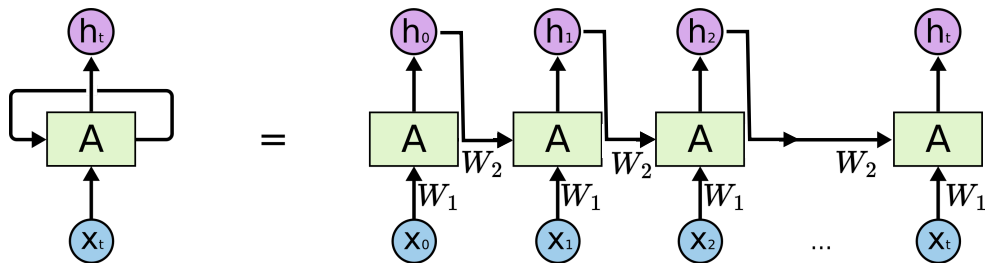


RÉSEAUX DE NEURONES RÉCURRENTS

Vincent Guigue,
inspiré des supports de Nicolas Baskiotis & Benjamin Piwowski
Illustrations de C. Olah [comme tout le monde]

Recurrent Neural Network



General RNN:

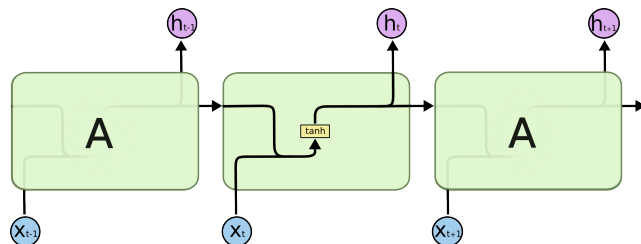
$$\mathbf{h}_t = \tanh(\mathbf{x}_t W_1 + \mathbf{h}_{t-1} W_2 + b) \text{ or } \tanh((\mathbf{x}_t \oplus \mathbf{h}_{t-1})W + b)$$

- **Few parameters**
- Ability to deal with **variable lengths**
- Capture the **dynamics** of the sequence



Cell details

Classical RNN :



■ Latent state $h_t \in \mathbb{R}^d$

■ input $x_t \in \mathbb{R}^n$

■ $h_t = f_W(h_{t-1}, x_t) = \tanh((x_t \oplus h_{t-1})W + b)$

■ $W \in \mathbb{R}^{n \times d}, b \in \mathbb{R}$

■ Initial state: h_1

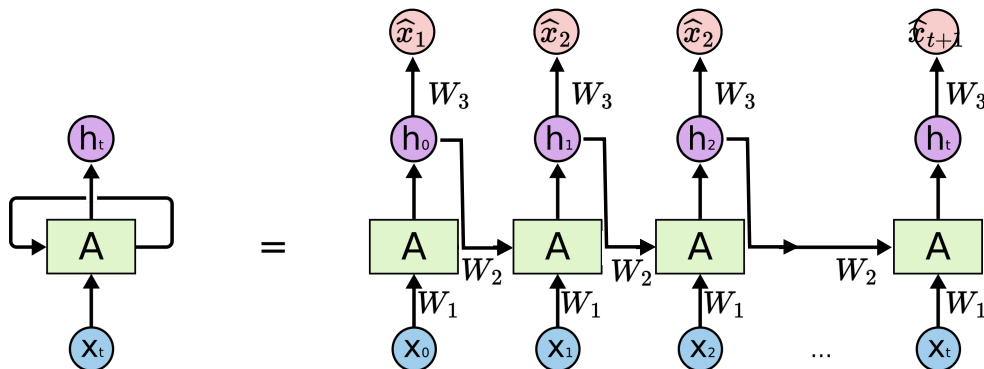
■ Sequence $[x_1, \dots, x_T] \Rightarrow [h_1, \dots, h_T]$

■ May be computed left \rightarrow right (h_t) ... Or right \rightarrow left (h_t^R)

RNN... For which purpose?

