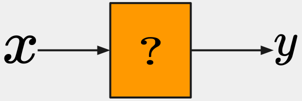
VIR - Lecture 9

Structure, recurrency, convolutions and more.

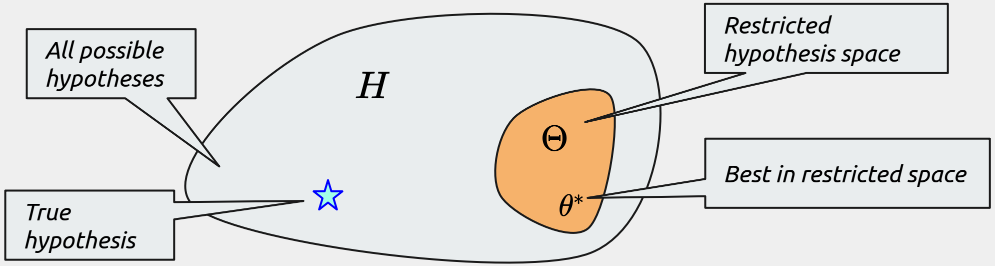
Quick learning review



Task formulation:

Given dataset $D = \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$ find hypothesis $h \in H$ which “explains” the data

Step 1: Restrict hypothesis space to something more manageable: $f_\theta : \theta \in \Theta$



Step 2: Assume that $(x_i, y_i) \sim P_{xy}$ and that it was 'reasonably' sampled (i.i.d, etc)

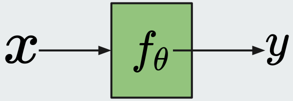
Step 3: Minimize empirical risk and hope for the best.

$$\theta^* = \arg\min_{\theta} \mathcal{L}(D, f_\theta)$$

Using what you know as a 'Loss function' $\mathcal{L}(D, f_\theta)$

Q: How do we restrict the hypothesis space?

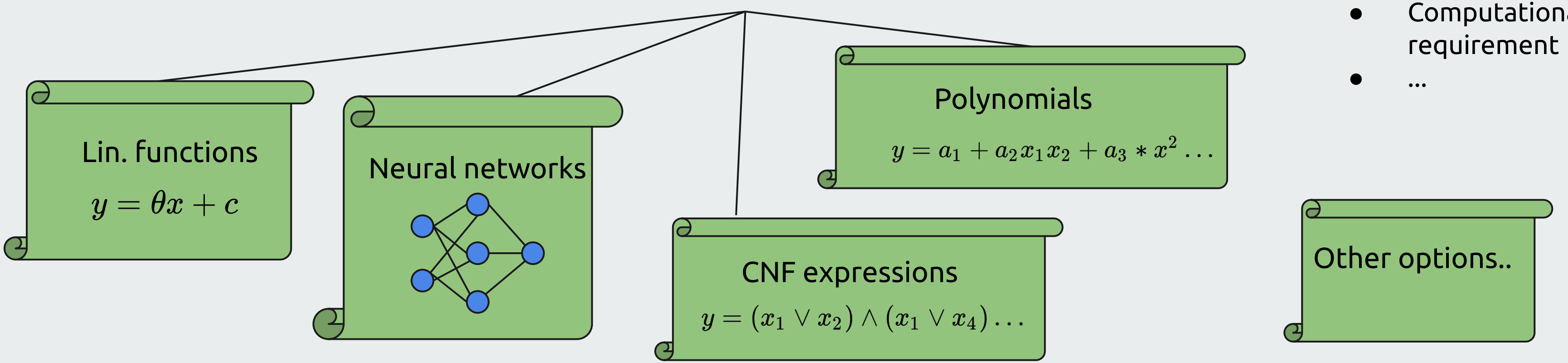
A: Propose a suitable parameterized function.



Criteria:

- Expressive power
- Ease of use
- Computational requirement
- ...

Many potential candidates...

