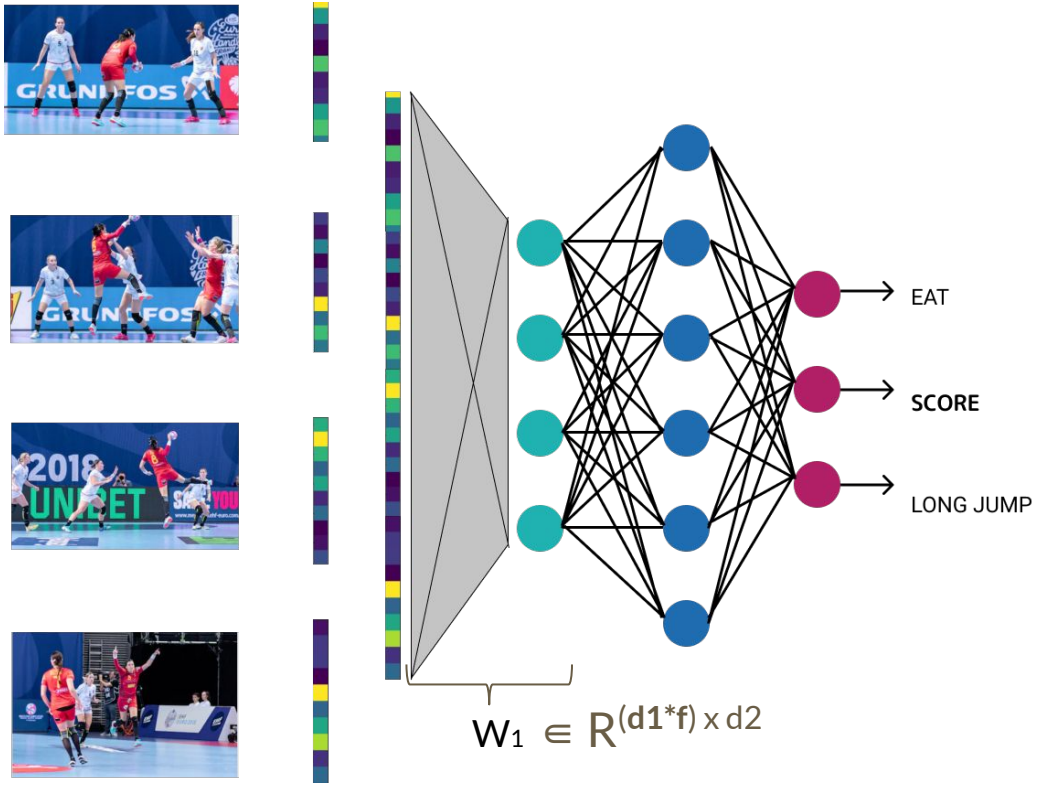
Video classification:



Why not?

- **huge** number of parameters

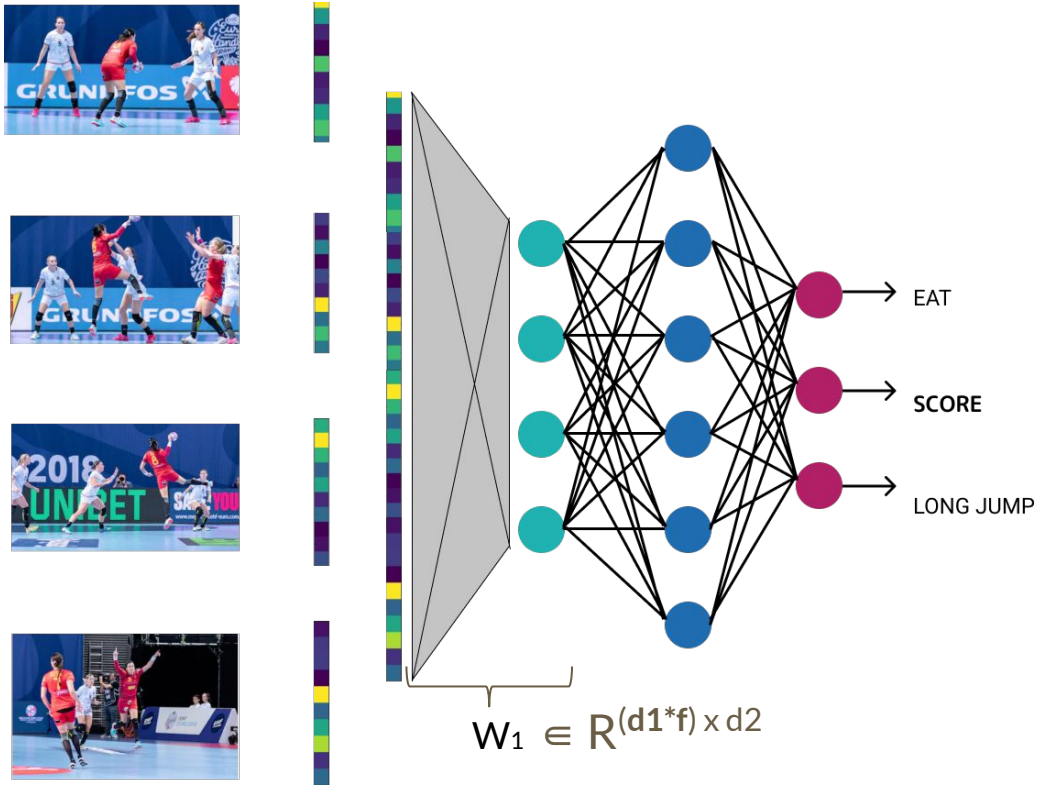
Video classification:



Why not?

- **huge** number of parameters
- **no** sequentiality

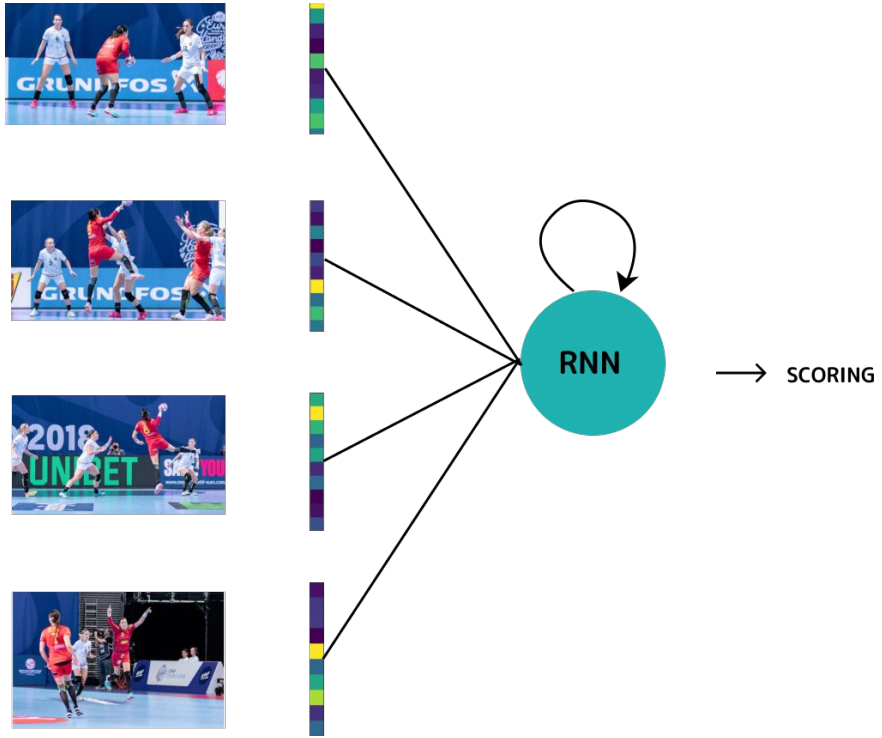
Video classification:



Why not?

- **huge** number of parameters
- **no** sequentiality
- **fixed** input length

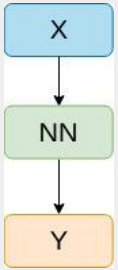
Video classification:



Solution:

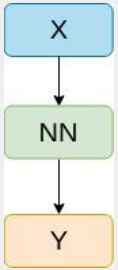
Recurrent Neural Network

Feed-forward vs RNN

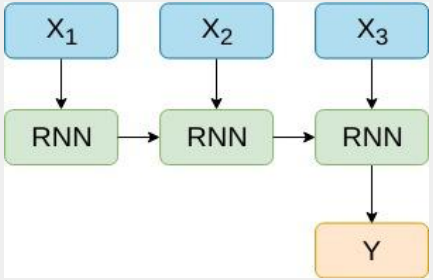


ONE - TO - ONE

Feed-forward vs RNN

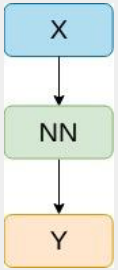


ONE - TO - ONE

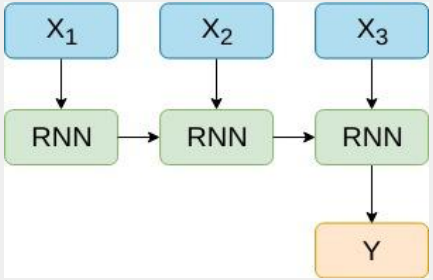


MANY - TO - ONE

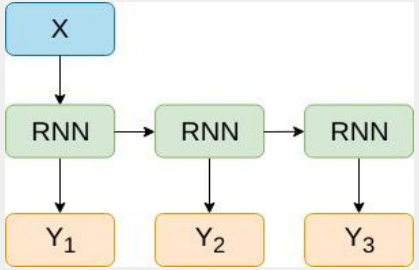
Feed-forward vs RNN



ONE - TO - ONE

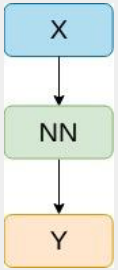


MANY - TO - ONE

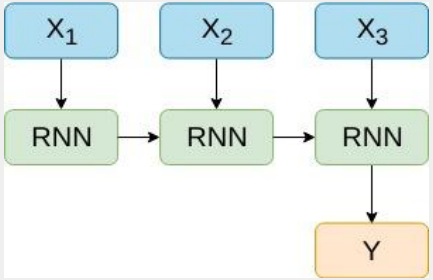


ONE - TO - MANY

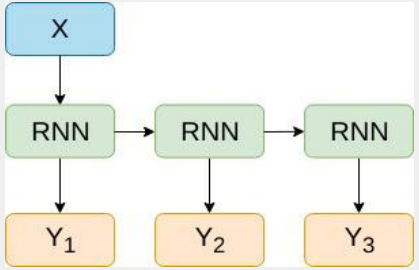
Feed-forward vs RNN



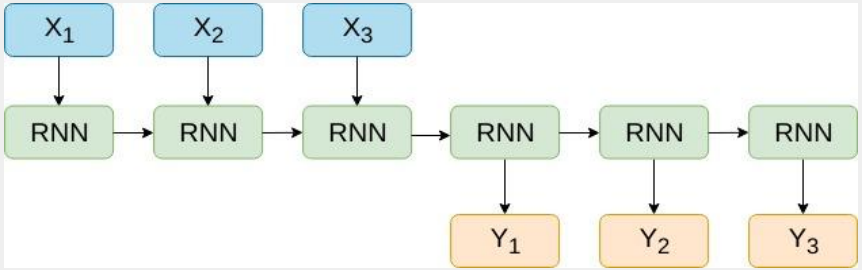
ONE - TO - ONE



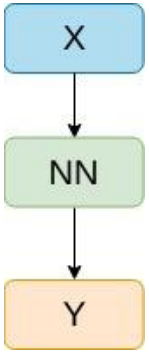
MANY - TO - ONE



ONE - TO - MANY



MANY - TO - MANY

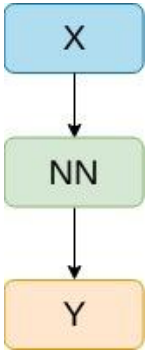


$$x \rightarrow y$$

$$y = f_W(x)$$

$$y = pool(x * W_1) * W_2$$

$$y = \sigma(W_2^T \sigma(W_1^T x + b_1) + b_2)$$

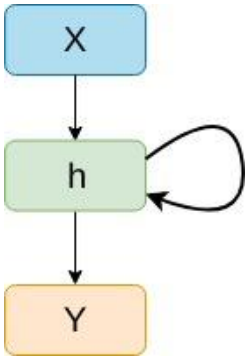


$$x \rightarrow y$$

$$y = f_W(x)$$

$$y = pool(x * W_1) * W_2$$

$$y = \sigma(W_2^T \sigma(W_1^T x + b_1) + b_2)$$



$$x_1, x_2 .. x_{T_1} \rightarrow y_1, y_2 .. y_{T_2}$$

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

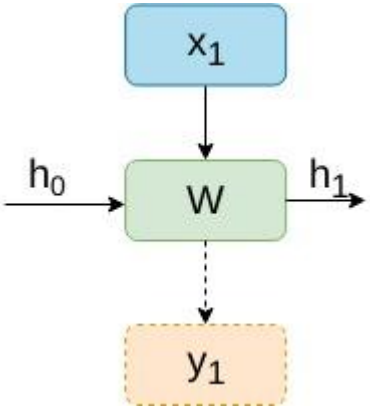
$$y_t = f_O(h_t)$$

x - *input*

h - *hidden state / context*

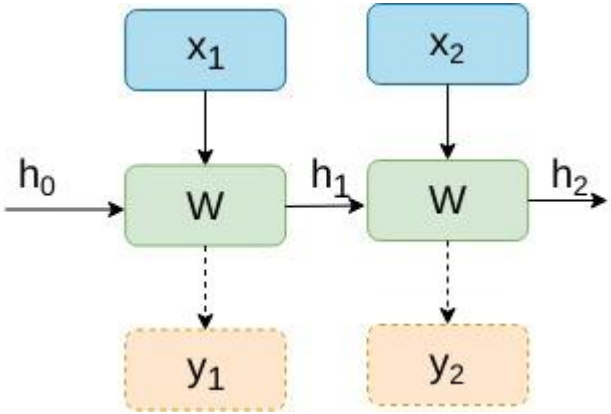
y - *output*

Forward step



$$h_1 = f_W(h_0, x_1)$$
$$[y_1 = f_O(h_1)]$$

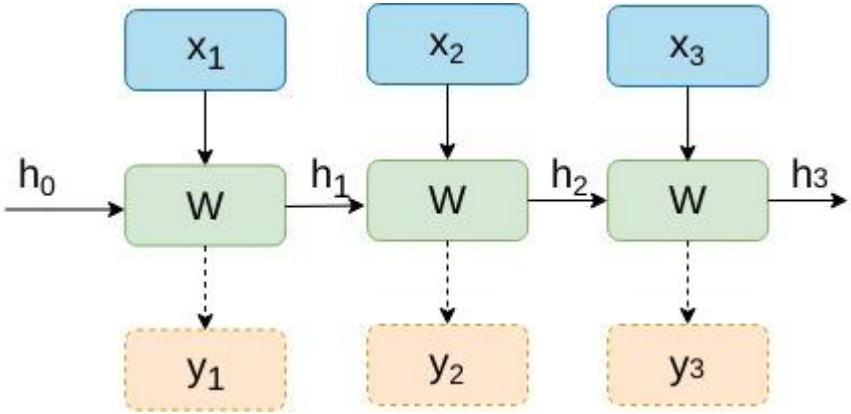
Forward step



$$h_1 = f_W(h_0, x_1)$$
$$[y_1 = f_O(h_1)]$$

$$h_2 = f_W(h_1, x_2)$$
$$[y_2 = f_O(h_2)]$$

Forward step

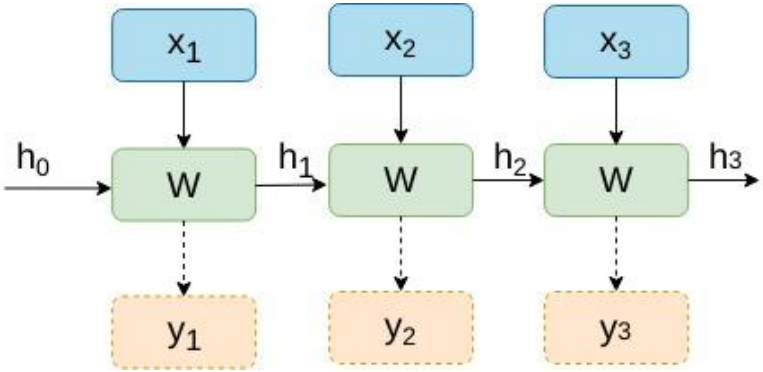


$$h_1 = f_W(h_0, x_1)$$
$$[y_1 = f_O(h_1)]$$

$$h_2 = f_W(h_1, x_2)$$
$$[y_2 = f_O(h_2)]$$

$$h_3 = f_W(h_2, x_3)$$
$$[y_3 = f_O(h_3)]$$

Forward step



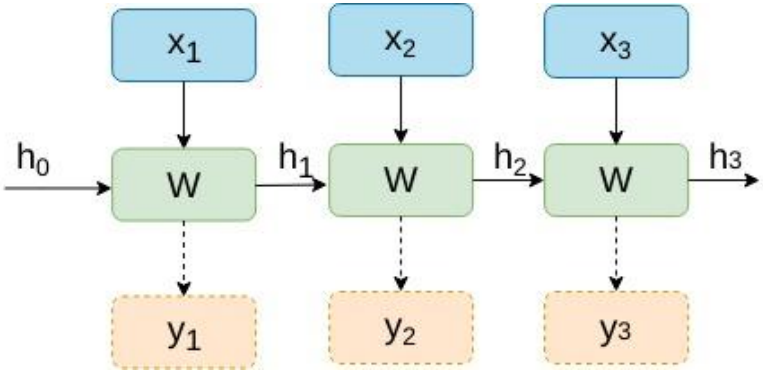
same W / same O at each time step

$$h_1 = f_W(h_0, x_1)$$
$$[y_1 = f_O(h_1)]$$

$$h_2 = f_W(h_1, x_2)$$
$$[y_2 = f_O(h_2)]$$

$$h_3 = f_W(h_2, x_3)$$
$$[y_3 = f_O(h_3)]$$

Forward step



$$h_1 = f_W(h_0, x_1)$$
$$[y_1 = f_O(h_1)]$$

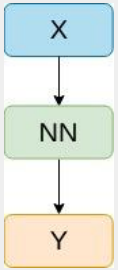
$$h_2 = f_W(h_1, x_2)$$
$$[y_2 = f_O(h_2)]$$

$$h_3 = f_W(h_2, x_3)$$
$$[y_3 = f_O(h_3)]$$

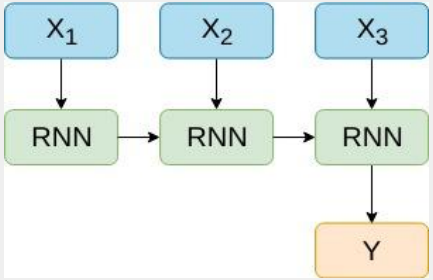
same W / same O at each time step

- allow input sequence of **variable lengths**
- given a context and an input, all timesteps should be **process in the same way**

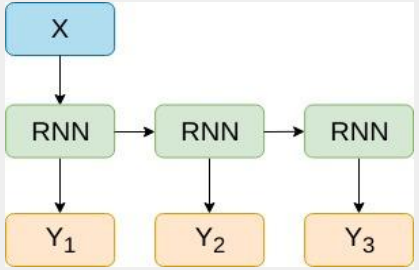
Feed-forward vs RNN



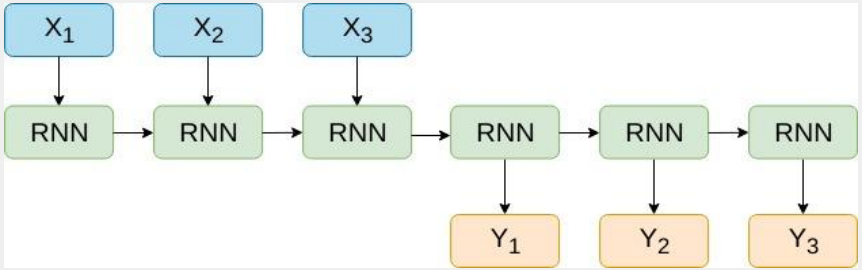
ONE - TO - ONE



MANY - TO - ONE

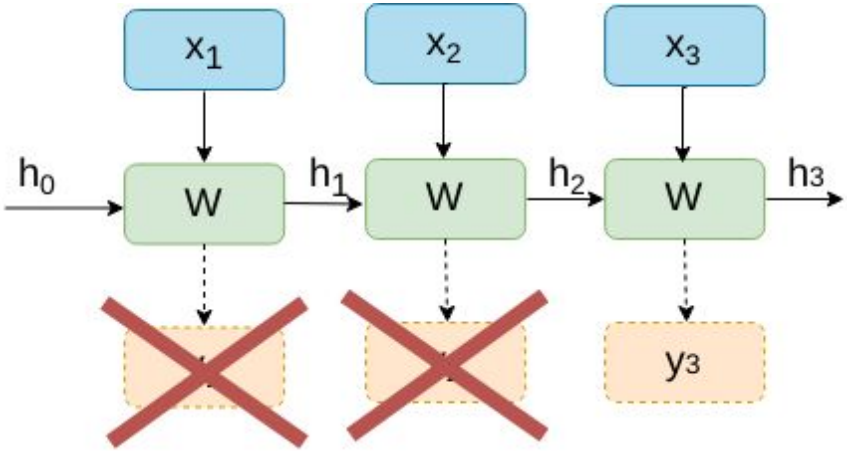


ONE - TO - MANY



MANY - TO - MANY

Forward step - single output



$$h_1 = f_W(h_0, x_1)$$

$$\cancel{[y_1 = f_O(h_1)]}$$

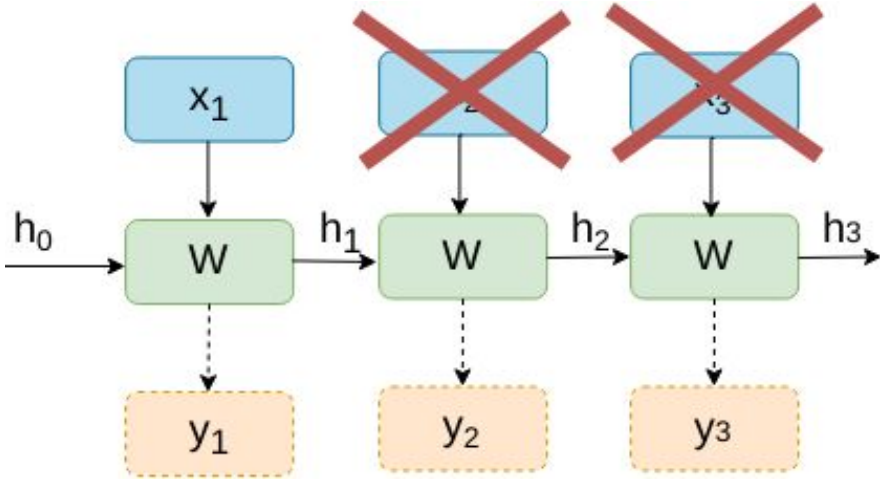
$$h_2 = f_W(h_1, x_2)$$

$$\cancel{[y_2 = f_O(h_2)]}$$

$$h_3 = f_W(h_2, x_3)$$

$$[y_3 = f_O(h_3)]$$

Forward step - single input



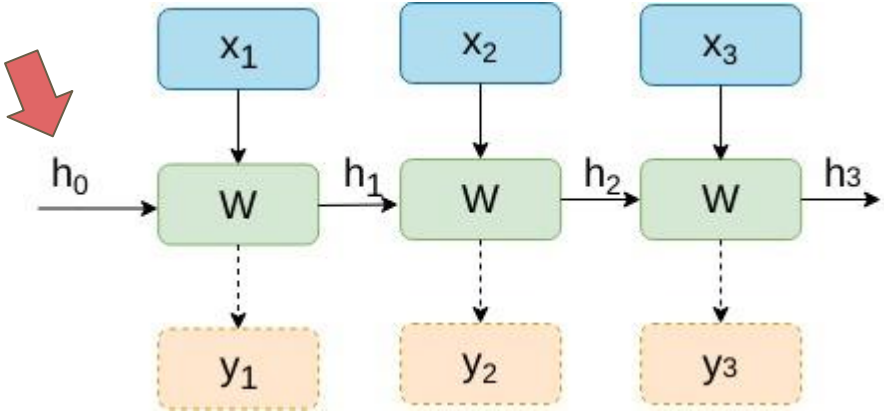
$$h_1 = f_W(h_0, x_1)$$
$$[y_1 = f_O(h_1)]$$

$$h_2 = f_W(h_1, x_2)$$
$$[y_2 = f_O(h_2)]$$

$$h_3 = f_W(h_2, x_3)$$
$$[y_3 = f_O(h_3)]$$

DUMMY INPUT

Forward step - initial state



What about h_0 ?

