CENG501 – Deep Learning

Week 5

Fall 2024

Sinan Kalkan

Dept. of Computer Engineering, METU

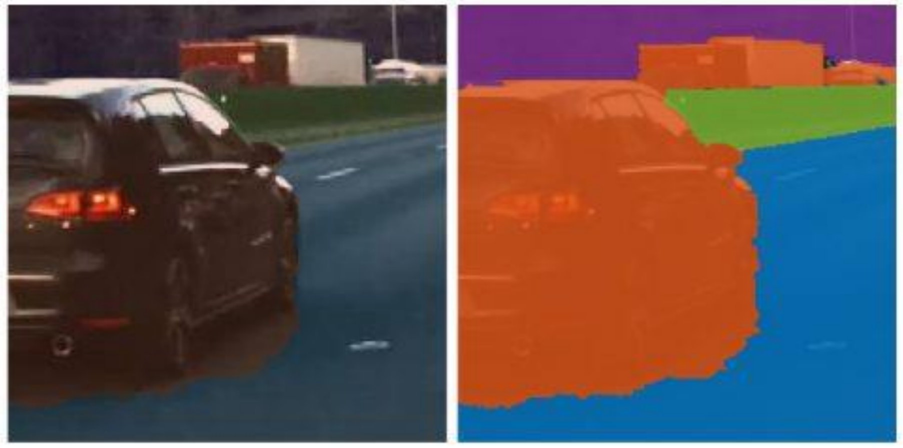
Disadvantages of MLPs: Dimensionality

- The number of parameters in an MLP is high for practical problems
  - e.g., for grayscale images with 1000x1000 resolution, a fully-connected layer with 1000 neurons requires 10^9 parameters.
- The number of parameters in an MLP increases quadratically with an increase in input dimensionality
- For example, for a fully-connected layer with n_{in} input neurons and n_{out} output neurons:
  - Number of parameters: $n_{in} \times n_{out}$
  - Assuming proportional decrease in layer size, e.g. $n_{out} = n_{in}/10$, gives: $n_{in} \times n_{out} = n_{in}^2/10$
  - Increasing n_{in} by d yields a change of $\mathcal{O}(d^2)$.
- This is a problem because:
  - More parameters => larger model size & more computational complexity.
- Teaser for CNNs:
  - Input size does not affect model size (in general)

Previously on CENG501

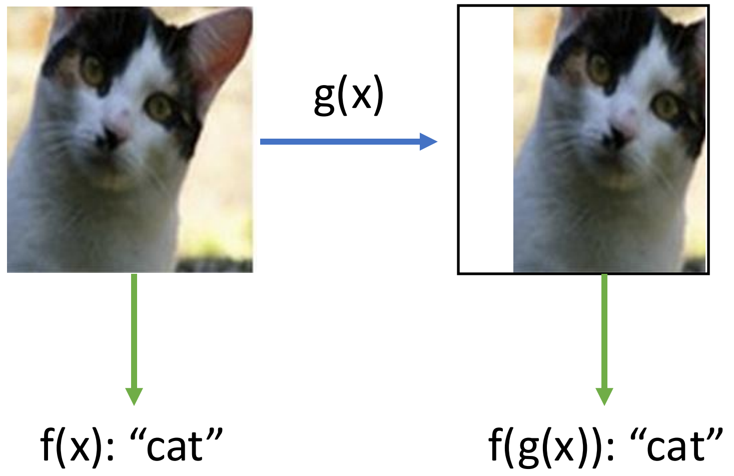
Equivariance vs. Invariance

- Equivariant problem: image segmentation.
  - f(g(x)) = g(f(x))



https://www.mathworks.com/discovery/image-segmentation.html

- Invariant problem: object recognition.
  - f(g(x)) = f(x)



- Pooling provides invariance, convolution provides equivariance.

An Alternative to MLPs

Solution: Neocognitron (Fukushima, 1979):

A neural network model unaffected by shift in position, applied to Japanese handwritten character recognition.

- S (simple) cells: local feature extraction.
- C (complex) cells: provide tolerance to deformation, e.g. shift.
- Self-organized learning method.

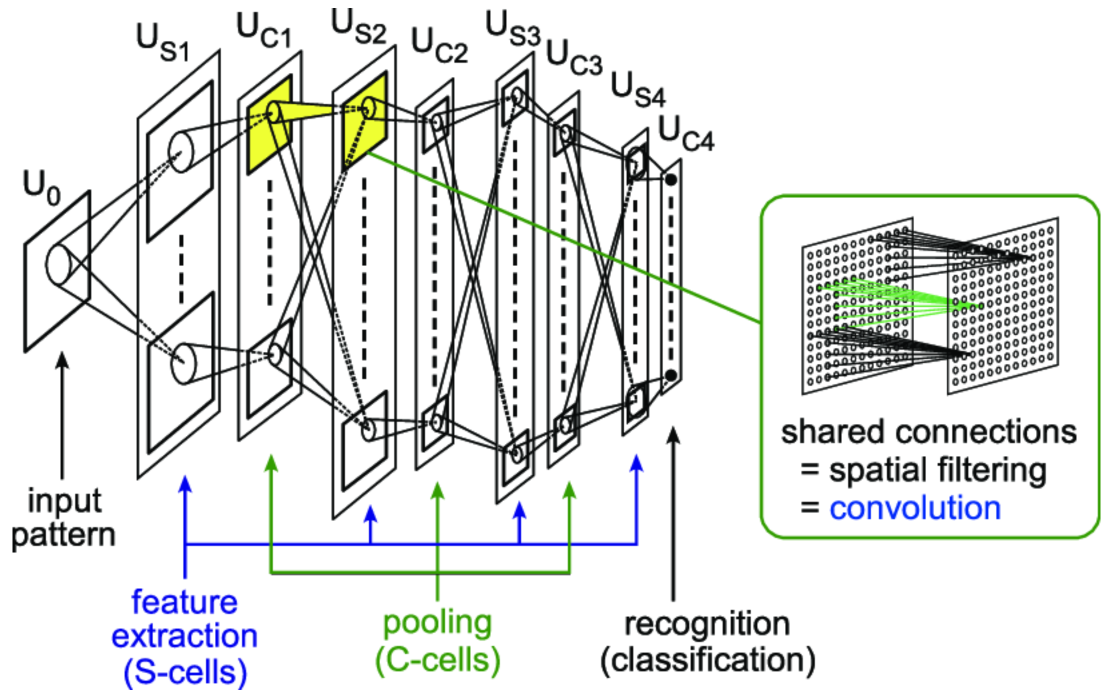
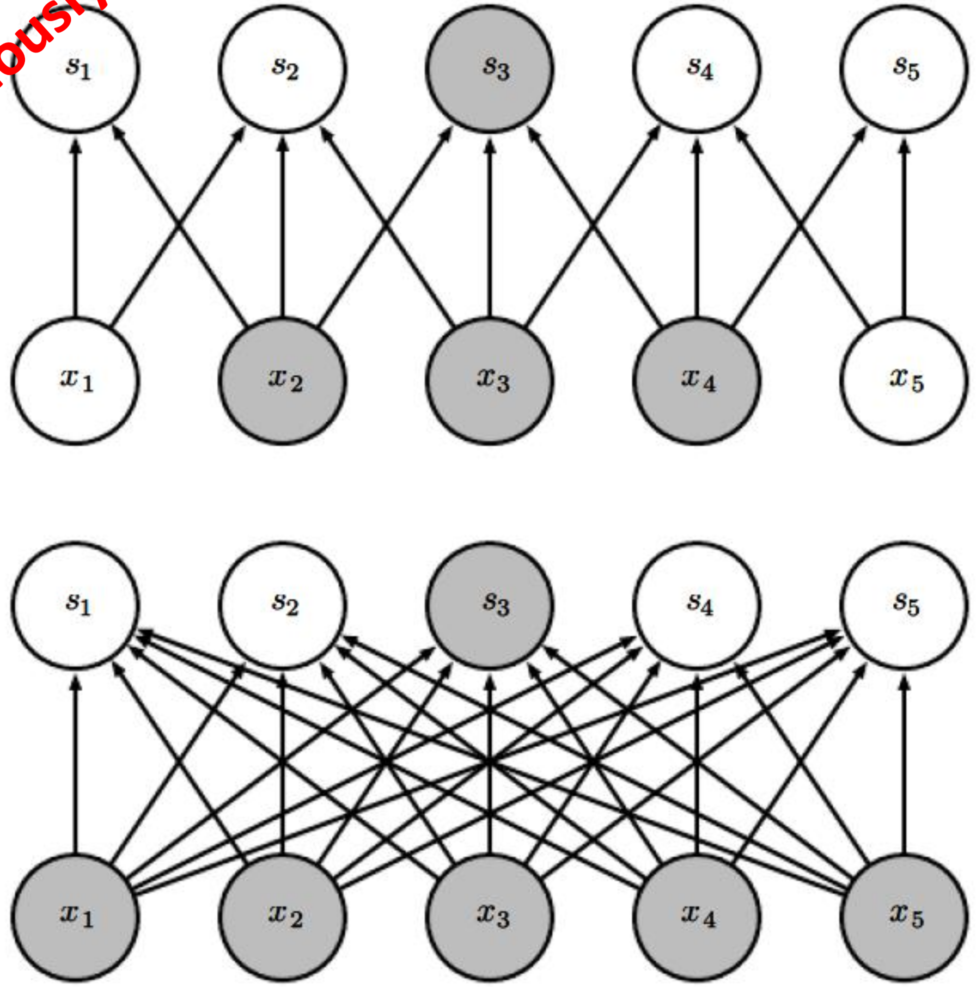

U_{S1}
U_{C1}
U_{S2}
U_{C2}
U_{S3}
U_{C3}
U_{S4}
U_{C4}
U_0
input pattern
feature extraction (S-cells)
pooling (C-cells)
recognition (classification)
shared connections = spatial filtering = convolution

Figure: Fukushima (2019), Recent advances in the deep CNN neocognitron.

CNNs vs. MLPs: Curse of Dimensionality



When things go deep, an output may depend on all or most of the input:

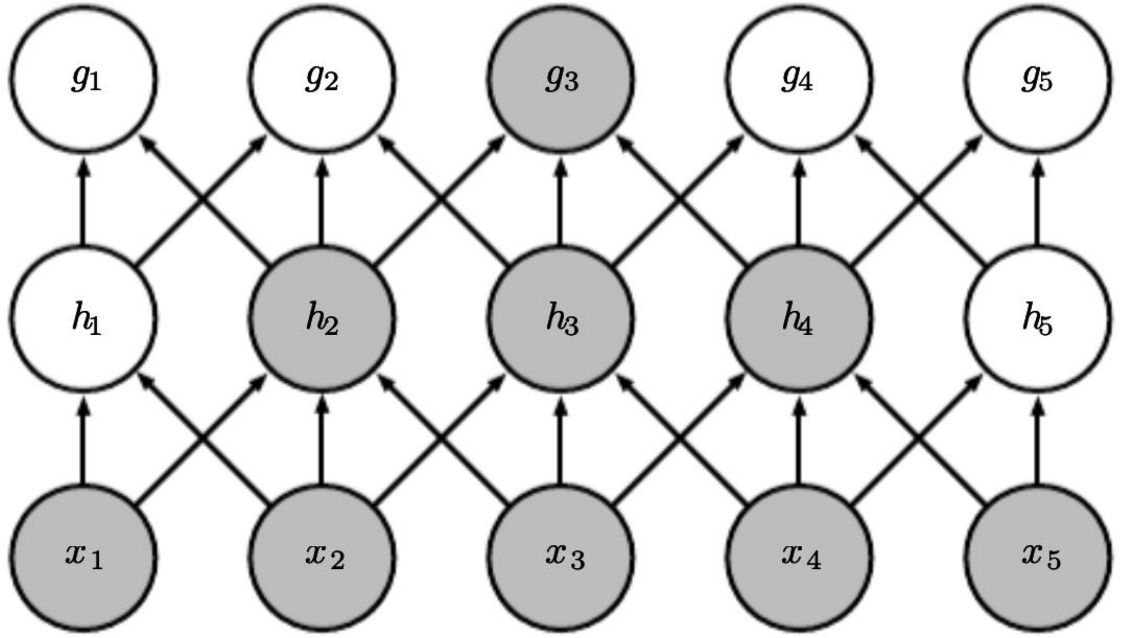
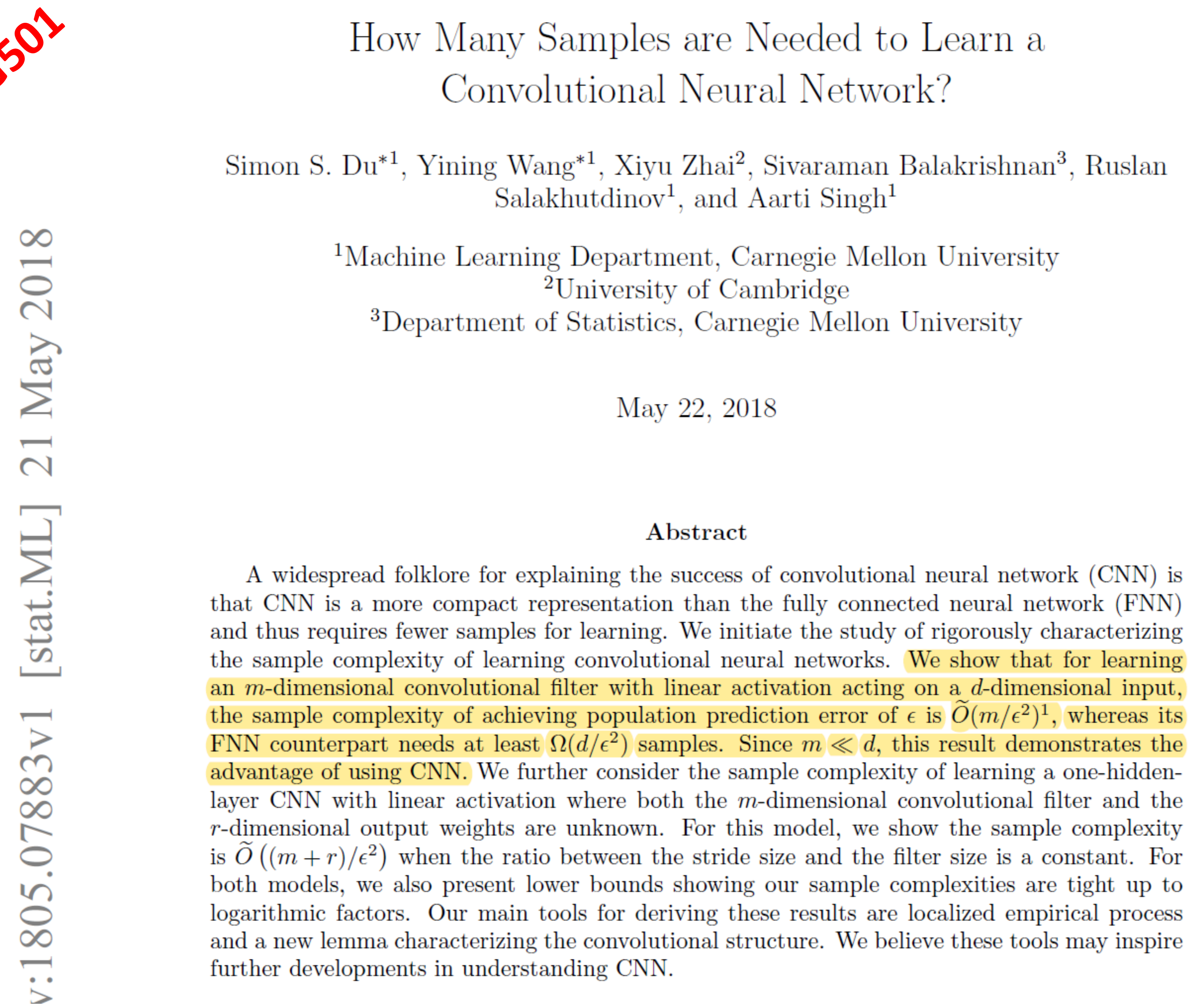


Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

Previously on CENG501



How Many Samples are Needed to Learn a Convolutional Neural Network?

Simon S. Du*[1], Yining Wang*[1], Xiyu Zhai[2], Sivaraman Balakrishnan[3], Ruslan Salakhutdinov[1], and Aarti Singh[1]

[1]Machine Learning Department, Carnegie Mellon University
[2]University of Cambridge
[3]Department of Statistics, Carnegie Mellon University

May 22, 2018

Abstract

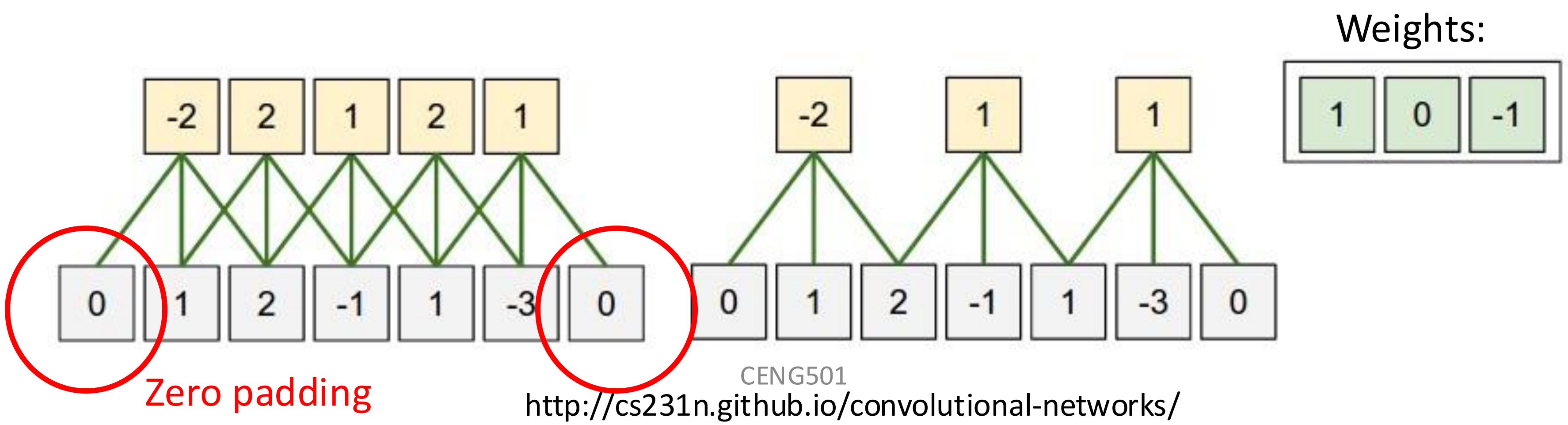
A widespread folklore for explaining the success of convolutional neural network (CNN) is that CNN is a more compact representation than the fully connected neural network (FNN) and thus requires fewer samples for learning. We initiate the study of rigorously characterizing the sample complexity of learning convolutional neural networks. We show that for learning an m-dimensional convolutional filter with linear activation acting on a d-dimensional input, the sample complexity of achieving population prediction error of ϵ is $\widetilde{O}(m/\epsilon^2)$[1], whereas its FNN counterpart needs at least $\Omega(d/\epsilon^2)$ samples. Since $m \ll d$, this result demonstrates the advantage of using CNN. We further consider the sample complexity of learning a one-hidden-layer CNN with linear activation where both the m-dimensional convolutional filter and the r-dimensional output weights are unknown. For this model, we show the sample complexity is $\widetilde{O}\left((m+r)/\epsilon^2\right)$ when the ratio between the stride size and the filter size is a constant. For both models, we also present lower bounds showing our sample complexities are tight up to logarithmic factors. Our main tools for deriving these results are localized empirical process and a new lemma characterizing the convolutional structure. We believe these tools may inspire further developments in understanding CNN.

Size of the next layer

- Along a dimension:
  - W: Size of the input
  - F: Size of the receptive field
  - S: Stride
  - P: Amount of zero-padding
- Then: the number of neurons as the output of a convolution layer:

$$\frac{W - F + 2P}{S} + 1$$

- If this number is not an integer, your strides are incorrect and your neurons cannot tile nicely to cover the input volume



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

Types of Convolution

Previously on CENG501

- Unshared
- Dilated
- Transposed
- 3D
- 1x1
- Separable and Depth-wise Separable
- Group
- Deformable
- Position-sensitive

Pooling

- 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.

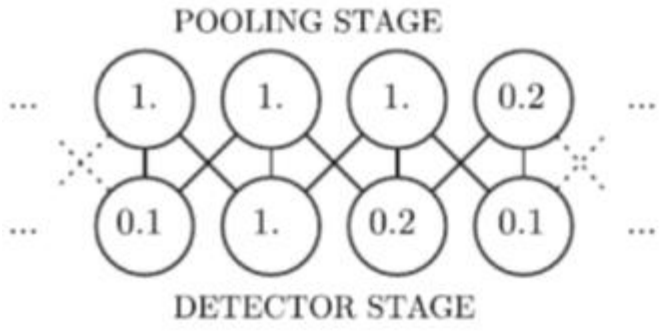
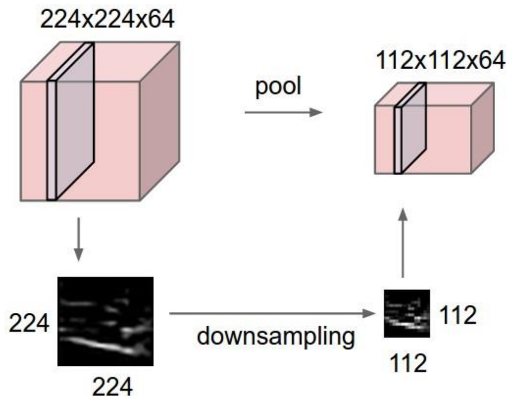
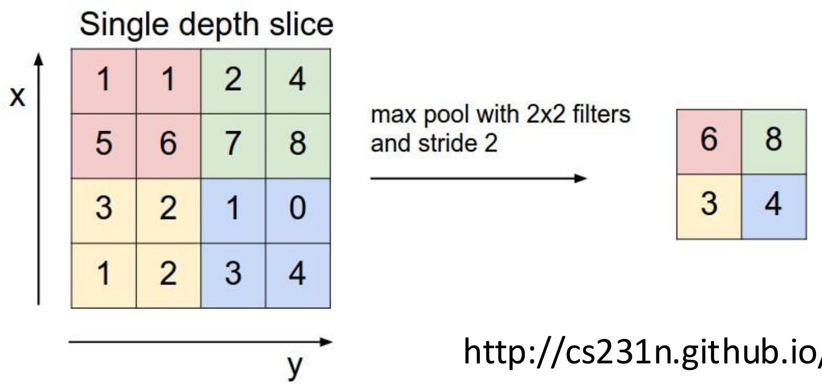


Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

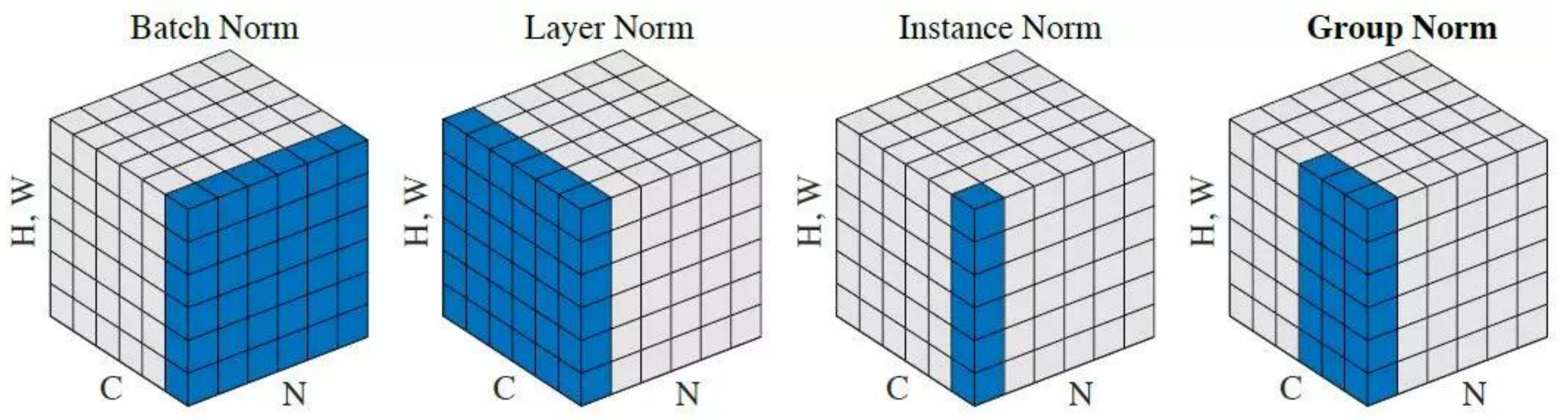




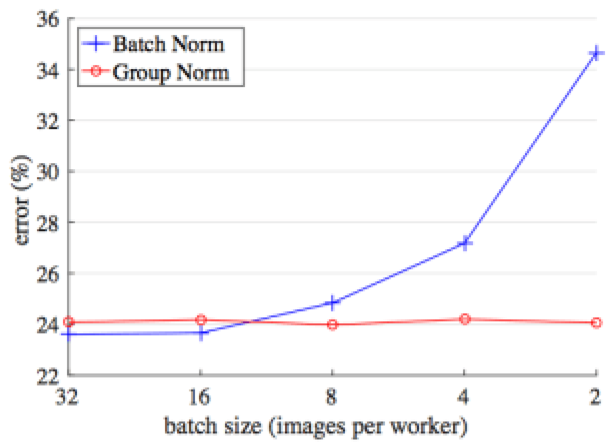
http://cs231n.github.io/convolutional-networks/

Normalization

Previously on CENG501



For each channel independently.

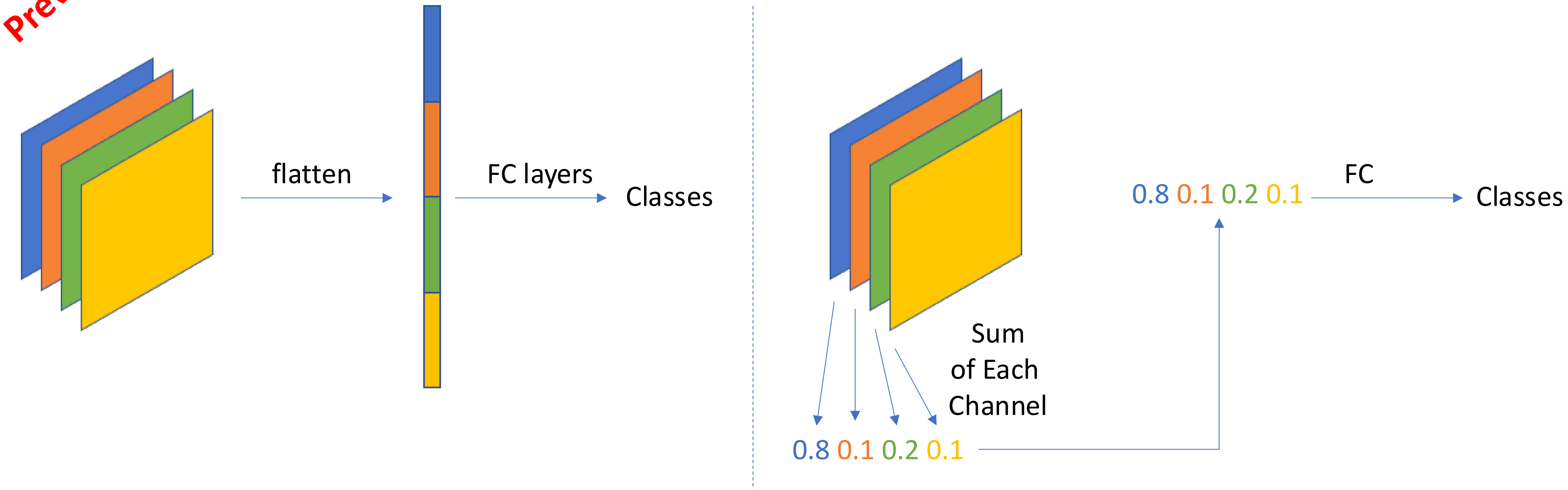


https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

Previously on CENG501

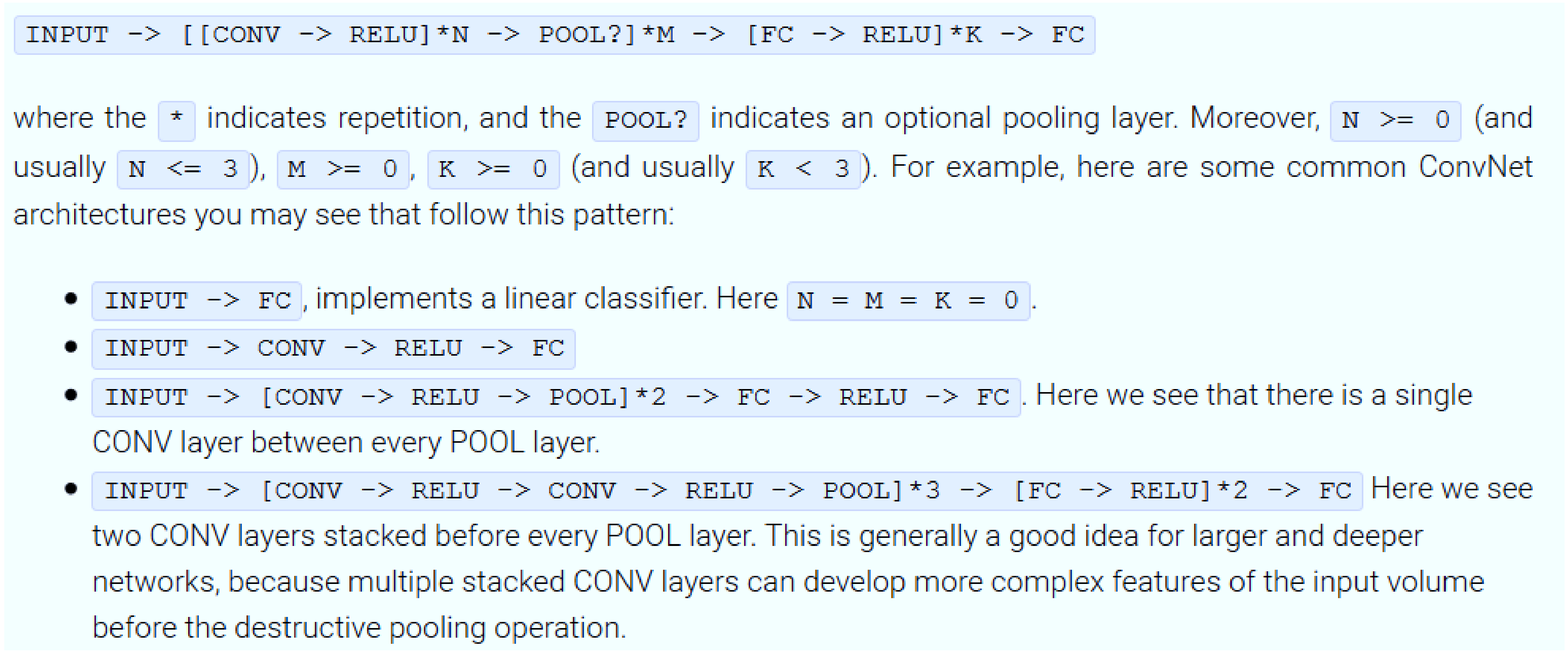
Alternative to FC: Global Average Pooling

