Advanced Learning Models

Chapter II - Advanced CNN and RNN models

Julien Mairal & Xavier Alameda-Pineda

with the help of Jakob Verbeek and Laurent Besacier

MSIAM/MoSIG – 2019-2020

Table of Contents

Course Organisation (remastered)

Ressources

You can visit (often) the web page of the course

`http://lear.inrialpes.fr/people/mairal/teaching/2019-2020/MSIAM/`

Grading revisited

Homework (twice, 50%), Data Challenge (50%) and final exam (40%).

→ Homework 1: Given before Xmas.

→ Homework 2: Given on January 16th.

→ Data Challenge: given at the beginning of January. Results with at least one neural network and one kernel method.

NO machine learning libraries allowed!

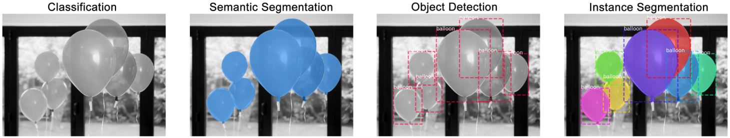
Advanced CNN architectures

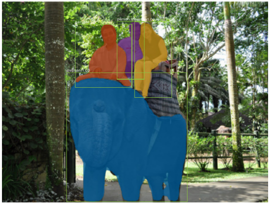
Advanced CNN architectures

– Object Detection –

Goal: To localise and classify each object in an image.

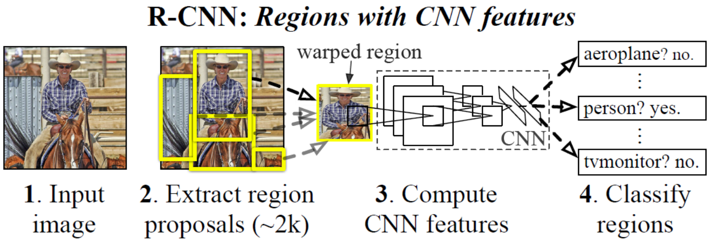
1. Find potential object locations/bounding boxes.
2. Characterise those candidates.
3. Classify them into object/non-object and object class.
4. Produce a final localisation bouding box (and segment them).





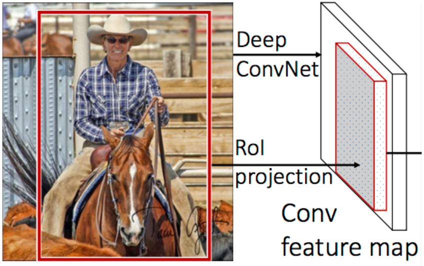
R-CNN (2014) [Girshick et al., 2014]

- Apply the Selective Search method to propose bounding boxes $\rightarrow$ 2k per image!!!
- Resize each of the b-boxes and feed-forward through AlexNet.
- Train an SVM to classify bounding boxes and a linear regressor to refine them.



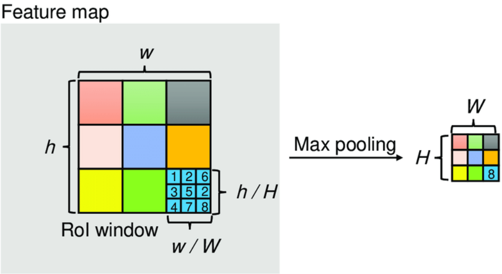
Fast R-CNN (2015) [Girshick, 2015]

- Apply Selective Search to propose bounding boxes.
- In parallel: feed-forward the original image (once).



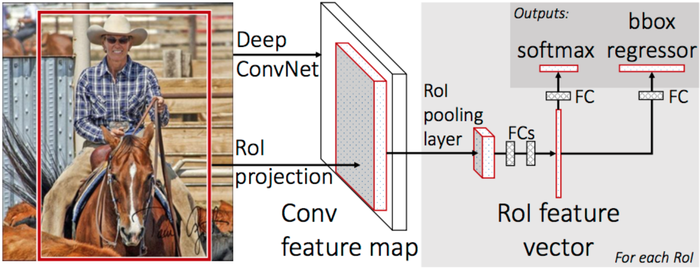
Fast R-CNN (2015) [Girshick, 2015]

- Apply Selective Search to propose bounding boxes.
- In parallel: feed-forward the original image (once).
- Apply Region-of-Interest (ROI) pooling.



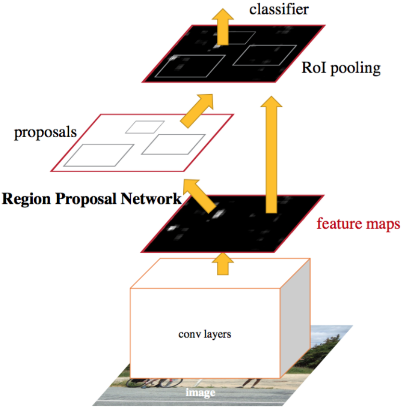
Fast R-CNN (2015) [Girshick, 2015]

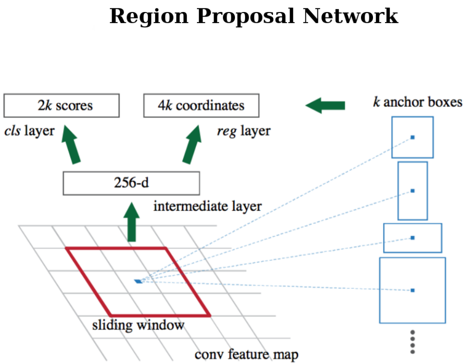
- Apply Selective Search to propose bounding boxes.
- In parallel: feed-forward the original image (once).
- Apply Region-of-Interest (ROI) pooling.
- Use a classification and linear regression layers.



Faster R-CNN (2015) [Ren et al., 2015]

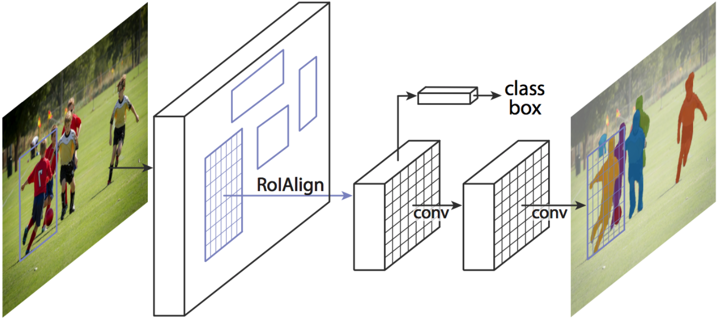
- Include the b-box proposal in the network.
  $\rightarrow$ Region Proposal Network.
- Sliding window providing b-box and objectness of each *anchor*.
- Classification and regression layers as well.





Mask R-CNN (2017) [He et al., 2017]

- On top of the Faster R-CNN object detection mechanism.
- For each b-box, add a pixel-wise binary mask.
- Segment each instance of each class.



Advanced CNN architectures

– Deep Metric Learning –

Metric Learning

Goal: Learn a metric (distance) in which samples are better spread for the task at hand.

- Similar to kernel methods (data projection).
- The projections are not designed, but learned.

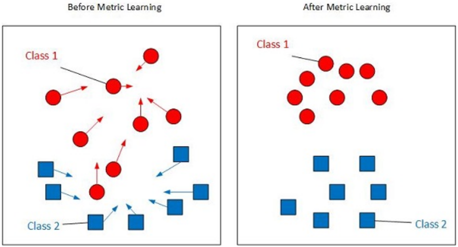


Image from `http://ml.cecs.ucf.edu/`.

Formalising Metric Learning

Learning from **link** information.

Must-link / cannot-link constraints:

$$\mathcal{S} = \{(x_i, x_j) : i, j \text{ should be similar.}\}$$

$$\mathcal{D} = \{(x_i, x_j) : i, j \text{ should be dissimilar.}\}$$

Relative constraints:

$$\mathcal{R} = \{(x_i, x_j, x_k) : i \text{ should be more similar to } j \text{ than to } k.\}$$

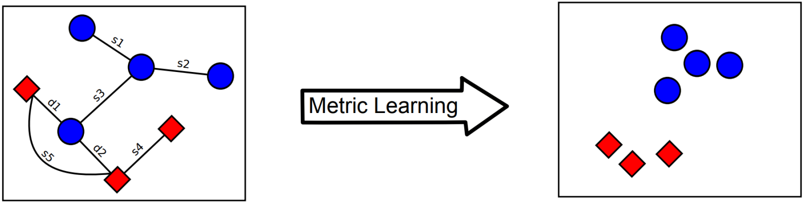


Image from http://researchers.lille.inria.fr/abellet/talks/metric_learning_tutorial_CIL.pdf.

Formalising Metric Learning

Learning from **link** information.

