CENG501 – Deep Learning

Week 5

Spring 2026

Sinan Kalkan

Dept. of Computer Engineering, METU

Types of Convolution: Dilated (Atrous) Convolution

Purpose: Increase effective receptive field size without increasing parameters.

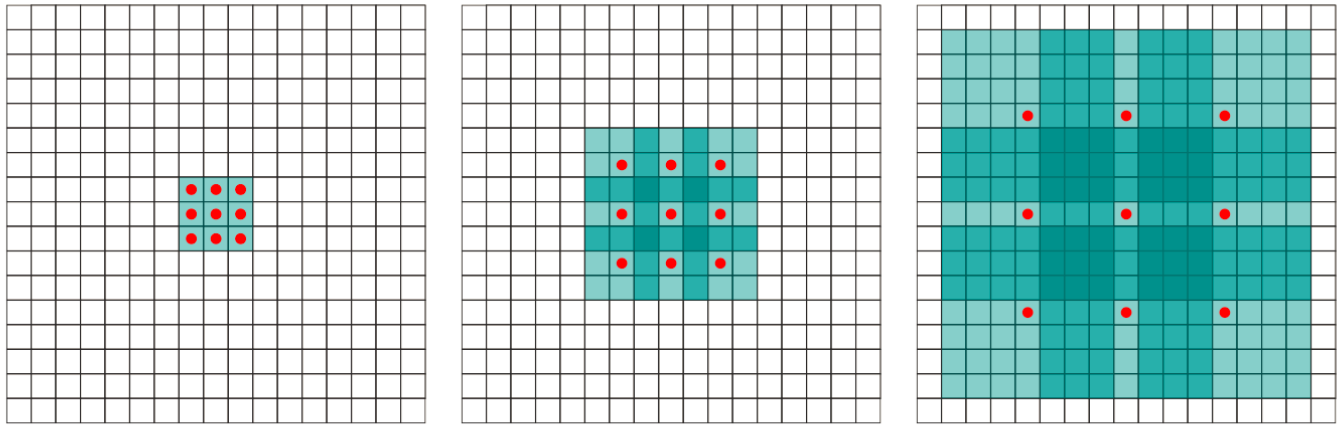


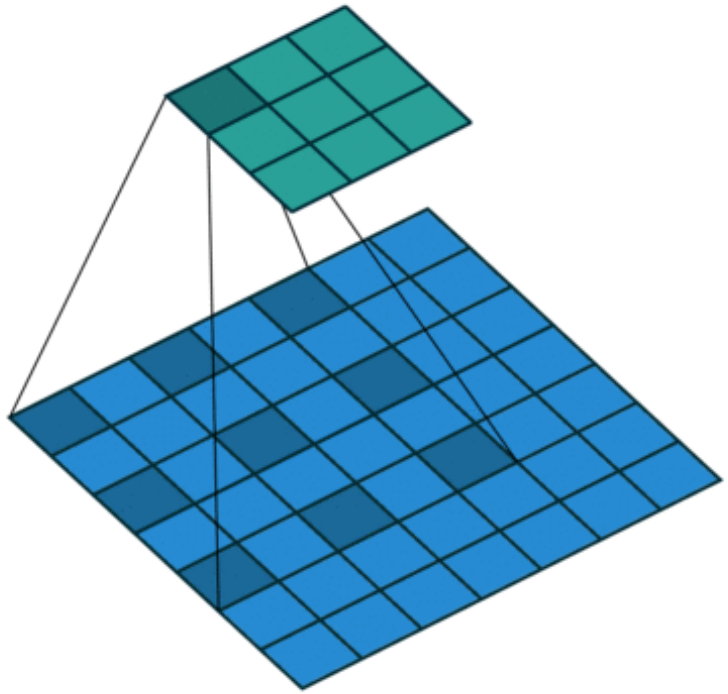
Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

Published as a conference paper at ICLR 2016

MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS

Fisher Yu
Princeton University

Vladlen Koltun
Intel Labs



https://github.com/vdumoulin/conv_arithmetic

Previously on CENG501

Types of Convolution: Transposed Convolution

Purpose: Increasing layer width+height (upsampling).

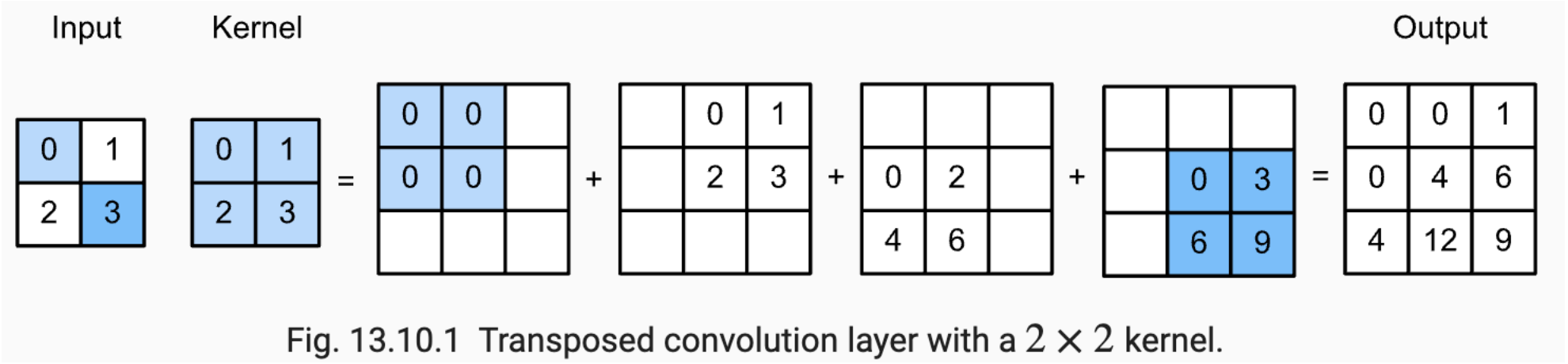


Fig. 13.10.1 Transposed convolution layer with a 2×2 kernel.

Figure: https://d2l.ai/chapter_computer-vision/transposed-conv.html

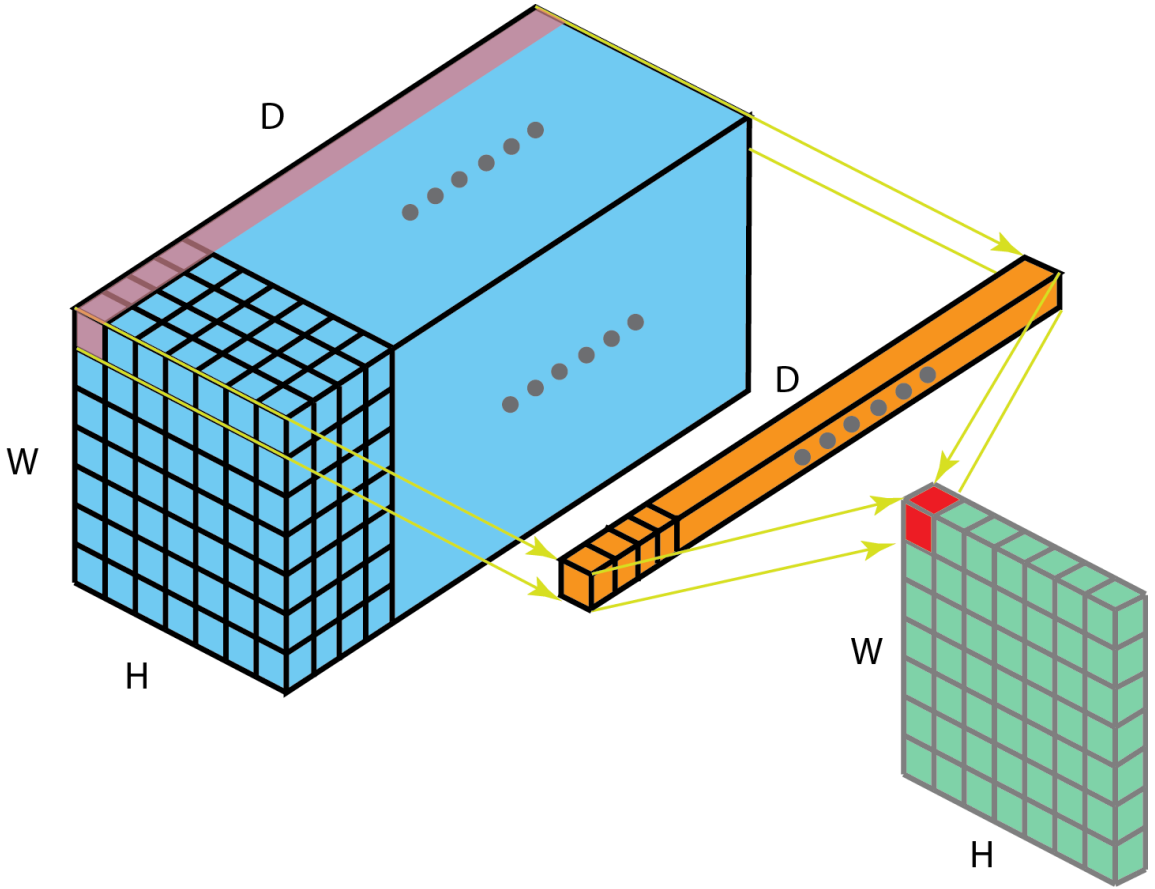
The size of the output:

- Regular convolution: $O = \frac{W-F+2\times P}{S} + 1$
- Transpose convolution: $W = (O-1) \times S + F - 2 \times P$

Types of Convolution: 1x1 Convolution

Previously on CENG501

Purpose: Reduce number of channels.



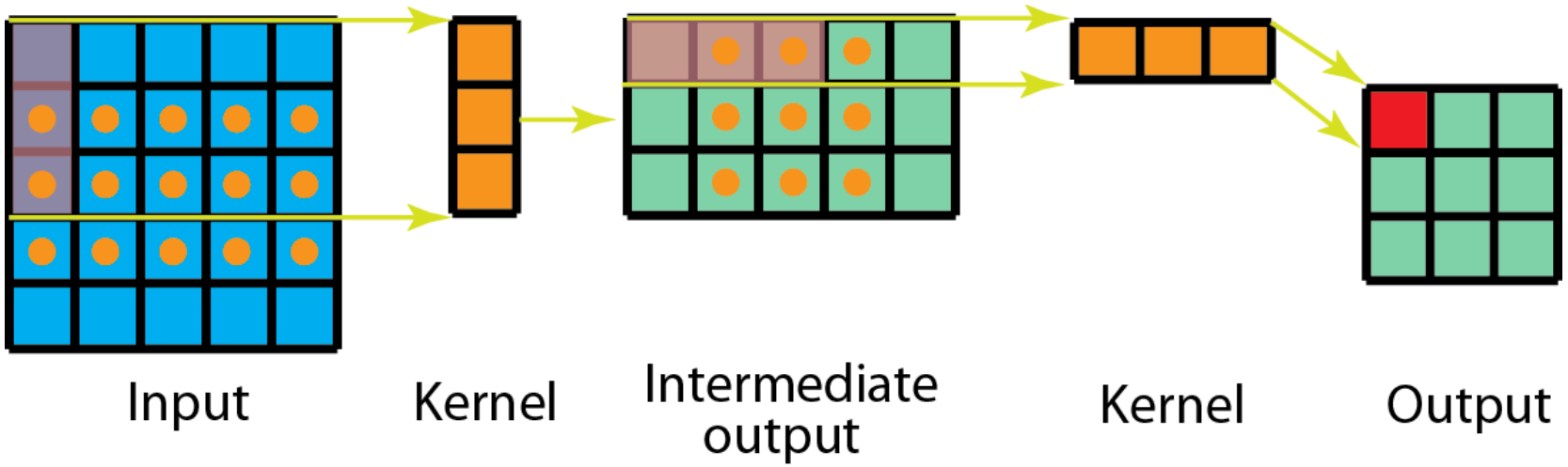
https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Separable Convolution

Previously on CENG501

Purpose: Reduce number of parameters and multiplications.

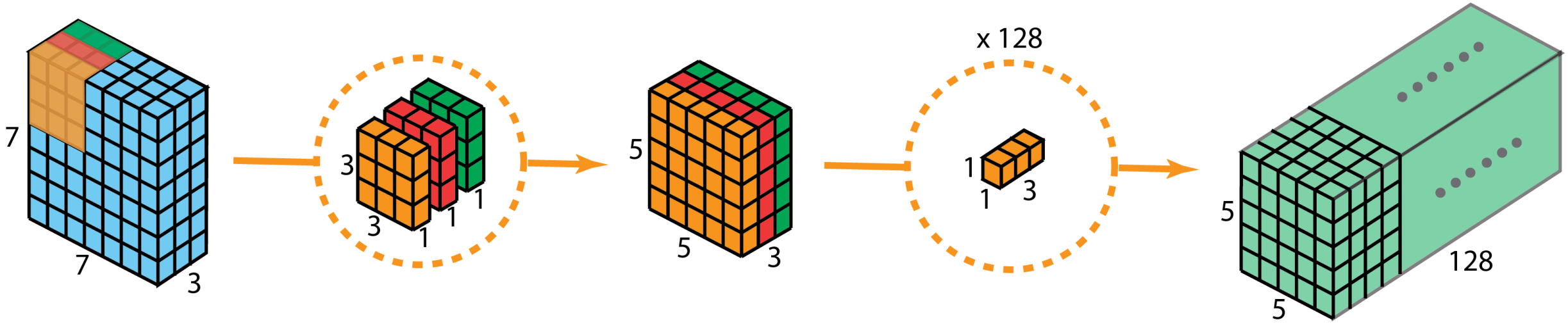
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Depth-wise Separable Convolution

Purpose: Reduce number of parameters and multiplications.



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Deformable Convolution

Previously on CENG501

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., & Wei, Y. (2017). Deformable convolutional networks. ICCV.

Purpose: Flexible receptive field.

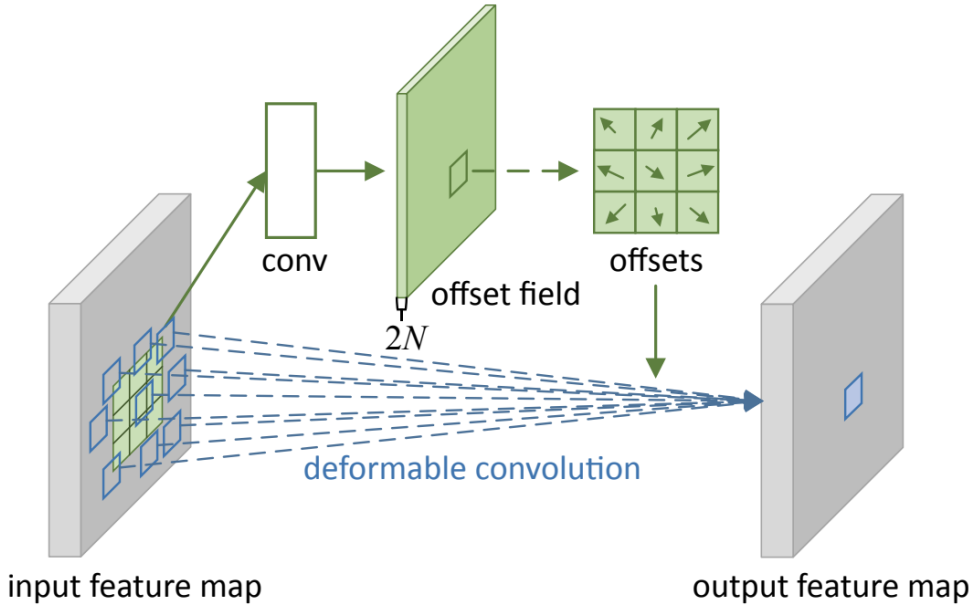


Figure 2: Illustration of 3×3 deformable convolution.

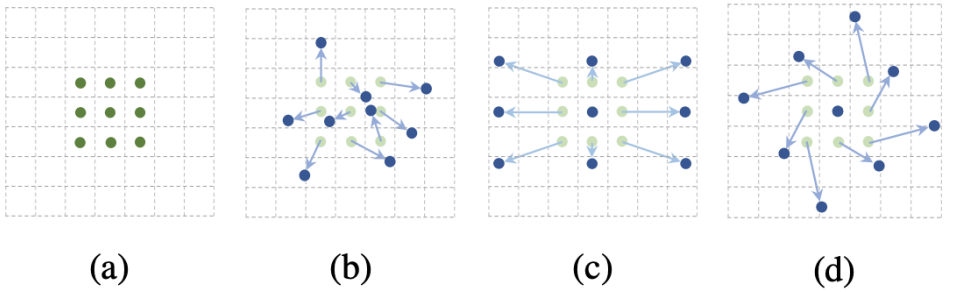
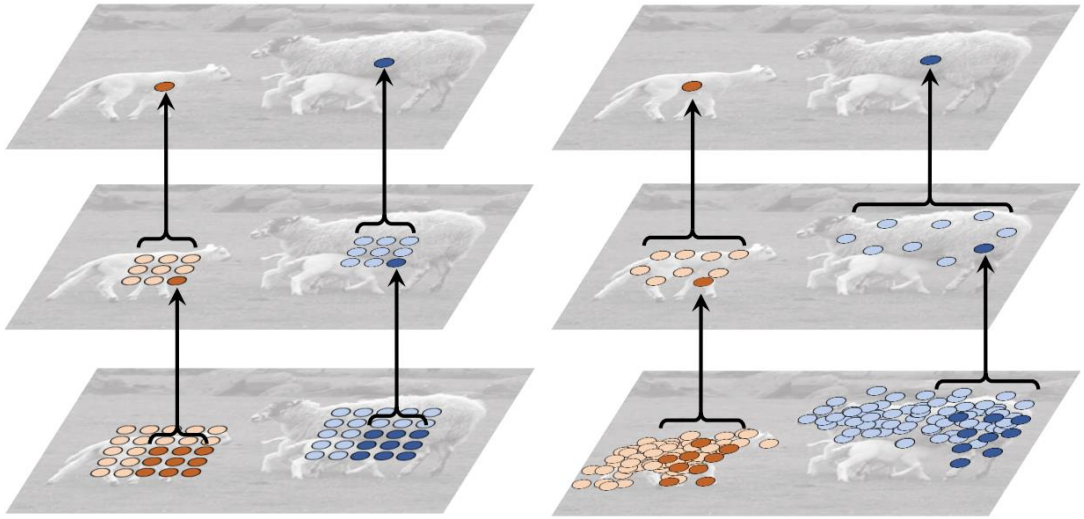


Figure 1: Illustration of the sampling locations in 3×3 standard and deformable convolutions. (a) regular sampling grid (green points) of standard convolution. (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution. (c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.



(a) standard convolution (b) deformable convolution

Pooling

Previously on CENG501

- Example
  - Pooling layer with filters of size 2x2
  - With stride = 2
  - Discards 75% of the activations
  - Depth dimension remains unchanged
- Max pooling with F=3, S=2 or F=2, S=2 are quite common.
  - Pooling with bigger receptive field sizes can be destructive
- Avg pooling is an obsolete choice. Max pooling is shown to work better in practice.

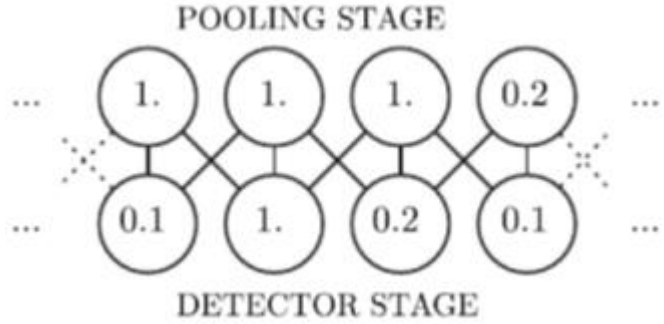
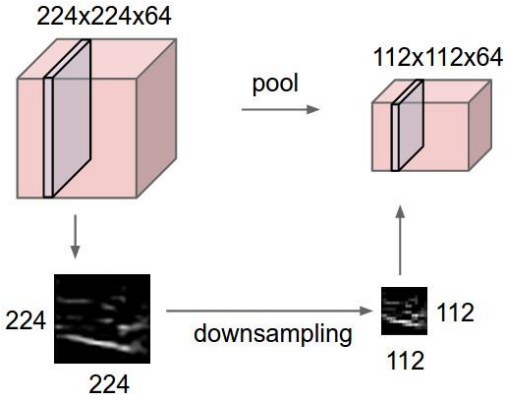
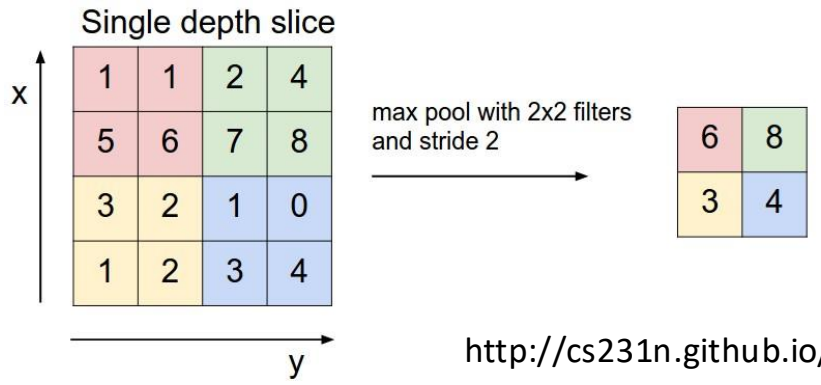


Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

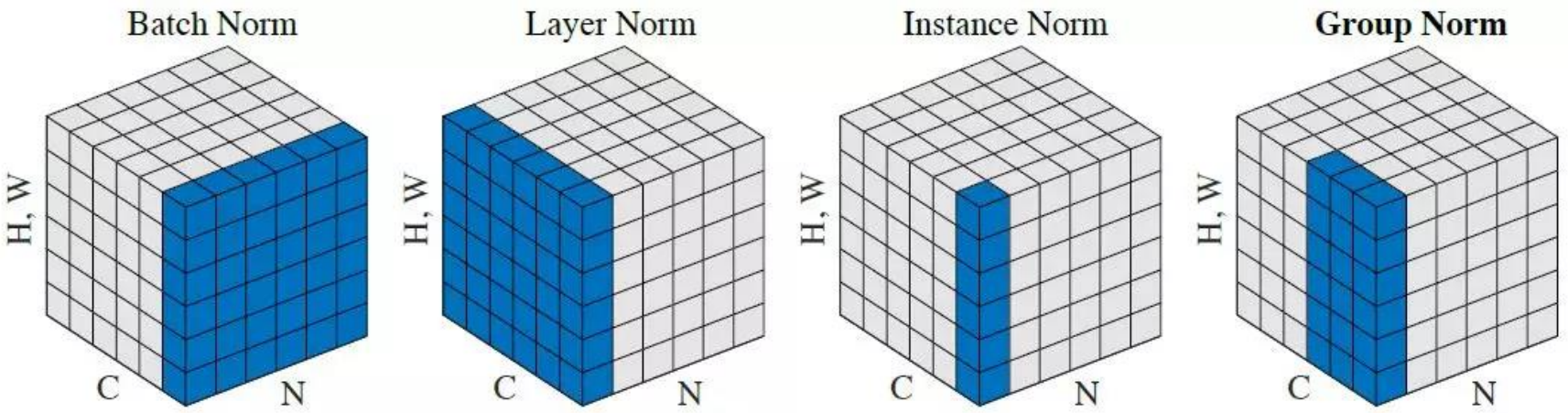




http://cs231n.github.io/convolutional-networks/

Normalization

Previously on CENG501



mean-std for each channel, for each neuron, over the samples

mean-std for each neuron over all channels.

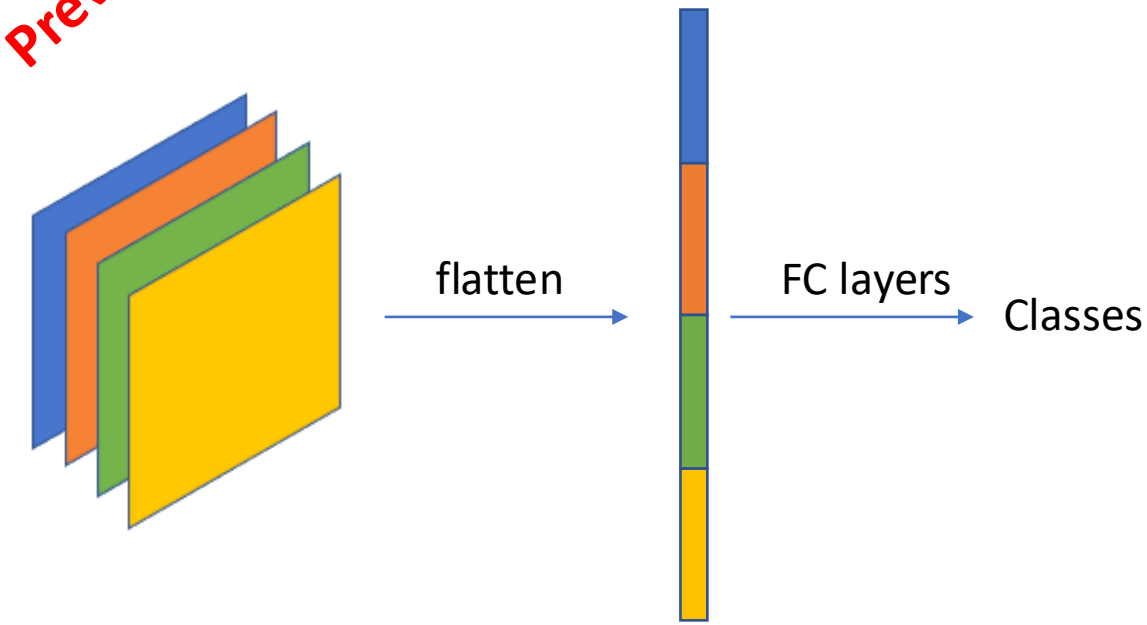
mean-std for each channel over all neurons.

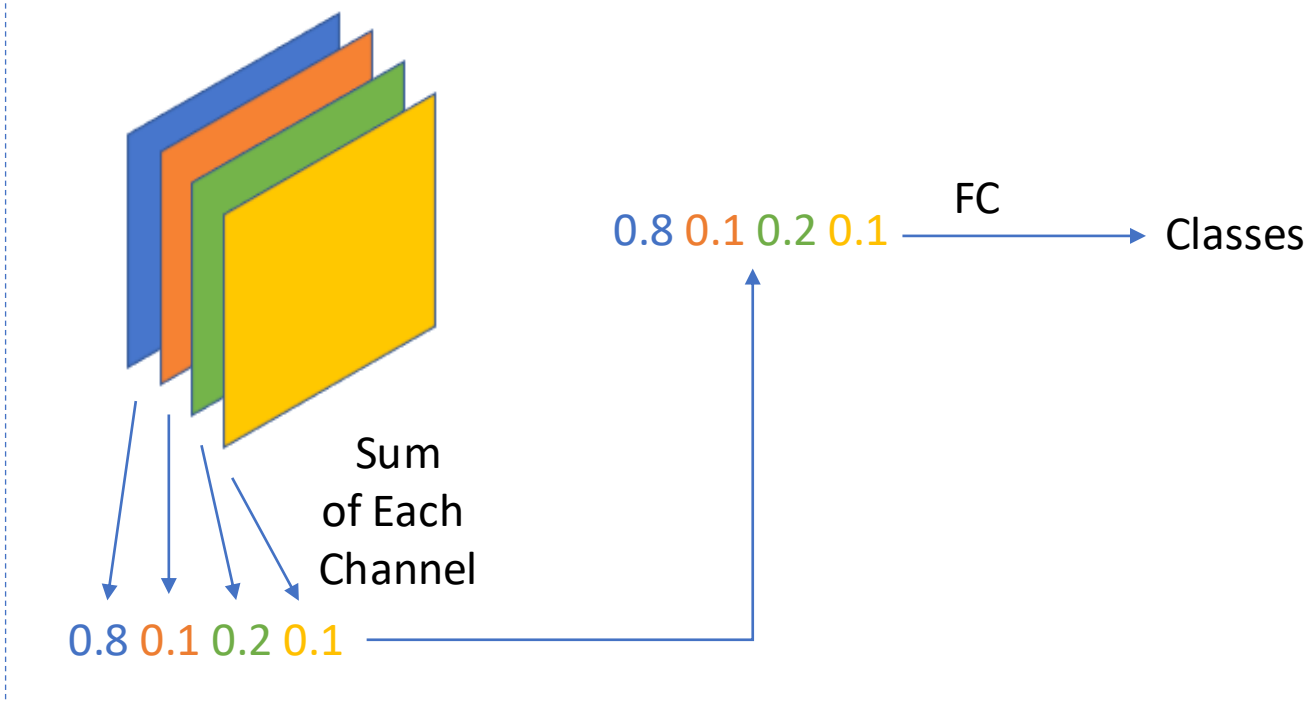
mean-std for each group over all neurons.

Previously on CENG501

Alternative to FC: Global Average Pooling

"Network In Network", https://arxiv.org/pdf/1312.4400.pdf





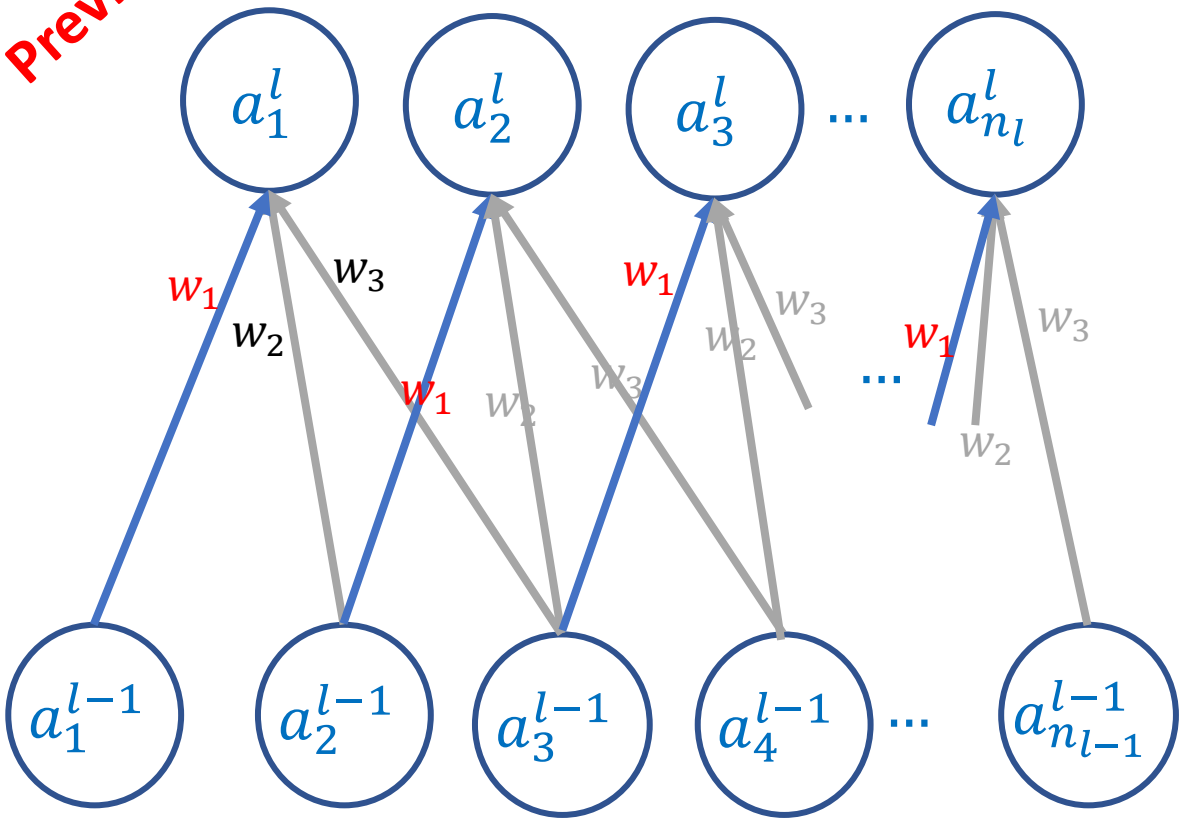
Backpropagation through convolution

Previously on CENG501

Feedforward:

$$a_i^l = \sigma(net_i^l)$$

$$net_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$



Gradient wrt. weights:

$$\frac{\partial L}{\partial w_k} = ?$$

$$= \frac{\partial L}{\partial a_1^l}\frac{\partial a_1^l}{\partial w_k} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial w_k} \dots$$

$$= \sum_i \frac{\partial L}{\partial a_i^l}\frac{\partial a_i^l}{\partial w_k}$$

$$= \sum_i \frac{\partial L}{\partial a_i^l}\frac{\partial a_i^l}{\partial net_i^l}\frac{\partial net_i^l}{\partial w_k}$$

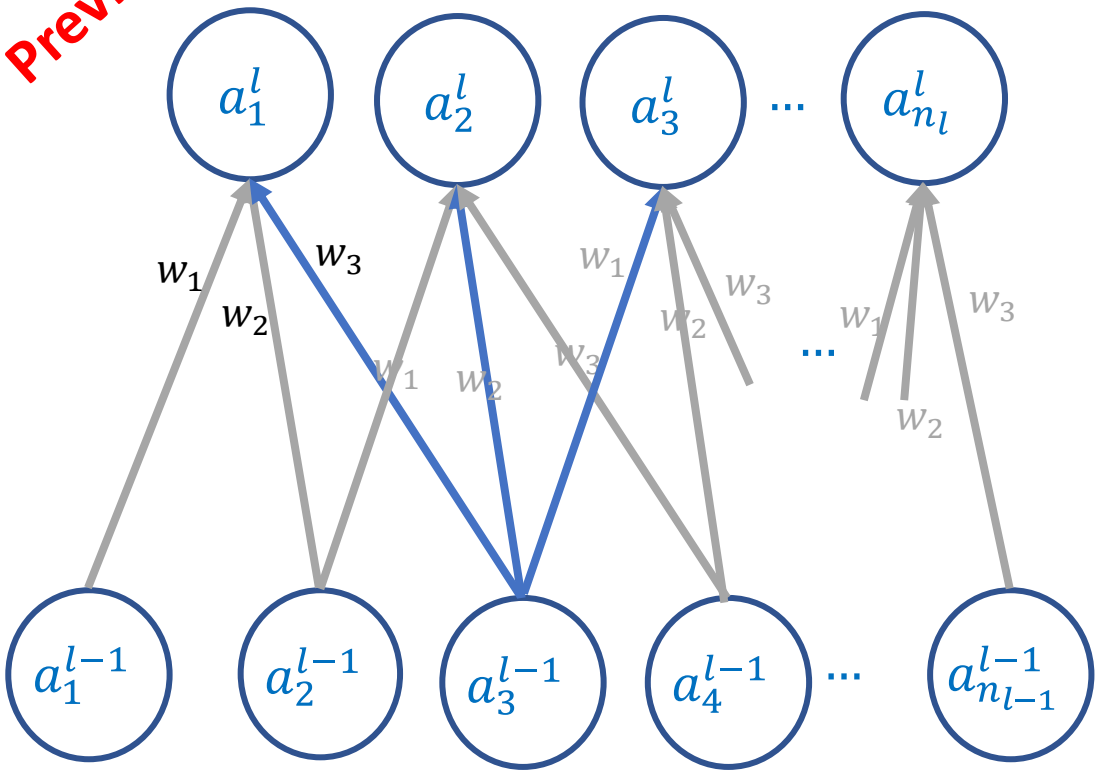
Backpropagation through convolution

Previously on CENG501

Feedforward:

$$a_i^l = \sigma(net_i^l)$$

$$net_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$



Gradient wrt. input layer:

$$\frac{\partial L}{\partial a_3^{l-1}} = ?$$

$$= \frac{\partial L}{\partial a_1^l}\frac{\partial a_1^l}{\partial net_1^l}\frac{\partial net_1^l}{\partial a_3^{l-1}} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial net_2^l}\frac{\partial net_2^l}{\partial a_3^{l-1}} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial net_2^l}\frac{\partial net_2^l}{\partial a_3^{l-1}}$$

$$= \frac{\partial L}{\partial net_1^l} w_3 + \frac{\partial L}{\partial net_2^l} w_2 + \frac{\partial L}{\partial net_3^l} w_1$$

This is also convolution!