“Network In Network”, https://arxiv.org/pdf/1312.4400.pdf



Previously on CENG501

A Blueprint for CNNs



```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

where the `*` indicates repetition, and the `POOL?` indicates an optional pooling layer. Moreover, `N >= 0` (and usually `N <= 3`), `M >= 0`, `K >= 0` (and usually `K < 3`). For example, here are some common ConvNet architectures you may see that follow this pattern:

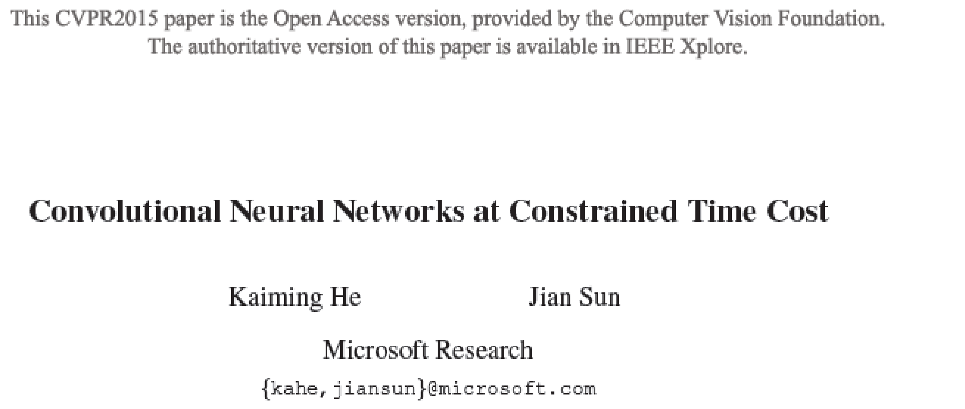
- `INPUT -> FC`, implements a linear classifier. Here `N = M = K = 0`.
- `INPUT -> CONV -> RELU -> FC`
- `INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC`. Here we see that there is a single CONV layer between every POOL layer.
- `INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC` Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.

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

Trade-offs in architecture

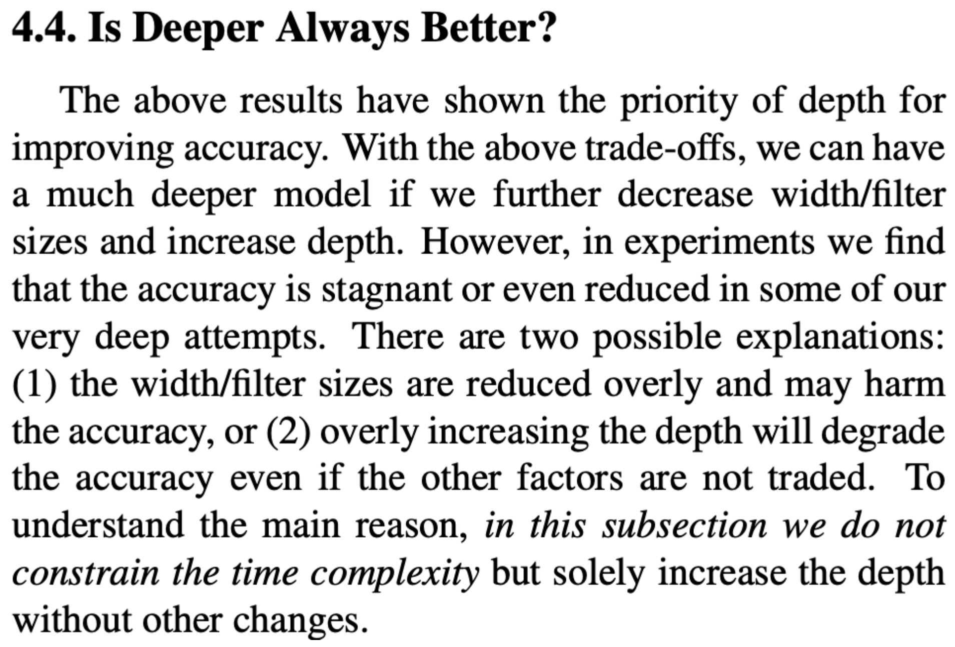
Previously on CENG501

- 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.

Previously on CENG501

Finetuning

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

Previously on CENG501

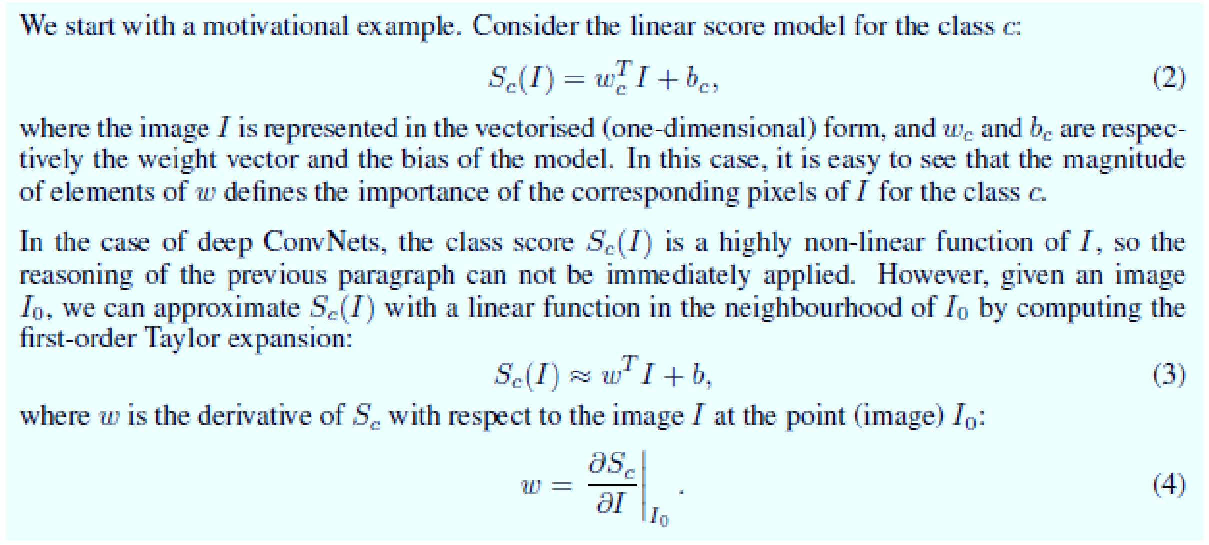
Many mechanisms for visualization

- Visualize layer activations
- Visualize the weights (i.e., filters)
- Visualize examples that maximally activate a neuron
- Visualize a 2D embedding of the inputs based on their CNN codes
- Occlude parts of the window and see how the prediction is affected
- Data gradients

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:

$$S_c(I) = w_c^T I + b_c, \quad (2)$$

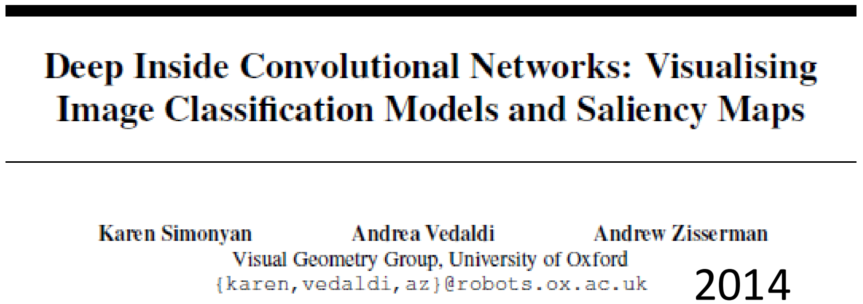
where the image I is represented in the vectorised (one-dimensional) form, and w_c and b_c are respectively the weight vector and the bias of the model. In this case, it is easy to see that the magnitude of elements of w defines the importance of the corresponding pixels of I for the class c.

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:

$$S_c(I) \approx w^T I + b, \quad (3)$$

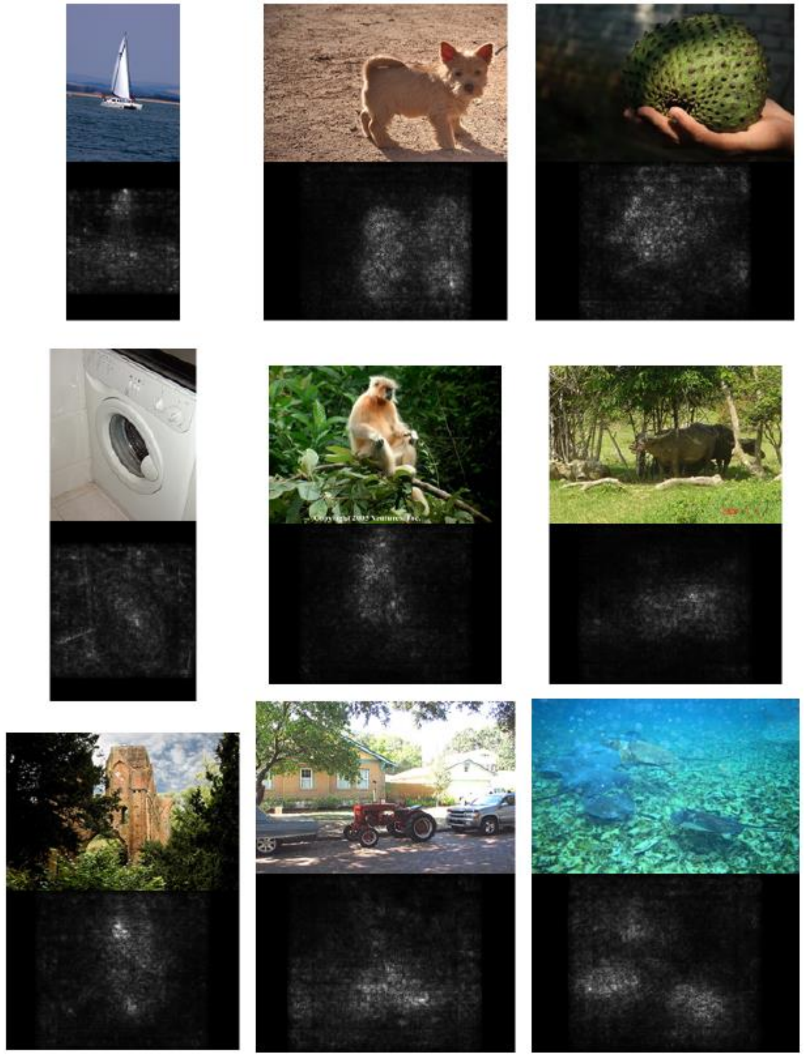
where w is the derivative of S_c with respect to the image I at the point (image) I_0:

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


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



Class Activation Maps

- Weighted combination of the feature maps before GAP:

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

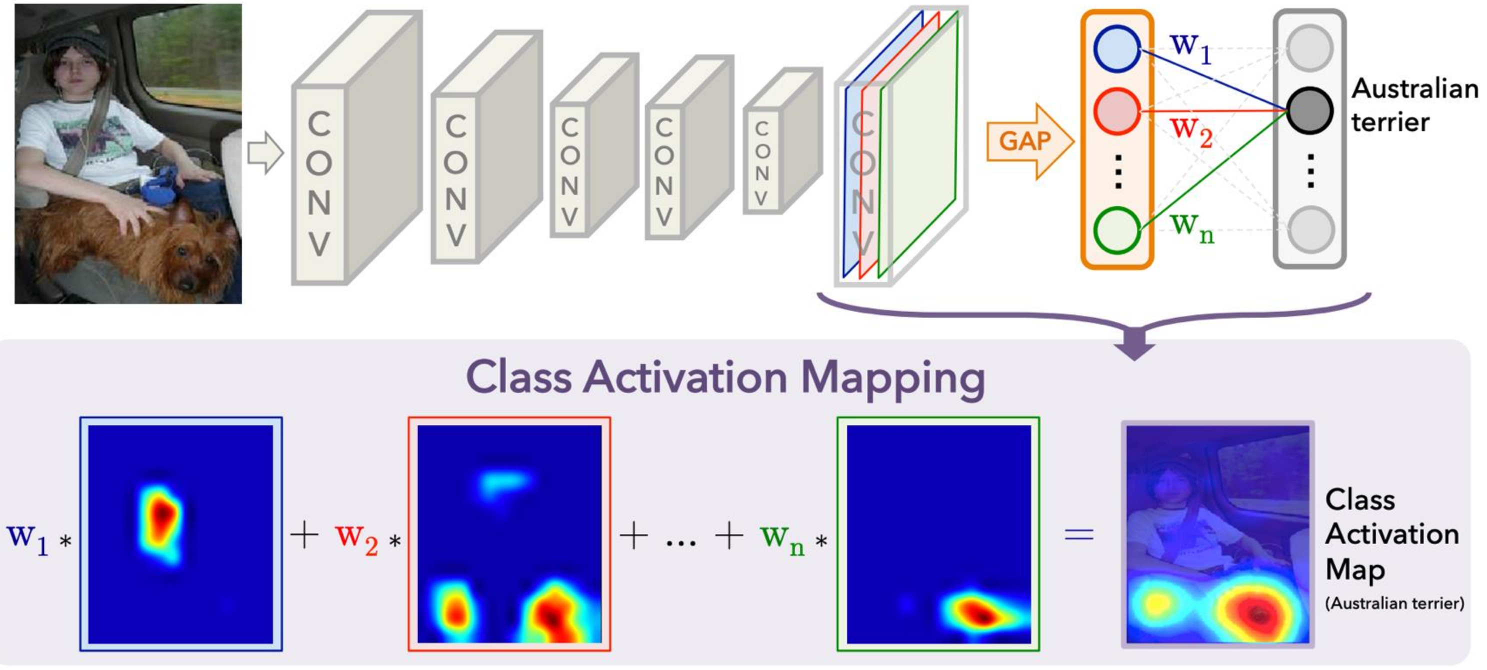


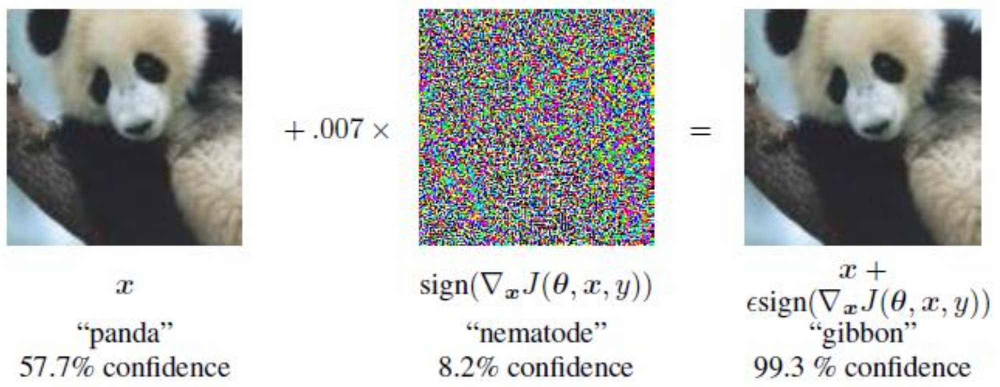
Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

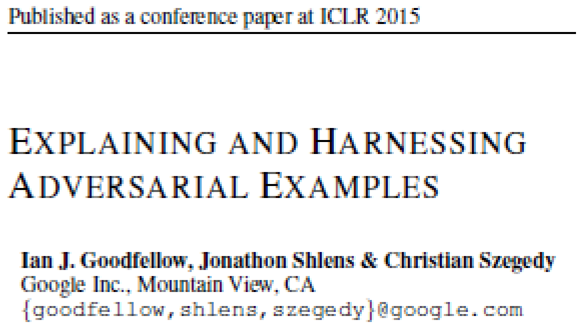
B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2921–2929.

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:

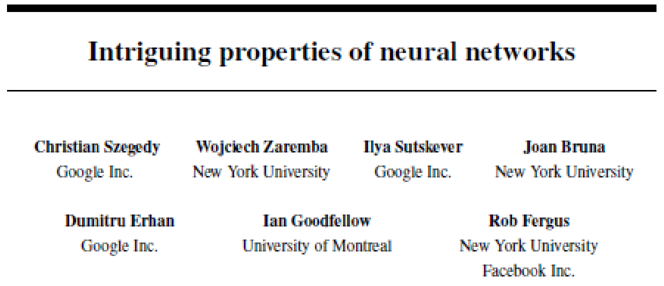
$$\arg\min_{r} loss(I+r, l_2) + c|r|$$



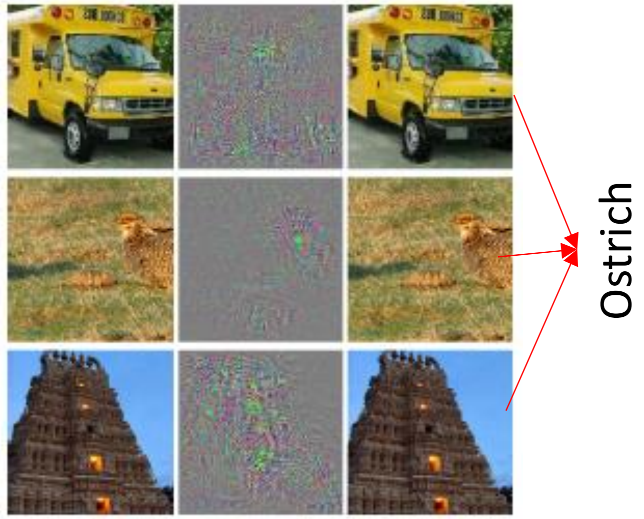

Published as a conference paper at ICLR 2015

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.



Today

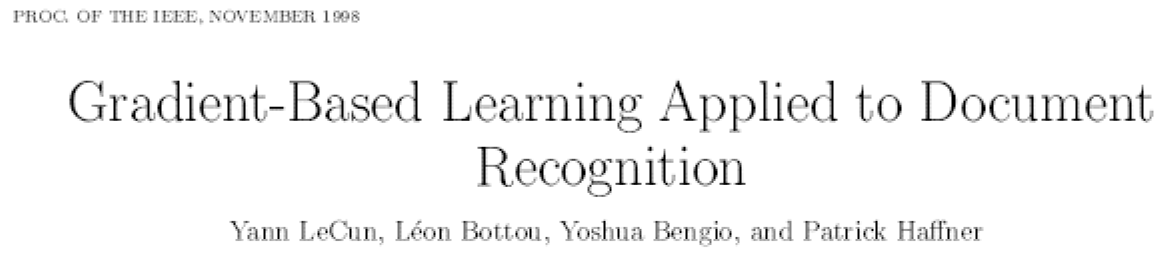
- CNNs
    - Popular CNN architectures
- Sequence Modeling
- Recurrent Neural Networks
- Long Short Term Memory (LSTM)
- Application: Language Modeling
- Application: Image Captioning
- Application: Machine Translation
- Echo State Networks

Administrative Notes

- No quiz today
- Hopfield Networks, Boltzmann Machines and the Associated Nobel Prize
  - Today at 18:30 in “Cavid Erginsoy Salonu” in the Dept. of Physics
- Paper Selection
  - Feedback provided
  - Deadline: This Sunday

Popular
CNN models

LeNet (1998)

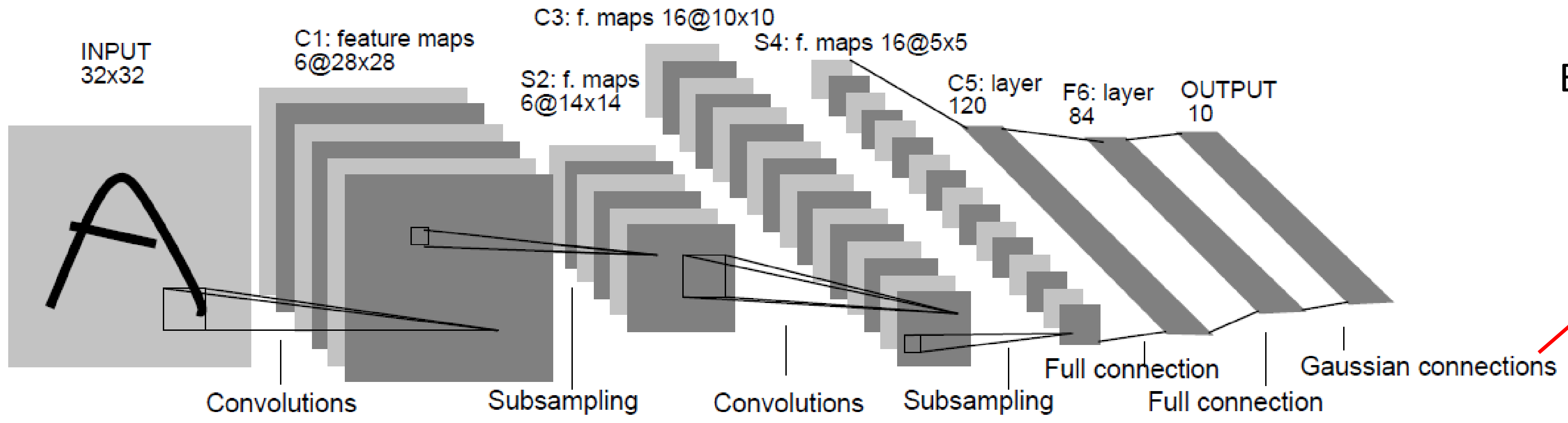


PROC. OF THE IEEE, NOVEMBER 1998

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

- For reading zip codes and digits



Euclidean RBF:

$$y_i = \sum_j (x_j - w_{ij})^2.$$

F=5x5
S=1
P=0

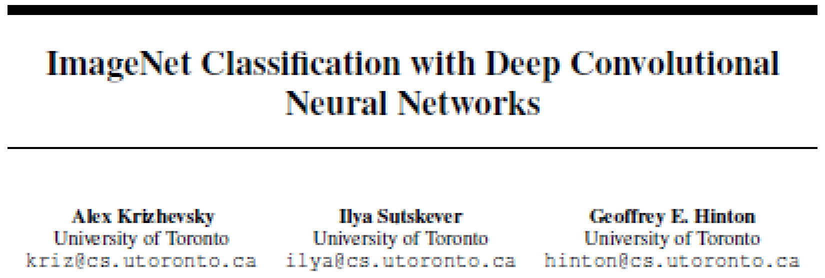
F=2x2
S=2
sigm($\alpha \times avg + b$)
α & b: trainable

F=5x5
S=1
P=0

F=2x2
S=2
sigm($\alpha \times avg + b$)
α & b: trainable

F=5x5
S=1
P=0

AlexNet (2012)



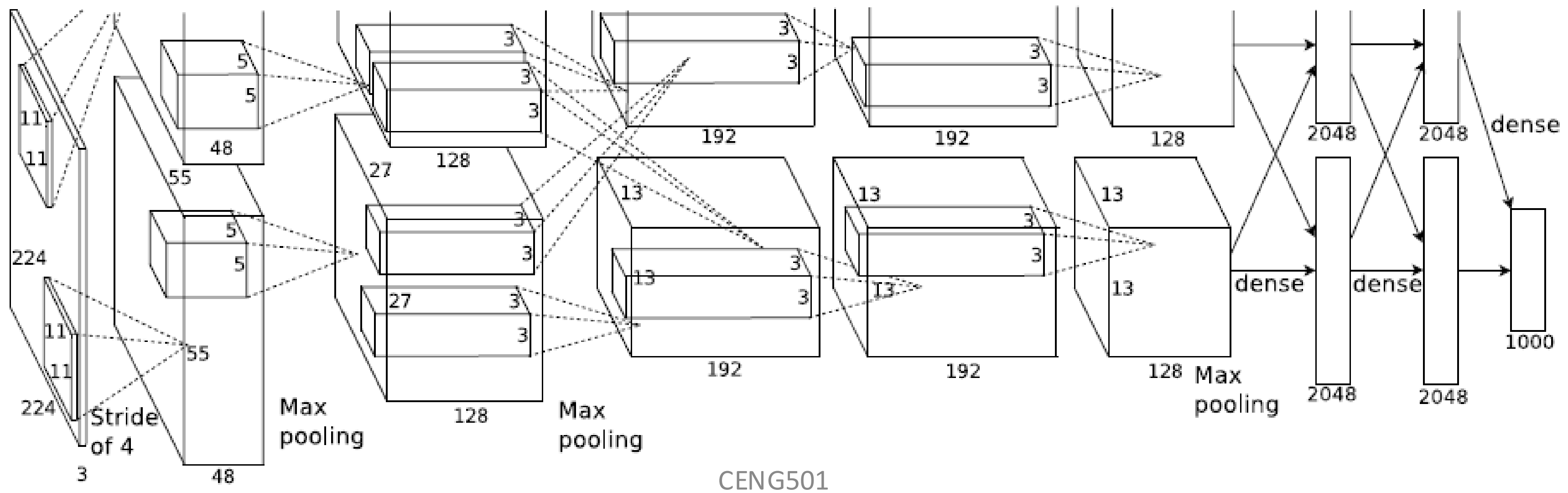
ImageNet Classification with Deep Convolutional Neural Networks

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

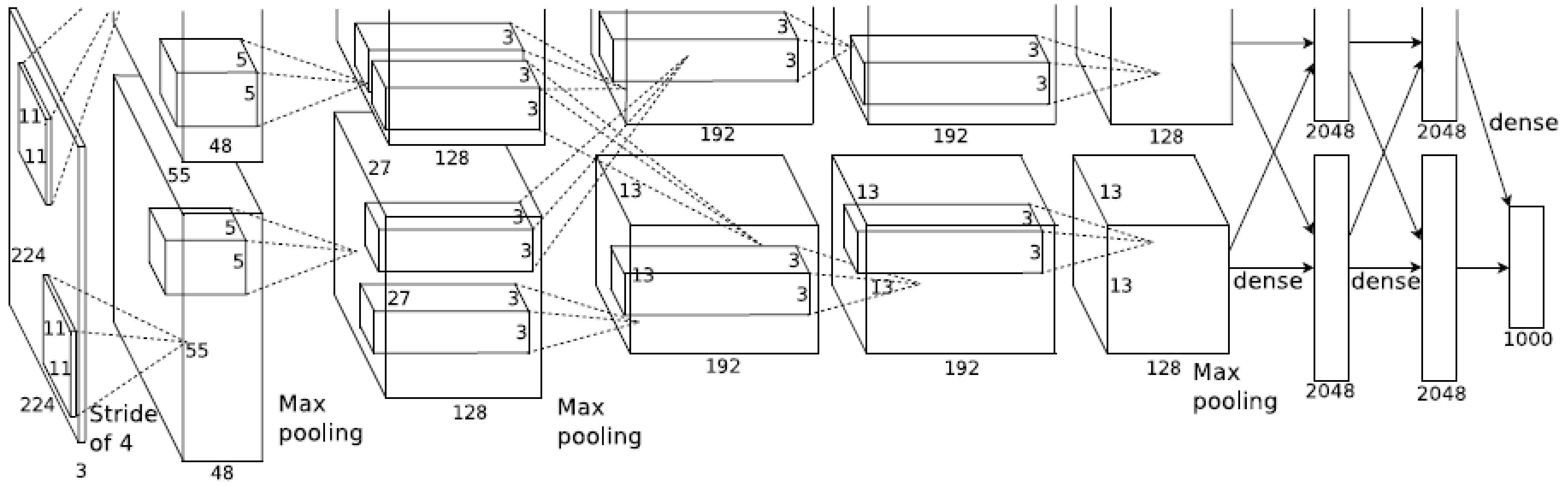
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

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.

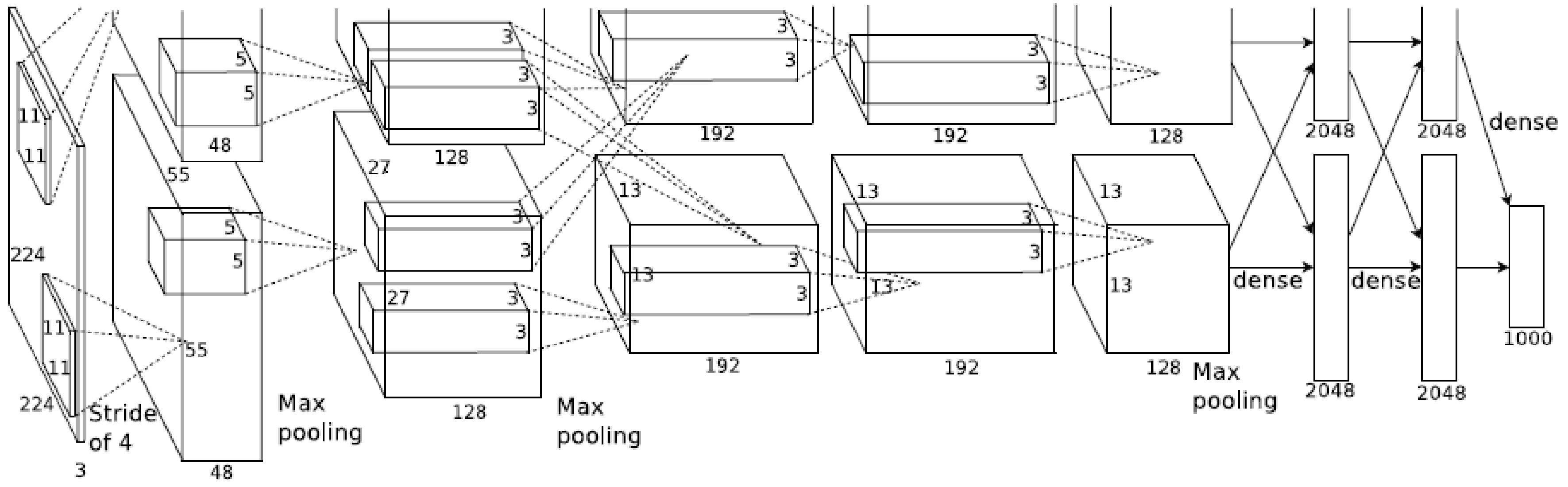


AlexNet (2012) Details



- Since the network is too big to fit in on GPU, it is divided into two.
- Note the cross connections between the “pathways”.
- Uses ReLU as non-linearity after every convolutional and fully-connected layer.
- Normalization layer is placed after the first & the second convolutional layers.
- Max-pooling layer is placed only after the normalization layers & the fifth convolutional layer.
- Last layer is a soft-max.

AlexNet (2012) Training



- Data augmentation & dropout are used during training to avoid overfitting.
- Stochastic Gradient Descent with a batchsize of 128 examples is used.
- Momentum with coefficient 0.9 is employed.
- Weight decay (L2 regularization cost) with factor 0.0005 is also used in the loss function.
- Weights are initialized from a zero-mean Gaussian distribution with 0.01 std.
- Learning rate started with 0.01 and manually divided by 10 when the validation error rate stopped improving.
- Trained on 1.2 million images, which took 5-6 days on two GPUs.

AlexNet (2012): The learned filters

- Do you notice anything strange with the filters?

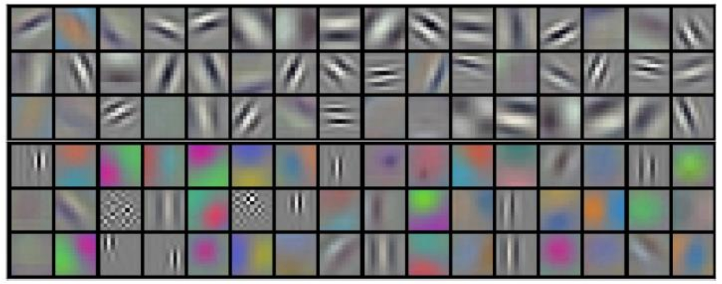


Figure 3: 96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

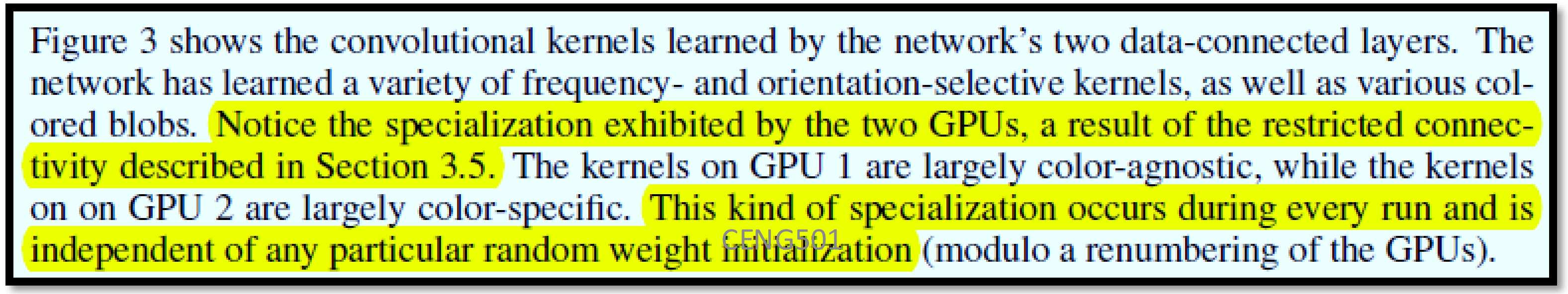
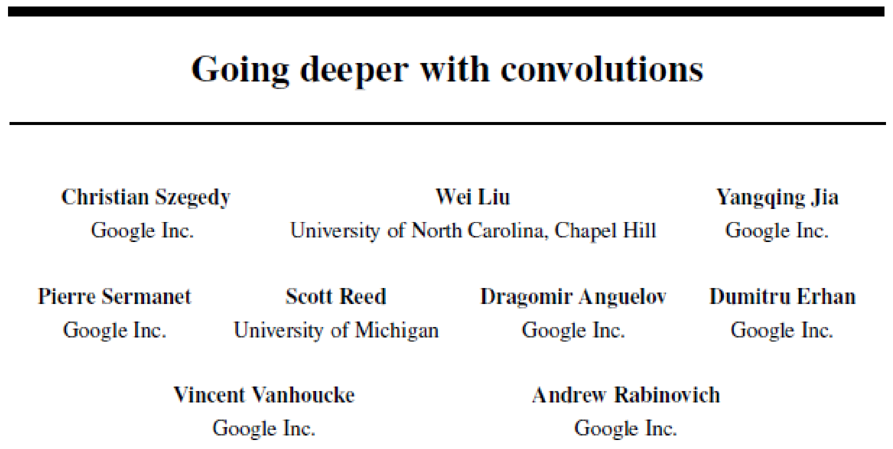


Figure 3 shows the convolutional kernels learned by the network's two data-connected layers. The network has learned a variety of frequency- and orientation-selective kernels, as well as various colored blobs. Notice the specialization exhibited by the two GPUs, a result of the restricted connectivity described in Section 3.5. The kernels on GPU 1 are largely color-agnostic, while the kernels on on GPU 2 are largely color-specific. This kind of specialization occurs during every run and is independent of any particular random weight initialization (modulo a renumbering of the GPUs).

GoogleNet (2014)



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

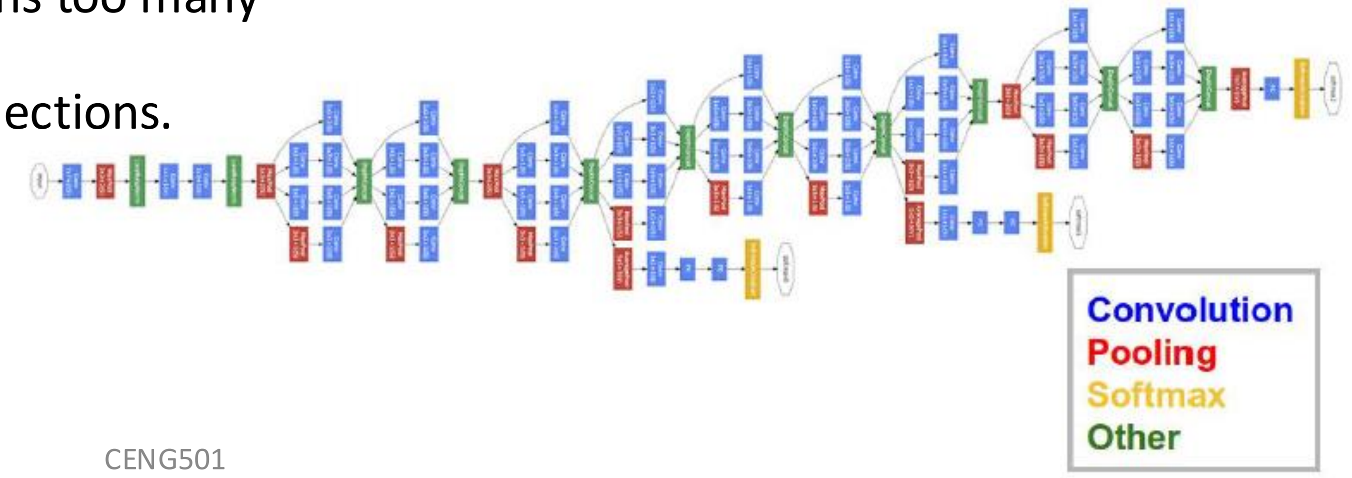
Dragomir Anguelov
Google Inc.

Dumitru Erhan
Google Inc.

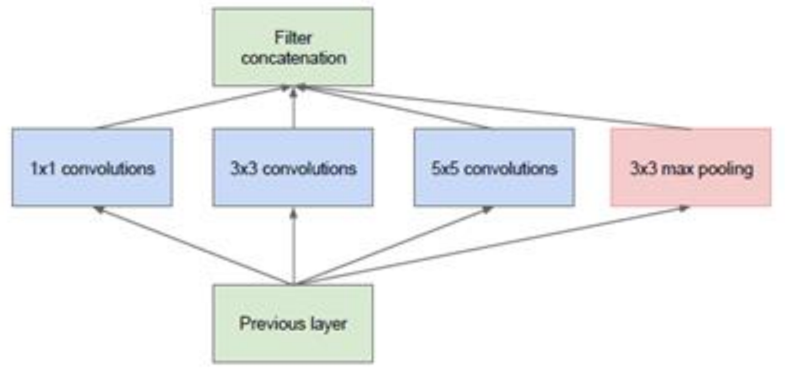
Vincent Vanhoucke
Google Inc.

Andrew Rabinovich
Google Inc.

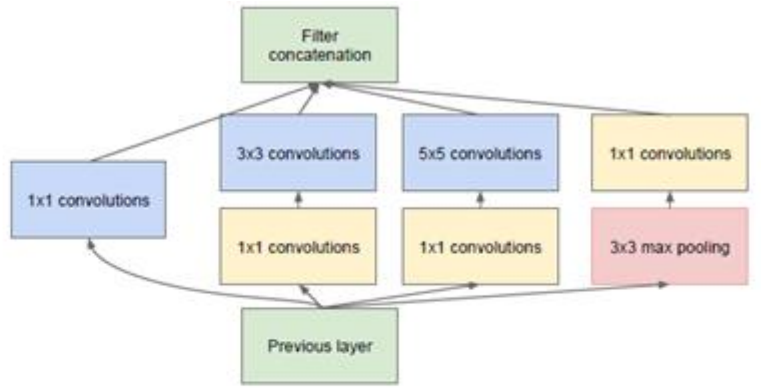
- ImageNet 2014 winner
- Contributions:
  - Inception module
    - Dramatically reduced parameters (from 60M in AlexNet to 4M)
  - Avg Pooling at the top, instead of fully-connected layer ➔ Reduced number of parameters
- Motivation:
  - Going bigger (in depth or width) means too many parameters.
  -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



Inception module: “network in network” (inspired from Lin et al., 2013)

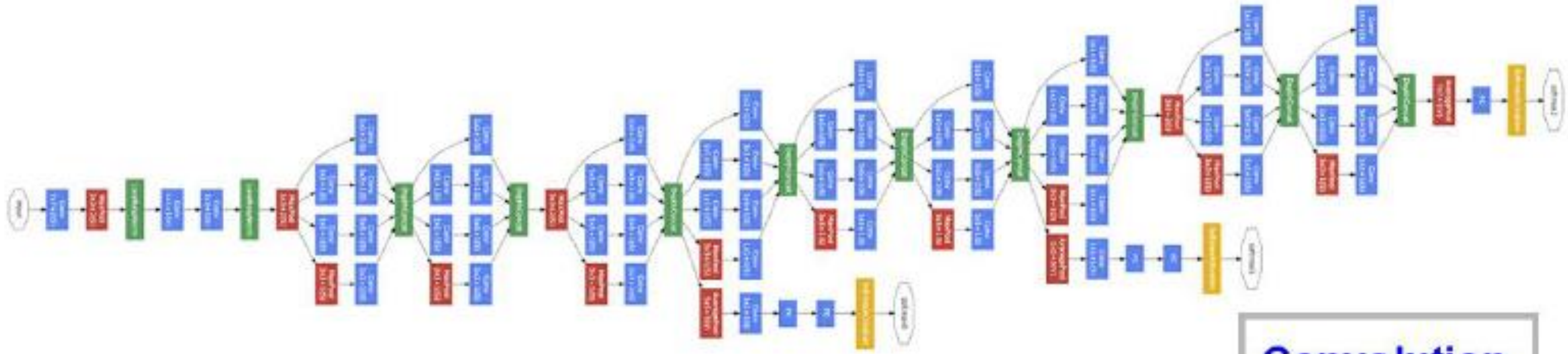


(a) Inception module, naïve version

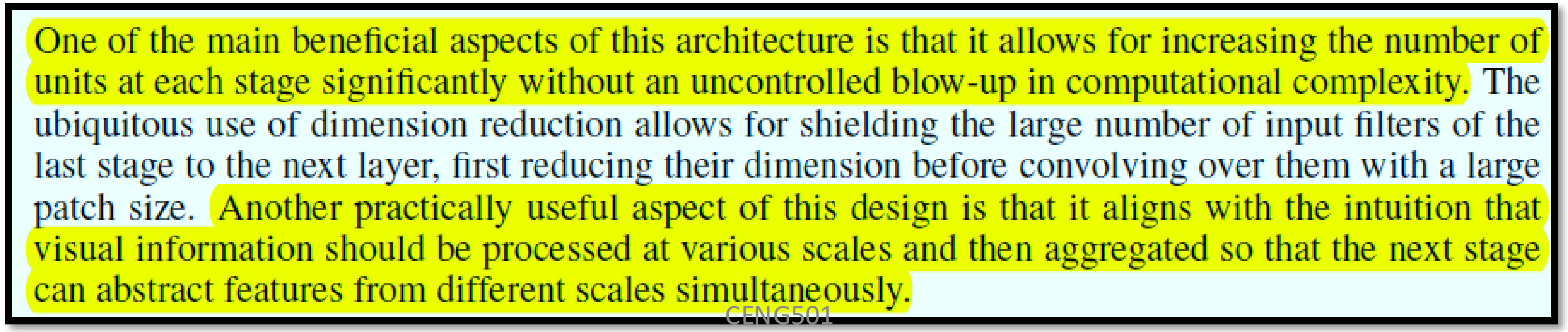


(b) Inception module with dimension reductions

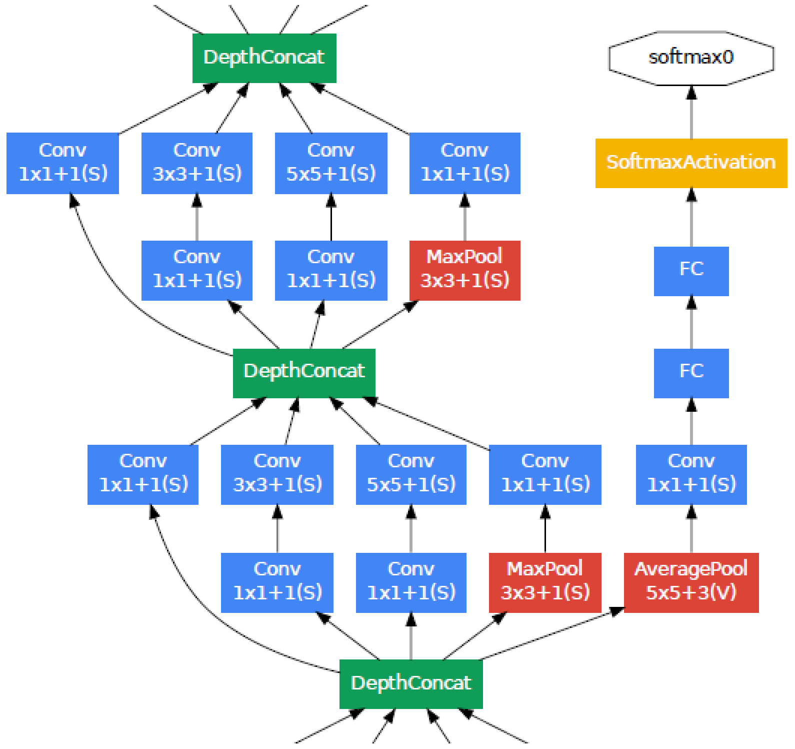
- Concatenation is performed along the “columns” (depth).
  - The output of inception layers must have the same size.
- The naïve version has a tendency to blow up in number of channels.
  - Why? Max-pooling does not change the number of channels. When concatenated with other filter responses, number of channels increase with every layer.
  - Solution: Do 1x1 convolution to decrease the number of channels.
    - Also called “bottleneck”.
- In order to decrease the computational complexity of 3x3 and 5x5 pooling, they are also preceded by 1x1 convolution (i.e., the number of channels are reduced).



Convolution
Pooling
Softmax
Other


One of the main beneficial aspects of this architecture is that it allows for increasing the number of units at each stage significantly without an uncontrolled blow-up in computational complexity. The ubiquitous use of dimension reduction allows for shielding the large number of input filters of the last stage to the next layer, first reducing their dimension before convolving over them with a large patch size. Another practically useful aspect of this design is that it aligns with the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously.

- Since the intermediate layers learn to discriminate features specific to a class, we can directly link them to the loss term.
  - Encourages these layers to become more discriminative
  - Increases propagation of gradient signal to earlier stages



GoogleNet: More Details

- ReLU after all layers
- Max pooling in inception modules as well as a whole layer occasionally
- Avg pooling instead of fully-connected layers
  - Only a minor change in the accuracy (0.6%)
  - However, less number of parameters
- Other usual tricks (e.g., dropout, augmentation etc.) are used.
- Trained on CPUs using a distributed machine learning system.
- SGD with momentum (0.9).
- Fixed learning rate scheme with 4% decrease every 8 epochs