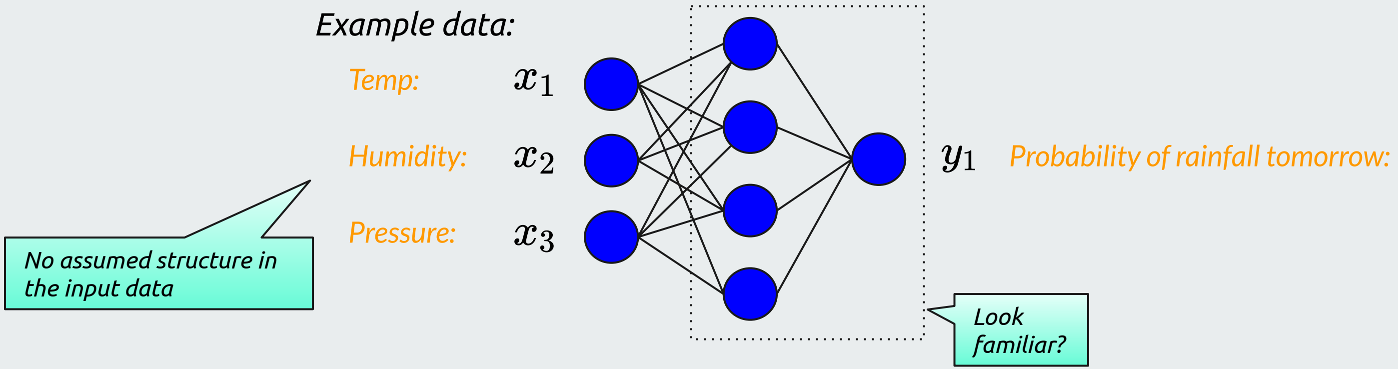


We will be looking mostly at neural networks and their extensions

Q: Ok, so what is the “neural network” class?

A: Loosely defined as a highly parametric functions which assume connected structure

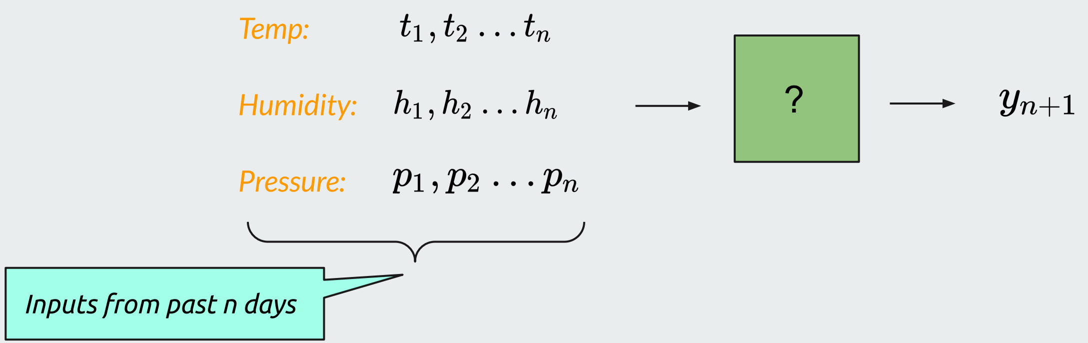
Vanilla MLP (Multi layer perceptron)



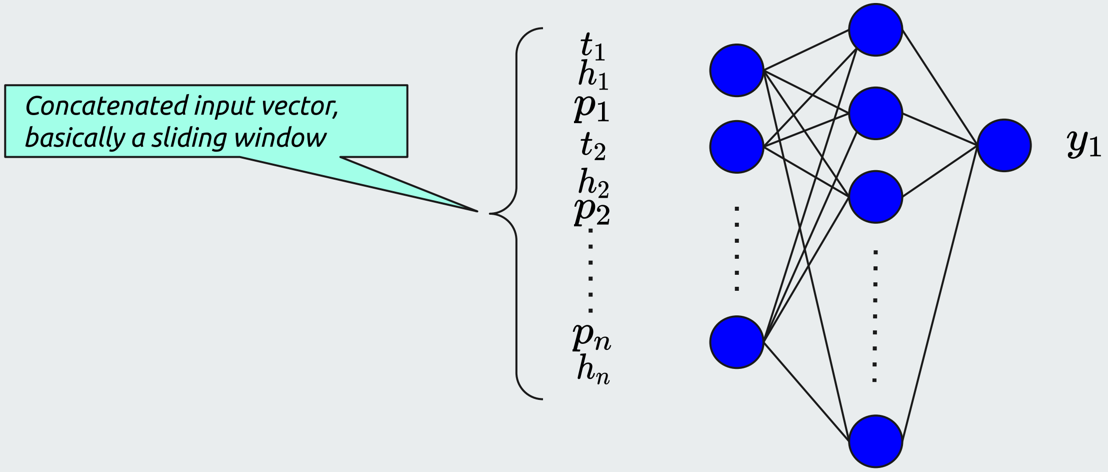
Sometimes the current state is not enough for an optimal decision. $H(Y|X)$ Is the conditional entropy

$$H(Y|X_t) < H(Y|X_t, X_{t-1}, X_{t-2} \ldots) \iff \text{System not markovian}$$

So what do we do when we have a sequence of data?