In general:

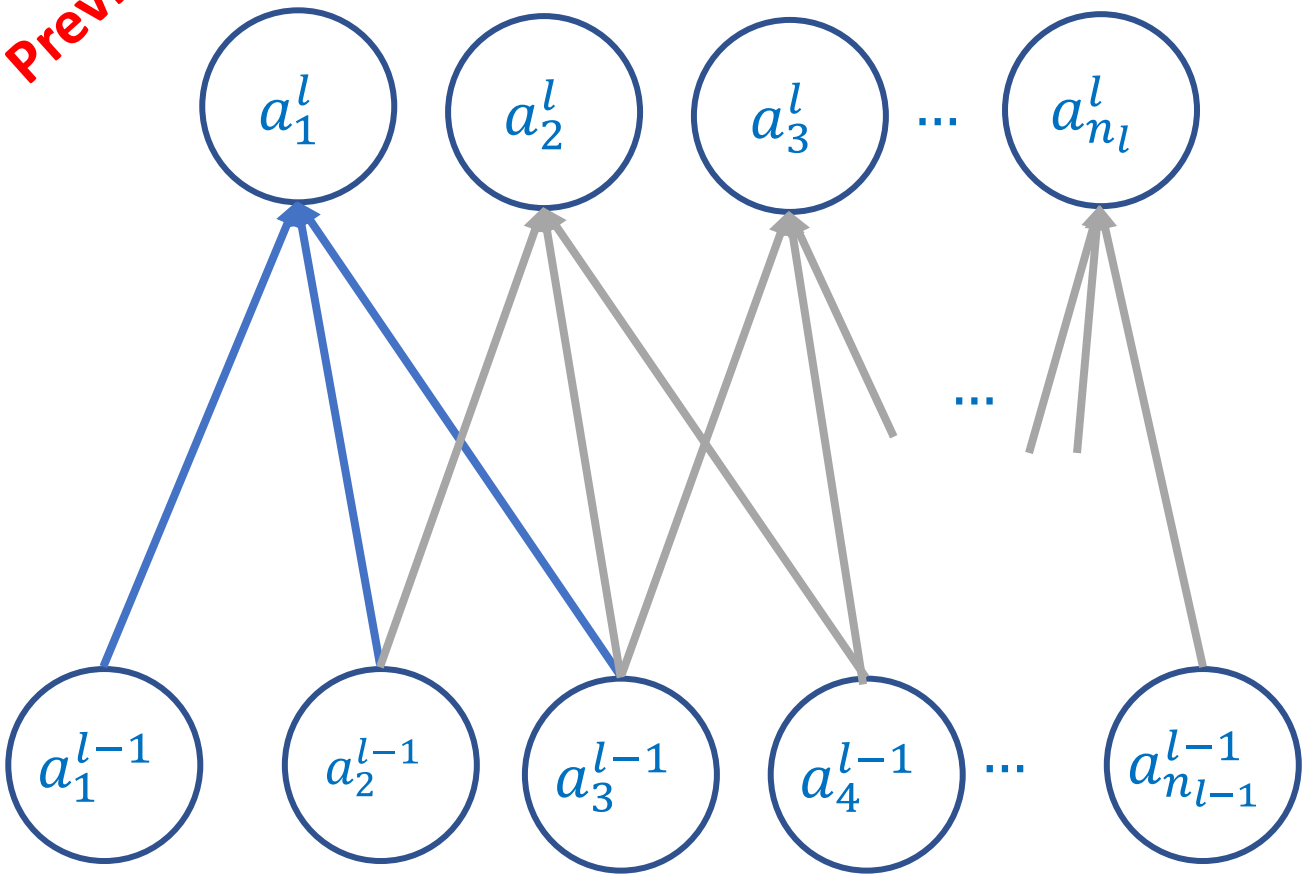
$$\frac{\partial L}{\partial a_i^{l-1}} = \sum_{j=1} \frac{\partial L}{\partial net_{i-j+1}^l} w_j$$

Previously on CENG501

Backpropagation through pooling

Feedforward:

$$a_i^l = \max\{a_{i+j-1}^{l-1}\}_{j=1}^{F}$$



Using derivative of max:

$$\frac{\partial L}{\partial a_i^{l-1}} = \frac{\partial L}{\partial net_k^l}\frac{\partial net_k^l}{\partial a_i^{l-1}}$$

$$= \begin{cases} \dfrac{\partial L}{\partial net_k^l}, & a_i^{l-1} \text{ is max} \\ 0, & \text{otherwise} \end{cases}$$

This requires that we save the index of the max activation (sometimes also called *the switches*) so that gradient "routing" is handled efficiently during backpropagation.

Trade-offs in architecture

Previously on CENG501

- Between filter size and number of layers (depth)
  - Keep the layer widths fixed.
  - *"When the time complexity is roughly the same, the deeper networks with smaller filters show better results than the shallower networks with larger filters."*
- Between layer width and number of layers (depth)
  - Keep the size of the filters fixed.
  - *"We find that increasing the depth leads to considerable gains, even the width needs to be properly reduced."*
- Between filter size and layer width
  - Keep the number of layers (depth) fixed.
  - No significant difference

Convolutional Neural Networks at Constrained Time Cost

Kaiming He Jian Sun
Microsoft Research
{kahe, jiansun}@microsoft.com

4.4. Is Deeper Always Better?

The above results have shown the priority of depth for improving accuracy. With the above trade-offs, we can have a much deeper model if we further decrease width/filter sizes and increase depth. However, in experiments we find that the accuracy is stagnant or even reduced in some of our very deep attempts. There are two possible explanations: (1) the width/filter sizes are reduced overly and may harm the accuracy, or (2) overly increasing the depth will degrade the accuracy even if the other factors are not traded. To understand the main reason, *in this subsection we do not constrain the time complexity* but solely increase the depth without other changes.

Finetuning

Previously on CENG501

1. If the new dataset is small and similar to the original dataset used to train the CNN:
   - Finetuning the whole network may lead to overfitting
   - Just train the newly added layer

2. If the new dataset is big and similar to the original dataset:
   - The more, the merrier: go ahead and train the whole network

3. If the new dataset is small and different from the original dataset:
   - Not a good idea to train the whole network
   - However, add your new layer not to the top of the network, since those parts are very dataset (problem) specific
   - Add your layer to earlier parts of the network

4. If the new dataset is big and different from the original dataset:
   - We can "finetune" the whole network
   - This amounts to a new training problem by initializing the weights with those of another network

Embed the codes in a lower-dimensional space

Previously on CENG501

- Place images into a 2D space such that images which produce similar CNN codes are placed close.
- You can use, e.g., t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE embedding of a set of images based on their CNN codes. Images that are nearby each other are also close in the CNN representation space, which implies that the CNN "sees" them as being very similar. Notice that the similarities are more often class-based and semantic rather than pixel and color-based. For more details on how this visualization was produced the associated code, and more related visualizations at different scales refer to t-SNE visualization of CNN codes.

http://cs231n.github.io/convolutional-networks/

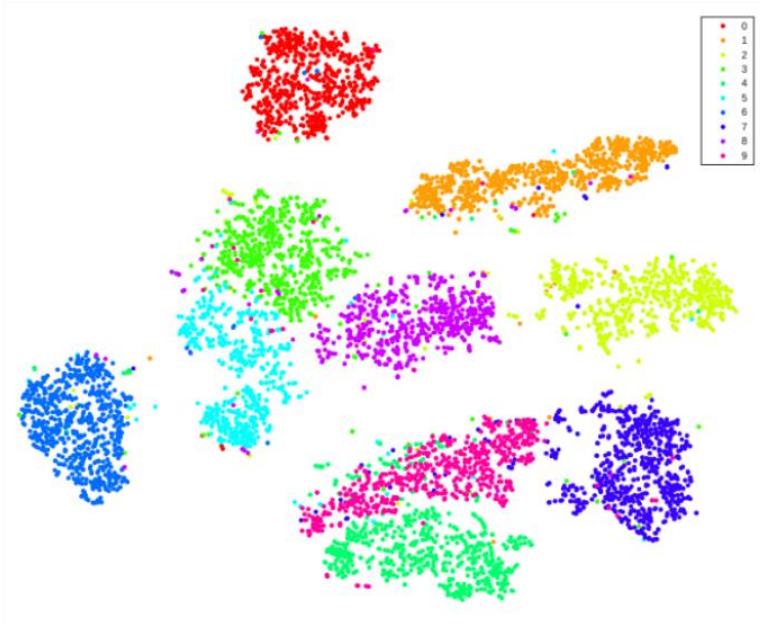


Figure 1 : Illustration of t-SNE on MNIST dataset

Figure: Laurens van der Maaten and Geoffrey Hinton

Previously on CENG501

Data gradients

- The gradient with respect to the input is high for pixels which are on the object

We start with a motivational example. Consider the linear score model for the class c:

$$S_c(I) = w_c^T I + b_c, \tag{2}$$

where the image I is represented in the vectorised (one-dimensional) form, and w_c and b_c are respectively the weight vector and the bias of the model. In this case, it is easy to see that the magnitude of elements of w defines the importance of the corresponding pixels of I for the class c.

In the case of deep ConvNets, the class score $S_c(I)$ is a highly non-linear function of I, so the reasoning of the previous paragraph can not be immediately applied. However, given an image I_0, we can approximate $S_c(I)$ with a linear function in the neighbourhood of I_0 by computing the first-order Taylor expansion:

$$S_c(I) \approx w^T I + b, \tag{3}$$

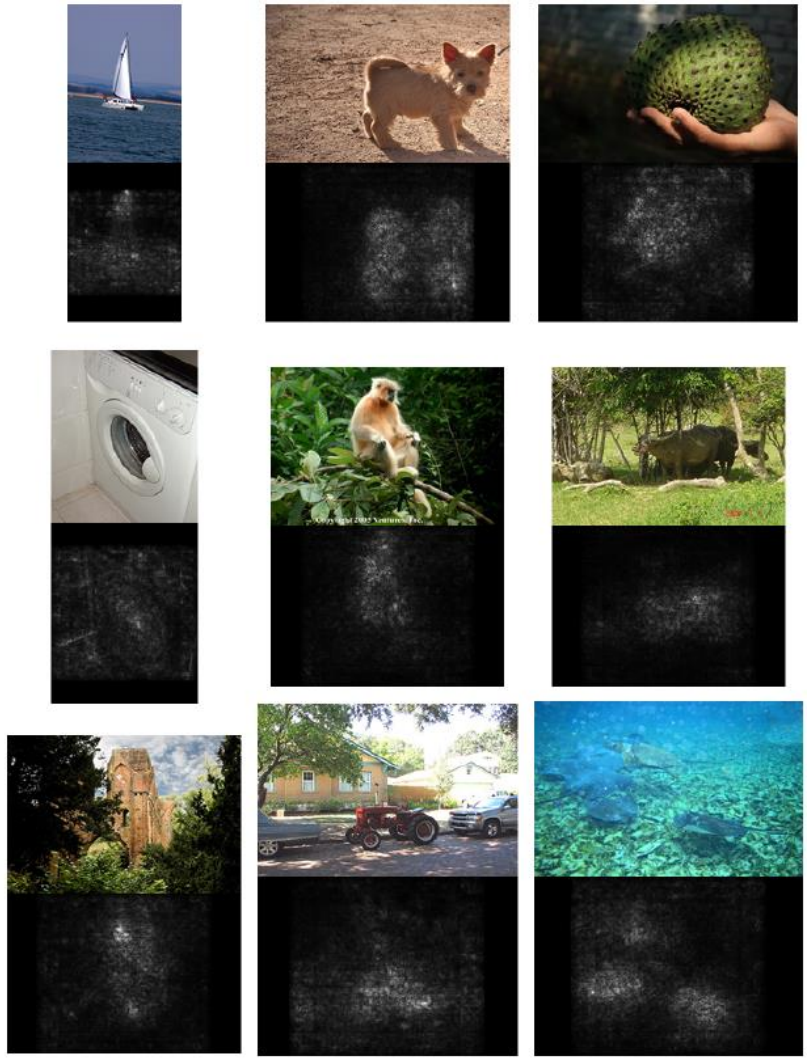
where w is the derivative of S_c with respect to the image I at the point (image) I_0:

$$w = \left.\frac{\partial S_c}{\partial I}\right|_{I_0}. \tag{4}$$

Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps

Karen Simonyan Andrea Vedaldi Andrew Zisserman
Visual Geometry Group, University of Oxford
{karen,vedaldi,az}@robots.ox.ac.uk

2014



Previously on CENG501

Class Activation Maps

- Weighted combination of the feature maps before GAP:

$$M(x,y) = \sum_k w_k^c f_k(x,y)$$

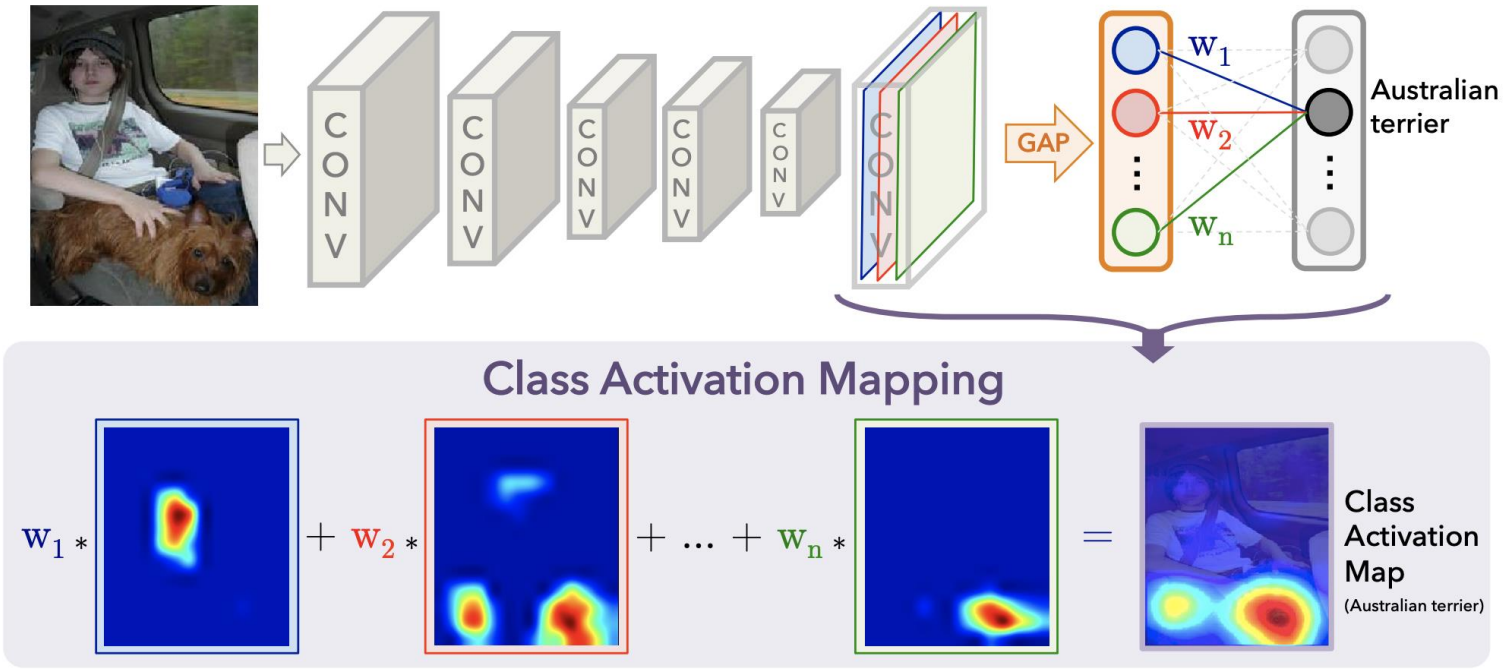


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2921–2929.

Previously on CENG501

Class Activation Maps

- GradCAM:

$$\alpha_k^c = \sum_{x,y} \frac{\partial S_c}{\partial f_k(x,y)}$$

$$M^c(x,y) = ReLU\left(\sum_k \alpha_k^c f_k(x,y)\right)$$

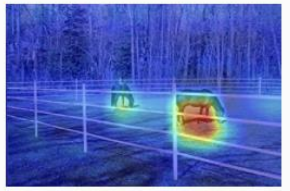
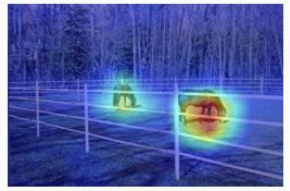
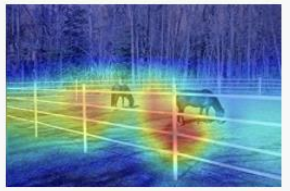
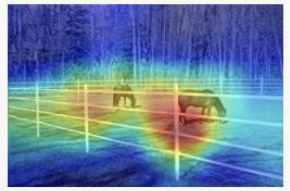
Network	Image	GradCAM	GradCAM++
VGG16			
Resnet50			

Figure: https://pypi.org/project/grad-cam/

R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization," arXiv preprint arXiv:1610.02391, 2016.

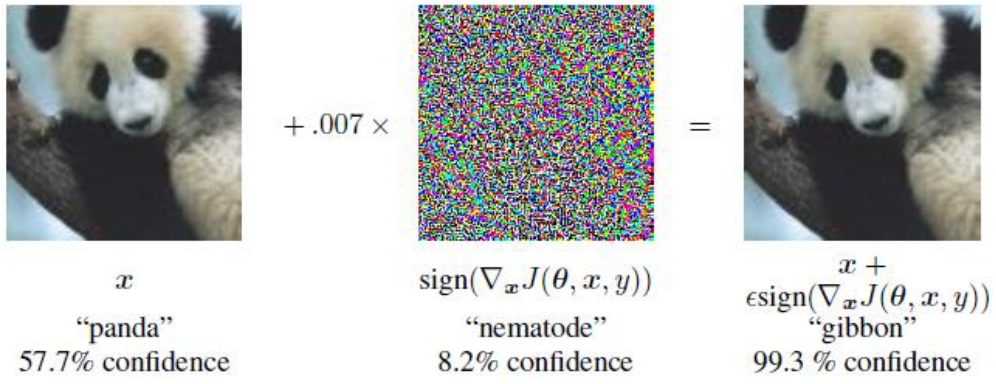
Chattopadhay, A., Sarkar, A., Howlader, P., & Balasubramanian, V. N. (2018, March). Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)* (pp. 839-847). IEEE.

Previously on CENG501

Fooling ConvNets

- Given an image I labeled as l_1, find minimum "r" (noise) such that $I + r$ is classified as a different label, l_2.
- I.e., minimize:

$$\arg\min_r loss(I + r, l_2) + c|r|$$



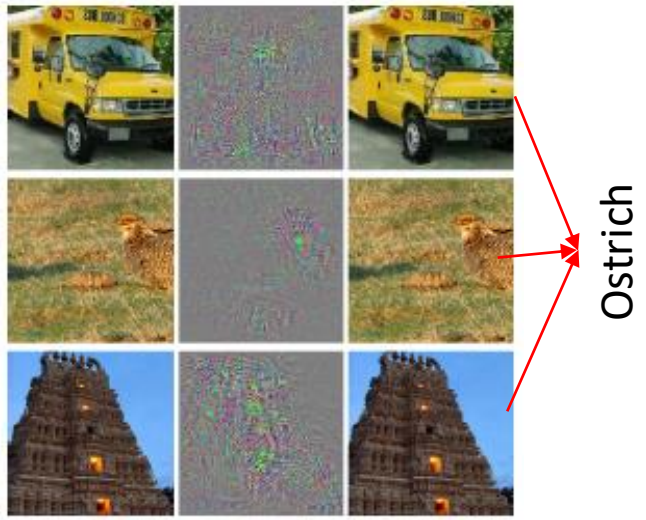
Published as a conference paper at ICLR 2015

EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
Google Inc., Mountain View, CA
{goodfellow,shlens,szegedy}@google.com

Intriguing properties of neural networks

Christian Szegedy Google Inc.
Wojciech Zaremba New York University
Ilya Sutskever Google Inc.
Joan Bruna New York University
Dumitru Erhan Google Inc.
Ian Goodfellow University of Montreal
Rob Fergus New York University
Facebook Inc.



AlexNet (2012)

Previously on CENG501

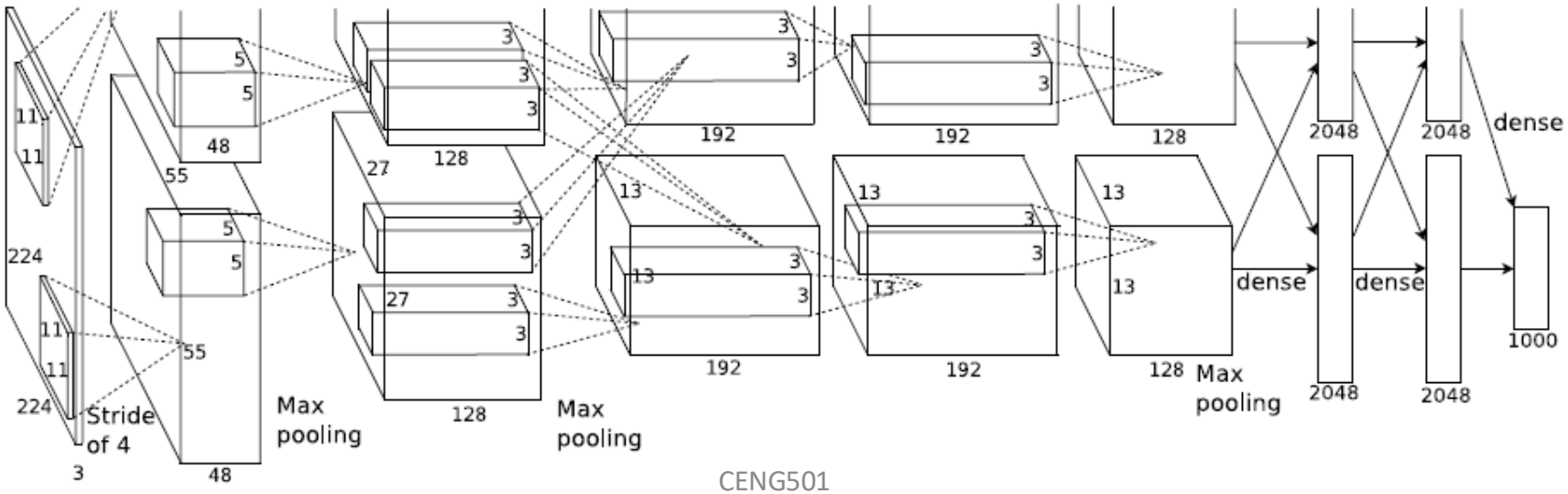
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

- Popularized CNN in computer vision & pattern recognition
- ImageNet ILSVRC challenge 2012 winner
- Similar to LeNet
  - Deeper & bigger
  - Many CONV layers on top of each other (rather than adding immediately a pooling layer after a CONV layer)
  - Uses GPU
- 650K neurons. 60M parameters. Trained on 2 GPUs for a week.



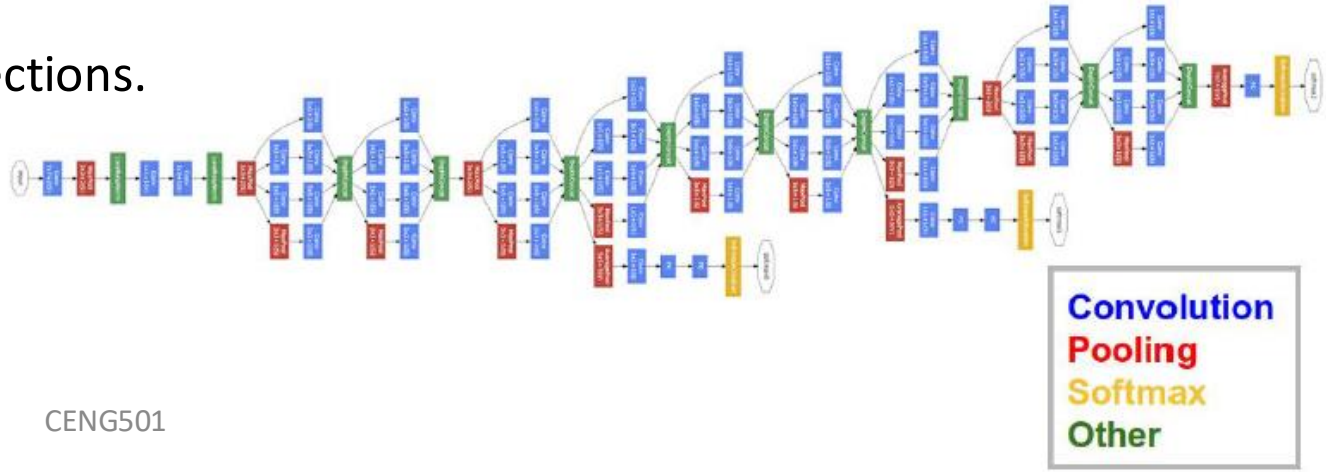
GoogleNet (2014)

Previously on CENG501

- ImageNet 2014 winner
- Contributions:
  - Inception module
    - Dramatically reduced parameters (from 60M in AlexNet to 4M)
  - Avg Pooling at the top, instead of fully-connected layer ➔ Reduced number of parameters
- Motivation:
  - Going bigger (in depth or width) means too many parameters.
  - Go bigger by maintaining sparse connections.

Going deeper with convolutions

Christian Szegedy, Google Inc.
Wei Liu, University of North Carolina, Chapel Hill
Yangqing Jia, Google Inc.
Pierre Sermanet, Google Inc.
Scott Reed, University of Michigan
Dragomir Anguelov, Google Inc.
Dumitru Erhan, Google Inc.
Vincent Vanhoucke, Google Inc.
Andrew Rabinovich, Google Inc.



Previously on CENG501

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Previously on CENG501

ResNet (2015)

Residual (shortcut) connections

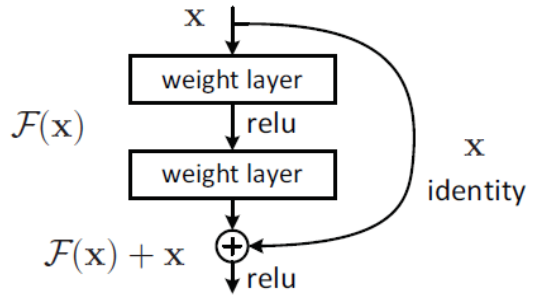


Figure 2. Residual learning: a building block.

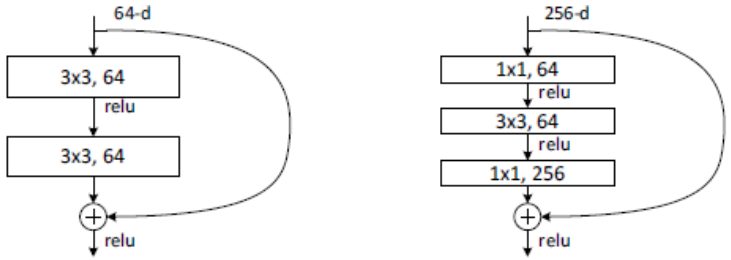
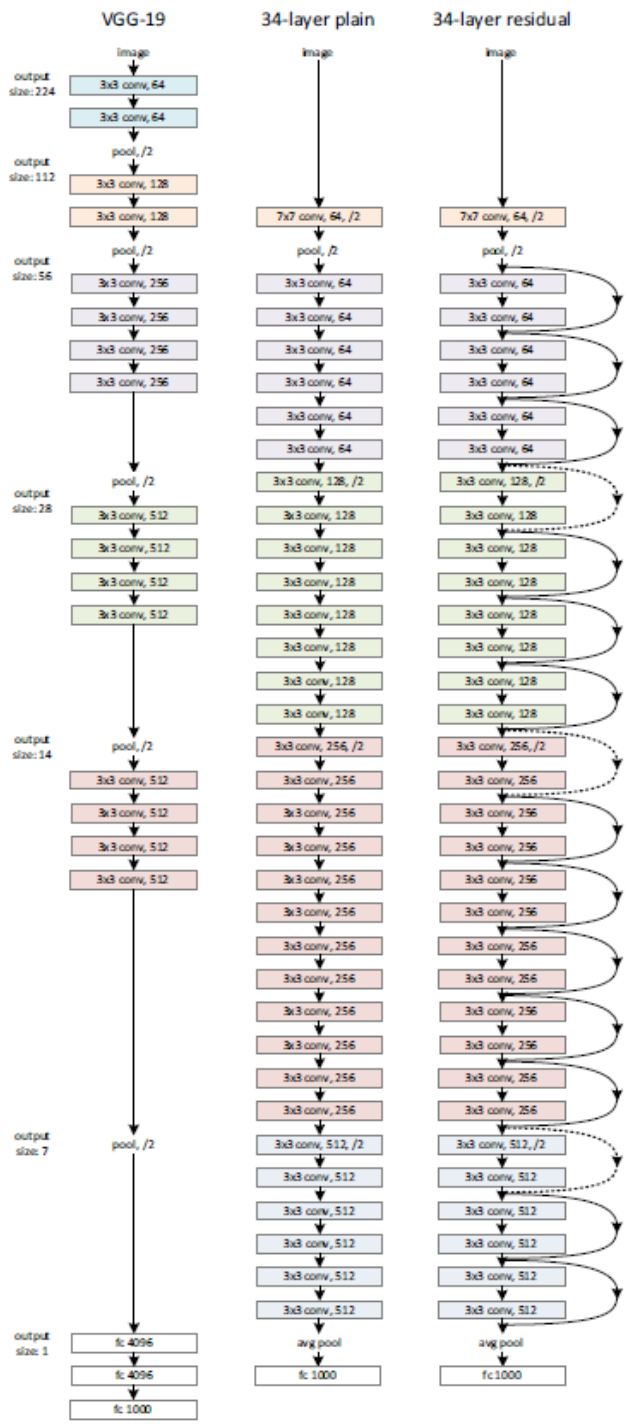
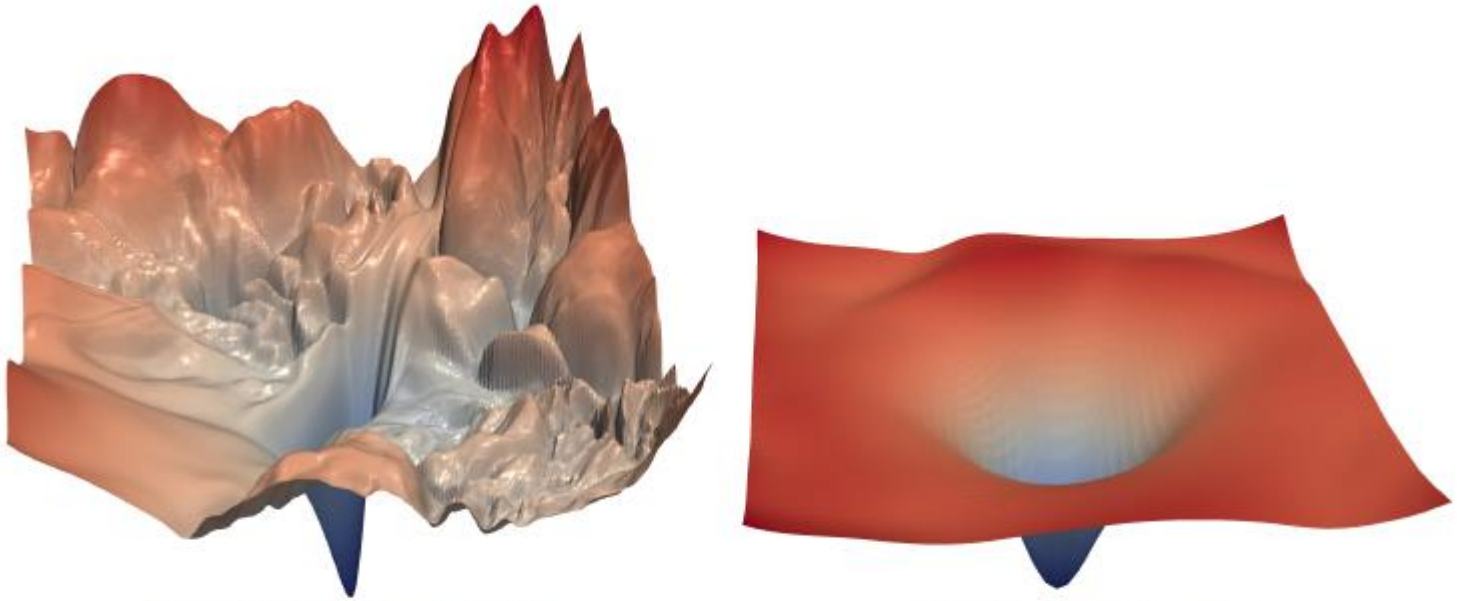


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.



Previously on CENG501

Effect of residual connections



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The vertical axis is logarithmic to show dynamic range. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

VISUALIZING THE LOSS LANDSCAPE OF NEURAL NETS

2018

Hao Li[1], Zheng Xu[1], Gavin Taylor[2], Christoph Studer[3], Tom Goldstein[1]
[1]University of Maryland, College Park, [2]United States Naval Academy, [3]Cornell University
{haoli,zhengxu,tomg}@cs.umd.edu, taylor@usna.edu, studer@cornell.edu

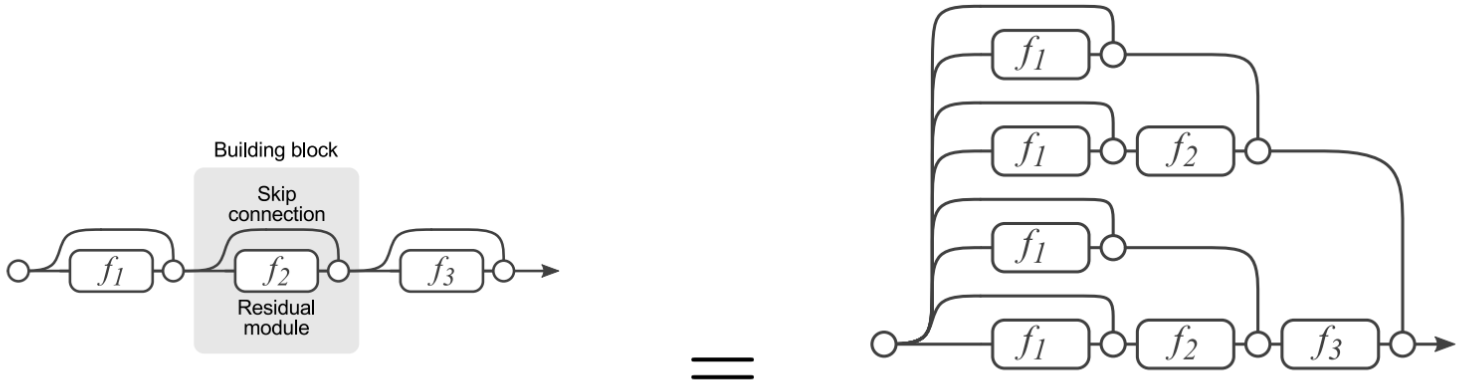
ResNet: Ensemble of Shallow Networks

Previously in CENG501

Residual Networks Behave Like Ensembles of Relatively Shallow Networks

Andreas Veit **Michael Wilber** **Serge Belongie**
Department of Computer Science & Cornell Tech
Cornell University
{av443, mjw285, sjb344}@cornell.edu

2016



(a) Conventional 3-block residual network (b) Unraveled view of (a)

Figure 1: Residual Networks are conventionally shown as (a), which is a natural representation of Equation (1). When we expand this formulation to Equation (6), we obtain an *unraveled view* of a 3-block residual network (b). Circular nodes represent additions. From this view, it is apparent that residual networks have $O(2^n)$ implicit paths connecting input and output and that adding a block doubles the number of paths.