- They trained many different models with different initializations and parameters. They combined these models using different methods and tricks. There is no single training method that yields the results they achieved.

VGGNet (2014)

- ImageNet runner up in 2014
- Contribution:
  - Use small RFs & increase depth as much as possible
  - 16 CONV/FC layers.
  - 3x3 CONVs and 2x2 pooling from beginning to the end
- Although performs slightly worse than GoogleNet in image classification, VGGNet may perform better at other tasks (such as transfer learning problems).
- Downside: Needs a lot of memory & parameters (140M)

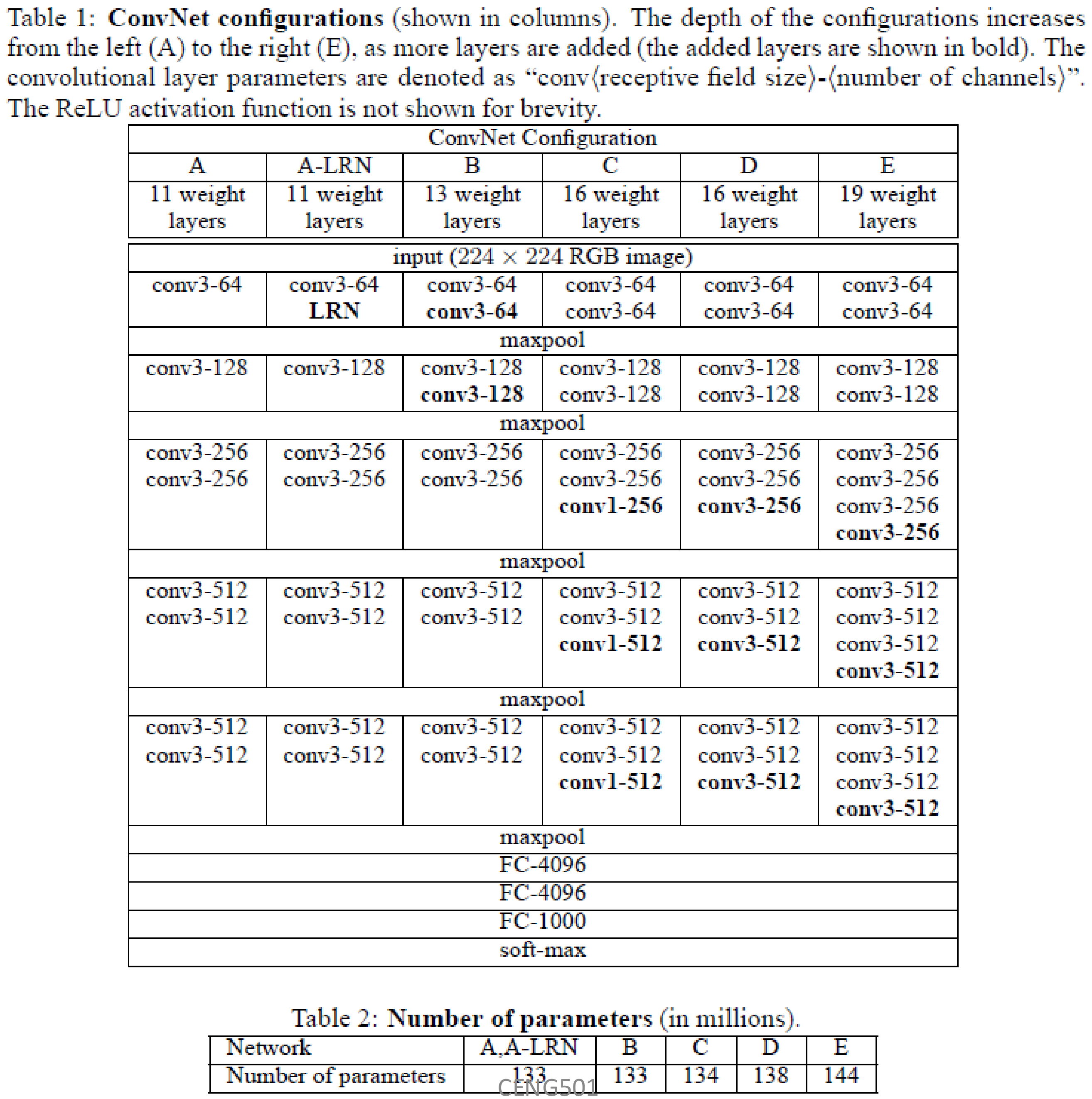


Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The 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

ResNet (2015)

Deep Residual Learning for Image Recognition

Kaiming He   Xiangyu Zhang   Shaoqing Ren   Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

- Increasing the depth naively may not give you better performance after a number of depths
  - Why?
    - This is shown to be not due to overfitting (since training error also gets worse) or vanishing gradients (suitable non-linearities used)
    - Accuracy is somehow saturated. Though reported in several studies.
- Solution: Make shortcut connections

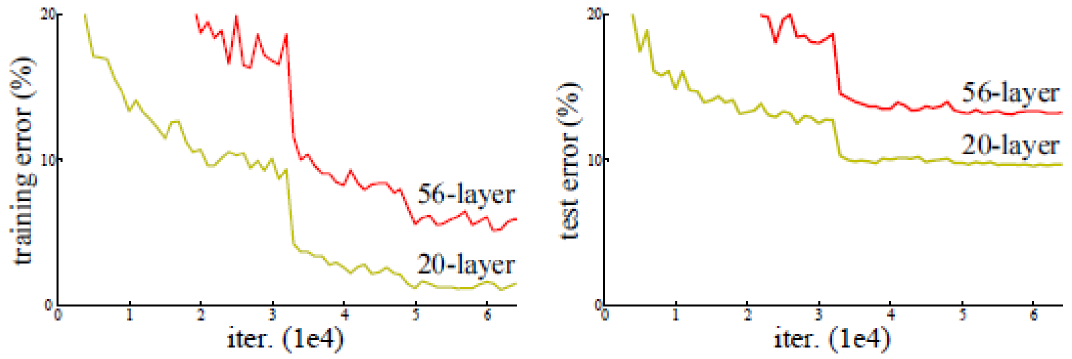


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

ResNet (2015)

- Residual (shortcut) connections

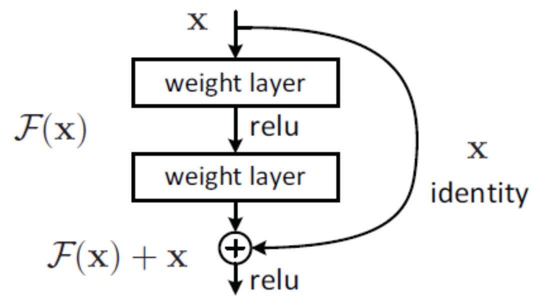

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.

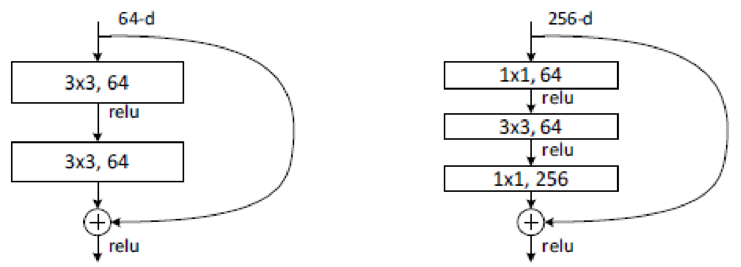
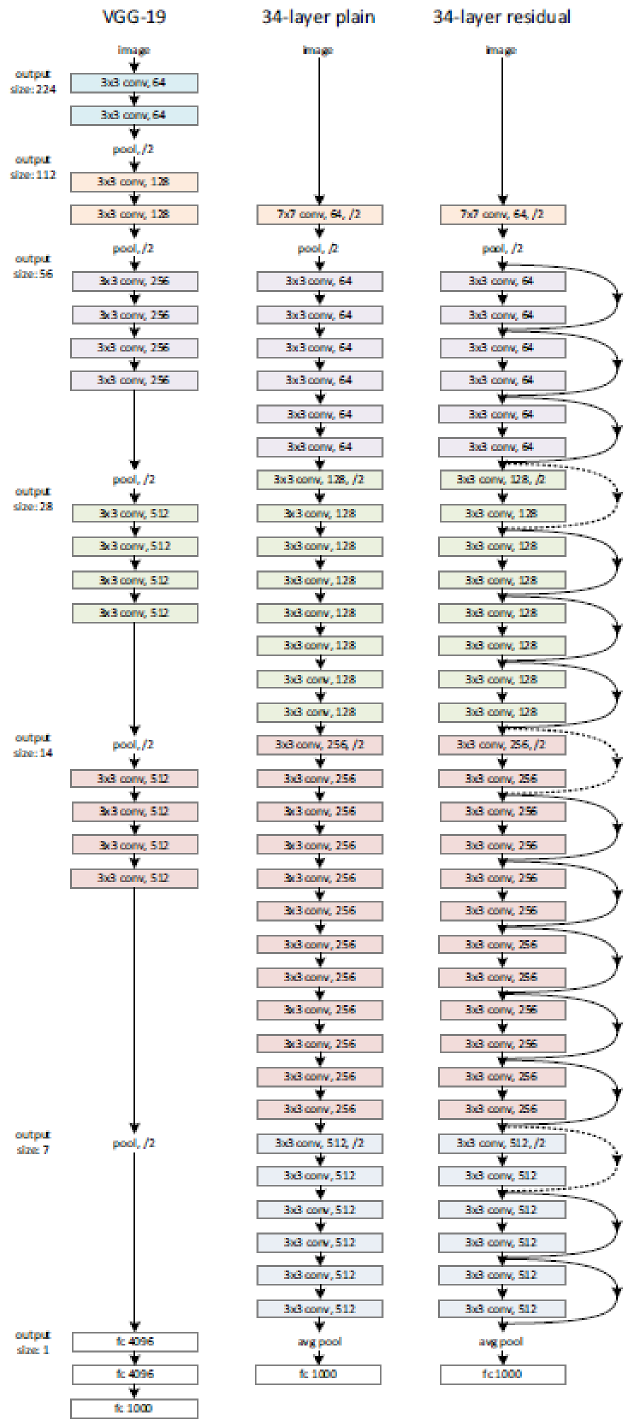


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.



ResNet (2015)

- Residual (shortcut) connections

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43[†]
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except [†] reported on the test set).

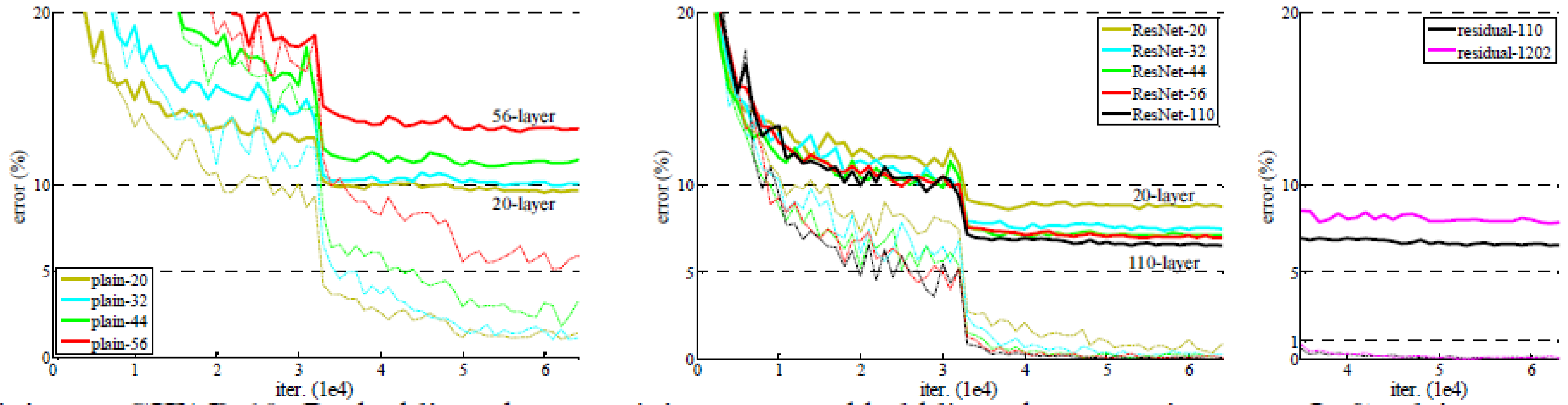
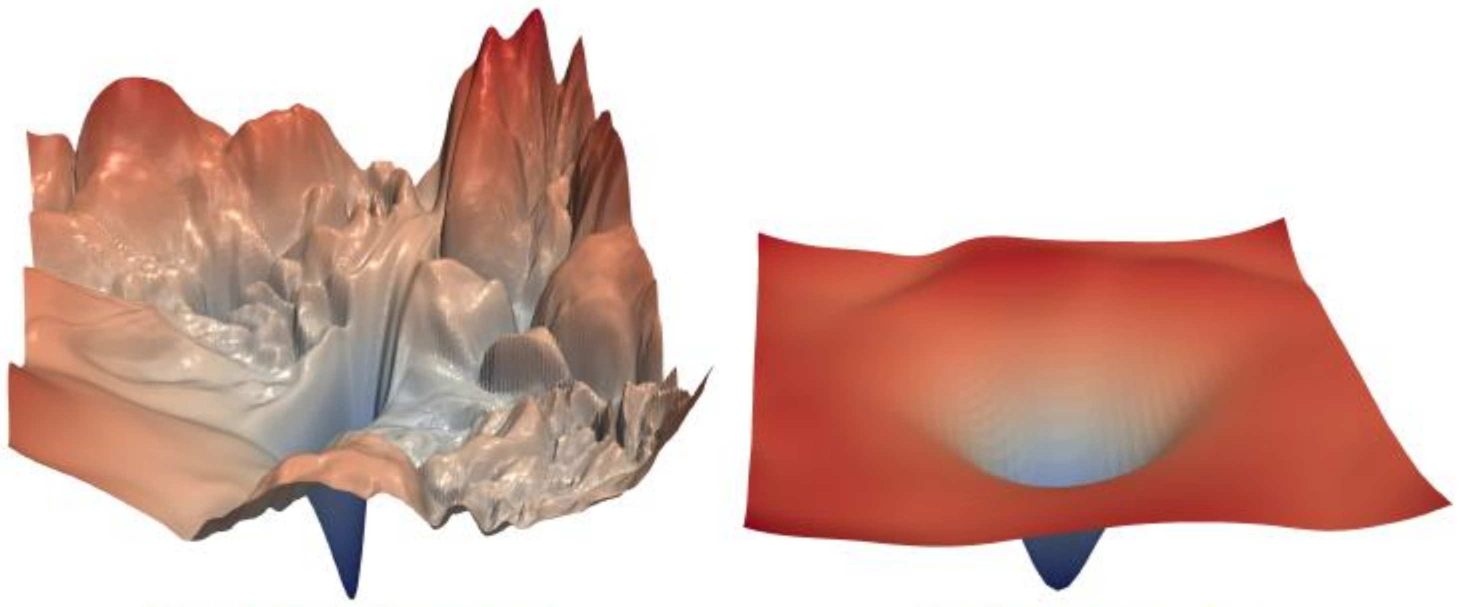


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left**: plain networks. The error of plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1202 layers.

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

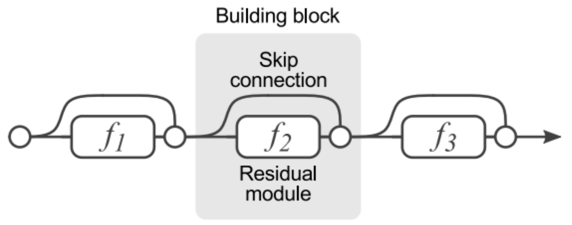
Hao Li[1], Zheng Xu[1], Gavin Taylor[2], Christoph Studer[3], Tom Goldstein[1]
[1]University of Maryland, College Park, [2]United States Naval Academy, [3]Cornell University
{haoli,xuzh,tomg}@cs.umd.edu, taylor@usna.edu, studer@cornell.edu

ResNet: Ensemble of Shallow Networks

Residual Networks Behave Like Ensembles of Relatively Shallow Networks

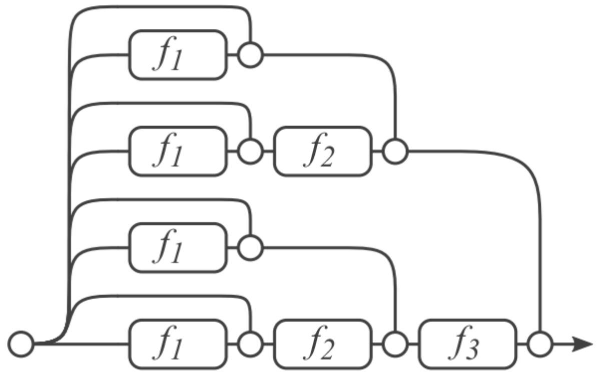
Andreas Veit **Michael Wilber** **Serge Belongie**
Department of Computer Science & Cornell Tech
Cornell University
{av443, mjw285, sjb344}@cornell.edu

2016



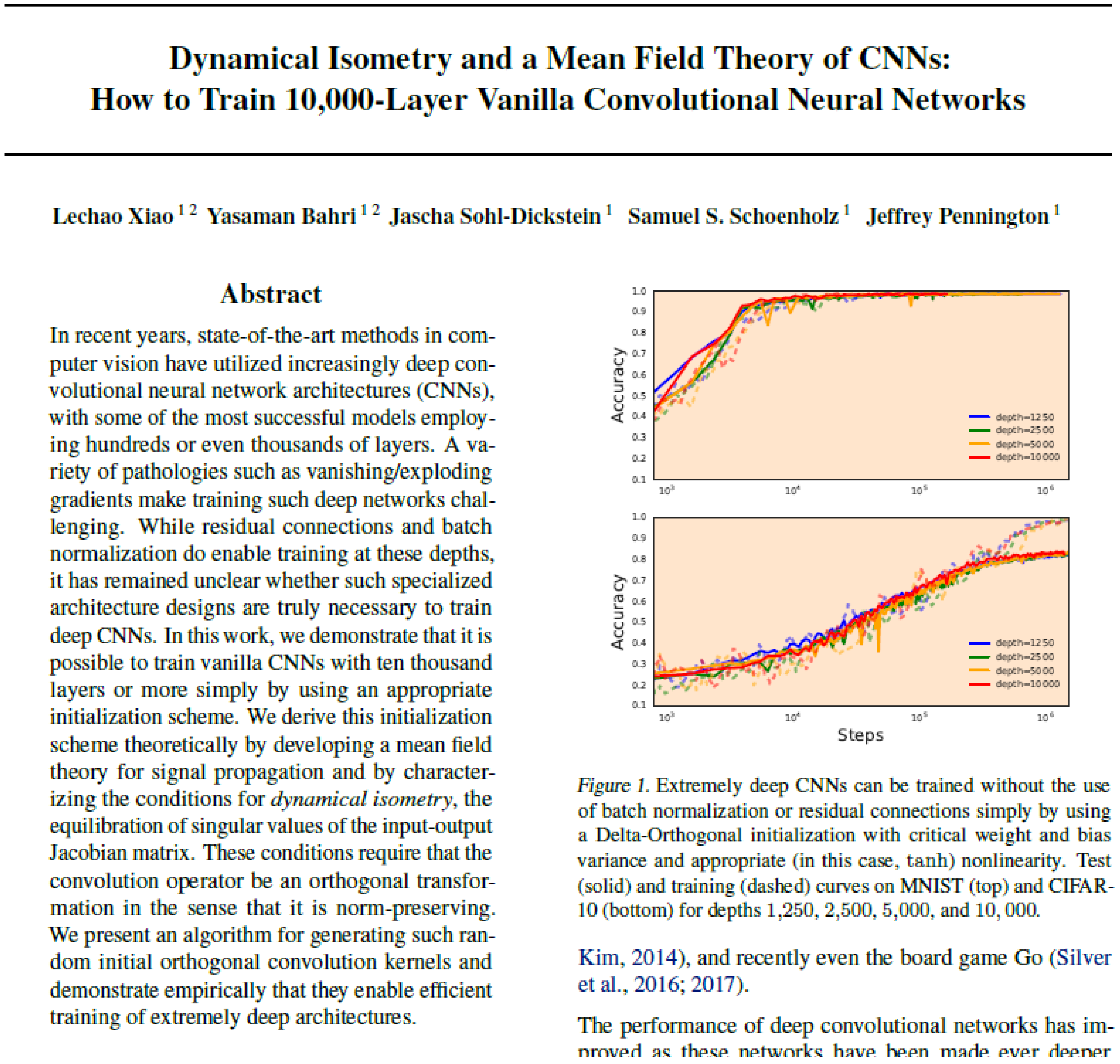
(a) Conventional 3-block residual network

=



(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.



Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks

Lechao Xiao [1 2] Yasaman Bahri [1 2] Jascha Sohl-Dickstein [1] Samuel S. Schoenholz [1] Jeffrey Pennington [1]

Abstract

In recent years, state-of-the-art methods in computer vision have utilized increasingly deep convolutional neural network architectures (CNNs), with some of the most successful models employing hundreds or even thousands of layers. A variety of pathologies such as vanishing/exploding gradients make training such deep networks challenging. While residual connections and batch normalization do enable training at these depths, it has remained unclear whether such specialized architecture designs are truly necessary to train deep CNNs. In this work, we demonstrate that it is possible to train vanilla CNNs with ten thousand layers or more simply by using an appropriate initialization scheme. We derive this initialization scheme theoretically by developing a mean field theory for signal propagation and by characterizing the conditions for *dynamical isometry*, the equilibration of singular values of the input-output Jacobian matrix. These conditions require that the convolution operator be an orthogonal transformation in the sense that it is norm-preserving. We present an algorithm for generating such random initial orthogonal convolution kernels and demonstrate empirically that they enable efficient training of extremely deep architectures.

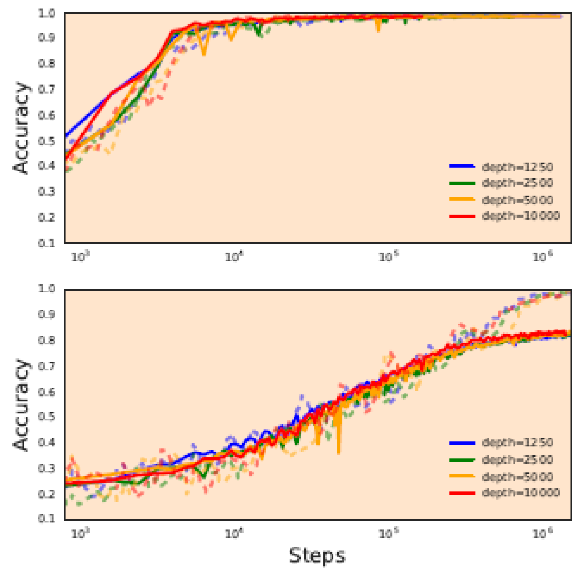


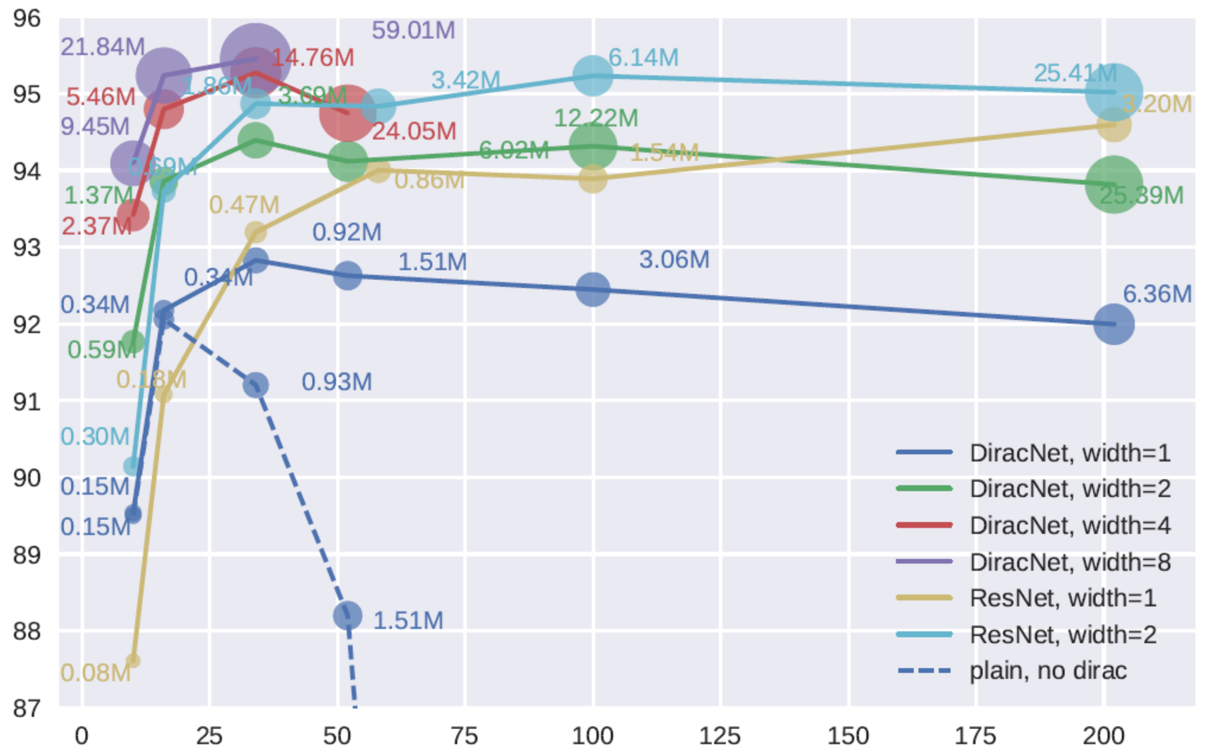
Figure 1. Extremely deep CNNs can be trained without the use of batch normalization or residual connections simply by using a Delta-Orthogonal initialization with critical weight and bias variance and appropriate (in this case, tanh) nonlinearity. Test (solid) and training (dashed) curves on MNIST (top) and CIFAR-10 (bottom) for depths 1,250, 2,500, 5,000, and 10, 000.

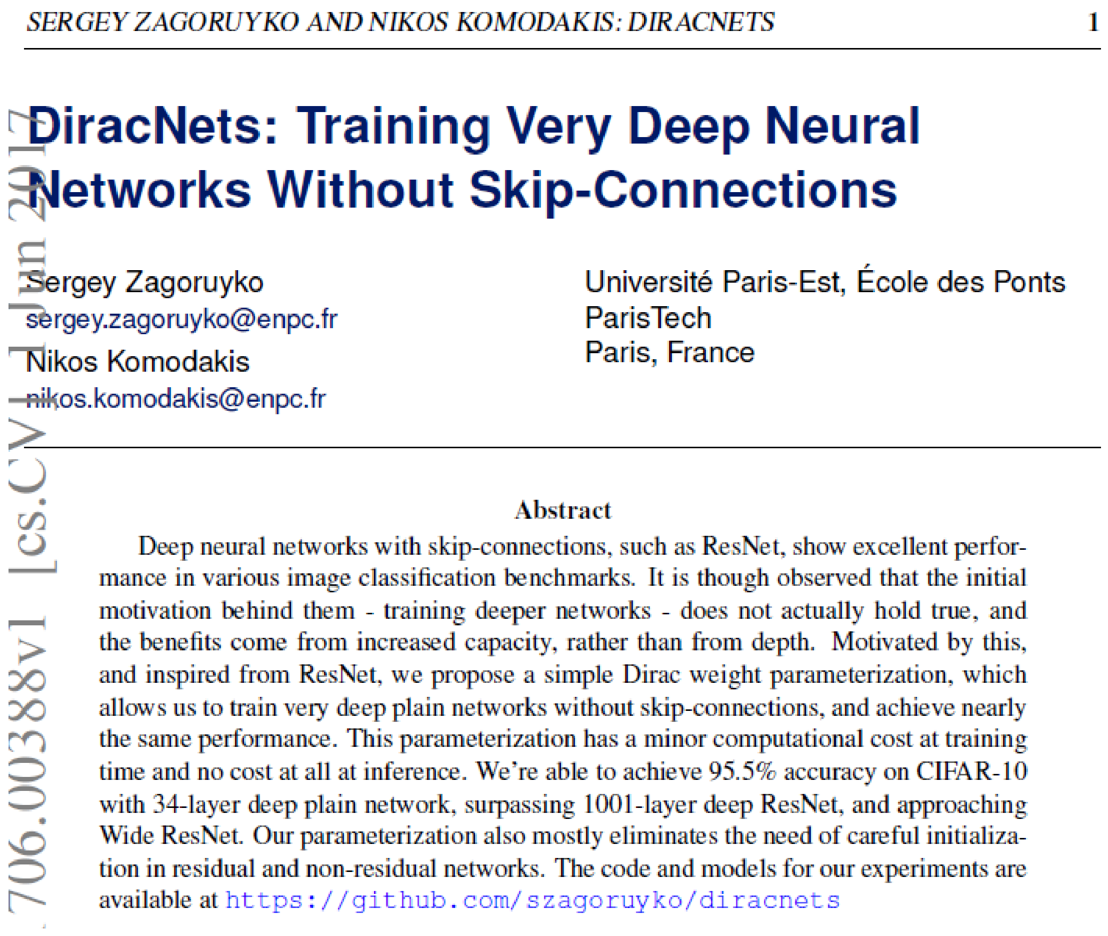
Kim, 2014), and recently even the board game Go (Silver et al., 2016; 2017).

The performance of deep convolutional networks has improved as these networks have been made ever deeper

"Our initial results suggest that past a certain depth, on the order of tens or hundreds of layers, the test performance for vanilla convolutional architecture saturates. These observations suggest that architectural features such as residual connections and batch normalization are likely to play an important role in defining a good model class, rather than simply enabling efficient training."

DiracNets





DiracNets: Training Very Deep Neural Networks Without Skip-Connections

Sergey Zagoruyko
sergey.zagoruyko@enpc.fr
Nikos Komodakis
nikos.komodakis@enpc.fr
Université Paris-Est, École des Ponts ParisTech
Paris, France

Abstract

Deep neural networks with skip-connections, such as ResNet, show excellent performance in various image classification benchmarks. It is though observed that the initial motivation behind them - training deeper networks - does not actually hold true, and the benefits come from increased capacity, rather than from depth. Motivated by this, and inspired from ResNet, we propose a simple Dirac weight parameterization, which allows us to train very deep plain networks without skip-connections, and achieve nearly the same performance. This parameterization has a minor computational cost at training time and no cost at all at inference. We're able to achieve 95.5% accuracy on CIFAR-10 with 34-layer deep plain network, surpassing 1001-layer deep ResNet, and approaching Wide ResNet. Our parameterization also mostly eliminates the need of careful initialization in residual and non-residual networks. The code and models for our experiments are available at https://github.com/szagoruyko/diracnets

ResNext

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

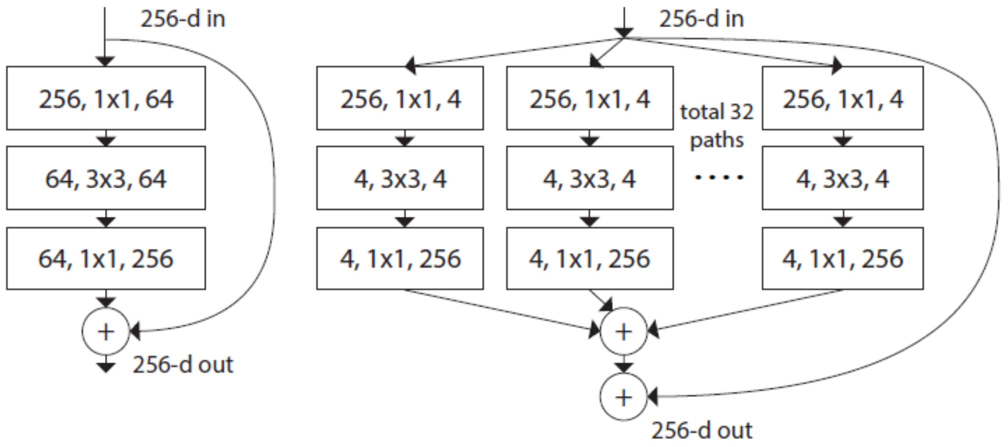


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

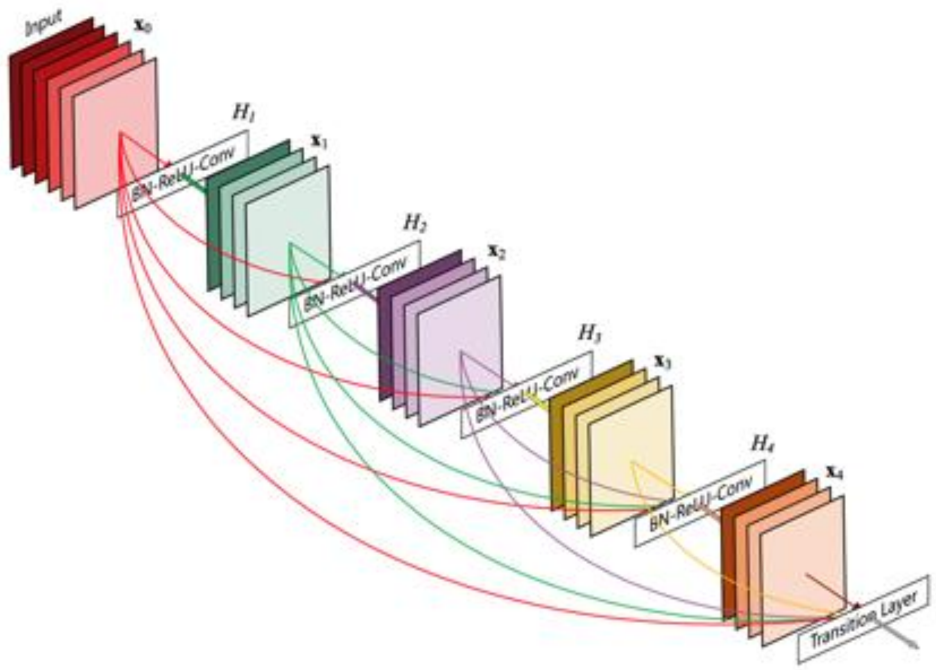
	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-200 [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

DenseNet

Densely Connected Convolutional Networks

Gao Huang* Cornell University gh349@cornell.edu
Zhuang Liu* Tsinghua University liuzhuang13@mails.tsinghua.edu.cn
Laurens van der Maaten Facebook AI Research lvdmaaten@fb.com
Kilian Q. Weinberger Cornell University kqw4@cornell.edu

2016;2018



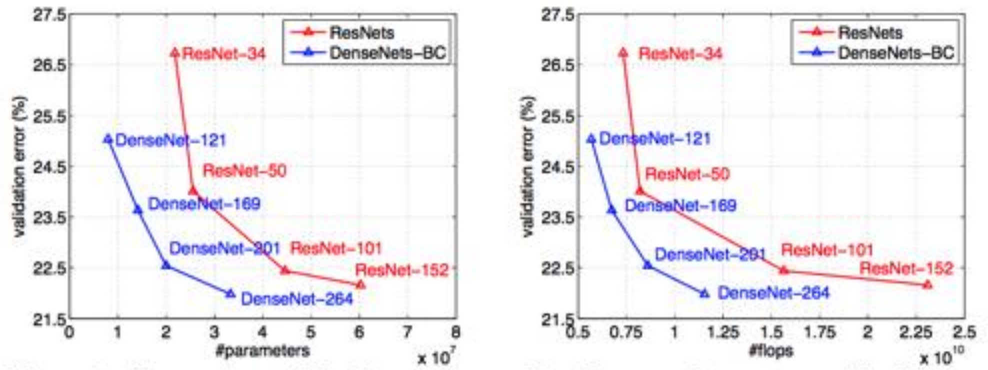
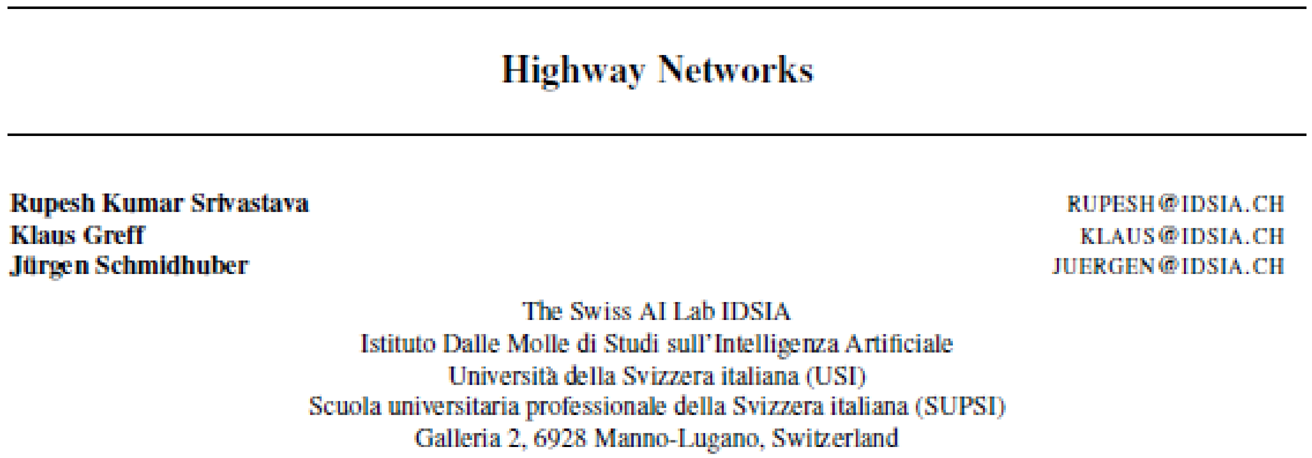


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

Highway networks

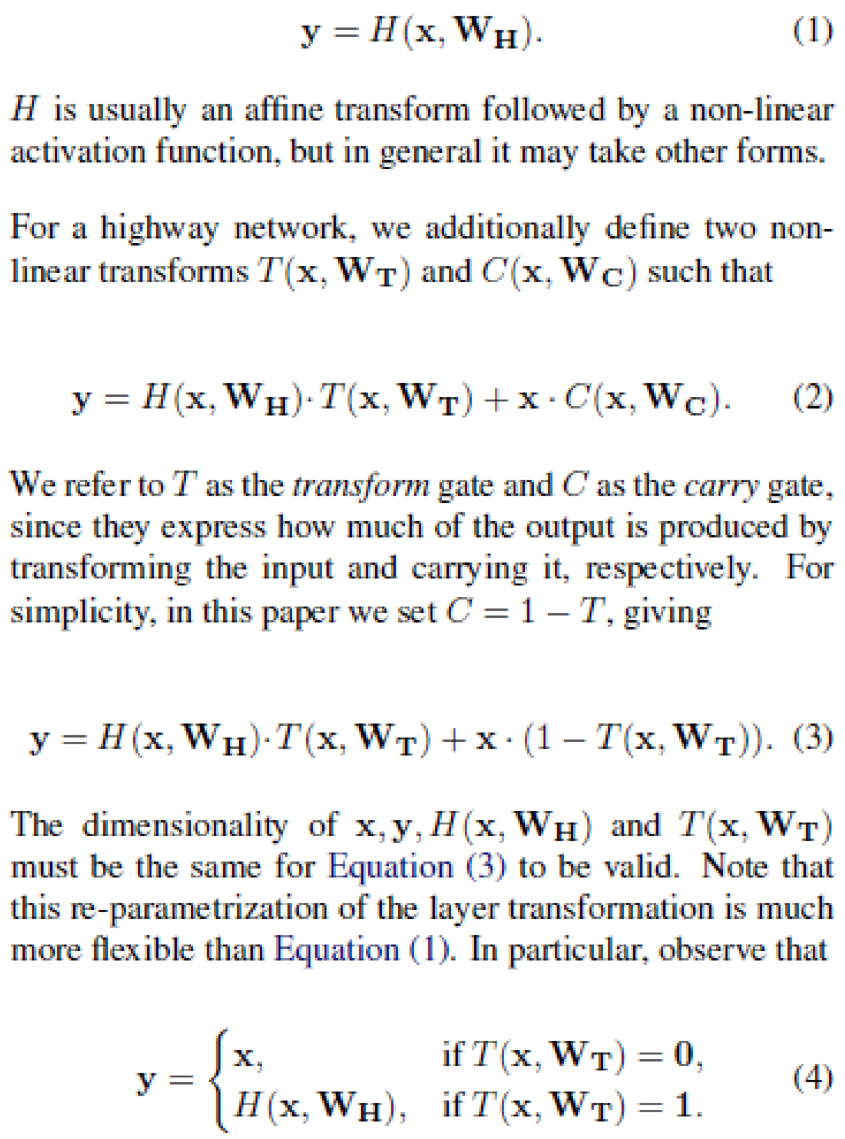
- This is a regular MLP with gated units.



Highway Networks

Rupesh Kumar Srivastava RUPESH@IDSIA.CH
Klaus Greff KLAUS@IDSIA.CH
Jürgen Schmidhuber JUERGEN@IDSIA.CH

The Swiss AI Lab IDSIA
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale
Università della Svizzera italiana (USI)
Scuola universitaria professionale della Svizzera italiana (SUPSI)
Galleria 2, 6928 Manno-Lugano, Switzerland



$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}). \tag{1}$$

H is usually an affine transform followed by a non-linear activation function, but in general it may take other forms.

For a highway network, we additionally define two non-linear transforms $T(\mathbf{x}, \mathbf{W_T})$ and $C(\mathbf{x}, \mathbf{W_C})$ such that

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}). \tag{2}$$

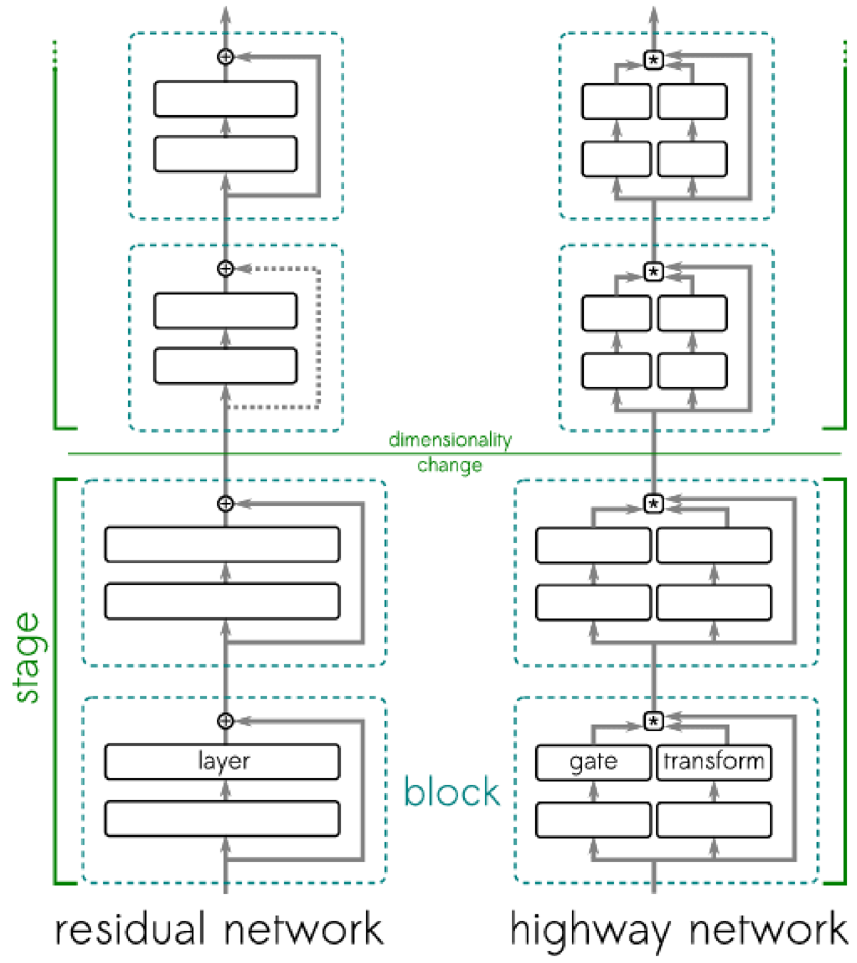
We refer to T as the *transform* gate and C as the *carry* gate, since they express how much of the output is produced by transforming the input and carrying it, respectively. For simplicity, in this paper we set $C = 1 - T$, giving

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})). \tag{3}$$

The dimensionality of $\mathbf{x}, \mathbf{y}, H(\mathbf{x}, \mathbf{W_H})$ and $T(\mathbf{x}, \mathbf{W_T})$ must be the same for Equation (3) to be valid. Note that this re-parametrization of the layer transformation is much more flexible than Equation (1). In particular, observe that

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases} \tag{4}$$

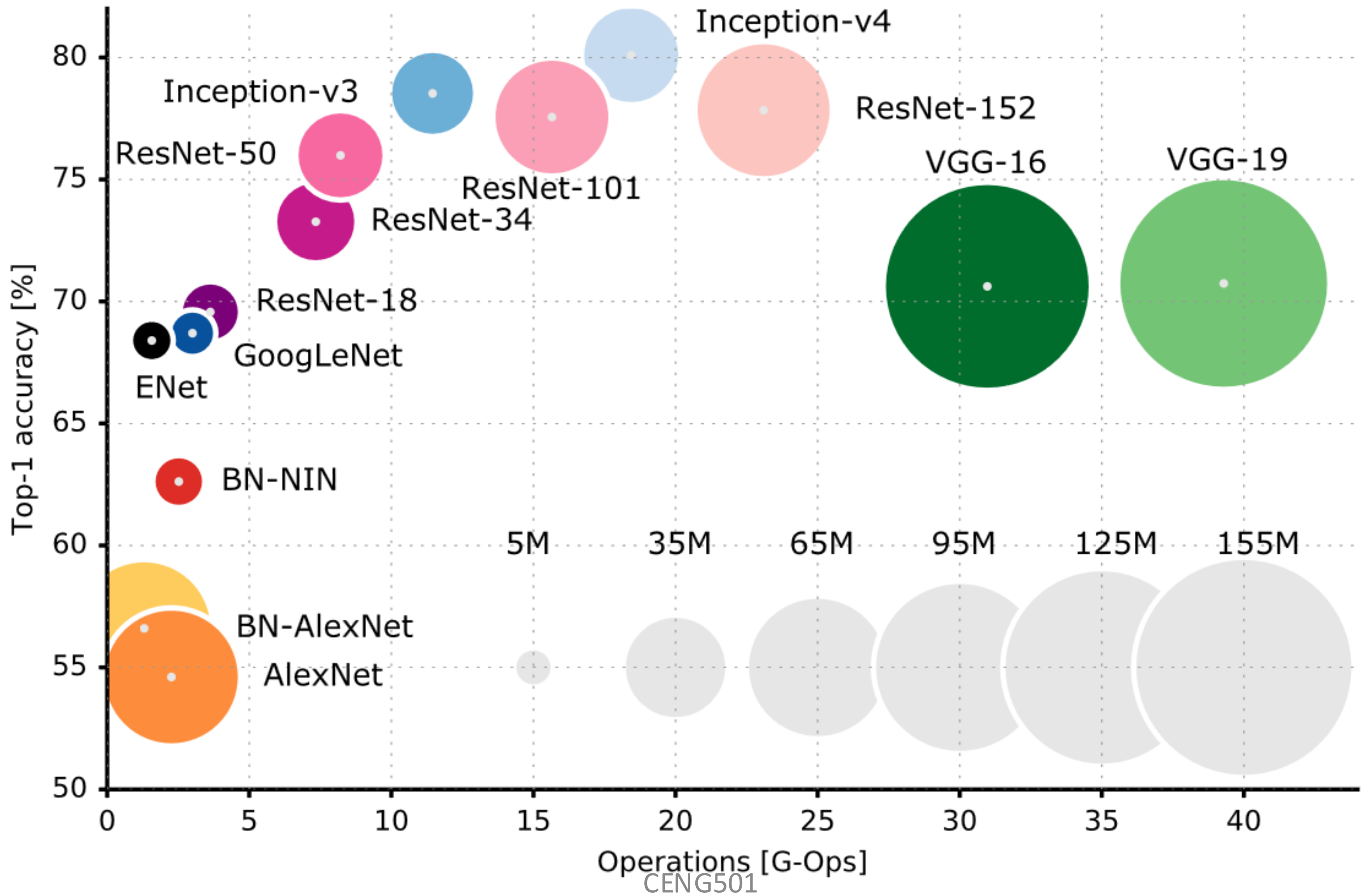