Must-link / cannot-link constraints:

$$\mathcal{S} = \{(x_i, x_j) : i, j \text{ should be similar.}\}$$

$$\mathcal{D} = \{(x_i, x_j) : i, j \text{ should be dissimilar.}\}$$

Relative constraints:

$$\mathcal{R} = \{(x_i, x_j, x_k) : i \text{ should be more similar to } j \text{ than to } k.\}$$

Metric learning optimisation problem

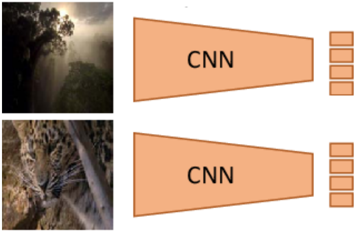
Find the optimal metric parameters M^*:

$$M^* = \arg\min_M \mathcal{L}(M; \mathcal{S}, \mathcal{D}, \mathcal{R}) + \lambda\Omega(M)$$

$\mathcal{L}$ penalises violated constraints.

Deep Metric Learning

Parametrize the new metric through a deep neural network:
$M = \{\text{convolutions, fully connected, etc.}\}$.



The **same** CNN for both input.

Shared weights.

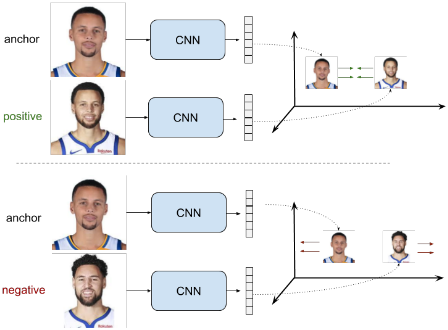
Loss pulling $\mathcal{S}$ samples together and pulling $\mathcal{D}$ samples apart.

Image from [Baraldi et al., 2015].

How to define this loss?

Contrastive (or pair-wise ranking) loss

$$\mathcal{L}_{\mathrm{c}}(\theta, \mathcal{S}, \mathcal{D}) = \begin{cases} d(\phi(x_i;\theta), \phi(x_j;\theta)) & (x_i, x_j) \in \mathcal{S} \\ \max(0, \tau - d(\phi(x_i;\theta), \phi(x_j;\theta))) & (x_i, x_j) \in \mathcal{D} \end{cases}$$



CNN ϕ parametrised by θ.

d is a standard distance (e.g. Euclidean).

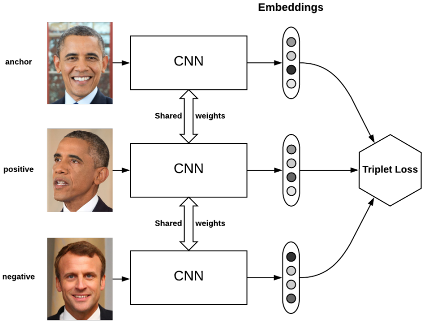
τ is a task-dependent parameter.

Image from `https://gombru.github.io/2019/04/03/ranking_loss/`.

What about relative constraints?

Recall the relative constraints:

$$\mathcal{R} = \{(x_i, x_j, x_k) : i \text{ should be more similar to } j \text{ than to } k.\}$$



$$\mathcal{L}_3(\theta, \mathcal{R}) = \max(0, d(\phi(x_i;\theta), \phi(x_j;\theta) - d(\phi(x_i;\theta), \phi(x_k;\theta)) + \tau)$$

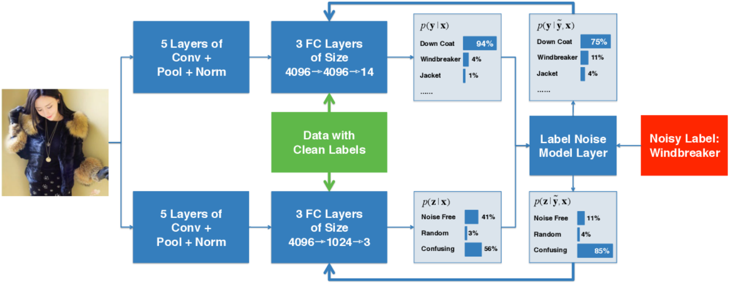
Image from `https://omoindrot.github.io/triplet-loss`.

Advanced CNN architectures

– Training with noisy labels –

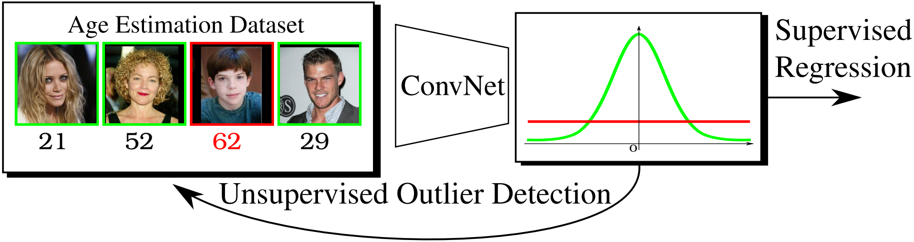
Training with noisy labels: classification [Xiao et al., 2015]

Motivation: annotating large-scale datasets is tedious.
$\rightarrow$ Train with noisy data (and perhaps a few clean data).



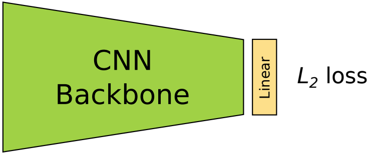
Two network paths estimating: the clean label y and the noise type z.
A probabilistic model mixes this information to "predict" a noisy label.
An EM is proposed to back-propagate the error.

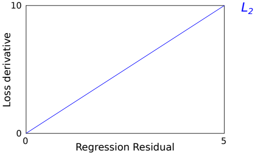
Same motivation, but for a regression task (continuous label).



Limitations of standard deep regression

Standard way: deep model + linear regression layer + L_2 loss:





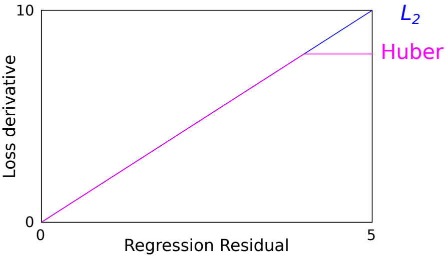
The larger the gradient, the more attention the network pays to it.

Gradient of the L_2 loss is 2δ, twice the residual.

Outliers have huge residual $\Rightarrow$ The network pays a lot of attention.

Existing solutions

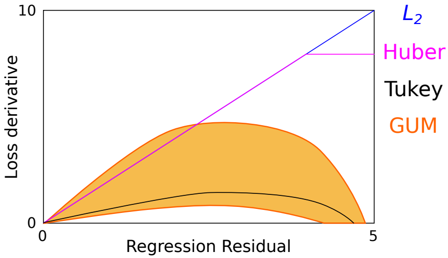
Let's take a look to existing solutions:



L_2/Huber large gradient for large δ.

Existing solutions

Let's take a look to existing solutions:



L_2/Huber large gradient for large δ.

Gaussian-Uniform Mixtures (GUM) offer a **family** of interpretable losses.

 21  52  62  29

Hypothesis: inliers $\leftrightarrow$ Gaussian outliers $\leftrightarrow$ Uniform.

$$p(y_i|x_i;\nu,\theta) = \underbrace{\rho}_{\text{Inlier prior}} \mathcal{N}(y_i;\phi(x_i;\theta),\Sigma) + \underbrace{(1-\rho)}_{\text{Outlier prior}} \mathcal{U}(y_i;\gamma),$$

Gaussian Uniform Mixtures

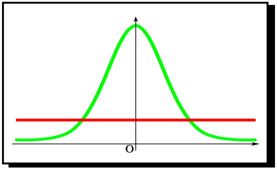
Hypothesis: inliers $\leftrightarrow$ Gaussian outliers $\leftrightarrow$ Uniform.

$$p(y_i|x_i;\nu,\theta) = \underbrace{\rho}_{\text{Inlier prior}} \mathcal{N}(y_i;\phi(x_i;\theta),\Sigma) + \underbrace{(1-\rho)}_{\text{Outlier prior}} \mathcal{U}(y_i;\gamma),$$

$\phi(\cdot;\theta)$: forward with weights θ

$\nu = \{\rho, \Sigma, \gamma\}$: parameters of GUM

Challenge: How to learn θ and ν?

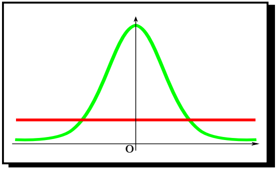


How to train?

Main idea: Expectation-maximisation (EM).