ResNeXt

Previously on CENG501

Aggregated Residual Transformations for Deep Neural Networks

Saining Xie[1] Ross Girshick[2] Piotr Dollár[2] Zhuowen Tu[1] Kaiming He[2]
[1]UC San Diego [2]Facebook AI Research
{s9xie,ztu}@ucsd.edu {rbg,pdollar,kaiminghe}@fb.com

2017

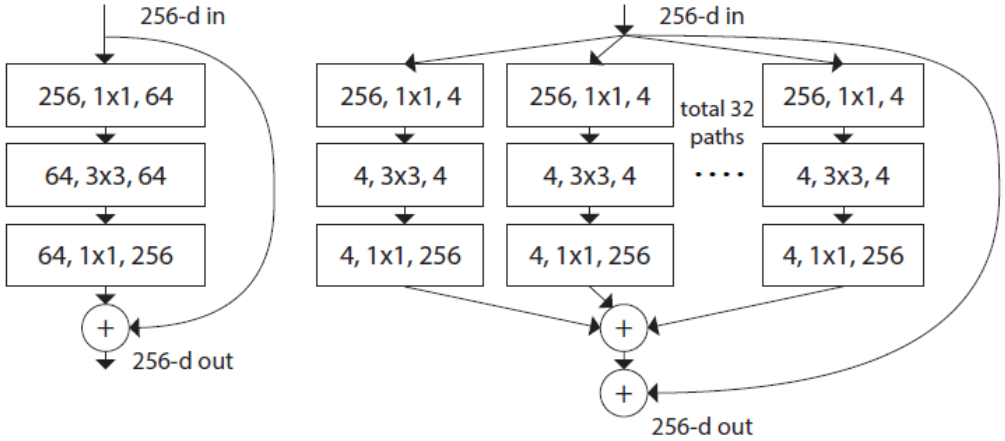
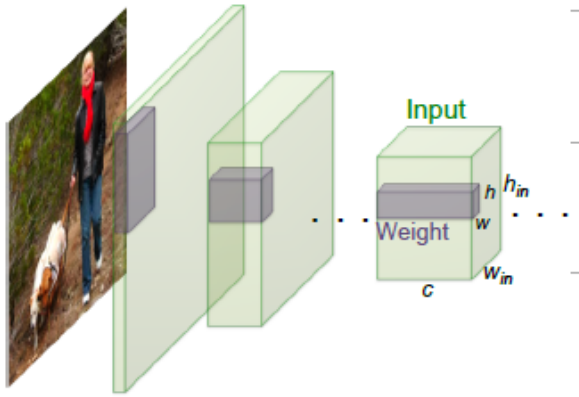


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Previously on CENG501

Binary networks



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Time Saving on CPU (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%53.8
BinaryWeight Binary Input **(XNOR-Net)**	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Fig. 1: We propose two efficient variations of convolutional neural networks. **Binary-Weight-Networks**, when the weight filters contains binary values. **XNOR-Networks**, when both weigh and input have binary values. These networks are very efficient in terms of memory and computation, while being very accurate in natural image classification. This offers the possibility of using accurate vision techniques in portable devices with limited resources.

Today

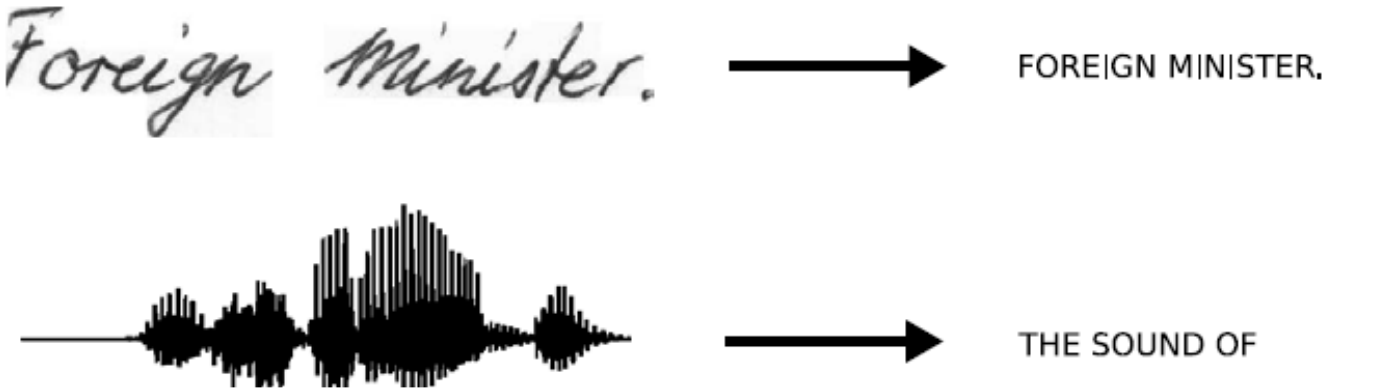
- Continue with CNNs
  - Different types of convolution in CNNs
  - Pooling and normalization operations
  - Designing CNN architectures
  - Backpropagation through a CNN
  - Transfer learning with CNNs
  - Popular CNN architectures

Administrative Notes

- Paper Selection
  - Feedback provided
  - Update your papers until the end of 15th of March, Sunday

Sequence Labeling/Modeling: Motivation

Why do we need them?



A. Graves, “Supervised Sequence Labelling with Recurrent Neural Networks”, 2012.

Different types of sequence learning / recognition problems

- Sequence Classification
  - A sequence to a label
  - E.g., recognizing a single spoken word
  - Length of the sequence is fixed
  - Why RNNs then? Because sequential modeling provides robustness against translations and distortions.
- Segment Classification
  - Segments in a sequence correspond to labels
- Temporal Classification
  - General case: sequence (input) to sequence (label) modeling.
  - No clue about where input or label starts.

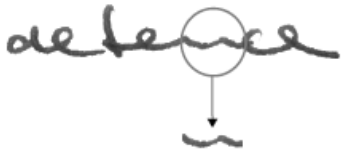
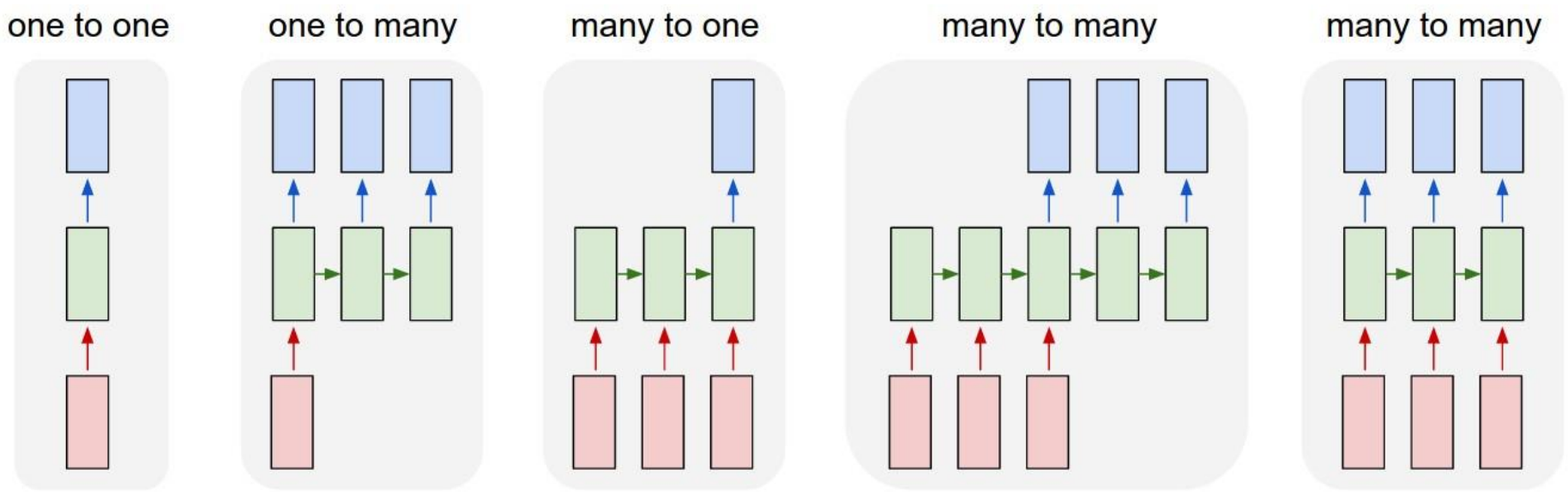


Fig. 2.3 Importance of context in segment classification. The word 'defence' is clearly legible. However the letter 'n' in isolation is ambiguous.

A. Graves, "Supervised Sequence Labelling with Recurrent Neural Networks", 2012.

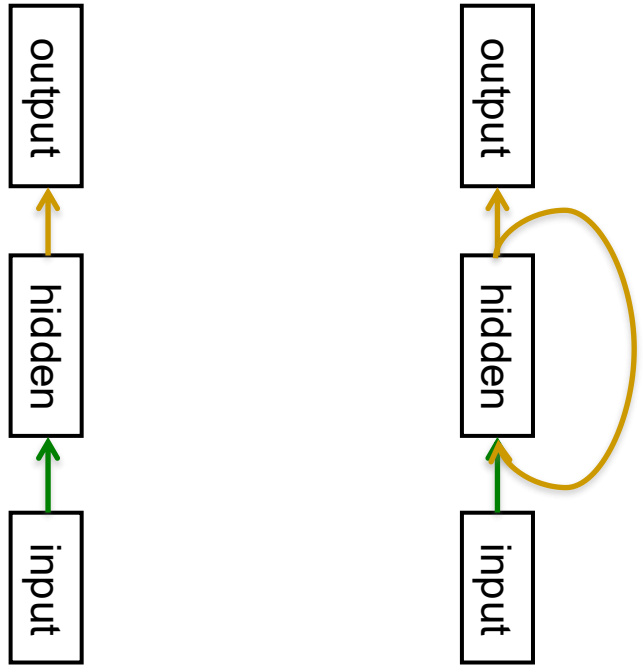
Different types of sequence learning / recognition problems



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

Recurrent Neural Networks

Recurrent Neural Networks (RNNs)



Feed-forward networks

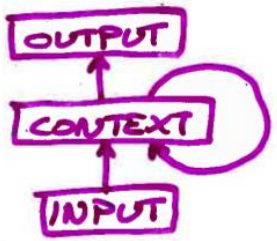
Recurrent networks

- RNNs are very powerful because:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.
- More formally, RNNs are Turing complete.

Adapted from Hinton

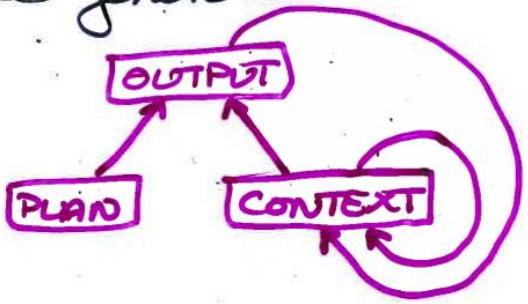
Some examples

– Temporal pattern recognition



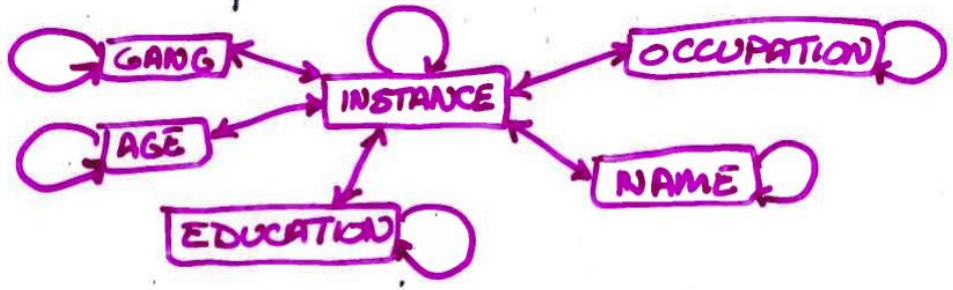
INPUT1, INPUT2, INPUT3, ..., INPUT+
⇒ OUTPUT+
e.g., speech recognition
e.g., event recognition
e.g., natural language understand

– Sequence generation



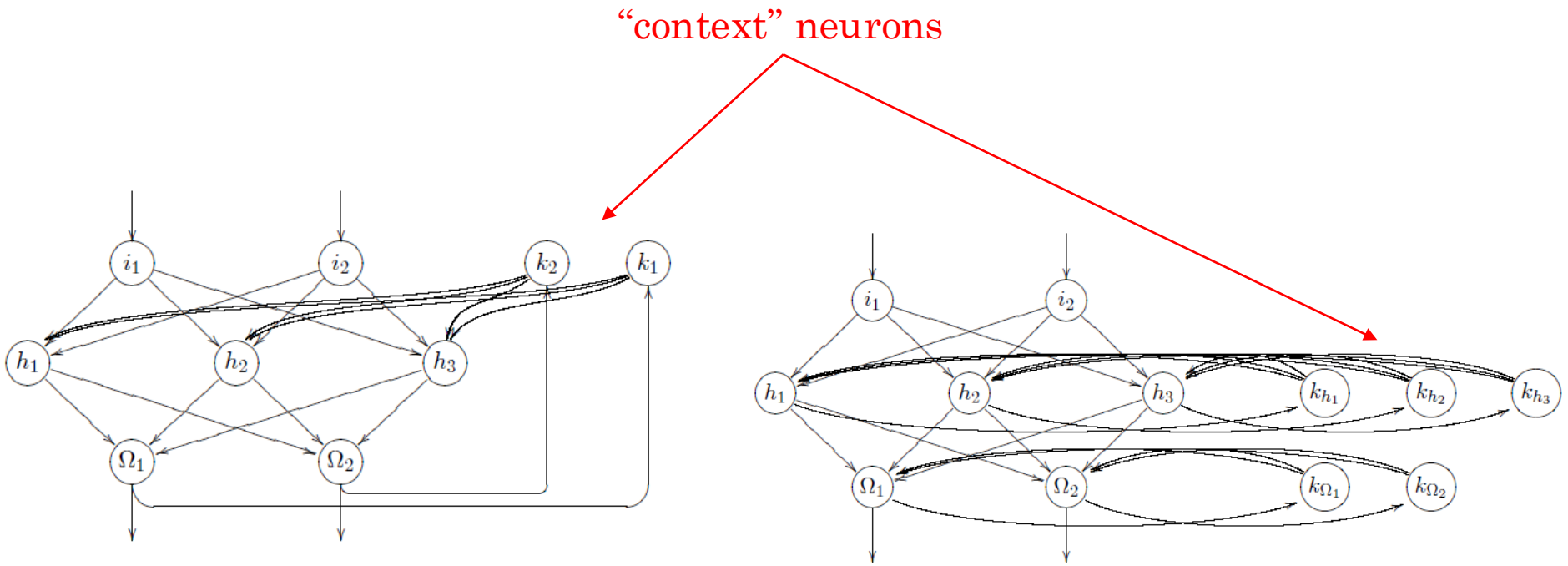
PLAN ⇒ OUTPUT1, OUTPUT2
OUTPUT3, ..., OUTPUT+
e.g., speech production
e.g., motor control
e.g., planning and acting

– Pattern completion / constraint satisfaction



Slide: Michael Mozer

Some examples



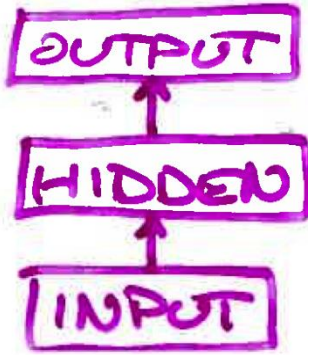
Jordan Networks

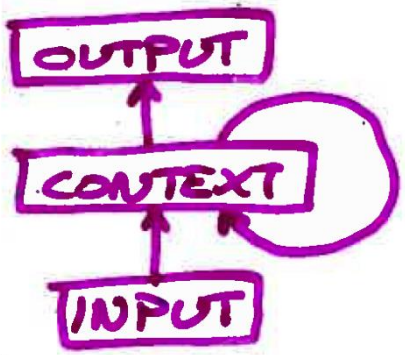
Elman Networks

Figs: David Kriesel

Challenge

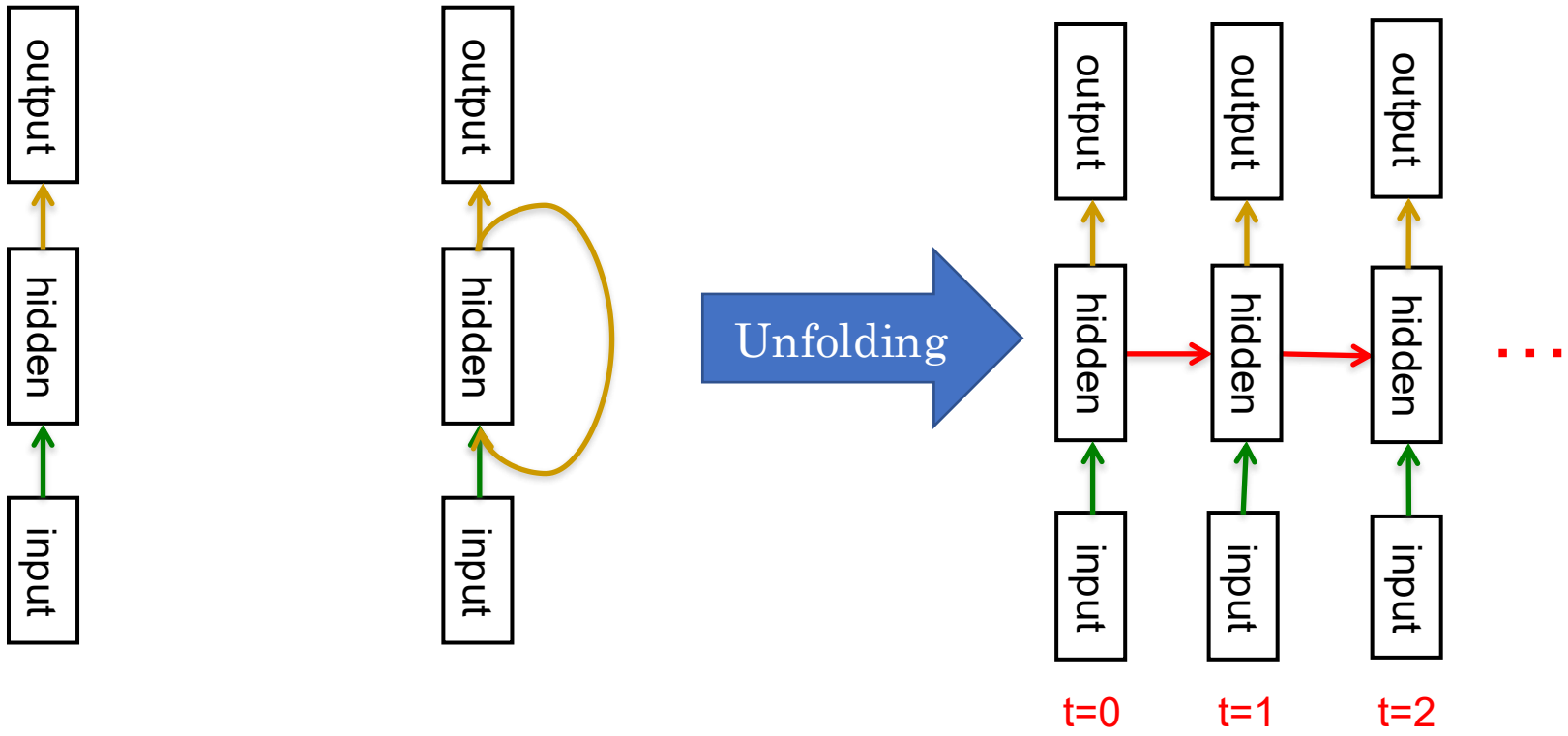
- Back propagation is designed for feedforward nets
- What would it mean to back propagate through a recurrent network?
  - error signal would have to travel back in time





Slide: Michael Mozer

Unfolding



Feed-forward networks

Recurrent networks

Unfolding (another example)

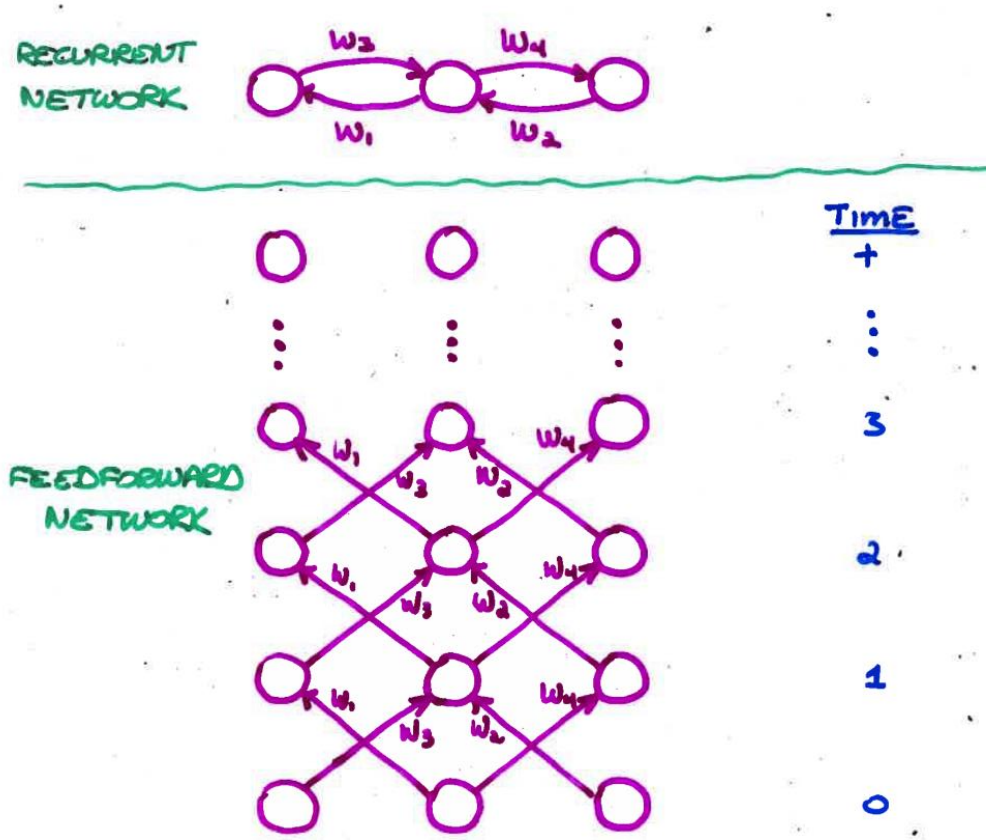
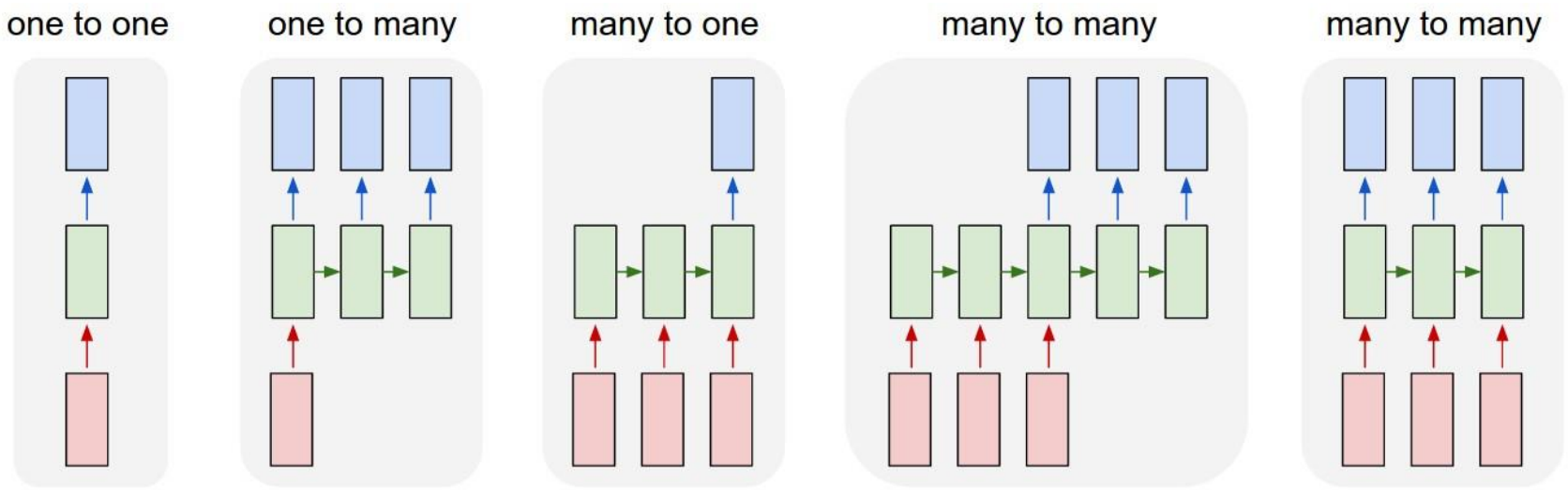
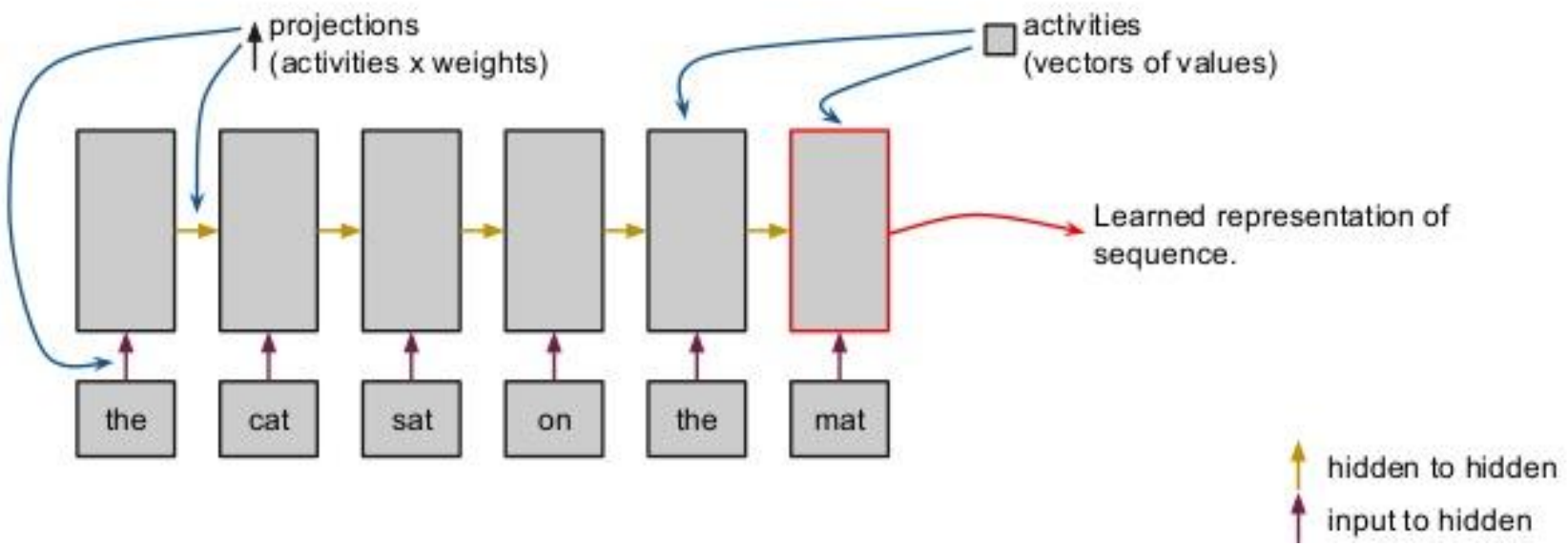


Figure: Michael Mozer



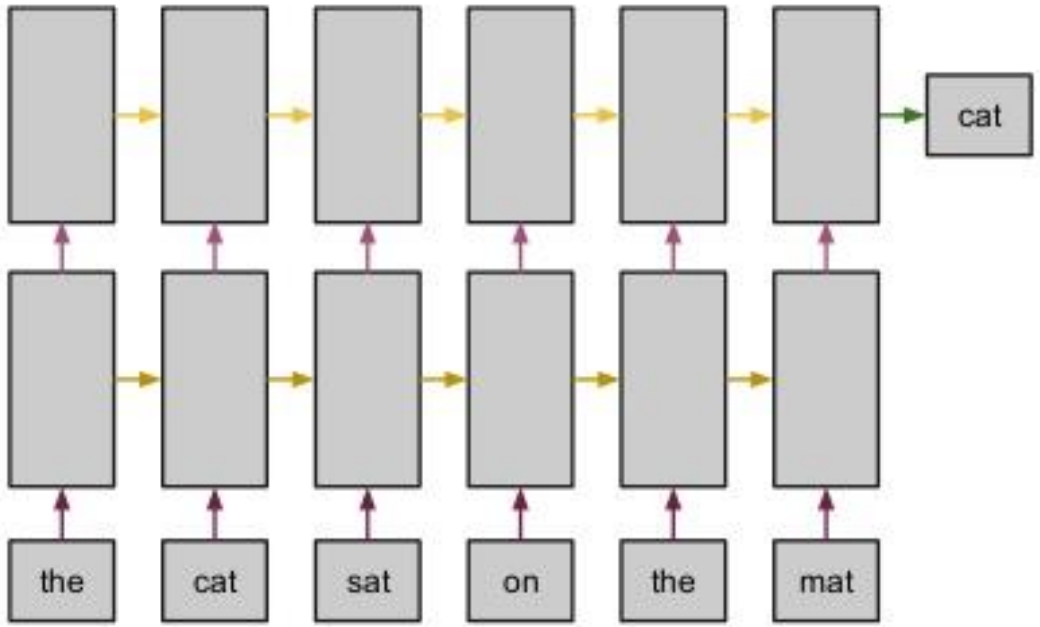
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

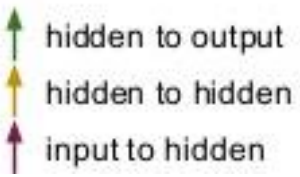
How an RNN works



Alec Radford

You can stack them too





Alec Radford

Unfolding implications

- Entails duplication of weights => weight sharing
- Sharing weights means their gradients will be accumulated over time and reflected on the weights
- Unfolded network has the same dynamics of the RNN for a fixed number of time steps!

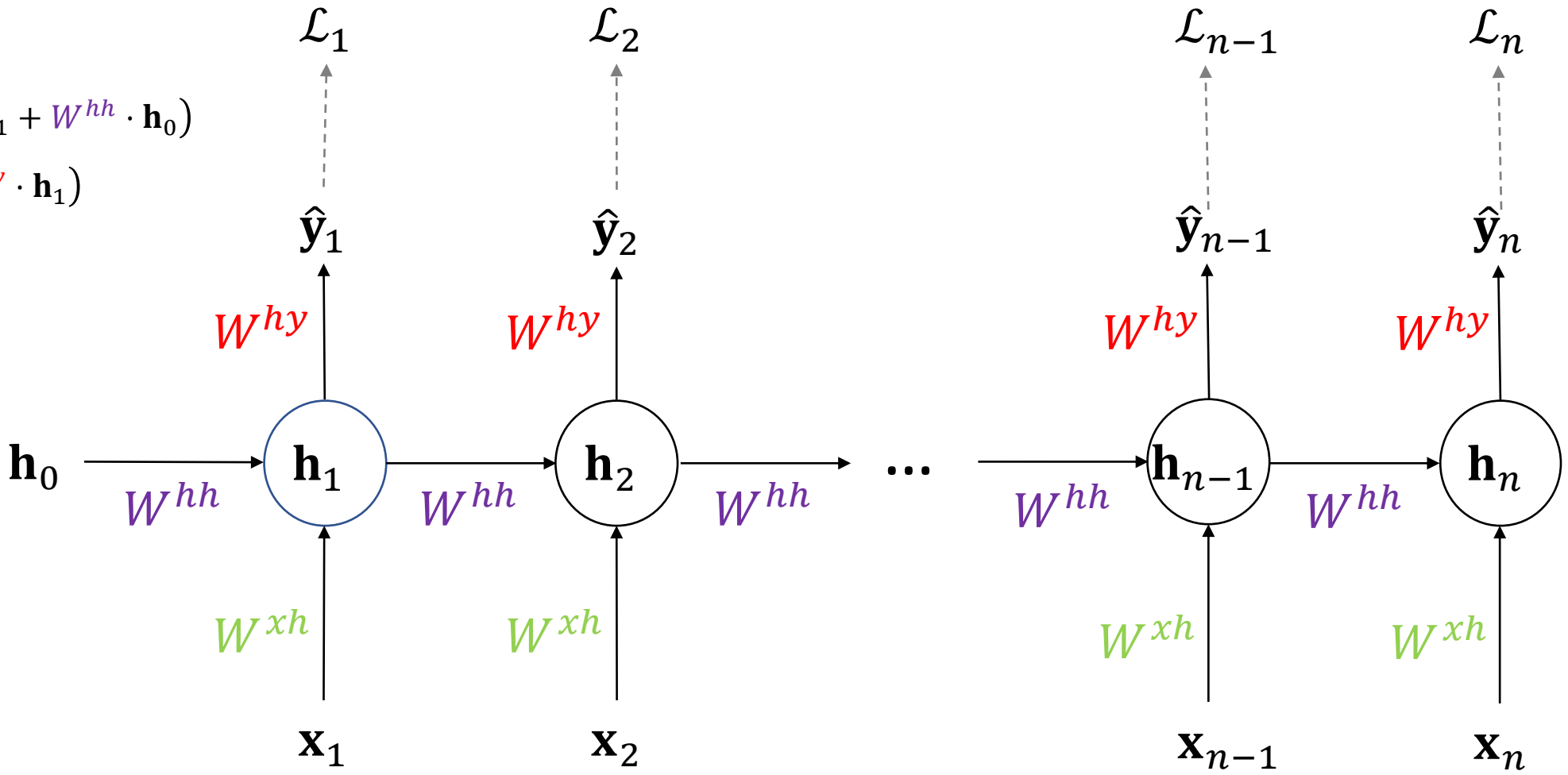
Back-propagation Through Time

Feedforward through Vanilla RNN

$$\mathbf{h}_1 = tanh\left(W^{xh} \cdot \mathbf{x}_1 + W^{hh} \cdot \mathbf{h}_0\right)$$

$$\hat{\mathbf{y}}_1 = softmax\left(W^{hy} \cdot \mathbf{h}_1\right)$$

$$\mathcal{L}_1 = CE(\hat{\mathbf{y}}_1, \mathbf{y}_1)$$



Feedforward through Vanilla RNN

The Vanilla RNN Model

First time-step ($t = 1$):

$$\mathbf{h}_1 = tanh\left(W^{xh} \cdot \mathbf{x}_1 + W^{hh} \cdot \mathbf{h}_0\right)$$

$$\hat{\mathbf{y}}_1 = softmax\left(W^{hy} \cdot \mathbf{h}_1\right)$$

$$\mathcal{L}_1 = CE(\hat{\mathbf{y}}_1, \mathbf{y}_1)$$

In general:

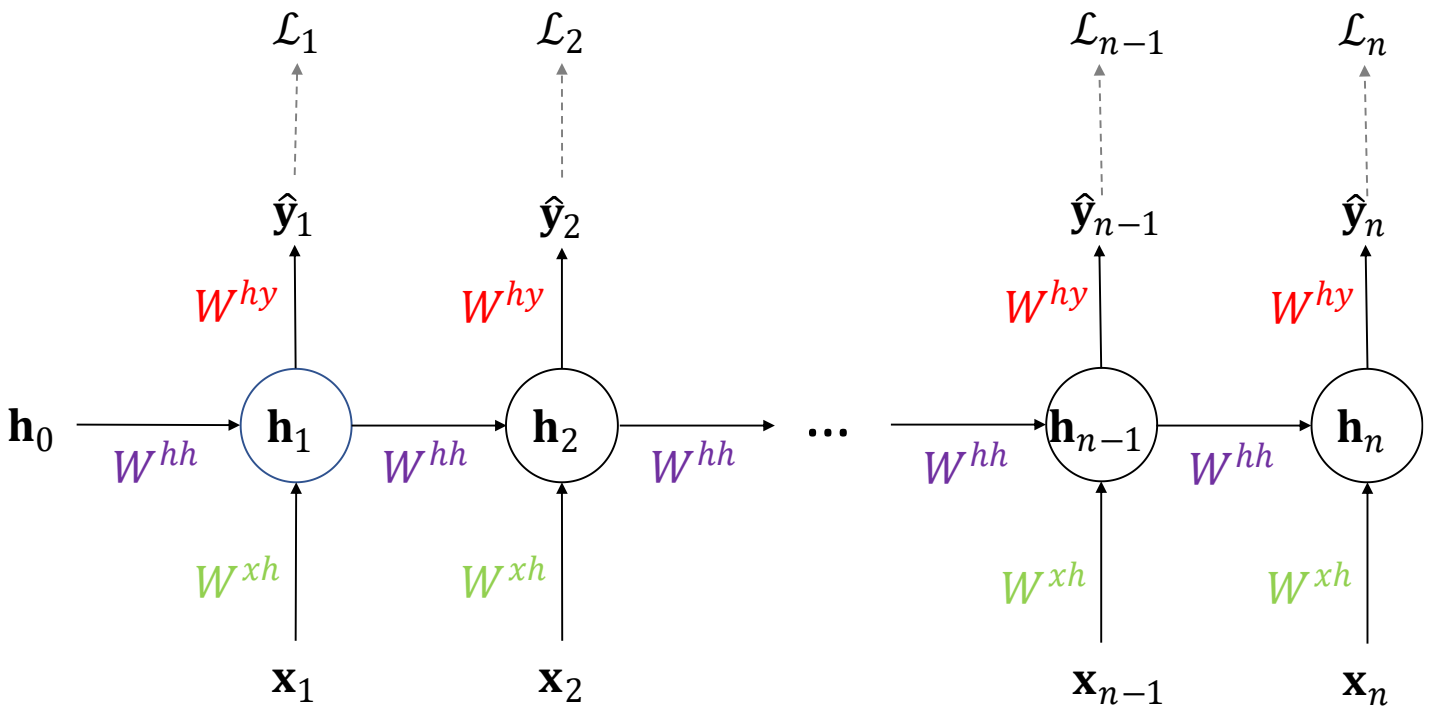
$$\mathbf{h}_t = tanh\left(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1}\right)$$

$$\hat{\mathbf{y}}_t = softmax\left(W^{hy} \cdot \mathbf{h}_t\right)$$

$$\mathcal{L}_t = CE(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

In total:

$$\mathcal{L} = \sum_t \mathcal{L}_t$$



Backpropagation through Vanilla RNN

The Vanilla RNN Model

In general:

$$\mathbf{h}_t = tanh\left(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1}\right)$$

$$\hat{\mathbf{y}}_t = softmax\left(W^{hy} \cdot \mathbf{h}_t\right)$$

$$\mathcal{L}_t = CE(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

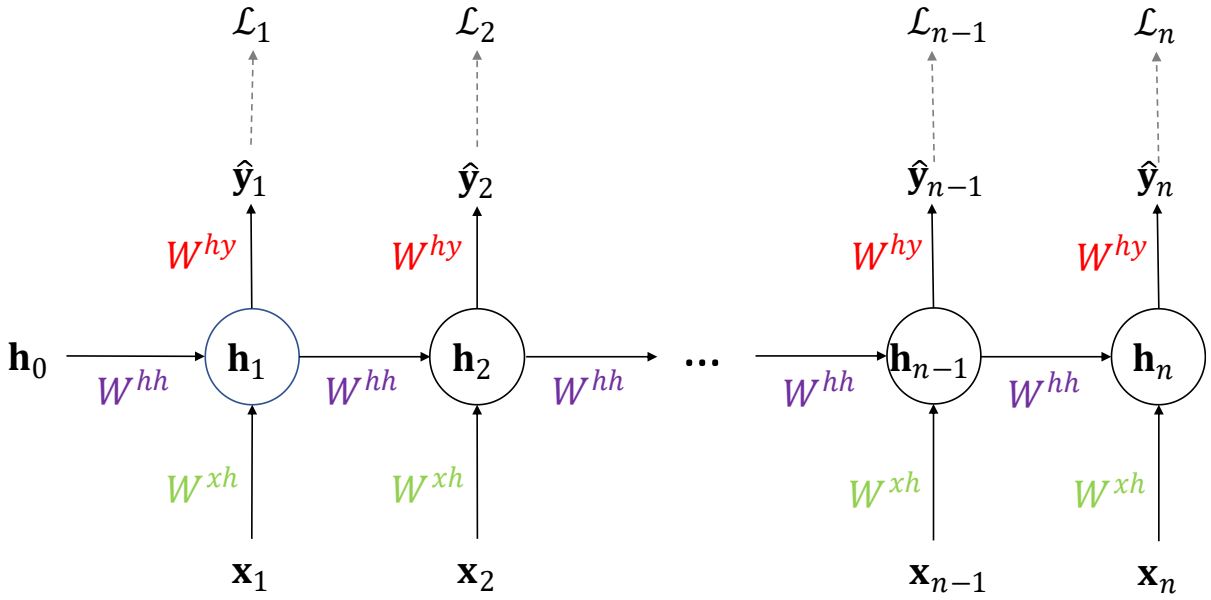
In total:

$$\mathcal{L} = \sum_t \mathcal{L}_t$$

$$\frac{\partial \mathcal{L}}{\partial W^{hy}} = ?$$

$$= \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_n}\frac{\partial \hat{\mathbf{y}}_n}{\partial W^{hy}} + \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_{n-1}}\frac{\partial \hat{\mathbf{y}}_{n-1}}{\partial W^{hy}} + \cdots + \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_1}\frac{\partial \hat{\mathbf{y}}_1}{\partial W^{hy}}$$

$$= \sum_{t=1..n} \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_t}\frac{\partial \hat{\mathbf{y}}_t}{\partial W^{hy}}$$



Backpropagation through Vanilla RNN

The Vanilla RNN Model

In general:

$$\mathbf{h}_t = tanh\left(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1}\right)$$

$$\hat{\mathbf{y}}_t = softmax\left(W^{hy} \cdot \mathbf{h}_t\right)$$

$$\mathcal{L}_t = CE(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

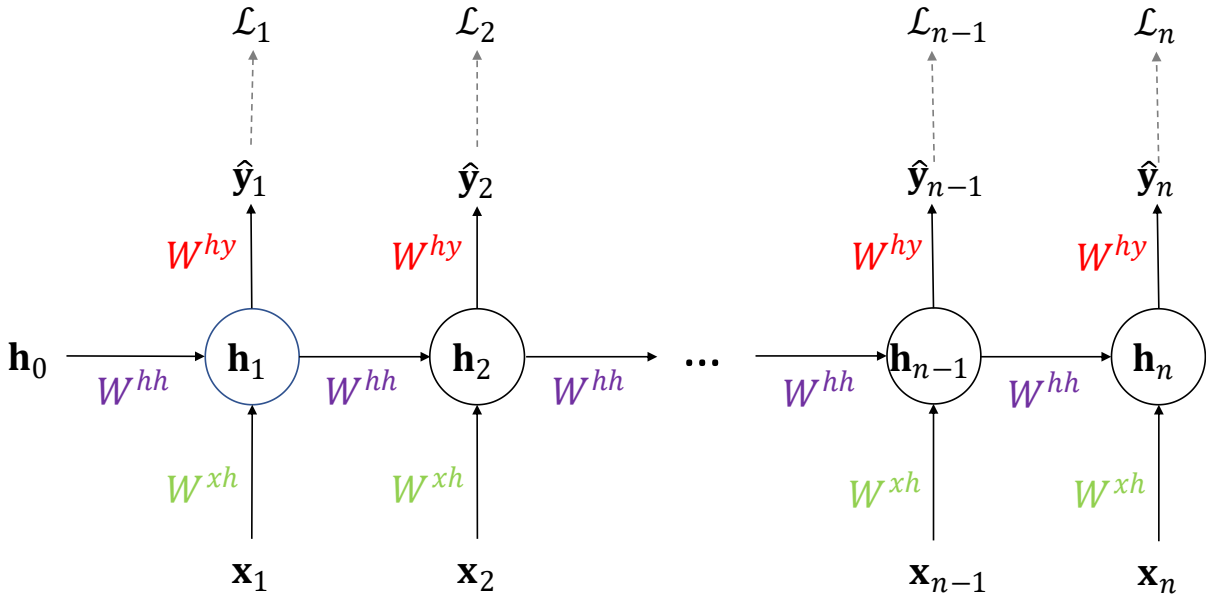
In total:

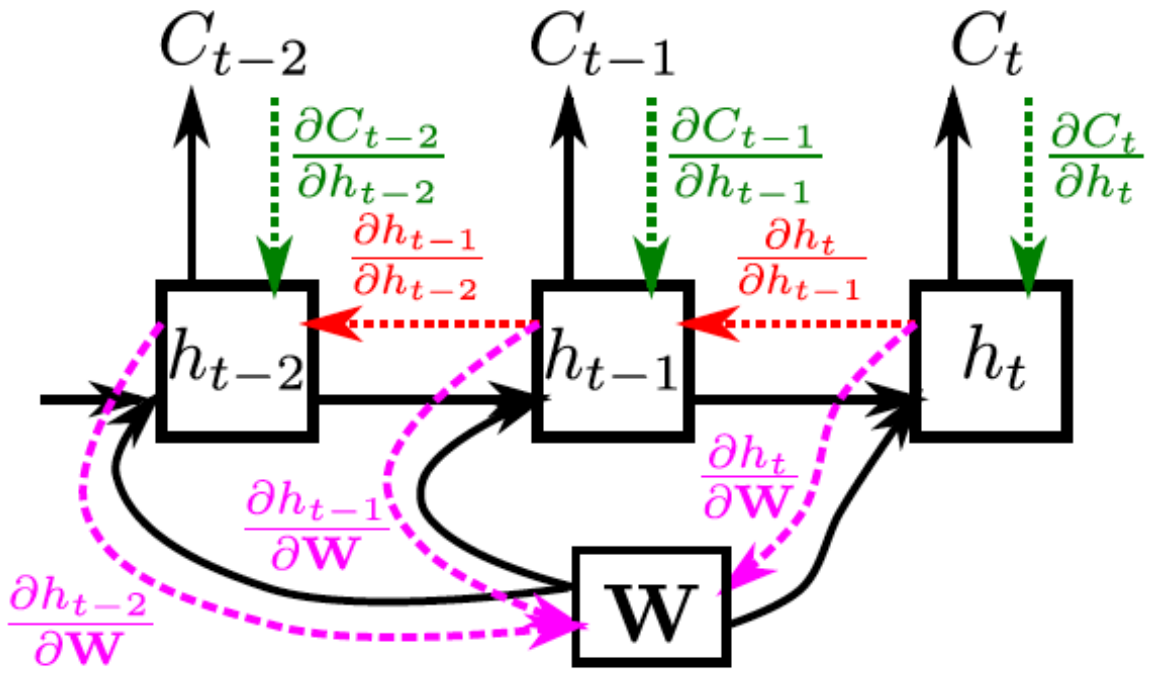
$$\mathcal{L} = \sum_t \mathcal{L}_t$$

$$\frac{\partial \mathcal{L}}{\partial W^{hh}} = ?$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{h}_n}\frac{\partial \mathbf{h}_n}{\partial W^{hh}} + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{n-1}}\frac{\partial \mathbf{h}_{n-1}}{\partial W^{hh}} + \cdots + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial W^{hh}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_t}\frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t+1}}\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$$





$$\frac{\partial C_t}{\partial \mathbf{W}} = \sum_{t'=1}^{t} \frac{\partial C_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t'}} \frac{\partial h_{t'}}{\partial \mathbf{W}}, \text{ where } \frac{\partial h_t}{\partial h_{t'}} = \prod_{k=t'+1}^{t} \frac{\partial h_k}{\partial h_{k-1}}$$

Backpropagation through Vanilla RNN

The Vanilla RNN Model

In general:

$$\mathbf{h}_t = tanh\left(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1}\right)$$

$$\hat{\mathbf{y}}_t = softmax\left(W^{hy} \cdot \mathbf{h}_t\right)$$

$$\mathcal{L}_t = CE(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

In total:

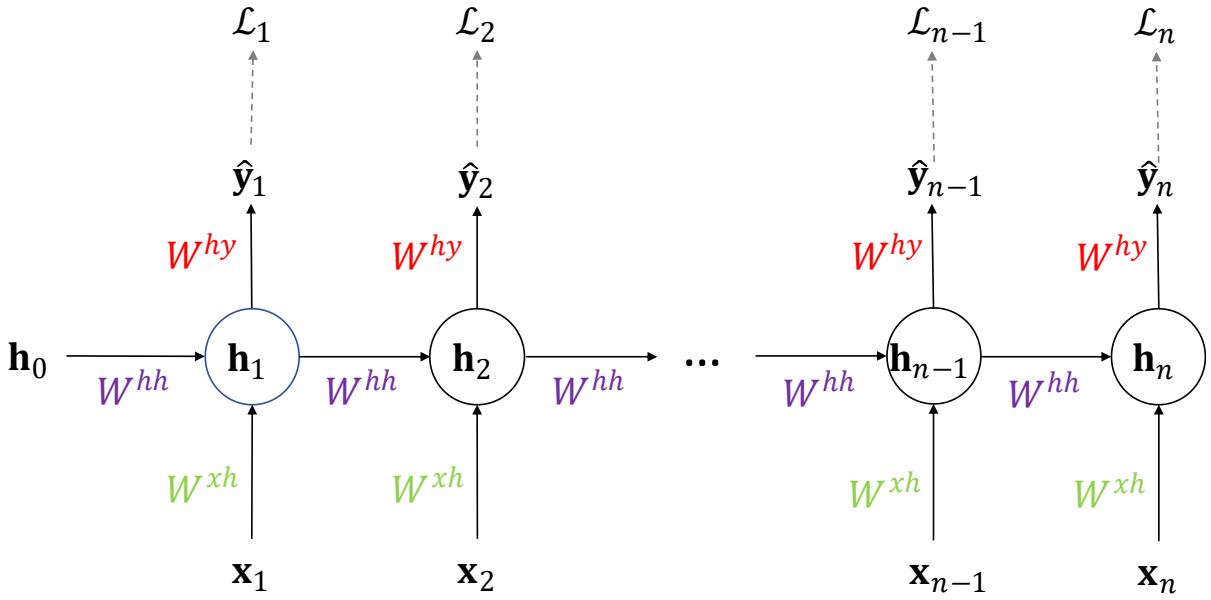
$$\mathcal{L} = \sum_t \mathcal{L}_t$$

$$\frac{\partial \mathcal{L}}{\partial W^{xh}} = ?$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{h}_n}\frac{\partial \mathbf{h}_n}{\partial W^{xh}} + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{n-1}}\frac{\partial \mathbf{h}_{n-1}}{\partial W^{xh}} + \cdots + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial W^{xh}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}_t}\frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} + \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{t+1}}\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$$

(calculated before)



Initial hidden state

- We need to specify the initial activity state of all the hidden units.
- We could just fix these initial states to have some default value like 0.5.
- But it is better to treat the initial states as learned parameters.
- We learn them in the same way as we learn the weights.
  - Start off with an initial random guess for the initial states.
  - At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state.
  - Adjust the initial states by following the negative gradient.

Slide: Hinton

Initializing parameters

- Since an unfolded RNN is a deep MLP, we can use Xavier initialization.

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
  - If the weights are small, the gradients shrink exponentially.
  - If the weights are big the gradients grow exponentially.
- Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers.

- In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.
  - So RNNs have difficulty dealing with long-range dependencies.

Slide: Hinton

Exploding and vanishing gradients problem

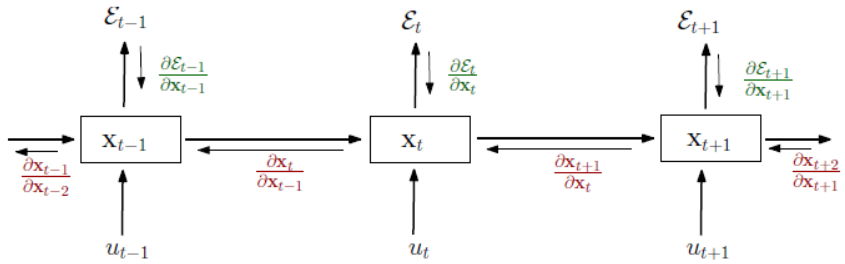
- Solution 1: Gradient clipping for exploding gradients:

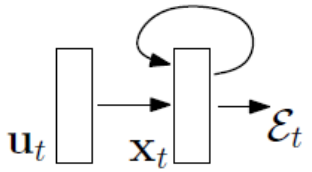
Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

- For vanishing gradients: Regularization term that penalizes changes in the magnitudes of back-propagated gradients

$$\Omega = \sum_k \Omega_k = \sum_k \left(\frac{\left\| \frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}} \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k} \right\|}{\left\| \frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}} \right\|} - 1 \right)^2$$





On the difficulty of training recurrent neural networks

Razvan Pascanu PASCANUR@IRO.UMONTREAL.CA
Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

Tomas Mikolov T.MIKOLOV@GMAIL.COM
Speech@FIT, Brno University of Technology, Brno, Czech Republic

Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

2012

Exploding and vanishing gradients problem

- Solution 2:
  - Use methods like LSTM

Long Short Term Memory (LSTM)

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carrousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.

RNN

- Basic block diagram

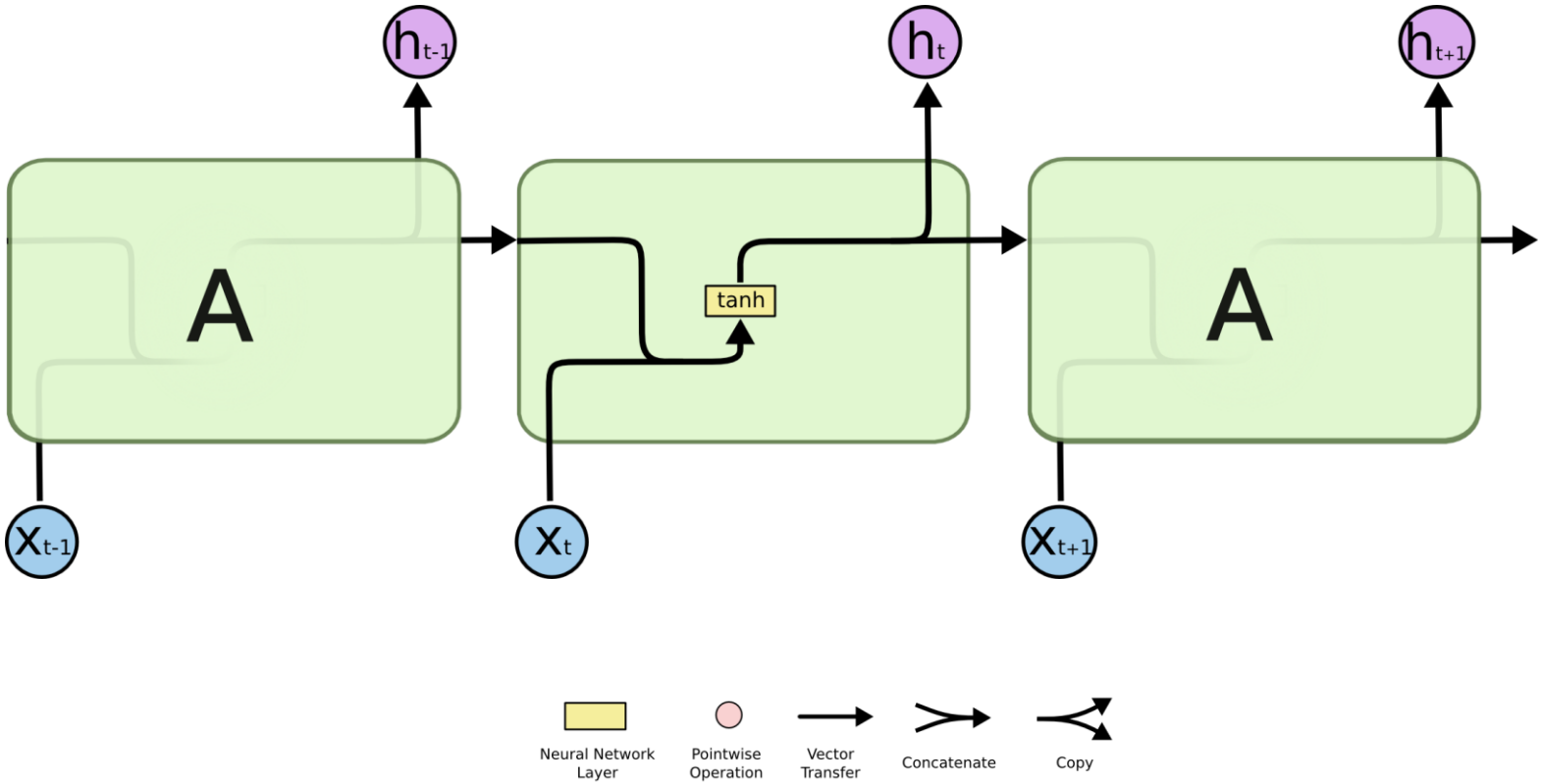


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Key Problem

- Learning long-term dependencies is hard

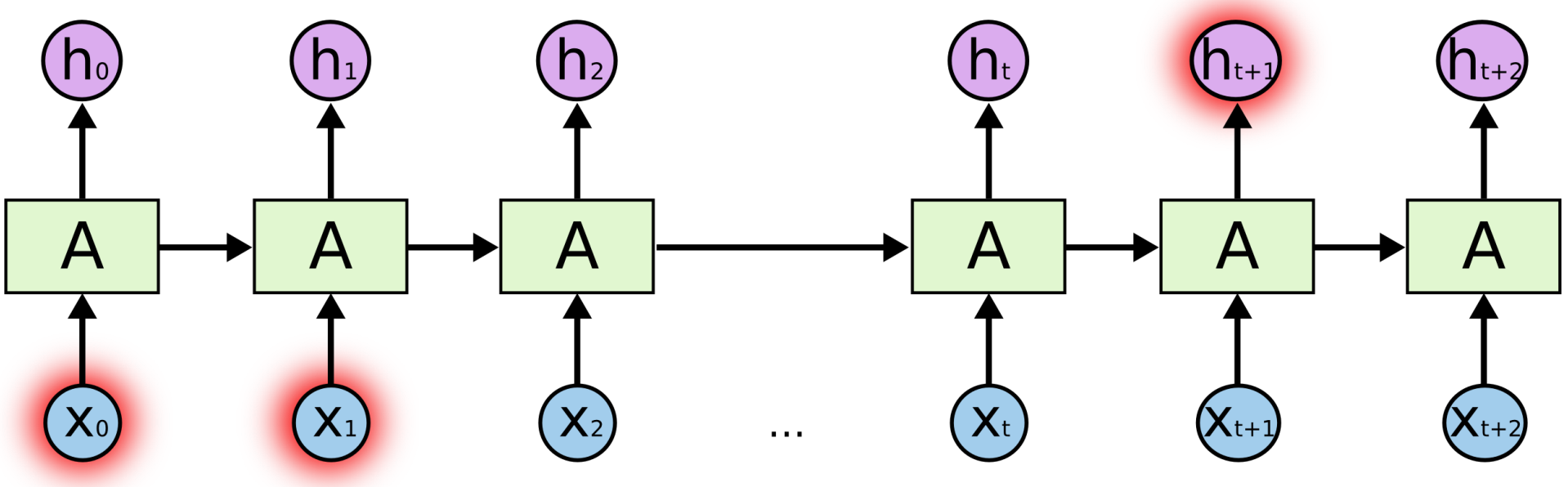


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).
- They designed a memory cell using logistic and linear units with multiplicative interactions.
- Information gets into the cell whenever its “write” gate is on.
- The information stays in the cell so long as its “keep” gate is on.
- Information can be read from the cell by turning on its “read” gate.

Slide: Hinton

Meet LSTMs

- How about we explicitly encode memory?

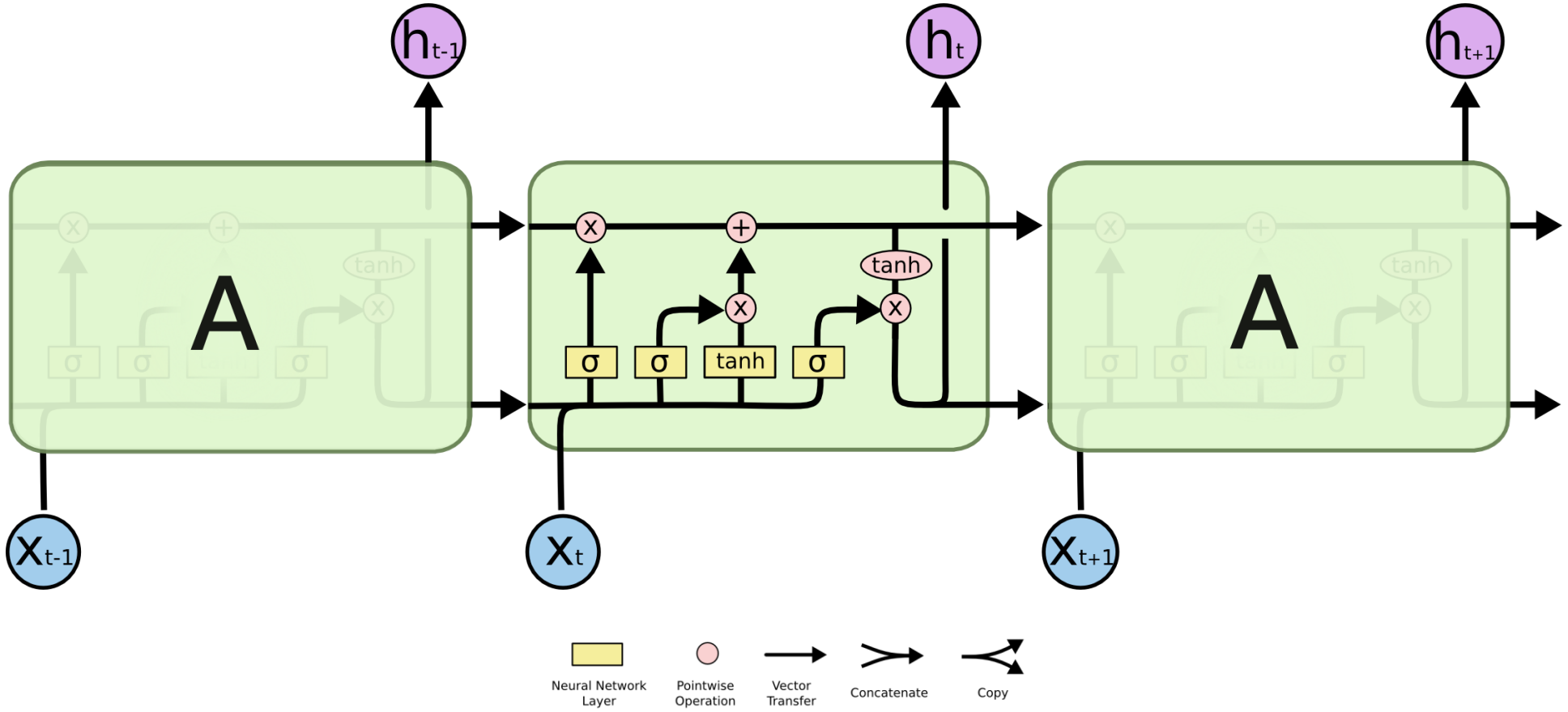


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM in detail

- We first compute an activation vector, a:

$$a = W_x x_t + W_h h_{t-1} + b$$

- Split this into four vectors of the same size:

$$a_i, a_f, a_o, a_g \leftarrow a$$

- We then compute the values of the gates:

$$i = \sigma(a_i) \quad f = \sigma(a_f) \quad o = \sigma(a_o) \quad g = \tanh(a_g)$$

where σ is the sigmoid.

- The next cell state c_t and the hidden state h_t:

$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

where $\odot$ is the element-wise product of vectors

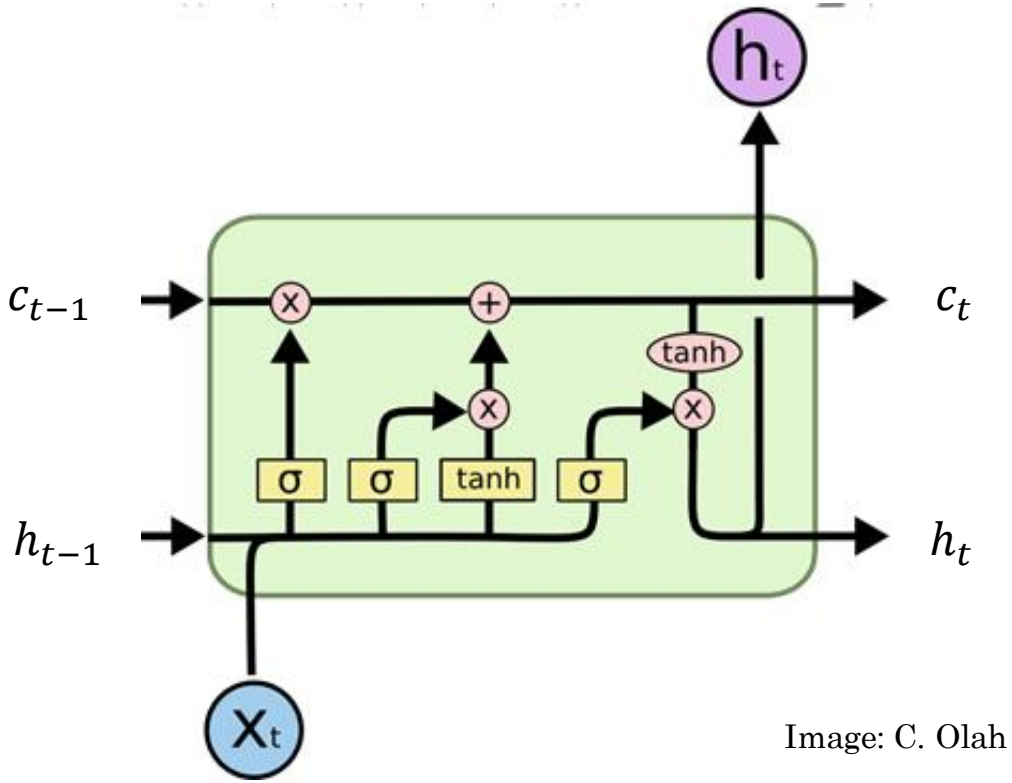


Image: C. Olah

Alternative formulation:

$$i_t = g(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$f_t = g(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$o_t = g(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

Eqs: Karpathy

LSTMs Intuition: Memory

- Cell State / Memory

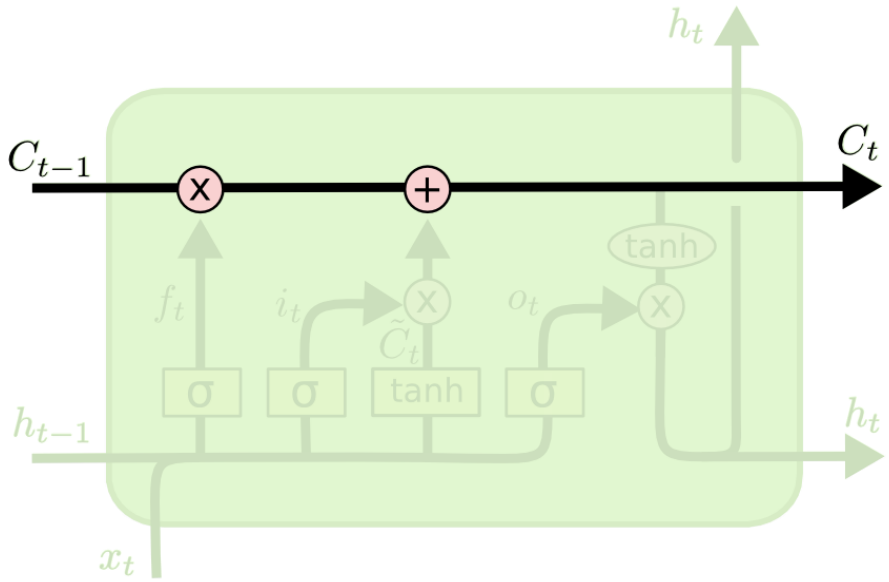
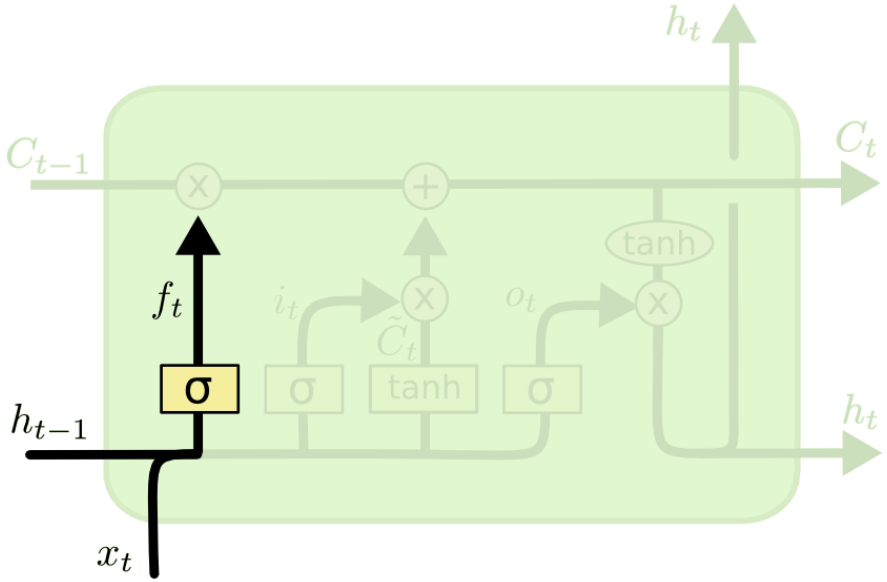


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Forget Gate

- Should we continue to remember this “bit” of information or not?

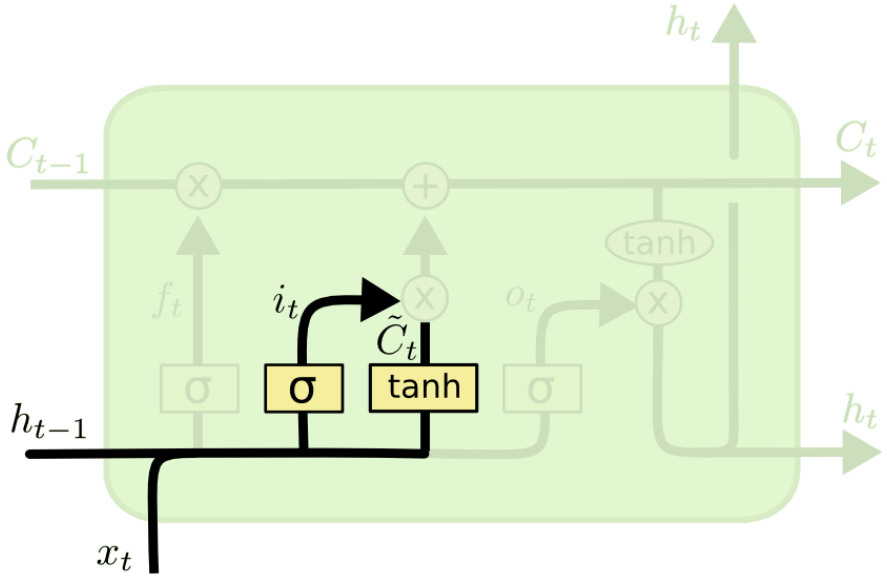


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Input Gate

- Should we update this "bit" of information or not?
  - If so, with what?