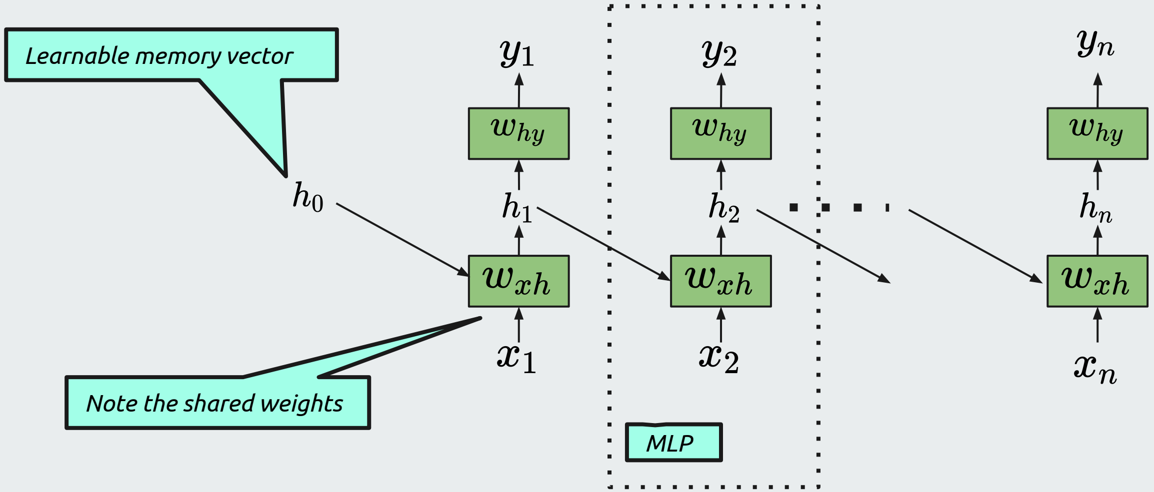


Simple solution: Concatenate everything and feed to MLP.



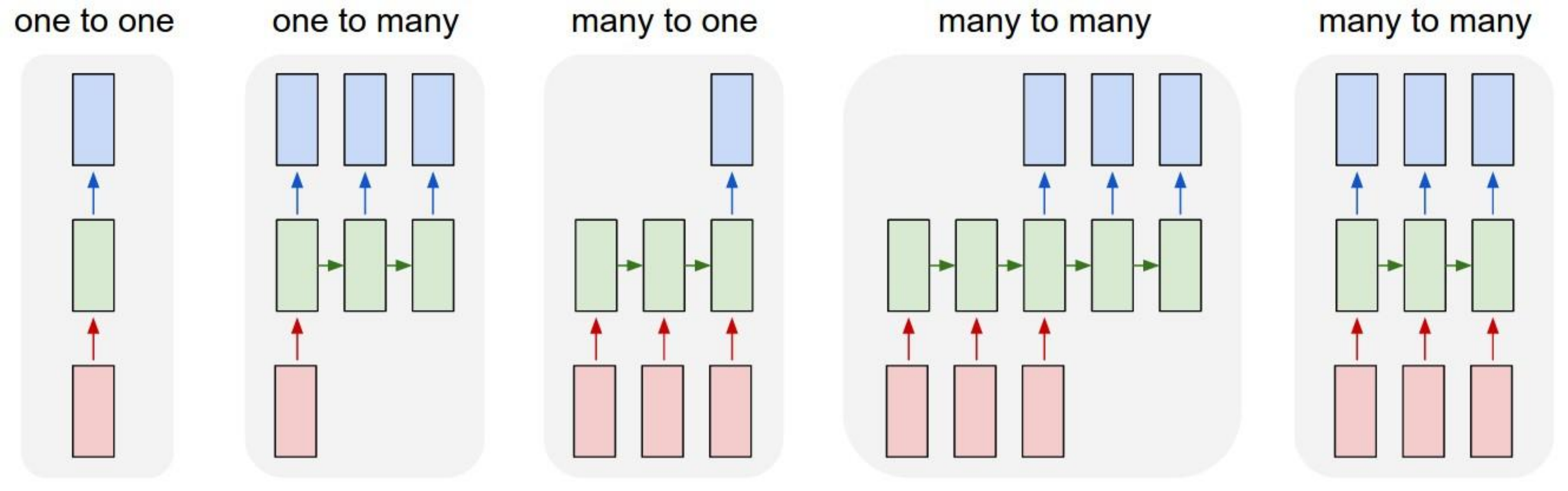
Will it work? Pros / cons?

A better idea is to process data chronologically.



