Introduction to Deep Learning for Facial and Gesture Understanding

Part V: RNNs

R·I·T

DEEP LEARNING INSTITUTE

www.nvidia.com/dli

Raymond Ptucha,
Rochester Institute of Technology, USA

Tutorial-2
May 14, 2019, 2-6pm

14th IEEE International Conference on Automatic Face and Gesture Recognition
FG2019

Fair Use Agreement

This agreement covers the use of all slides in this document, please read carefully.

- You may freely use these slides, if:
  - You send me an email telling me the conference/venue/company name in advance, and which slides you wish to use.
  - You receive a positive confirmation email back from me.
  - My name (R. Ptucha) appears on each slide you use.

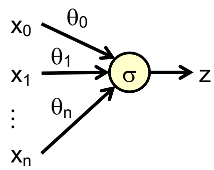
Agenda

- Part I: Introduction
- Part II: Convolutional Neural Nets
- Part III: Fully Convolutional Nets
- Break
- Part IV: Facial Understanding
- **Part V: Recurrent Neural Nets**
- Hands-on with NVIDIA DIGITS

Recurrent Neural Networks

- Feed forward Artificial Neural Networks (ANNs) are great at classification, but are limited at predicting future given the past.
- Need framework that determines output based upon current and previous inputs.
- Recurrent or Recursive Neural Networks (RNNs) capture sequential information and are used in speech recognition, activity recognition, NLP, weather prediction, etc.

Adding Recurrence

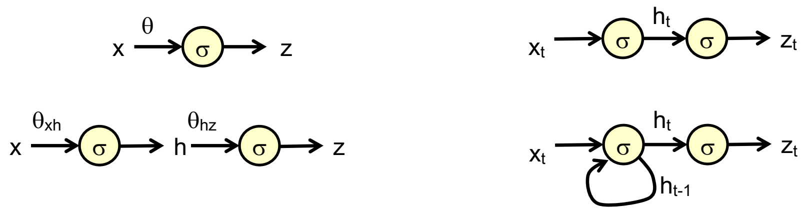


$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

Activation function

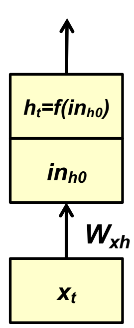
$$z = \sigma(x_0\theta_0 + x_1\theta_1 + \ ... + x_n\theta_n) = \sigma\left(\sum_{i=0}^{n} x_i\theta_i\right)$$

$$z = \sigma(\theta^T x)$$



Neural Networks

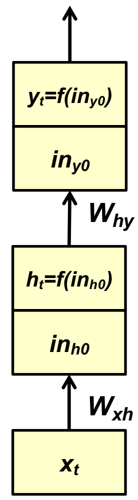
$$in_{h0} = (W_{xh}x_t)$$
$$h_t = f(in_{h0})$$



Where:

- x_t, is the input values
- W_{xh}, is the weight matrix for input
- in_{h0} is the inputs to activation function
- f is some activation function
- h_t is is the output values

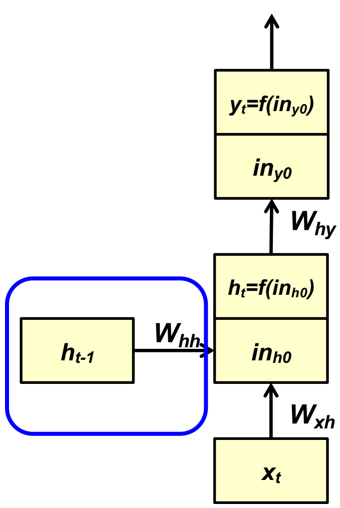
Neural Networks



$$in_{h0} = (W_{xh}x_t)$$
$$h_t = f(in_{h0})$$
$$in_{y0} = W_{hy}h_t$$
$$y_t = f(in_{y0})$$

Where:
- x_t is the input values
- W_{xh}, *is* the weight matrix for input
- in_{h0} is the inputs to activation function
- f is some activation function
- h_t is is the intermediate output values
- W_{hy} is the weight matrix for intermediate value
- y_t is the output values

Recurrent Networks

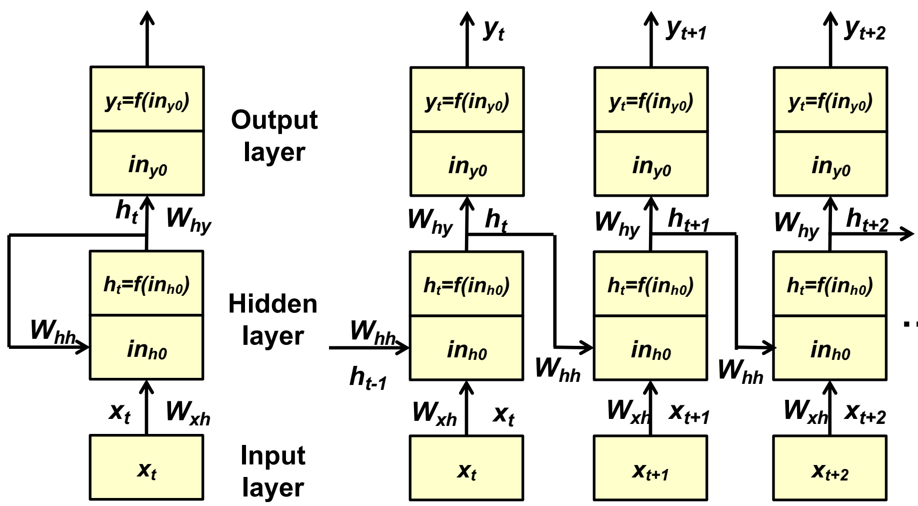


$$in_{h0} = (W_{xh}x_t + W_{hh}h_{t-1})$$
$$h_t = f(in_{h0})$$
$$in_{y0} = W_{hy}h_t$$
$$y_t = f(in_{y0})$$

Where:
- x_t is the input values
- W_{xh}, *is* the weight matrix for input
- in_{h0} is the inputs to activation function
- f is some activation function
- h_t, h_{t-1} are current hidden and previous hidden values
- W_{xh}, W_{hh} and W_{hy} are the weight matrices for input, hidden and output stages respectively
- y_t is the output values