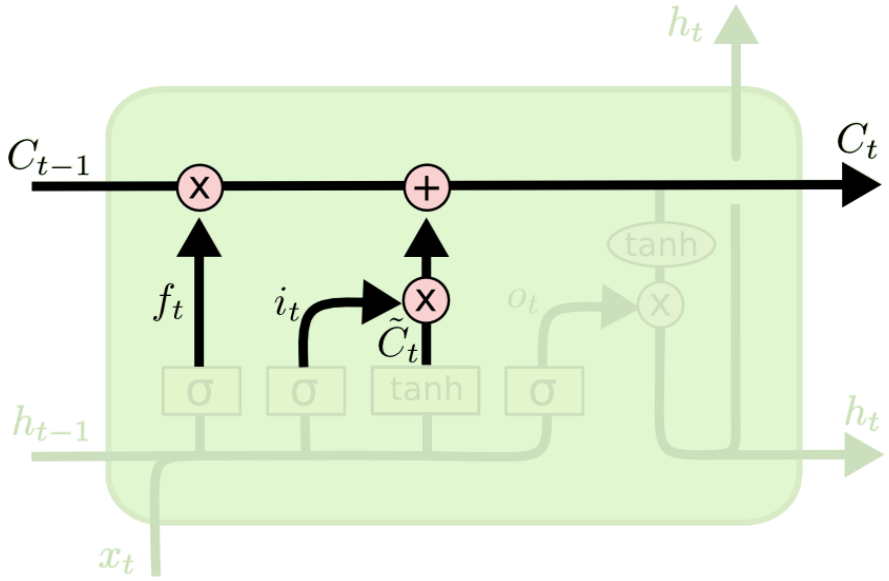
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Memory Update

- Forget that + memorize this

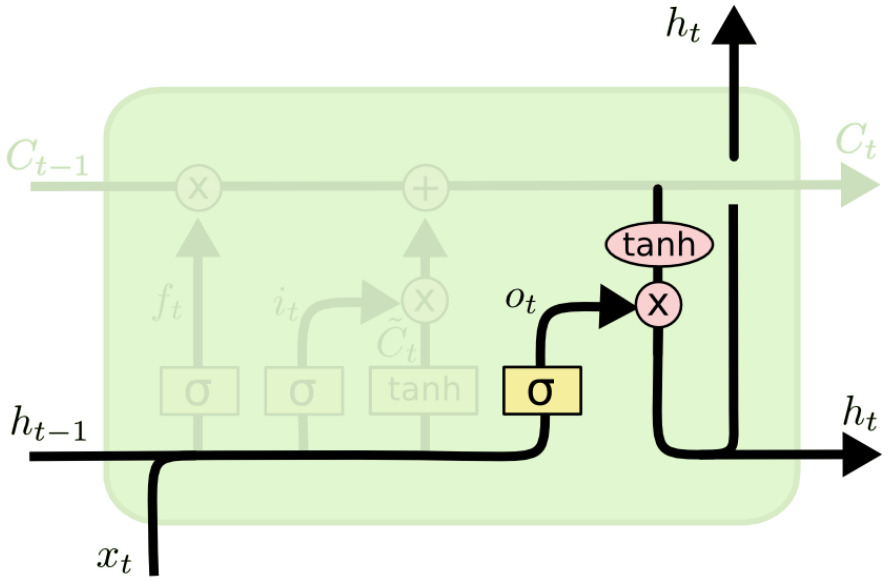


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Output Gate

- Should we output this “bit” of information to “deeper” layers?



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs

- A pretty sophisticated cell

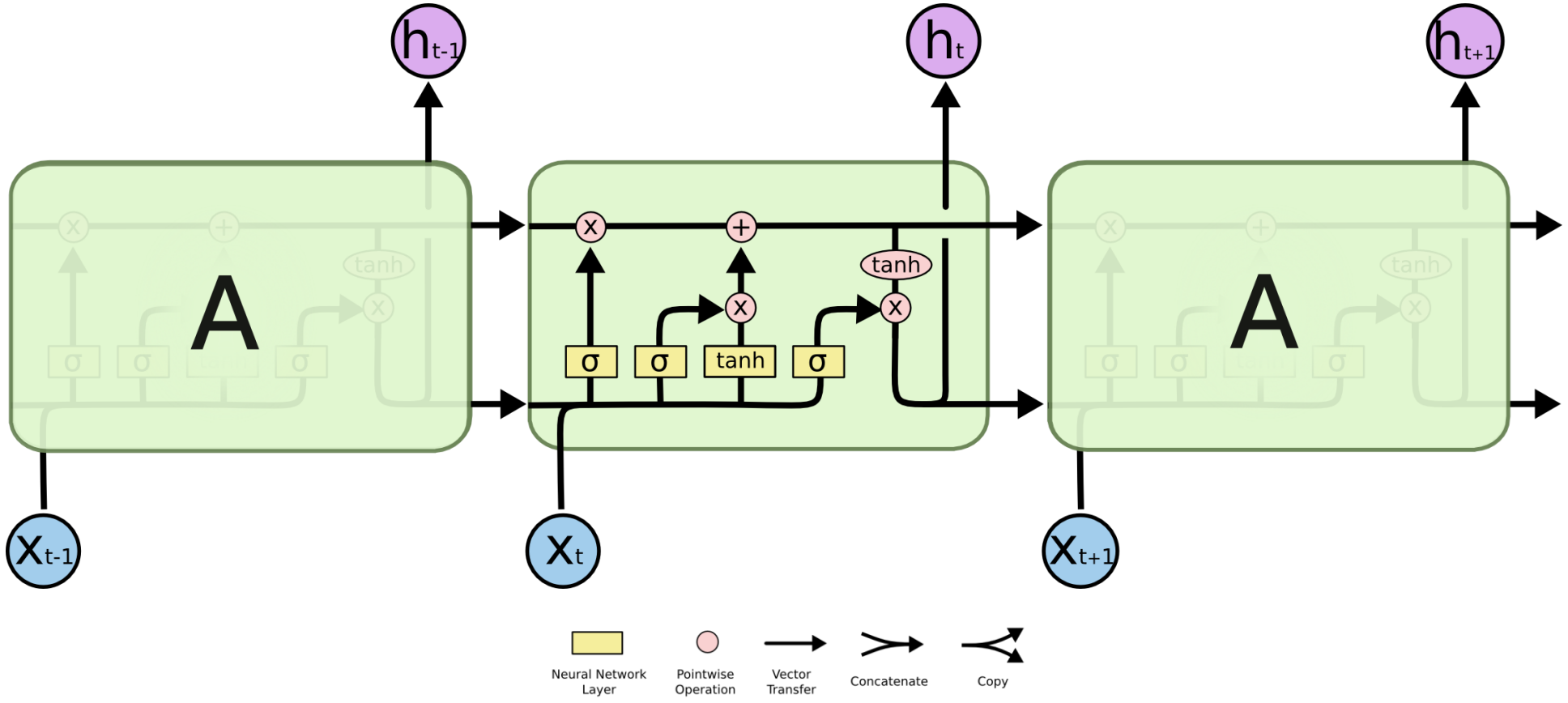
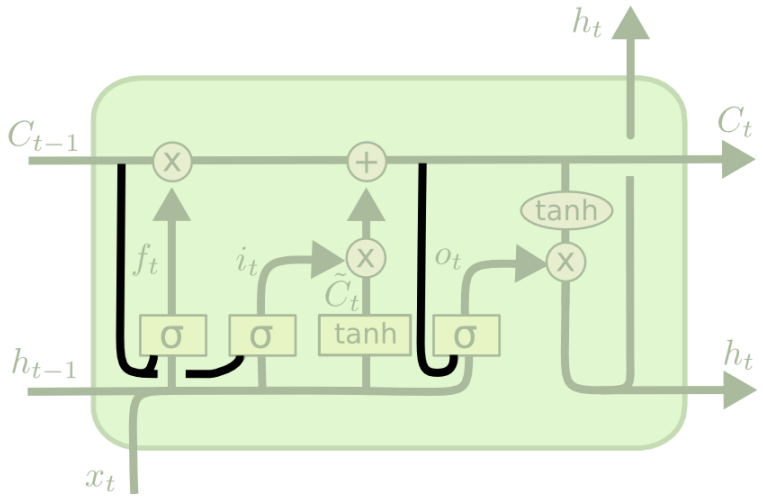


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM Variants #1: Peephole Connections

- Let gates see the cell state / memory

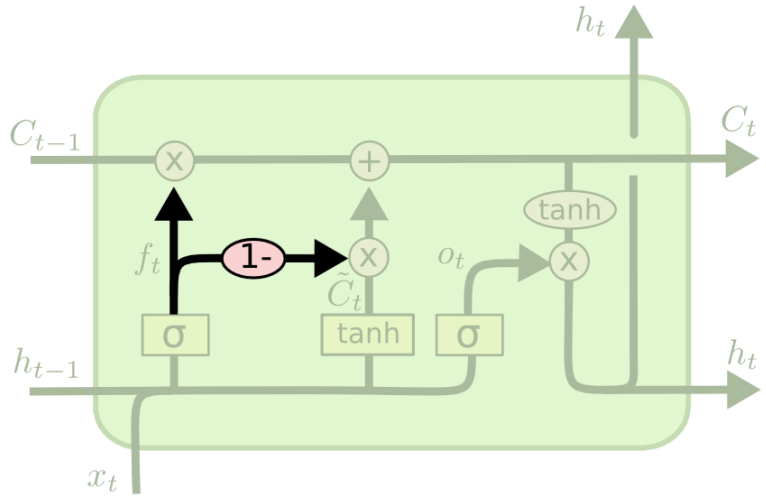


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM Variants #2: Coupled Gates

- Only memorize new if forgetting old

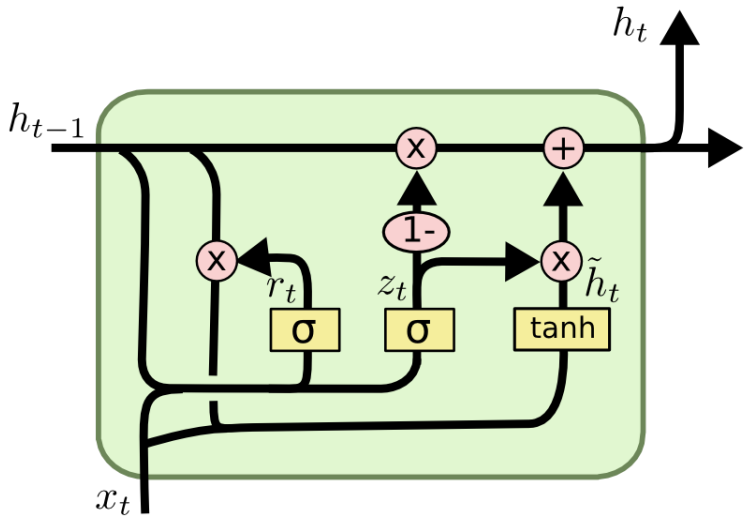


$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM Variants #3: Gated Recurrent Units

- Changes:
  - No explicit memory; memory = hidden output
  - Z = memorize new and forget old



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM vs. GRU

Accepted as a short paper in ACL 2018

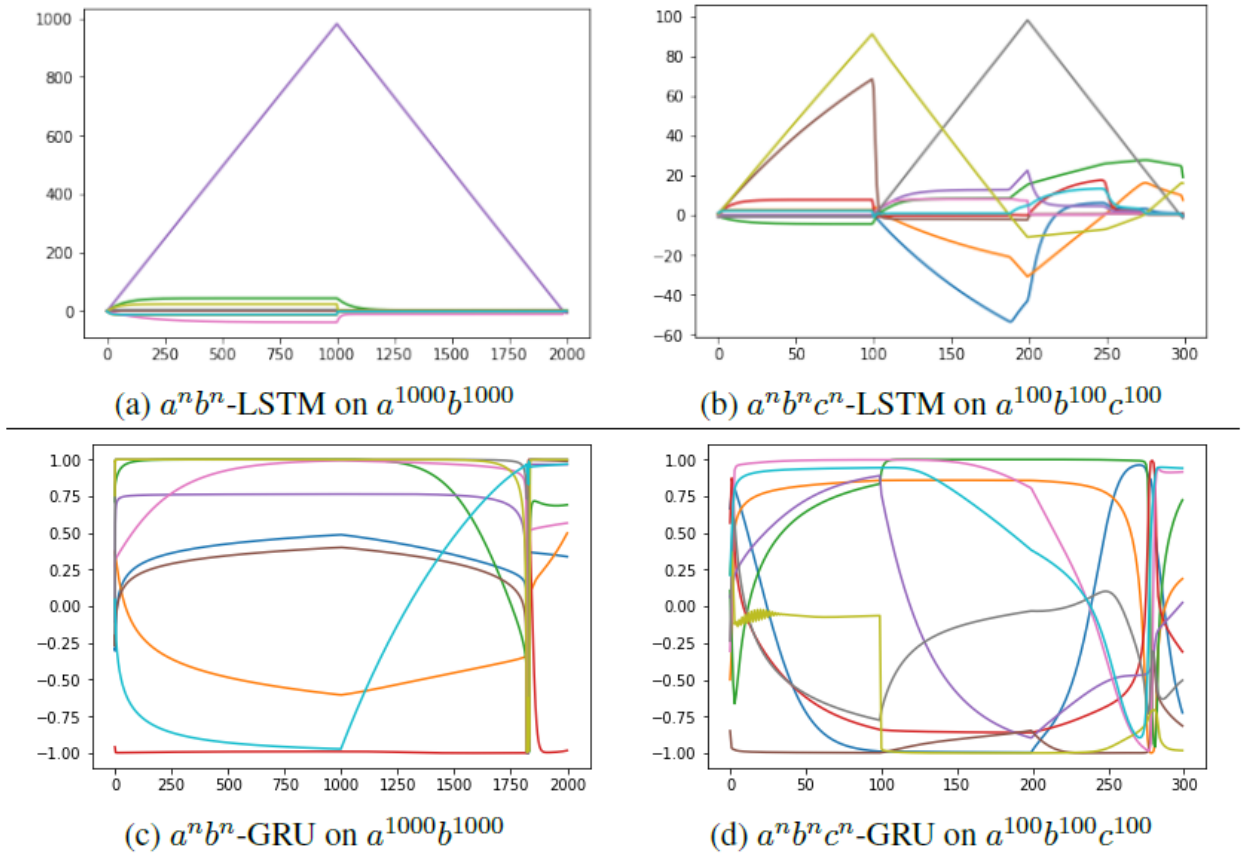
On the Practical Computational Power of Finite Precision RNNs for Language Recognition

Gail Weiss
Technion, Israel

Yoav Goldberg
Bar-Ilan University, Israel

Eran Yahav
Technion, Israel

{sgailw,yahave}@cs.technion.ac.il
yogo@cs.biu.ac.il



(a) $a^n b^n$-LSTM on $a^{1000} b^{1000}$

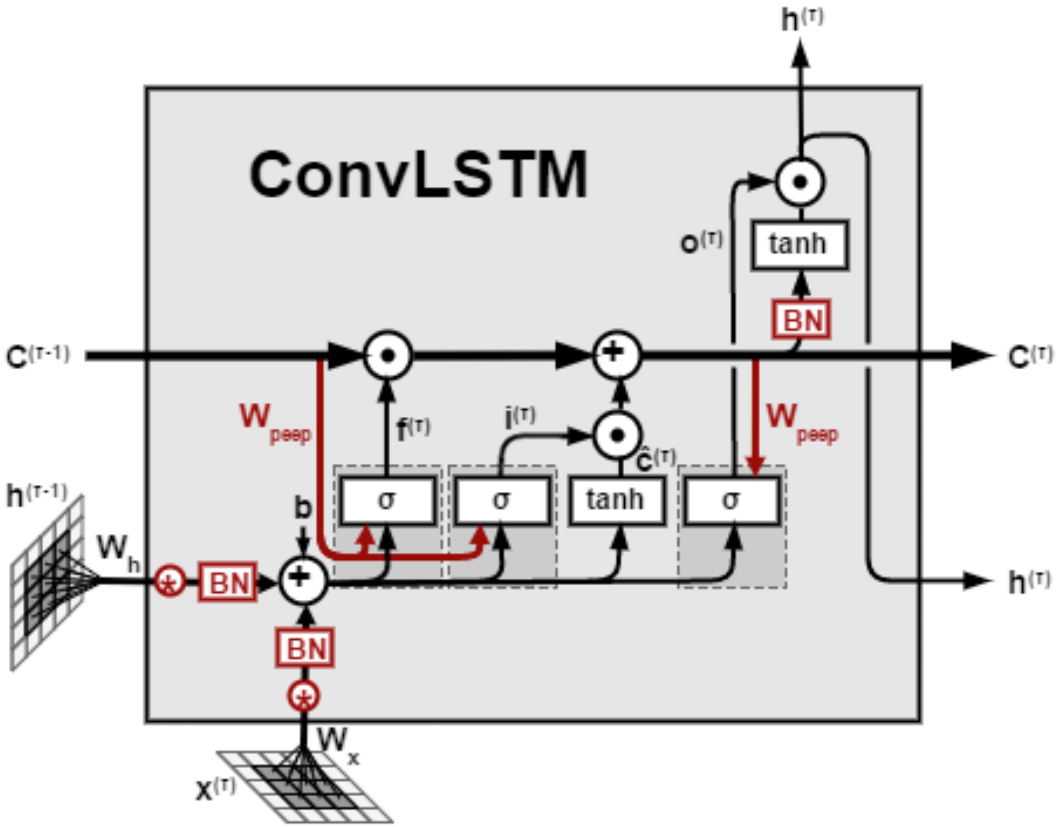
(b) $a^n b^n c^n$-LSTM on $a^{100} b^{100} c^{100}$

(c) $a^n b^n$-GRU on $a^{1000} b^{1000}$

(d) $a^n b^n c^n$-GRU on $a^{100} b^{100} c^{100}$

Figure 1: Activations — c for LSTM and h for GRU — for networks trained on $a^n b^n$ and $a^n b^n c^n$. The LSTM has clearly learned to use an explicit counting mechanism, in contrast with the GRU.

ConvLSTM



https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting

Xingjian Shi **Zhourong Chen** **Hao Wang** **Dit-Yan Yeung**
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
{xshiab,zchenbb,hwangaz,dyyeung}@cse.ust.hk

Wai-kin Wong **Wang-chun Woo**
Hong Kong Observatory
Hong Kong, China
{wkwong,wcwoo}@hko.gov.hk

2015

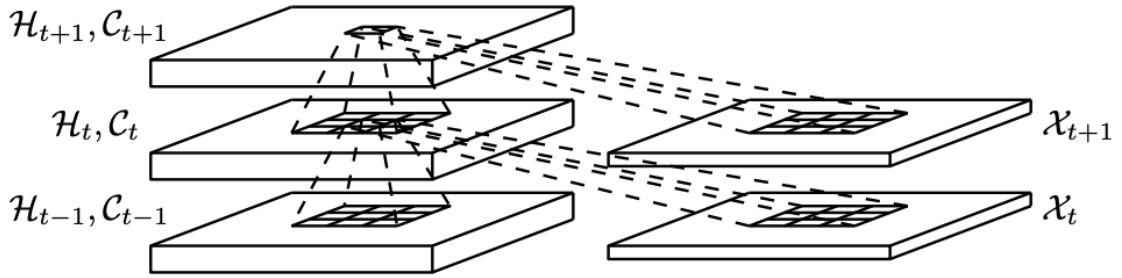


Figure 2: Inner structure of ConvLSTM

Reference

- A very detailed explanation with nice figures

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

CNN vs RNN

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

Shaojie Bai [1] J. Zico Kolter [2] Vladlen Koltun [3]

Abstract

For most deep learning practitioners, sequence modeling is synonymous with recurrent networks. Yet recent results indicate that convolutional architectures can outperform recurrent networks on tasks such as audio synthesis and machine translation. Given a new sequence modeling task or dataset, which architecture should one use? We conduct a systematic evaluation of generic convolutional and recurrent architectures for sequence

chine translation (van den Oord et al., 2016; Kalchbrenner et al., 2016; Dauphin et al., 2017; Gehring et al., 2017a;b). This raises the question of whether these successes of convolutional sequence modeling are confined to specific application domains or whether a broader reconsideration of the association between sequence processing and recurrent networks is in order.

We address this question by conducting a systematic empirical evaluation of convolutional and recurrent architectures

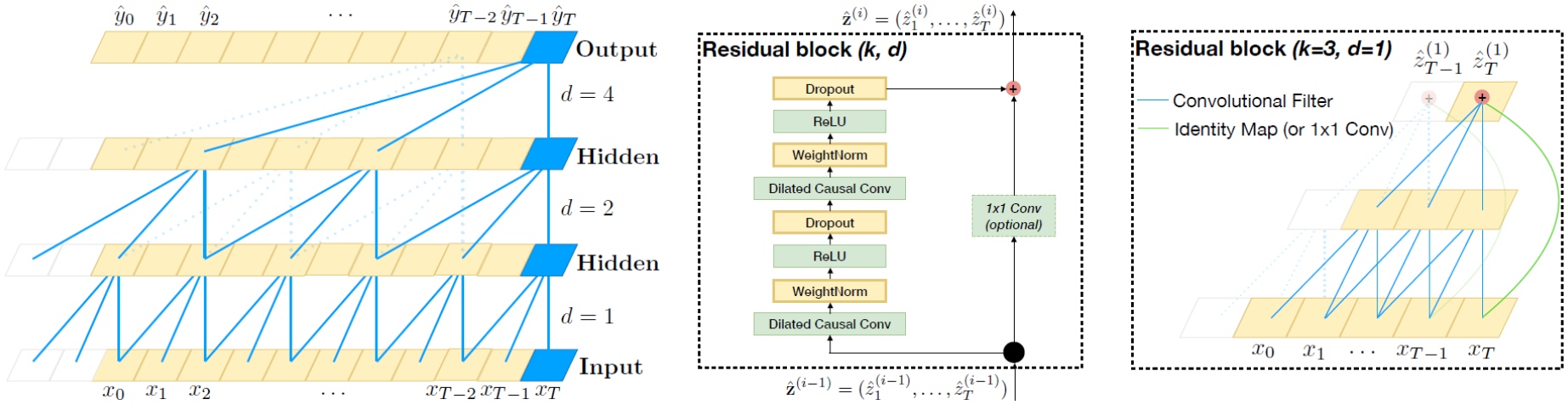


Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

Example: Character-level Text Modeling

Character-level Text Modeling

- Problem definition: Find c_{n+1} given $c_1, c_2, \ldots, c_n$.
- Modelling:

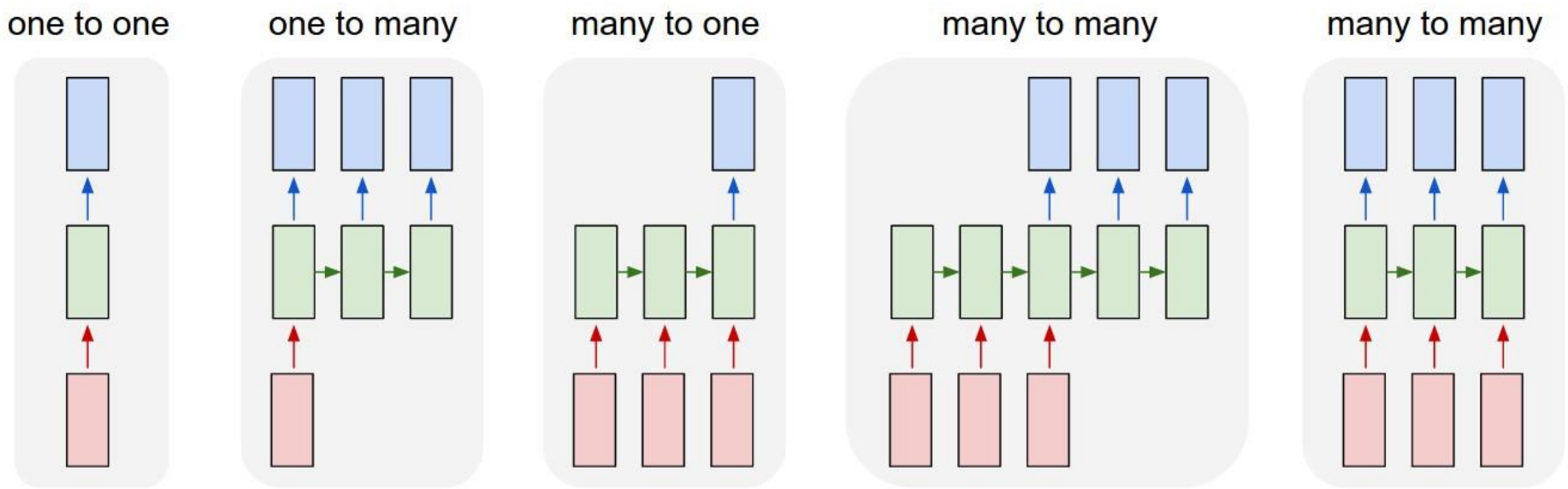
$$p(c_{n+1} \mid c_n, \ldots, c_1)$$

- In general, we just take the last N characters:

$$p(c_{n+1} \mid c_n, \ldots, c_{n-(N-1)})$$

- Learn $p(c_{n+1} = 'a' \mid 'Ankar')$ from data such that

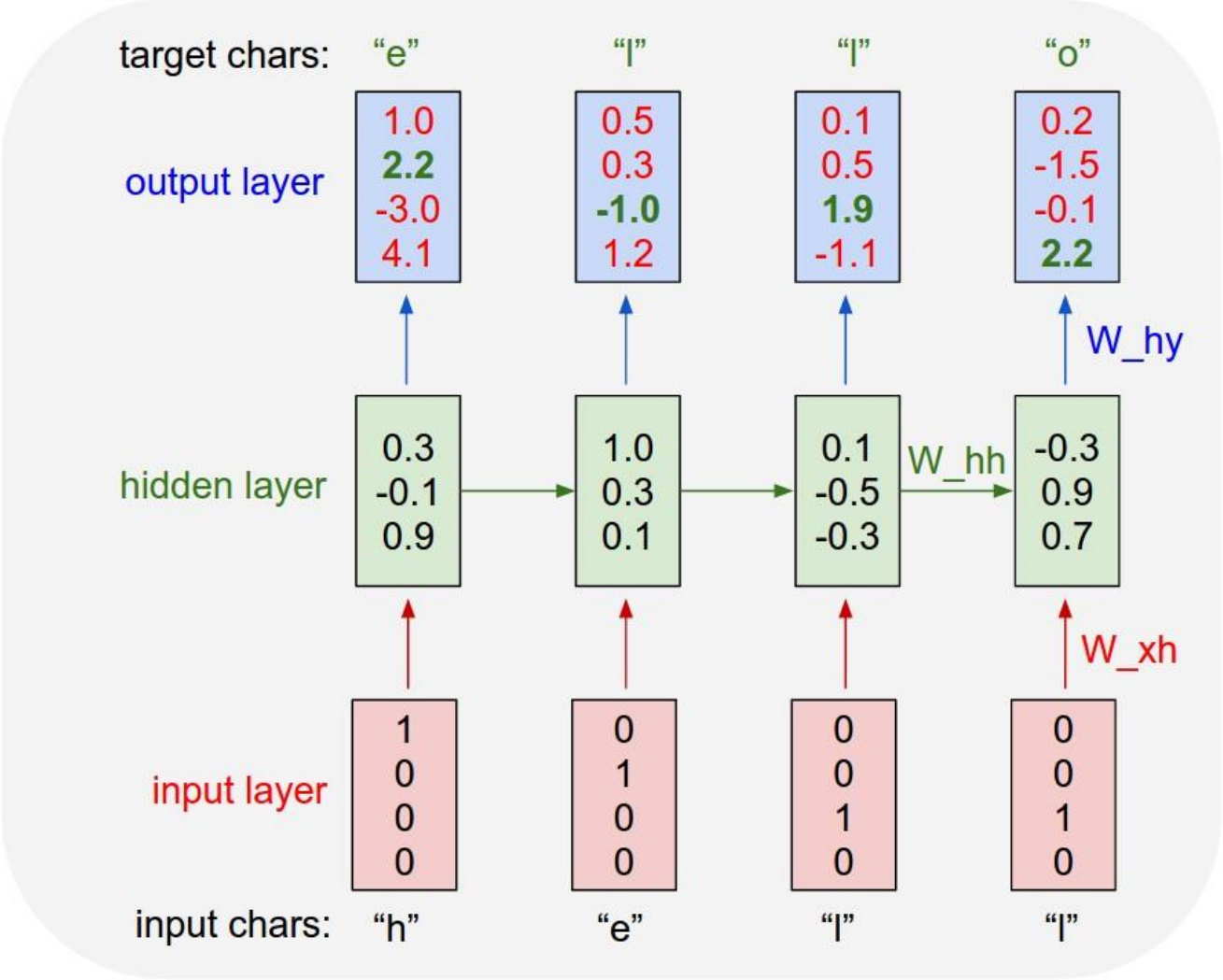
$$p(c_{n+1} = 'a' \mid 'Ankar') > p(c_{n+1} = 'o' \mid 'Ankar')$$



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

A simple scenario

- Alphabet: h, e, l, o
- Text to train to predict: “hello”



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

Sampling: Greedy

- Greedy sampling: Take the most likely word at each step

```python
from numpy import array
from numpy import argmax

# greedy decoder
def greedy_decoder(data):
    # index for largest probability each row
    return [argmax(s) for s in data]

# define a sequence of 10 words over a vocab of 5 words
data = [[0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1]]
data = array(data)
# decode sequence
result = greedy_decoder(data)
print(result)
```

Running the example outputs a sequence of integers that could then be mapped back to words in the vocabulary.

```
[4, 0, 4, 0, 4, 0, 4, 0, 4, 0]
```

Code: https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/

Sampling: Beam Search

- What happens if we want *k* most likely sequences instead of one?
- Beam search: Consider *k* most likely words at each step, and expand search.

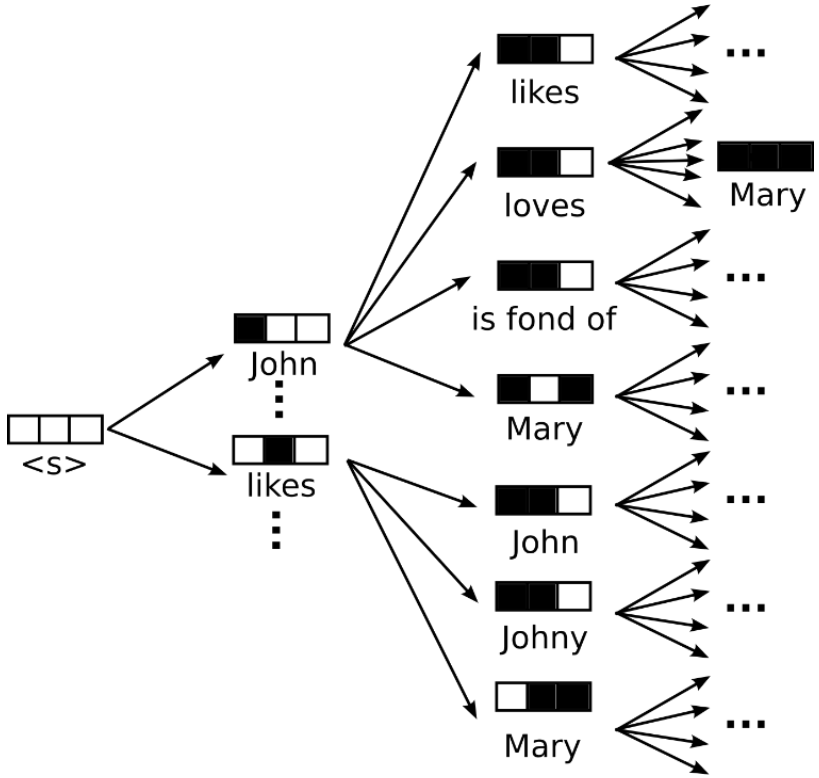


Figure: http://mttalks.ufal.ms.mff.cuni.cz/index.php

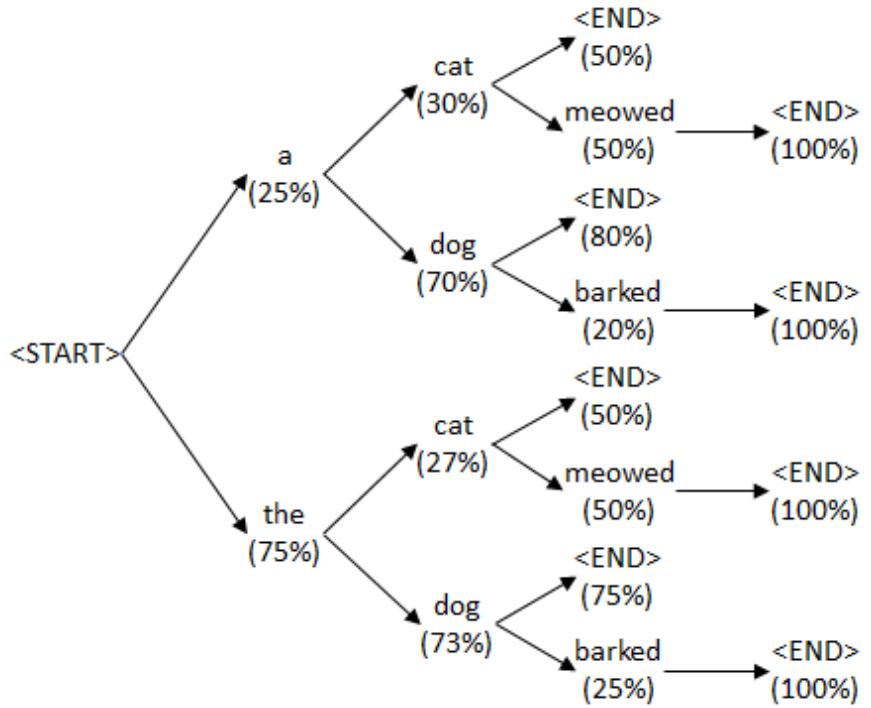


Figure: https://geekyisawesome.blogspot.com.tr/2016/10/using-beam-search-to-generate-most.html

Sampling: Beam Search

- Beam search: Consider *k* most likely words at each step, and expand search.

  (take log for numerical stability; take –log() for minimizing the score)

```python
from math import log
from numpy import array
from numpy import argmax

# beam search
def beam_search_decoder(data, k):
    sequences = [[list(), 0.0]]
    # walk over each step in sequence
    for row in data:
        all_candidates = list()
        # expand each current candidate
        for i in range(len(sequences)):
            seq, score = sequences[i]
            for j in range(len(row)):
                candidate = [seq + [j], score - log(row[j])]
                all_candidates.append(candidate)
        # order all candidates by score
        ordered = sorted(all_candidates, key=lambda tup:tup[1])
        # select k best
        sequences = ordered[:k]
    return sequences
```

```python
# define a sequence of 10 words over a vocab of 5 words
data = [[0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1],
        [0.1, 0.2, 0.3, 0.4, 0.5],
        [0.5, 0.4, 0.3, 0.2, 0.1]]
data = array(data)
# decode sequence
result = beam_search_decoder(data, 3)
# print result
for seq in result:
    print(seq)
```

```
[[4, 0, 4, 0, 4, 0, 4, 0, 4, 0], 6.931471805599453]
[[4, 0, 4, 0, 4, 0, 4, 0, 4, 1], 7.154615356913663]
[[4, 0, 4, 0, 4, 0, 4, 0, 3, 0], 7.154615356913663]
```

Code: https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/

More on beam search

- Beam search is applied during inference.
- With modifications on the training procedure, it is possible to use it during training as well.