1. zeros vector
2. random vector
3. representation of data used as conditional information for current model

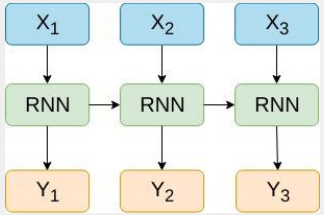
Vanilla RNN

$$\boxed{\begin{aligned} h_t &= f_W(h_{t-1}, x_t) \\ [y_t &= f_O(h_t)] \end{aligned}}$$

$$\begin{aligned} h_t &= \sigma(W_{hh} h_{t-1} + W_{xh} x_t) \\ [y_t &= W_{hy} h_t] \end{aligned} \quad \equiv \quad \begin{aligned} h_t &= \sigma((W_{hh}; W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \\ [y_t &= W_{hy} h_t] \end{aligned}$$

Vanilla RNN vs Multi-modal RNN

One-modal input



$$\boxed{\begin{aligned} h_t &= f_W(h_{t-1}, x_t) \\ [y_t &= f_O(h_t)] \end{aligned}}$$

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

$$[y_t = W_{hy} h_t]$$

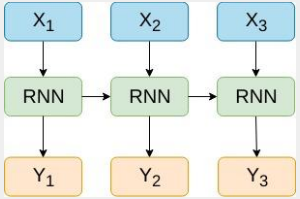
$$\equiv$$

$$h_t = \sigma((W_{hh}; W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix})$$

$$[y_t = W_{hy} h_t]$$

Vanilla RNN vs Multi-modal RNN

One-modal input



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

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

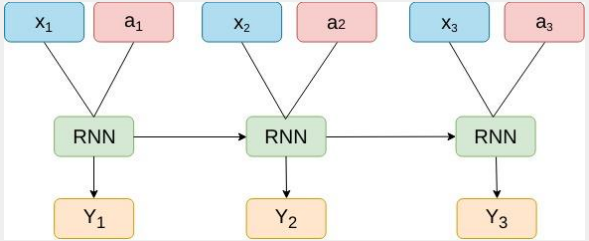
$$[y_t = W_{hy} h_t]$$

$$\equiv$$

$$h_t = \sigma((W_{hh}; W_{xh}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix})$$

$$[y_t = W_{hy} h_t]$$

Multi-modal input



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

$$h_t = \sigma(W_{hh} h_{t-1} + W_{xh} x_t + W_{ah} a_t + ..)$$

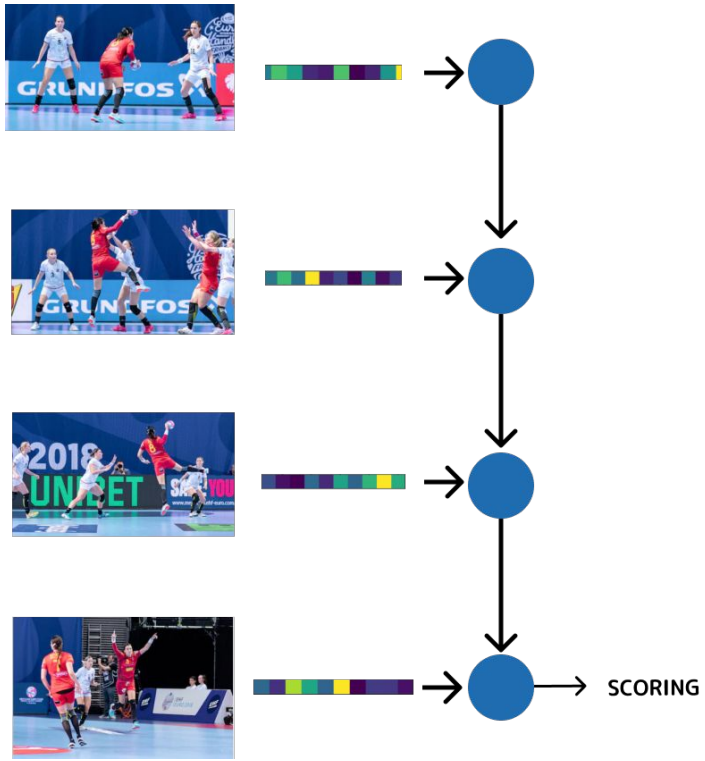
$$[y_t = W_{hy} h_t]$$

$$\equiv$$

$$h_t = \sigma((W_{hh}; W_{xh}; W_{ah}) \begin{pmatrix} h_{t-1} \\ x_t \\ a_t \\ \dots \end{pmatrix})$$

$$[y_t = W_{hy} h_t]$$

Video classification: RNN



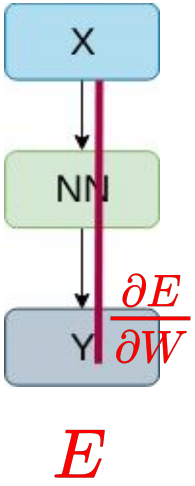
- one **CNN** to extract features
- one **RNN** to process the frames features
- output from the final step (many to one)

Let's take a little

coffee break

RECAP

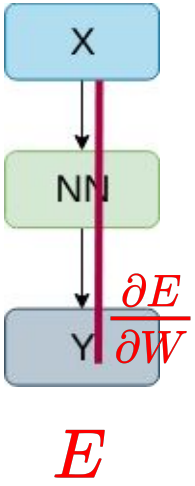
Backpropagation



- Loss function (E) measures how close we are to the optimal solution
- At train time, update parameters to minimize the loss
- The update is based on **gradient** descent

RECAP

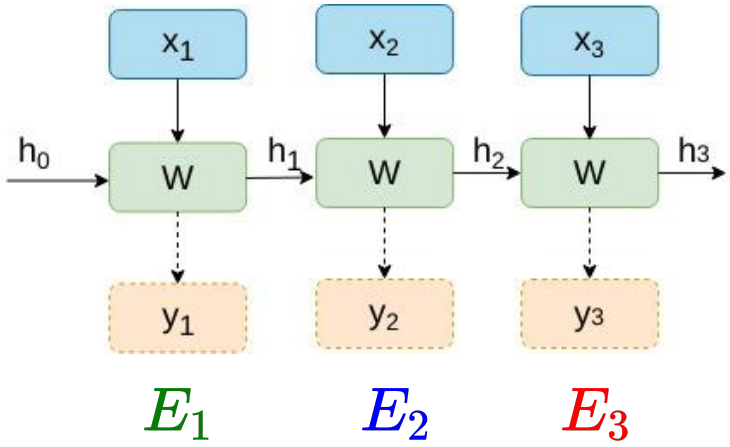
Backpropagation



- Loss function (E) measures how close we are to the optimal solution
- At train time, update parameters to minimize the loss
- The update is based on **gradient** descent

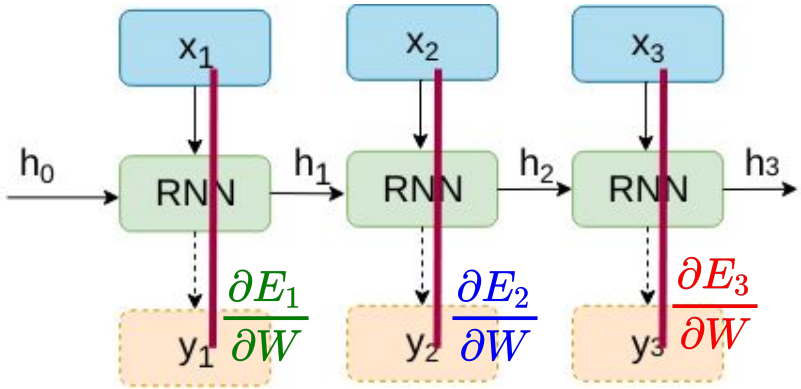
We need to compute $\frac{\partial E}{\partial W}$

Backpropagation through time (BPTT)



Fancy name for regular backpropagation on an unrolled RNN

Backpropagation through time (BPTT)



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

$$[y_t = W_{hy}\sigma(h_t)]$$

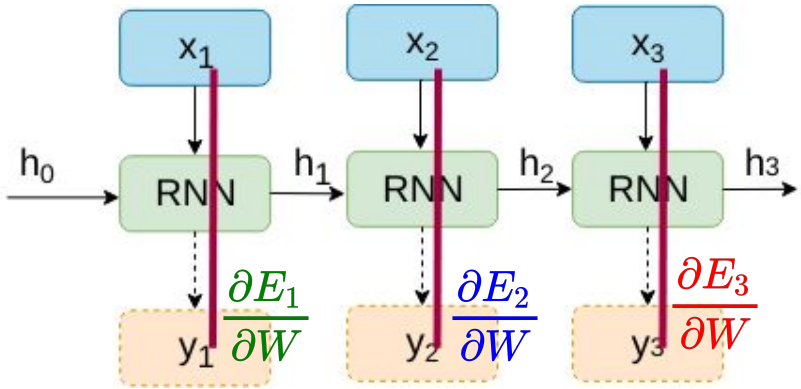
*

$$E = \sum_{t=1}^{T} E_t$$

$$E_t(y, y^*) = f_{loss}(y_t, y_t^*)$$

* Slightly change the way we define this, for an easier computation. The intuition remains the same

Backpropagation through time (BPTT)

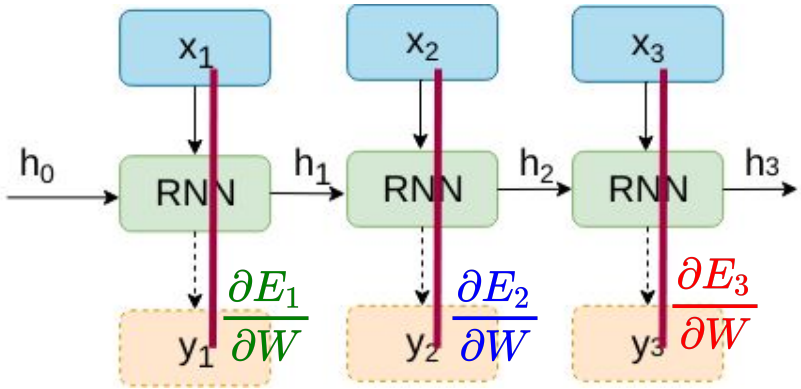


$$h_t = (W_{hh}\sigma(h_{t-1}) + W_{xh}x_t)$$
$$[y_t = W_{hy}\sigma(h_t)]$$

$$E = \sum_{t=1}^{T} E_t$$
$$E_t(y, y^*) = f_{loss}(y_t, y_t^*)$$

$$\frac{\partial E}{\partial W} = \frac{\partial}{\partial W}(\sum_{t=1}^{T} E_t) = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

Backpropagation through time (BPTT)



$$h_t = (W_{hh}\sigma(h_{t-1}) + W_{xh}x_t) = f(W, h_{t-1}, x_t)$$

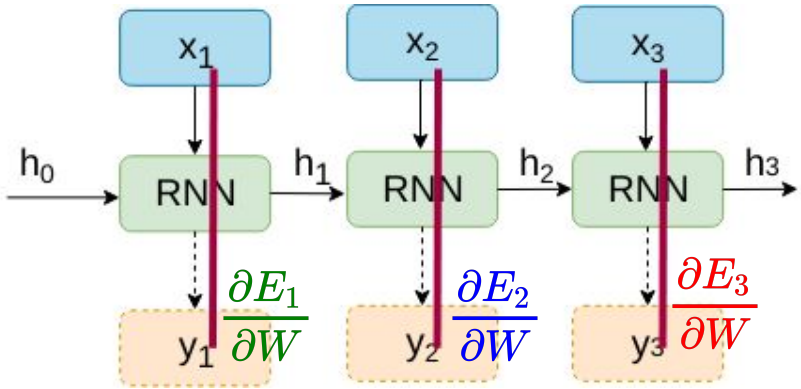
$$y_t = W_{hy}\sigma(h_t) = f(h_t)$$

$$E_t(y, y^*) = f_{loss}(y_t, y_t^*)$$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial W}$$

Backpropagation through time (BPTT)



$$h_t = (W_{hh}\sigma(h_{t-1}) + W_{xh}x_t) = f(W, h_{t-1}, x_t)$$

$$y_t = W_{hy}\sigma(h_t) = f(h_t)$$

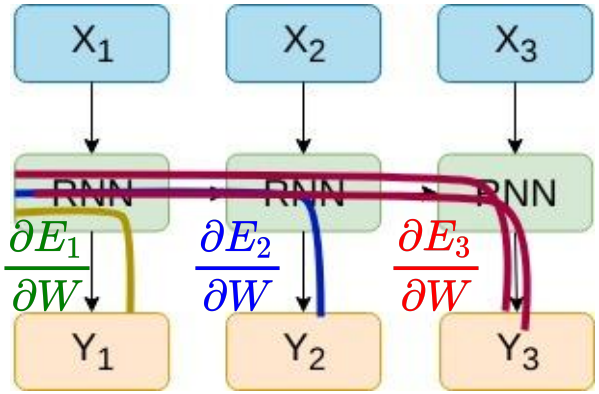
$$E_t(y, y^*) = f_{loss}(y_t, y_t^*)$$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial W}$$

NO!

Backpropagation through time (BPTT)



$$h_{t-1} = (W_{hh}\sigma(h_{t-2}) + W_{xh}x_{t-1})$$

$$h_t = (W_{hh}\sigma(h_{t-1}) + W_{xh}x_t) = f(W, h_{t-1}, x_t)$$

$$y_t = W_{hy}\sigma(h_t) = f(h_t)$$

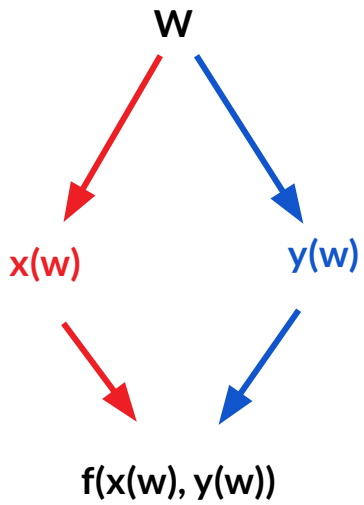
$$E_t(y, y^*) = f_{loss}(y_t, y_t^*)$$

$$\cancel{\frac{\partial E_t}{\partial W}} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t}\frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

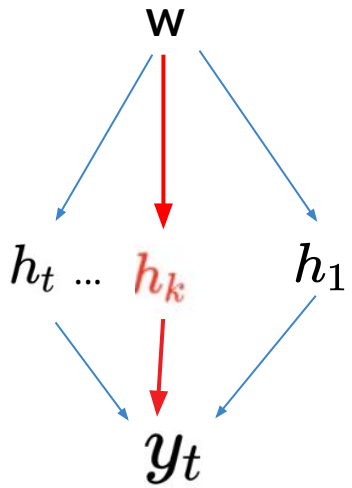
YES!