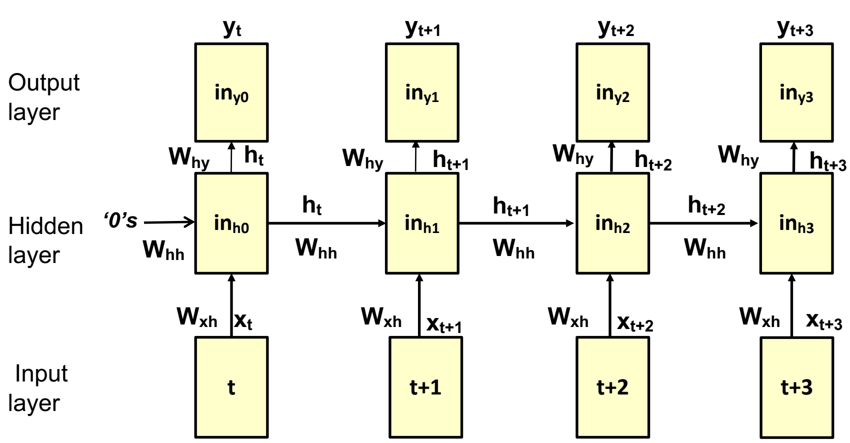
Recurrent Networks

Both figures represent the same architecture



R. Ptucha '19 10

Forward Propagation of Recurrent Networks

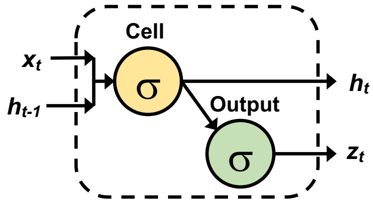


Note: regardless of how many time steps taken, only learning a single W_{xh}, W_{hh}, and W_{hy}. Each are learned via standard back propagation.

R. Ptucha '19 11

Recurrent Networks

Recurrent Neural Network "neuron"



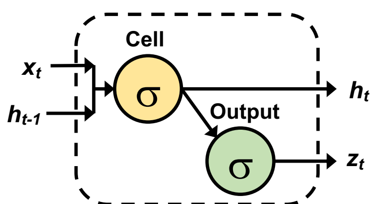
P(next event | previous events)

- Unfortunately, these vanilla RNNs don't always work.
- Can't store info over long periods of time.
- Suffer from vanishing and/or exploding gradients.

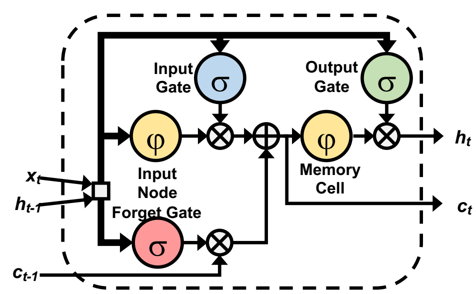
R. Ptucha '19 12

Recurrent Networks

Recurrent Neural Network "neuron"



Long Short Term Memory "neuron"

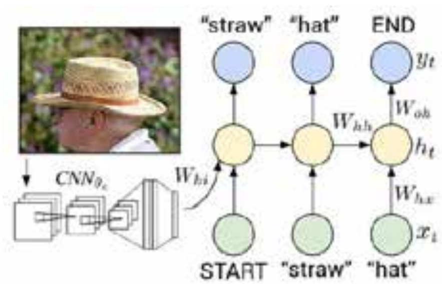


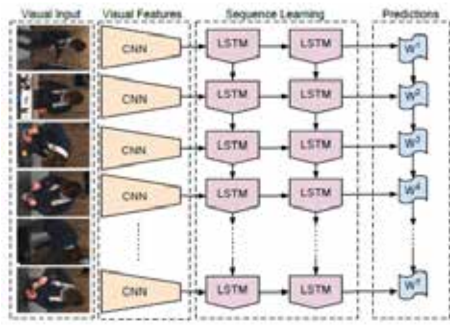
Donahue et al., 2015

- LSTM's allow read/write/reset functions to neurons.
- Remember past to predict the future- (over long time periods).
- Can have many hidden neurons per layer and many layers.