Highway Networks



https://www.researchgate.net/publication/311842587_Highway_and_Residual_Networks_learn_Unrolled_Iterative_Estimation

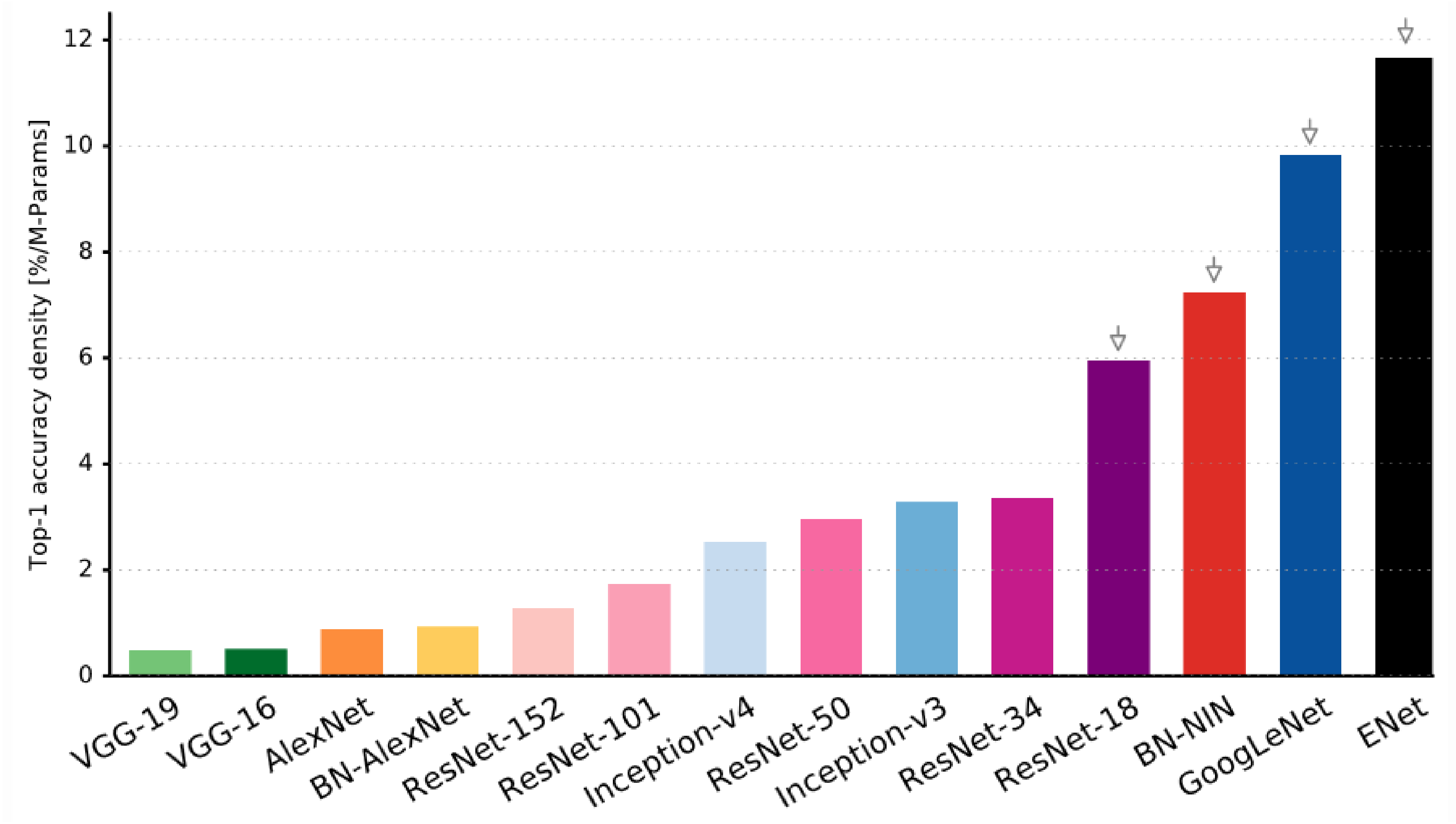
Comparison:

https://arxiv.org/pdf/1605.07678.pdf

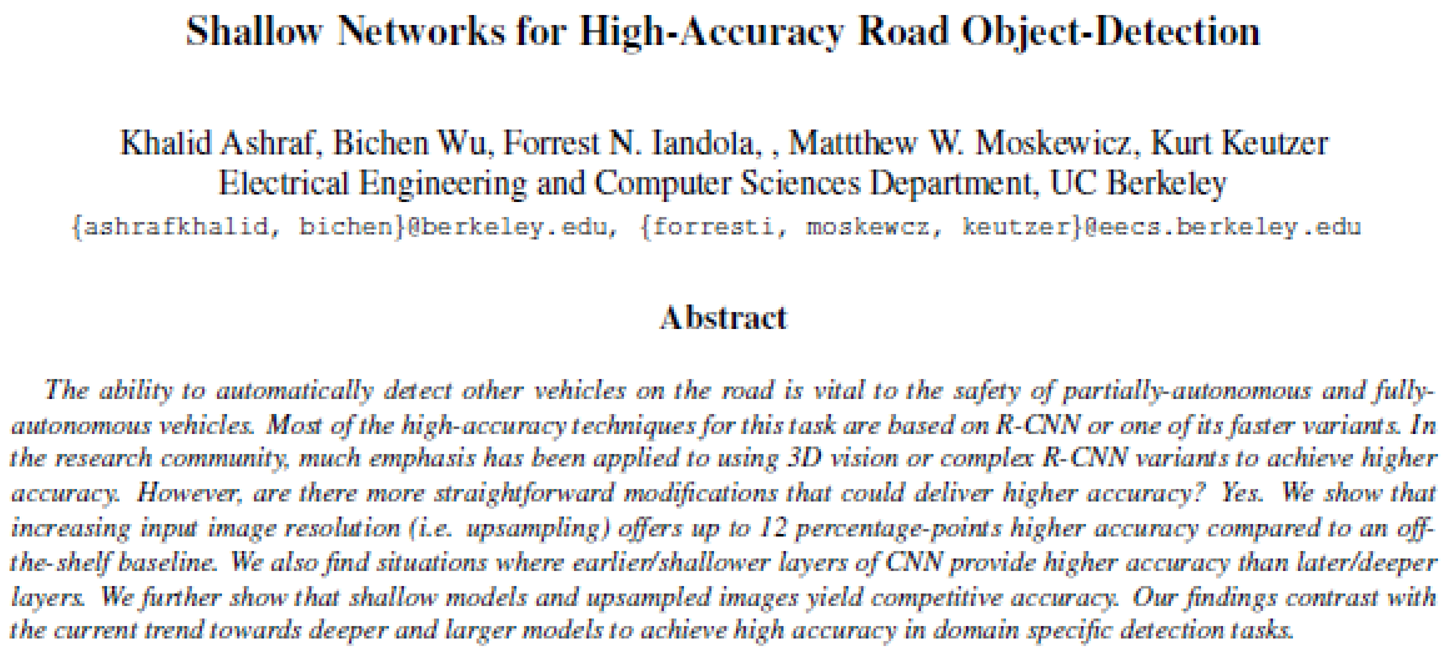


Comparison:

https://arxiv.org/pdf/1605.07678.pdf



Going deep may not be the only answer



Shallow Networks for High-Accuracy Road Object-Detection

Khalid Ashraf, Bichen Wu, Forrest N. Iandola, , Mattthew W. Moskewicz, Kurt Keutzer
Electrical Engineering and Computer Sciences Department, UC Berkeley
{ashrafkhalid, bichen}@berkeley.edu, {forresti, moskewcz, keutzer}@eecs.berkeley.edu

Abstract

The ability to automatically detect other vehicles on the road is vital to the safety of partially-autonomous and fully-autonomous vehicles. Most of the high-accuracy techniques for this task are based on R-CNN or one of its faster variants. In the research community, much emphasis has been applied to using 3D vision or complex R-CNN variants to achieve higher accuracy. However, are there more straightforward modifications that could deliver higher accuracy? Yes. We show that increasing input image resolution (i.e. upsampling) offers up to 12 percentage-points higher accuracy compared to an off-the-shelf baseline. We also find situations where earlier/shallower layers of CNN provide higher accuracy than later/deeper layers. We further show that shallow models and upsampled images yield competitive accuracy. Our findings contrast with the current trend towards deeper and larger models to achieve high accuracy in domain specific detection tasks.

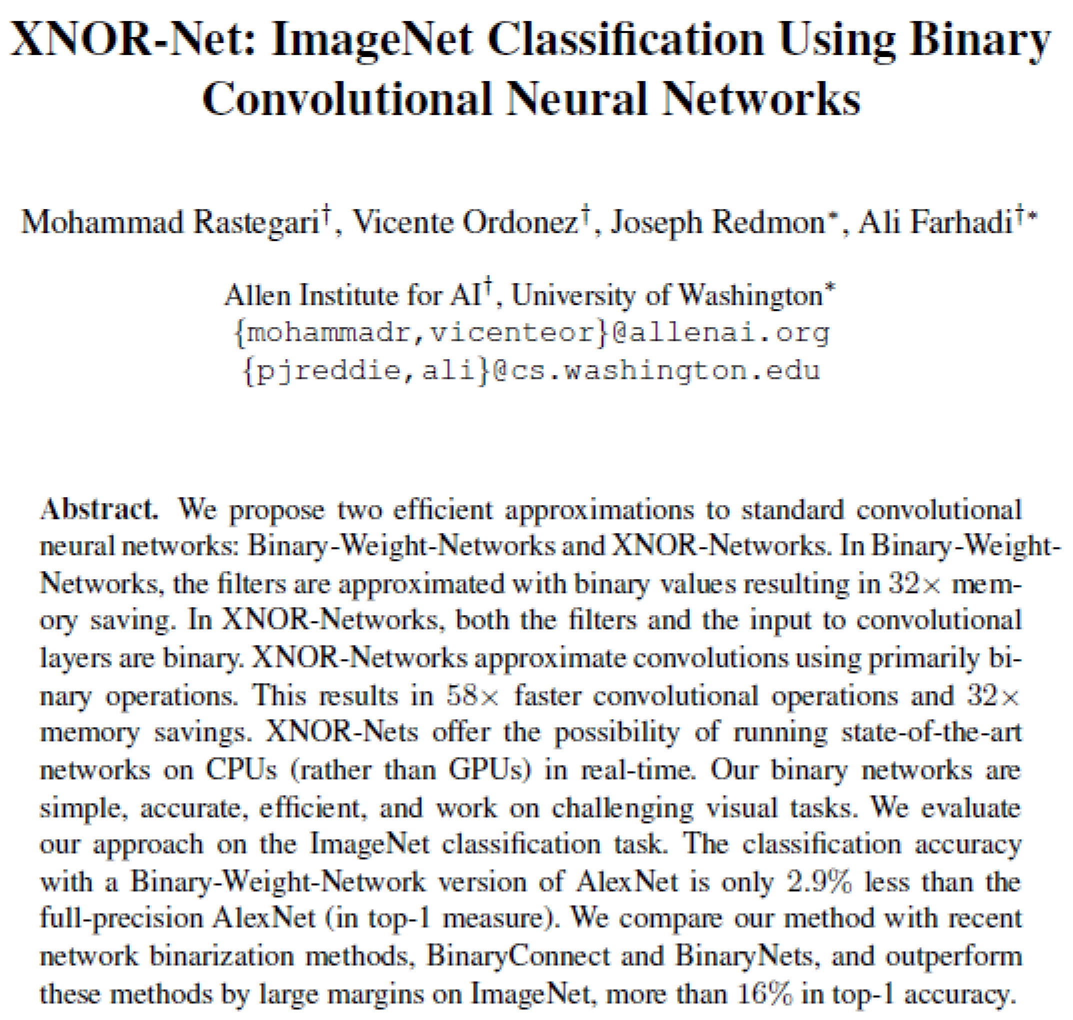
Recent work

“Towards Principled Design of Deep Convolutional Networks: Introducing SimpNet”

- Major winning Convolutional Neural Networks (CNNs), such as VGGNet, ResNet, DenseNet, \etc, include tens to hundreds of millions of parameters, which impose considerable computation and memory overheads. This limits their practical usage in training and optimizing for real-world applications. On the contrary, light-weight architectures, such as SqueezeNet, are being proposed to address this issue. However, they mainly suffer from low accuracy, as they have compromised between the processing power and efficiency. These inefficiencies mostly stem from following an ad-hoc designing procedure. In this work, we discuss and propose several crucial design principles for an efficient architecture design and elaborate intuitions concerning different aspects of the design procedure. Furthermore, we introduce a new layer called {\it SAF-pooling} to improve the generalization power of the network while keeping it simple by choosing best features. Based on such principles, we propose a simple architecture called {\it SimpNet}. We empirically show that SimpNet provides a good trade-off between the computation/memory efficiency and the accuracy solely based on these primitive but crucial principles. SimpNet outperforms the deeper and more complex architectures such as VGGNet, ResNet, WideResidualNet \etc, on several well-known benchmarks, while having 2 to 25 times fewer number of parameters and operations. We obtain state-of-the-art results (in terms of a balance between the accuracy and the number of involved parameters) on standard datasets, such as CIFAR10, CIFAR100, MNIST and SVHN. The implementations are available at \href{url}{this https URL}.

https://arxiv.org/abs/1802.06205

Binary networks



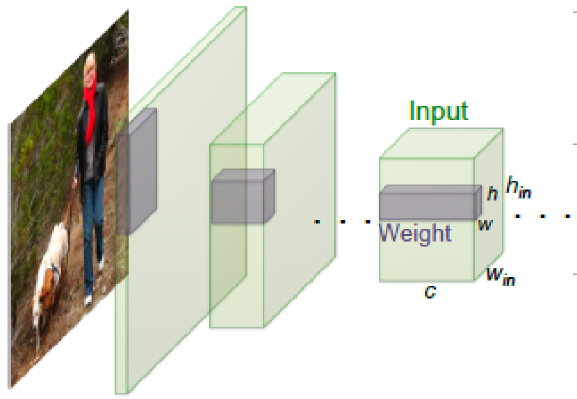
XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks

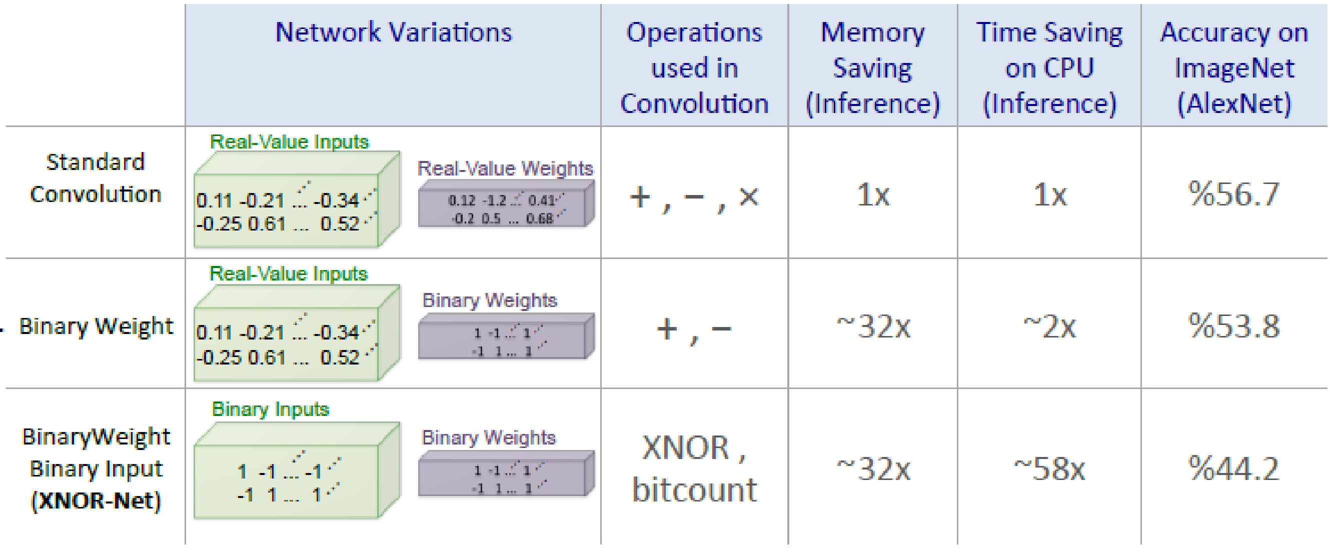
Mohammad Rastegari[†], Vicente Ordonez[†], Joseph Redmon[*], Ali Farhadi[†*]

Allen Institute for AI[†], University of Washington[*]
{mohammadr,vicenteor}@allenai.org
{pjreddie,ali}@cs.washington.edu

Abstract. We propose two efficient approximations to standard convolutional neural networks: Binary-Weight-Networks and XNOR-Networks. In Binary-Weight-Networks, the filters are approximated with binary values resulting in $32\times$ memory saving. In XNOR-Networks, both the filters and the input to convolutional layers are binary. XNOR-Networks approximate convolutions using primarily binary operations. This results in $58\times$ faster convolutional operations and $32\times$ memory savings. XNOR-Nets offer the possibility of running state-of-the-art networks on CPUs (rather than GPUs) in real-time. Our binary networks are simple, accurate, efficient, and work on challenging visual tasks. We evaluate our approach on the ImageNet classification task. The classification accuracy with a Binary-Weight-Network version of AlexNet is only 2.9% less than the full-precision AlexNet (in top-1 measure). We compare our method with recent network binarization methods, BinaryConnect and BinaryNets, and outperform these methods by large margins on ImageNet, more than 16% in top-1 accuracy.

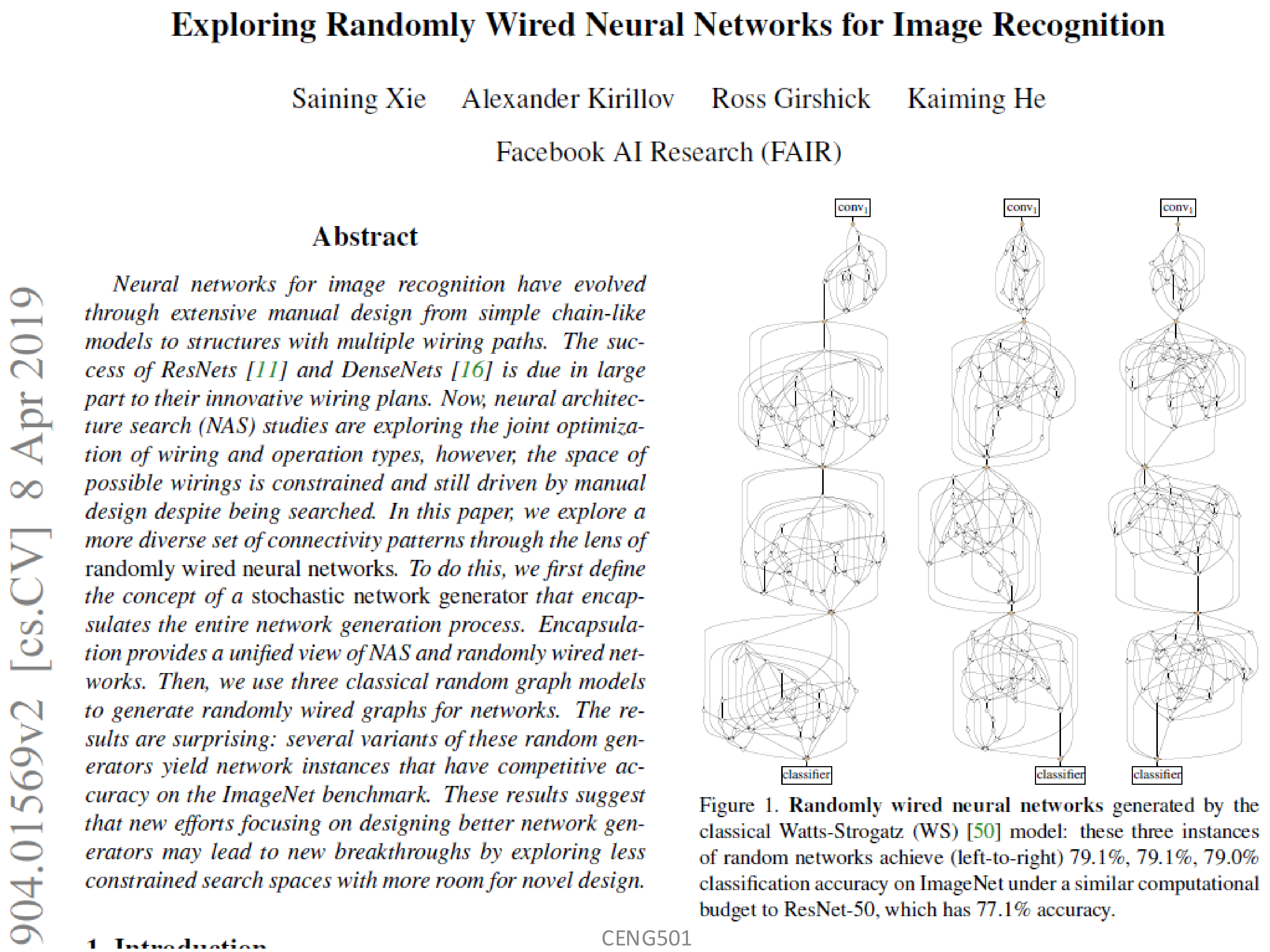
Binary networks





	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.



Exploring Randomly Wired Neural Networks for Image Recognition

Saining Xie   Alexander Kirillov   Ross Girshick   Kaiming He

Facebook AI Research (FAIR)

Abstract

Neural networks for image recognition have evolved through extensive manual design from simple chain-like models to structures with multiple wiring paths. The success of ResNets [11] and DenseNets [16] is due in large part to their innovative wiring plans. Now, neural architecture search (NAS) studies are exploring the joint optimization of wiring and operation types, however, the space of possible wirings is constrained and still driven by manual design despite being searched. In this paper, we explore a more diverse set of connectivity patterns through the lens of randomly wired neural networks. *To do this, we first define the concept of a* stochastic network generator *that encapsulates the entire network generation process. Encapsulation provides a unified view of NAS and randomly wired networks. Then, we use three classical random graph models to generate randomly wired graphs for networks. The results are surprising: several variants of these random generators yield network instances that have competitive accuracy on the ImageNet benchmark. These results suggest that new efforts focusing on designing better network generators may lead to new breakthroughs by exploring less constrained search spaces with more room for novel design.*

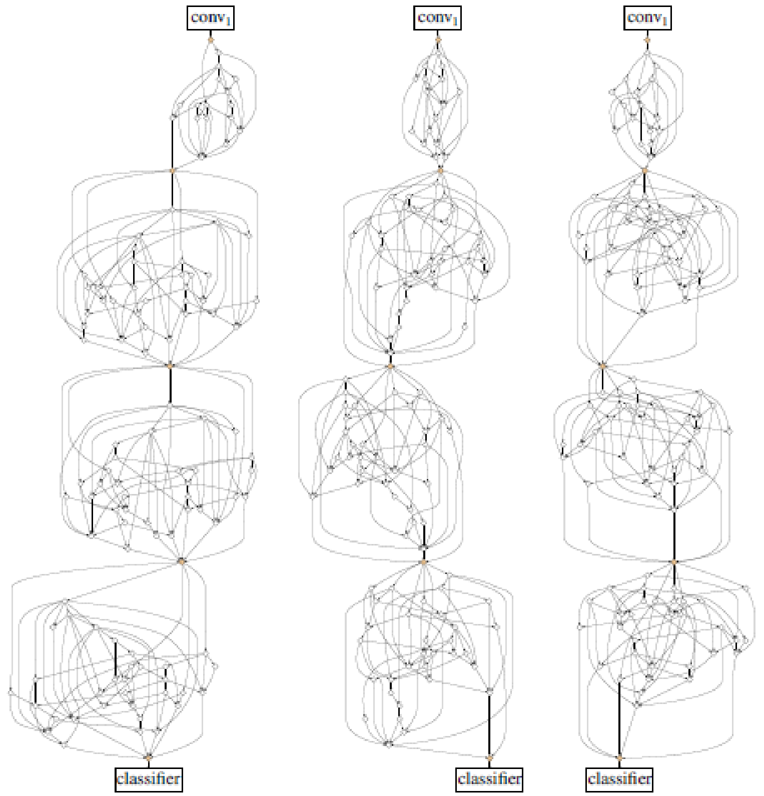


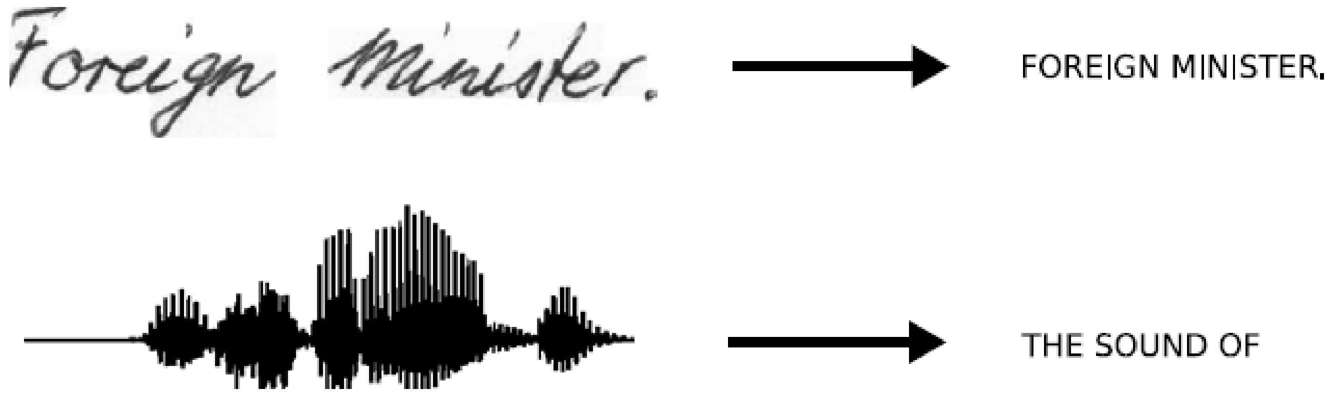
Figure 1. **Randomly wired neural networks** generated by the classical Watts-Strogatz (WS) [50] model: these three instances of random networks achieve (left-to-right) 79.1%, 79.1%, 79.0% classification accuracy on ImageNet under a similar computational budget to ResNet-50, which has 77.1% accuracy.

CNNs: summary & future directions

- Less parameters
- Allows going deeper
- High flexibility
  - In operations
  - In organization of layers
  - In the overall architecture etc.
- Future directions:
  - Understanding them better
  - Making them deeper, faster and more efficient
  - Compressing a big network into a smaller & cheaper one.
  - …

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.

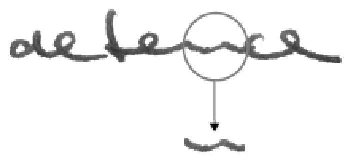
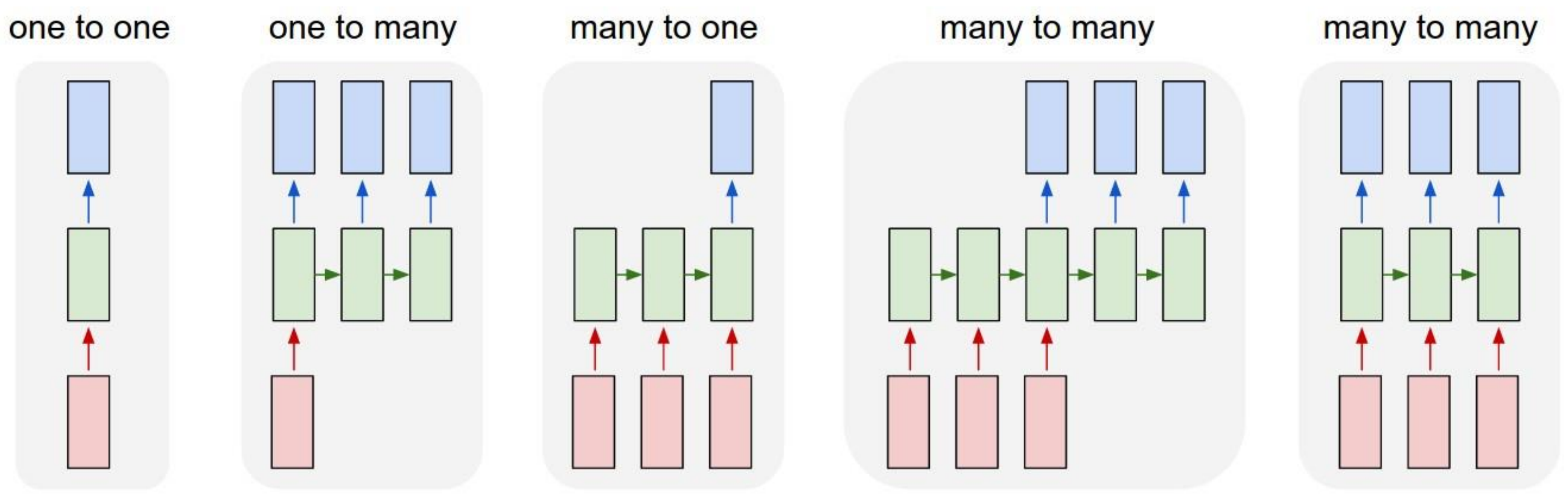


Fig. 2.3 Importance of context in segment classification. The word 'defence' is clearly legible. However the letter 'n' in isolation is ambiguous.

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

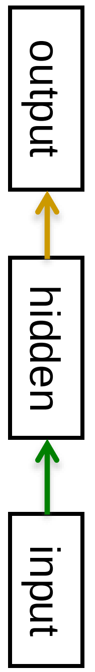
Different types of sequence learning / recognition problems



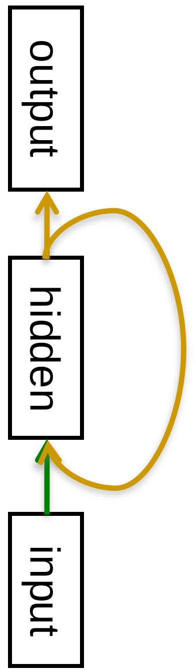
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Neural Networks

Recurrent Neural Networks (RNNs)



Feed-forward networks



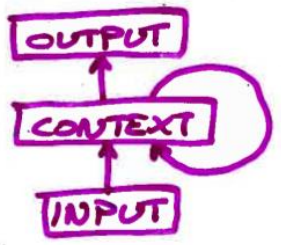
Recurrent networks

- 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

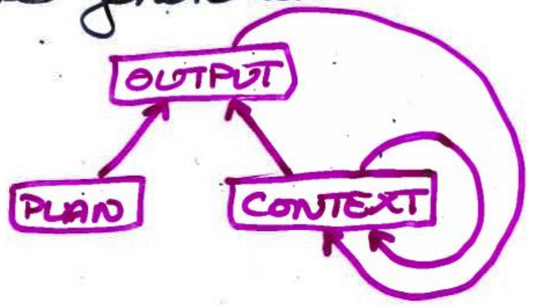
– Temporal pattern recognition



INPUT1, INPUT2, INPUT3, …, INPUTt
⇒ OUTPUTt

e.g., speech recognition
e.g., event recognition
e.g., natural language understand

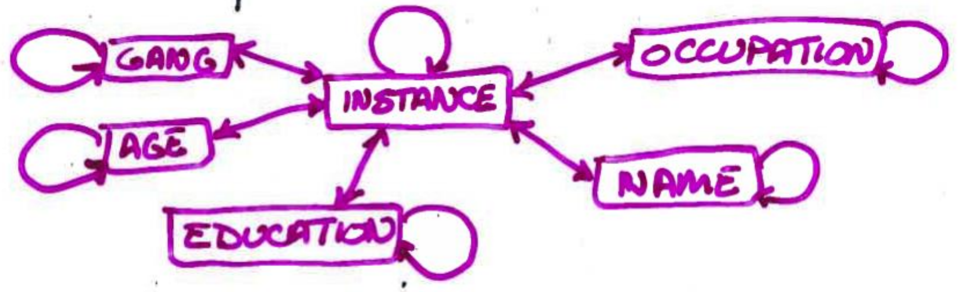
– Sequence generation



PLAN ⇒ OUTPUT1, OUTPUT2,
OUTPUT3, …, OUTPUTt

e.g., speech production
e.g., motor control
e.g., planning and acting

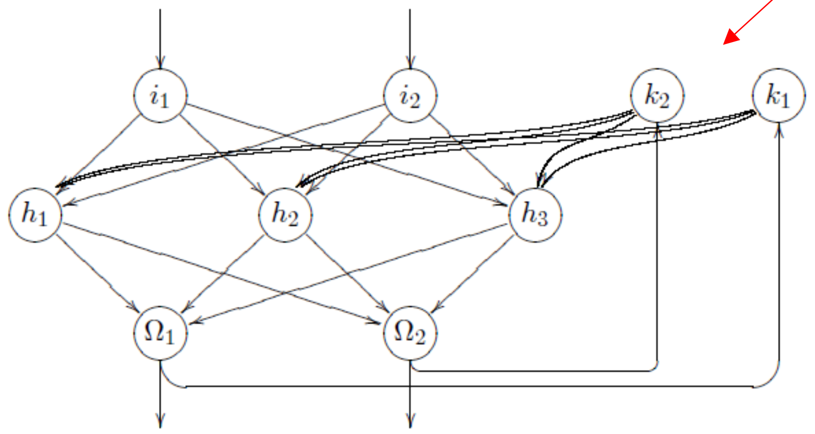
– Pattern completion / constraint satisfaction



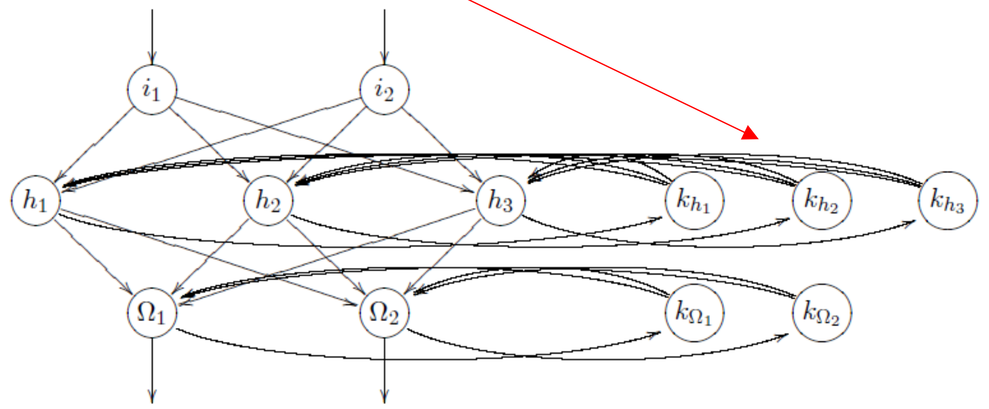
Slide: Michael Mozer

Some examples

“context” neurons



Jordan Networks

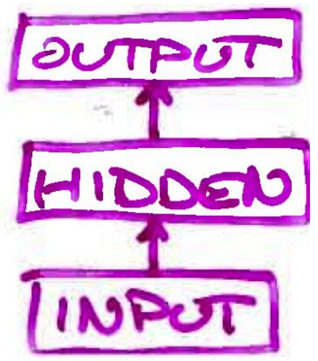


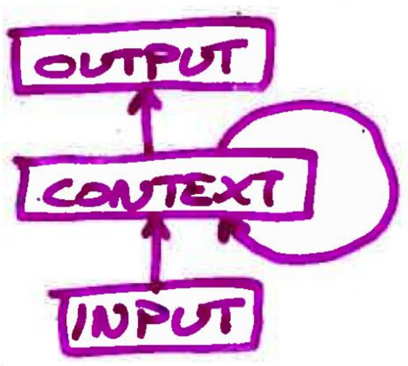
Elman Networks

Figs: David Kriesel

Challenge

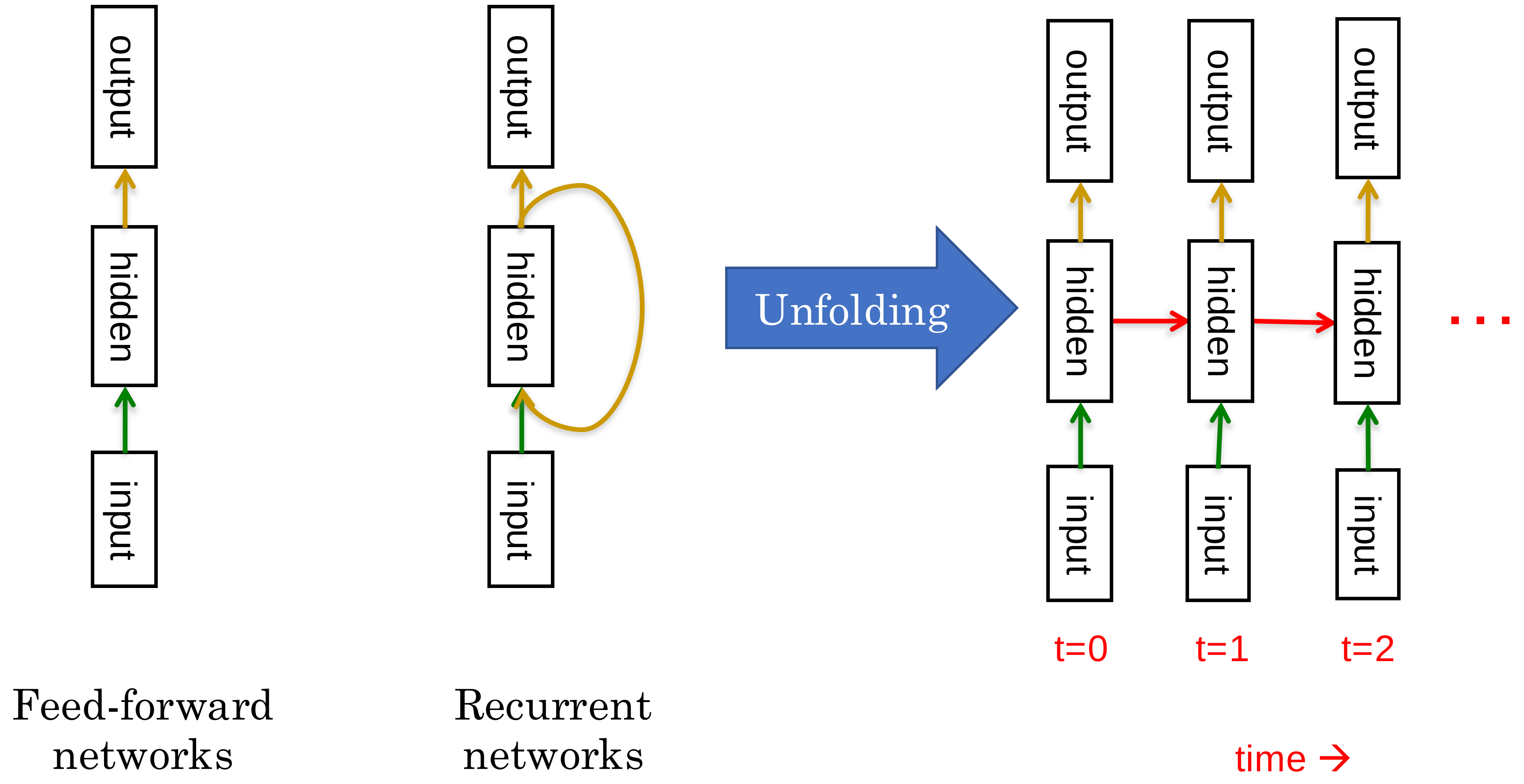
- Back propagation is designed for feedforward nets
- What would it mean to back propagate through a recurrent network?
    - error signal would have to travel back in time





Slide: Michael Mozer

Unfolding


output
hidden
input
Feed-forward networks
output
hidden
input
Recurrent networks
Unfolding
output
hidden
input
t=0
output
hidden
input
t=1