This architecture is called a recurrent neural network (RNN)

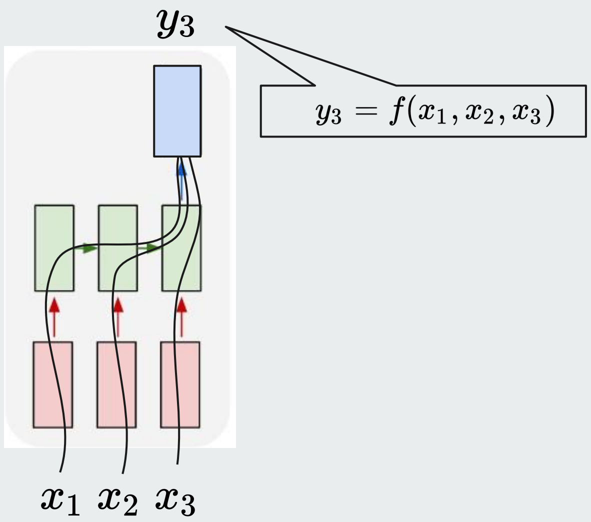
How we can make use of the RNN:



[Source: Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks]

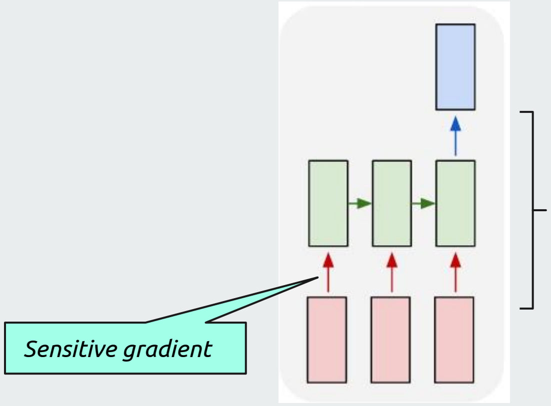
Training an RNN:

- Forward pass: Same as before, but always on the entire sequence.



X_dim = [batchsize, seqlen, n_features]

- Backward pass: Same as before, but with aggregated gradients



One, two, and three gradient calculations on the same weight matrices.

[Image partly taken from: Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks]

Running an RNN in Pytorch

```
import torch as T
import torch.nn as nn

x_dim, y_dim, hid_dim, num_layers = 3, 1, 8, 1

rnn = nn.RNN(input_size=x_dim,hidden_size=hid_dim,
             num_layers=num_layers,batch_first=True)

rnn_cell = nn.RNNCell(input_size=x_dim,
                      hidden_size=hid_dim)

fc = nn.Linear(hid_dim, y_dim)

batchsize = 24
seq_len = 10

# All at once
X = T.randn(batchsize, seq_len, x_dim, requires_grad=True)
H, _ = rnn(X)
Y = fc(H)

# Step by step with Cell
h = None
for i in range(seq_len):
    h = rnn_cell(X[:,i,:], h)
    y = fc(h)
```

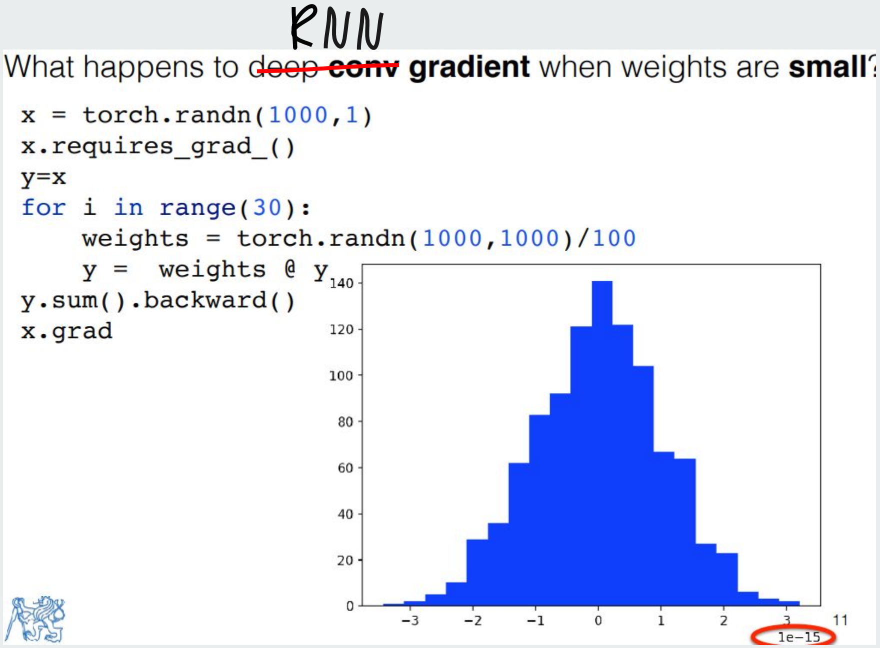
You can specify multiple recurrent layers