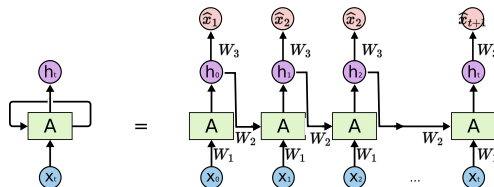
- Predicting the next step (from h_T)
- Generation = prediction (in loop)
- Classifying a sequence (from h_T)
- Detecting an event (at every h_t)

Bi-RNN: use h_t & h_t^R
(h_T & h_1^R = caract. whole sequence)





Learn to generate



■ Teacher forcing:

$$\hat{x}_{t+1} = g(h_t), \quad h_{t+1} = f_W(h_t, x_{t+1})$$

■ Free Generation:

$$\hat{x}_{t+1} = g(h_t), \quad h_{t+1} = f_W(h_t, \hat{x}_{t+1})$$

In practice: curriculum learning

- Start with teacher forcing (more stable) \Rightarrow then switch to free generation
- Sequential learning also = more complicated solution

\Rightarrow **Reinforcement Learning**



RNN In practice

```
1 # Define the model
2 model = torch.nn.RNN(input_size=100, hidden_size=50,
3                       num_layers=1,
4                       nonlinearity='tanh', bias=True)
5
6 # data (x_1, ..., x_T)
7 # Seq. length T=23 & batch of size 10
8 # Input x_t \in R^n dimension 100
9 seq_x = torch.rand(23, 10, 100)
10
11 # seq_h is a tensor (23, 10, 50)
12 # For classification purpose, we use last_h
13 seq_h, last_h = model(seq_x)
```

Convention

Tensor of size **time x sample/batch x (in) representation**

SPECIFICITIES & DIFFICULTIES



Problems with RNN

Simple use case, linear + no entries

- Error function: E
- $h_{t+1} = W h_t$
- t step \Rightarrow computing W^t (power of t)
- Given the decomposition: $W = U\Sigma U^T$, Σ = diag matrix of eigen values

Gradient computation leads to:

$$\frac{\partial E}{\partial h_t} = \frac{\partial E}{\partial h_N} (W^t)^{N-t} = \Delta U \begin{pmatrix} \sigma_1^{N-t} & & \\ & \ddots & \\ & & \sigma_n^{N-t} \end{pmatrix}$$

- if $\sigma_i > 1$ exponential growth, exploding gradient
- if $\sigma_i < 1$ gradient vanishing



Exploding gradient

- Gradient values > 1 at every iteration
- Weak performances
- *NaN* everywhere quickly !

⇒ **Gradient clipping**

$$\nabla_W E = \begin{cases} \frac{\nabla_W E}{\|\nabla_W E\|} & \text{if: } \|\nabla_W E\| > 1 \\ \nabla_W E & \text{else} \end{cases}$$

```
1 # use before 'optimizer.step()'
2 nn.utils.clip_grad_norm_(
3     model.parameters(),
4     max_norm=1.,
5     norm_type=2
6 )
```

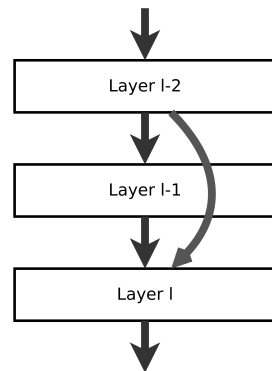


Vanishing gradient \Rightarrow Residual architecture (e.g. ResNet)

$$f(x) = x + f_0(x)$$
$$\frac{\partial f(x)}{\partial x_j} f(x) = \mathbf{1} + \frac{\partial f_0(x)}{\partial x_j}$$

In the back propagation process:

$$\frac{\partial E_f(x)}{\partial x_j} f(x) = \frac{\partial E_y}{\partial y_j} + \sum_i \frac{\partial E_y}{\partial y_i} \frac{\partial y_i}{\partial x_j}$$

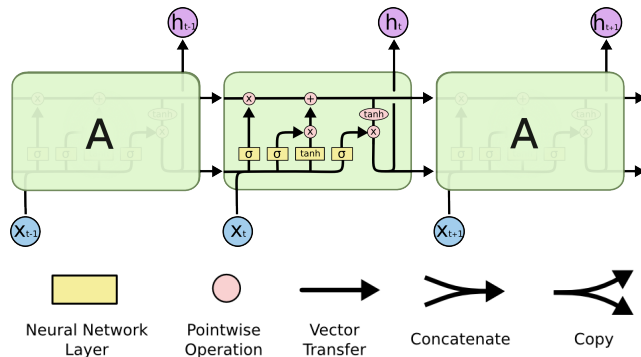




Vanishing gradient \Rightarrow gated unit (LSTM)

The phenomenon has been understood & (partially) overcome:

Neurons **learn** what should be **kept in memory** and what should be **forgotten**



Gated architecture



S. Hochreiter, J. Schmidhuber, Neural computation 1997
Long short-term memory

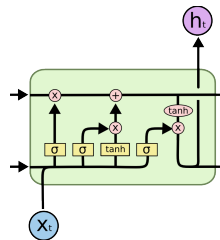
Chris Olah's blog <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Vanishing gradient \Rightarrow gated unit (LSTM)

The phenomenon has been understood & (partially) overcome:

Neurons **learn** what should be **kept in memory** and what should be **forgotten**

- Memory c_t (modified by simple operators)
- External representation h_t extracted from c_t
- Gates $y = \sigma(p) \otimes x$,
 \oplus : concat. & \otimes : term by term product & σ : sigmoid

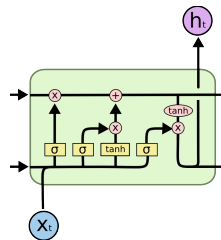


Vanishing gradient \Rightarrow gated unit (LSTM)

The phenomenon has been understood & (partially) overcome:

Neurons **learn** what should be **kept in memory** and what should be **forgotten**

- Memory c_t (modified by simple operators)
- External representation h_t extracted from c_t
- Gates $y = \sigma(p) \otimes x$,
 \oplus : concat. & \otimes : term by term product & σ : sigmoid



Gate 1 (forgetting):

$$g_t^{(f)} = \sigma[W_f(x_t \oplus h_{t-1}) + b_f]$$

$$c_t^{(f)} = g_t^{(f)} \otimes c_{t-1}$$

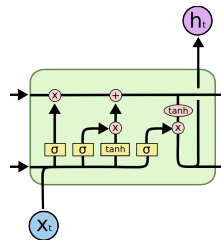


Vanishing gradient \Rightarrow gated unit (LSTM)

The phenomenon has been understood & (partially) overcome:

Neurons **learn** what should be **kept in memory** and what should be **forgotten**

- Memory c_t (modified by simple operators)
- External representation h_t extracted from c_t
- Gates $y = \sigma(p) \otimes x$,
 \oplus : concat. & \otimes : term by term product & σ : sigmoid



Gate 2 (memory update):

$$g_t(i) = \sigma[W_{i,g}(x_t \oplus h_{t-1}) + b_{i,g}]$$

$$v_t(i) = \tanh[W_{i,v}(x_t \oplus h_{t-1}) + b_{i,v}]$$

$$c_t = c_t^{(f)} + g_t^{(i)} \otimes v_t^{(i)}$$

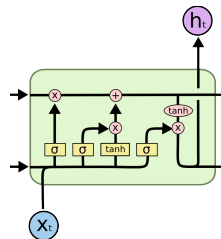


Vanishing gradient \Rightarrow gated unit (LSTM)

The phenomenon has been understood & (partially) overcome:

Neurons **learn** what should be **kept in memory** and what should be **forgotten**

- Memory c_t (modified by simple operators)
- External representation h_t extracted from c_t
- Gates $y = \sigma(p) \otimes x$,
 \oplus : concat. & \otimes : term by term product & σ : sigmoid



Gate 3 (output):

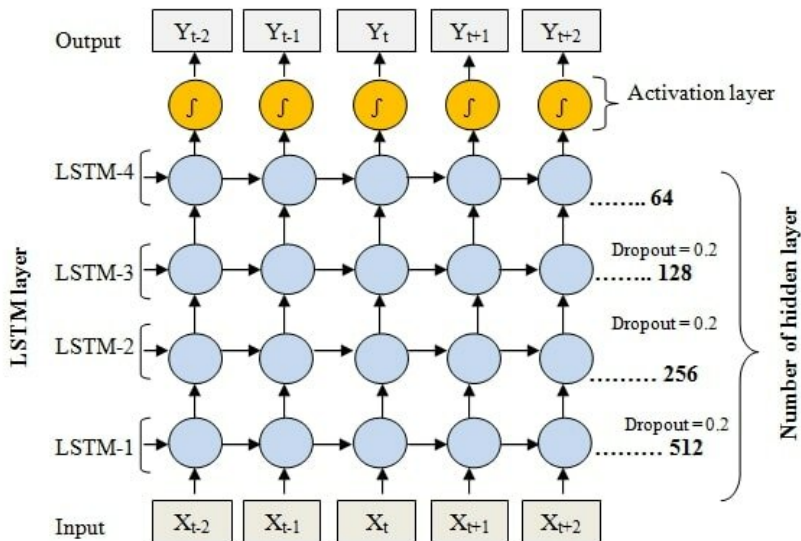
$$g_t(o) = \sigma[W_{o,g}(x_t \oplus h_{t-1}) + b_{o,g}]$$