R. Ptucha '19 13

Recurrent Applications

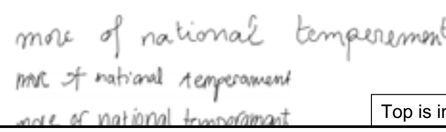




Karpathy, Fei-Fei, 2015

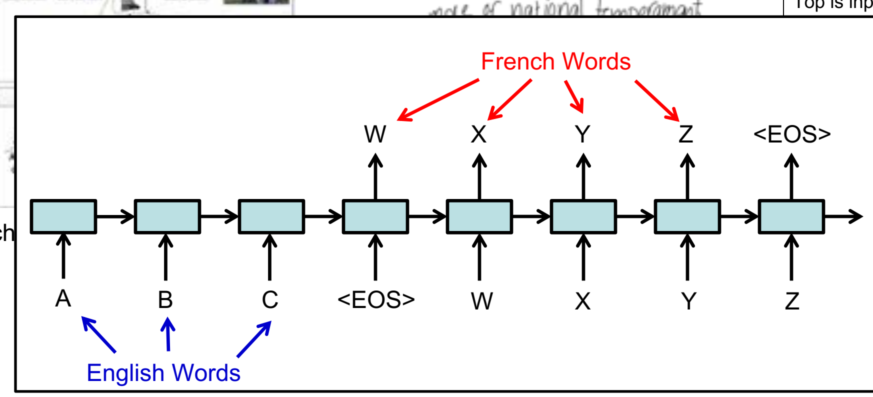
Donahue, et al., 2015

Recurrent Applications



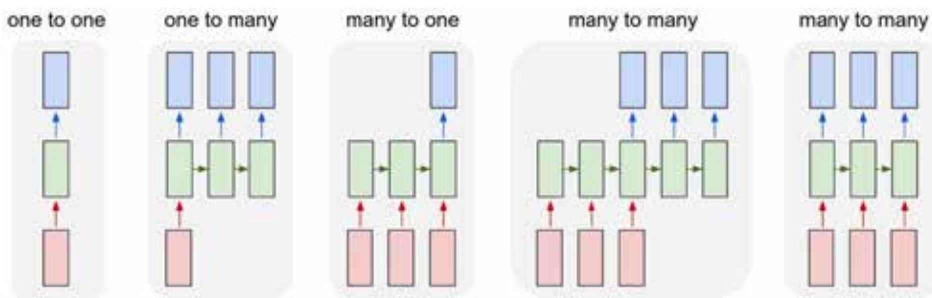
Top is input,

Soch



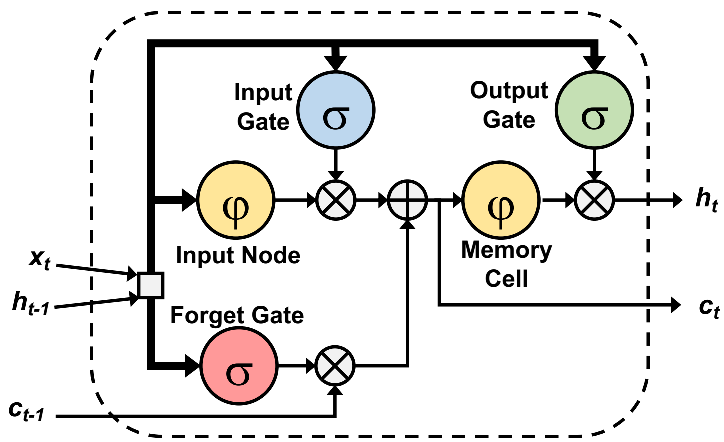
Sutskever et al., 2014

Many Flavors



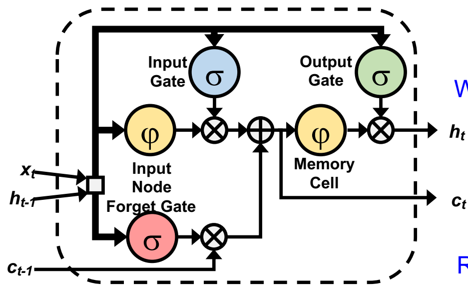
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LSTMs



LSTMs

Convert standard neuron into a complex memory cell



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Write, read, reset governors:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1})$

Real input to memory cell:

Input node: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

Looks just like our RNN cell!

Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

$$c_t = (f_t c_{t-1} + i_t g_t)$$

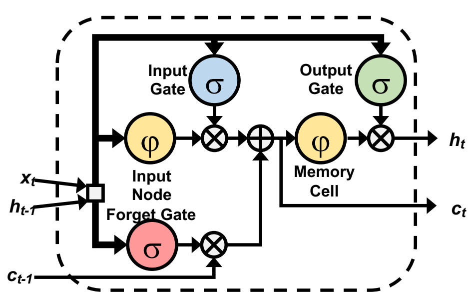
Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

R. Ptucha '19

26

The input node summarizes the input and past output, which will be governed by the input gate.



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1})$

Input node: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

$$c_t = (f_t c_{t-1} + i_t g_t)$$

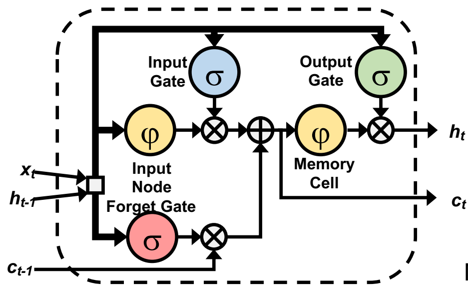
Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

R. Ptucha '19

27

Write: The input gate gives the provision to determine importance of current input and past hidden state.



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1})$

Modulation gate: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

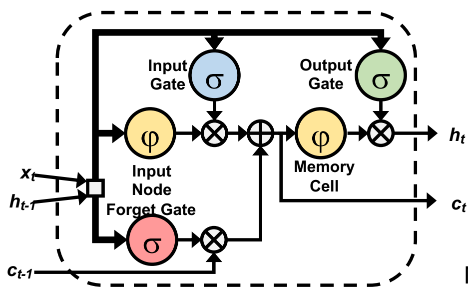
$$c_t = (f_t c_{t-1} + i_t g_t)$$

Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

R. Ptucha '19

Read: The output gate determines what parts of the cell output are necessary for the next time step.



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1})$

Modulation gate: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

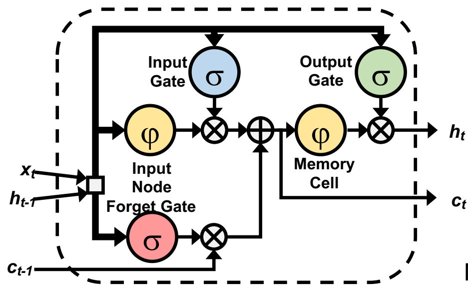
$$c_t = (f_t c_{t-1} + i_t g_t)$$

Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

R. Ptucha '19

Reset: The forget gate gives the provision for the hidden layer to discard or forget the historical data



With $\sigma()$=sigmoid activation function and $\phi()$=tanh activation function, x_t and the previous cell output h_{t-1} calculate:

Input gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1})$

Output gate: $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1})$

Forget gate: $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1})$

Modulation gate: $g_t = \phi(W_{xc}x_t + W_{hc}h_{t-1})$

Calculate a memory cell which is the summation of the previous memory cell, governed by the forget gate and the input and previous output governed by independent combinations of the same:

$$c_t = (f_t c_{t-1} + i_t g_t)$$

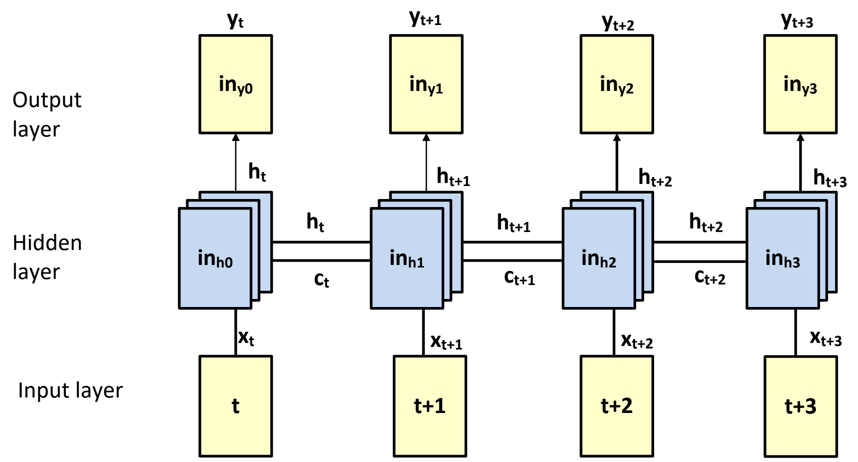
Calculate a new hidden state, governed by the output gate:

$$h_t = o_t \phi(c_t)$$

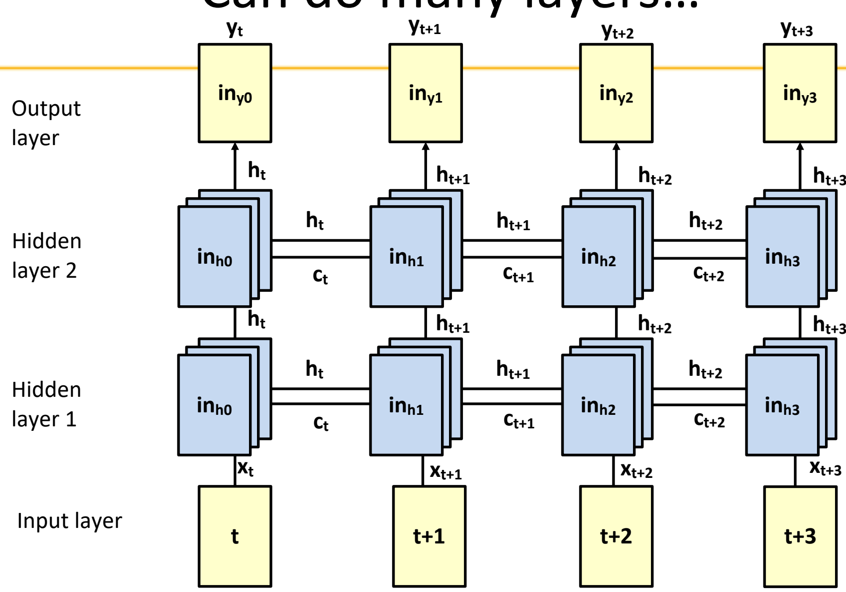
Using LSTMs

- The LSTM memory cells are analogous to a single neuron.
- As such many hundreds of these memory cells are used in a layer, each of which passes its output h_t to the next time step, h_{t+1}.

Same architecture as RNNs, but middle neurons are now LSTM memory cells



Can do many layers…



Learning Shakespeare

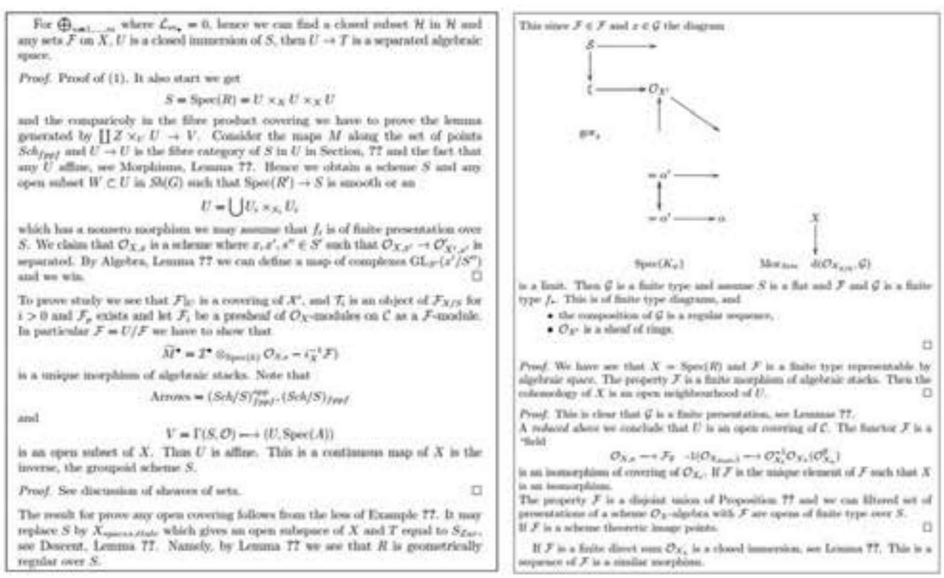
- LSTMs can learn structure and style in the data.
- Karparthy downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file.
- Train a 3-layer LSTM with 512 hidden nodes on each layer.
- After we train the network for a few hours Karpathy obtained samples such as:


```
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.
```

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

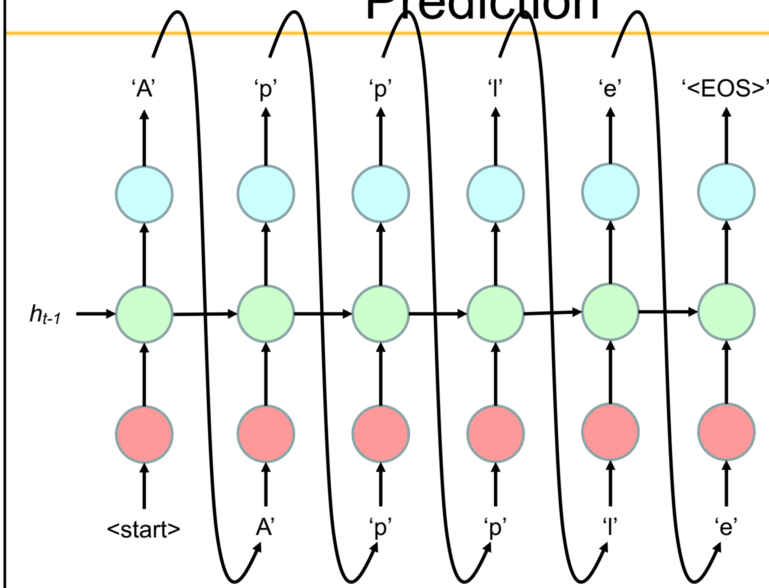
Learning LaTeX

- The results above suggest that the model is actually quite good at learning complex syntactic structures.
- Karpathy and Johnston downloaded the raw Latex source file (a 16MB file) of a book on algebraic stacks/geometry and trained a multilayer LSTM.
- Amazingly, the resulting sampled LaTex *almost* compiled.
- They had to step in and fix a few issues manually but then they get plausible looking math:



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

Recurrent Networks for Character Prediction

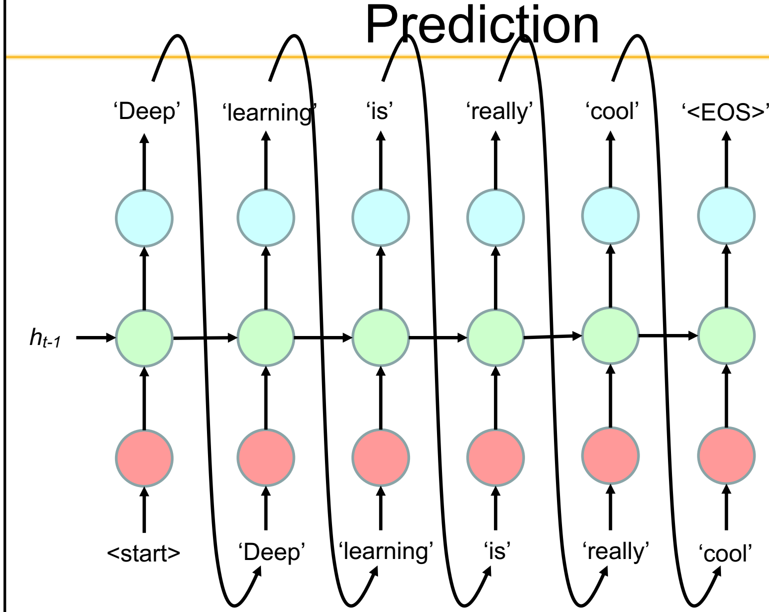


For this to work, we need to represent characters as some latent vector numerical representation.

R. Ptucha ‘19

42

Recurrent Networks for Word Prediction



For this to work, we need to represent words as some latent vector numerical representation.

R. Ptucha ‘19

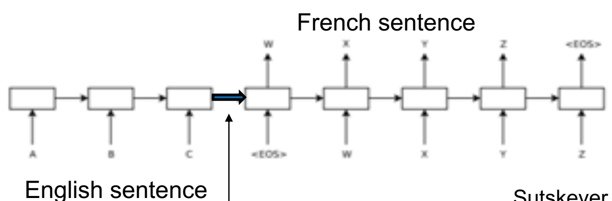
43

Word2vec

- In the simplest form, we can start with a one-hot encoded vector of all words, and then learn a model which converts to a lower dimensional representation.
- Word2vec, glove, and skip-gram are popular metrics which encode words to a latent vector representation (~300 dimensions).
- Now we have a way to represent images, characters, and words as vectors.

Sent2vec

- In the English to French translation, we have:



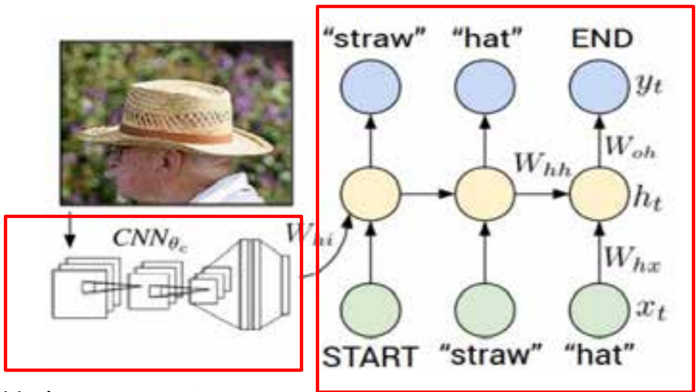
Sutskever et al., NIPS '14

…but wait, this point in the RNN is a representation (sent2vec) of all the words in the English sentence!

- Now we have a way to represent images, characters, words, and sentences as vectors…can extend to paragraphs and documents…

Image Captioning

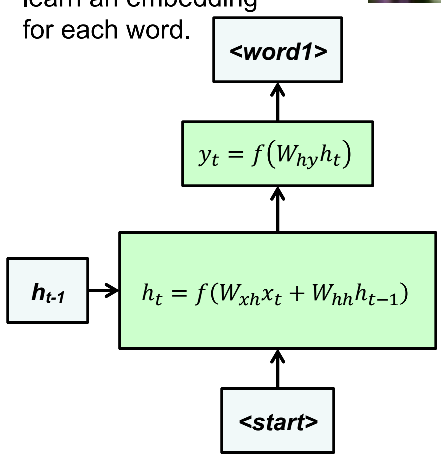
RNN takes in a latent representation of an image, and generates a sequence.



CNN helps represent an image as a numeric value. (image2vec)

Karpathy & Li, CVPR'15

- We may have 50K words. Instead of one-hot encoding, we learn an embedding for each word.



- Glove embedding (300 long vector/word) is very popular.
- Alternately, can learn embedding- learn a matrix which goes from (50K) one-hot to 300, ie: $W_{ix} \in R^{50K \times 300}$
- Embedding and unembedding can be learned or inverses of one another.