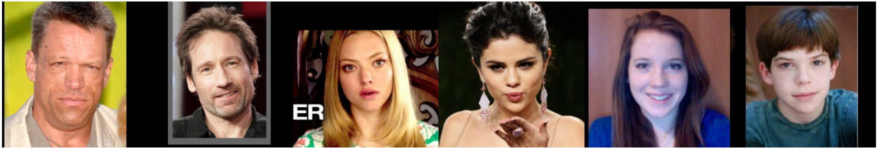
- E-step: $r_i(\nu^{(r)}) = p(x_i, y_i | \nu^{(r)})$.
- M-ν step: update ν with (almost) standard formulae.
- M-θ step: update θ by minimising

$$\mathcal{L}_{\mathrm{GUM}} = \sum_{i=1}^{I} r_i(\nu^{(r)}) \|y_i - \phi(x_i; \theta)\|^2.$$



Outliers detected (age estimation)

(a) 14 (b) 14 (c) 14 (d) 16 (e) 20 (f) 23



(g) 49 (h) 51 (i) 60 (j) 60 (k) 60 (l) 62

Advanced CNN architectures

– Object tracking –

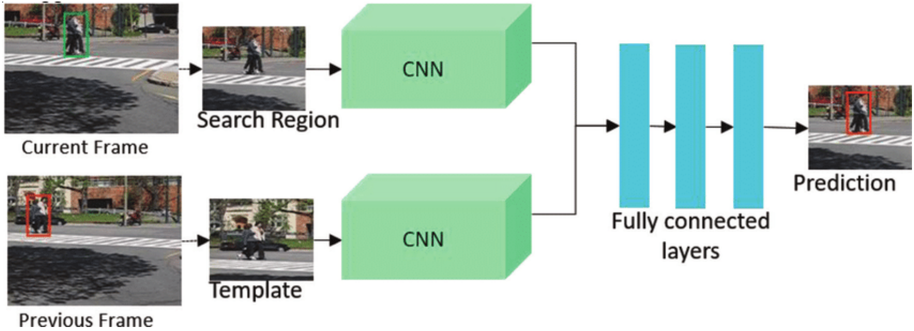
Object tracking problem

Goal: provide the localisation of an object over time.

- The object is generic: unknown appearance.
- The appearance of the object changes over time (illumination, distance, etc).
- The object moves within a reasonable range.

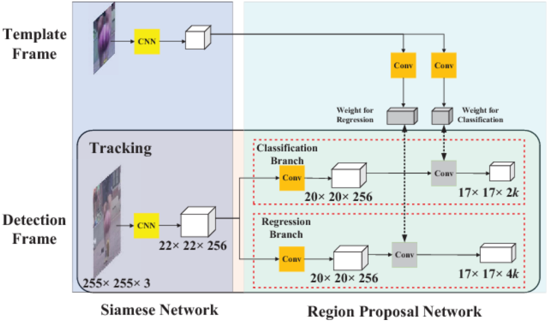
Reformulate as image search [Henriques et al., 2014]

- The problem is reformulated as the task of finding a target image (template, previous frame) within a search region (current frame).
- Convolutional features are extracted, and then trained with the fully connected layers to predict the bbox.



Siamese RPN [Li et al., 2018]

- Similar logic, but features are extracted with a siamese network.
- A region proposal network is then used to propose bboxes through the anchoring mechanism.



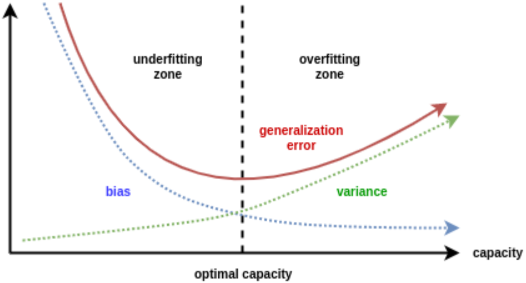
CNN and Deep learning – Meta

CNN and Deep learning – Meta

– Early stopping –

Early stopping

- Neural networks have millions of parameters.
- They are prone to **overfit**, i.e. have limited generalisation.
- In practice, performance in training $>>$ than in test.



Underfit: high bias but low error variance.
Overfit: low bias but high error variance.

Early stopping: point in which the validation error stops decreasing ($=$ the generalisation capacity stops increasing).

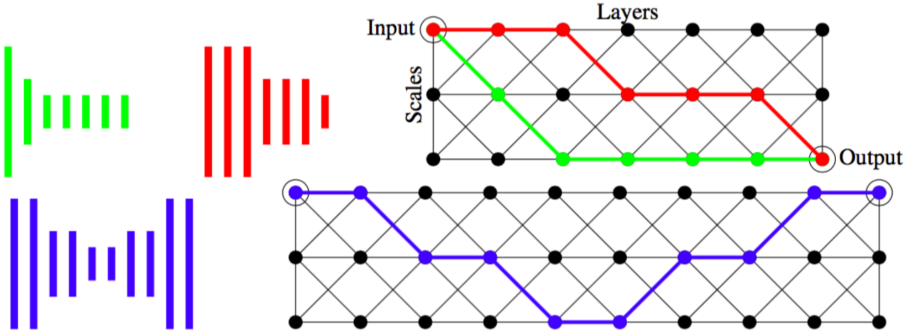
CNN and Deep learning – Meta

– Architecture search –

Architecture search

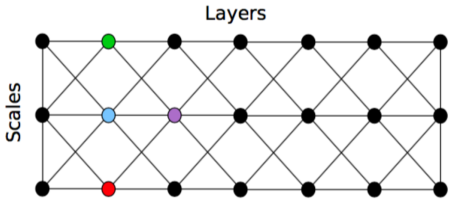
Motivation: how to find the right architecture choice: # layers, resolution, etc.

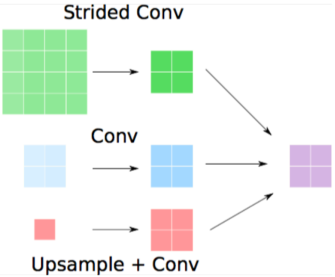
- Several approaches: one possibility is to train the all at once!
- Grid of network layers across multiple scales.
- U-net and standard conv are special cases.



Convolutional neural fabrics [Saxena and Verbeek, 2016]

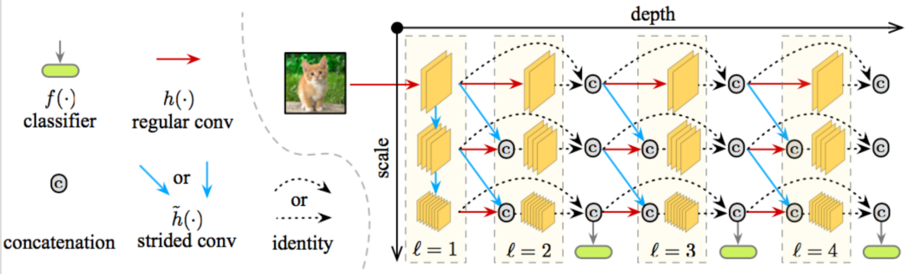
- Each feature map receives input from three others
  - Scale finer: strided convolution
  - Scale coarser: stride coarse activations on finer resolution, then covolution
  - Same scale: standard convolution
- Generalizes very large class of networks with “standard” layers
- With enough layers and feature channels, 3x3 convolutions suffice for
  - Average pooling, max-pooling, and strided convolition
  - Nearest-neighbor, bi-linear, and general deconvolution up-sampling
  - Filters of any size by distribution over layers





Multi-scale Dense Convolutional Networks [Huang et al., 2017]

- Grid of network layers across multiple scales
- Feed-forward and dense connections across the horizontal “layer axis”
- Down-sampling across all layers for classification
- Intermediate classifiers for any-time prediction
- Efficient any-time prediction model

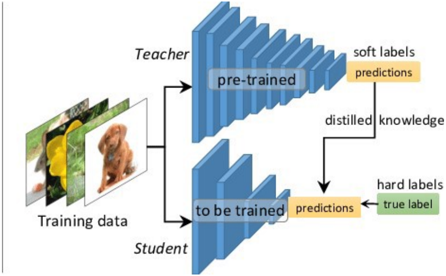


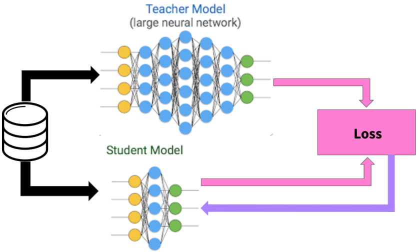
CNN and Deep learning – Meta

– Distilling knowledge –

Distillation [Hinton et al., 2015]

- A teacher (large) and a student (small) network.
- Teacher is pre-trained.
- The student is trained to imitate the output of the teacher network (before soft-max).
- Training the student directly does not work!