$$v_t(o) = \sigma[W_{o,v}c_t + b_{o,v}]$$

$$h_t = g_t^{(o)} \otimes v_t^{(o)}$$

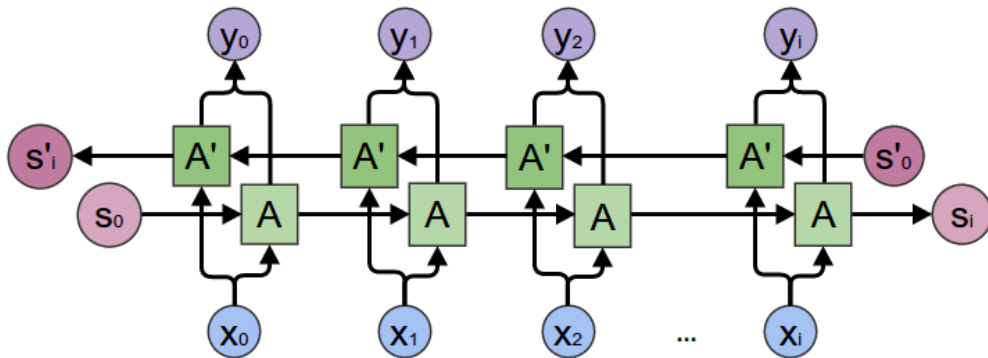
Bi-RNN, multi-layer-RNN, ...

Multilayer RNN : more complex temporal aggregation



Bi-RNN, multi-layer-RNN, ...

BiRNN : reading the sequence both directions





Conclusion

- **LSTM** and variation (**GRU**) are widely used
- biRNN (biLSTM / BiGRU) offer often faster convergence
- Gradient is always an issue: **watch it!**

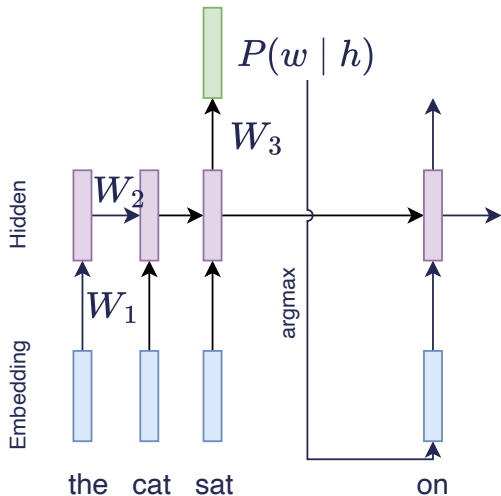
```
1 # Calcul de || \nabla_W E ||^2
2 with torch.no_grad():
3     grad_sql2 = sum((p.grad ** 2).sum() for p in l.parameters() )
```

- **Dropout** is useful to fight against overfitting
- Layers can be stacked (!) \Rightarrow num_layers

```
1 # Module definition
2 model = torch.nn.LSTM(input_size=100, hidden_size=50,
3                        batch_first=False, num_layers=1,
4                        dropout=.3)
```

DATA GENERATION

Basic data generation

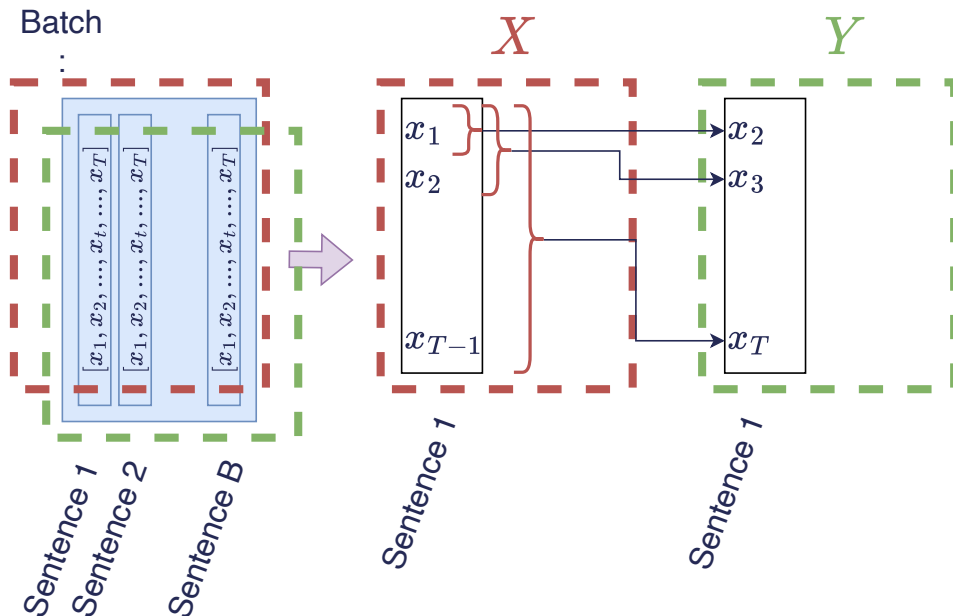


Word prediction

- $W_3 \approx$ vocabulary classification
 - Several variation: argmax / sampling...
- ⇒ reduce vocabulary
- at the letter scale (dim ≈ 60)
 - byte pair encoding \Rightarrow most freq. ngrams of letters
- Iteration at $t + 1$ only requires:
 - x_{t+1}
 - h_t (that encode the whole sequence of previous words)