Multivariable Chain Rule



$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial w} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial w}$$

Multivariable Chain Rule

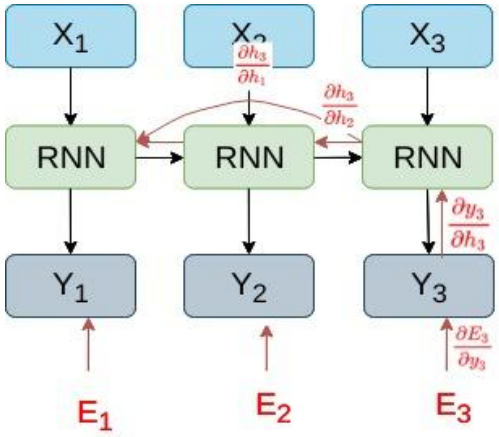


$$h_t = (W_{hh}\sigma(h_{t-1}) + W_{xh}x_t) = f(W, h_{t-1}(W), x_t)$$

$$y_t = W_{hy}\sigma(h_t) = f(h_t)$$

$$\frac{\partial y_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial y_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Backpropagation through time (BPTT)



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

$$[y_t = W_{hy}\sigma(h_t)]$$

$$E_t(y^*, y) = f_{loss}(y_t^*, y_t)$$

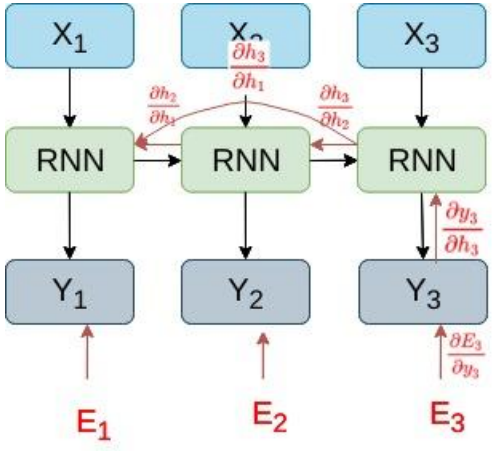
$$E = \sum_{t=1}^{T} E_t$$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

Backpropagation through time (BPTT)



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

$$[y_t = W_{hy}\sigma(h_t)]$$

$$E_t(y^*, y) = f_{loss}(y_t^*, y_t)$$

$$E = \sum_{t=1}^{T} E_t$$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial W}$$

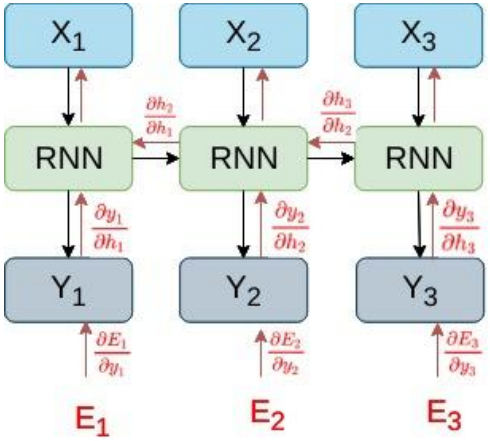
$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\boxed{\frac{\partial h_t}{\partial h_k}}\frac{\partial h_k}{\partial W}$$

$$= \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\boxed{\left(\prod_{j=k+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}\right)}\frac{\partial h_k}{\partial W}$$

$$= \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\left(\prod_{j=k+1}^{t}(W^T diag[\sigma'(h_{j-1})])\right)\frac{\partial h_k}{\partial W}$$

Backpropagation through time



$$\frac{\partial E}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \times$$

$$\times \left(\prod_{j=k+1}^{t} (W^T diag[\sigma'(h_{j-1})]) \right) \frac{\partial h_k}{\partial W}$$

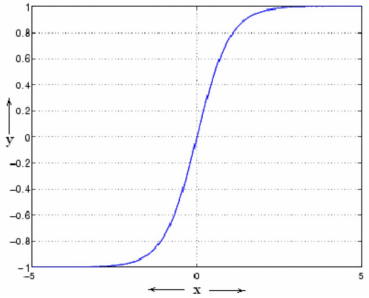
$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} [W^T diag[\sigma'(h_{j-1})]$$

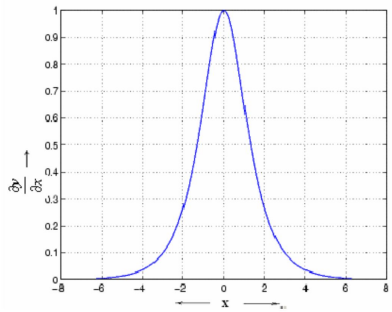
$$\left| \frac{\partial h_t}{\partial h_k} \right| \leq \prod_{j=k+1}^{t} |W^T| |diag[\sigma'(h_{j-1})]|$$

Let's take a little

coffee break

Backpropagation through time





$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} [W^T diag[\sigma'(h_{j-1})]$$

$$|\frac{\partial h_t}{\partial h_k}| \leq \prod_{j=k+1}^{t} |W^T||diag[\sigma'(h_{j-1})]|$$

For large number of timesteps (t):

$$\text{if } |W^T| < 1 \rightarrow 0$$ vanishing gradients

image from: https://hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4

Backpropagation through time

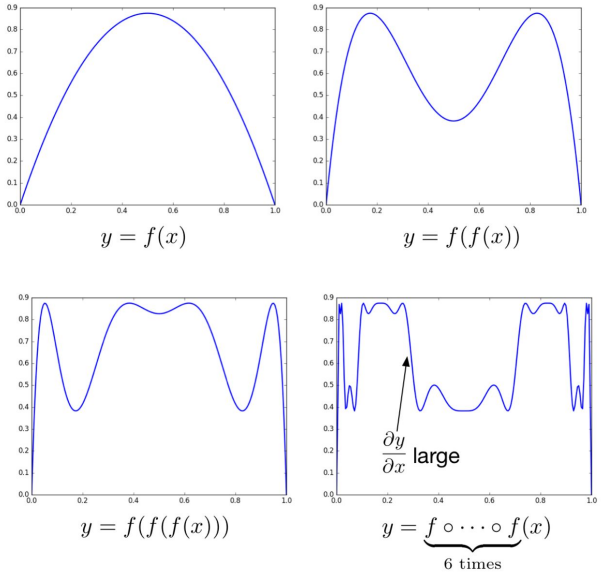


Figure 2: Iterations of the function $f(x) = 3.5\,x\,(1-x)$.

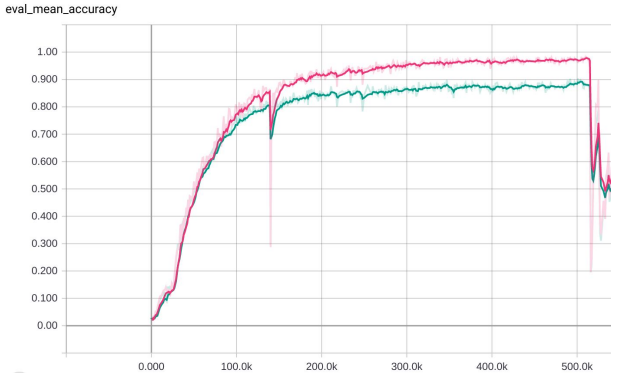
$$\left|\frac{\partial h_t}{\partial h_k}\right| \leq \prod_{j=k+1}^{t} |W^T||diag[\sigma'(h_{j-1})]|$$

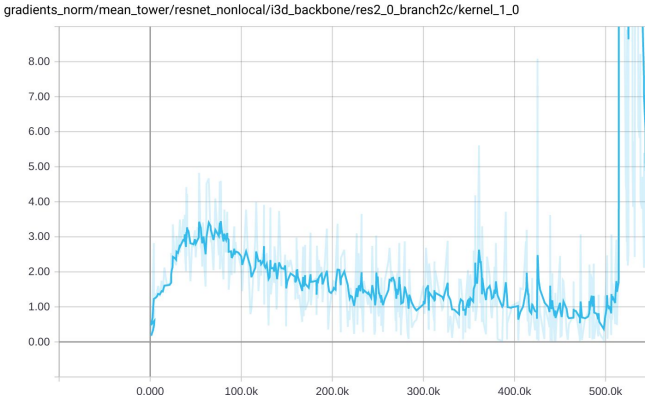
For large number of timesteps (t):

$$\begin{cases} \text{if } |W^T| < 1 \rightarrow 0 & \text{vanishing gradients} \\ \text{if } |W^T| > 1 \rightarrow \infty & \text{exploding gradients} \end{cases}$$

image from Toronto University CSC321 course 2017, R. Grosse

Backpropagation through time





$$\left|\frac{\partial h_t}{\partial h_k}\right| \leq \prod_{j=k+1}^{t} |W^T||diag[\sigma'(h_{j-1})]|$$

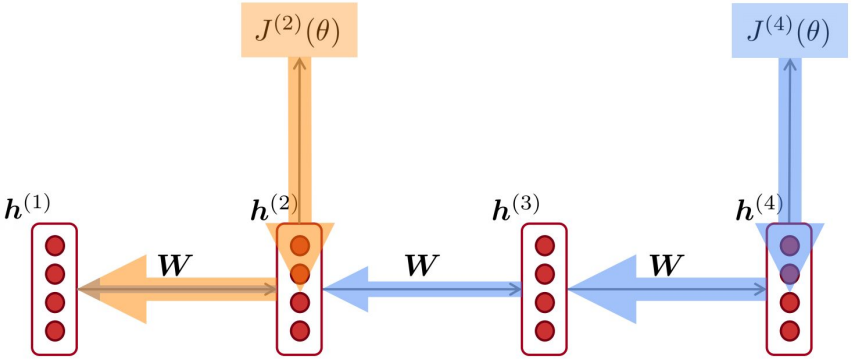
For large number of timesteps (t):

$$\begin{cases} \text{if } |W^T| < 1 \rightarrow 0 & \text{vanishing gradients} \\ \text{if } |W^T| > 1 \rightarrow \infty & \text{exploding gradients} \end{cases}$$

HARD TO TRAIN

WHY?

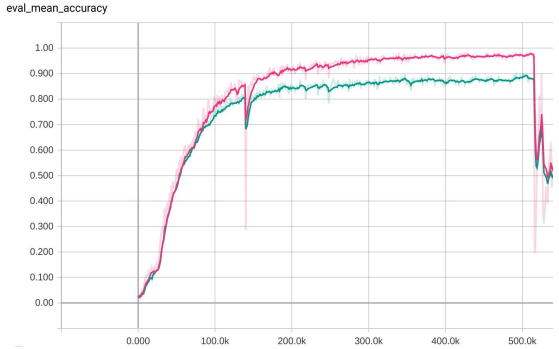
Vanishing gradients

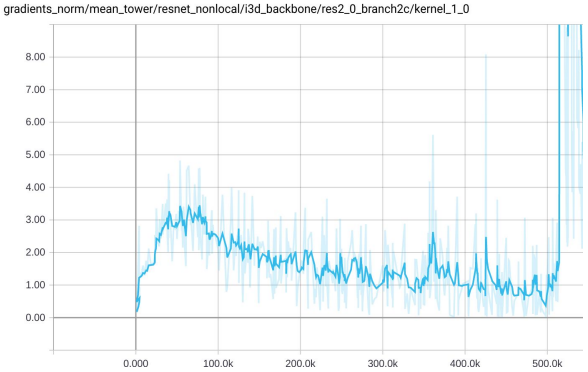


- unable to capture long-distance dependencies
- induce a bias towards recent events

Gradient can be seen as a measure for the impact of the past state for a future state

Exploding gradients



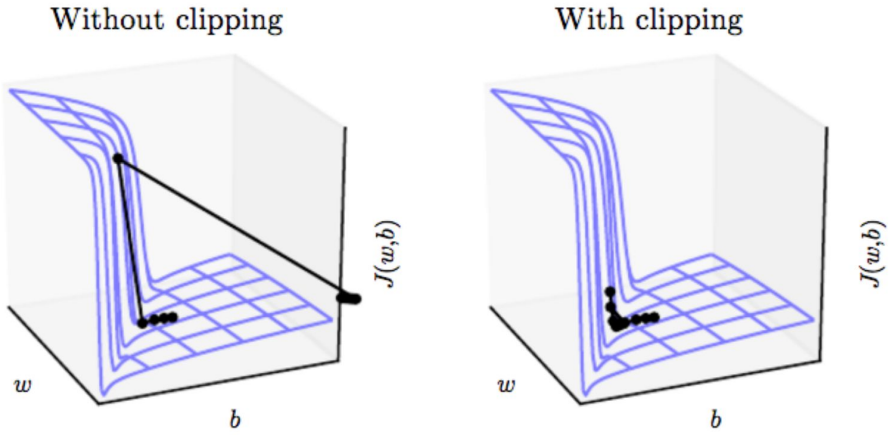


- Bad updates
- Huge step reaching a bad configuration
- Could result in NaN parameters

Vanishing / exploding gradients

Solution 1: Gradient clipping

$$g \leftarrow \begin{cases} \frac{\eta g}{|g|} & if \ \ |g| > \eta \\ g & otherwise \end{cases}$$



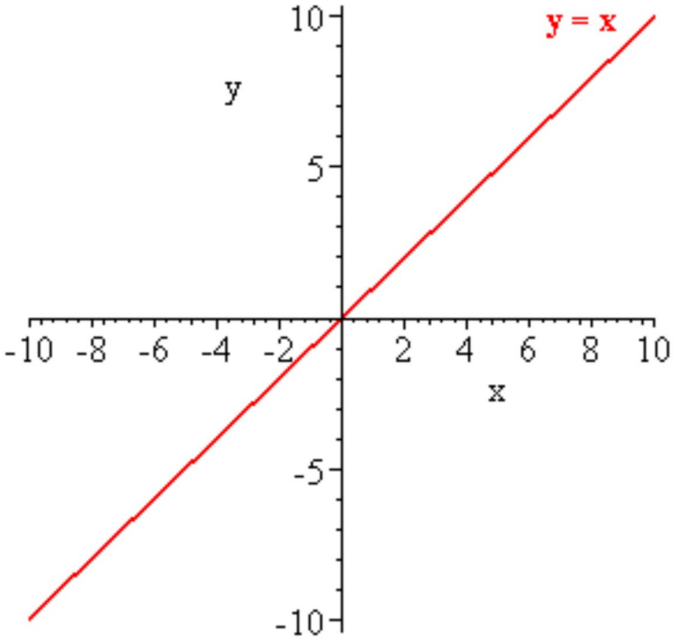
- avoid exploding gradients
- preserve direction, but a smaller step

Goodfellow et al. Deep Learning.