Make output out of hidden layer

Works on a sequence

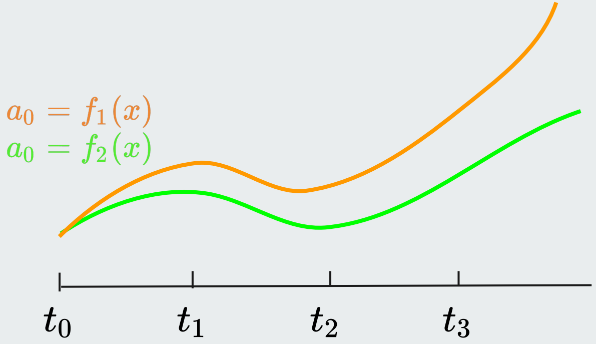
When you don’t know the whole sequence in advance

The repeated multiplication problem in RNNs

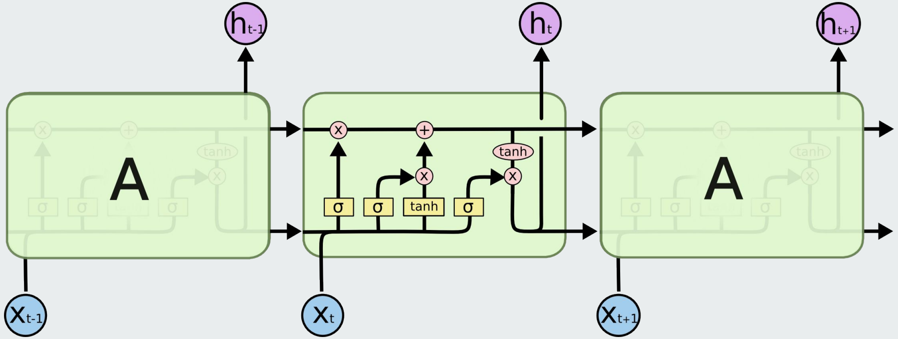


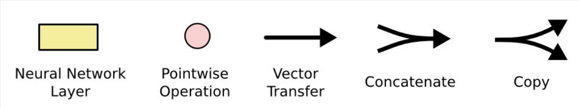
Unedited slide from VIR - Lecture IV by K. Zimmerman

- This affects both forward and backward passes
- In optimal control, this is called sensitivity to initial conditions in the ‘Shooting problem’



Gated Recurrent networks: LSTM (Sepp Horchreiter, Jurgen Schmidhuber)





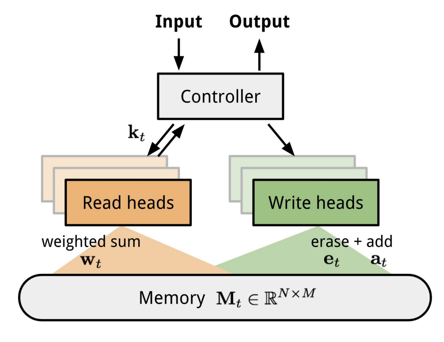
Mechanism functions takeaways

- Decide what to remember
- Decide what to forget
- Decide what to feed through

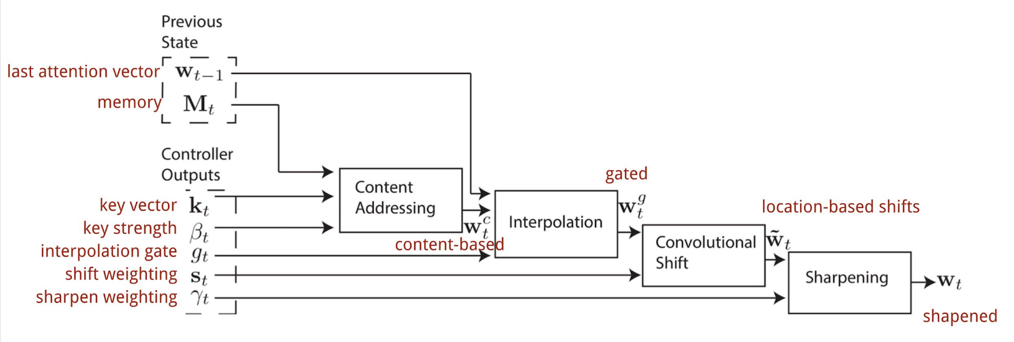
These mechanisms Ameliorate 'Forgetting' issues and gradient problems.
There are also GRU networks which are slightly simpler and do the same thing.