output
hidden
input
t=2
…
time →

Unfolding (another example)

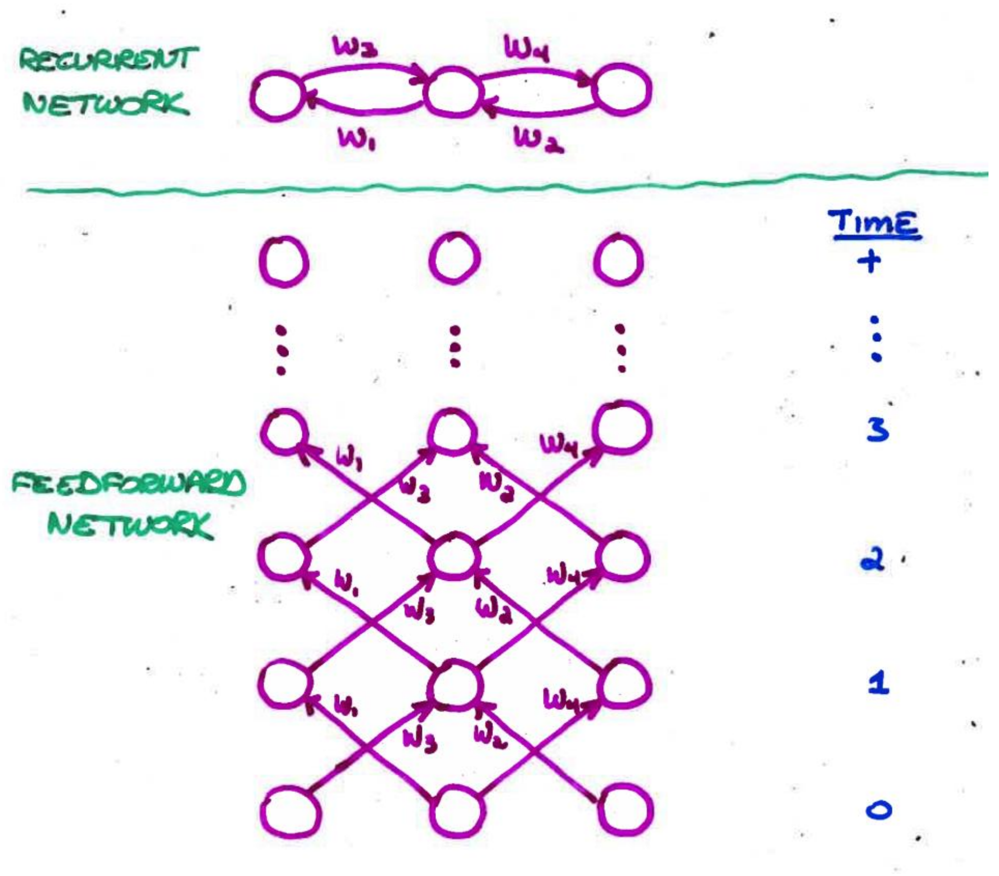
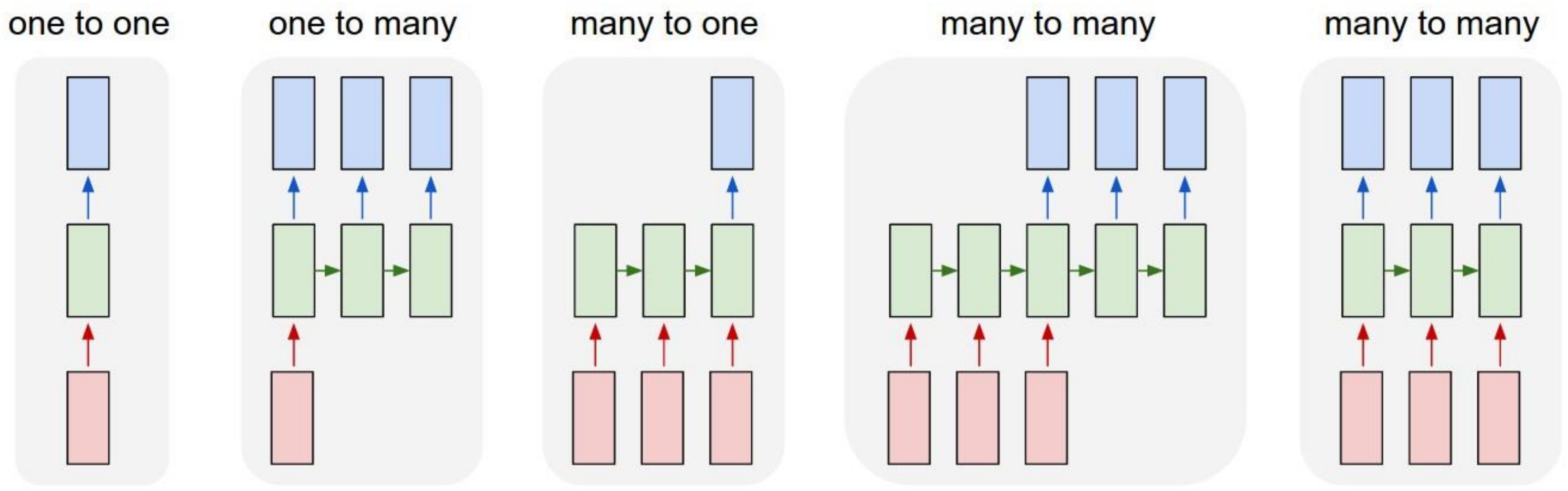
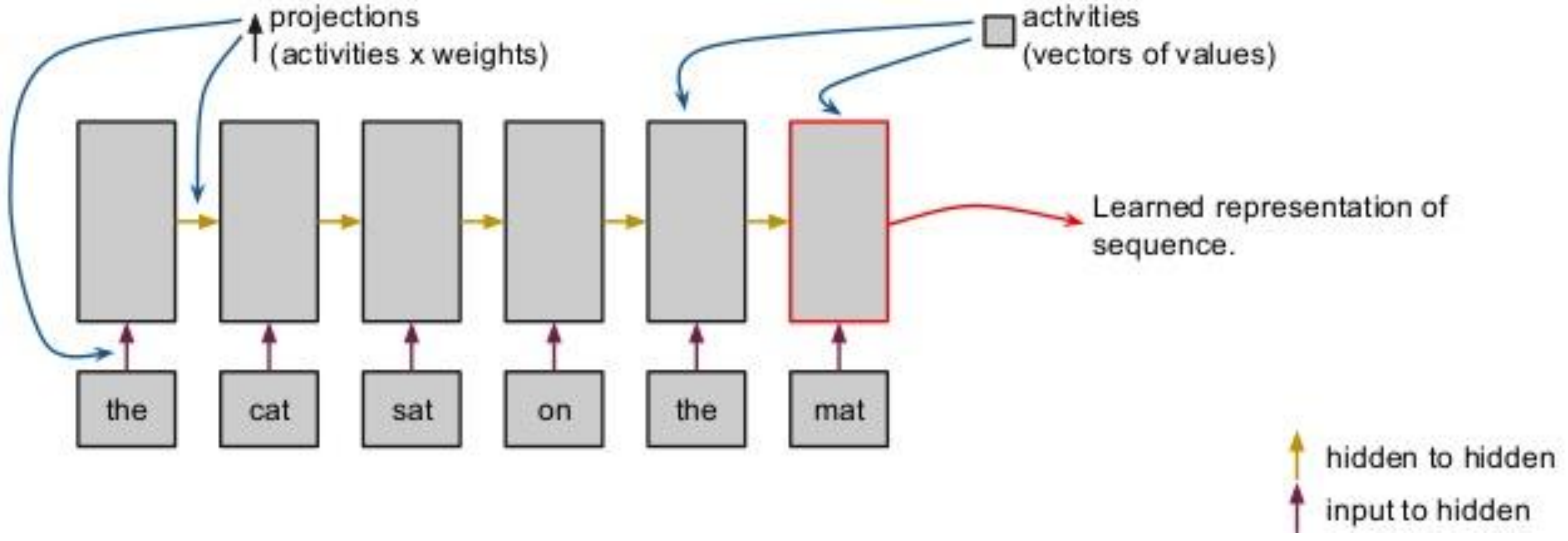


Figure: Michael Mozer



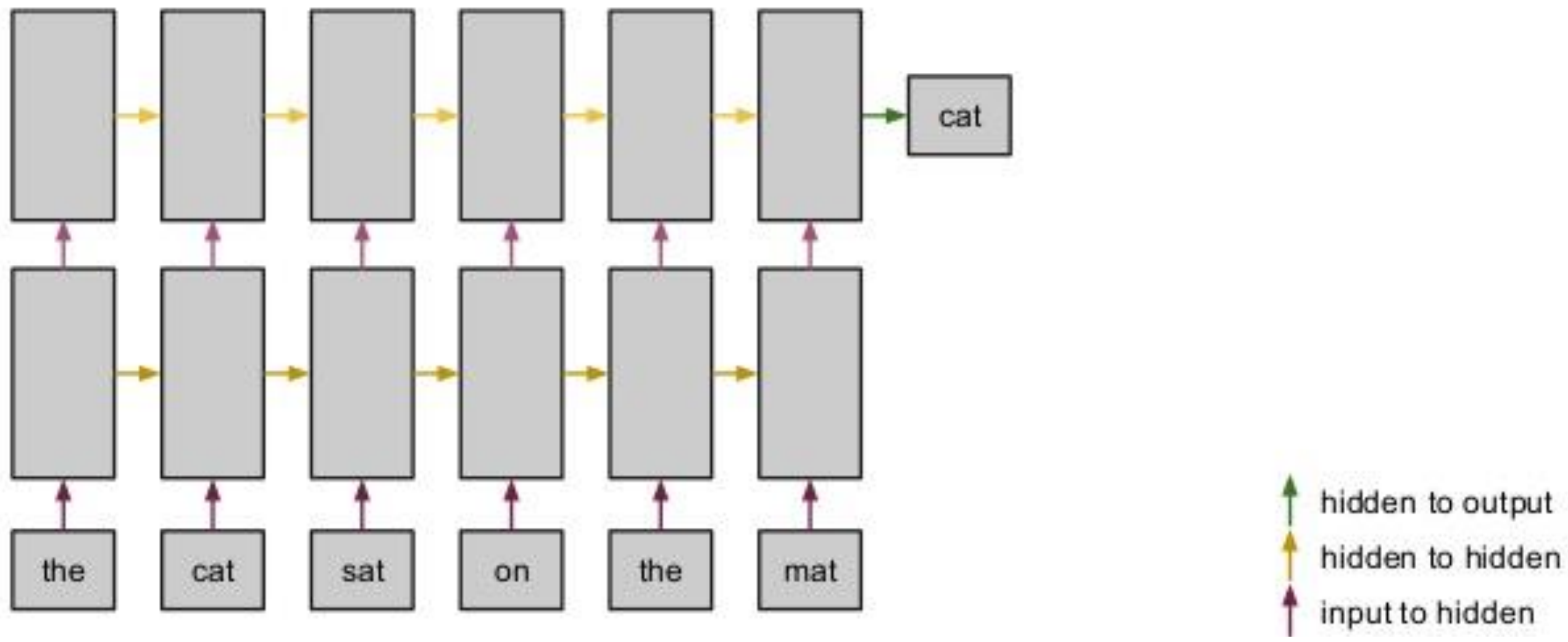
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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!

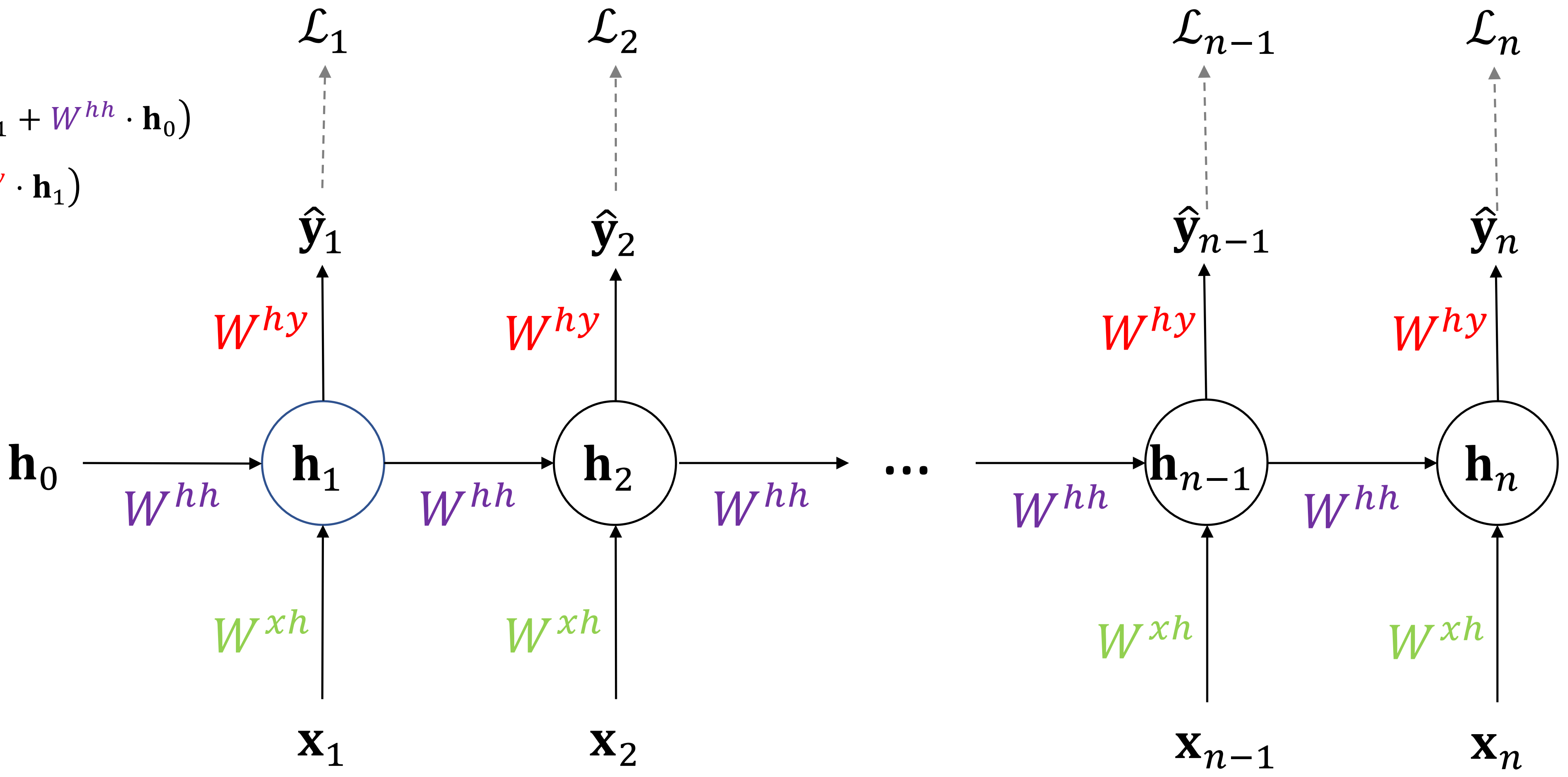
Back-propagation Through Time

Feedforward through Vanilla RNN

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

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

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



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:

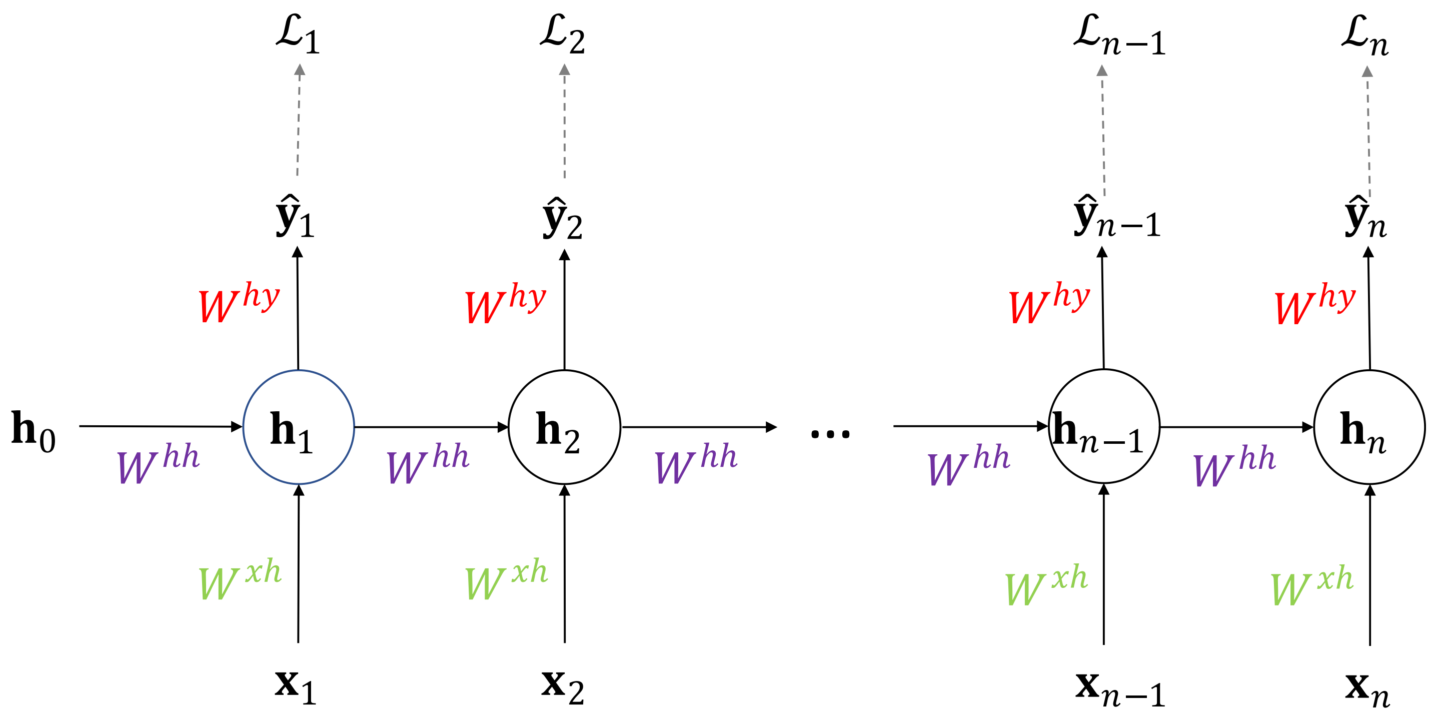
$$\mathbf{h}_t = tanh\left(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1}\right)$$

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

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

In total:

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



Backpropagation through Vanilla RNN

The Vanilla RNN Model

In general:

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

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

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

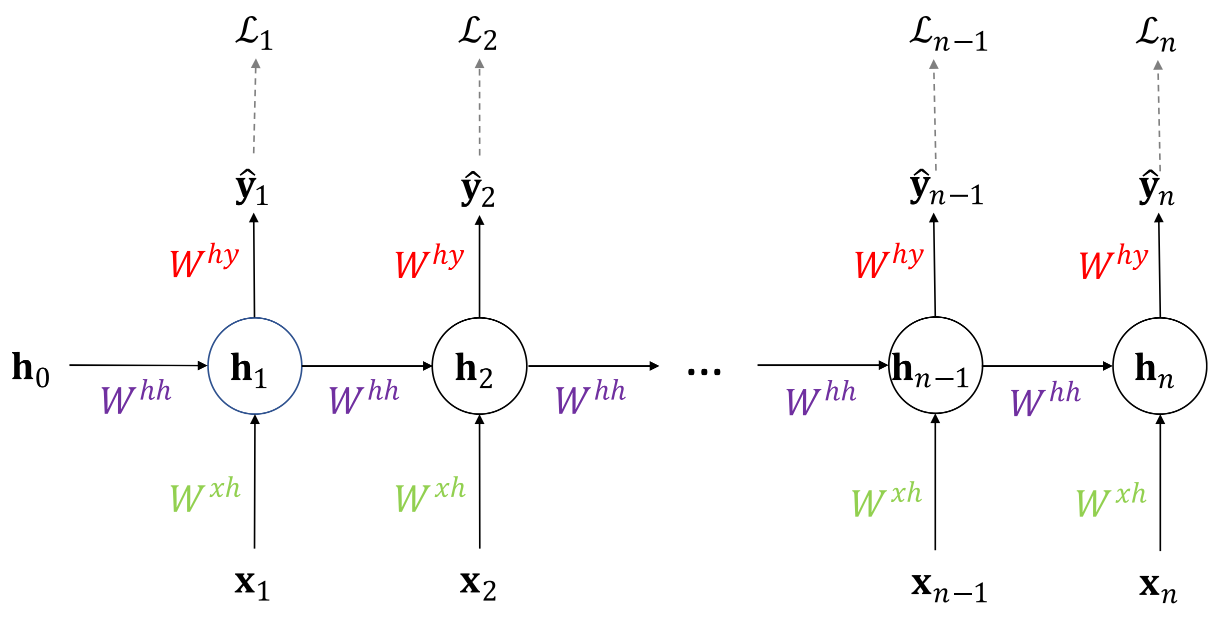
In total:

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

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

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

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



Backpropagation through Vanilla RNN

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

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

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

The Vanilla RNN Model

In general:

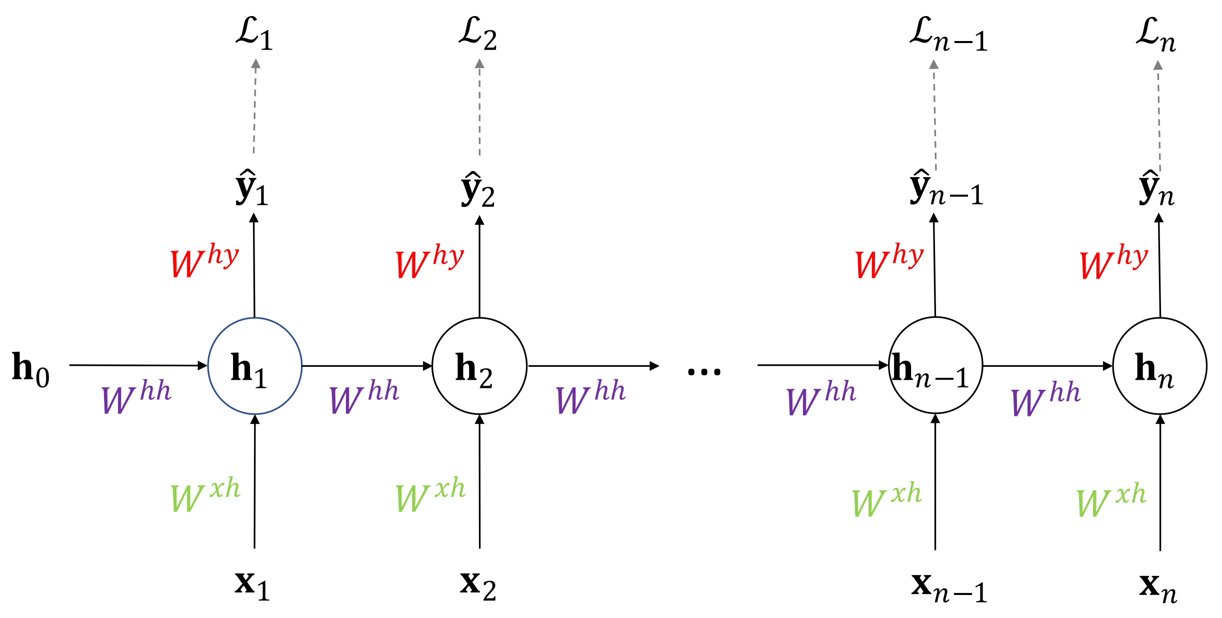
$$\mathbf{h}_t = tanh(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1})$$

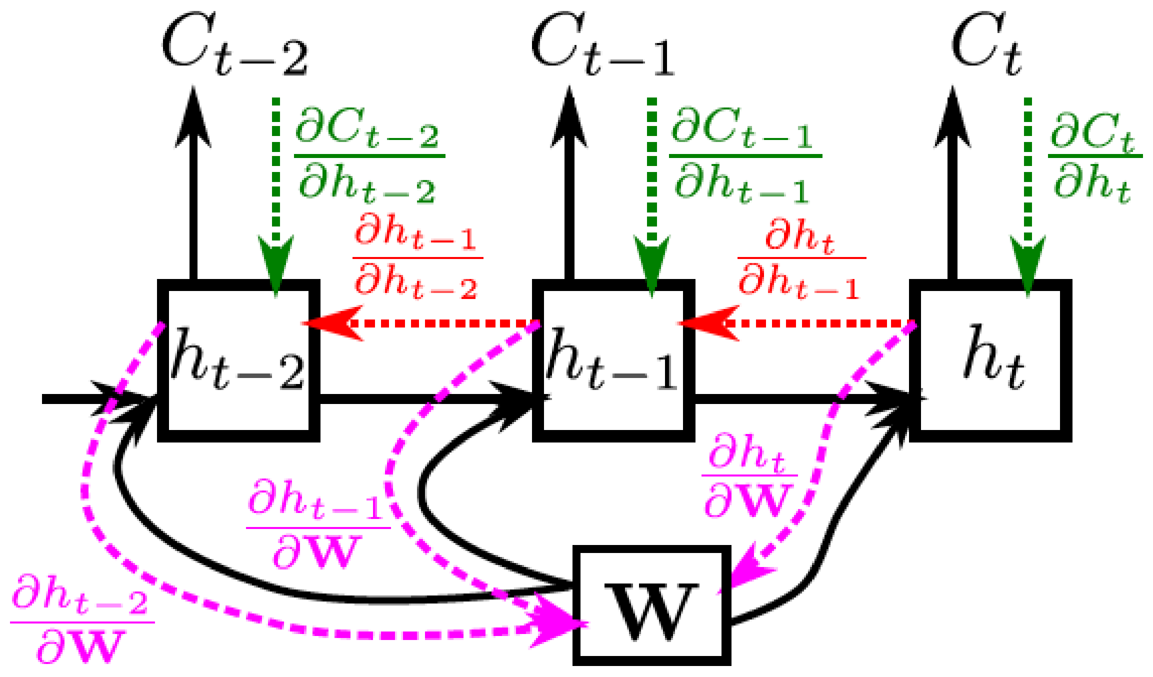
$$\hat{\mathbf{y}}_t = softmax(W^{hy} \cdot \mathbf{h}_t)$$

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

In total:

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





C_{t-2}
C_{t-1}
C_t
$\frac{\partial C_{t-2}}{\partial h_{t-2}}$
$\frac{\partial C_{t-1}}{\partial h_{t-1}}$
$\frac{\partial C_t}{\partial h_t}$
$\frac{\partial h_{t-1}}{\partial h_{t-2}}$
$\frac{\partial h_t}{\partial h_{t-1}}$
h_{t-2}
h_{t-1}
h_t
$\frac{\partial h_t}{\partial \mathbf{W}}$
$\frac{\partial h_{t-1}}{\partial \mathbf{W}}$
$\frac{\partial h_{t-2}}{\partial \mathbf{W}}$
$\mathbf{W}$

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

Backpropagation through Vanilla RNN

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

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

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

(calculated before)

The Vanilla RNN Model

In general:

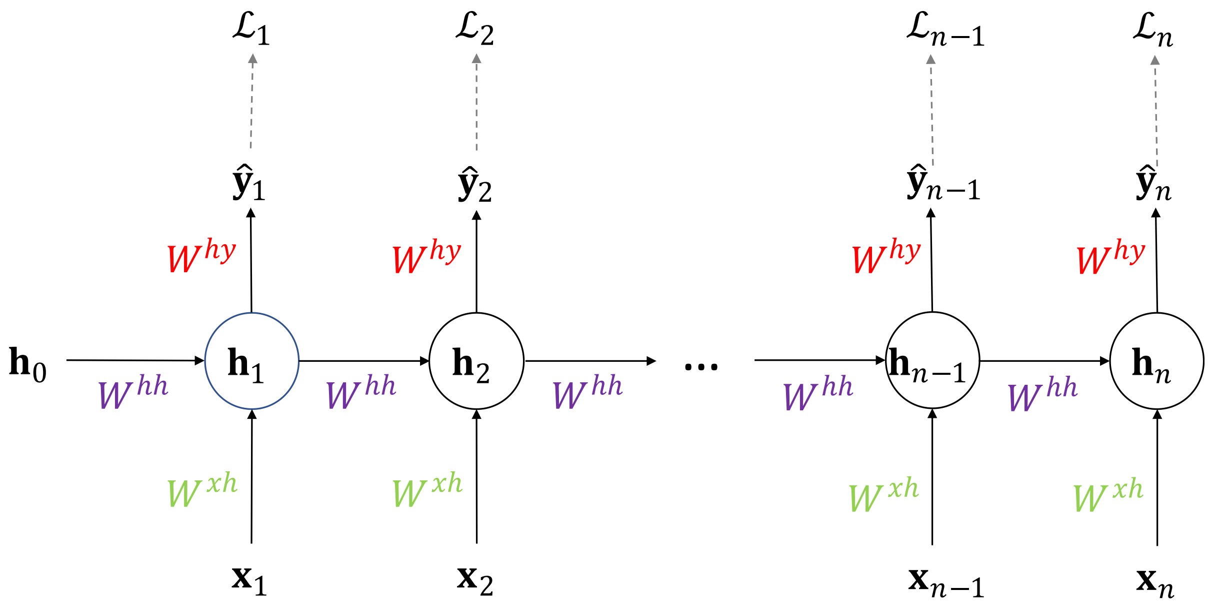
$$\mathbf{h}_t = tanh(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1})$$

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

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

In total:

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



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

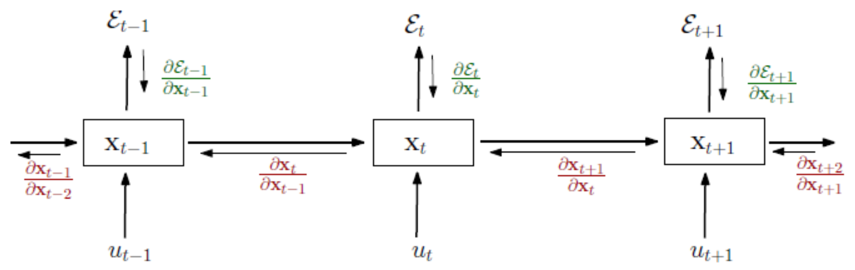
- Solution 1: Gradient clipping for exploding gradients:

Algorithm 1 Pseudo-code for norm clipping

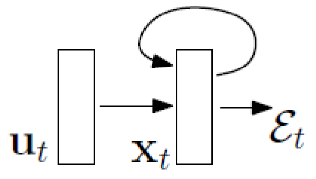
```
$\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{g}\| \geq threshold$ then
    $\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|} \hat{g}$
end if
```

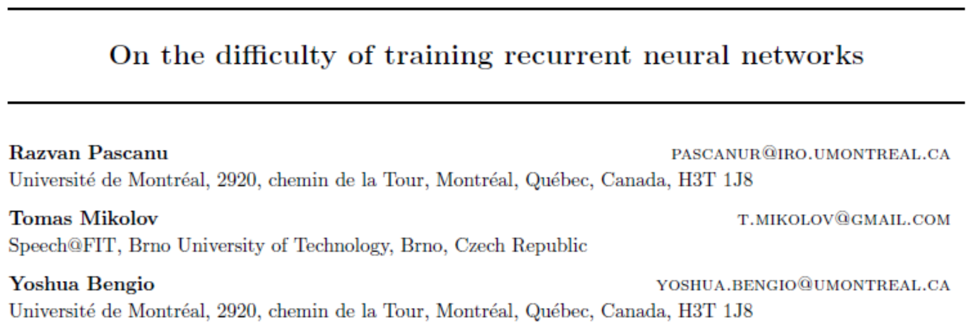
  - For vanishing gradients: Regularization term that penalizes changes in the magnitudes of back-propagated gradients

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



$\mathcal{E}_{t-1}$
$\mathcal{E}_t$
$\mathcal{E}_{t+1}$
$\frac{\partial \mathcal{E}_{t-1}}{\partial \mathbf{x}_{t-1}}$
$\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t}$
$\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{x}_{t+1}}$
$\mathbf{x}_{t-1}$
$\mathbf{x}_t$
$\mathbf{x}_{t+1}$
$\frac{\partial \mathbf{x}_{t-1}}{\partial \mathbf{x}_{t-2}}$
$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}}$
$\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$
$\frac{\partial \mathbf{x}_{t+2}}{\partial \mathbf{x}_{t+1}}$
u_{t-1}
u_t
u_{t+1}





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

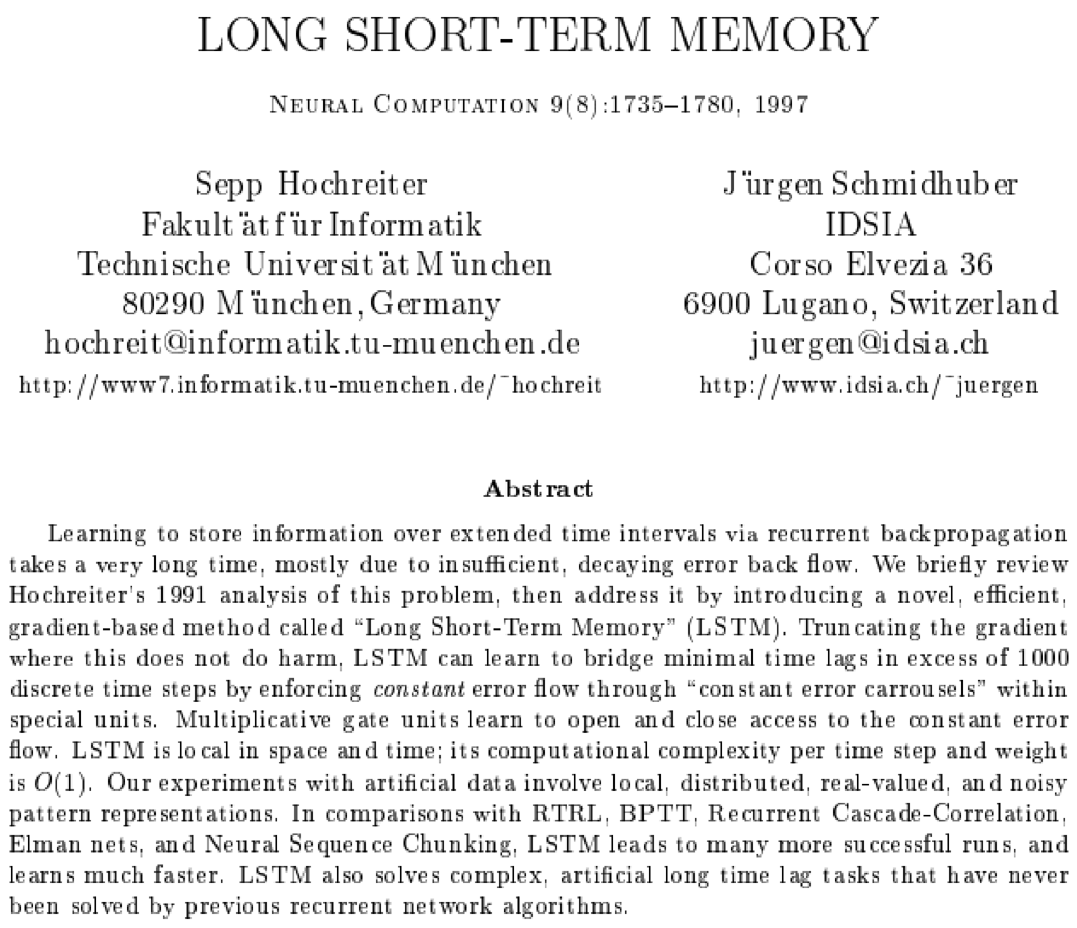
Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Université de Montréal, 2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

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

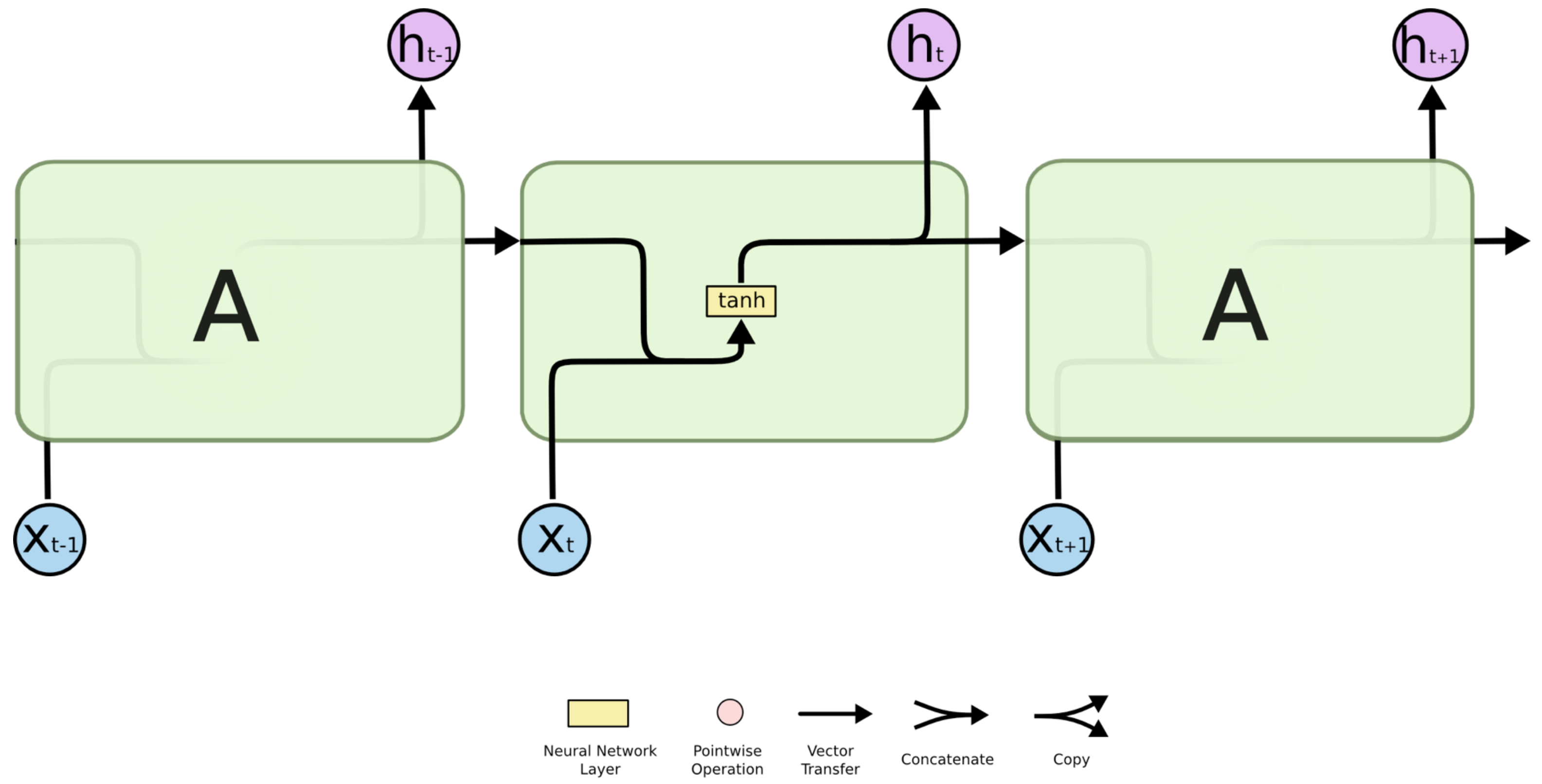


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Key Problem

- Learning long-term dependencies is hard

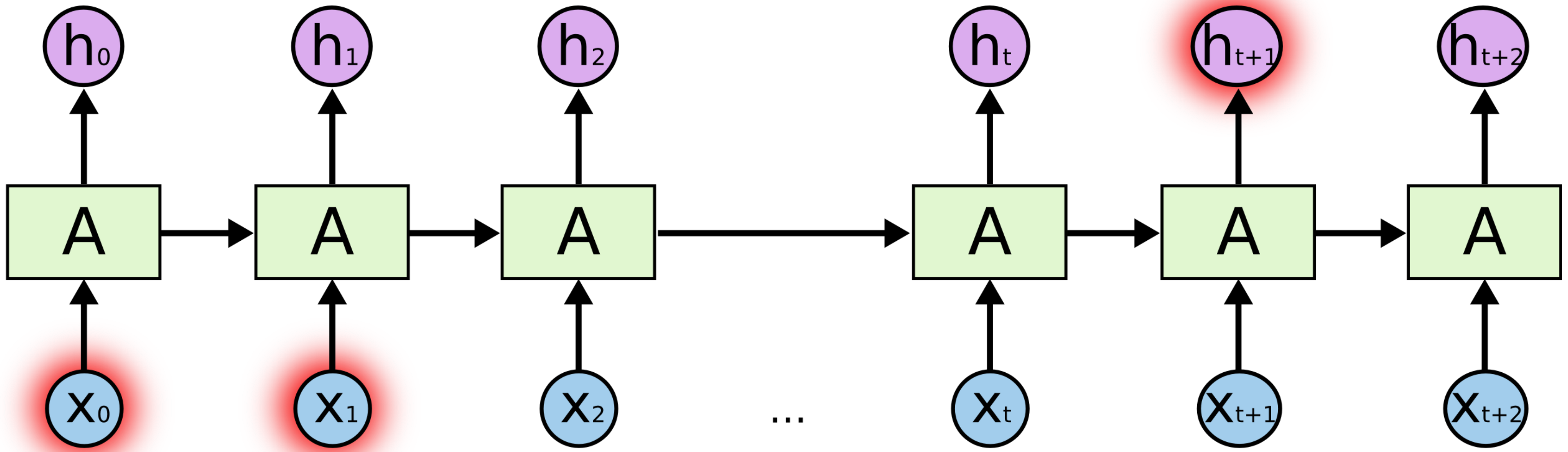


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

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?

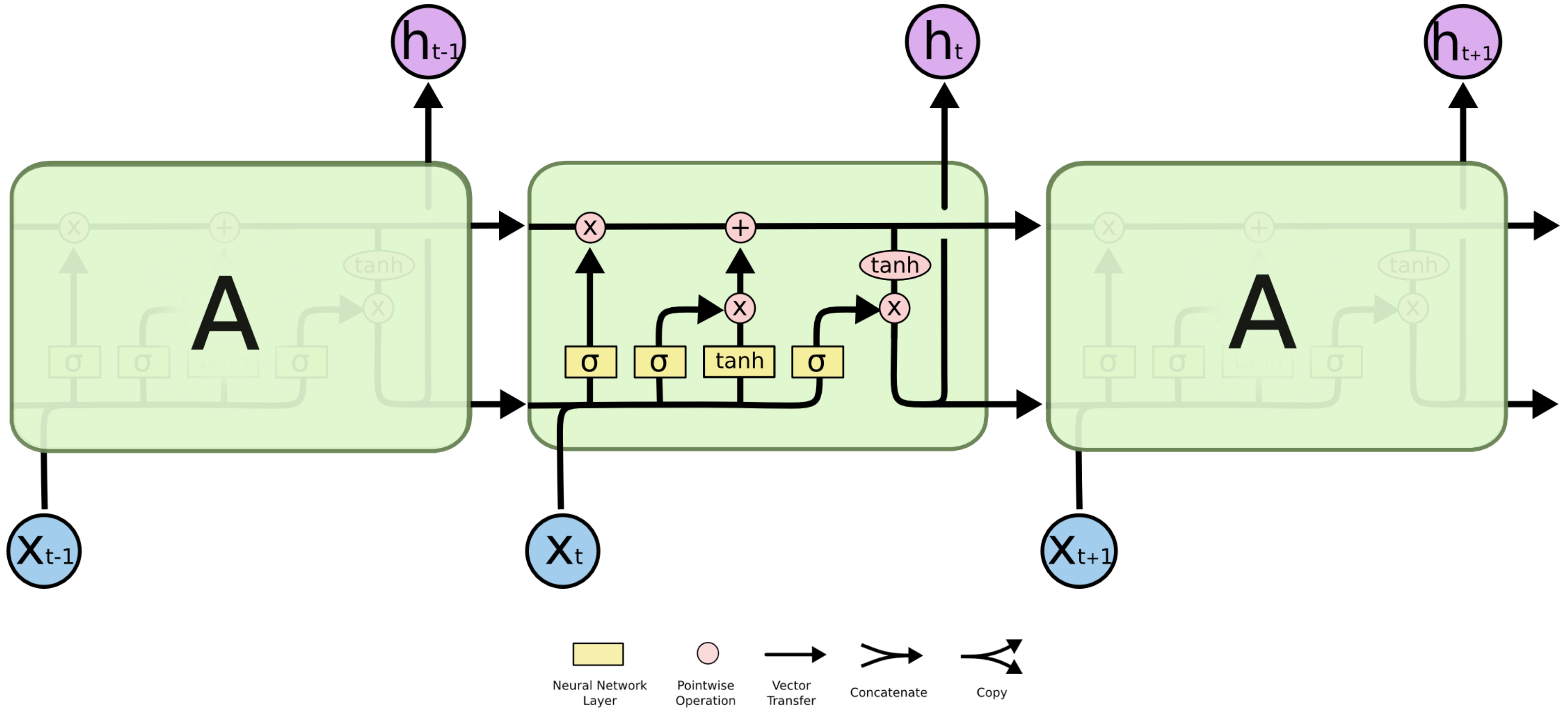


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

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

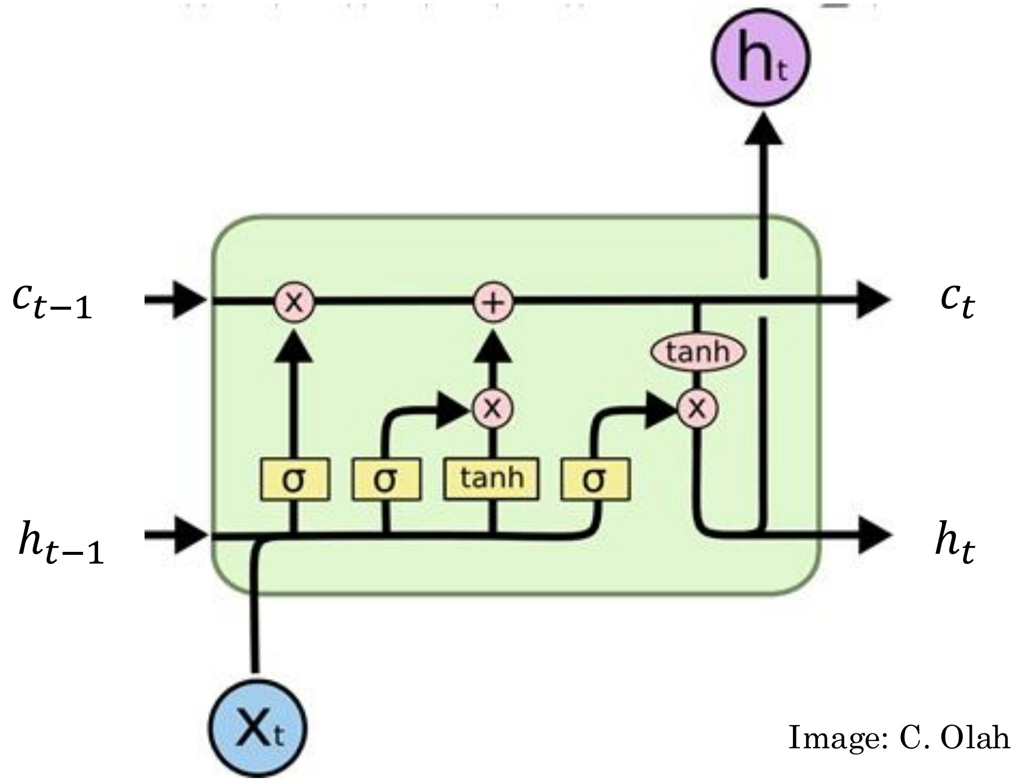


Image: C. Olah

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

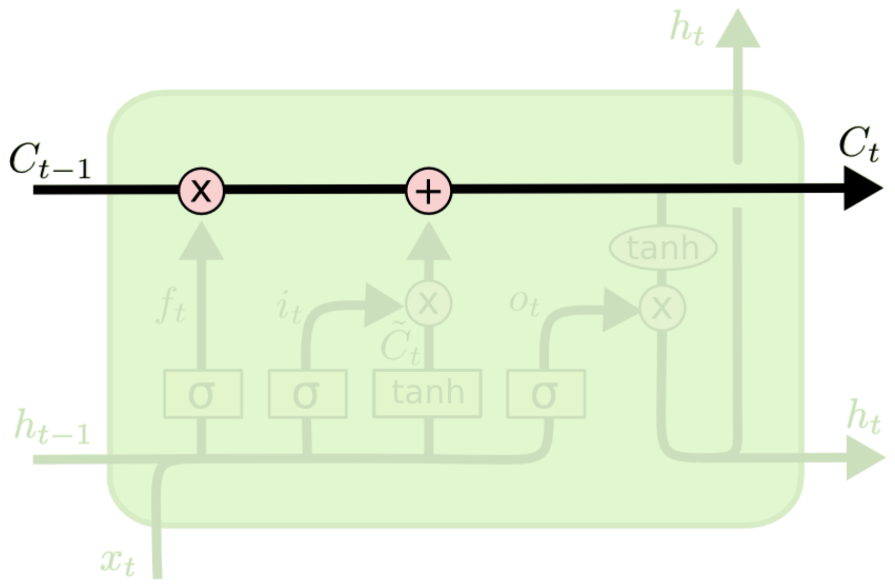
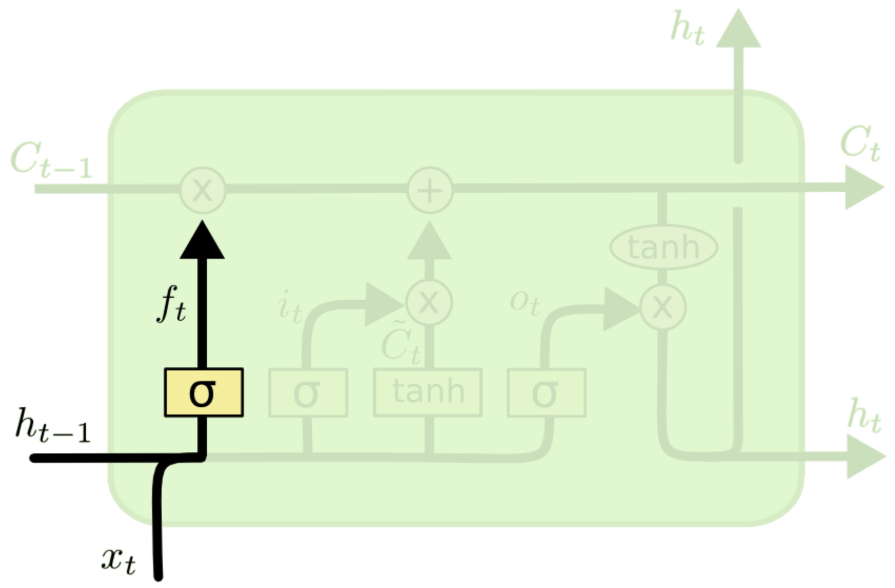


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTMs Intuition: Forget Gate

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

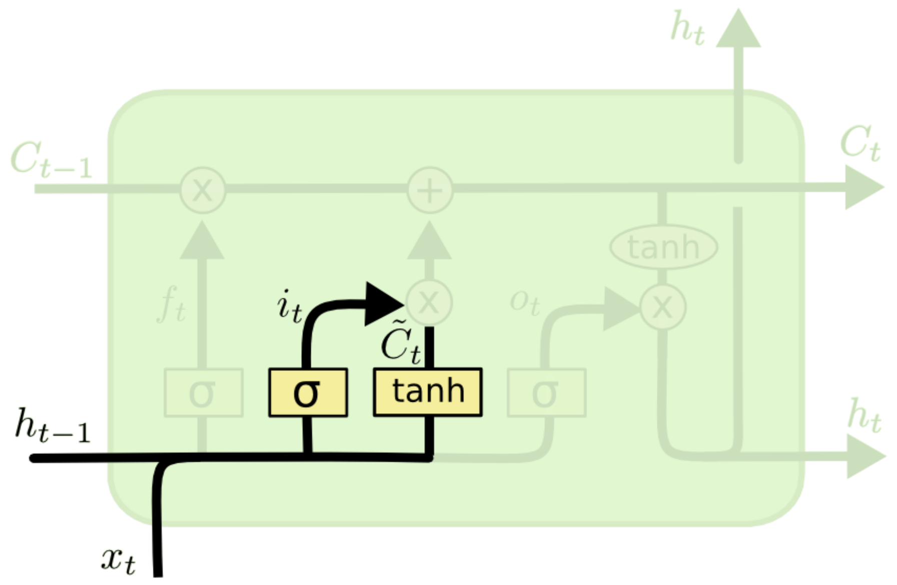


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

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?



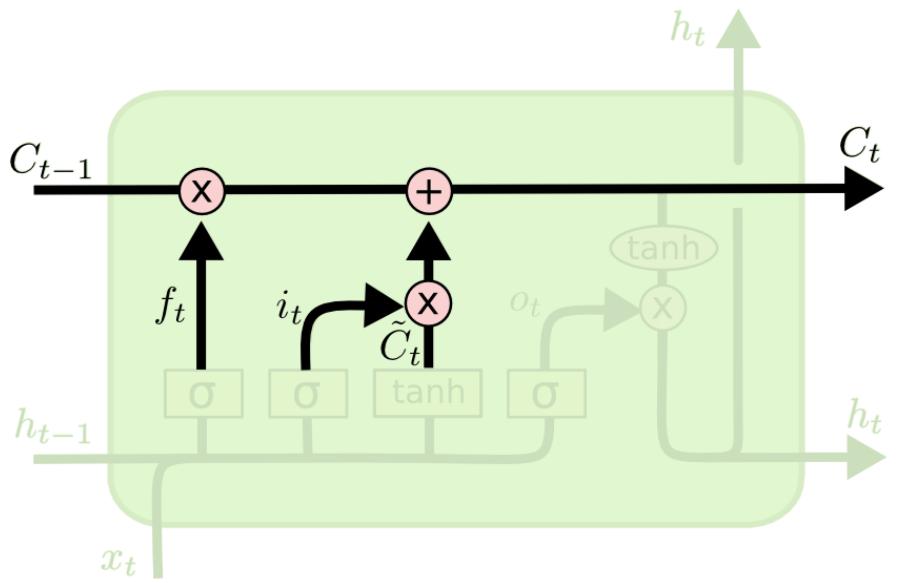
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

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

LSTMs Intuition: Memory Update

- Forget that + memorize this

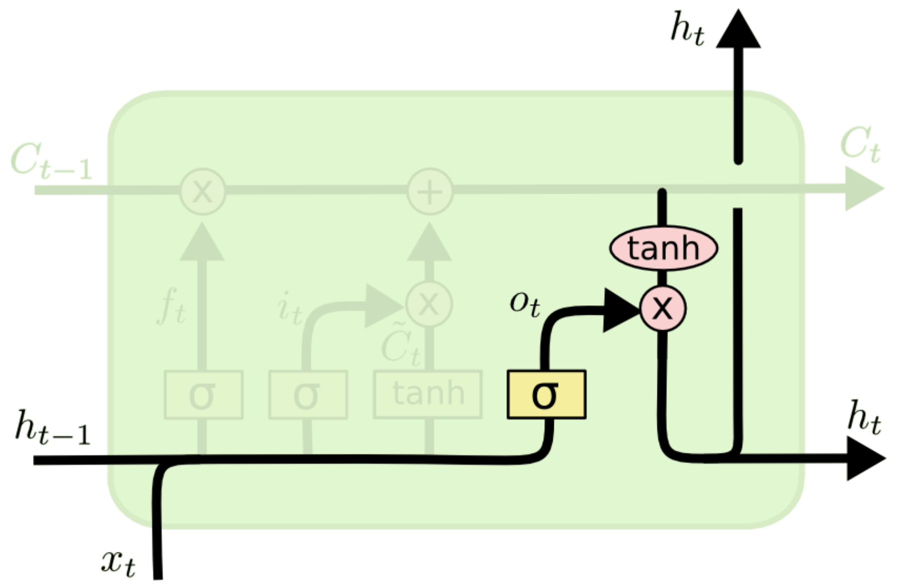


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

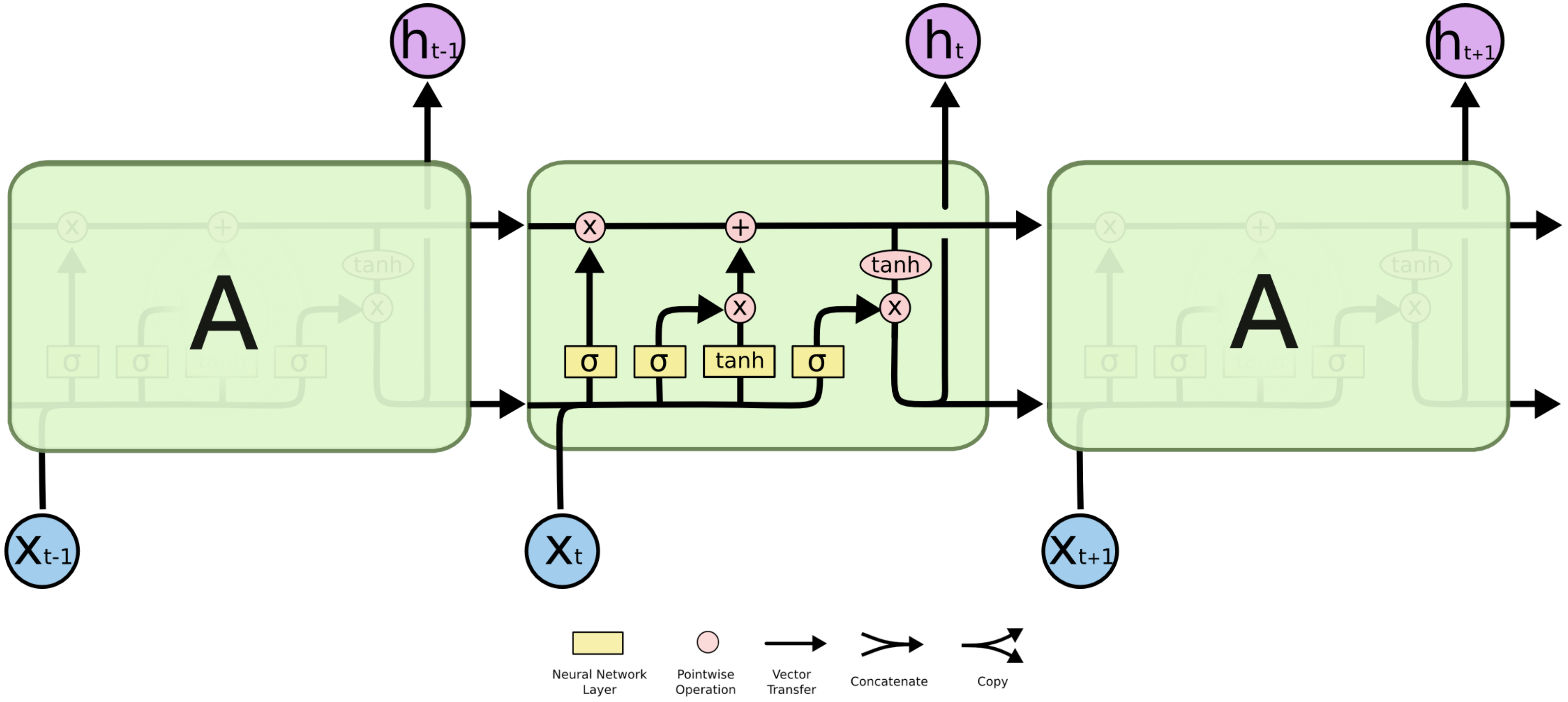
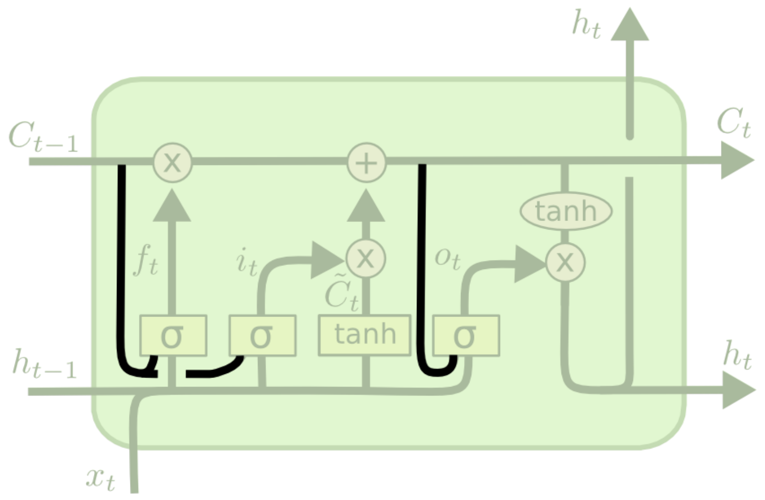


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

LSTM Variants #1: Peephole Connections

- Let gates see the cell state / memory

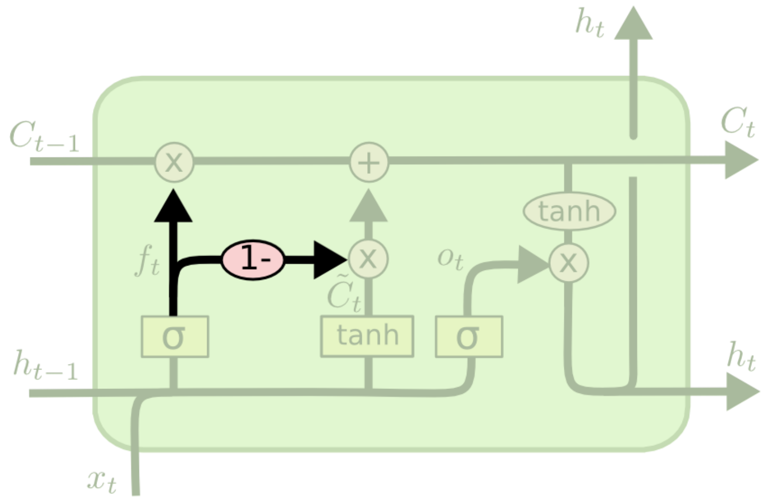


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

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

LSTM Variants #2: Coupled Gates

- Only memorize new if forgetting old

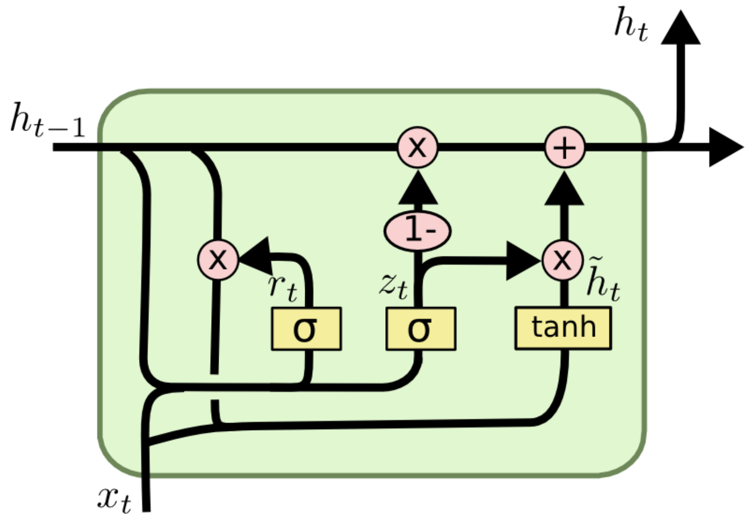


$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

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

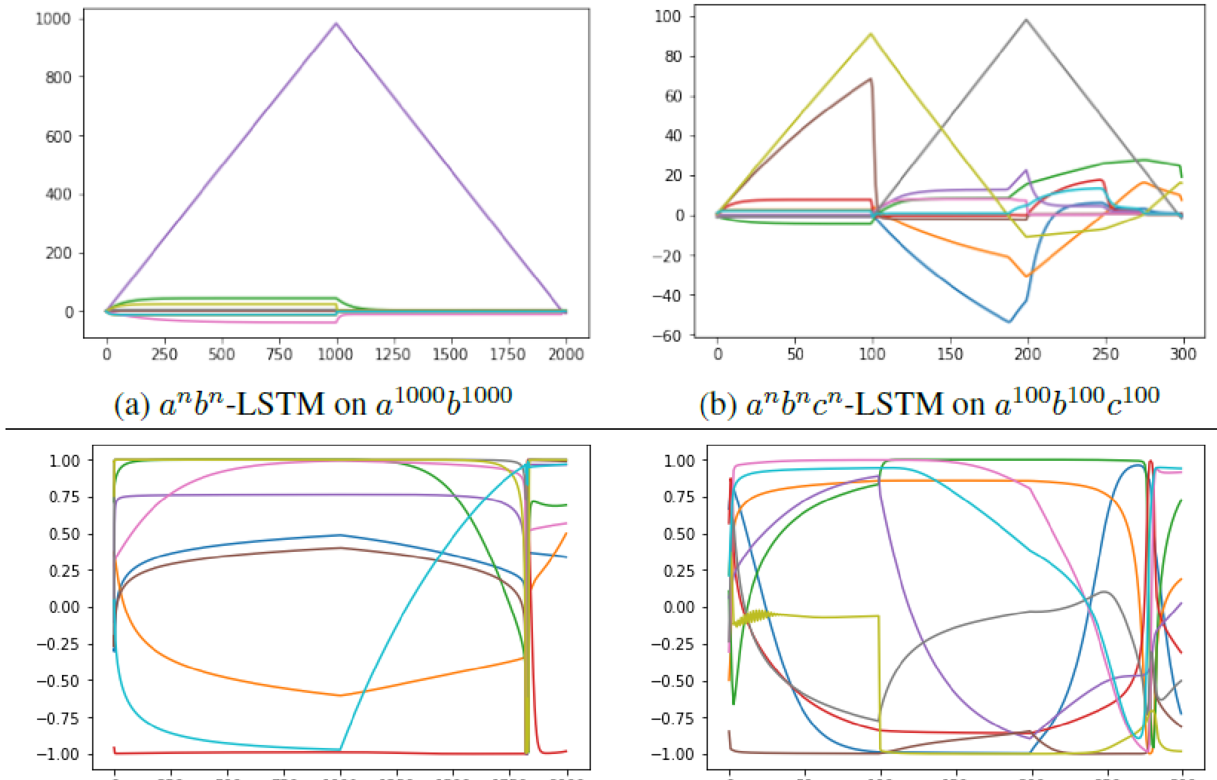
On the Practical Computational Power of Finite Precision RNNs for Language Recognition

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

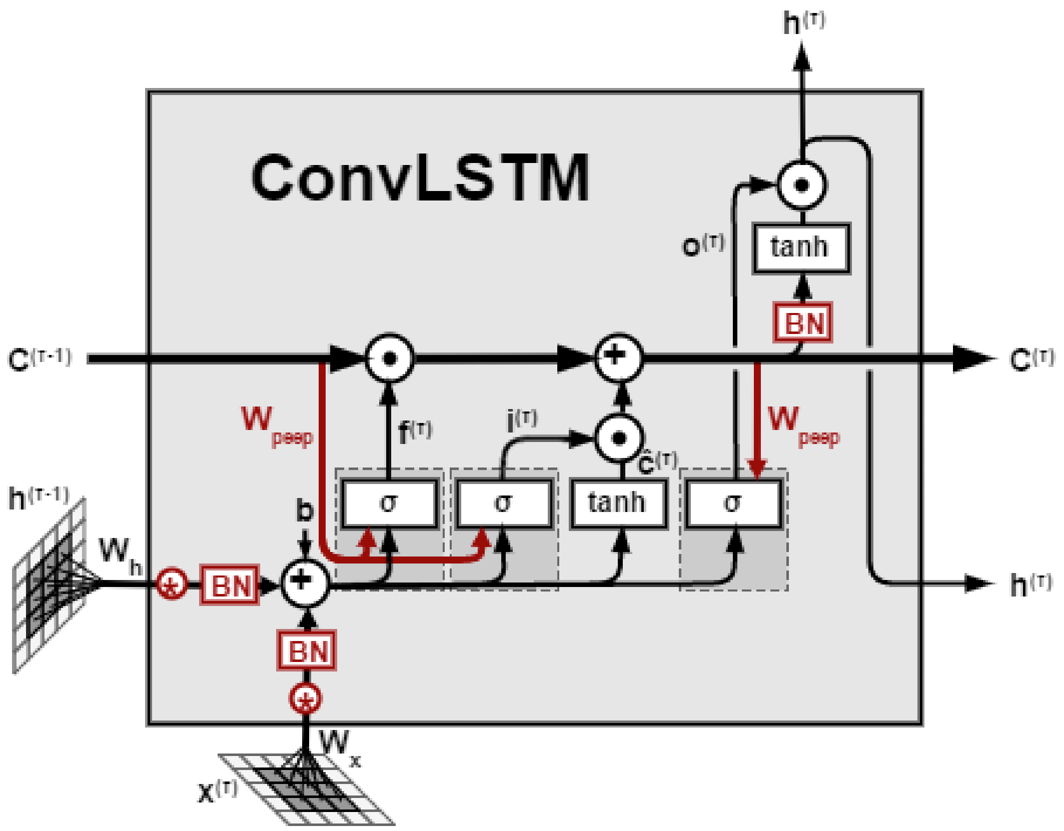


(a) $a^n b^n$-LSTM on $a^{1000}b^{1000}$ (b) $a^n b^n c^n$-LSTM on $a^{100}b^{100}c^{100}$

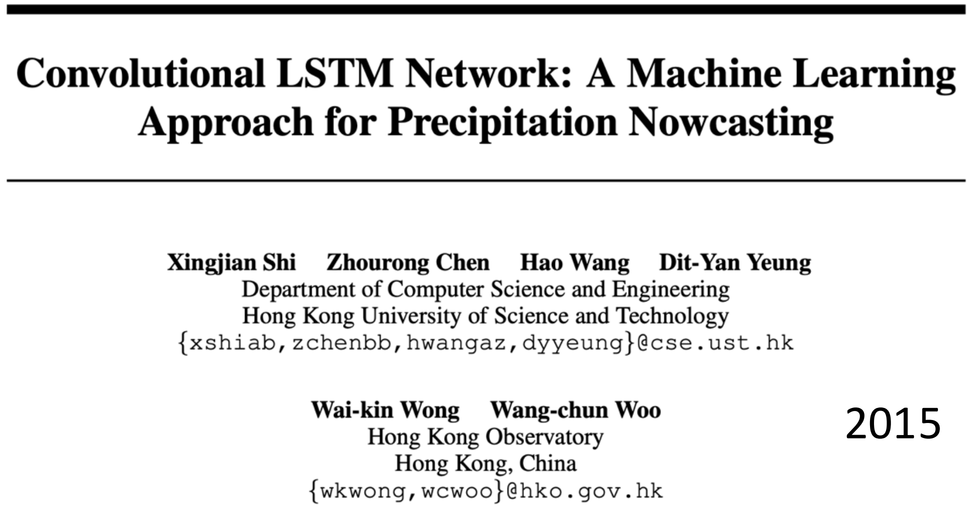
(c) $a^n b^n$-GRU on $a^{1000}b^{1000}$ (d) $a^n b^n c^n$-GRU on $a^{100}b^{100}c^{100}$

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

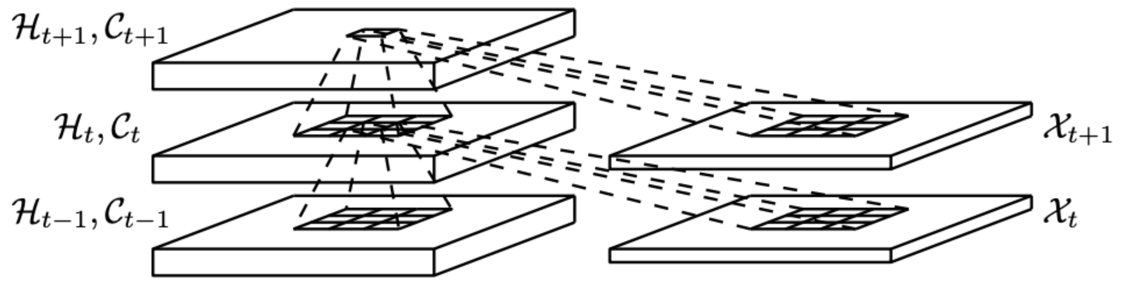

$\mathcal{H}_{t+1}, \mathcal{C}_{t+1}$
$\mathcal{H}_t, \mathcal{C}_t$
$\mathcal{H}_{t-1}, \mathcal{C}_{t-1}$
$\mathcal{X}_{t+1}$
$\mathcal{X}_t$

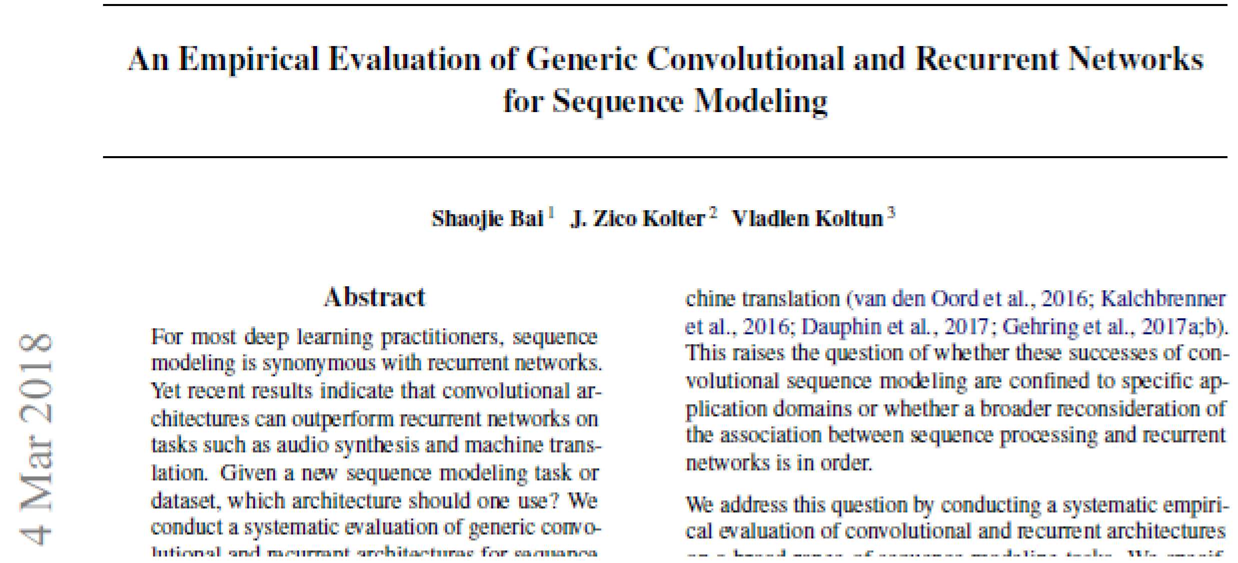
Figure 2: Inner structure of ConvLSTM

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]

4 Mar 2018

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 convo-

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

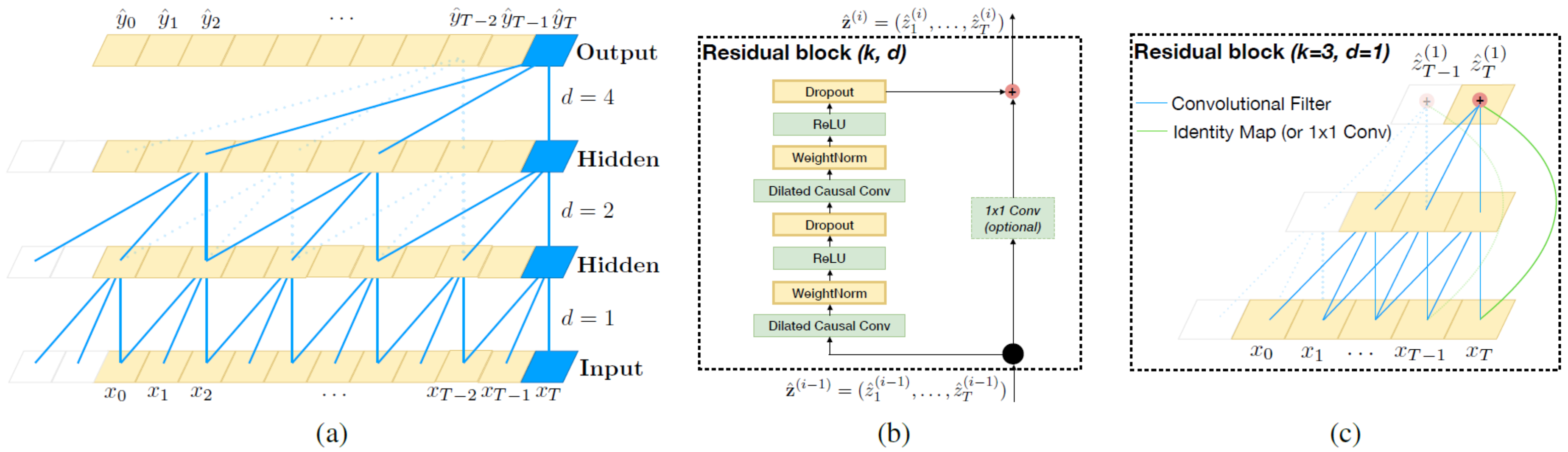

$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ ... $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 ... x_{T-2} x_{T-1} x_T
(a)
$\hat{\mathbf{z}}^{(i)} = (\hat{z}_1^{(i)}, \dots, \hat{z}_T^{(i)})$
Residual block (k, d)
Dropout
ReLU
WeightNorm
Dilated Causal Conv
Dropout
ReLU
WeightNorm
Dilated Causal Conv
1x1 Conv (optional)
$\hat{\mathbf{z}}^{(i-1)} = (\hat{z}_1^{(i-1)}, \dots, \hat{z}_T^{(i-1)})$
(b)
Residual block (k=3, d=1)
$\hat{z}_{T-1}^{(1)}$ $\hat{z}_T^{(1)}$
Convolutional Filter
Identity Map (or 1x1 Conv)
x_0 x_1 ... x_{T-1} x_T
(c)

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:

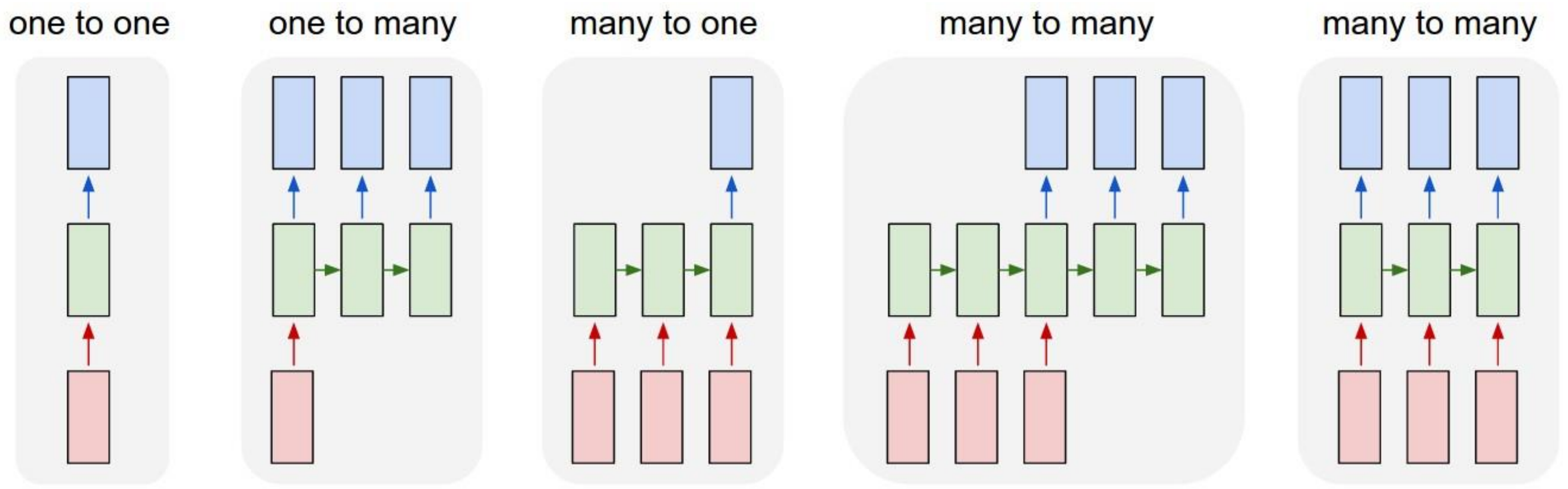
$$p(c_{n+1} \mid c_n, \ldots, c_1)$$

- 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

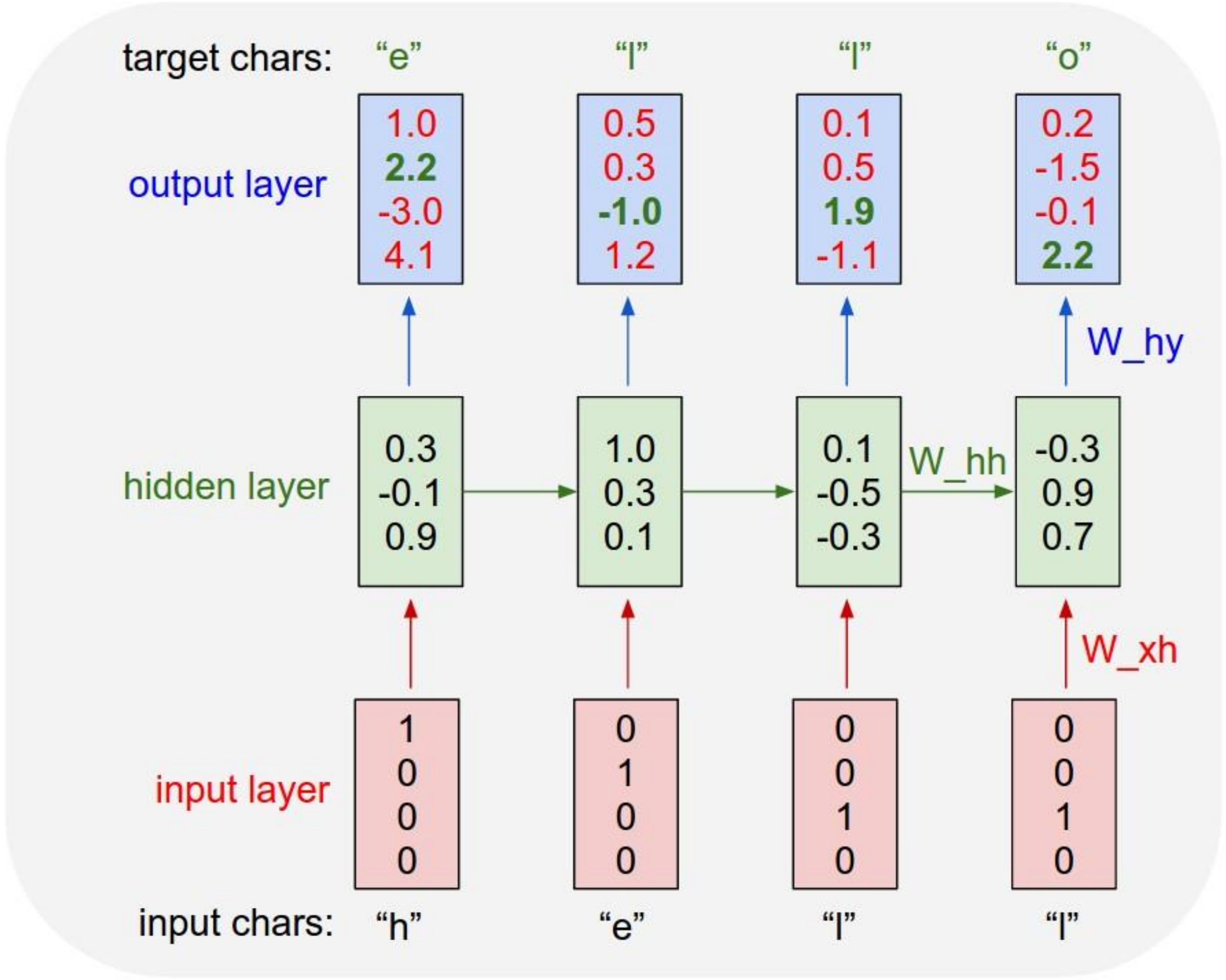
$$p(c_{n+1} = 'a' \mid 'Ankar') > p(c_{n+1} = 'o' \mid 'Ankar')$$



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

A simple scenario

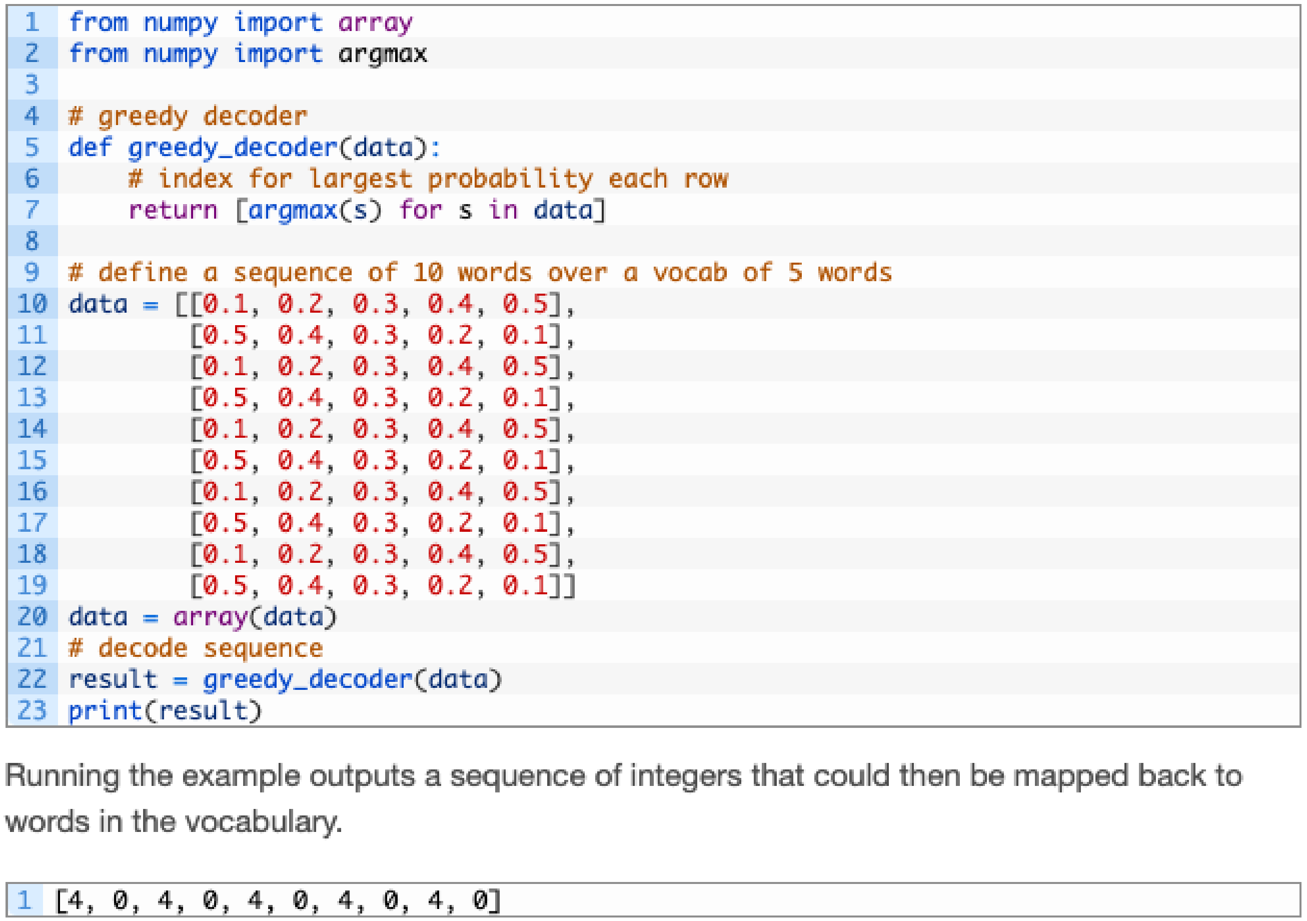
- Alphabet: h, e, l, o
- Text to train to predict: "hello"



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.

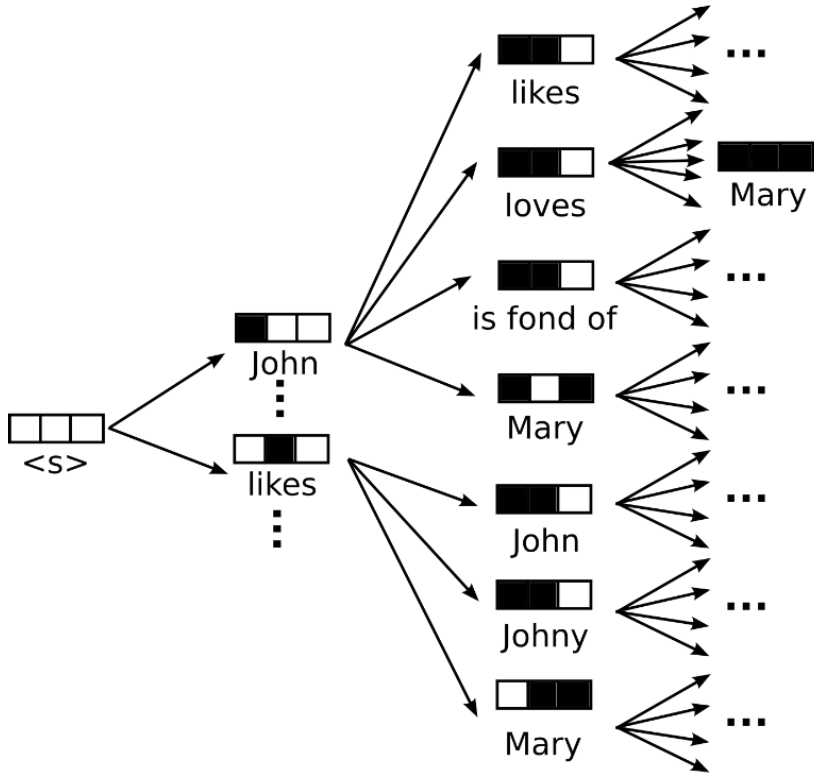


Figure: http://mttalks.ufal.ms.mff.cuni.cz/index.php

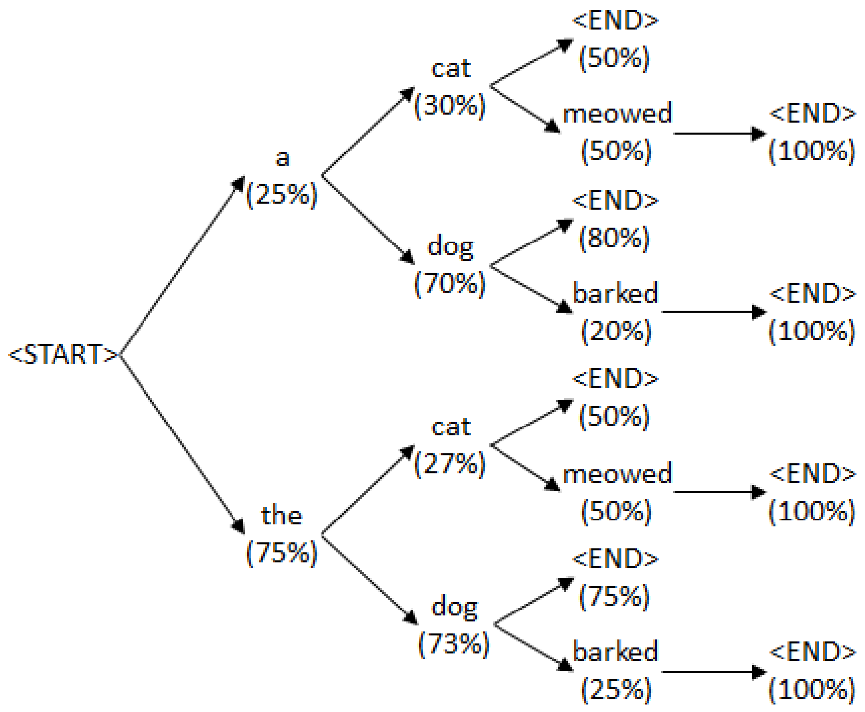
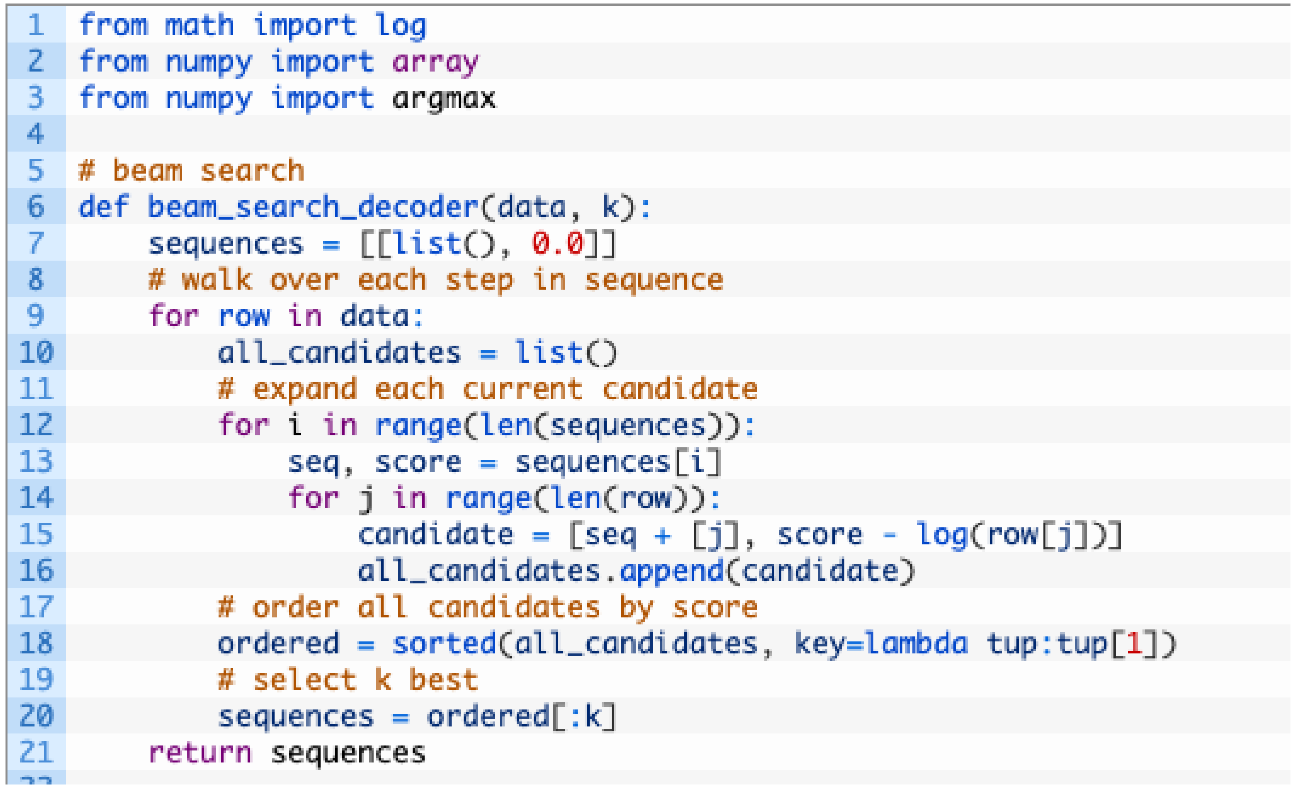


Figure: https://geekyisawesome.blogspot.com.tr/2016/10/using-beam-search-to-generate-most.html

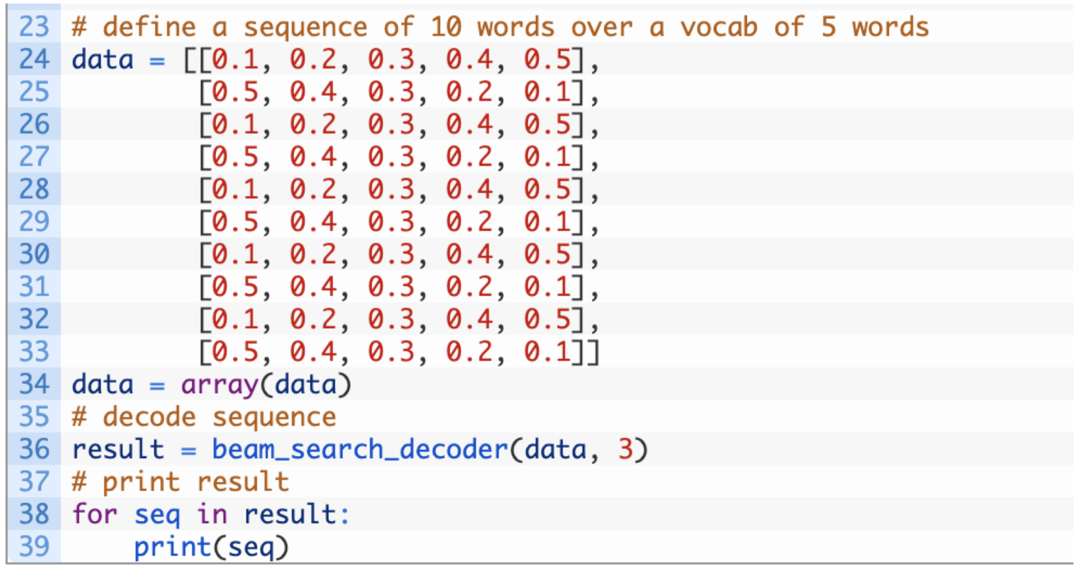
Sampling: Beam Search

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

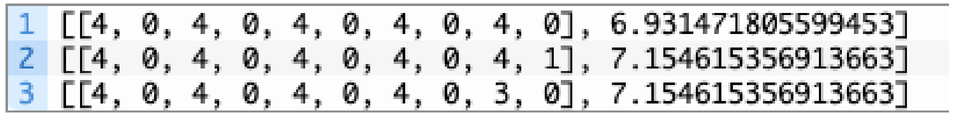
  (take log for numerical stability; take –log() for minimizing the score)