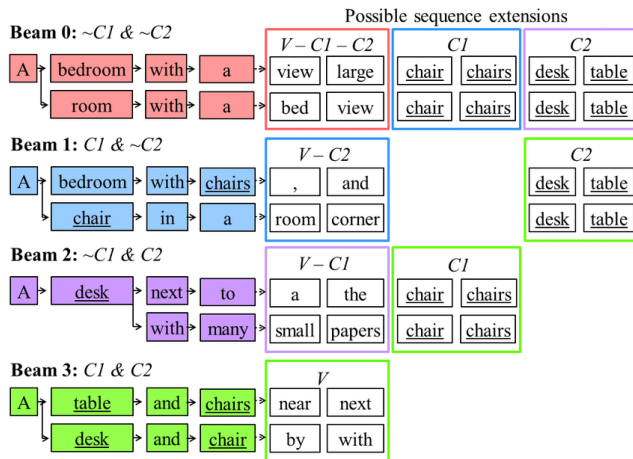
RNN In practice: teacher forcing & data shape



Beam search

Very difficult to find the best word at time $t...$

⇒ explore several possibilities, keep the most likely !



- Sampling function required (cf practical session)

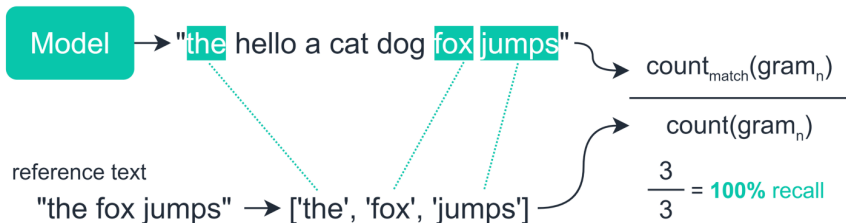


Evaluation: BLEU/ROUGE (translation & summary)

How to evaluate text generation?

Very difficult...

still an open issue !



- f1 computations
- on N-grams, on longest common subsequence...

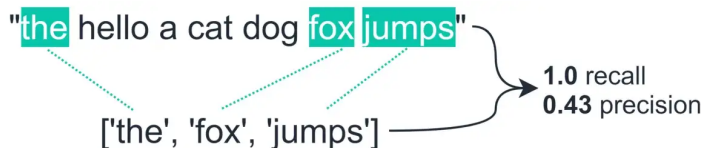


Evaluation: BLEU/ROUGE (translation & summary)

How to evaluate text generation?

Very difficult...

still an open issue !



$$2 * \frac{0.43 * 1.0}{0.43 + 1.0} = 0.6$$

60% f1 score

- f1 computations
- on N-grams, on longest common subsequence...



Conclusion

- Evaluation in the latent-space (cf BERT-score / CLS)
- Content evaluation (entity, figures...) \Rightarrow hallucination problems

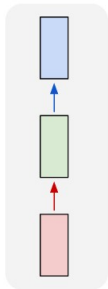
\Rightarrow Generative AI is an opportunity in various fields... But **benchmarks are not adapted to this tool** (yet)