Images taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent network with external memory (Pretty hard to train)



Graves et al, edit: Lil-log (Lillian Weng)



Graves et al, edit: Lil-log (Lillian Weng)

Neural turing machines (Graves et al)

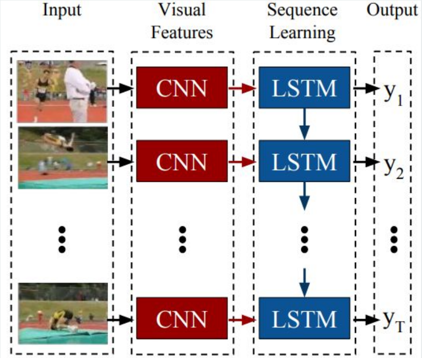
What does Turing have to do with this?

Many other works in this direction:

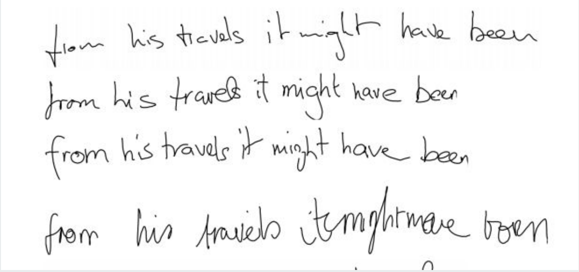
- Neural Slam (Zhang et al.)
- Memory networks (Weston et al.)
- etc..

Examples of successes using RNNs

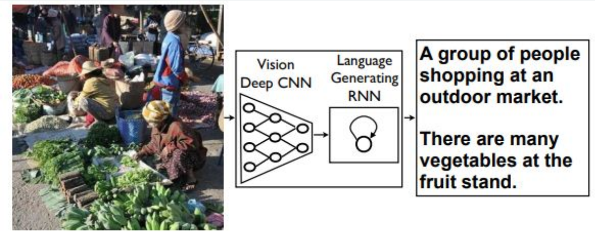
VQA: Visual Question Answering (Agrawal, lu, Antol et al)



Recognition in video (Jeff Donahue et al.)



Generating handwriting sequences with RNN (Graves et al.)

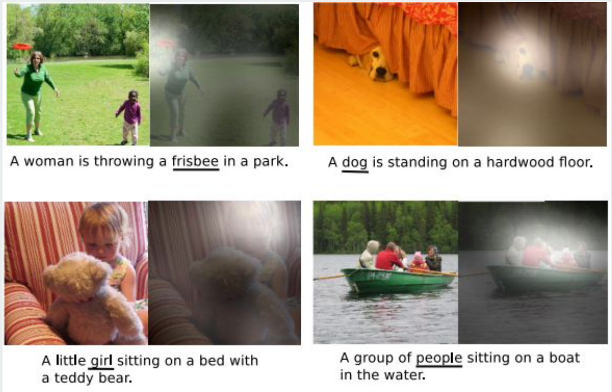


Show and Tell, Image Captioning (Vinyals et al.)

And many more...

Attention mechanisms: Focusing on relevant information

Soft attention in images:



Show, Attend and Tell: Xu, et al.

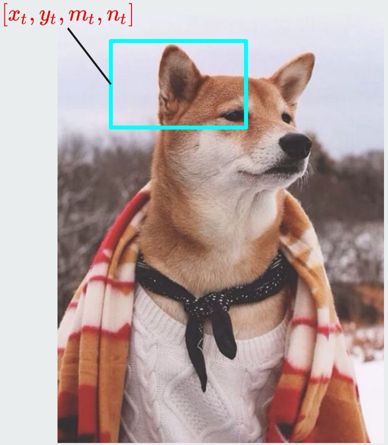
Differentiable mask, convex combination of inputs:

$$y_i^{masked} = a_1 \cdot y_i$$

Outputs weighted by mask coefficient

$$a_1 + a_2 \dots a_n = 1$$

Hard attention in images: Retina model

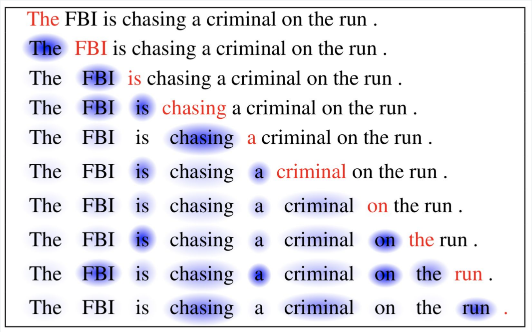


Instagram @mensweardog.