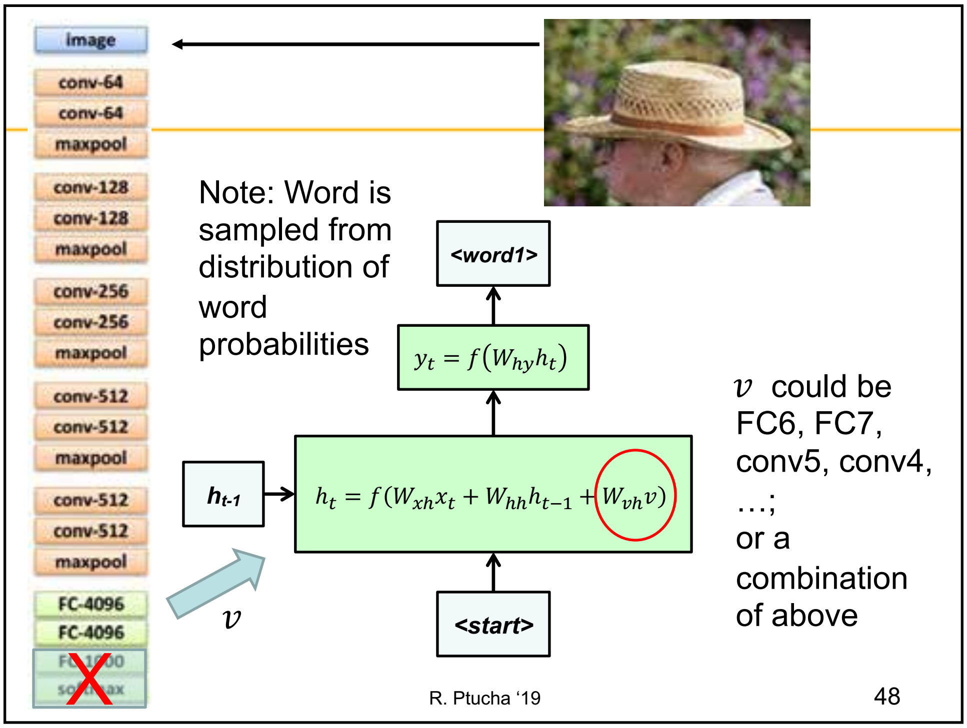


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
Note: Word is sampled from distribution of word probabilities
<word1>
$y_t = f(W_{hy}h_t)$
h_{t-1}
$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + W_{vh}v)$
v
<start>
v could be FC6, FC7, conv5, conv4, …; or a combination of above
R. Ptucha '19
48

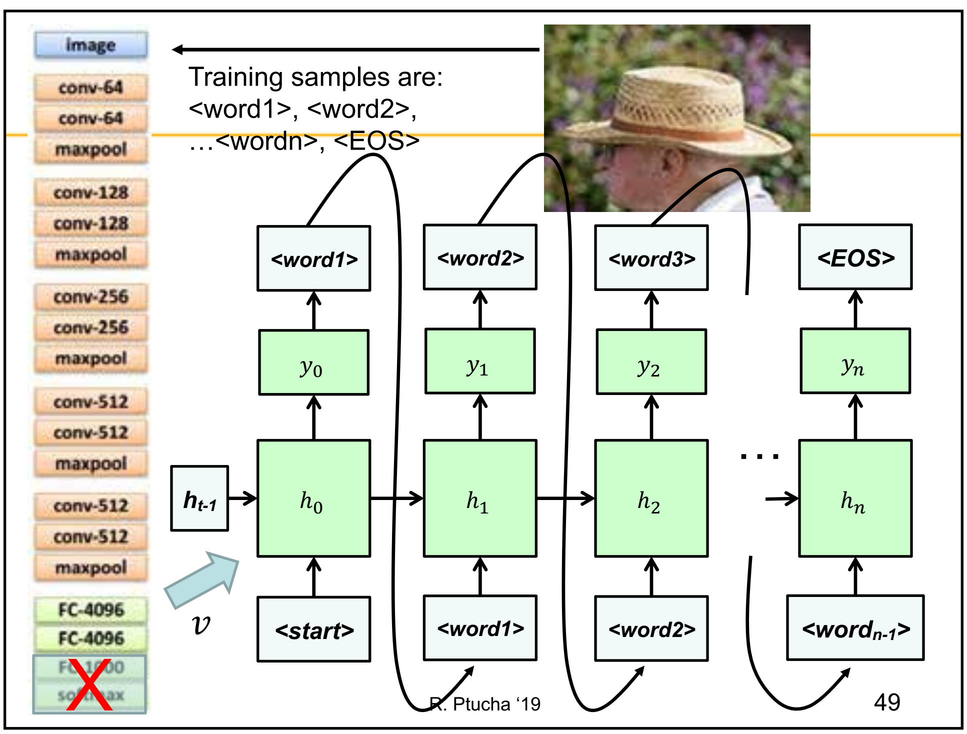


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
Training samples are:
<word1>, <word2>,
…<wordn>, <EOS>
<word1>
<word2>
<word3>
<EOS>
y_0
y_1
y_2
y_n
h_{t-1}
h_0
h_1
h_2
…
h_n
v
<start>
<word1>
<word2>
<word$_{n-1}$>
R. Ptucha '19
49