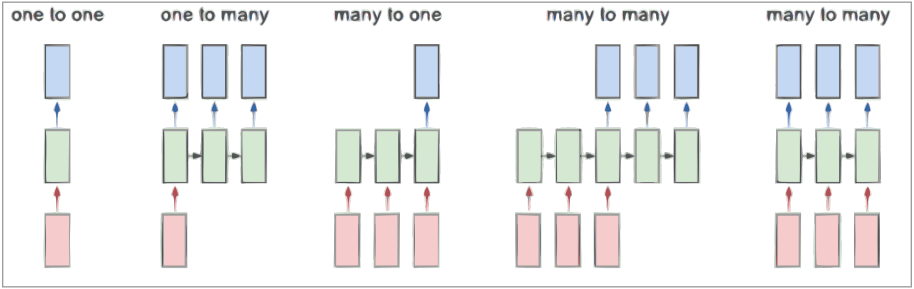


Recurrent Neural Networks

Recurrent Neural Networks

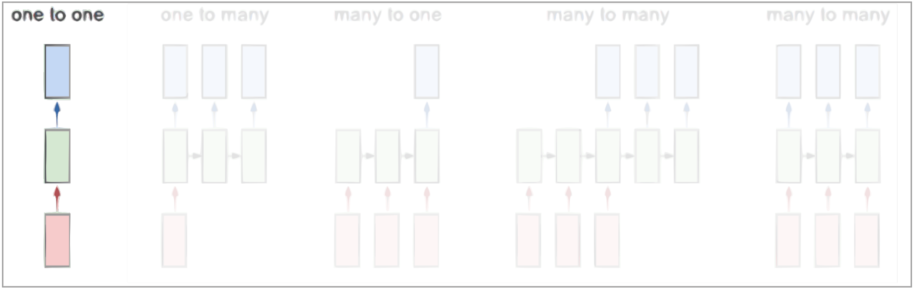
– Principle of RNN –

RNN: Uses



Sequential information – variable length – for input/output (or both).

RNN: Uses



Classification/regression: one image $\leftrightarrow$ one label.

RNN: Uses

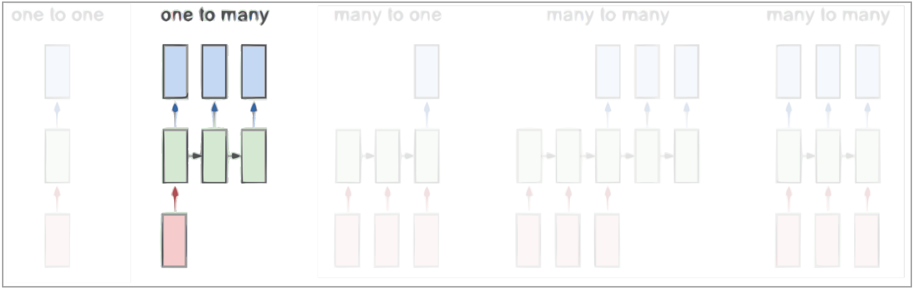
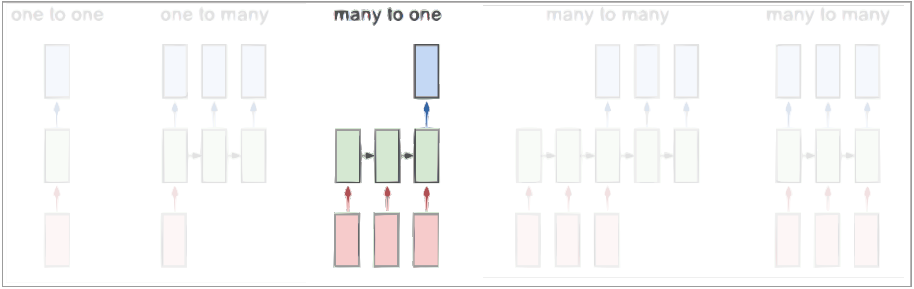


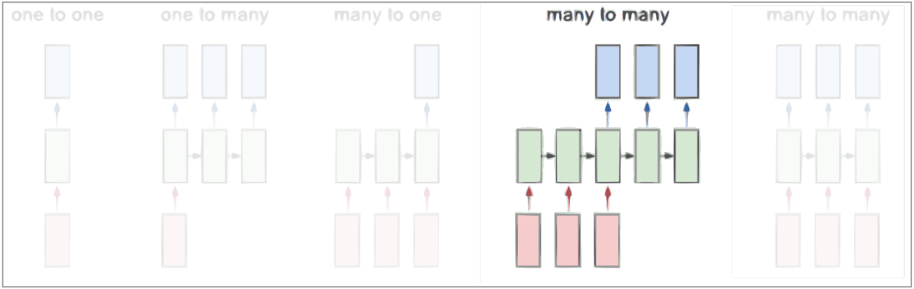
Image captioning: one image $\leftrightarrow$ a word sequence.

RNN: Uses



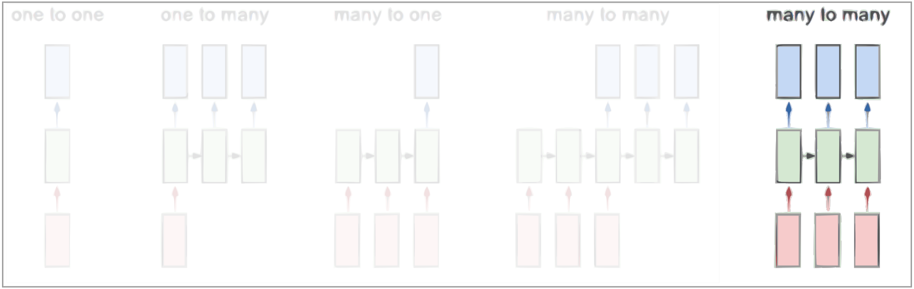
Rating assessing: one evaluation comment $\leftrightarrow$ a satisfaction score.

RNN: Uses



Machine translation: text in language A $\leftrightarrow$ text in language B.

RNN: Uses

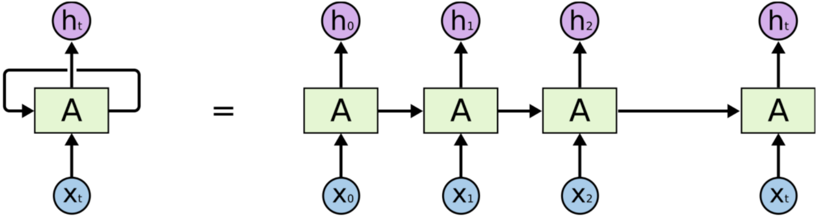


Paired sequential data: predict phoneme labels over time.

RNN: Rationale

- Recurrent computation of hidden units from t to $t+1$:
  - Hidden state accumulates information on entire sequence, since the field of view spans entire sequence processed so far
  - Time-invariant function makes it applicable to arbitrarily long sequences
- Similar ideas used in:
  - Hidden Markov models for arbitrarily long sequences
  - Parameter sharing across space in convolutional neural networks
  - But has limited field of view: parallel instead of sequential processing

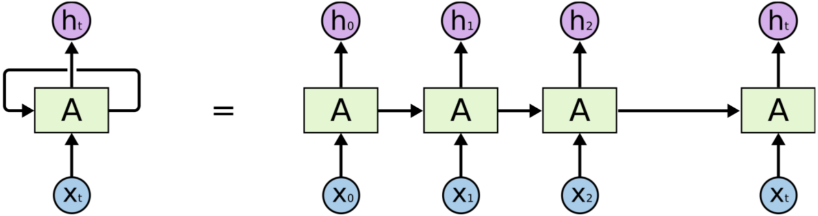
RNN unfolding



Left: *folded* representation $\rightarrow$ loops.
Right: *unfolded* representation $\Rightarrow$ we call them "deep".

- Unfolded representation shows an acyclic directed graph.
- Size of the graph (horizontally) is variable, given by sequence length.
- Weights shared across horizontal replications.
- Gradient computation known as "back-propagation through time."

RNN training



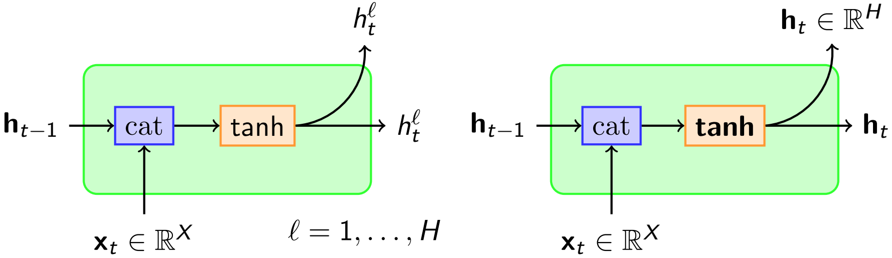
- Deterministic feed-forward network from inputs to outputs.
- Softmax can be used to map the output $\mathbf{h}_t$ into a discrete distribution.
- Independent prediction of elements in output given input sequence.
- We can still maximise the log-probability of the class:

$$\mathcal{L}(\mathbf{W}) = -\sum_{t=1}^{T} \log p(\mathrm{softmax}(\mathbf{h}_t) | x_{1:t}; \mathbf{W})$$

Recurrent Neural Networks

– Formalising RNN –

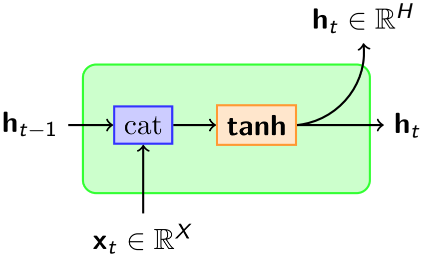
Basic RNN building block:



- Left: scalar output. Right: vector output.
- Input are concatenated before a linear transformation.
- Non-linear activation (tanh) or its element-wise form (**tanh**).

RNN forward pass

Concatenation, linear transform, element-wise activation.