Sequence-to-Sequence Learning as Beam-Search Optimization

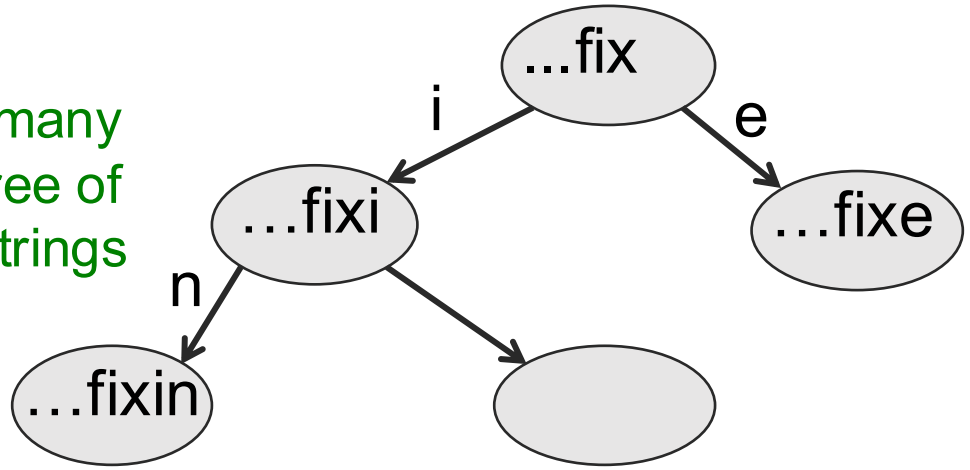
Sam Wiseman and **Alexander M. Rush**
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA, USA
{swiseman,srush}@seas.harvard.edu

2016

https://arxiv.org/abs/1606.02960

A sub-tree in the tree of all character strings

There are exponentially many nodes in the tree of all character strings of length N.



In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- If the nodes are implemented as hidden states in an RNN, different nodes can share structure because they use distributed representations.
- The next hidden representation needs to depend on the conjunction of the current character and the current hidden representation.

Slide: Hinton

Modeling text: Advantages of working with characters

- The web is composed of character strings.
- Any learning method powerful enough to understand the world by reading the web ought to find it trivial to learn which strings make words (this turns out to be true, as we shall see).
- Pre-processing text to get words is a big hassle
  - What about morphemes (prefixes, suffixes etc)
  - What about subtle effects like “sn” words?
  - What about New York?
  - What about Finnish?

ymmartamattomyydellansakaan

Slide: Hinton

Sample predictions
(when trained on the works of Shakespeare):

- 3-level RNN with 512 hidden nodes in each layer

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```


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

Sample predictions
(when trained on Wikipedia):

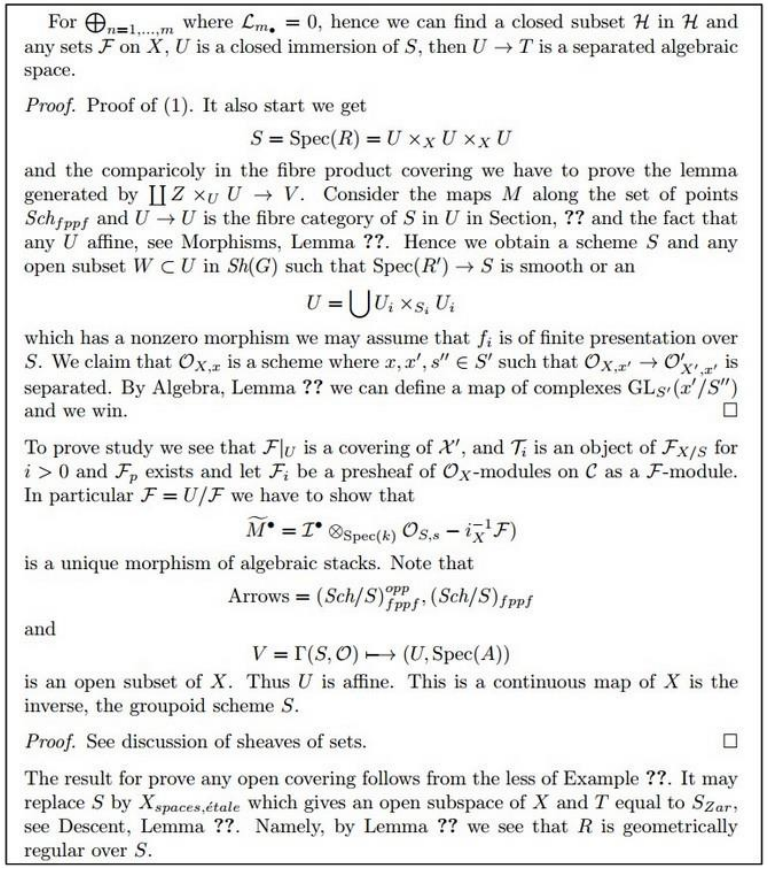
- Using LSTM

```
Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be said to be directly in Cantonese
Communication, which followed a ceremony and set inspired prison, training. The
emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest of India with the conflict.
Copyright was the succession of independence in the slop of Syrian influence that
was a famous German movement based on a more popular servicious, non-doctrinal
and sexual power post. Many governments recognize the military housing of the
[[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]],
that is sympathetic to be to the [[Punjab Resolution]]
(PJS)[http://www.humah.yahoo.com/guardian.
cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those Socialism's rule,
was starting to signing a major tripad of aid exile.]]
```


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

Sample predictions
(when trained on Latex documents):

- Using multi-layer LSTM



For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciy in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ??, and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation ??. Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

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

From Ilya Sutskever (using a variant of character-level RNN)

He was elected President during the Revolutionary War and forgave Opus Paul at Rome. The regime of his crew of England, is now Arab women's icons in and the demons that use something between the characters‘ sisters in lower coil trains were always operated on the line of the ephemerable street, respectively, the graphic or other facility for deformation of a given proportion of large segments at RTUS). The B every chord was a "strongly cold internal palette pour even the white blade.”

Slide: Hinton

Some completions produced by the model

- Sheila thrunges (most frequent)
- People thrunge (most frequent next character is space)
- Shiela, Thrungelini del Rey (first try)
- The meaning of life is literary recognition. (6^{th} try)

- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger.
  (one of the first 10 tries for a model trained for longer).

Slide: Hinton

What does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers.
- It is good at balancing quotes and brackets.
  - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what form this knowledge has.
  - Its syntactic knowledge is not modular.
- It knows a lot of weak semantic associations
  - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable.

Slide: Hinton

Example: Word-level Text Modeling

Word-level Text Modeling

- Problem definition: Find ω_{n+1} given $\omega_1, \omega_2, \ldots, \omega_n$.
- Modelling:

$$p(\omega_{n+1} \mid \omega_n, \ldots, \omega_1)$$

- In general, we just take the last N words:

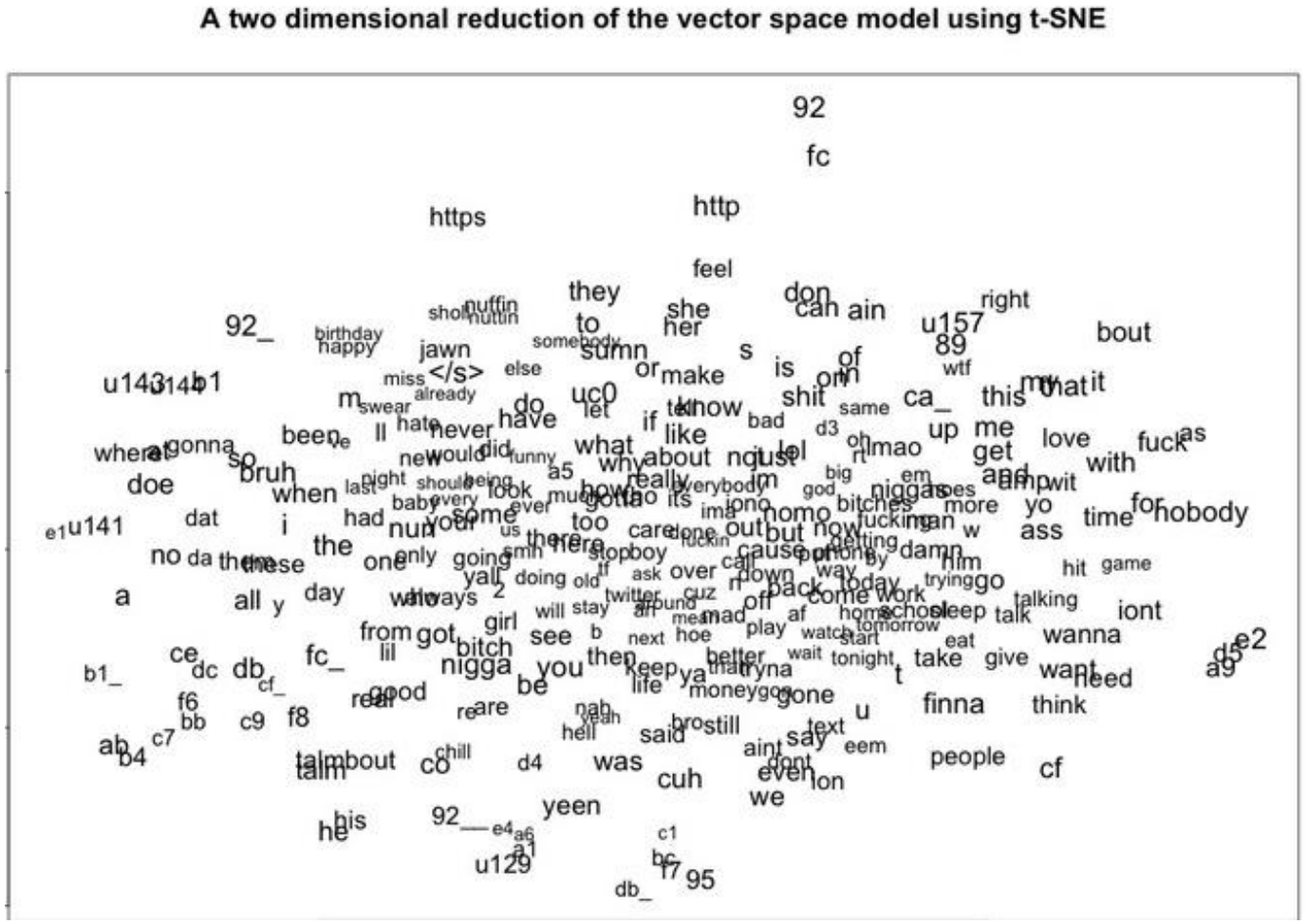
$$p(\omega_{n+1} \mid \omega_n, \ldots, \omega_{n-(N-1)})$$

- Learn $p(\omega_{n+1} = 'Turkey' \mid 'Ankara\ is\ the\ capital\ of\ ')$ from data such that:

$$p(\omega_{n+1} = 'Turkey' \mid 'Ankara\ is\ the\ capital\ of\ ') \quad > \quad p(\omega_{n+1} = 'UK' \mid 'Ankara\ is\ the\ capital\ of\ ')$$

A handicap

- The number of characters is low enough to handle without doing anything extra.
  - English has 26 characters.
- The situation is very different for words.
  - English has ~ 170,000 different words!
- This increases dimensionality and makes it difficult to capture “semantics”.
- Solution: Map words to a lower dimensional space, a.k.a. word embedding (word2vec).



Word Embedding (word2vec)

Fig: http://www.languagejones.com/blog-1/2015/11/1/word-embedding

Why do we embed words?

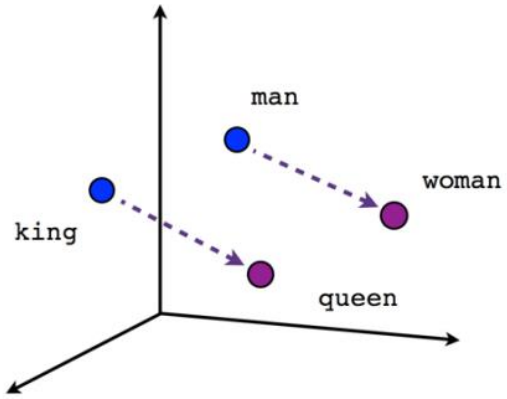
- 1-of-n encoding is not suitable to learn from
  - It is sparse
  - Similar words have different representations
  - Compare this with the pixel-based representation of images: Similar images/objects have similar pixels
- Embedding words in a map allows
  - Encoding them with fixed-length vectors
  - “Similar” words having similar representations
  - Allows complex reasoning between words:
    - king - man + woman = queen

EXPRESSION	NEAREST TOKEN
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montral Canadiens - Montreal + Toronto	Toronto Maple Leafs

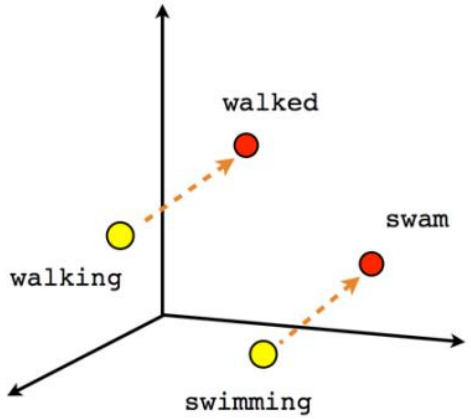
Table 1: Mikolov et al. [3] showcase simple additive properties of their word embeddings.

Table: https://devblogs.nvidia.com/parallelforall/understanding-natural-language-deep-neural-networks-using-torch/

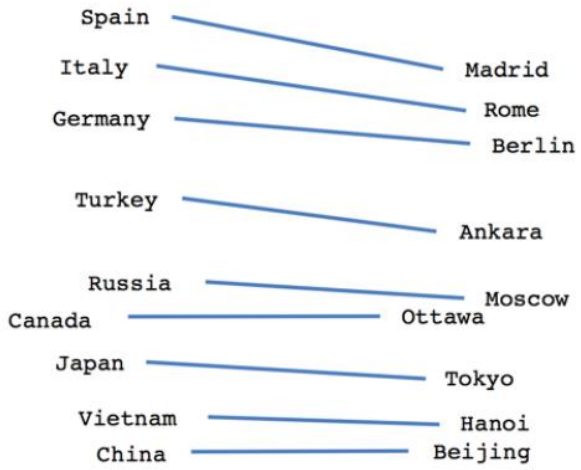
More examples



Male-Female



Verb tense



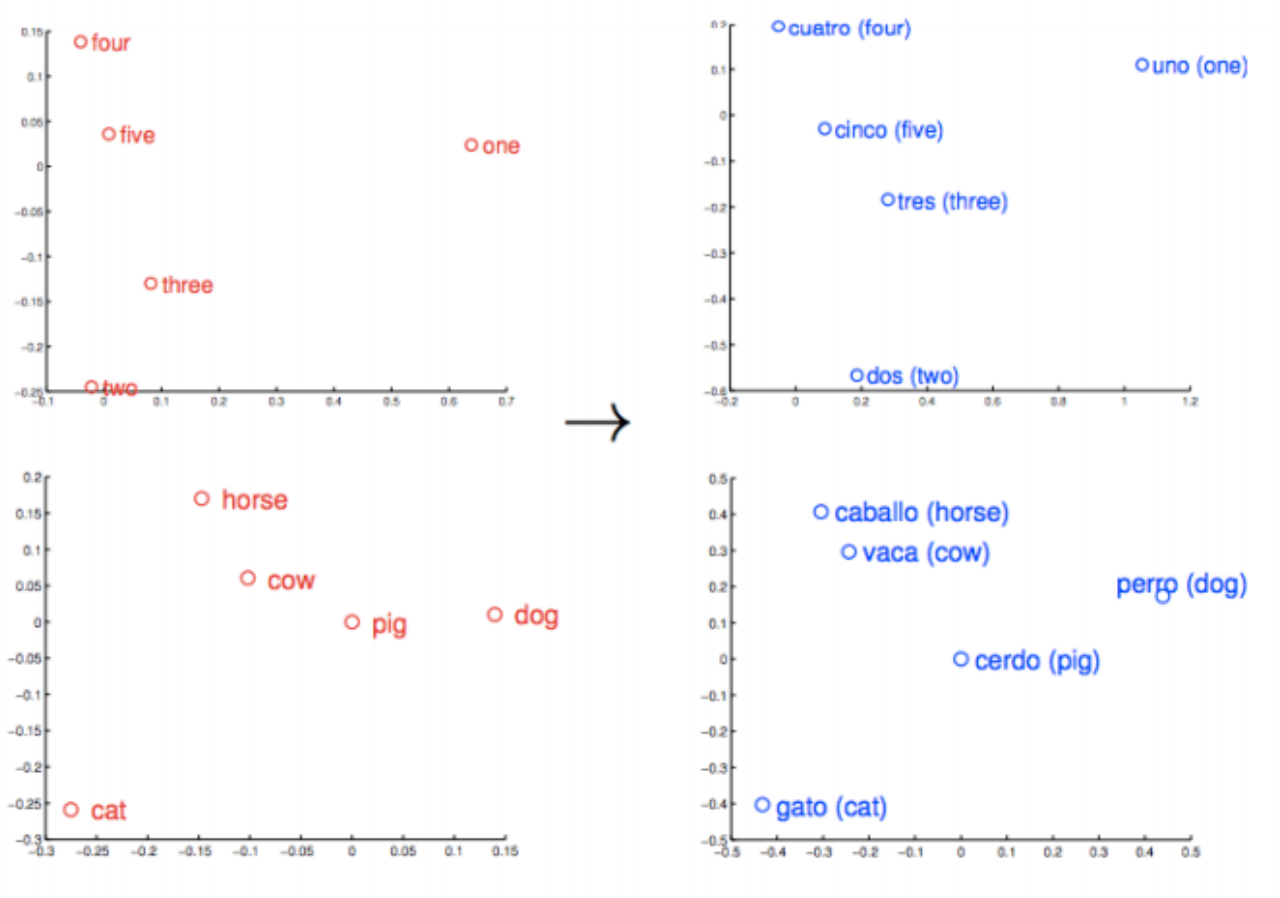
Country-Capital

More examples

- Geopolitics: *Iraq - Violence = Jordan*
- Distinction: *Human - Animal = Ethics*
- *President - Power = Prime Minister*
- *Library - Books = Hall*

http://deeplearning4j.org/word2vec

More examples



http://deeplearning4j.org/word2vec

word2vec

- “Similarity” to Sweden (cosine distance between their vector representations)

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

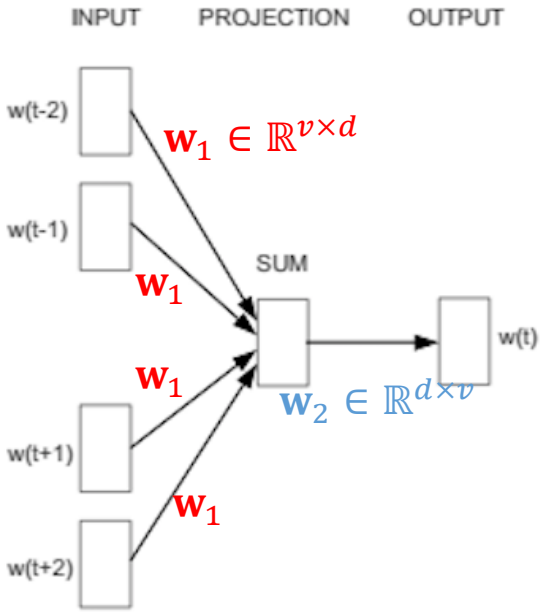
http://deeplearning4j.org/word2vec

Two different ways to train

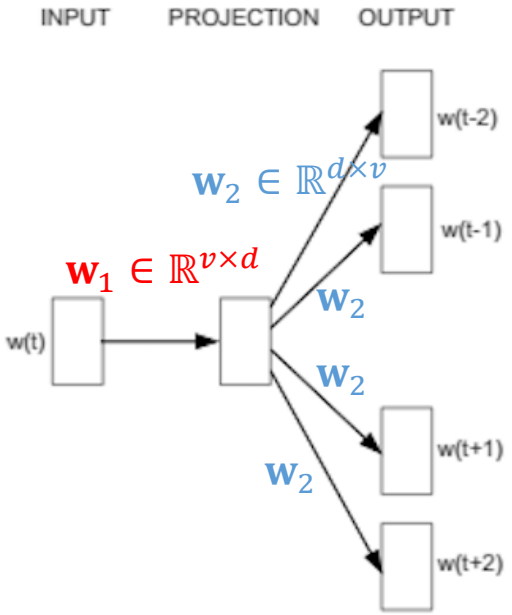
1. Using context to predict a target word (~ continuous bag-of-words)
2. Using word to predict a target context (skip-gram)

- If the vector for a word cannot predict the context, the mapping to the vector space is adjusted
- Since similar words should predict the same or similar contexts, their vector representations should end up being similar

v: vocabulary size
d: hidden dimension



CBOW

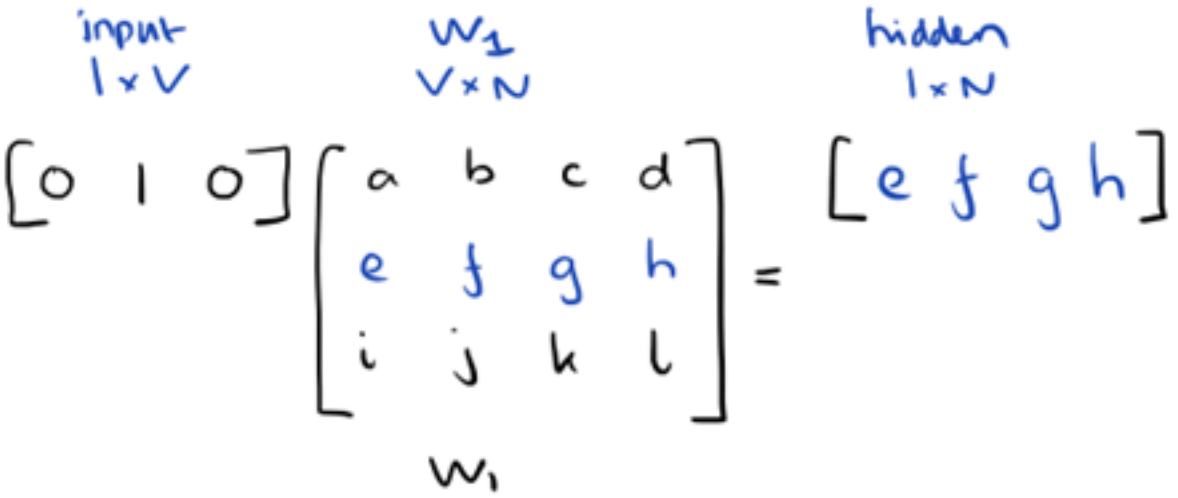


Skip-gram

http://deeplearning4j.org/word2vec

Note that the weight matrix is a look-up table

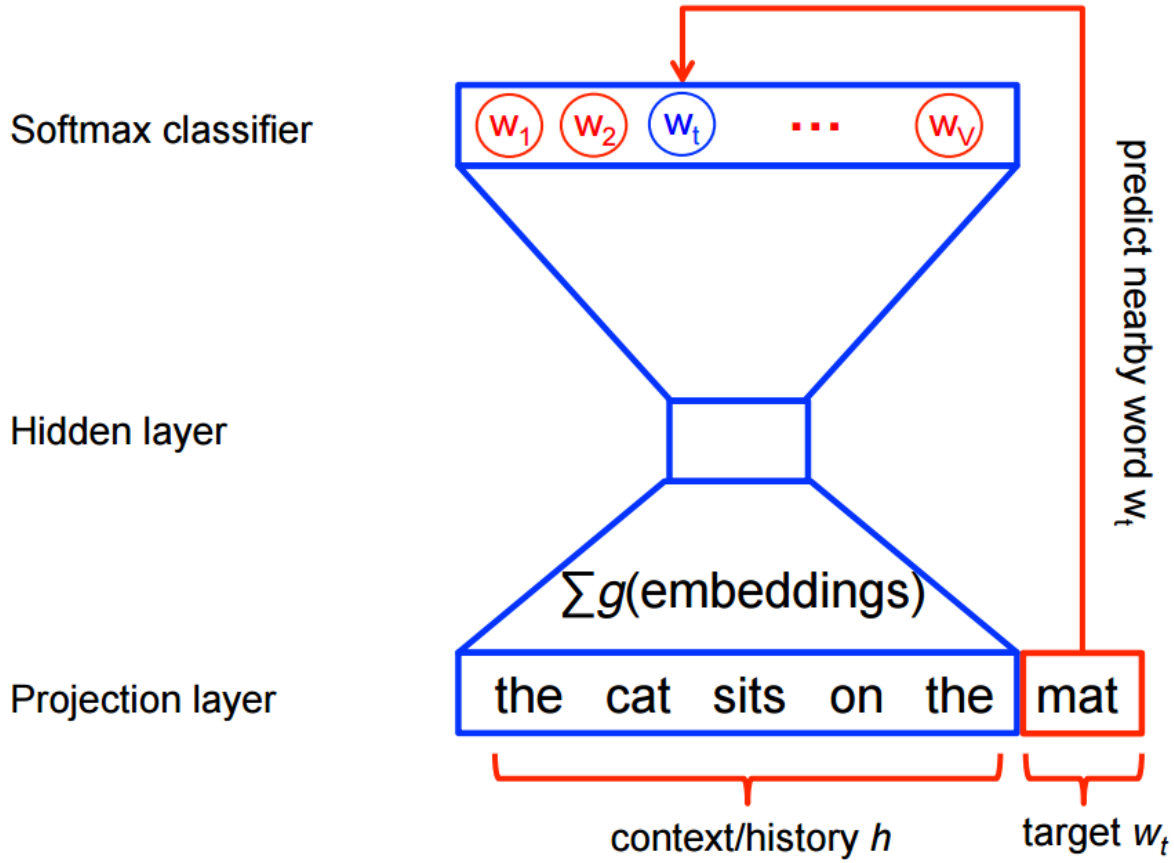
- In both approaches, the weight matrix is used as follows:



https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1

Two different ways to train

1.Using context to predict a target word (~ continuous bag-of-words)



https://www.tensorflow.org/versions/r0.7/tutorials/word2vec/index.html

Two different ways to train

2.Using word to predict a target context (skip-gram)

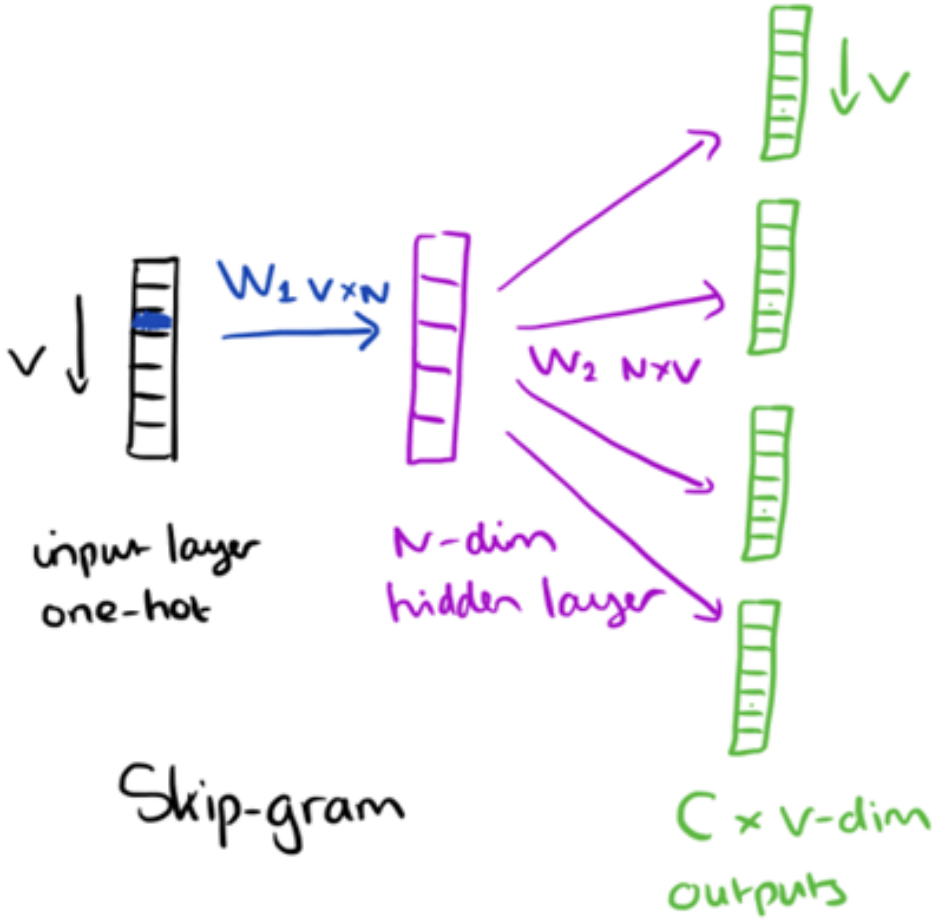
- Given a sentence:

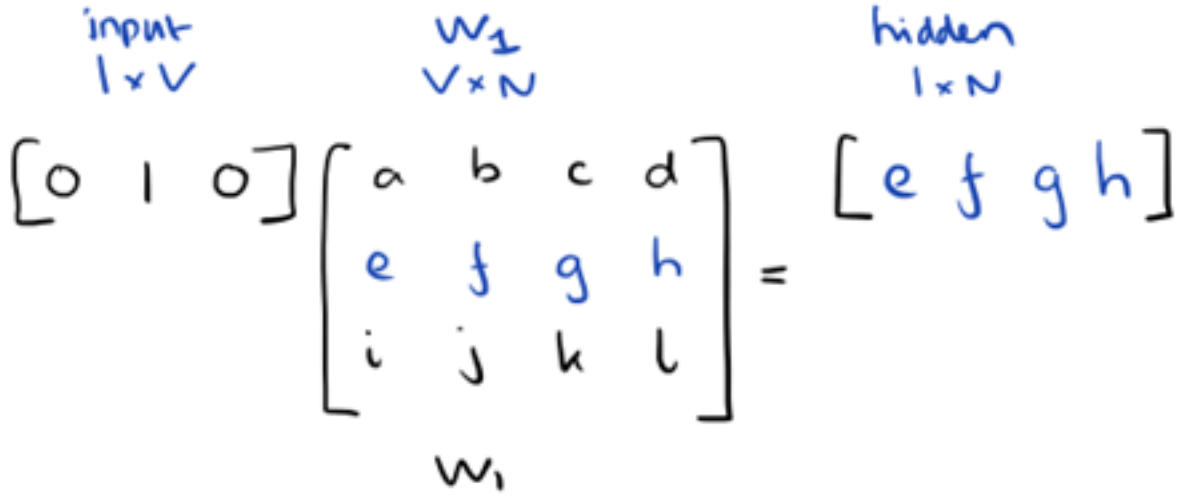
  the quick brown fox jumped over the lazy dog
- For each word, take context to be

  (N-words to the left, N-words to the right)
- If N = 1 (context, word):

  ([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), ...

Note that the weight matrix is a look-up table





https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1

Two different ways to train

2. Using word to predict a target context (skip-gram)

Window Size	Text	Skip-grams
2	[The **wide** road shimmered] in the hot sun.	wide, the wide, road wide, shimmered
	The [wide road **shimmered** in the] hot sun.	shimmered, wide shimmered, road shimmered, in shimmered, the
	The wide road shimmered in [the hot **sun**].	sun, the sun, hot
3	[The **wide** road shimmered in] the hot sun.	wide, the wide, road wide, shimmered wide, in
	[The wide road **shimmered** in the hot] sun.	shimmered, the shimmered, wide shimmered, road shimmered, in shimmered, the shimmered, hot
	The wide road shimmered [in the hot **sun**].	sun, in sun, the sun, hot

https://www.tensorflow.org/tutorials/text/word2vec

Some notes

- CBOW is called continuous BOW since the context is regarded as a BOW and it is continuous.
- In both approaches, the networks are composed of linear units
- The output units are usually normalized with the softmax
- According to Mikolov:
  - *“Skip-gram: works well with small amount of the training data, represents well even rare words or phrases.*
  - *CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words”*

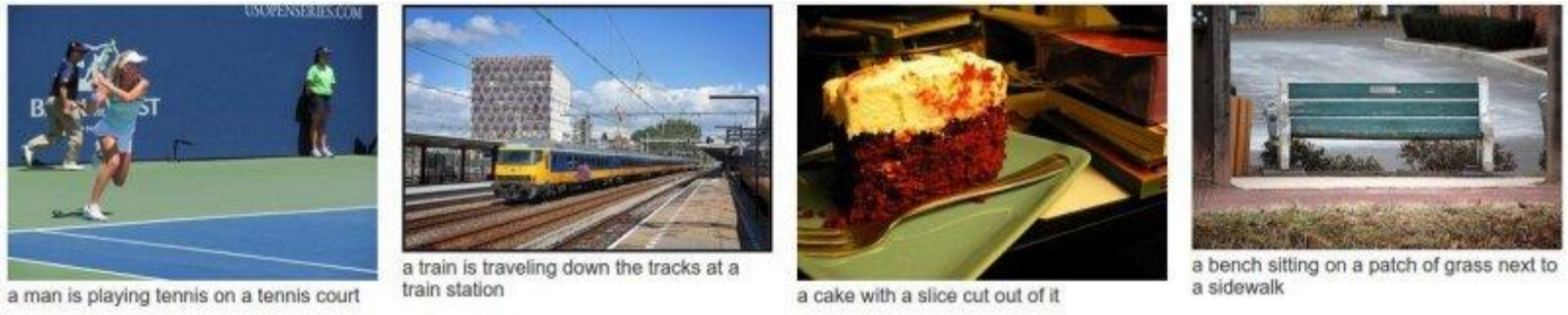


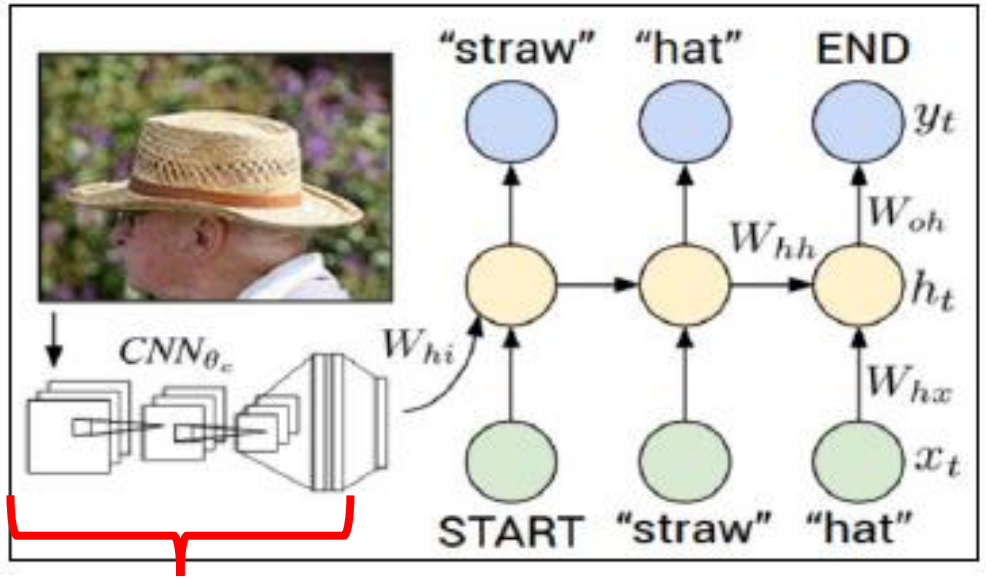
Fig: https://github.com/karpathy/neuraltalk2