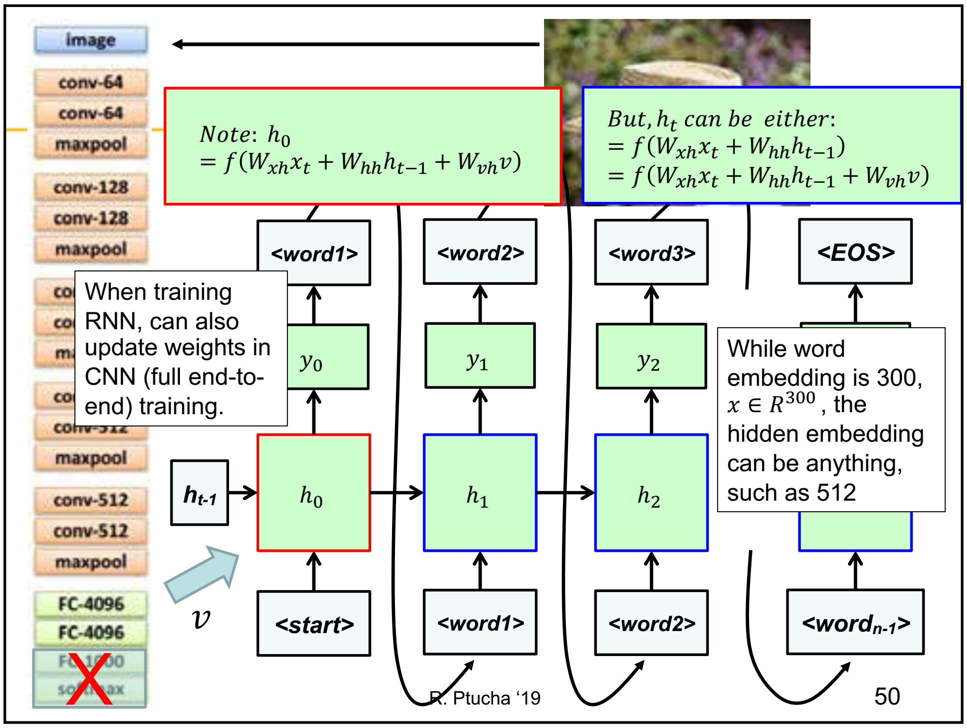
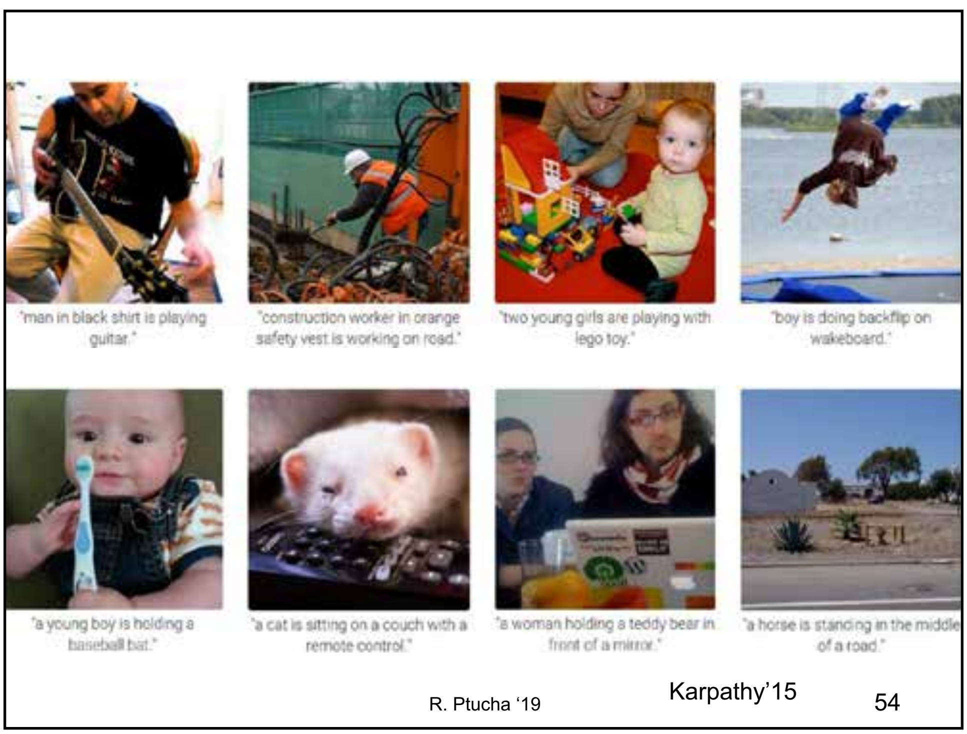
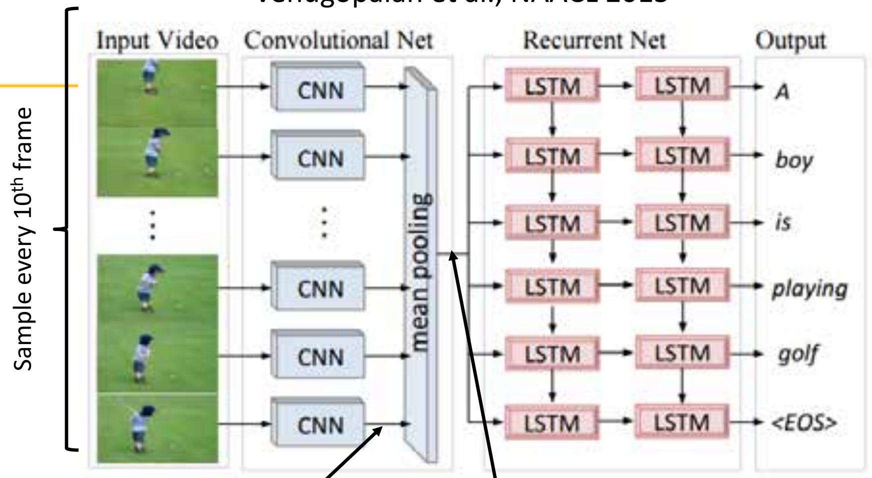


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
Note: $h_0 = f(W_{xh}x_t + W_{hh}h_{t-1} + W_{vh}v)$
But, h_t can be either: $= f(W_{xh}x_t + W_{hh}h_{t-1})$ $= f(W_{xh}x_t + W_{hh}h_{t-1} + W_{vh}v)$
When training RNN, can also update weights in CNN (full end-to-end) training.
While word embedding is 300, $x \in R^{300}$, the hidden embedding can be anything, such as 512
<word1>
<word2>
<word3>
<EOS>
y_0
y_1
y_2
h_{t-1}
h_0
h_1
h_2
v
<start>
<word1>
<word2>
<word$_{n-1}$>
R. Ptucha '19
50


"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."
"boy is doing backflip on wakeboard."
"a young boy is holding a baseball bat."
"a cat is sitting on a couch with a remote control."
"a woman holding a teddy bear in front of a mirror."
"a horse is standing in the middle of a road."
R. Ptucha '19
Karpathy'15
54

Venugopalan et al., NAACL 2015

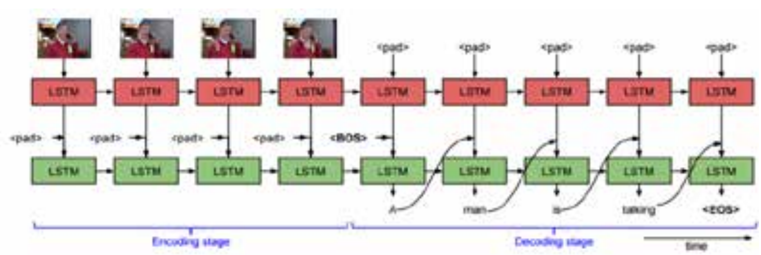


AlexNet 4K FC7

Mean pooling over f frames $V = \frac{1}{f}\sum_{i=1}^{f}[v_1^i, v_2^i, \cdots, v_{4096}^i]$

Pre-train on alternate caption datasets, fine tune to your dataset

Video Captioning



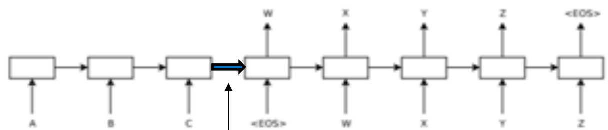
SV2T, Venugopalan, 2015

- Single LSTM for both encode and decode state.
- Two layer LSTM, 1000 hidden units each:
  - First LSTM learns video concepts
  - Second LSTM concentrates on language details.

Video2vec

- We can generically use the same seq2seq operation for video:

Output caption



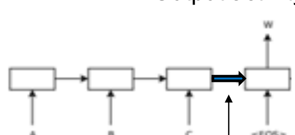
CNN encoding frame by frame

…this point in the RNN is a representation (video2vec) of all the frames in the video!