SEQUENCE TO SEQUENCE & END-TO-END ARCHITEC- TURES

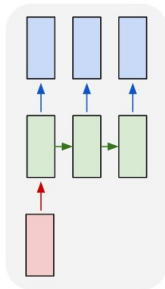


RNN architecture : different settings

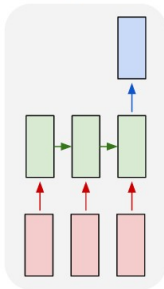
one to one



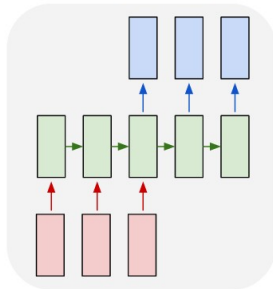
one to many



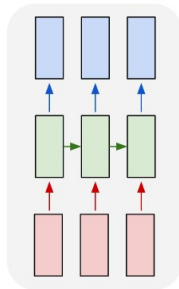
many to one



many to many



many to many

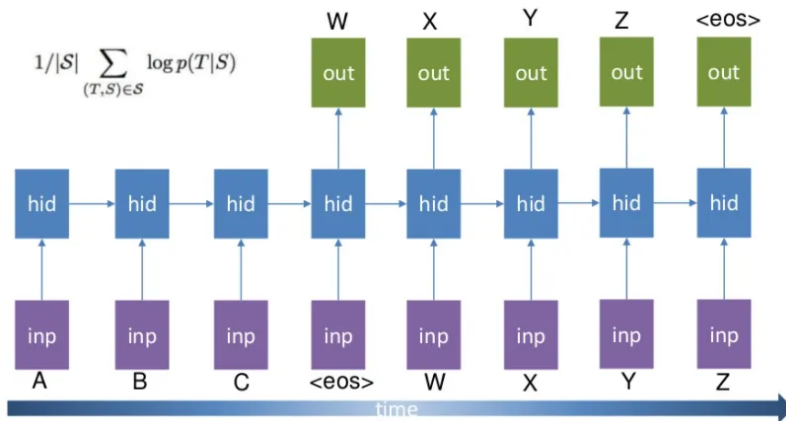


- One to many : image annotation
- many to one : signal classification
- many to many : POS/NER tagging, sequence annotation
- seq to seq : machine translation

Karpathy's blog <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>



Encoder-decoder architecture



- Exploiting seq-2-seq paradigm in time series
- Encoding the time-series (& being able to decode it)



Sutskever et al., 2014,
Sequence to Sequence Learning with Neural Networks



How to train a Machine Translator?

- Predicting next word(s) in both language
 - What about the vocabulary?
 - One or two Neural Networks?
- From word prediction to translation
 - How to learn the translator?

Curriculum Learning & end-to-end architectures are complementary

TIME SERIES & MACHINE- LEARNING



Machine-Learning vs Deep learning for time series

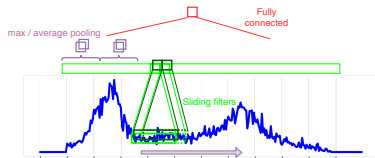
- Which applications?
- Which signal description? Which context? Which target?
- Classical approaches:
 - 1 Historical stats: AR, ARMA, SARIMA
 - 2 Machine-Learning: SVM, Ridge, Random Forest, XGB
 - Which descriptors? cf pandas
 - Stats (min, max, mean, std, ...)
 - Freq. (Fourier, PSD, ...)
 - Context (Day, Weather, Location)
 - 3 Dynamic Time Wrapping (DTW) :
 - efficient computation of distance between signals
- When to use deep learning?



Neural architectures & (multivariate) time series

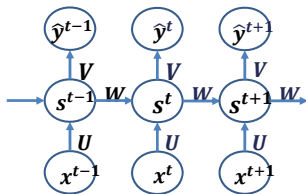
■ CNN

- Different filters for each channel / same filters



■ RNN :

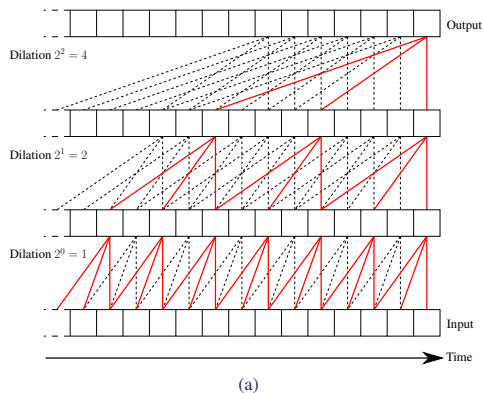
- no problem to give a vector as input at each time step



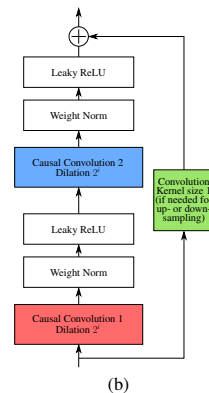


Unsupervised learning framework

Step 1: Encoding



+ max pooling



Franceschi et al., NeurIPS 2019,

Unsupervised Scalable Representation Learning for Multivariate Time Series

Unsupervised learning framework

Step 2: Unsupervised learning

- Given subseq x^{pos} and local contexts x^{ref} :

Idea Skip Gram: $\arg \max_{\theta} \prod_{x^{ref}} \prod_{x^{pos} \in x^{ref}} p(x^{ref} | x^{pos}; \theta)$

- $p(D = 1 | x^{ref}, x^{pos}; \theta) \Rightarrow$ proba. that x^{pos} occur in the context x^{ref}
- Triplet loss

$$\arg \max_{\theta} \prod_{ref, pos} p(D = 1 | x^{ref}, x^{pos}; \theta) + \underbrace{\prod_{ref, neg} p(D = 0 | x^{ref}, x^{neg}; \theta)}_{\text{Negative Sampling}}$$

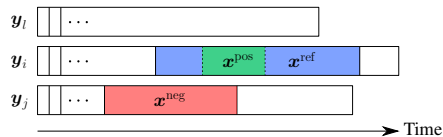


Figure 1: Choices of x^{ref} , x^{pos} and x^{neg} .