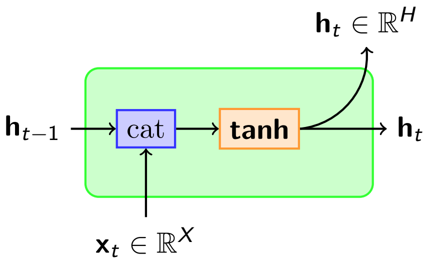
$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

$$\mathbf{z}_t = \mathbf{W}^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\text{with} \quad \mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \ldots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \ldots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \ldots \mathbf{W}_1^H \end{pmatrix} \in \mathbb{R}^{(H+X+1)\times H}.$$

RNN backward pass

What gradients do we need (if we have $\dfrac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)?



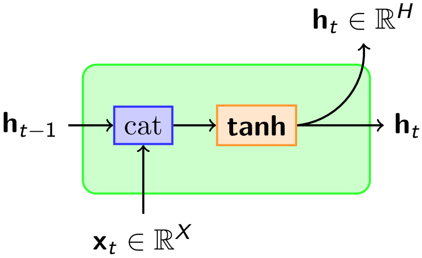
$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

$$\mathbf{z}_t = \mathbf{W}^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \ldots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \ldots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \ldots \mathbf{W}_1^H \end{pmatrix}.$$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



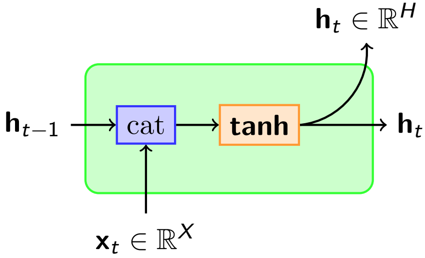
$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

$$\mathbf{z}_t = \mathbf{W}^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \ldots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \ldots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \ldots \mathbf{W}_1^H \end{pmatrix}.$$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

$$\mathbf{z}_t = \mathbf{W}^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

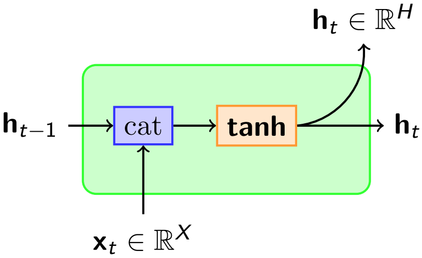
$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \dots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \dots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \dots \mathbf{W}_1^H \end{pmatrix}.$$

(0) Let's start with: $\frac{\partial h_t^\ell}{\partial z_t^k} =$

$\tanh' = 1 - \tanh^2$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

$$\mathbf{z}_t = \mathbf{W}^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \dots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \dots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \dots \mathbf{W}_1^H \end{pmatrix}.$$

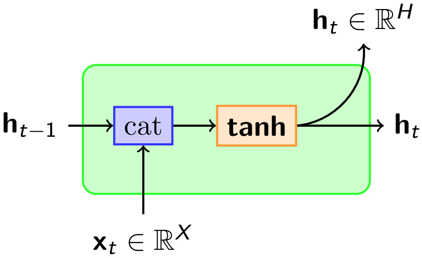
(0) Let's start with: $\frac{\partial h_t^\ell}{\partial z_t^k} = \delta_{kl}(1 - \tanh^2(z_t^\ell))$.

$\tanh' = 1 - \tanh^2$

And therefore: $\frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} =$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

$$\mathbf{z}_t = \mathbf{W}^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \dots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \dots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \dots \mathbf{W}_1^H \end{pmatrix}.$$

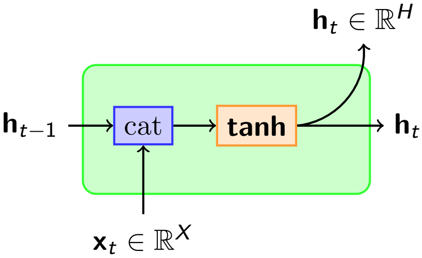
(0) Let's start with: $\frac{\partial h_t^\ell}{\partial z_t^k} = \delta_{kl}(1 - \tanh^2(z_t^\ell))$.

$\tanh' = 1 - \tanh^2$

And therefore: $\frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} = \mathrm{diag}(1 - \mathbf{tanh}^2(\mathbf{z}_t)) \in \mathbb{R}^{H \times H}$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

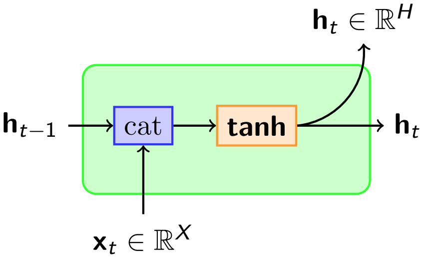
$$\mathbf{z}_t = \mathbf{W}^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \dots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \dots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \dots \mathbf{W}_1^H \end{pmatrix}.$$

(1) $\frac{\partial z_t^\ell}{\partial \mathbf{h}_{t-1}} =$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

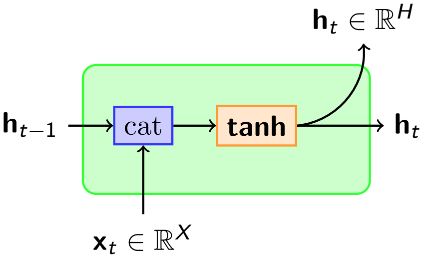
$$\mathbf{z}_t = \mathbf{W}^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \dots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \dots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \dots \mathbf{W}_1^H \end{pmatrix}.$$

(1) $\frac{\partial z_t^\ell}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_{h\cdot}^\ell \Rightarrow \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} =$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

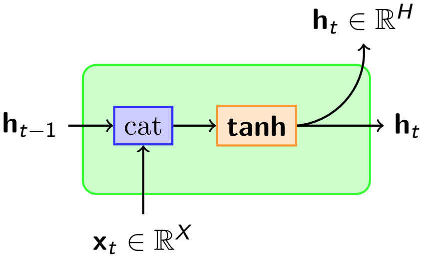
$$\mathbf{z}_t = \mathbf{W}^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \dots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \dots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \dots \mathbf{W}_1^H \end{pmatrix}.$$

(1) $\frac{\partial z_t^\ell}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_h^\ell. \Rightarrow \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_h^\top \Rightarrow \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} =$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

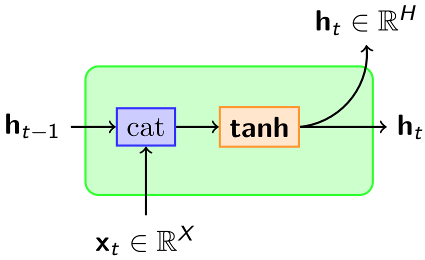
$$\mathbf{z}_t = \mathbf{W}^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \dots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \dots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \dots \mathbf{W}_1^H \end{pmatrix}.$$

$$(1)\ \frac{\partial z_t^\ell}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_h^\ell. \Rightarrow \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_h^\top \Rightarrow \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_h^\top \mathrm{diag}(1 - \mathbf{tanh}^2(\mathbf{z}_t))$$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

$$\mathbf{z}_t = \mathbf{W}^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

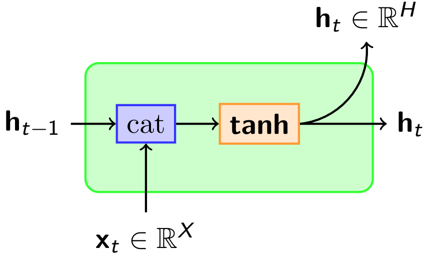
$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \dots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \dots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \dots \mathbf{W}_1^H \end{pmatrix}.$$

$$(1)\ \frac{\partial z_t^\ell}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_h^\ell. \Rightarrow \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_h^\top \Rightarrow \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \mathbf{W}_h^\top \mathrm{diag}(1 - \mathbf{tanh}^2(\mathbf{z}_t))$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-T}} = \prod_{\tau=t-T+1}^{t} \mathbf{W}_h^\top \mathrm{diag}(1 - \mathbf{tanh}^2(\mathbf{z}_\tau))$$

RNN backward pass

What gradients do we need (if we have $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$)? (1) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ and (2) $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$.



$$\mathbf{h}_t = \mathbf{tanh}(\mathbf{z}_t),$$

$$\mathbf{z}_t = \mathbf{W}^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1],$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h^1 & \dots \mathbf{W}_h^H \\ \mathbf{W}_x^1 & \dots \mathbf{W}_x^H \\ \mathbf{W}_1^1 & \dots \mathbf{W}_1^H \end{pmatrix}.$$

(2) Not more difficult than that, but tedious to write. One needs to first consider the column-wise vectorisation of $\mathbf{W}$, $\mathrm{vec}(\mathbf{W})$:

$$\frac{\partial \mathbf{h}_t}{\mathrm{vec}(\mathbf{W})} = \mathrm{blkdiag}_\ell \left((1 - \tanh^2(z_t^\ell))[\mathbf{h}_{t-1}; \mathbf{x}_t; 1]^\top \right)$$

Vanishing/Exploding gradient

Let us retake:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-T}} = \prod_{\tau=t-T+1}^{t} \mathbf{W}_h^\top \mathrm{diag}(1 - \mathbf{tanh}^2(\mathbf{z}_\tau)).$$

- $\mathrm{diag}(1 - \mathbf{tanh}^2(\mathbf{z}_\tau))$ are diagonal matrices with elements in $[0, 1]$.
  $\rightarrow$ the gradient is multiplied by small numbers, specially for saturated neurons.
- An alternative could be to use ReLu activation
  $\rightarrow$ the forward pass would easily overflow.
- Long-short term memory (LSTM) recurrent networks were proposed in 1997 to overcome this problem.
  $\rightarrow$ use **gates** to stop/let pass the information to the next time step.