Vanishing / exploding gradients

Solution 2: Identity initialization

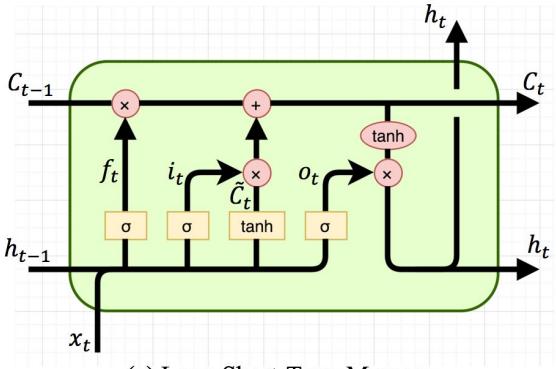
- recurrent weights are initialized to the **identity** matrix
- activation functions are **ReLU**



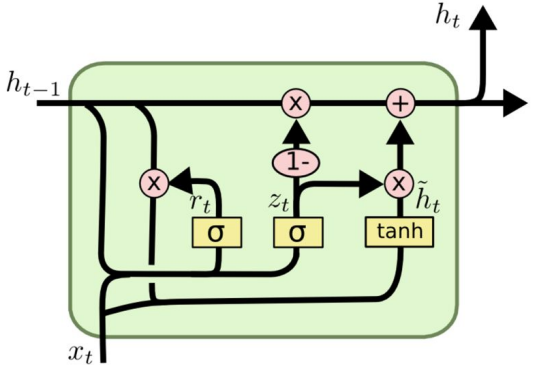
Vanishing / exploding gradients

Solution 3: Architectures:

- **LSTM**
- **GRU**



(a) Long Short-Term Memory



(b) Gated Recurrent Unit

- avoid vanishing gradients

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 1997)

Cho, Kyunghyun et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." *EMNLP* (2014).

Gating mechanism

Vanilla RNN

$$h_t = \sigma(W_{hh} h_{t-1} + W_{xh} x_t)$$
$$[y_t = W_{hy} h_t]$$

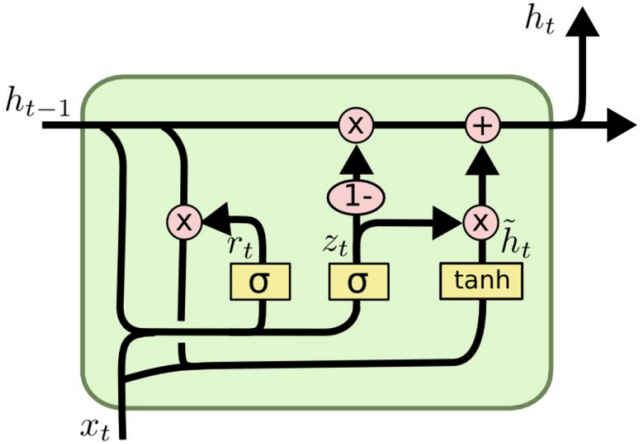
Simple Gate

$$g_t = \sigma(W_{hh} h_{t-1} + W_{xh} x_t)$$
$$c_t = g_t \odot c_{t-1} + (1 - g_t) \odot x_t$$
$$h_t = tanh(c_t)$$
$$y_t = W_{hy} h_t$$

- **Gates** control the flow of information into the cell, allowing the model to forget irrelevant information
  -> easier to retain long-term info (by setting gate to 0)

- $g_t \in [0, 1]$ $g_t = 0$ - dismiss all the information
  $g_t = 1$ - keep all the information

GRU



GRU use a gating mechanism with 2 gates:

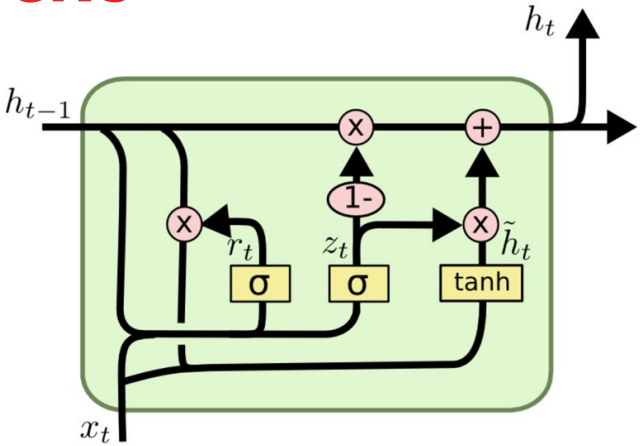
a. **update gate**: determine how much of the past information needs to be passed along to the future

$$z_t = \sigma(W^{(z)} x_t + U^{(z)} h_{t-1})$$

b. **reset gate**: determine what parts of previous hidden state are used to compute new content

$$r_t = \sigma(W^{(r)} x_t + U^{(r)} h_{t-1})$$

GRU



GRU use a gating mechanism with 2 gates:

a. **update gate**: determine how much of the past information needs to be passed along to the future

$$z_t = \sigma(W^{(z)} x_t + U^{(z)} h_{t-1})$$

b. **reset gate**: decide how to combine new input with previous state

$$r_t = \sigma(W^{(r)} x_t + U^{(r)} h_{t-1})$$

selects useful parts of prev hidden state to compute crt

integrate crt info into the previous one

$$\tilde{h}_t = tanh(W x_t + r_t \odot U h_{t-1})$$

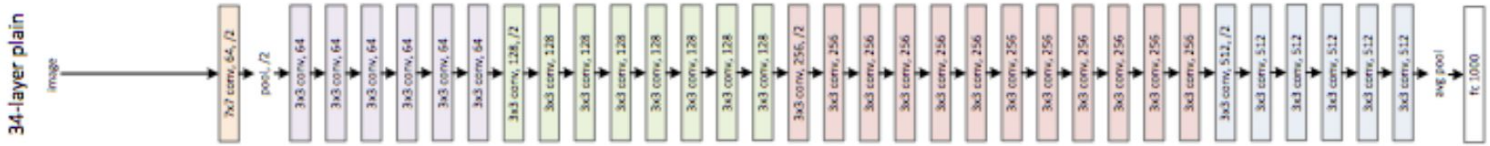
keep/reset previous info for current target state

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

copy/ignore previous info

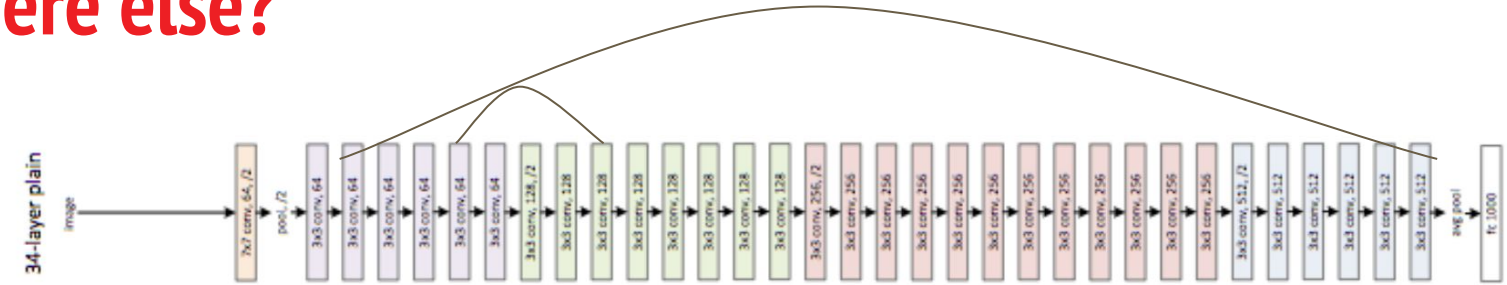
Where else?

Where else?



- can be a problem for any neural network
- use skip-connection to preserve identity
- Several architecture: *ResidualNet; DenseNet; HighwayNet*

Where else?

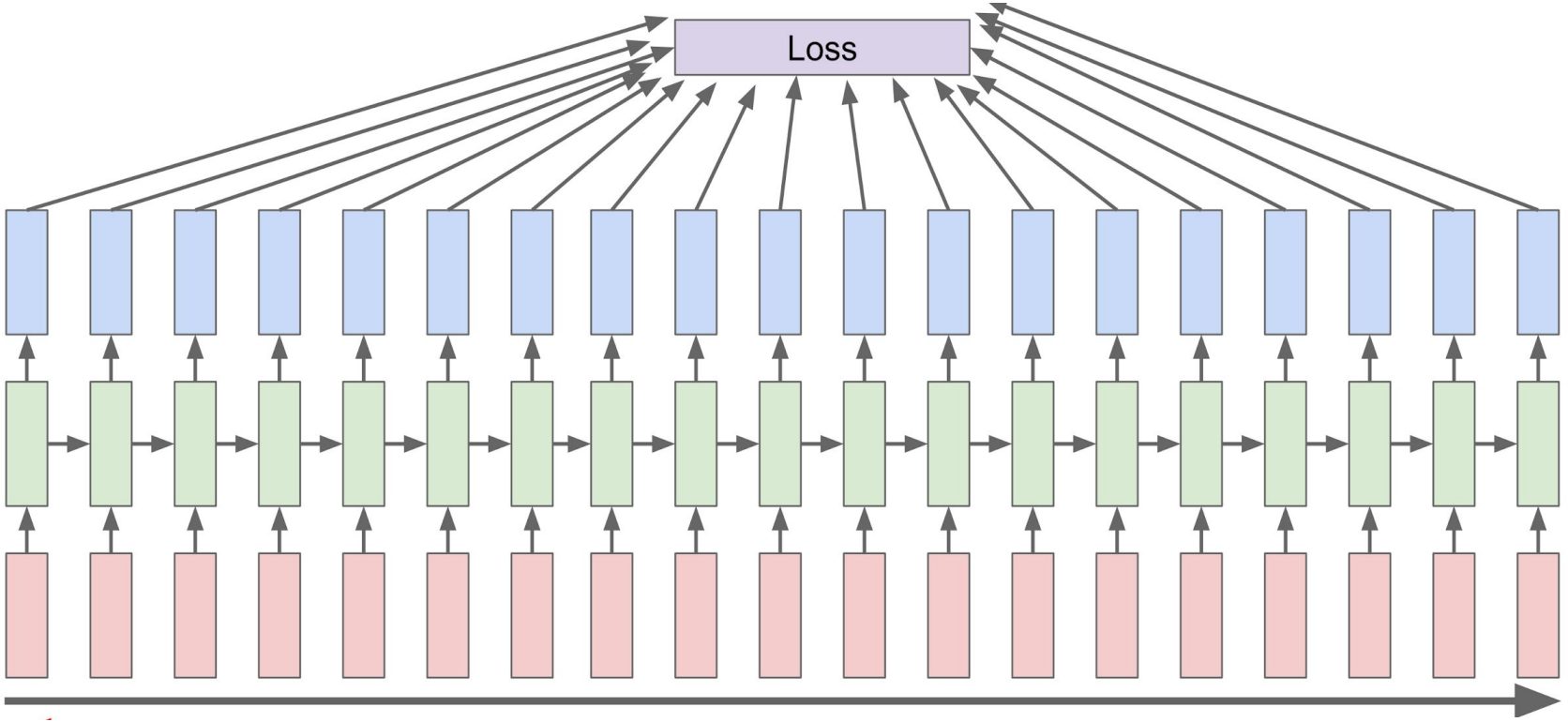


- can be a problem for any neural network
- use skip-connection to preserve identity
- Several architecture: *ResidualNet; DenseNet; HighwayNet*

Let's take a little

coffee break

Truncated backpropagation through time (TBPTT)



[slide from Stanford CS231 course 2018, Li Fei-Fei]

Truncated backpropagation through time (TBPTT)

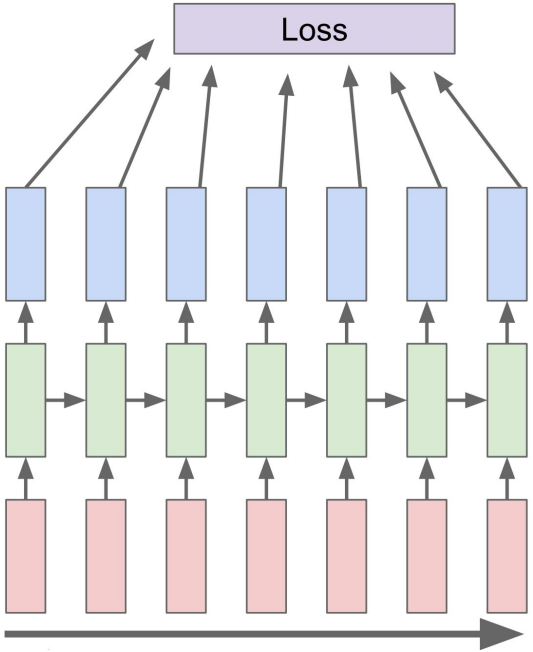
When the sequence is very long, BPTT accumulate gradients over many timesteps:

- learning is computationally expensive (slow)
- learning suffer from vanishing / exploding gradients

TBPTT:

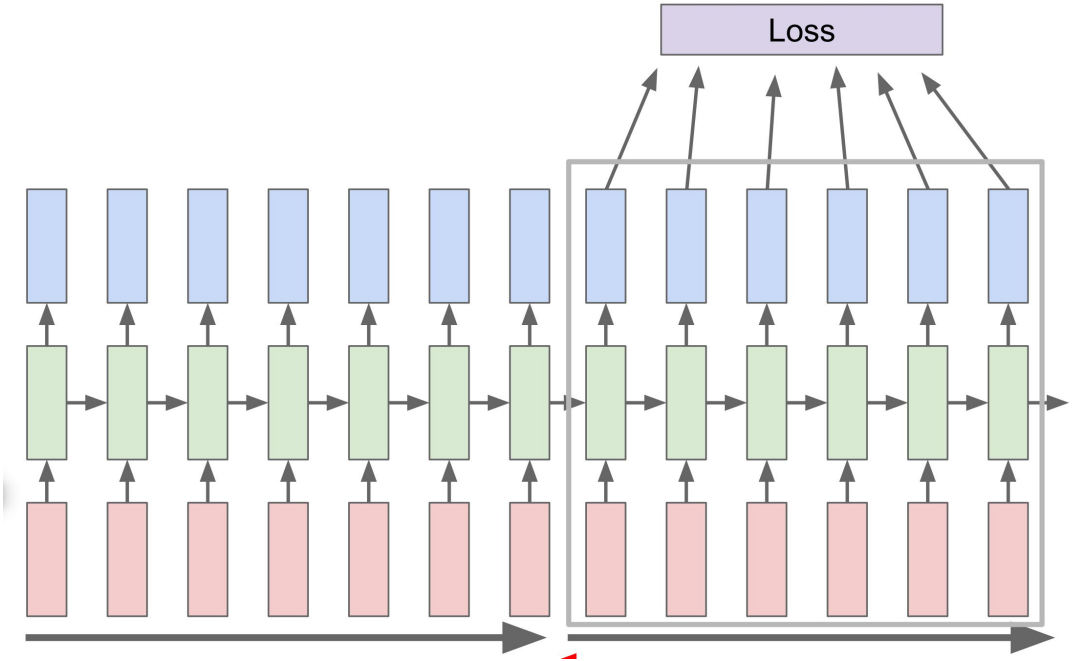
- update step is applied after each sequence of k forward-pass
- each update accumulates gradients over a limited, fixed number of timesteps (k)

Truncated backpropagation through time (TBPTT)



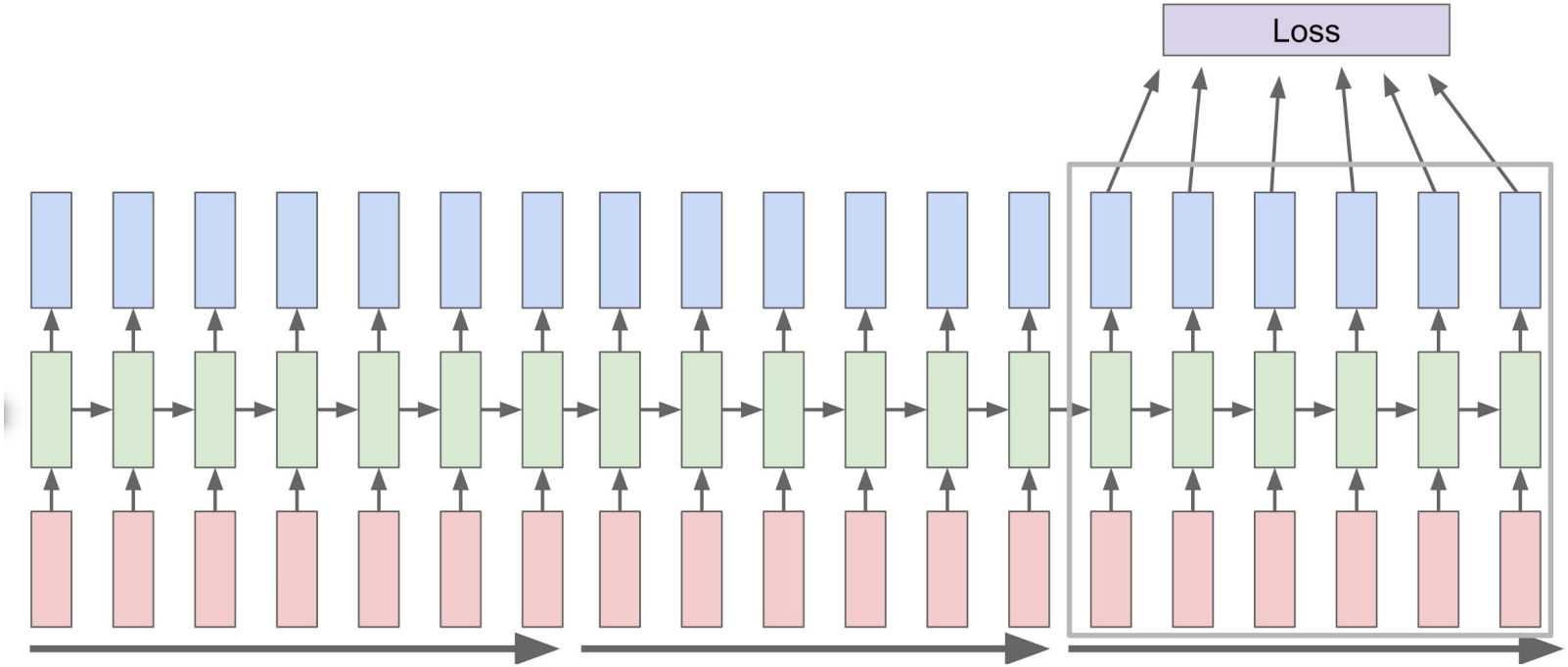
[slide from Stanford CS231 course 2018, Li Fei-Fei]

Truncated backpropagation through time (TBPTT)



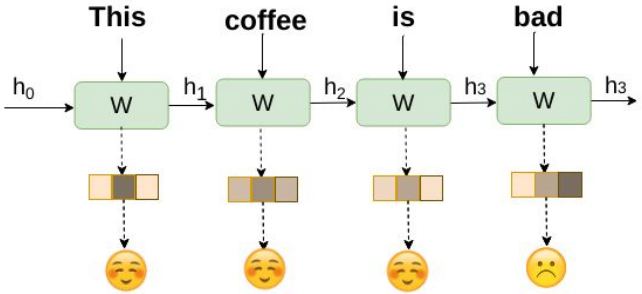
[slide from Stanford CS231 course 2018, Li Fei-Fei]

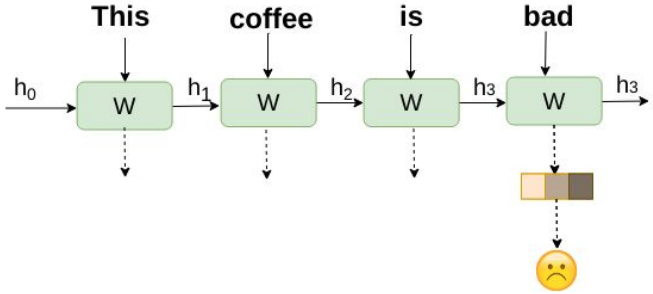
Truncated backpropagation through time (TBPTT)



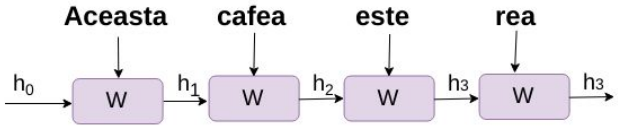
[slide from Stanford CS231 course 2018, Li Fei-Fei]

Classification vs Generation



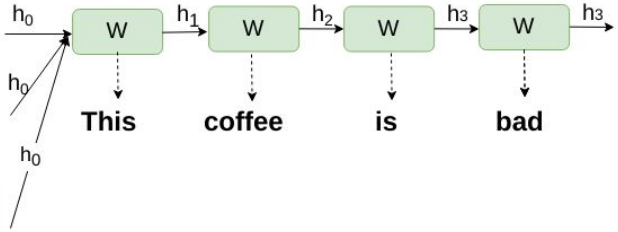


CLASSIFICATION



OR

NONE

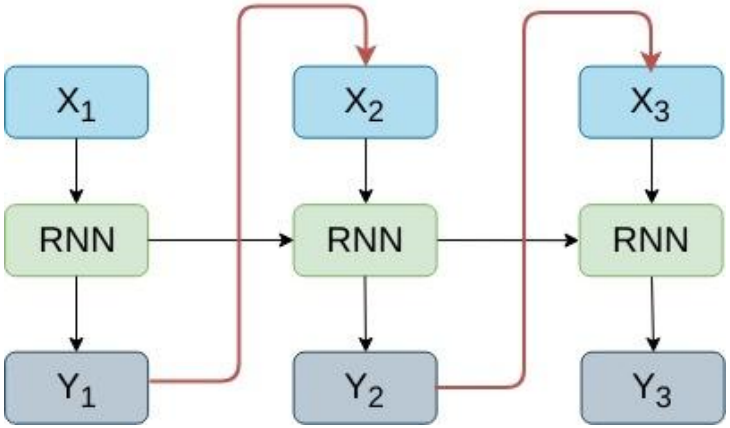


GENERATION

OR

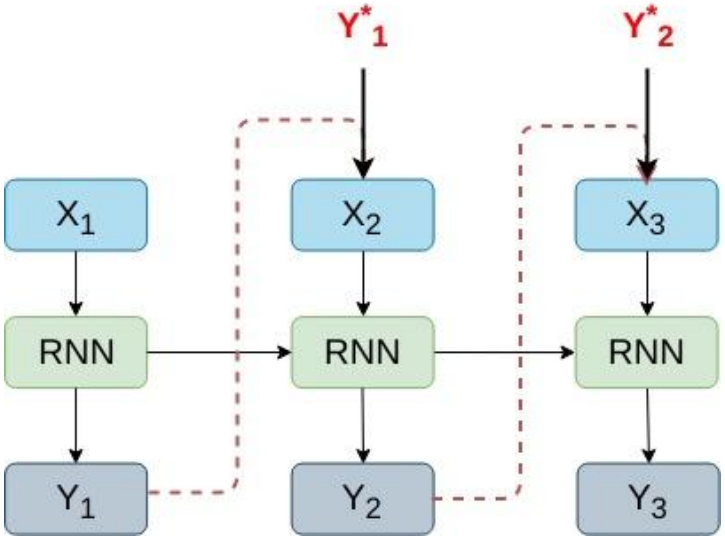
[Image from https://www.buzzfeed.com/stephenlaconte/coffee-is-bad-people-who-drink-it-are-bad]

Generate Sequences - Teacher forcing



- Language models usually use the output from step t-1 as input for step t
- Early mistakes propagates in later steps:
  - Slow convergence
  - Instability of model

Generate Sequences - Teacher forcing

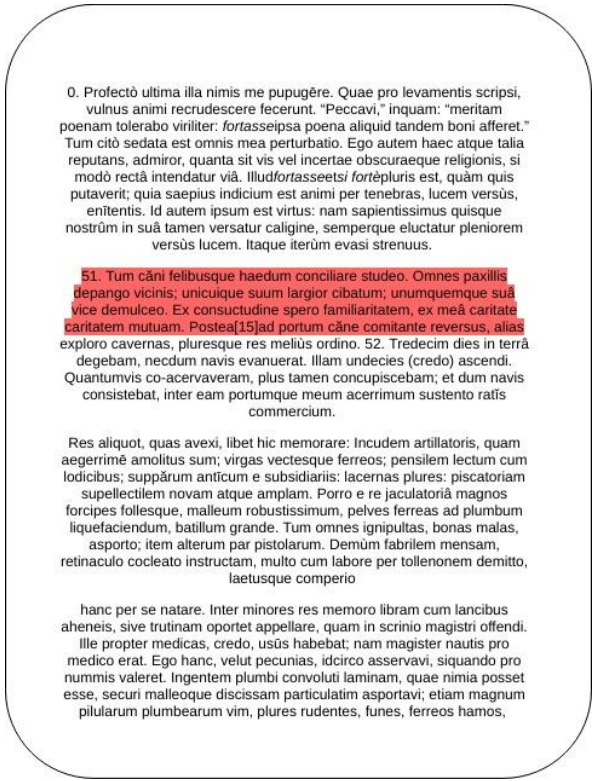


Teacher forcing:

- during train phase the model receive ground truth y* instead of model output y
  - later steps receive correct input even in the beginning of training

- during test phase use the model's output, because we don't have access to the ground truth

Process longer input



0. Profectò ultima illa nimis me pupugêre. Quae pro levamentis scripsi, vulnus animi recrudescere fecerunt. “Peccavi,” inquam: “meritam poenam tolerabo viriliter: *fortasse*ipsa poena aliquid tandem boni afferet.” Tum citò sedata est omnis mea perturbatio. Ego autem haec atque talia reputans, admiror, quanta sit vis vel incertae obscuraeque religionis, si modò rectâ intendatur viâ. Illud*fortasseetsi fortè*pluris est, quàm quis putaverit; quia saepius indicium est animi per tenebras, lucem versùs, enītentis. Id autem ipsum est virtus: nam sapientissimus quisque nostrûm in suâ tamen versatur caligine, semperque eluctatur pleniorem versùs lucem. Itaque iterùm evasi strenuus.