```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)
```

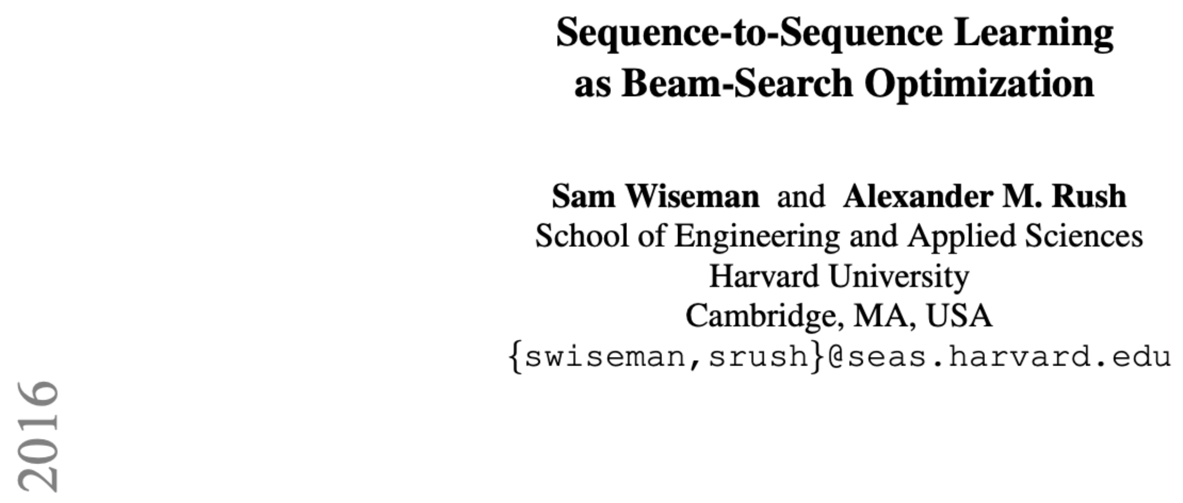