Recurrent Neural Networks

– Long-short term memory networks –

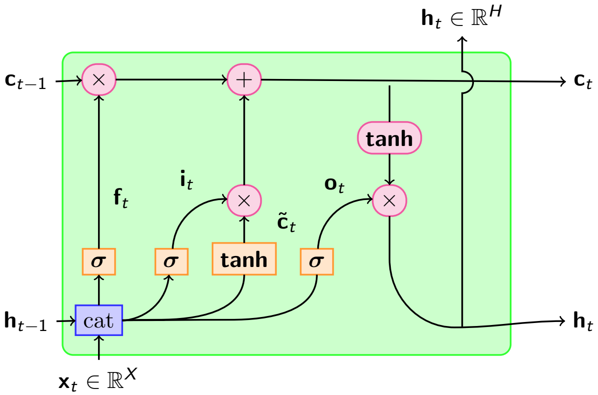
Long-short term memory (LSTM) networks

[Hochreiter and Schmidhuber, 1997]

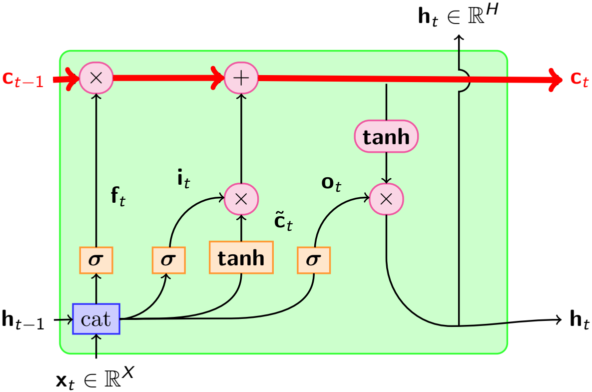
Intuition: use gates to stop/let pass the information.

- Gates are paired with "information" (classical) neurons.
- When a gate neuron fires, the information of the corresponding neuron must be *kept* for the next time step.
- Otherwise, this information must be *forgotten*.
- This behaviour is achieved with a sigmoid activation and element-wise product.
- From a computational perspective there is **NO** difference between gate and previous neurons: we are learning all weights at once.

LSTM: Diagram & forward pass

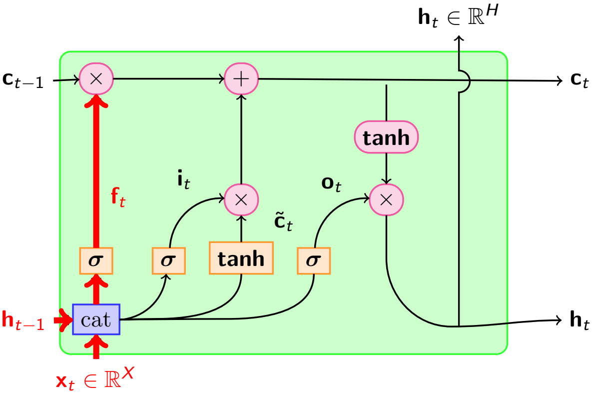


LSTM: Diagram & forward pass



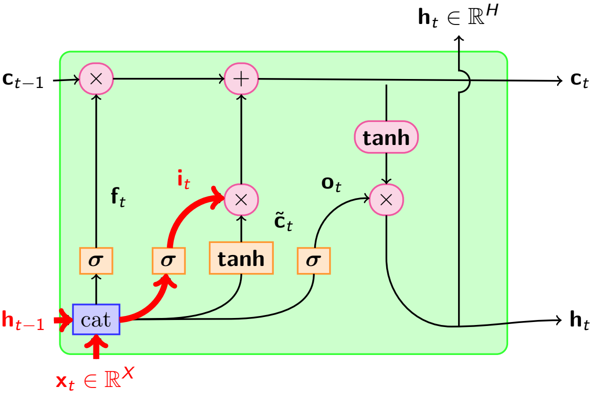
The information flows directly from previous step through the *cell state*.

LSTM: Diagram & forward pass



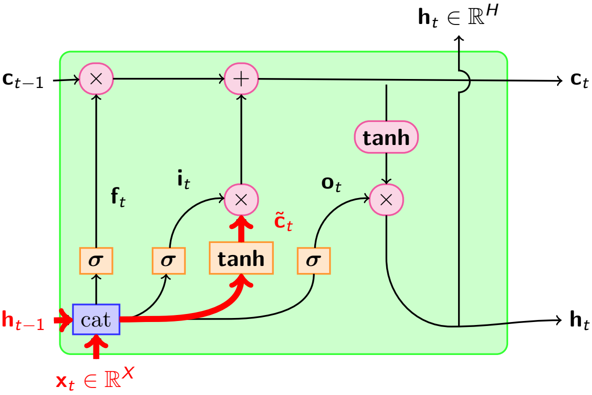
$$\mathbf{f}_t = \boldsymbol{\sigma}(\mathbf{z}_{ft}), \quad \mathbf{z}_{ft} = \mathbf{W}_f^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1], \quad \mathbf{W}_f \in \mathbb{R}^{H\times(H+X+1)}$$

LSTM: Diagram & forward pass



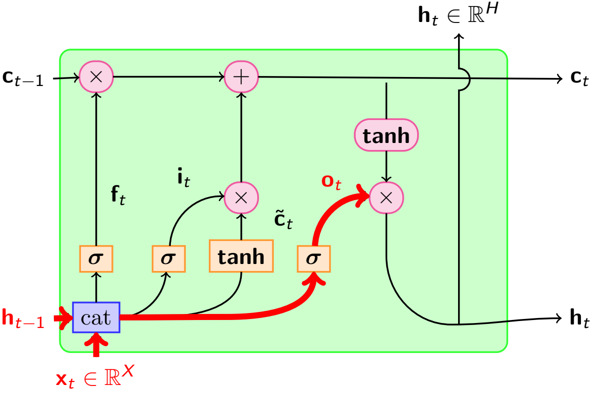
$$\mathbf{i}_t = \boldsymbol{\sigma}(\mathbf{z}_{it}), \quad \mathbf{z}_{it} = \mathbf{W}_i^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1], \quad \mathbf{W}_i \in \mathbb{R}^{H \times (H+X+1)}$$

LSTM: Diagram & forward pass



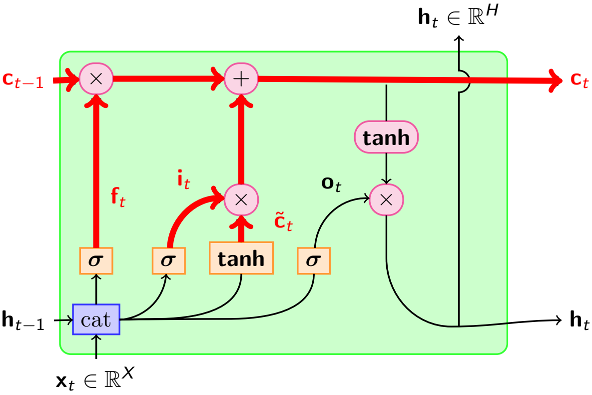
$$\tilde{\mathbf{c}}_t = \mathbf{tanh}(\mathbf{z}_{ct}), \quad \mathbf{z}_{ct} = \mathbf{W}_c^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1], \quad \mathbf{W}_c \in \mathbb{R}^{H \times (H+X+1)}$$

LSTM: Diagram & forward pass



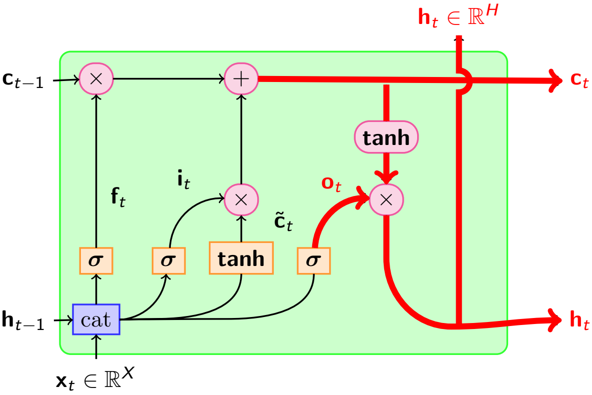
$$\mathbf{o}_t = \boldsymbol{\sigma}(\mathbf{z}_{ot}), \quad \mathbf{z}_{ot} = \mathbf{W}_o^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1], \quad \mathbf{W}_o \in \mathbb{R}^{H\times(H+X+1)}$$