51. Tum cāni felibusque haedum conciliare studeo. Omnes paxillis depango vicinis; unicuique suum largior cibatum; unumquemque suâ vice demulceo. Ex consuetudine spero familiaritatem, ex meâ caritate caritatem mutuam. Postea[15]ad portum cāne comitante reversus, alias exploro cavernas, pluresque res meliùs ordino. 52. Tredecim dies in terrâ degebam, necdum navis evanuerat. Illam undecies (credo) ascendi. Quantumvis co-acervaveram, plus tamen concupiscebam; et dum navis consistebat, inter eam portumque meum acerrimum sustento ratīs commercium.

Res aliquot, quas avexi, libet hic memorare: Incudem artillatoris, quam aegerrimē amolitus sum; virgas vectesque ferreos; pensilem lectum cum lodicibus; suppārum antīcum e subsidiariis: lacernas plures: piscatoriam supellectilem novam atque amplam. Porro e re jaculatorià magnos forcipes follesque, malleum robustissimum, pelves ferreas ad plumbum liquefaciendum, batillum grande. Tum omnes ignipultas, bonas malas, asporto; item alterum par pistolarum. Demùm fabrilem mensam, retinaculo cocleato instructam, multo cum labore per tollenonem demitto, laetusque comperio

hanc per se natare. Inter minores res memoro libram cum lancibus aheneis, sive trutinam oportet appellare, quam in scrinio magistri offendi. Ille propter medicas, credo, usūs habebat; nam magister nautis pro medico erat. Ego hanc, velut pecunias, idcirco asservavi, siquando pro nummis valeret. Ingentem plumbi convoluti laminam, quae nimia posset esse, securi malleoque discissam particulatim asportavi; etiam magnum pilularum plumbearum vim, plures rudentes, funes, ferreos hamos,

- **Problem:**
  - can’t fit the entire document in the model
  - need batch_size > 1
  - need previous context for each sub-sequence

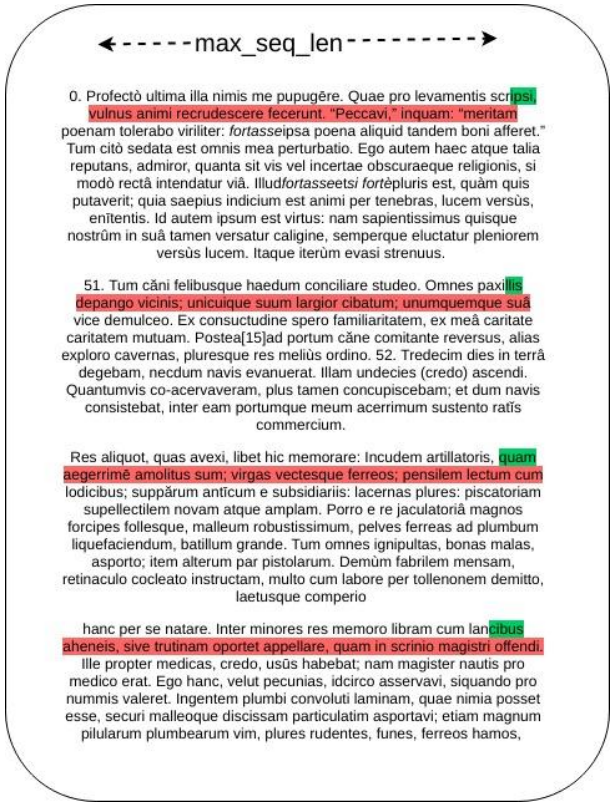
Fragment from: Rebilius Cruso (Francis William Newman) http://www.gutenberg.org/files/50732/50732-h/50732-h.html

Process longer input

- **Problem:**
  - can't fit the entire document in the model
  - need batch_size > 1
  - need previous context for each sub-sequence

- **Solution:**
  - split document in batch_size continuous chunks
  - one batch receive, in each training iteration, sequences from different chunks

Process longer input



- **Problem:**
  - can't fit the entire document in the model
  - need batch_size > 1
  - need previous context for each sub-sequence

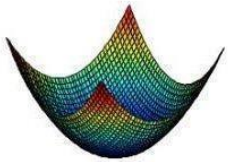
- **Solution:**
  - split document in batch_size continuous chunks
  - one batch receive, in each training iteration, sequences from different chunks

Let's take a little

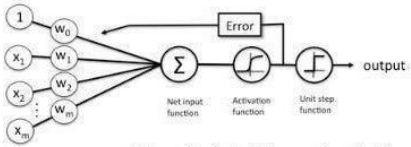
coffee break

Deep Neural Network

You

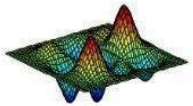


- Unique optimum: global/local.



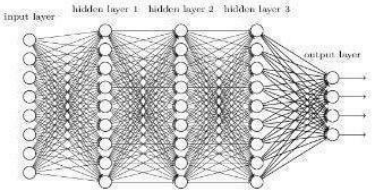
Schematic of a logistic regression classifier.

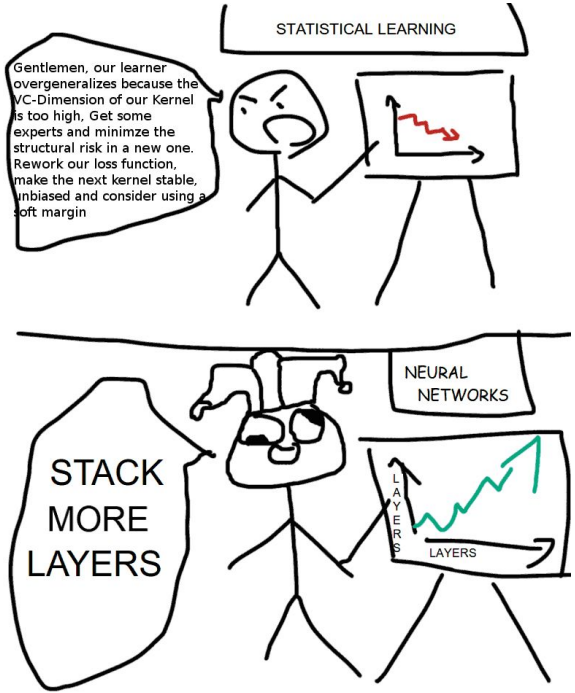
The guy she tells you not to worry about



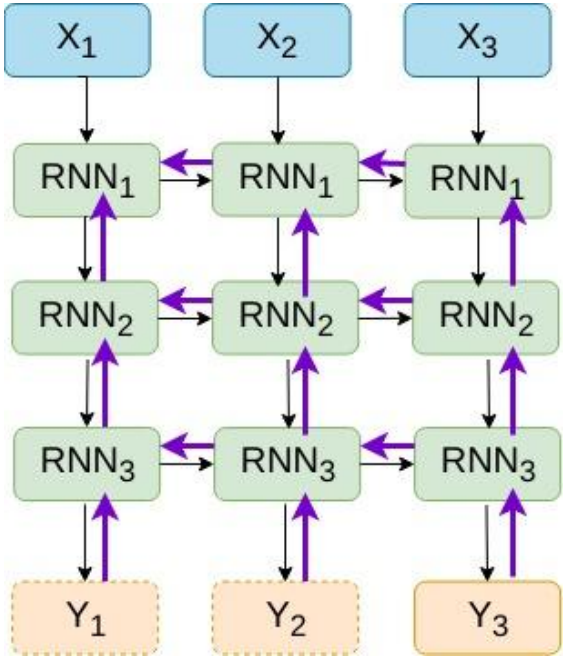
- Multiple local optima
- In high dimensions possibly

Deep neural network



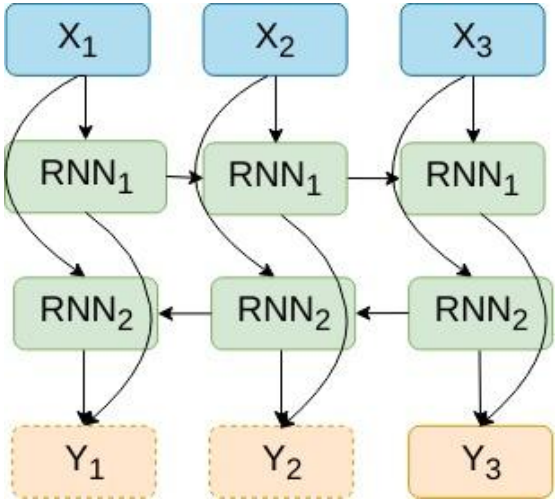


Multilayer RNN



- better capture the structure of the input sequence
- might be harder to optimize (longer paths)

Bidirectional RNN



$$\bar{h}_t = \sigma(\bar{W}_{hh}\bar{h}_{t-1} + \bar{W}_{xh}x_t)$$
$$\tilde{h}_t = \sigma(\tilde{W}_{hh}\tilde{h}_{t-1} + \tilde{W}_{xh}x_t)$$
$$[y_t = W_{hy}[\bar{h}_t; \tilde{h}_t]]$$

- prediction at timestep t depend on the whole input
- combine context from 2 RNNs: a **forward** RNN and a **backward** RNN

Schuster, Mike and Kuldip K. Paliwal. “Bidirectional recurrent neural networks.” *IEEE Trans. Signal Processing* 45 (1997): 2673-2681.

RECAP

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

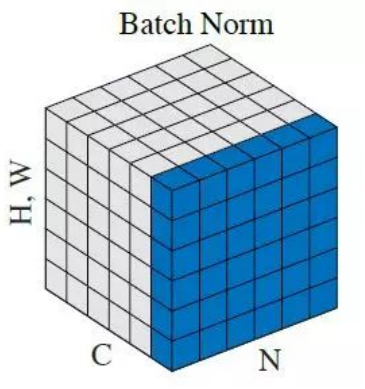
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

$x.\,shape = N \times H \times W \times C$
$\sigma_\beta.\,shape = C$



ONLY AT TRAINING TIME !!!

Normalization for RNN

- **Classic BatchNorm is not recommended**
  - Each timestep has different statistics (mean and variance)
    - statistics computed and used overall are not accurate

- Modified BatchNorm used sometimes (Cojimas et al. ICLR'17)
  - Keep different statistics for each timestep:

    $\sigma_{t=1}, \sigma_{t=2}, \sigma_{t=3} \cdot \cdot$

    $\mu_{t=1}, \mu_{t=2}, \mu_{t=3} \cdot \cdot$

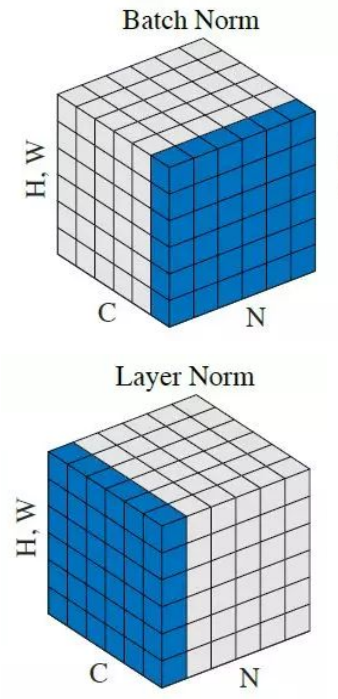
  - Converge when the number of timestep is large

Normalization for RNN: LayerNorm

- **LayerNorm** (Lei Ba et al. 2016)
  - Normalize within each example, across channels

$$\mu = \frac{1}{D}\sum_{d=1}^{D} x_i^d$$
$$\sigma^2 = \sum_{d=1}^{D}(x_i^d - \mu)^2$$

$$x.shape = N \times H \times W \times C$$
$$\sigma_\beta.shape = N$$



[Cooijmans et al., “Recurrent Batch Normalization.” ICLR 2017]
[Lei Ba et al., Layer Normalization, 2016]

Pros and Cons for RNNs

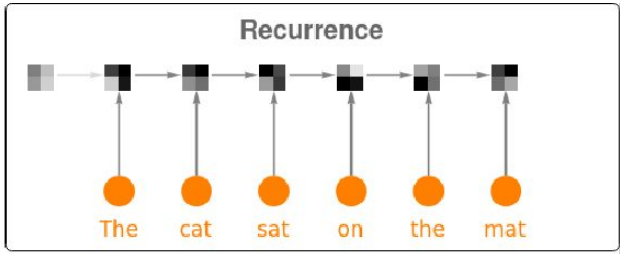
Pros:

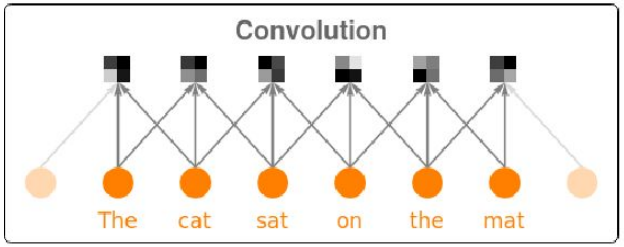
- input of variable length
- each step can depend on context from many steps back
- symmetric, intuitive processing