$$y_t = [x_t, y_t, m_t, n_t]$$

$$y_t = f(img, y_{t_1})$$

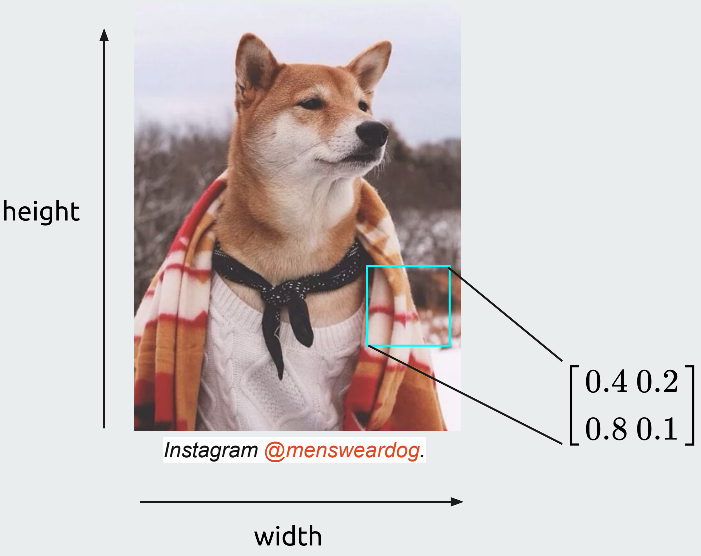
Soft attention in NLP:



Cheng et al., 2016

Tiny break...

Let's take a closer look at Convolution



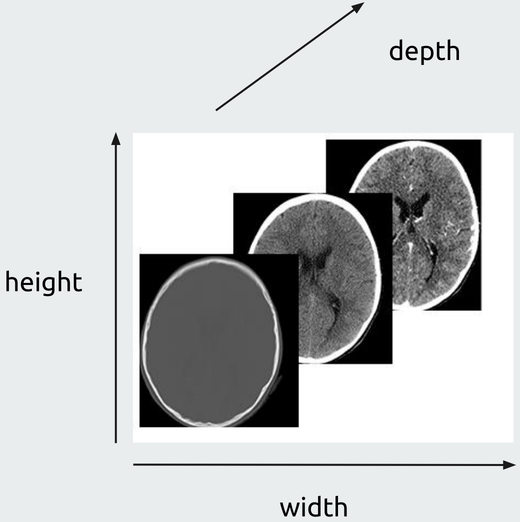
- 2D Images => 2D convolution
- Elementary filter size: [m x n]
- Weight dimension: [m x n x ic x oc]
- Filter is strided in 2 dimensions

Example of a single filter "Stamp" with filter size 2 on 1 input channel and 1 output channel:

$$\overset{c_1}{\begin{bmatrix} 0.2 & 0.5 \\ 0.1 & 0.6 \end{bmatrix}} * \overset{w_1}{\begin{bmatrix} 0.1 & 0.3 \\ 0.3 & 0.4 \end{bmatrix}} + \overset{b}{[1]} = \begin{matrix} 0.2 \cdot 0.1 \\ + 0.5 \cdot 0.3 \\ + 0.1 \cdot 0.3 \\ + 0.6 \cdot 0.4 \\ + 1 = 1.44 \end{matrix}$$

If we have multiple input channels, conv separately and add

We can extend convolution to 3D



- 3D Images (sequences of images) => 3D convolution
- Elementary filter size: [m x n x d]
- Weight dimension: [m x n x d x ic x oc]
- Filter is strided in 3 dimensions

Example of a single filter "Stamp" with filter size 2 on 1 input channel and 1 output channel. Computation is same as if we are doing an image with 2 input channels

$$\underset{c_1}{\begin{matrix} \begin{bmatrix} 0.2 & 0.5 \\ 0.1 & 0.6 \end{bmatrix} \\ \\ \begin{bmatrix} 0.4 & 0.4 \\ 0.7 & 0.1 \end{bmatrix} \end{matrix}} * \underset{w_1}{\begin{matrix} \begin{bmatrix} 0.1 & 0.3 \\ 0.3 & 0.4 \end{bmatrix} \\ \\ \begin{bmatrix} 0.3 & 0.6 \\ 0.4 & 0.6 \end{bmatrix} \end{matrix}} + \underset{b}{[1]} = 2.14$$