LSTM: Diagram & forward pass



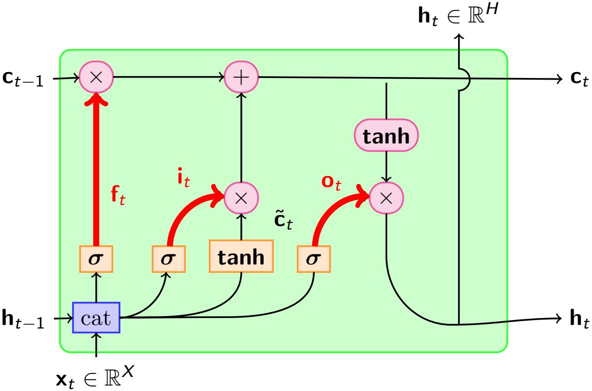
$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$ ($\odot$ is the element-wise product)

LSTM: Diagram & forward pass



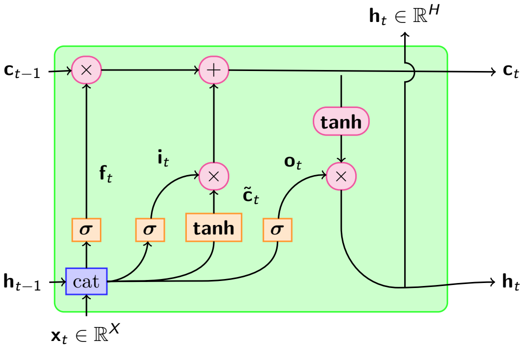
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LSTM: Diagram & forward pass



Gates: $\mathbf{f}_t$ forget $\mathbf{i}_t$ input $\mathbf{o}_t$ output.

LSTM: Backward pass



$$\mathbf{f}_t = \sigma(\mathbf{W}_f^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1])$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1])$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1])$$

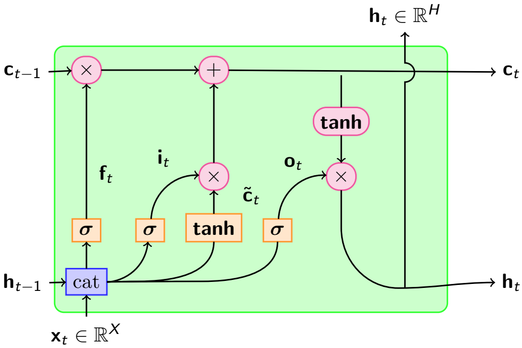
$$\mathbf{o}_t = \sigma(\mathbf{W}_o^\top [\mathbf{h}_{t-1}; \mathbf{x}_t; 1])$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Which one is the most interesting?

LSTM: Backward pass



$$\mathbf{f}_t = \sigma(\mathbf{W}_f^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1])$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1])$$

$$\tilde{\mathbf{c}}_t = \mathbf{tanh}(\mathbf{W}_c^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1])$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o^\top[\mathbf{h}_{t-1}; \mathbf{x}_t; 1])$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{tanh}(\mathbf{c}_t)$$

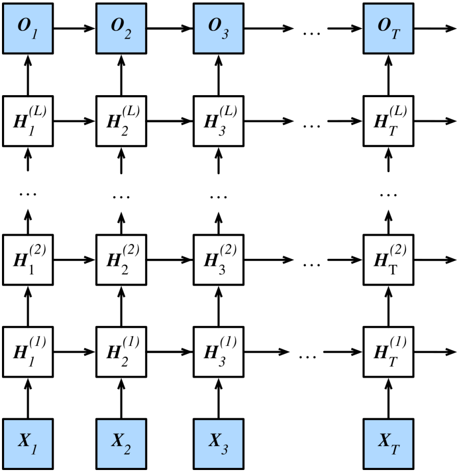
Which one is the most interesting?

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \mathrm{diag}(\mathbf{f}_t).$$

Recurrent Neural Networks

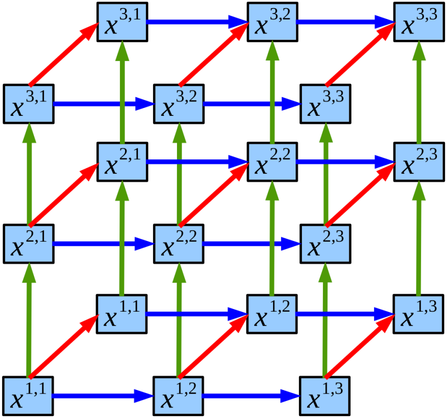
– Advanced RNN –

More topologies (I): deep RNN



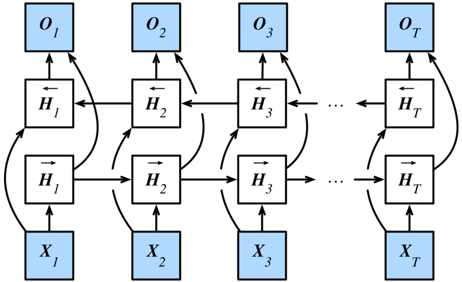
- Many layers with recurrent relationships.

More topologies (II): multi-dimensional RNN [Graves et al., 2007]



- Instead of a single recurrence: one per dimension.
- Each node recives input from predecessors, one per dimension.

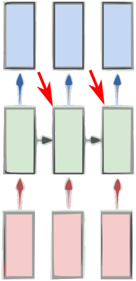
More topologies (III): bi-directional RNN [Graves et al., 2013]



- Use also right-to-left connections (anti-causal).
- Two separate recurrences, and aggregation.
- The decision computation of $\mathbf{o}_t$ depends on all input $\mathbf{x}_1, \ldots, \mathbf{x}_T$.

More topologies (IV): output feedback loops

- In many applications (machine translation), the output needs to be sampled (from output dist.).
- So far the output elements are independently sampled for each t given the state.
- When sampling the output to take a decision, the sample could be **fed back** into the next state.



- Without output-feedback: deterministic non-linear dynamical system.
- With output-feedback: stochastic non-linear dynamical system.

$$p(\mathbf{o}_{1:T}|\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{o}_t|\mathbf{x}_{1:t}) \quad \text{vs.} \quad p(\mathbf{o}_{1:T}|\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(\mathbf{o}_t|\mathbf{x}_{1:t}, \mathbf{o}_{1:t-1})$$

How do we sample from an RNN?

- RNN provide a distribution over the output sequence.
- Sample sequentially one output at a time:
  - Compute state from current input and previous state/output.
  - Compute the distribution on current output symbol.
  - Sample output symbol.

- Other interesting items to compute
  (not possible with output feedback loops):
  - Maximum likelihood sequence.
  - Marginal distribution of the n-th output symbol.
  - Marginal probability of a given symbol anywhere in the sequence.

How to train an RNN with/without output feedback?

Without output feedback

- Compute full state sequence given input.
- Sample output independently.
- Compute the loss and back-propagate (through time).

How to train an RNN with/without output feedback?

Without output feedback

- Compute full state sequence given input.
- Sample output independently.
- Compute the loss and back-propagate (through time).

With output feedback

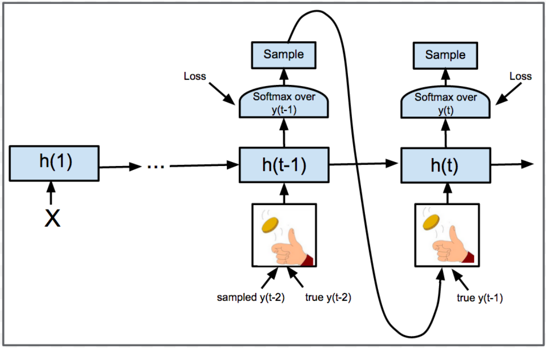
- Compute full state sequence given input **and ground-truth output**.
- Sample output, compute the loss and back-propagate (through time).

Notice train/test difference:

- Train: predict next symbol from causal input and **ground-truth output**.
- Test: predict next symbol from causal input and **generated output**.

Scheduled sampling for RNN training [Bengio et al., 2015]

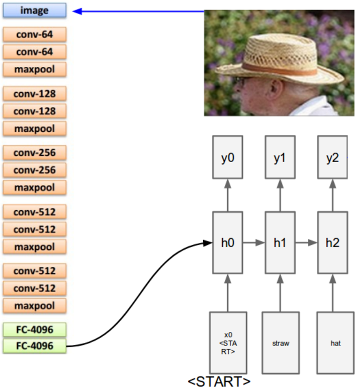
- Compensate train/test discrepancy by training from generated as well.
- Direct training from generated sequences does not work well: cumulated sequential error is huge!
- Choose randomly from generated or ground-truth output.
- Initialise with low probability of choosing generated, increase it with training progress.



Encoding/decoding recurrent architectures [Sutskever et al., 2014]

Example 1: Image captioning

- Encoder: CNN inputs an image and maps it to a vector.
- Decoder: RNN state initialised with the image vector.