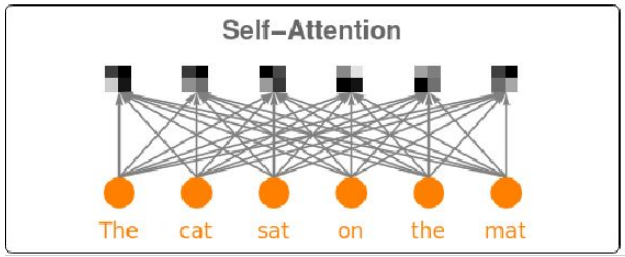
Cons

- slow in computation
- information from many steps back is, in practice, lost

Alternative approaches for Sequential Data



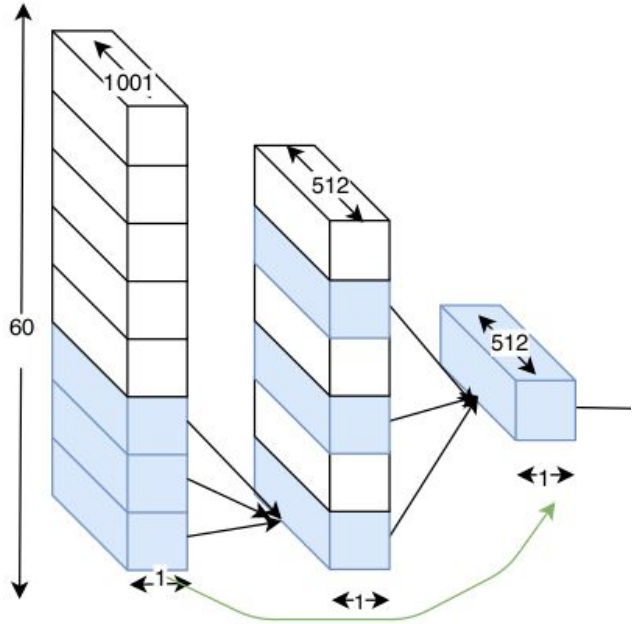




- Recurrent Neural Network
- Temporal Convolutional Network
- Self-Attention Network

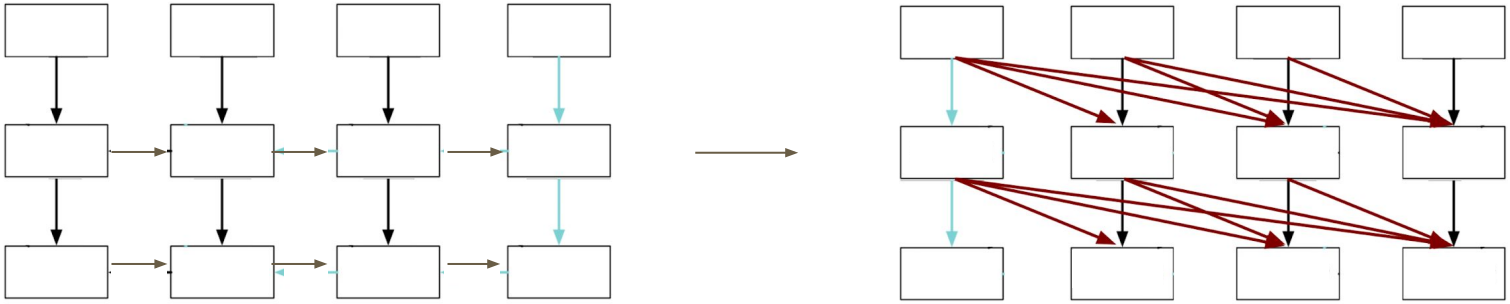
RECAP

Temporal Convolutional Network



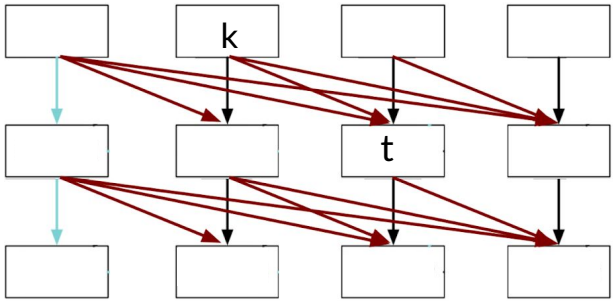
- 1D convolutional network to aggregate information from a sliding window
- use dilated convolutional layers to capture large context in few layers

Self-Attention



- each state depends on **all** of the others, not only the previous one
- improved parallelism
- based on an **attention mechanism**. Different than fully connected network.

Self-Attention



- Each hidden state is computed as a **weighted sum** of the information from the **entire sequence**

$$h_t = \sum_{k=1}^{T} \alpha_{tk} v_k$$

- The weight is computed based on a **dot-product similarity** with respect to the current state t

α_{tk} – similarity between timestep t and k

Until now:

- in RNN $h_t = f(h_{t-1}, v_t)$
- in FC $h_t = \sum_{k=1}^{T} \alpha_{tk} v_k$

α_{tk} network's parameter, not computed (dynamic) based on input

Self-Attention

Each timestep is represented by 3 vectors:

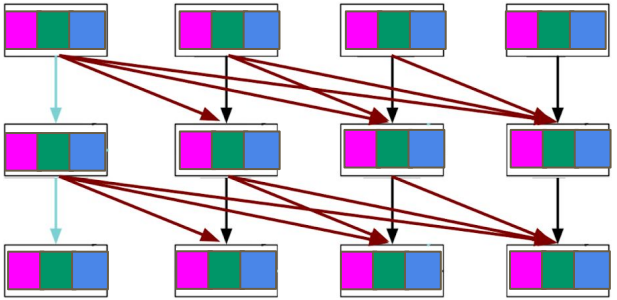
- query (Q)
- key (K)

} used to compute similarity

- value (V) — value to aggregate

$$h_t = \sum_{i=1}^{T} \alpha_{ti} v_i$$

$$\alpha_{ti} = q_t k_i$$



Self-Attention

Each timestep is represented by 3 vectors:

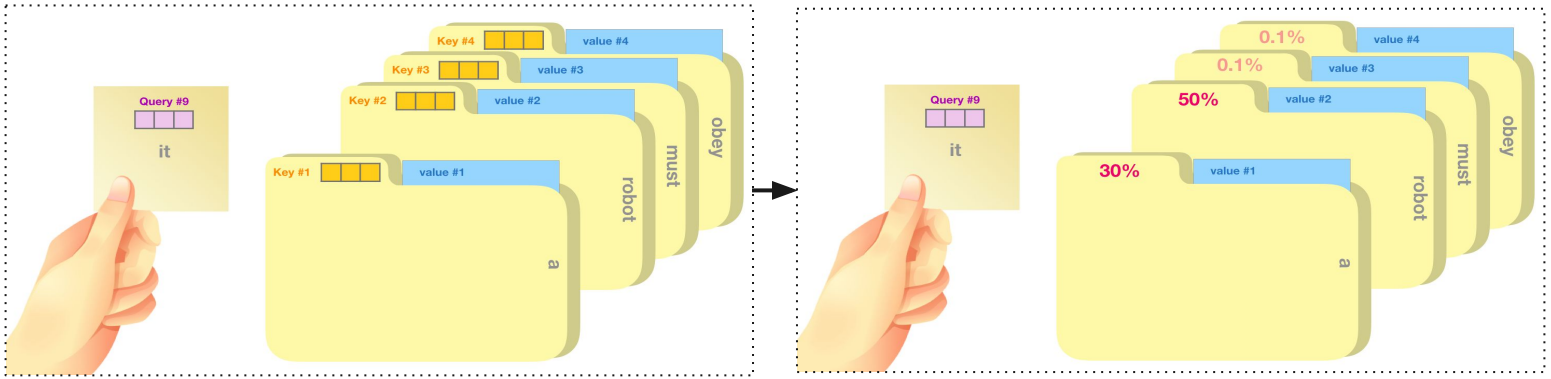
- query (Q)
- key (K)

} used to compute similarity

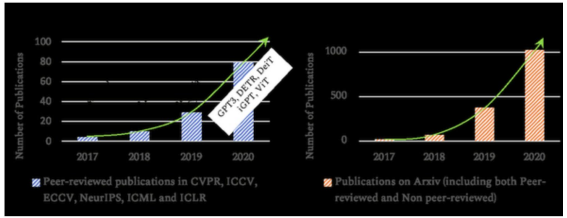
- value (V) — value to aggregate

$$h_t = \sum_{i=1}^{T} \alpha_{ti} v_i$$

$$\alpha_{ti} = q_t k_i$$



figures from http://jalammar.github.io/illustrated-gpt2/



The New York Times

Meet GPT-3. It Has Learned to Code (and Blog and Argue).

The latest natural-language system generates tweets, pens poetry, summarizes emails, answers trivia questions, translates languages and even writes its own computer programs.

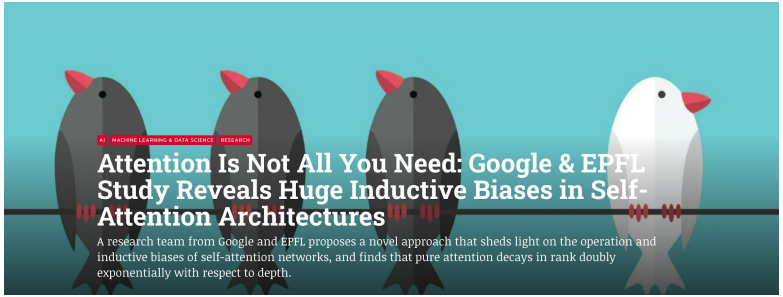
Money Is All You Need

Nick Debu
Tokyo Institute of Bamboo Steamer

Abstract

Transformer-based models routinely achieve state-of-the-art results on a number of tasks but training these models can be prohibitively costly, especially on long sequences. We introduce one technique to improve the performance of Transformers We replace NVIDIA P100s by TPUs, changing its memory from hoge GB to piyo GB. The resulting model performs on par with Transformer-based models while being much more ""TSUYO TSUYO"".

Attention is all you need!



Attention is all you need. Vaswani et al. 2017

Ongoing research

Self-attention / Transformer:

- requires lots of data
- requires lots of resources
- for long sequences, the number of pairwise computations increases dramatically

Published as a conference paper at ICLR 2020

REFORMER: THE EFFICIENT TRANSFORMER

Nikita Kitaev* U.C. Berkeley & Google Research kitaev@cs.berkeley.edu

Łukasz Kaiser* Google Research {lukaszkaiser,levskaya}@google.com

Anselm Levskaya Google Research

18 Feb 2020

Linformer: Self-Attention with Linear Complexity

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, Hao Ma
Facebook AI, Seattle, WA
{sinongwang, belindali, hanfang, mkhabsa, haom}@fb.com

14 Jun 2020

Published as a conference paper at ICLR 2021

RETHINKING ATTENTION WITH PERFORMERS

Krzysztof Choromanski*[1], Valerii Likhosherstov*[2], David Dohan*[1], Xingyou Song*[1]
Andreea Gane*[1], Tamas Sarlos*[1], Peter Hawkins*[1], Jared Davis*[3], Afroz Mohiuddin[1]
Lukasz Kaiser[1], David Belanger[1], Lucy Colwell[1,2], Adrian Weller[2,4]
[1]Google [2]University of Cambridge [3]DeepMind [4]Alan Turing Institute

ABSTRACT

We introduce *Performers*, Transformer architectures which can estimate regular (softmax) full-rank-attention Transformers with provable accuracy, but using only linear (as opposed to quadratic) space and time complexity, without relying on any priors such as sparsity or low-rankness. To approximate softmax attention-kernels, Performers use a novel *Fast Attention Via positive Orthogonal Random features* approach (FAVOR+), which may be of independent interest for scalable kernel methods. FAVOR+ can also be used to efficiently model kernelizable attention mechanisms beyond softmax. This representational power is crucial to accurately compare softmax with other kernels for the first time on large-scale tasks, beyond the reach of regular Transformers, and investigate optimal attention-kernels. Performers are linear architectures fully compatible with regular Transformers and with strong theoretical guarantees: unbiased or nearly-unbiased estimation of the attention matrix, uniform convergence and low estimation variance. We tested Performers on a rich set of tasks stretching from pixel-prediction through text models to protein sequence modeling. We demonstrate competitive results with other examined efficient sparse and dense attention methods, showcasing effectiveness of the novel attention-learning paradigm leveraged by Performers.

9 Mar 2021

Kitaev et al., Reformer: The Efficient Transformer
Wang et al., Linformer: Self-Attention with Linear Complexity
Choromanski et al., Rethinking Attention with Performers

TODAY

- RNNs are good for **sequential data**
- RNNs allow **flexibility** in architecture
- BPTT suffer from **vanishing** / **exploding** gradients
- **clipping gradients** and **LSTM/GRU** architectures are common ways to avoid them
- there are different alternatives: TCN, Self-Attention

Thank you!

(Next: Applications of RNN in NLP)

Other Resources

http://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/ - Lectures 15-16
https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/ - Lecture 12-13
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf - Lecture 10
http://cs224d.stanford.edu/syllabus.html Lecture 6-8

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

GRU: https://arxiv.org/abs/1406.1078
LSTM: https://www.bioinf.jku.at/publications/older/2604.pdf