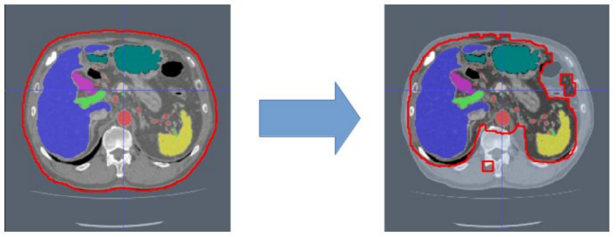
If we have multiple input channels, conv separately and add

Use cases:

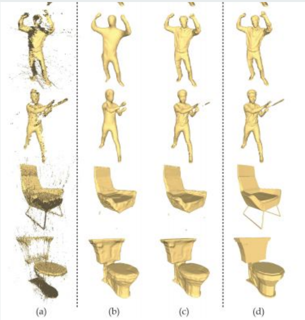
Video processing: Tracking, action recognition ...

3D Convolutional Neural Networks for Human Action Recognition: Shuiwang Ji, Wei Xu, Ming Yang

3D structure processing (3d object reconstruction, medical scans, etc.):



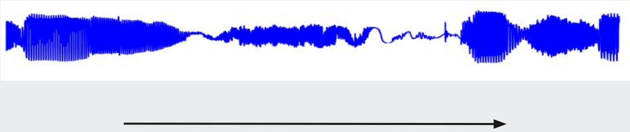
An application of cascaded 3D fully convolutional networks for medical image segmentation, R. Roth et al.



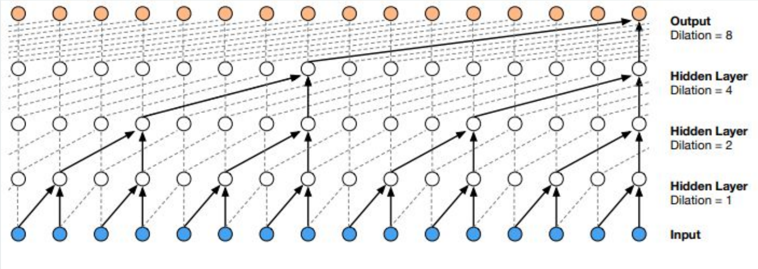
Learning to Reconstruct High-quality 3D Shapes with Cascaded Fully Convolutional Networks, Yan-Pei Cao, Zheng-Ning Liu et al.

Caveat: What is assumed when using 3D conv?

We can also use convolution for 1D sequences. (Actually a viable alternative for RNNs)



length



Wavenet. Van den Oord, etc al. Deepmind.

- 1D Sequences (such as audio) => 1D convolution
- Elementary filter size: [m x l]
- Weight dimension: [m x ic x oc]
- Filter is strided in 1 dimension

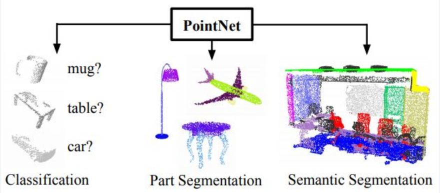
Example of a single filter “Stamp” with filter size 2 on 1 input channel and 1 output channel.

$$\underbrace{[0.3 \quad 0.1]}_{c_1} * \underbrace{[0.7 \quad 0.2]}_{w_1} + \underbrace{[1]}_{b} = 1.23$$

If we have multiple input channels, conv separately and add

People have been exploring how to use neural networks for other data structures

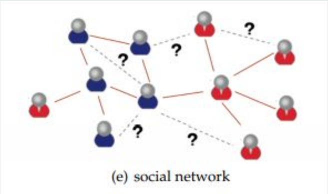
3D point clouds:

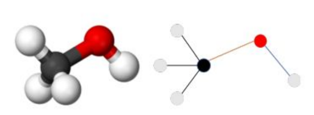


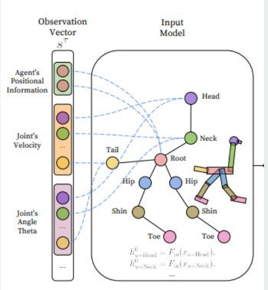
PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. R. Qi, su et al

Graphs:

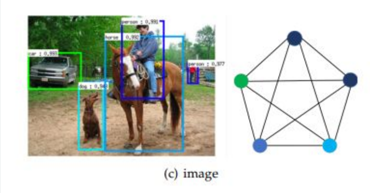
- Social connections
- Molecules
- Articulated robots
- Everything basically …







Nervenet, Wang et al



Graph Neural Networks: A Review of Methods and Applications. Zhou , Cui , Zhang et al.

Fin.
Have fun with your
semestral projects.