Example: Image Captioning

Demo video

https://vimeo.com/146492001

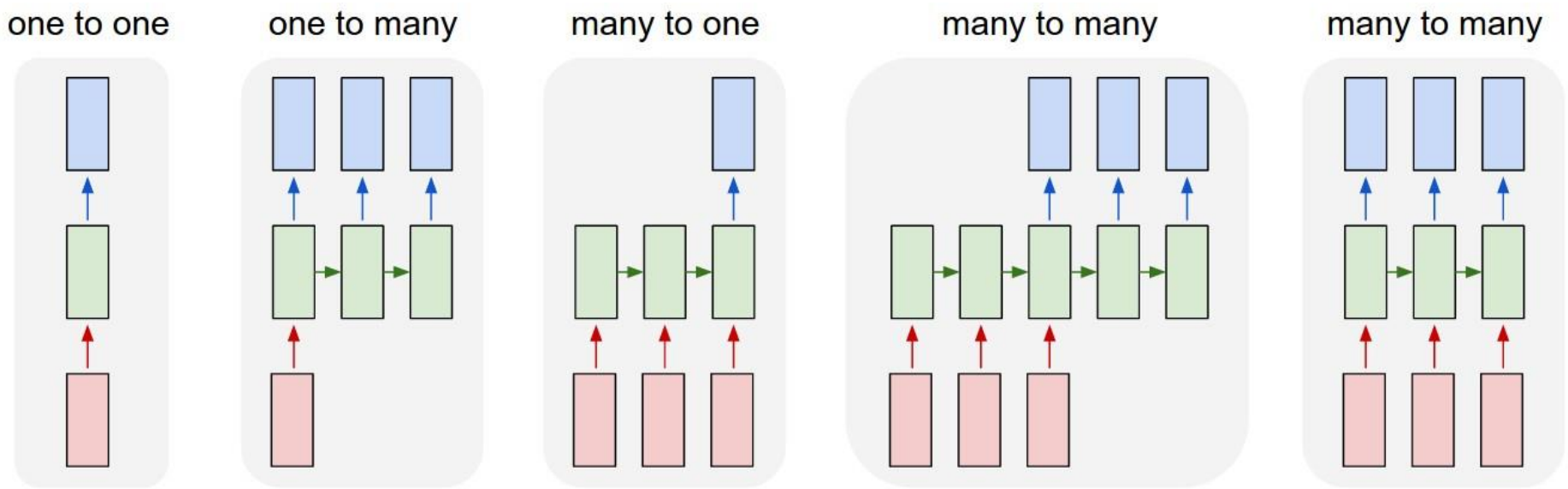
Overview



Pre-trained word embedding is also used

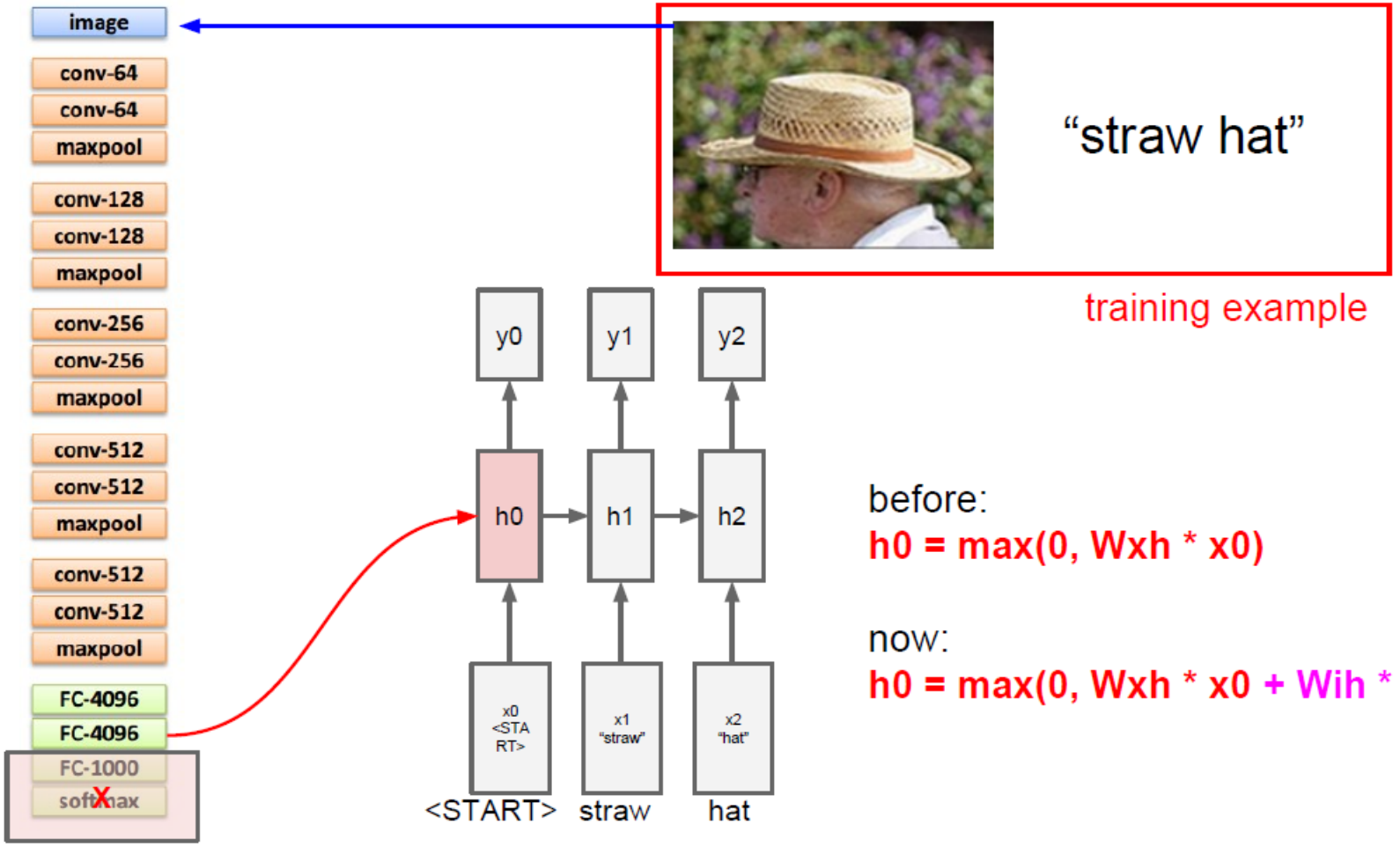
Pre-trained CNN
(e.g., on imagenet)

Image: Karpathy

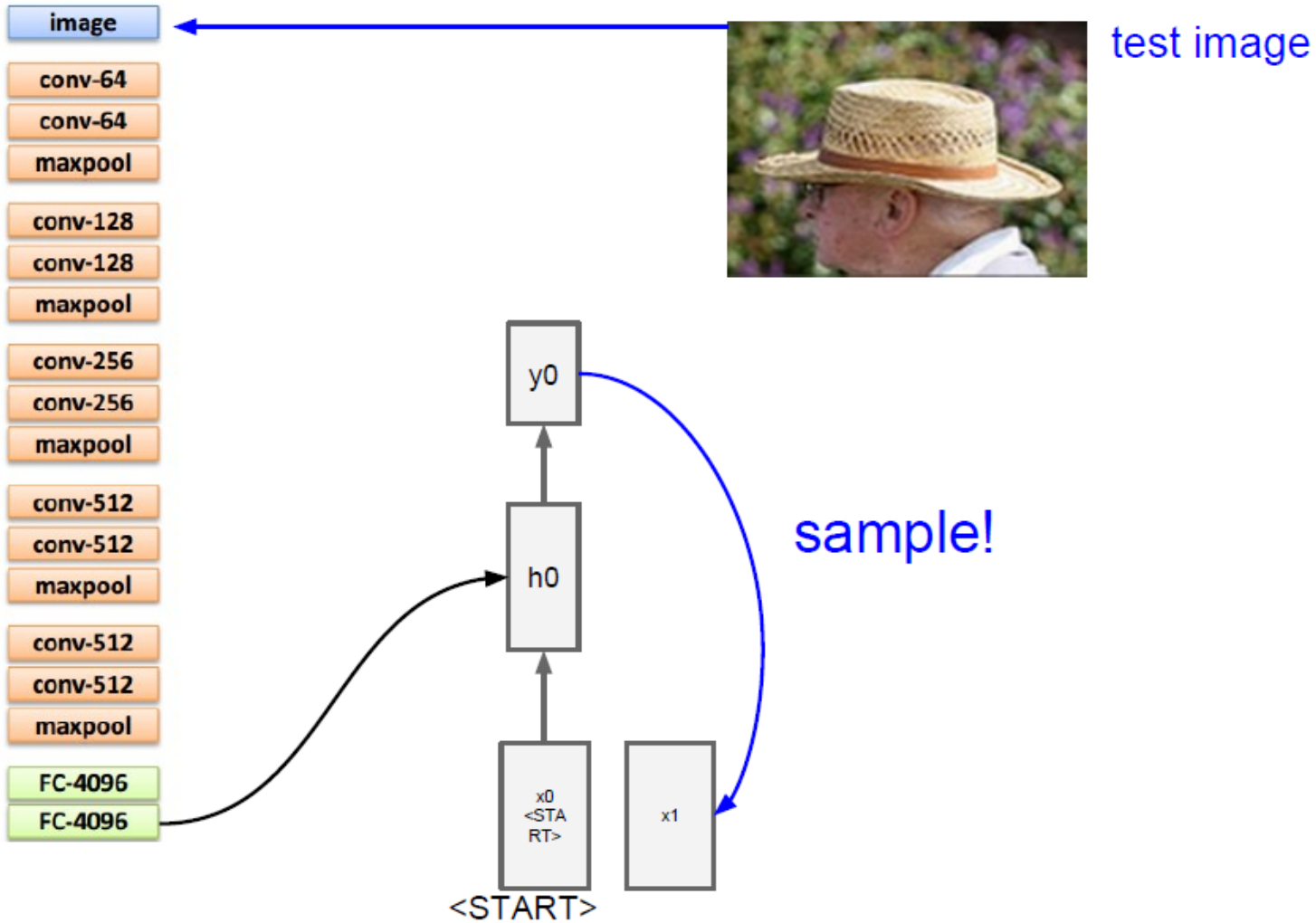


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

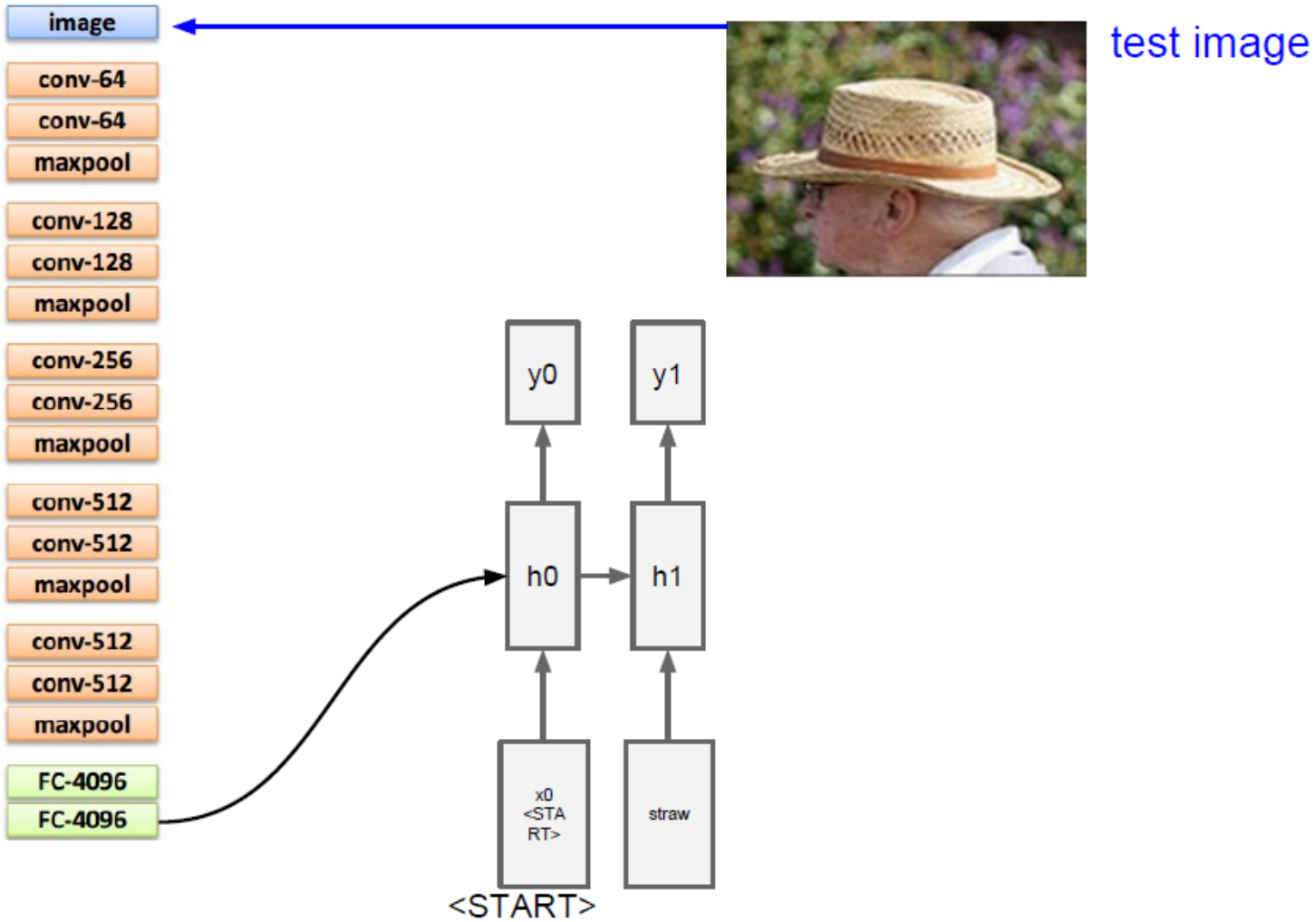
Training



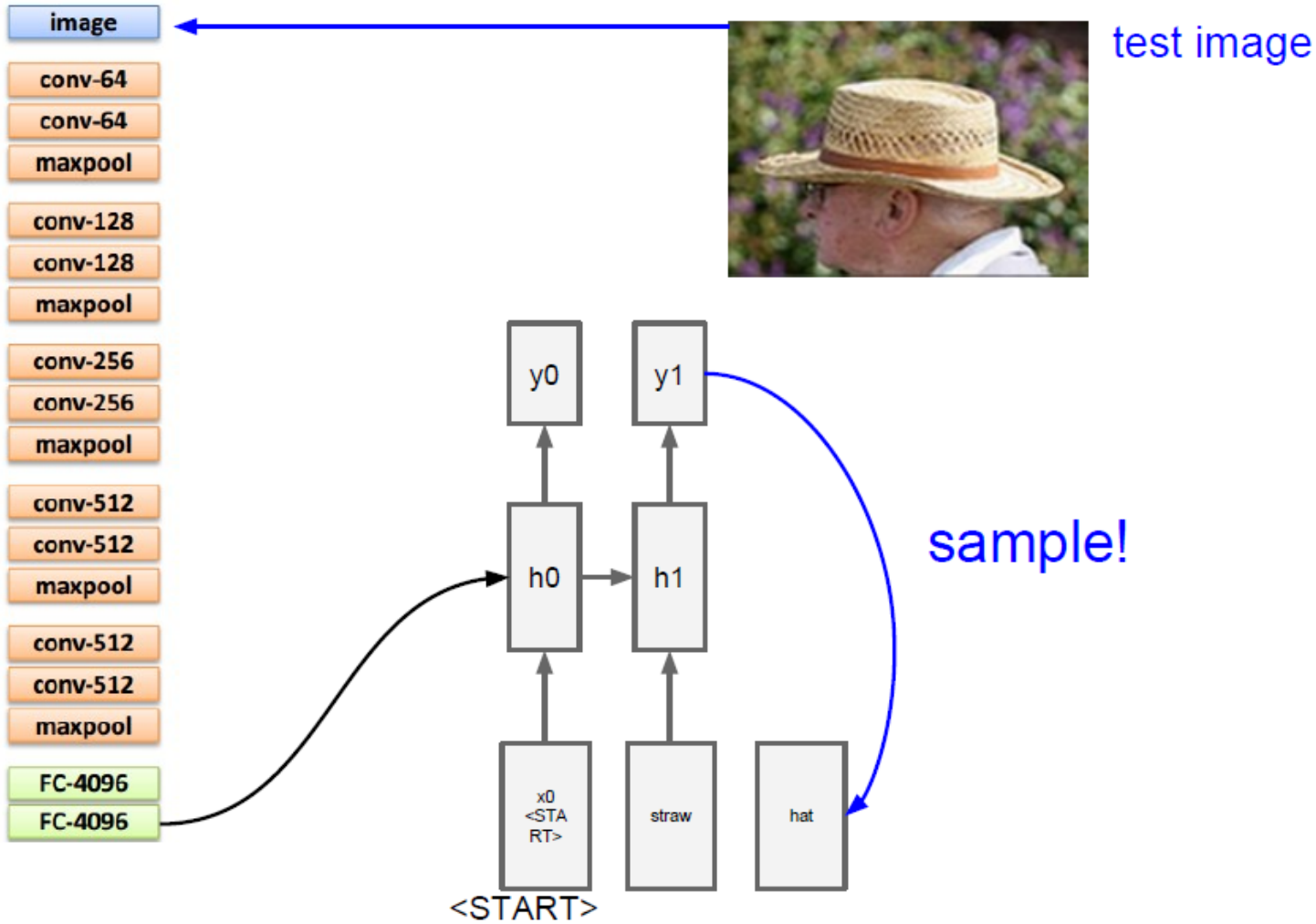
Slide: Karpathy



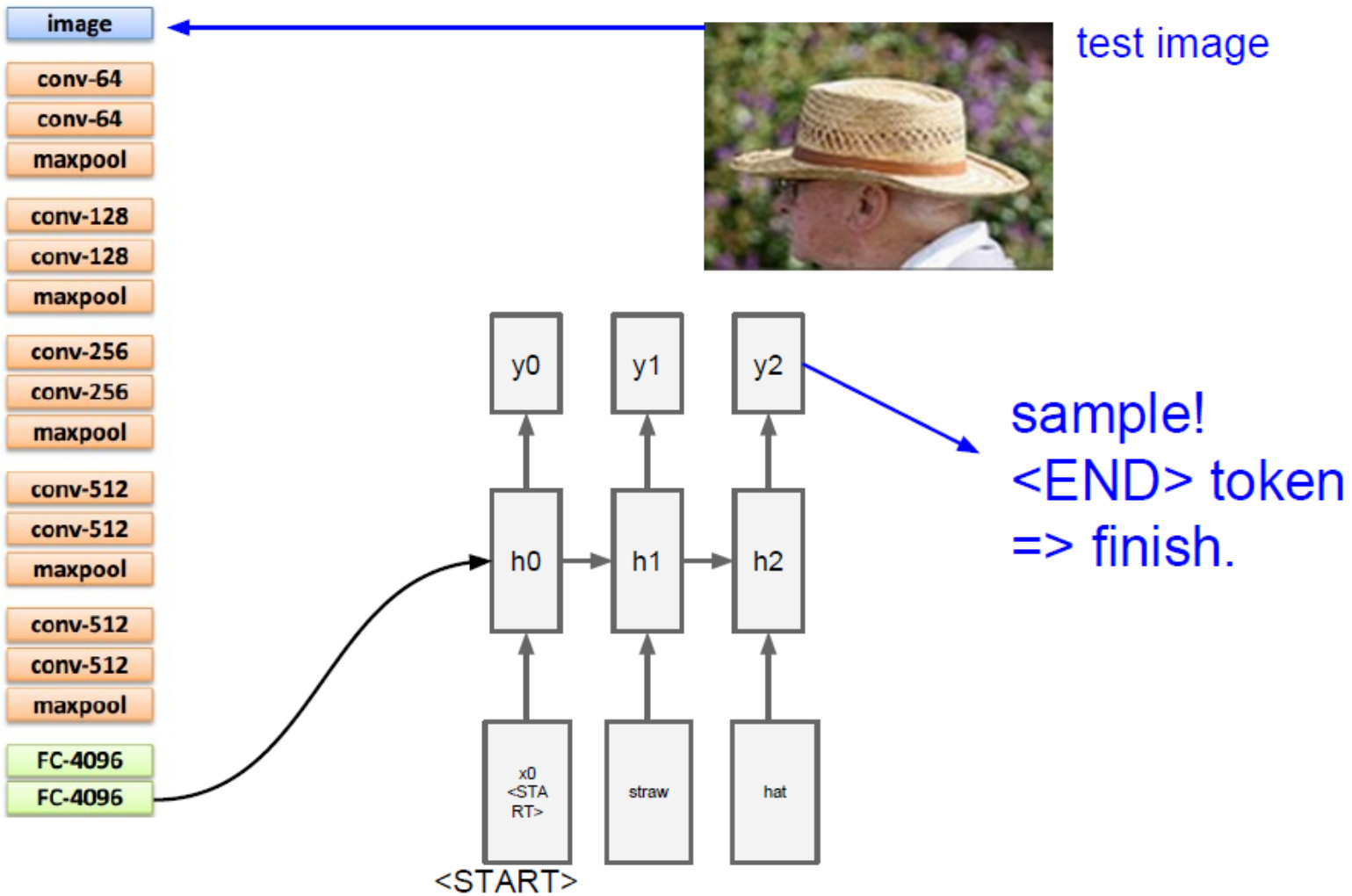
Slide: Karpathy



Slide: Karpathy



Slide: Karpathy



Slide: Karpathy

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

Kyunghyun Cho
Bart van Merriënboer **Caglar Gulcehre** Université de Montréal
firstname.lastname@umontreal.ca

Dzmitry Bahdanau Jacobs University, Germany
d.bahdanau@jacobs-university.de

Fethi Bougares **Holger Schwenk** Université du Maine, France
firstname.lastname@lium.univ-lemans.fr

Yoshua Bengio Université de Montréal, CIFAR Senior Fellow
find.me@on.the.web

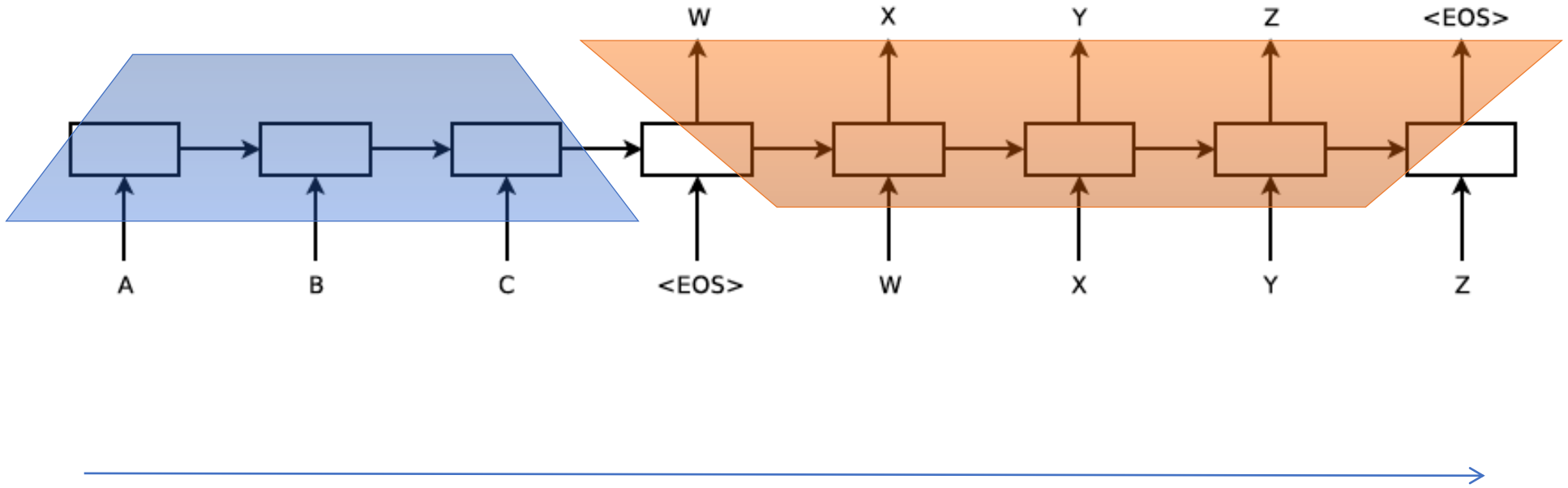
2014

Example: Neural Machine Translation

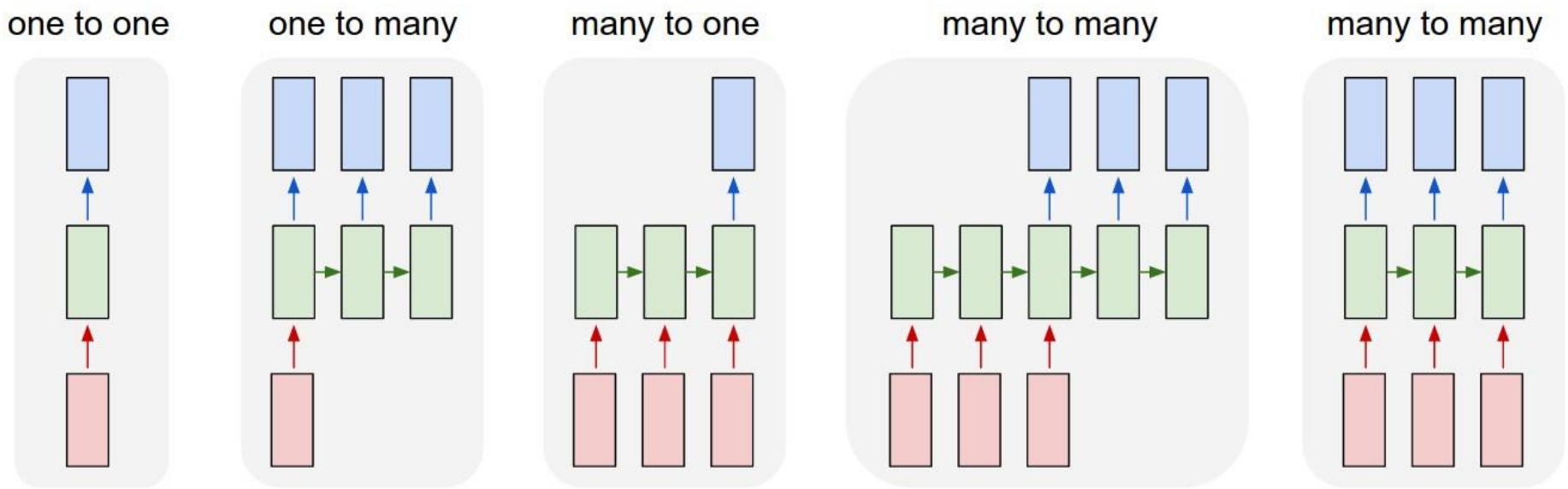
Neural Machine Translation

- Model

Each box is an LSTM or GRU cell.

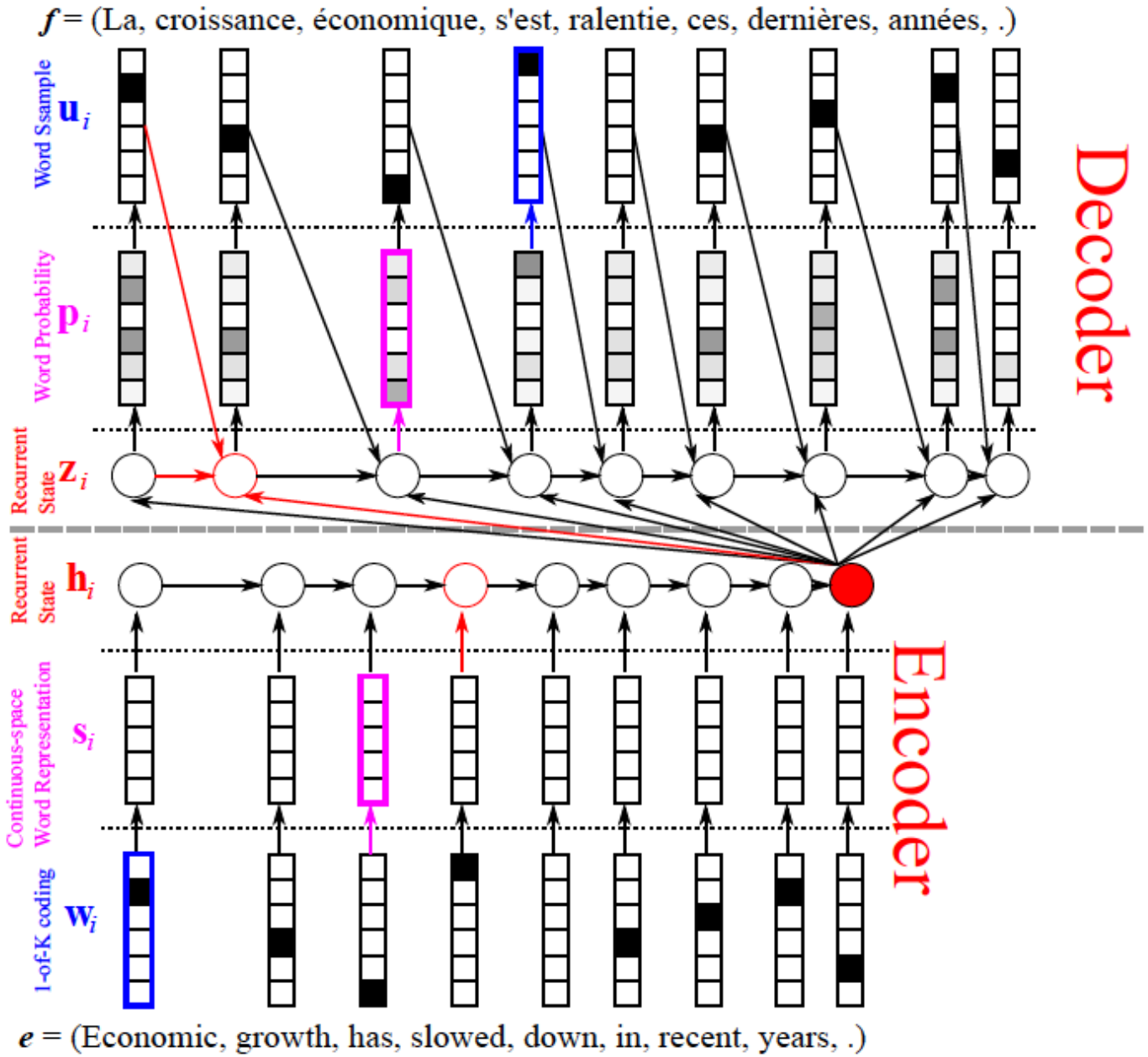


Sutskever et al. 2014



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

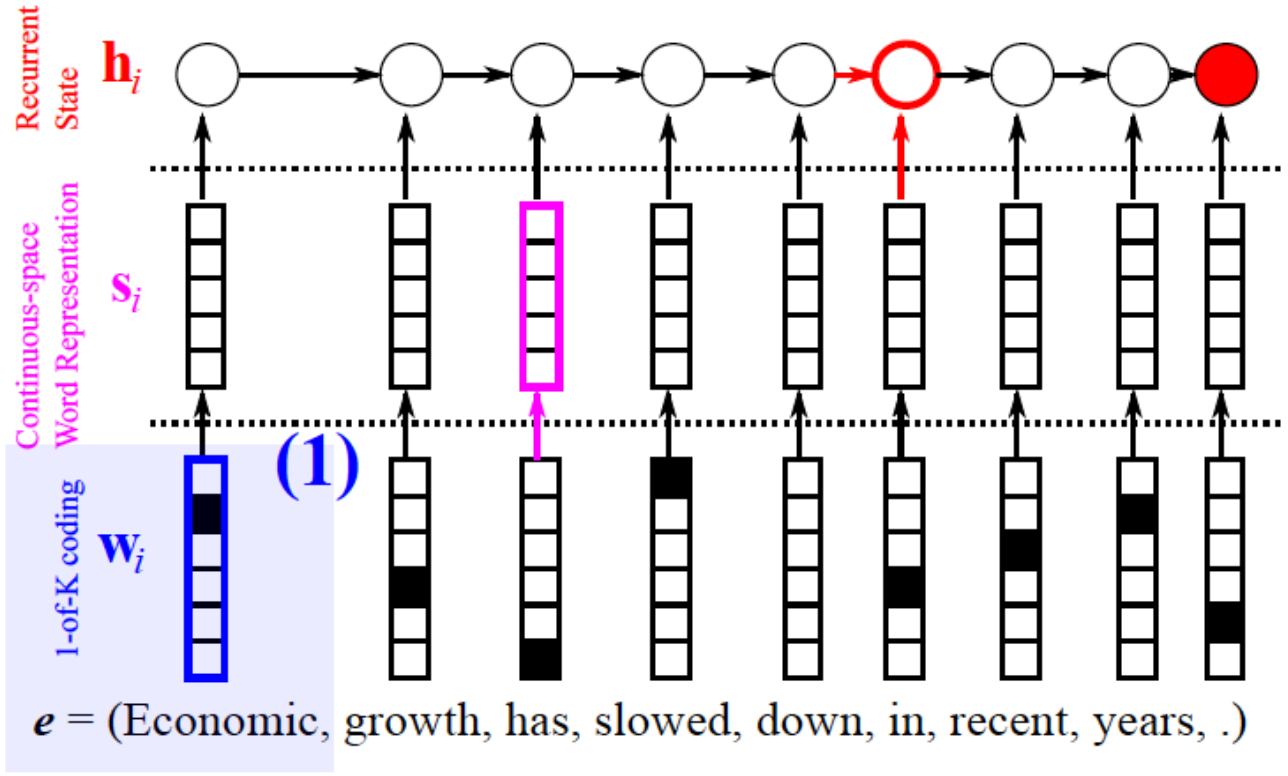
Neural Machine Translation



Cho: From Sequence Modeling to Translation

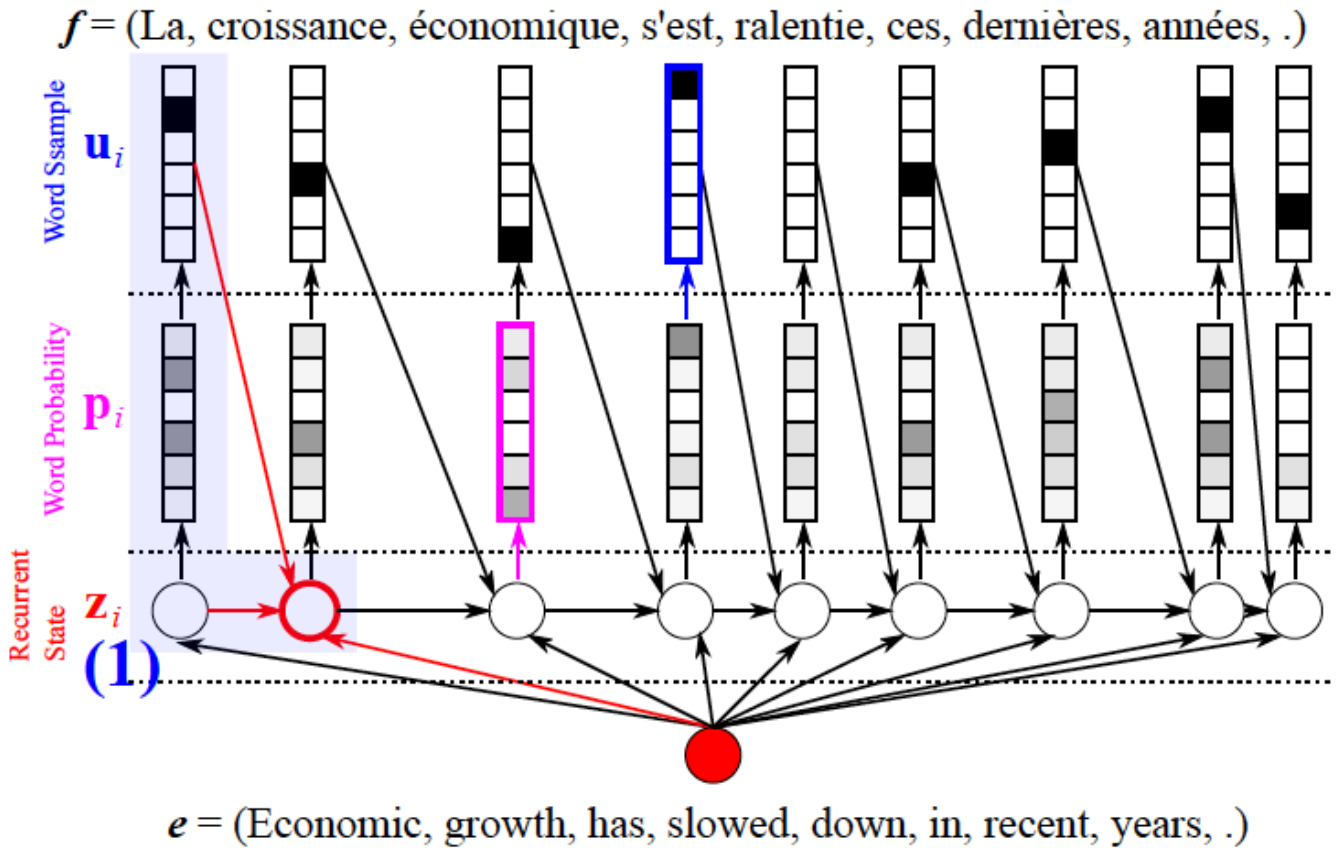
Neural Machine Translation

- Model- *encoder*



Neural Machine Translation

- Model- *decoder*



Cho: From Sequence Modeling to Translation

Haitham Elmarakeby

Decoder in more detail

Given

(i) the "summary" ($\mathbf{h}$) of the input sequence,

(ii) the previous output / word (f_{t-1})