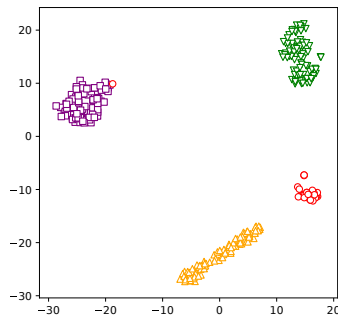


Franceschi et al., NeurIPS 2019,
Unsupervised Scalable Representation Learning for Multivariate Time Series

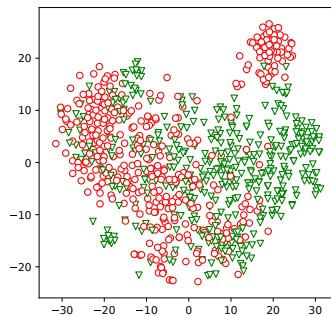


Unsupervised learning framework

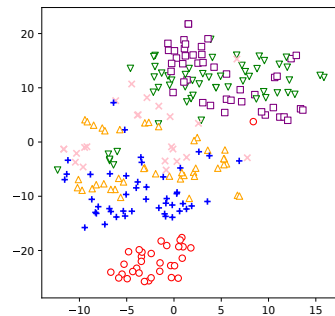
Interesting latent space for time-series:



(a) DiatomSizeReduction.



(b) FordB.



(c) OSULeaf.

- Very good classification results with simple Logistic Regression



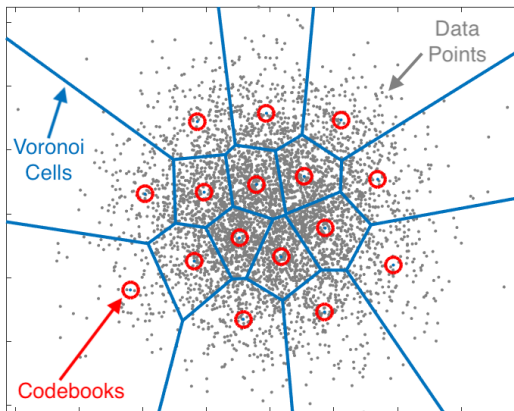
Franceschi et al., NeurIPS 2019,

Unsupervised Scalable Representation Learning for Multivariate Time Series

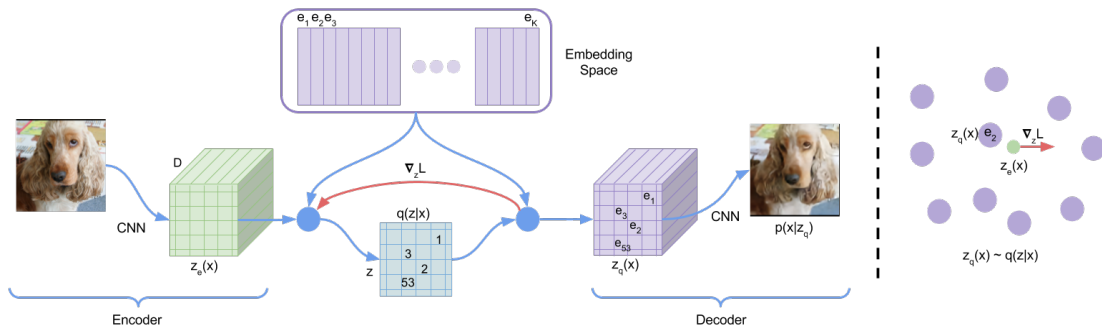


Vector Quantization (VQ)

- Machine Learning = mainly suitable for continuous values
- Quantization = continuous \Rightarrow discrete values
- Assumption: categorical values are more interpretable
 - Pattern identification