```
[[4, 0, 4, 0, 4, 0, 4, 0, 4, 0], 6.931471805599453]
[[4, 0, 4, 0, 4, 0, 4, 0, 4, 1], 7.154615356913663]
[[4, 0, 4, 0, 4, 0, 4, 0, 3, 0], 7.154615356913663]
```

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

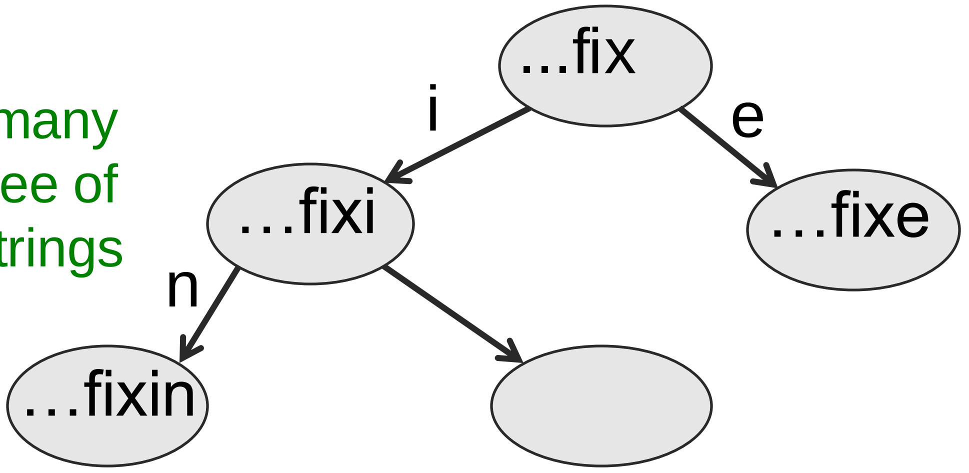
Sam Wiseman and **Alexander M. Rush**
School of Engineering and Applied Sciences
Harvard University
Cambridge, MA, USA
{swiseman,srush}@seas.harvard.edu

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

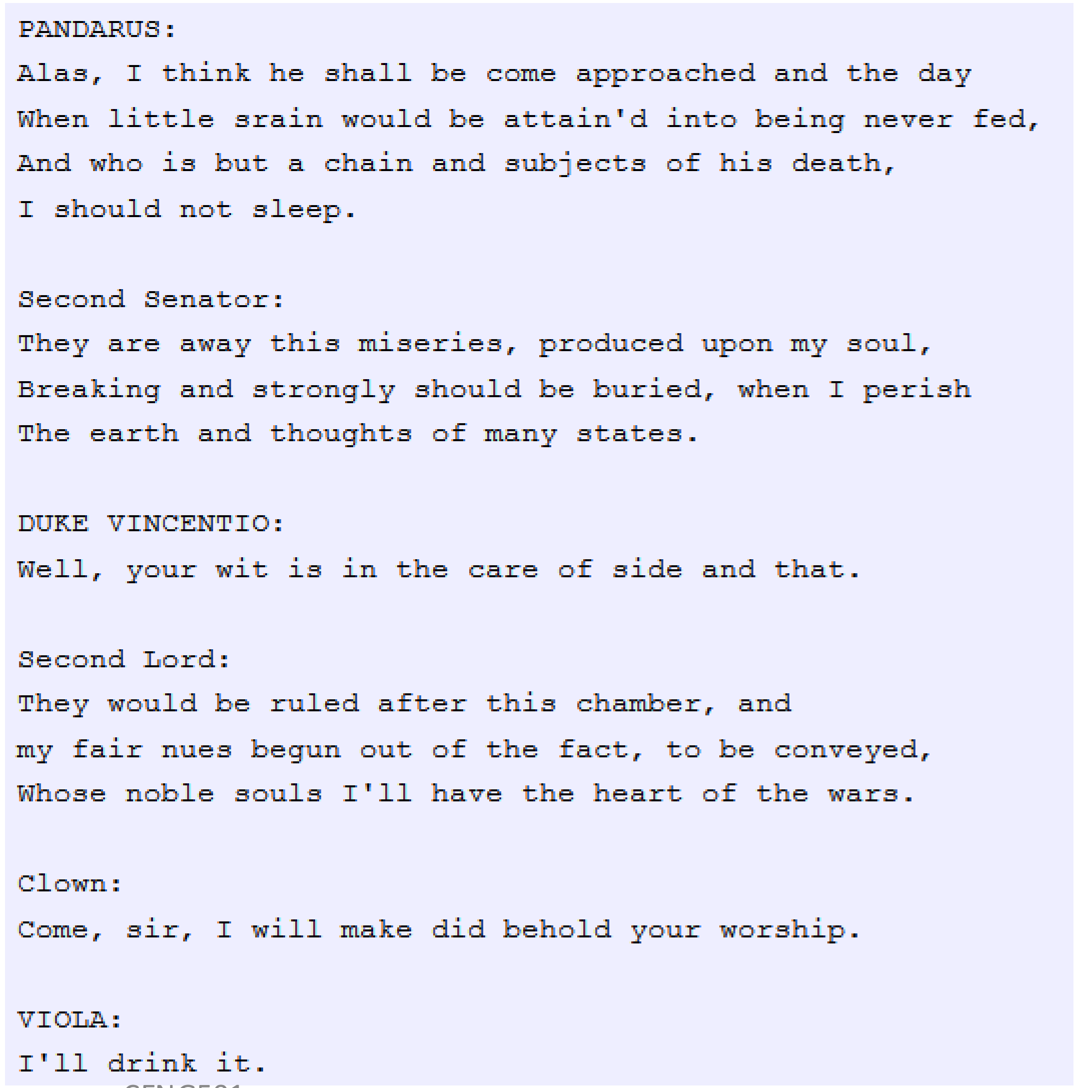
- The web is composed of character strings.
- Any learning method powerful enough to understand the world by reading the web ought to find it trivial to learn which strings make words (this turns out to be true, as we shall see).
- Pre-processing text to get words is a big hassle
  - What about morphemes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

ymmärtämättömyydellänsäkään

Slide: Hinton

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

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

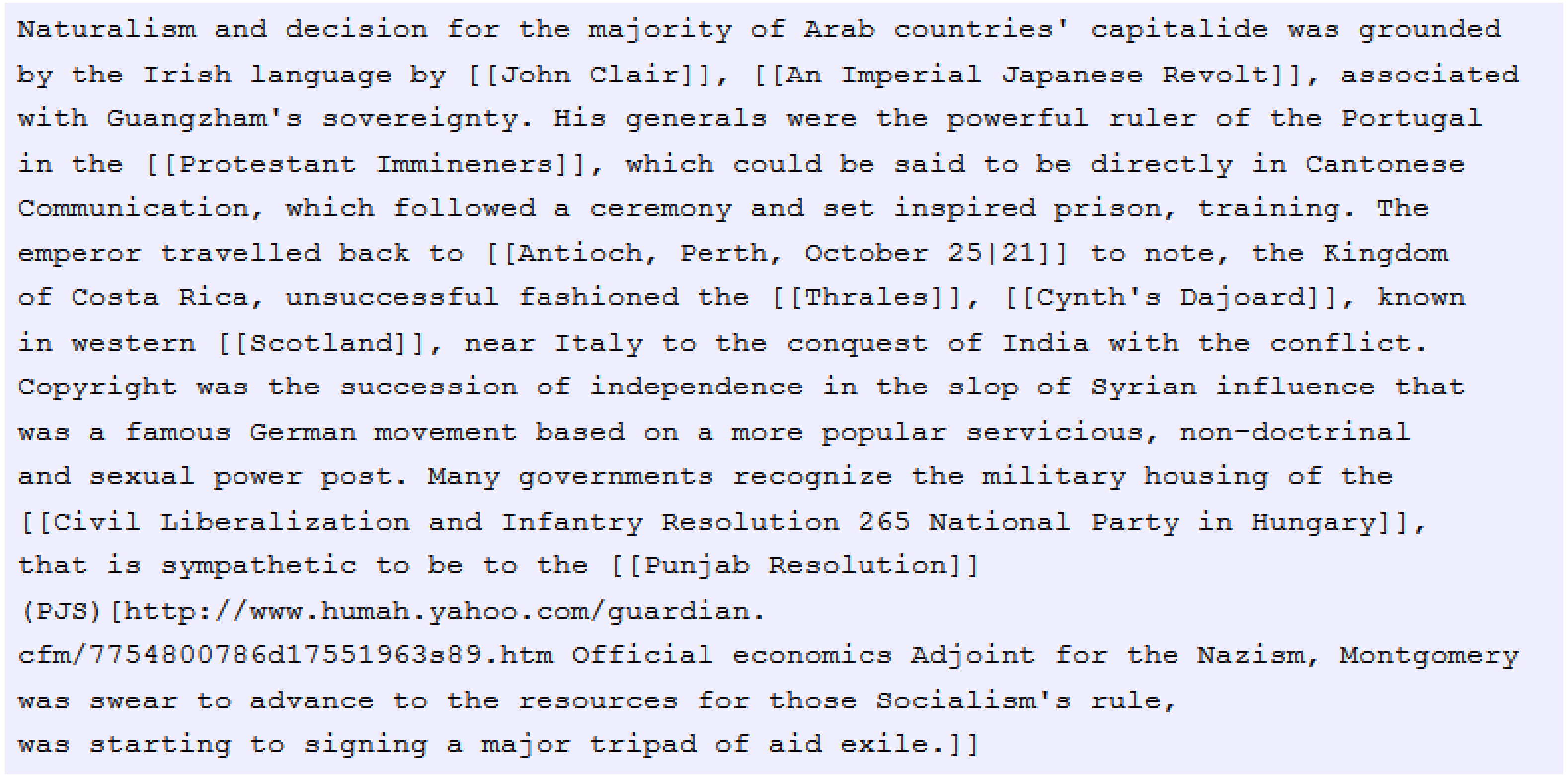