Example 1: Image captioning - sample

"man in black shirt is playing guitar."

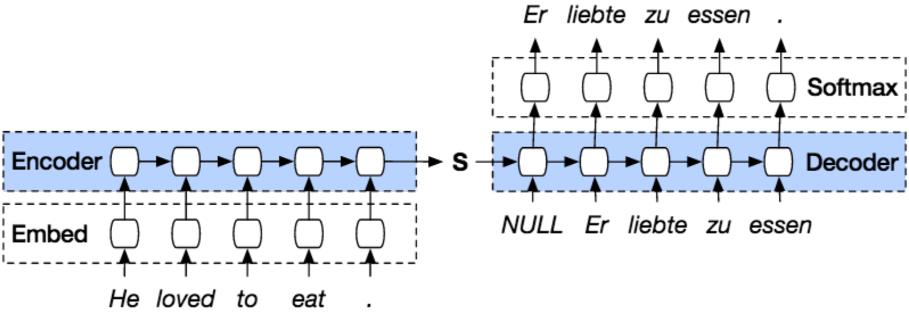
"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

Encoding/decoding recurrent architectures

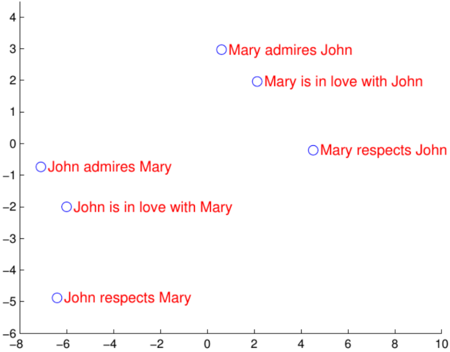
Example 2: Machine translation

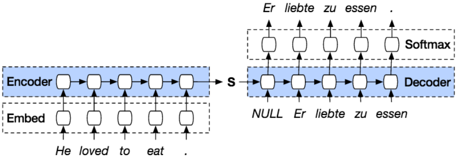
- Read source sentence with encoder RNN
  (Can use bidirectional RNN since input sequence is given)
- Generate target sentence with decoder RNN
  - Uses a different set of parameters
  - Uses output feedback to ensure output coherency
- Meaning of source sentence encoded in the RNN state vector passed between encoder and decoder



Example 2: Machine translation

- Meaning of source sentence encoded in the RNN state vector passed between encoder and decoder



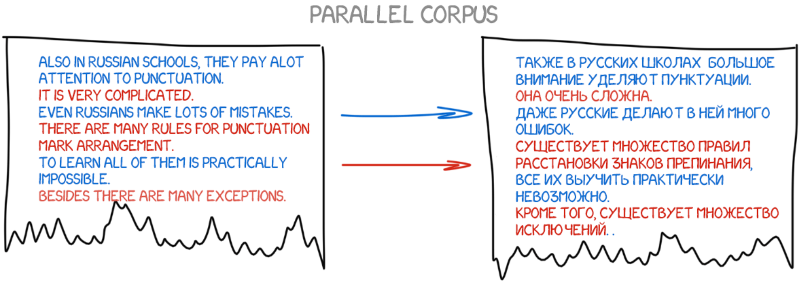


PCA projection of the source sentence encoded in **S**.

The word order matters!!!

Example 2: Machine translation - training

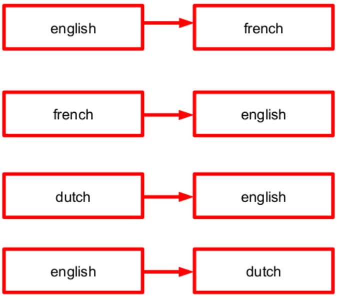
- A "parallel" or "paired" corpus is required.

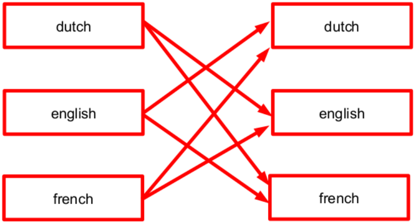


Encoding/decoding recurrent architectures

Example 2: Machine translation - training

- A “parallel” or “paired” corpus is required.
- Extension to multilanguage is easy thanks to encoder/decoder.

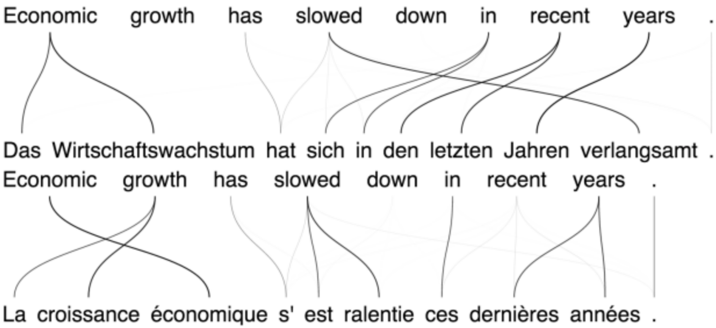




Attention Mechanisms in RNN

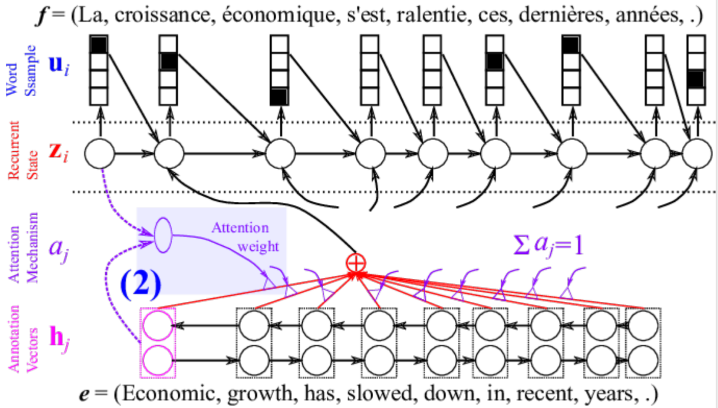
Motivation: Encoder-decoder based models compress the input sequence into a single vector $\rightarrow$ difficult to encode large sequences.

Using LSTM, BiRNN may help but does not suffice. The RNN cannot pay attention to EVERYTHING.



Attention Mechanisms in RNN (II)

- Let decoder attend to part of the input for each state update
  - Selectively: based on current state and input representation
  - Should work for input sequences of variable size
- Sub-network takes state and input, computes attention weights
- Feed weighted sum of inputs to the state update



References I

Baraldi, L., Grana, C., and Cucchiara, R. (2015).
A deep siamese network for scene detection in broadcast videos.
arXiv preprint arXiv:1510.08893.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015).
Scheduled sampling for sequence prediction with recurrent neural networks.
In *NeurIPS*, pages 1171–1179.

Girshick, R. (2015).
Fast r-cnn.
In *ICCV*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014).
Rich feature hierarchies for accurate object detection and semantic segmentation.
In *CVPR*.

Graves, A., Fernández, S., and Schmidhuber, J. (2007).
Multi-dimensional recurrent neural networks.
In *International conference on artificial neural networks*, pages 549–558. Springer.

Graves, A., Jaitly, N., and Mohamed, A.-r. (2013).
Hybrid speech recognition with deep bidirectional lstm.
In *IEEE workshop on automatic speech recognition and understanding*, pages 273–278. IEEE.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017).
Mask r-cnn.
In *ICCV*.

References II

Henriques, J. F., Caseiro, R., Martins, P., and Batista, J. (2014).
High-speed tracking with kernelized correlation filters.
PAMI, 37(3):583–596.

Hinton, G., Vinyals, O., and Dean, J. (2015).
Distilling the knowledge in a neural network.
arXiv preprint arXiv:1503.02531.

Hochreiter, S. and Schmidhuber, J. (1997).
Long short-term memory.
Neural computation, 9(8):1735–1780.

Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L., and Weinberger, K. Q. (2017).
Multi-scale dense convolutional networks for efficient prediction.
arXiv preprint arXiv:1703.09844.

Lathuilière, S., Mesejo, P., Alameda-Pineda, X., and Horaud, R. (2018).
Deepgum: Learning deep robust regression with a gaussian-uniform mixture model.
In *ECCV*.

Li, B., Yan, J., Wu, W., Zhu, Z., and Hu, X. (2018).
High performance visual tracking with siamese region proposal network.
In *CVPR*.

Ren, S., He, K., Girshick, R., and Sun, J. (2015).
Faster r-cnn: Towards real-time object detection with region proposal networks.
In *NeurIPS*.

References III

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988).
Learning representations by back-propagating errors.
Cognitive modeling, 5(3):1.

Saxena, S. and Verbeek, J. (2016).
Convolutional neural fabrics.
In *NeurIPS*.

Sutskever, I., Vinyals, O., and Le, Q. (2014).
Sequence to sequence learning with neural networks.
In *NeurIPS*.

Xiao, T., Xia, T., Yang, Y., Huang, C., and Wang, X. (2015).
Learning from massive noisy labeled data for image classification.
In *CVPR*.