VQ-Variational Auto-Encoder



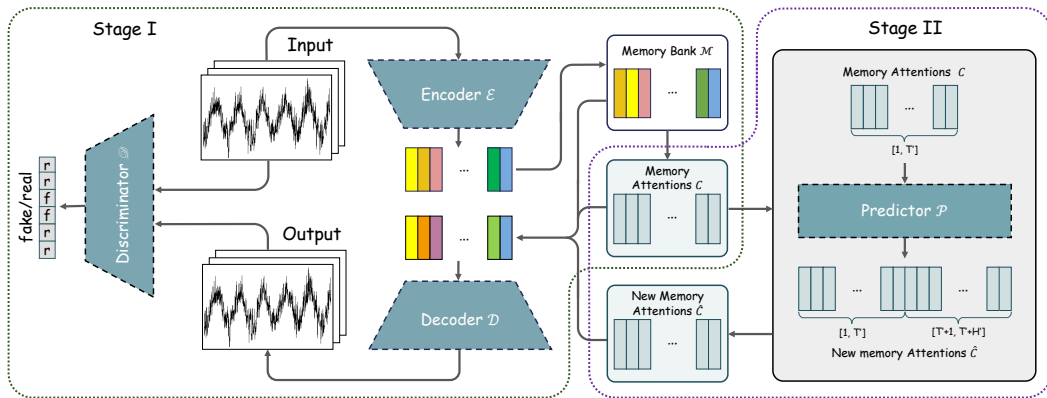
- Interpretability = discrete representation \Rightarrow object decomposition
- ... & limited codebook



A. van den Oord et al., *NeurIPS 2017*,
Neural Discrete Representation Learning



VQ-VAE Implementation for time-series



- Discrete decomposition, signal reconstruction
- + Aversarial discriminator (\Rightarrow noise reconstruction ?)





Conclusion

Numerous applications in time series

- Modeling/discriminating complex shape
- Prediction Ability
- Context modeling
 - e.g. public transportation

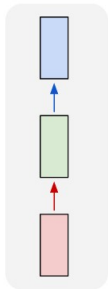
- How to choose among deep architectures?
 - RNN, CNN, ...

AGGREGATION & ATTENTION

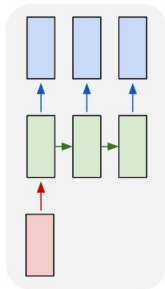


RNN architecture : different settings

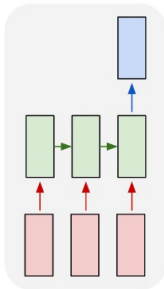
one to one



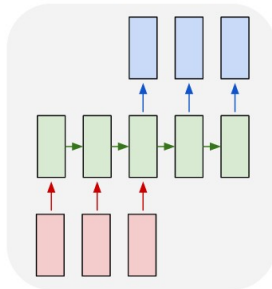
one to many



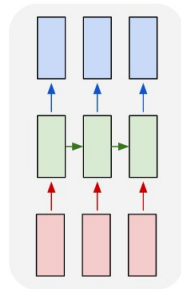
many to one



many to many



many to many



- One to many : image annotation
- many to one : signal classification
- many to many : POS/NER tagging, sequence annotation
- seq to seq : machine translation

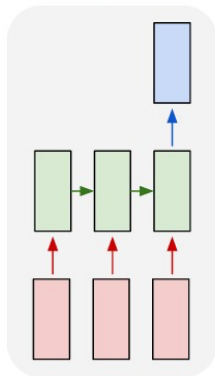
Karpathy's blog <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>



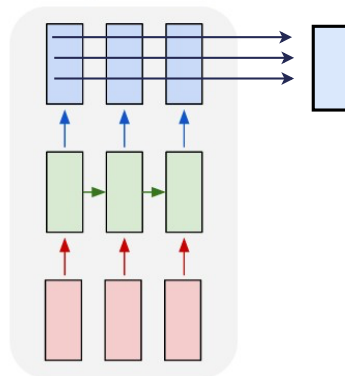
RNN, CNN & pooling

Architecture variation

many to one

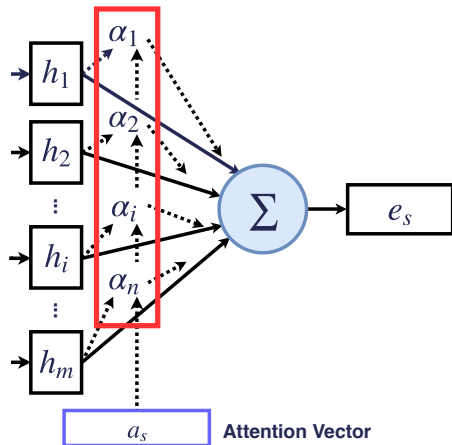


many to many



- What about CNN?

Attention & aggregation



- Learning to weight localized representation
- Many efficient applications [2014-2018]
- Few parameters
- Attention vector = Query vector
- How to code it in pytorch?

Practical Session:

Q1 Preliminary tests: uniform attention

Q2 Classical global attention



Advanced attention

- Attention depends on the input
- Attention on RNN outputs

- batch, padding & masked attention