Video2vec

- We can generically use the same seq2seq operation for video:

Output activity/action



CNN encoding frame by frame

…this point in the RNN is a representation (video2vec) of all the frames in the video!

C3D

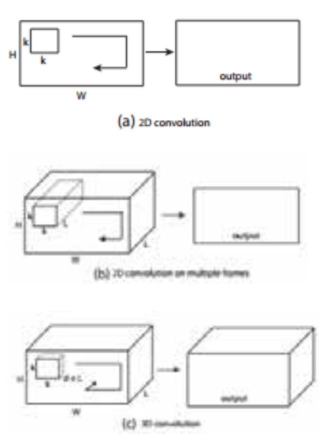
Tran et al. "Learning Spatiotemporal Features with 3D Convolutional Networks", ICCV 2015.

- Rather than learn a single vector (e.g. FC7), introduced a spatio-temporal video feature representation using deep 3D ConvNets.
- Not the first to propose 3D ConvNets, but first to exploit deep nets with large supervised datasets.
- Models appearance and motion.
- Showed that:
  - 3D ConvNets are better than 2D ConvNets
  - Simple architecture with 3×3×3 filters works very well
  - Learned features are then passed into simple linear classifier to give state-of-the-art results

2D and 3D Convolution

(will still work with *c* channels and *f* frames)

(Similar phenomenon for pooling)



Tran et al., 2015

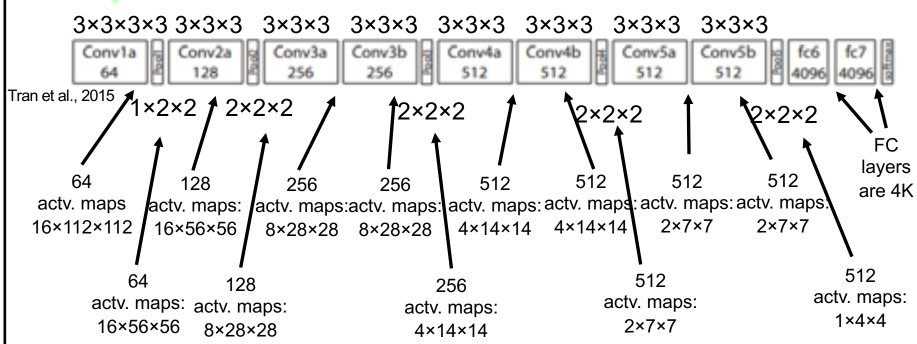
- 2D conv on a 2D image results in 2D image
- 2D conv on a 3D volume results in 2D image
  - Because filter depth matches volume depth.
- 3D conv on a 3D volume results in 3D volume
  - Preserves spatio-temporal information.

input
3×16×128×171
augmentation
3×16×112×112

Full C3D Architecture (Tran et al. ICCV'15)

Split video into 16-frame clips with 8-frame overlap

All convolution uses zero pad and stride=1



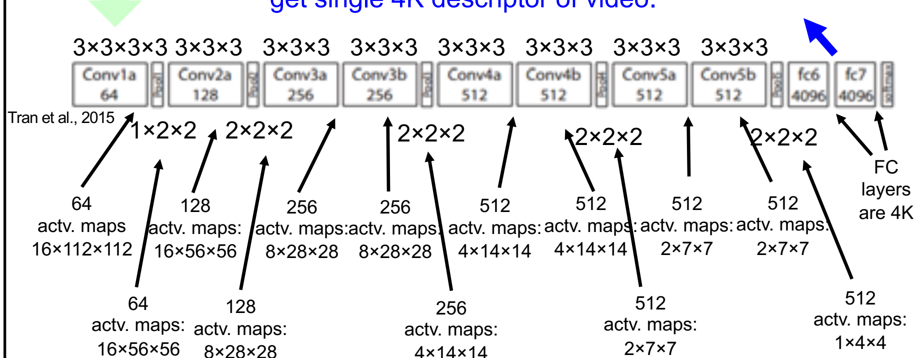
R. Ptucha '19 62

input
3×16×128×171
augmentation
3×16×112×112

Full C3D Architecture (Tran et al. ICCV'15)

Split video into 8-frame overlap 16-frame clips

When used as a video2vec feature descriptor, take output from all fc6 layers of all clips, and average to get single 4K descriptor of video.



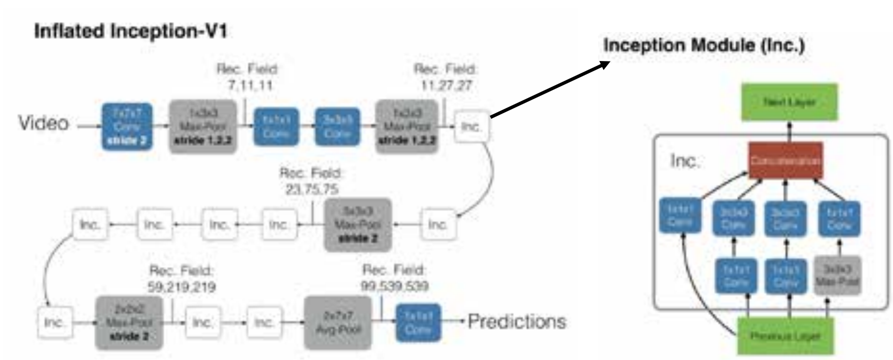
Pu et al., 2017:

The dimensions for features extracted from *pool2*, *pool3*, *pool4* and *pool5* are $28 \times 28 \times N/2 \times 128$, $14 \times 14 \times N/4 \times 256$, $7 \times 7 \times N/8 \times 512$ and $4 \times 4 \times N/16 \times 512$, respectively. N is the number of frames of input video. After the convolutional transformation, the dimensions will be all $4 \times 4 \times N/16 \times 512$.

R. Ptucha '19 63

Inflated Inception v1 for Video (I3D)

Filters and Pooling Increased from 2D to 3D



Same group who introduced VGGFace2 at FG'18

Quo Vadis Action Recognition: a New Model and the Kinetics Dataset. Carreira and Zisserman, CVPR 2017, http://openaccess.thecvf.com/content_cvpr_2017/papers/Carreira_Quo_Vadis_Action_CVPR_2017_paper.pdf

Thank you!!

Ray Ptucha
rwpeec@rit.edu

https://www.rit.edu/mil