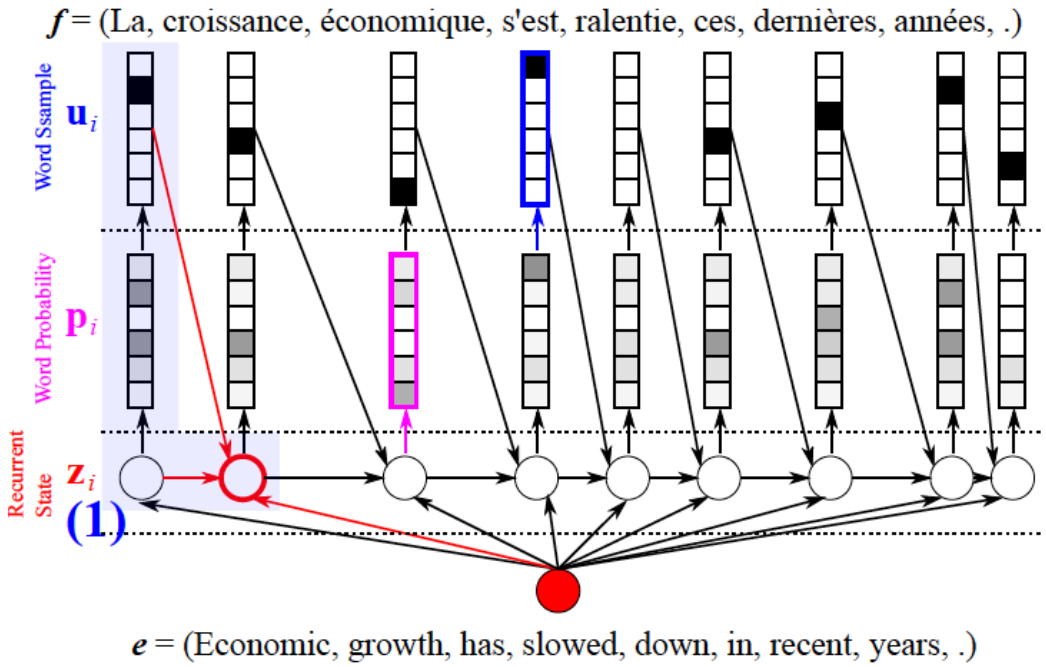
(iii) the previous state ($\mathbf{z}_{t-1}$)

the hidden state of the decoder is:

$$\mathbf{z}_t = RNN(\mathbf{z}_{t-1}, f_{t-1}, \mathbf{h})$$

Then, we can find the most likely next word:

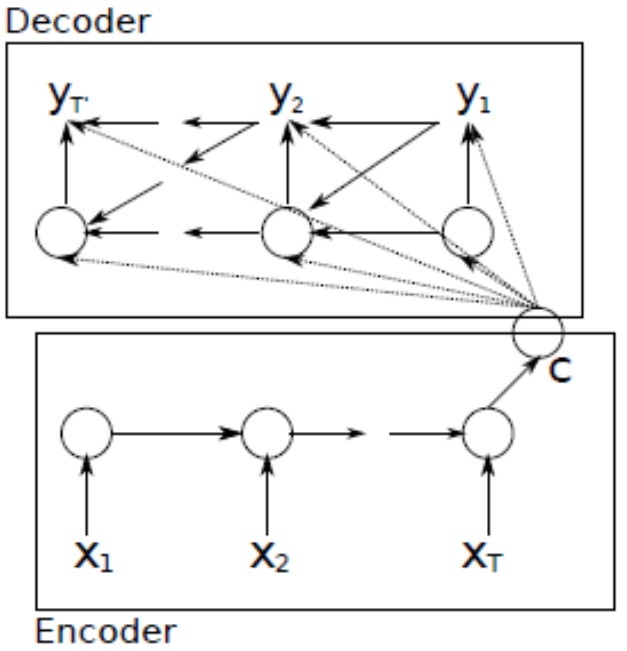
$$P(f_t \mid f_{t-1}, f_{t-2}, \ldots, \mathbf{h}) = p(f_t \mid \mathbf{z}_t, f_{t-1}, \mathbf{h})$$



Encoder-decoder

- Jointly trained to maximize

$$\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{y}_n \mid \mathbf{x}_n),$$



NMT can be done at char-level too

- http://arxiv.org/abs/1603.06147

This can be done with CNNs

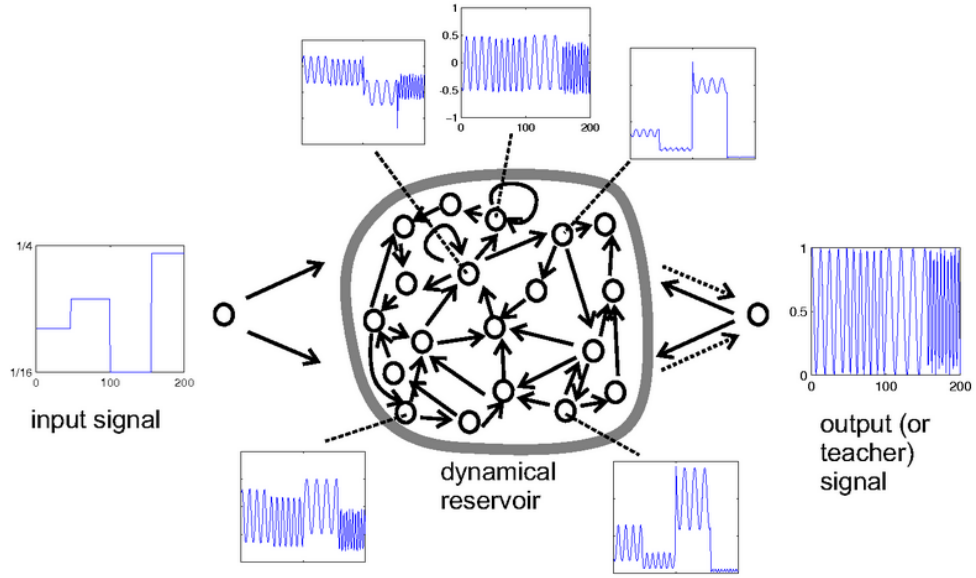
Convolutional Sequence to Sequence Learning

Jonas Gehring
Michael Auli
David Grangier
Denis Yarats
Yann N. Dauphin
Facebook AI Research

2017

Check the following tutorial

- http://smerity.com/articles/2016/google_nmt_arch.html



Echo State Networks

Reservoir Computing

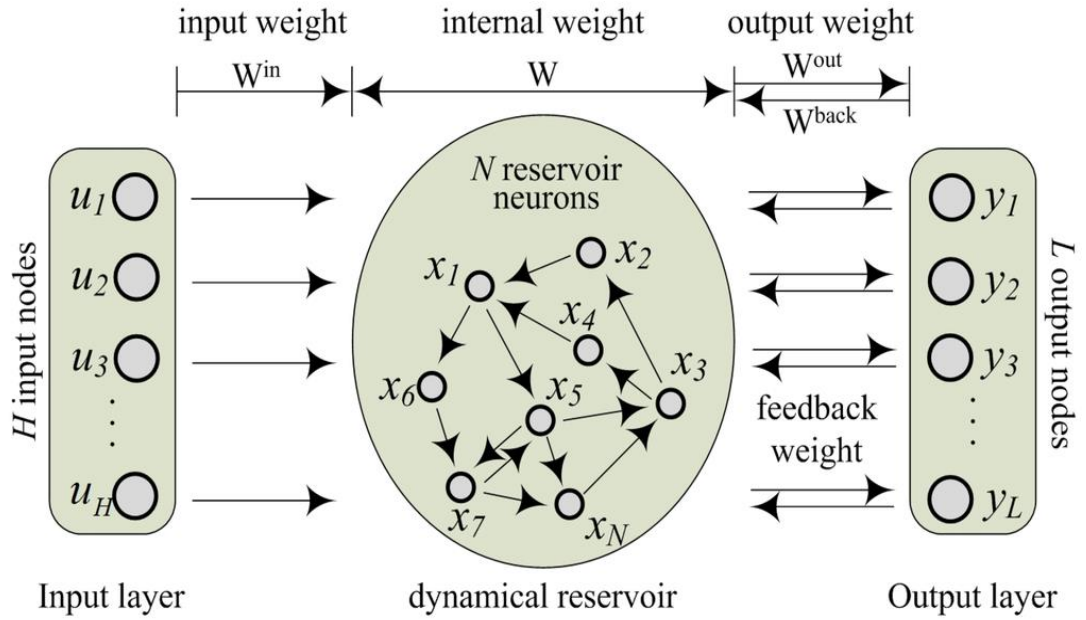
Motivation

- *"Schiller and Steil (2005) also showed that in traditional training methods for RNNs, where all weights (not only the output weights) are adapted, the dominant changes are in the output weights. In cognitive neuroscience, a related mechanism has been investigated by Peter F. Dominey in the context of modelling sequence processing in mammalian brains, especially speech recognition in humans (e.g., Dominey 1995, Dominey, Hoen and Inui 2006). Dominey was the first to explicitly state the principle of reading out target information from a randomly connected RNN. The basic idea also informed a model of temporal input discrimination in biological neural networks (Buonomano and Merzenich 1995)."*

http://www.scholarpedia.org/article/Echo_state_network

Echo State Networks (ESN)

- Reservoir of a set of neurons
  - Randomly initialized and fixed
  - Run input sequence through the network and keep the activations of the reservoir neurons
  - Calculate the “readout” weights using linear regression.
- Has the benefits of recurrent connections/networks
- No problem of vanishing gradient



Li et al., 2015.

The reservoir

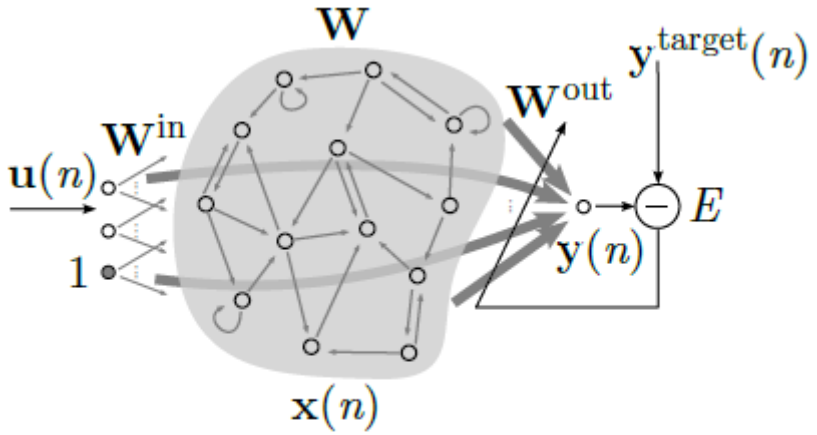


Fig. 1: An echo state network.

A Practical Guide to Applying Echo State Networks

Mantas Lukoševičius

- Provides non-linear expansion
  - This provides a "kernel" trick.
- Acts as a memory
- Parameters:
  - W_{in}, W and α (leaking rate).
- Global parameters:
  - Number of neurons: The more the better.
  - Sparsity: Connect a neuron to a fixed but small number of neurons.
  - Distribution of the non-zero elements: Uniform or Gaussian distribution. W_{in} is denser than W.
  - Spectral radius of W: Maximum absolute eigenvalue of W, or the width of the distribution of its non-zero elements.
    - Should be less than 1. Otherwise, chaotic, periodic or multiple fixed-point behavior may be observed.
    - For problems with large memory requirements, it should be bigger than 1.
  - Scale of the input weights.

A Practical Guide to Applying Echo State Networks

Mantas Lukoševičius

$$\tilde{\mathbf{x}}(n) = \tanh\left(\mathbf{W}^{\text{in}}[1; \mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1)\right), \tag{2}$$

$$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha\tilde{\mathbf{x}}(n), \tag{3}$$

where $\mathbf{x}(n) \in \mathbb{R}^{N_\text{x}}$ is a vector of reservoir neuron activations and $\tilde{\mathbf{x}}(n) \in \mathbb{R}^{N_\text{x}}$ is its update, all at time step n, $\tanh(\cdot)$ is applied element-wise, $[\cdot;\cdot]$ stands for a vertical vector (or matrix) concatenation, $\mathbf{W}^{\text{in}} \in \mathbb{R}^{N_\text{x}\times(1+N_\text{u})}$ and $\mathbf{W} \in \mathbb{R}^{N_\text{x}\times N_\text{x}}$ are the input and recurrent weight matrices respectively, and $\alpha \in (0, 1]$ is the leaking rate. Other sigmoid wrappers can be used besides the tanh, which however is the most common choice. The model is also sometimes used without the leaky integration, which is a special case of $\alpha = 1$ and thus $\tilde{\mathbf{x}}(n) \equiv \mathbf{x}(n)$.

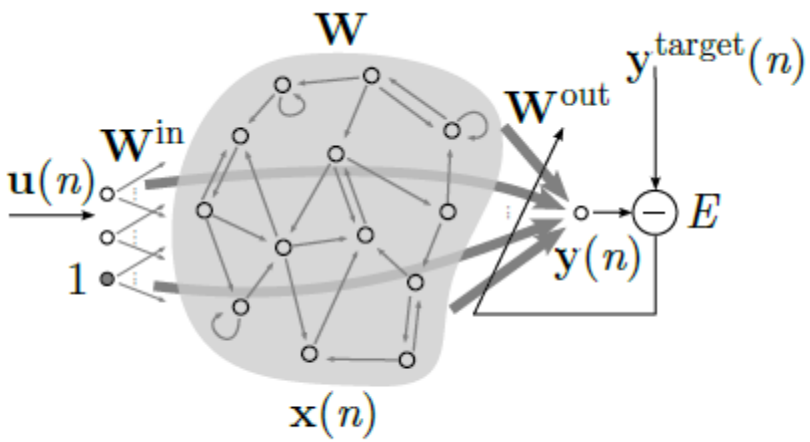


Fig. 1: An echo state network.

$$\mathbf{y}(n) = \mathbf{W}^{\text{out}}[1; \mathbf{u}(n); \mathbf{x}(n)],$$

again stands for a vertical vector (or matrix) concatenation. An additional nonlinearity can be applied to $\mathbf{y}(n)$ in (4), as well as feedback connections $\mathbf{W}^{\text{fb}}$ from $\mathbf{y}(n-1)$ to $\tilde{\mathbf{x}}(n)$ in (2). A graphical

Training ESN

$$\mathbf{Y}^{\text{target}} = \mathbf{W}^{\text{out}}\mathbf{X}$$

Probably the most universal and stable solution to (8) in this context is ridge regression, also known as regression with Tikhonov regularization:

$$\mathbf{W}^{\text{out}} = \mathbf{Y}^{\text{target}}\mathbf{X}^{\text{T}}\left(\mathbf{X}\mathbf{X}^{\text{T}} + \beta\mathbf{I}\right)^{-1}, \tag{9}$$

where β is a regularization coefficient explained in Section 4.2, and $\mathbf{I}$ is the identity matrix.

Overfitting (regularization):

$$\mathbf{W}^{\text{out}} = \arg\min_{\mathbf{W}^{\text{out}}} \frac{1}{N_{\text{y}}} \sum_{i=1}^{N_{\text{y}}} \left(\sum_{n=1}^{T} \left(y_i(n) - y_i^{\text{target}}(n)\right)^2 + \beta \left\|\mathbf{w}_i^{\text{out}}\right\|^2 \right),$$

Beyond echo state networks

- Good aspects of ESNs
  Echo state networks can be trained very fast because they just fit a linear model.
- They demonstrate that it's very important to initialize weights sensibly.
- They can do impressive modeling of one-dimensional time-series.
  - but they cannot compete seriously for high-dimensional data.

- Bad aspects of ESNs
  They need many more hidden units for a given task than an RNN that learns the hidden→hidden weights.

Slide: Hinton

Similar models

- Liquid State Machines (Maas et al., 2002)
  - A spiking version of Echo-state networks

- Extreme Learning Machines
  - Feed-forward network with a hidden layer.
  - Input-to-hidden weights are randomly initialized and never updated

Attention

Attention

BLEU: Bilingual Evaluation Understudy

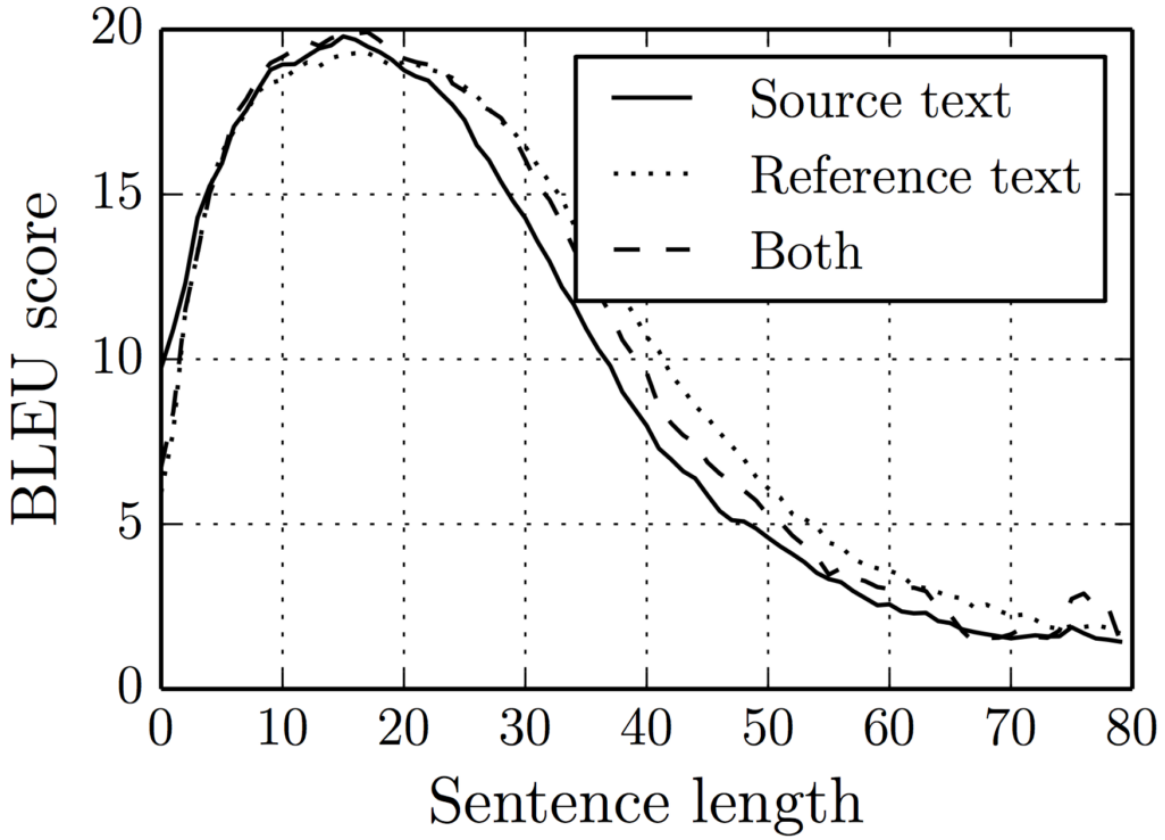
https://cloud.google.com/translate/automl/docs/evaluate#bleu

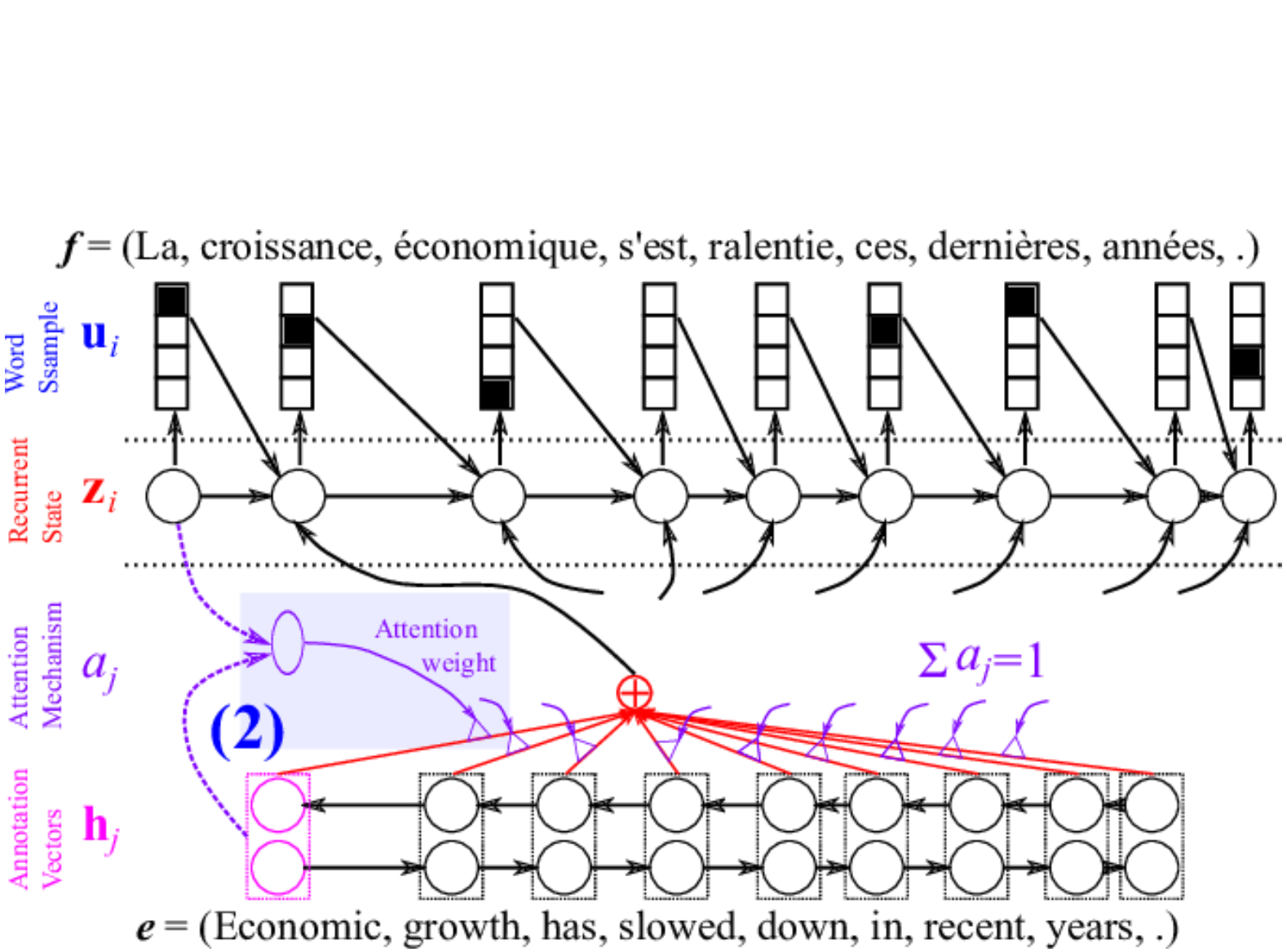
Published as a conference paper at ICLR 2015

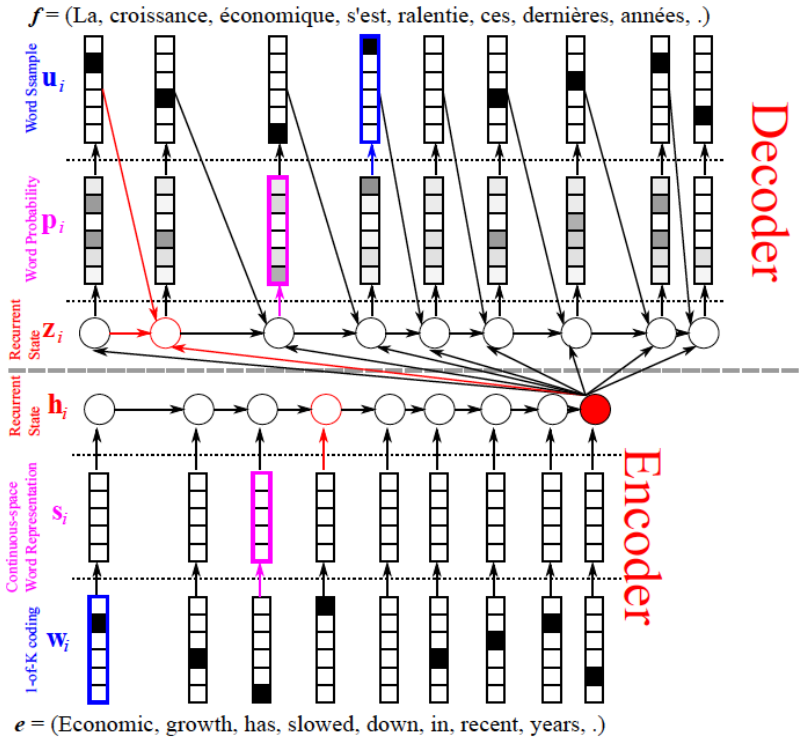
NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal




f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
u_i
Recurrent State
z_i
Attention Mechanism
a_j
Attention weight
Σ a_j=1
(2)
Annotation Vectors
h_j
e = (Economic, growth, has, slowed, down, in, recent, years, .)


f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
u_i
Word Probability
p_i
Recurrent State
z_i
Decoder
Recurrent State
h_i
Continuous-space Word Representation
s_i
1-of-K coding
w_i
Encoder
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Attention

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

In a new model architecture, we define each conditional probability in Eq. (2) as:

$$p(y_i|y_1,\dots,y_{i-1},\mathbf{x}) = g(y_{i-1},s_i,c_i), \tag{4}$$

where s_i is an RNN hidden state for time i, computed by

$$s_i = f(s_{i-1},y_{i-1},c_i).$$

It should be noted that unlike the existing encoder–decoder approach (see Eq. (2)), here the probability is conditioned on a distinct context vector c_i for each target word y_i.

The context vector c_i depends on a sequence of *annotations* $(h_1,\cdots,h_{T_x})$ to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence. We explain in detail how the annotations are computed in the next section.

The context vector c_i is, then, computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{5}$$

The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x}\exp(e_{ik})}, \tag{6}$$

where

$$e_{ij} = a(s_{i-1},h_j)$$

is an *alignment model* which scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden state s_{i-1} (just before emitting y_i, Eq. (4)) and the j-th annotation h_j of the input sentence.

We parametrize the alignment model a as a feedforward neural network which is jointly trained with all the other components of the proposed system. Note that unlike in traditional machine translation,

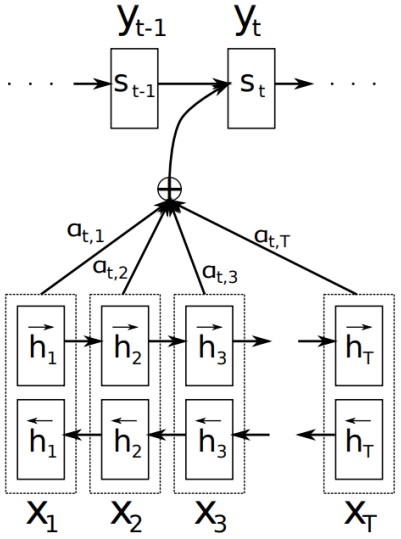
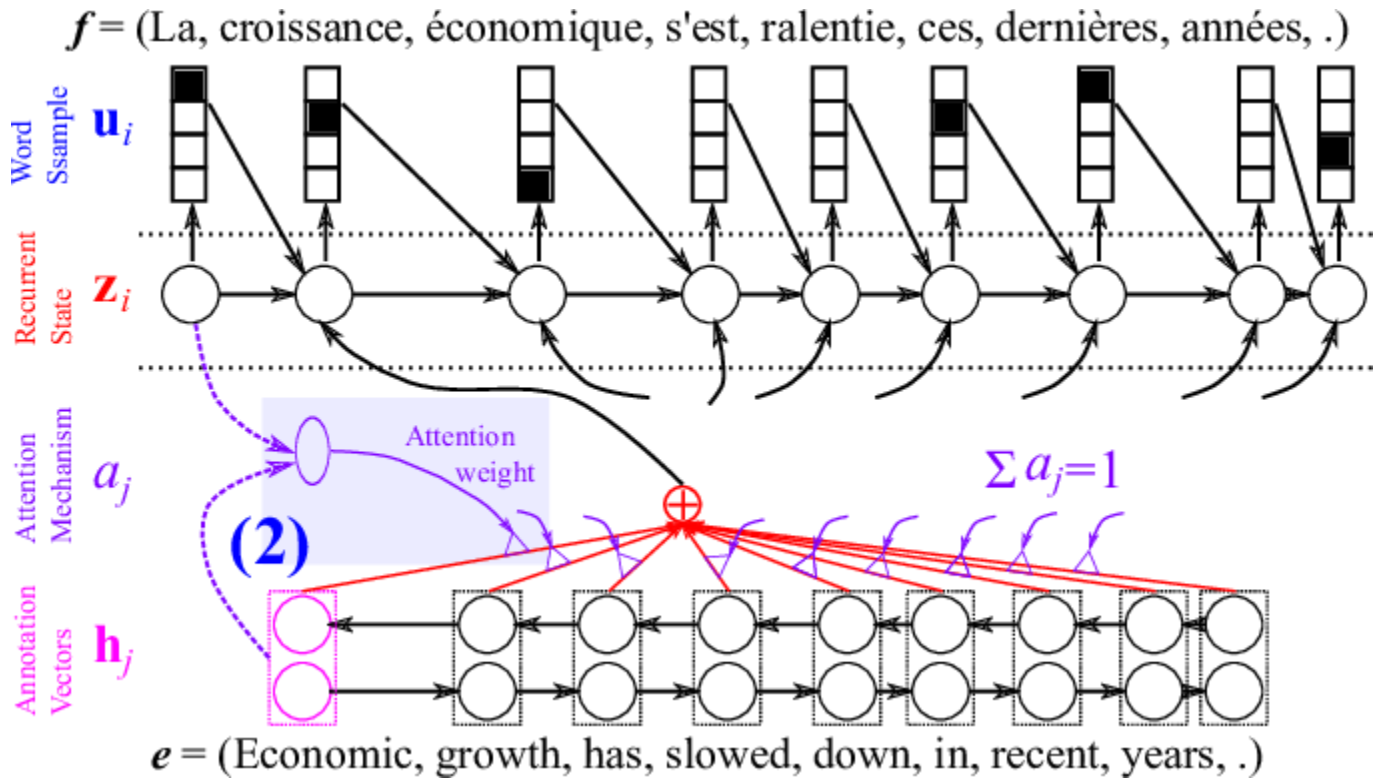


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1,x_2,\dots,x_T)$.

Attention



Attention mechanism: A two-layer neural network.

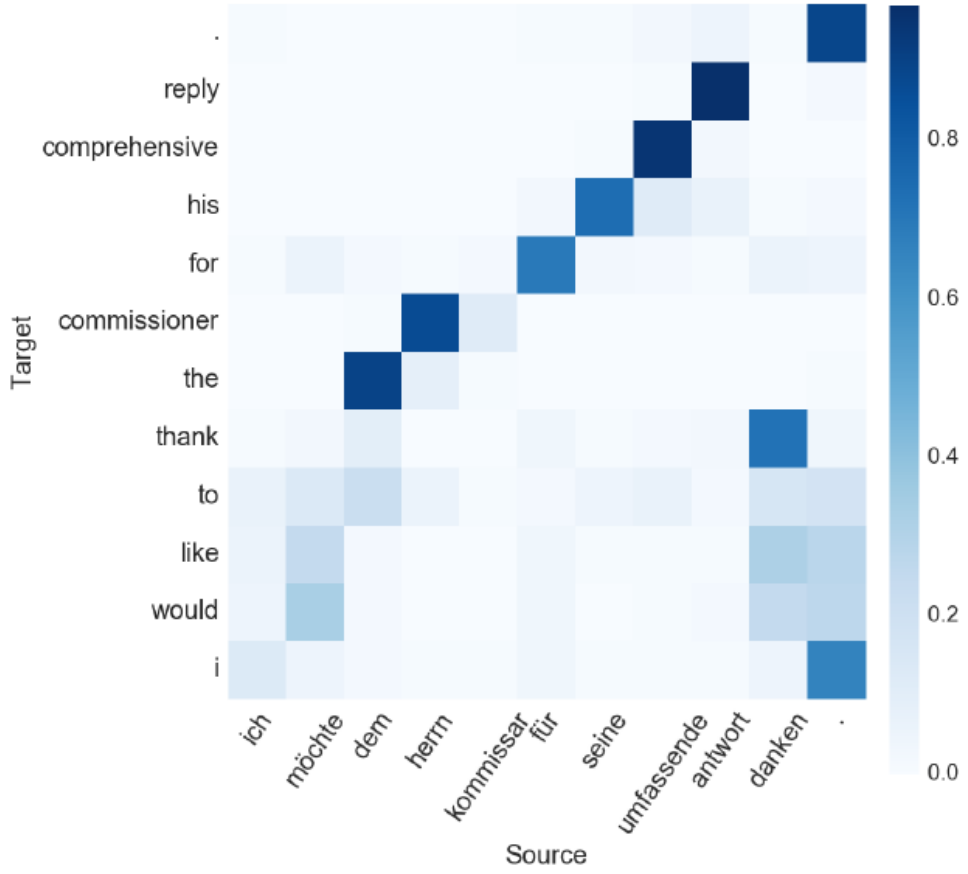
Input: z_i and h_j

Output: e_j, a scalar for the importance of word j.

The scores of words are normalized: $a_j = \mathrm{softmax}(\mathrm{e_j})$

https://devblogs.nvidia.com/introduction-neural-machine-translation-gpus-part-3/

Attention



What does Attention in Neural Machine Translation Pay Attention to?

Hamidreza Ghader and **Christof Monz**
Informatics Institute, University of Amsterdam, The Netherlands
h.ghader, c.monz@uva.nl

2017

Attention Types

- Let's rewrite Bahdanau et al.'s attention model:

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i \qquad ; \text{Context vector for output } y_t$$

$$\alpha_{t,i} = \text{align}(y_t, x_i) \qquad ; \text{How well two words } y_t \text{ and } x_i \text{ are aligned.}$$

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))} \qquad ; \text{Softmax of some predefined alignment score..}$$

$$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$$

where both $\mathbf{v}_a$ and $\mathbf{W}_a$ are weight matrices to be learned in the alignment model.

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Attention Types

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

(*) Referred to as "concat" in Luong, et al., 2015 and as "additive attention" in Vaswani, et al., 2017.
(^) It adds a scaling factor $1/\sqrt{n}$, motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Vanilla Self-attention

$$e_i' = \sum_j \frac{\exp\left(e_j^T e_i\right)}{\sum_m \exp(e_m^T e_i)} e_j$$

Attention: Transformer

- Vanilla self attention:

$$e_i' = \sum_j \frac{\exp(e_j^T e_i)}{\sum_m \exp(e_m^T e_i)} e_j$$

- Scaled-dot product attention:

$$e_i' = \sum_j \frac{\exp(k(e_j^T) q(e_i))}{\sum_m \exp(k(e_m^T) q(e_i))} v(e_j)$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com