```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

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

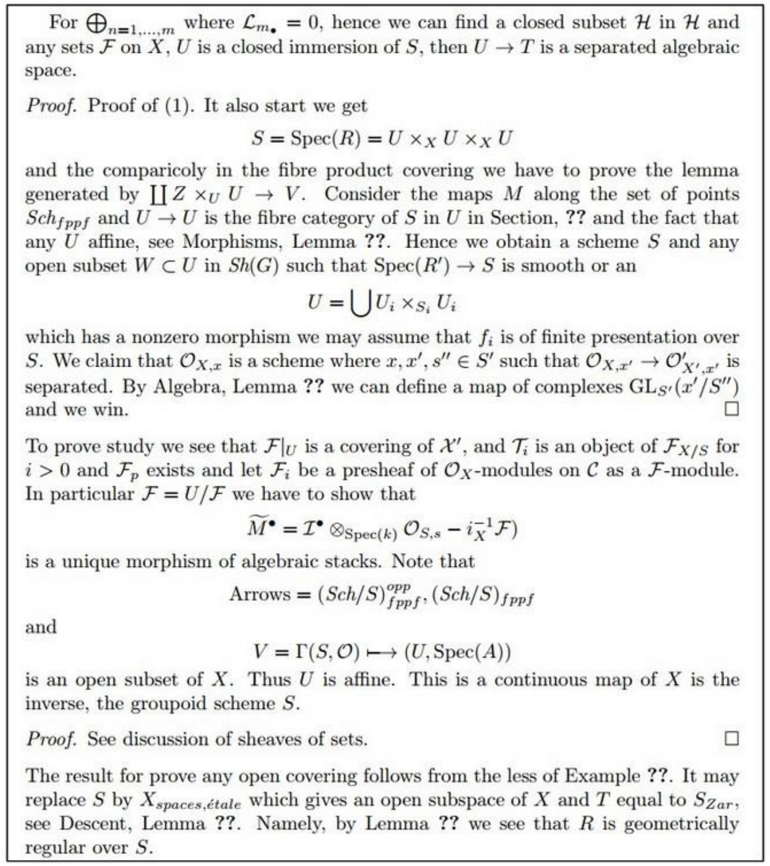


```
Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could 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 $\bigoplus_{n=1,\dots,m}$ 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 comparicoly 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

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

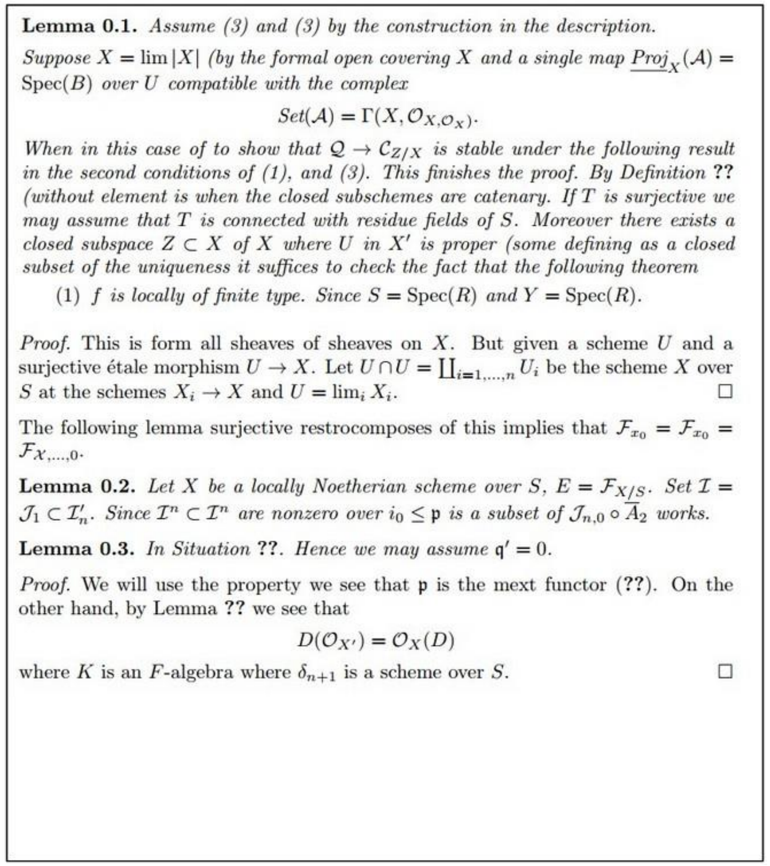
and

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

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 $U \cap U = \coprod_{i=1,\dots,n} U_i$ 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 $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\dots,0}$.

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.”

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. (6th 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).

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:

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

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

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

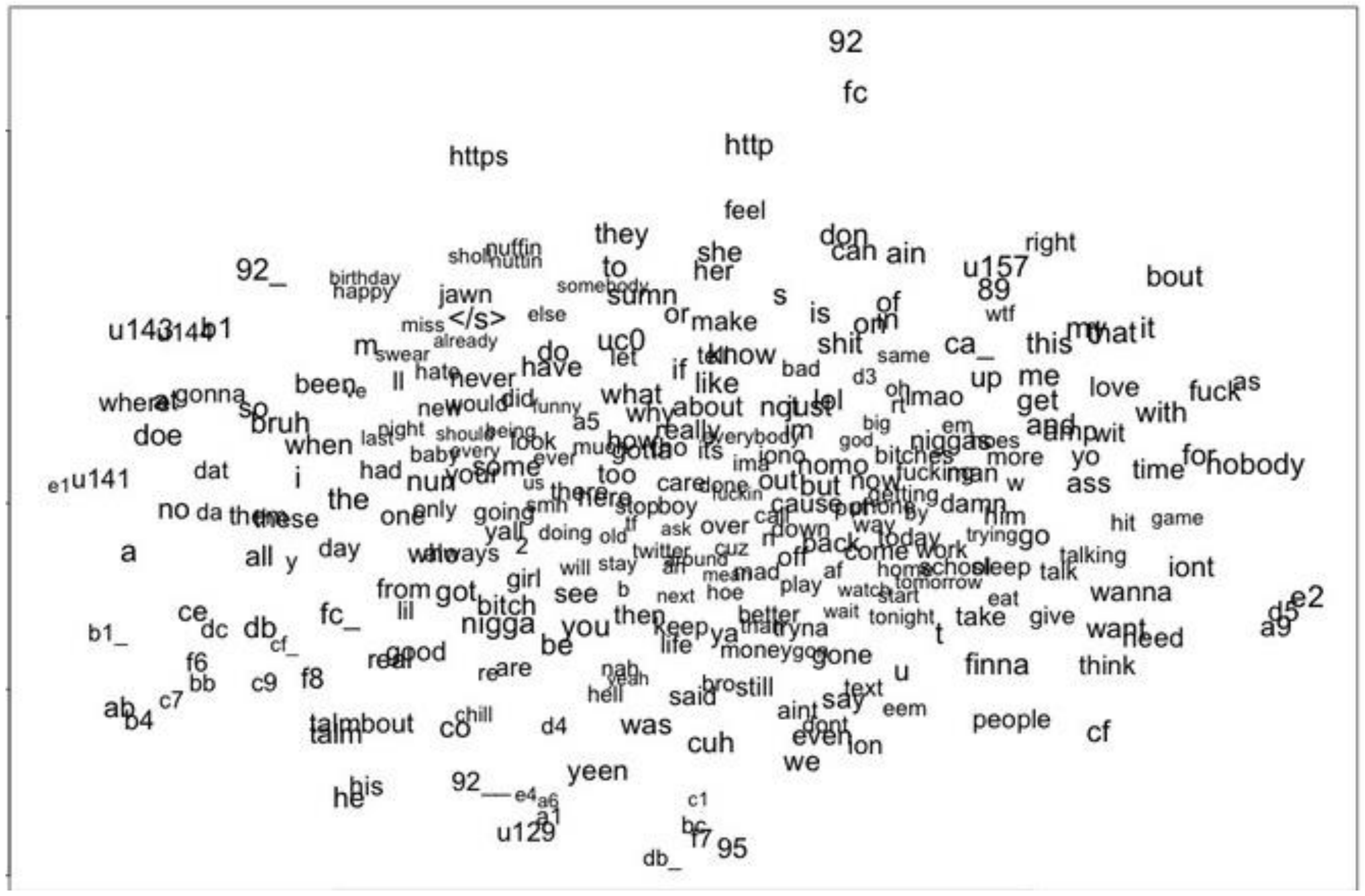
- Learn $p(\omega_{n+1} = 'Turkey' \;|\, 'Ankara\ is\ the\ capital\ of\ ')$ from data such that:

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

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).

A two dimensional reduction of the vector space model using t-SNE

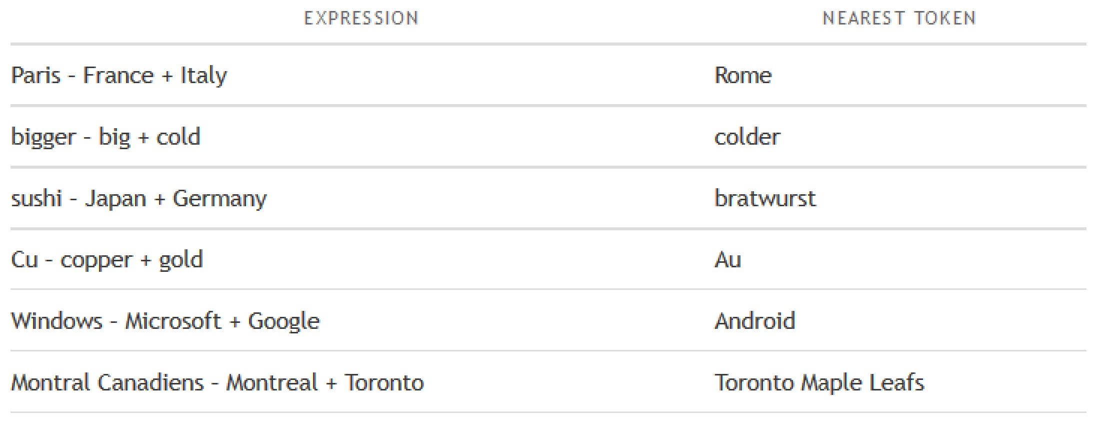


Word Embedding (word2vec)

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

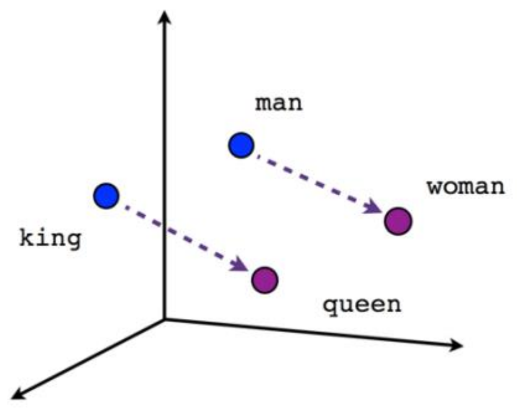


EXPRESSION	NEAREST TOKEN
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montral Canadiens - Montreal + Toronto	Toronto Maple Leafs

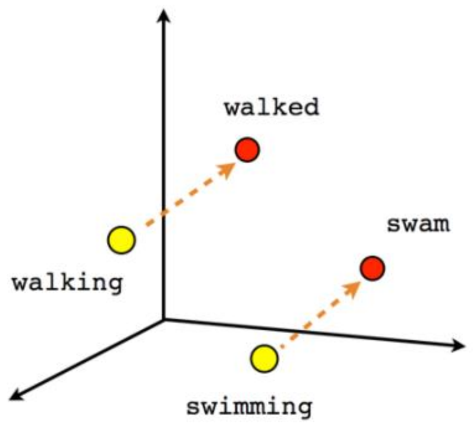
Table 1: Mikolov et al. [3] showcase simple additive properties of their word embeddings.

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

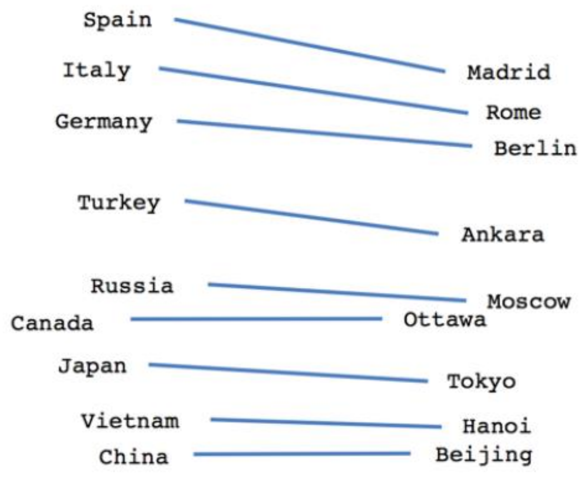
More examples



Male-Female



Verb tense



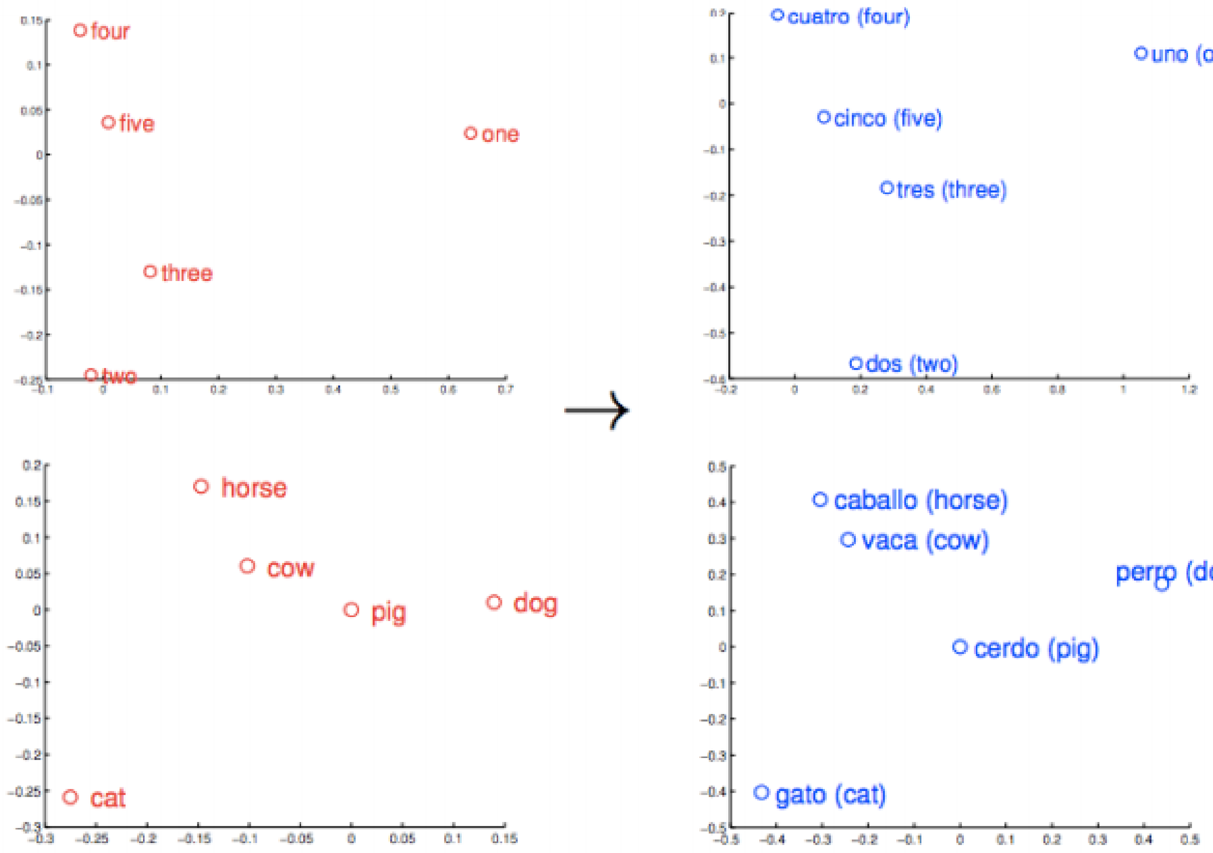
Country-Capital

More examples

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

http://deeplearning4j.org/word2vec

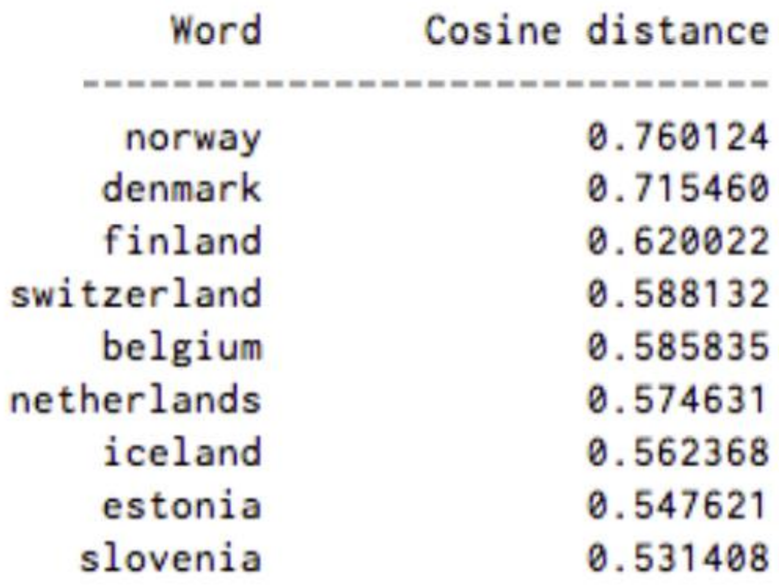
More examples



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
```

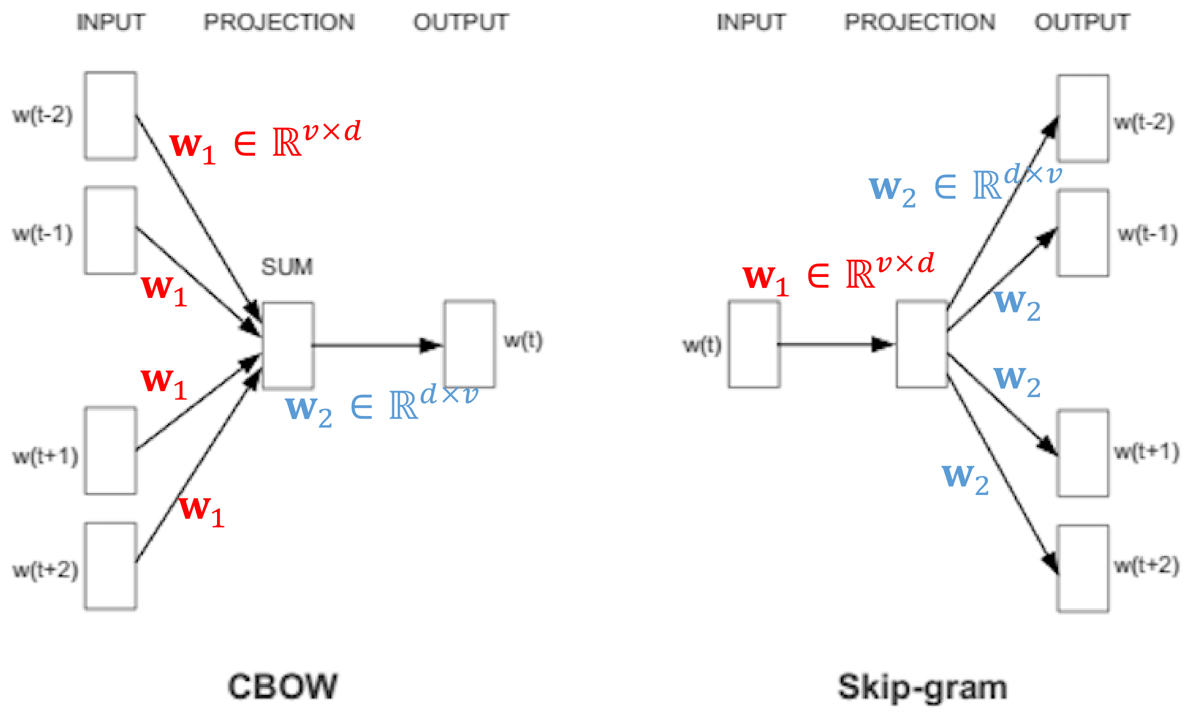
http://deeplearning4j.org/word2vec

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



http://deeplearning4j.org/word2vec

Note that the weight matrix is a look-up table

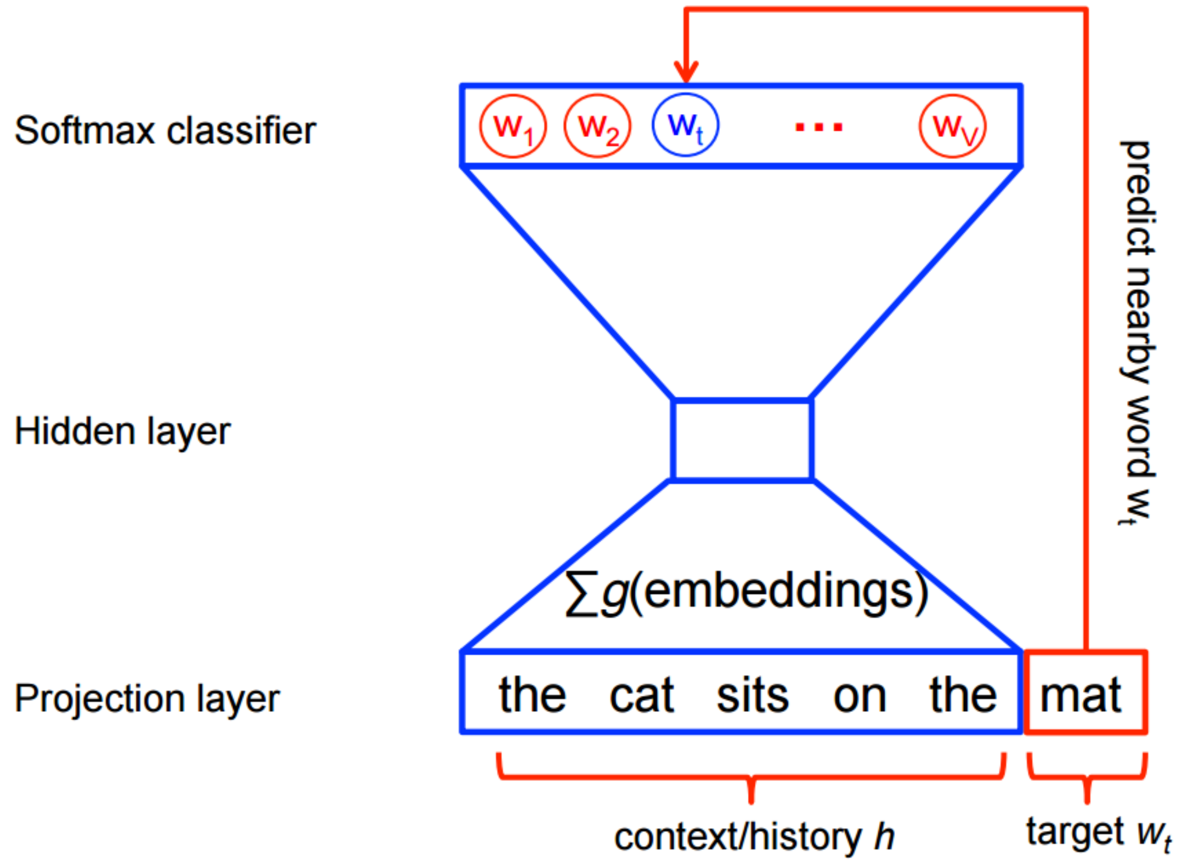
- In both approaches, the weight matrix is used as follows:

$$\underset{1\times V}{\overset{\text{input}}{\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}}}\;\underbrace{\overset{W_1}{\underset{V\times N}{}}\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}}_{W_1} = \underset{1\times N}{\overset{\text{hidden}}{\begin{bmatrix} e & f & g & h \end{bmatrix}}}$$

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

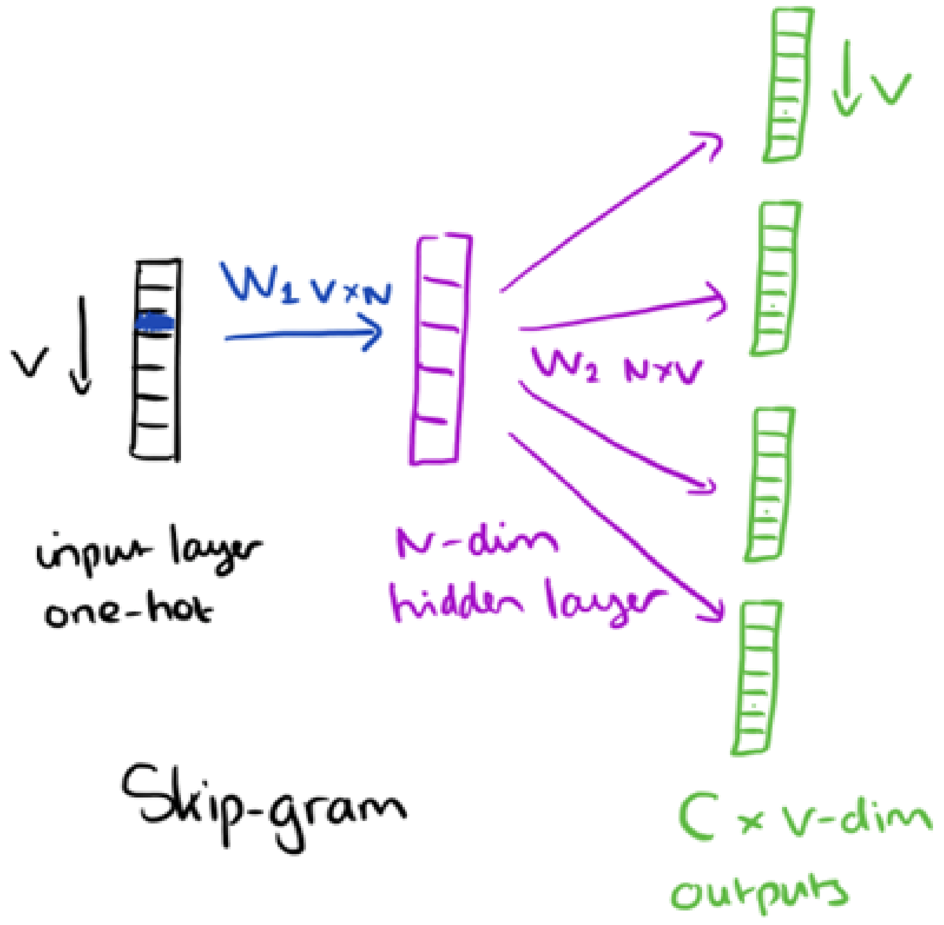
- For each word, take context to be

  (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

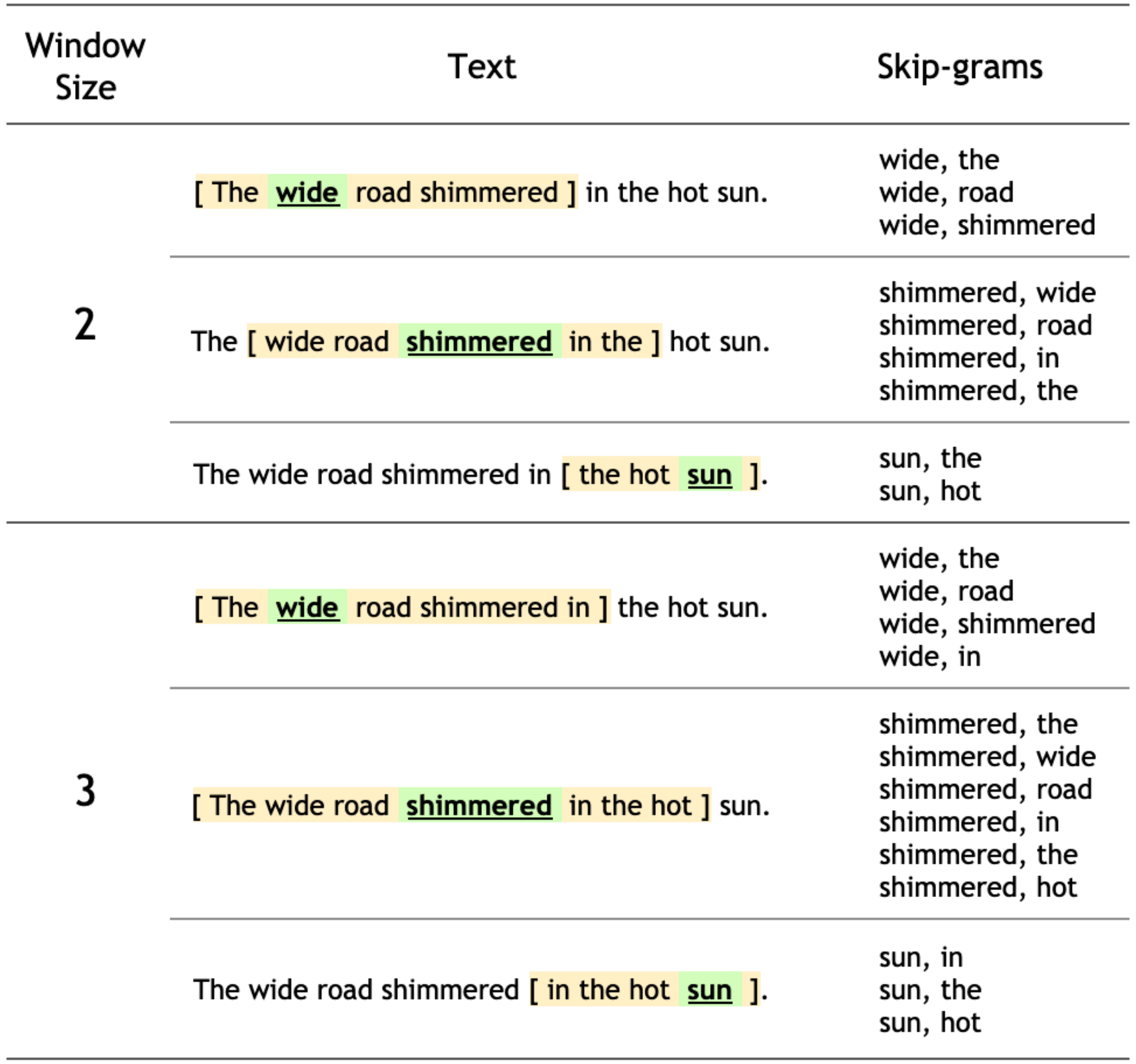


$$\overset{\substack{\text{input}\\ 1\times V}}{\begin{bmatrix}0 & 1 & 0\end{bmatrix}} \underset{W_1}{\overset{\substack{W_1\\ V\times N}}{\begin{bmatrix}a & b & c & d\\ e & f & g & h\\ i & j & k & l\end{bmatrix}}} = \overset{\substack{\text{hidden}\\ 1\times N}}{\begin{bmatrix}e & f & g & h\end{bmatrix}}$$

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”*

a man is playing tennis on a tennis court

a train is traveling down the tracks at a train station

a cake with a slice cut out of it

a bench sitting on a patch of grass next to a sidewalk

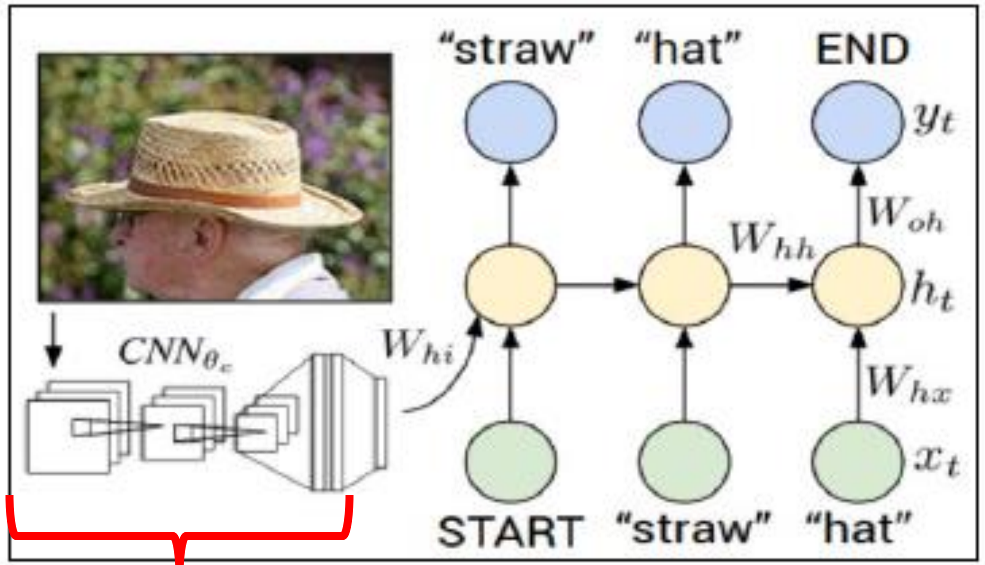
Fig: https://github.com/karpathy/neuraltalk2

Example: Image Captioning

Demo video

https://vimeo.com/146492001

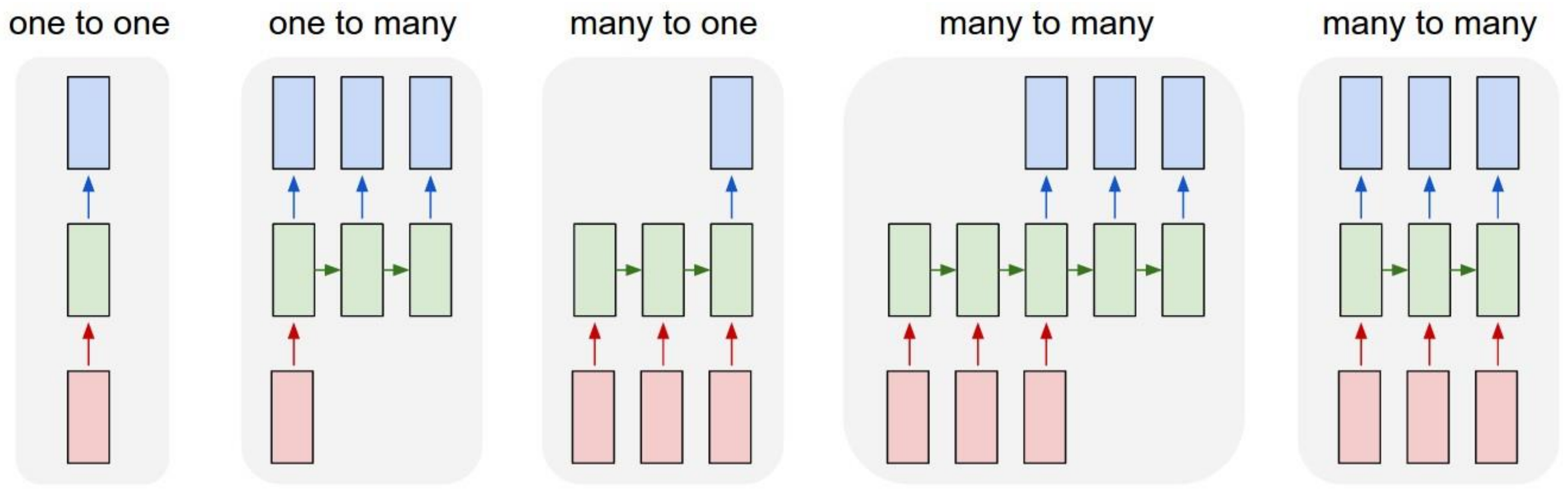
Overview



Pre-trained word embedding is also used

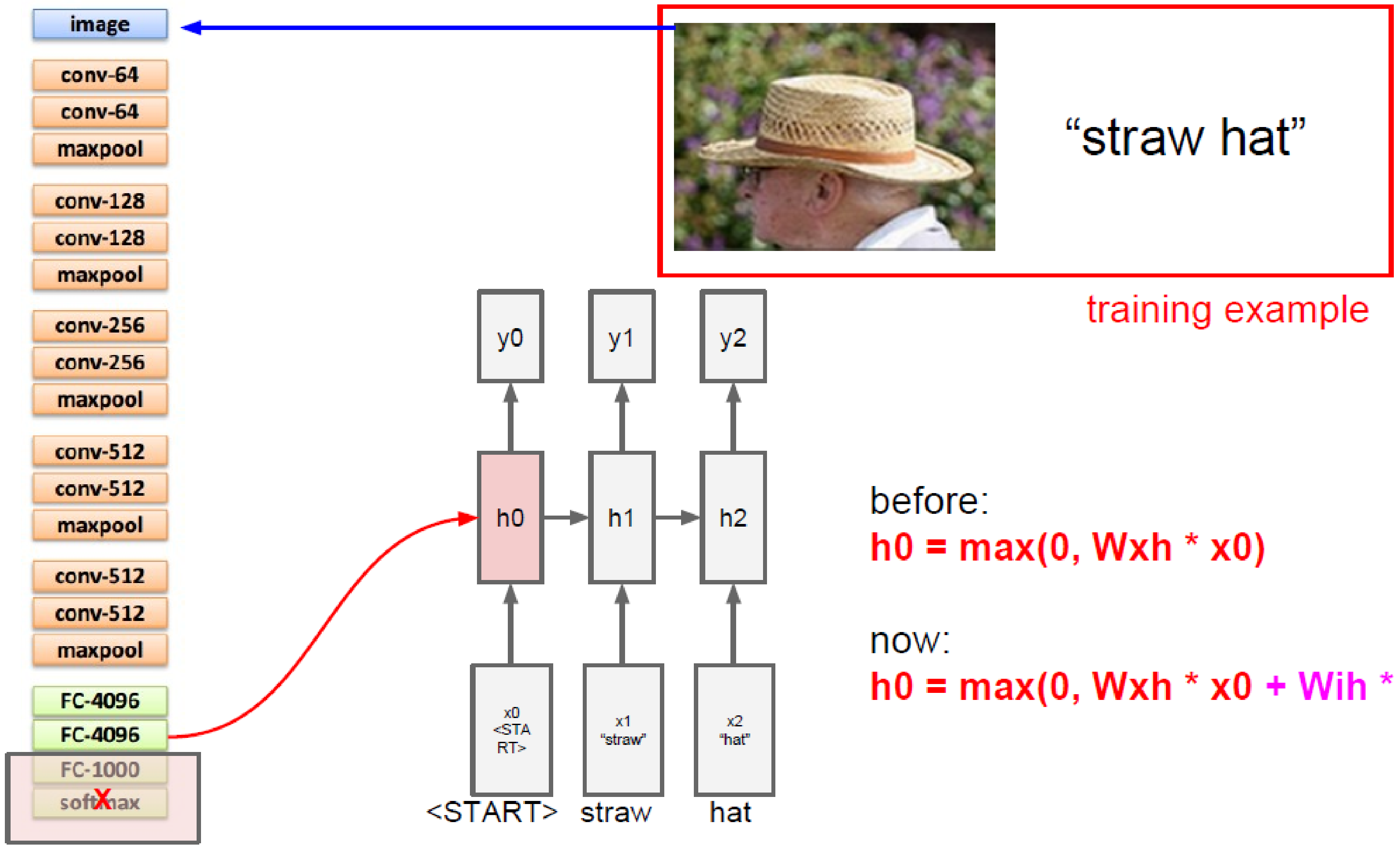
Pre-trained CNN (e.g., on imagenet)

Image: Karpathy



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

Training



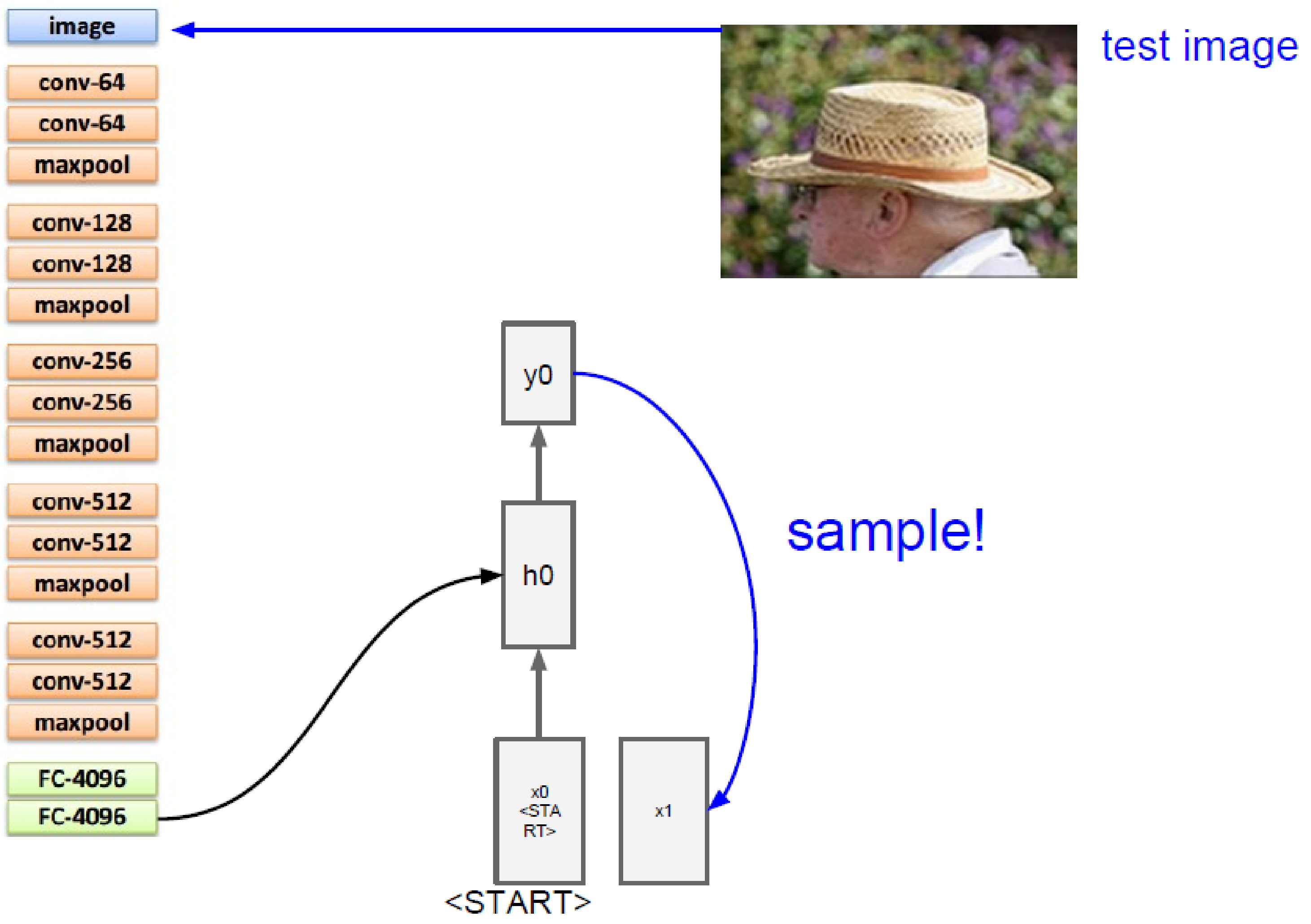
before:

h0 = max(0, Wxh * x0)

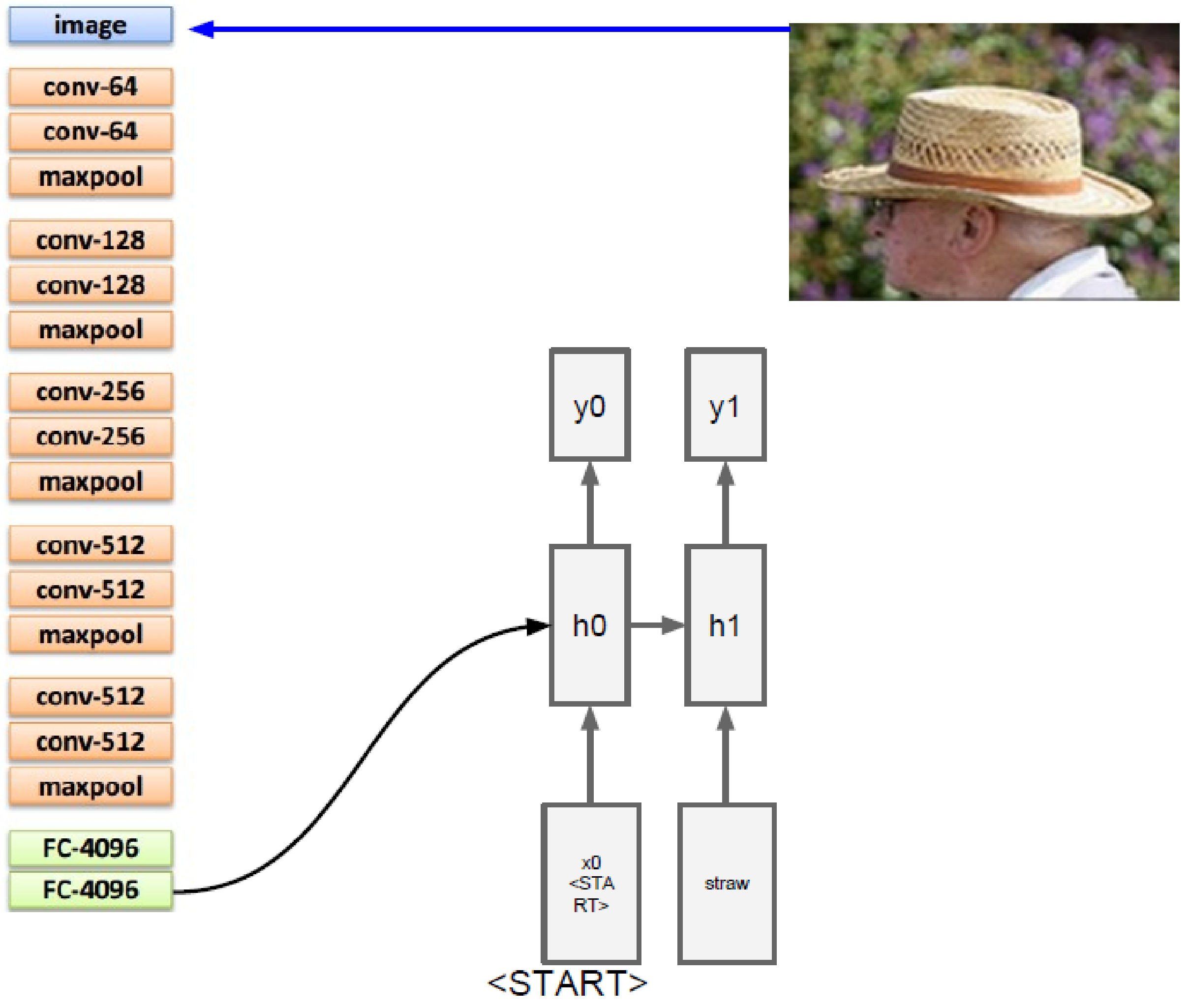
now:

h0 = max(0, Wxh * x0 + Wih * v)

Slide: Karpathy

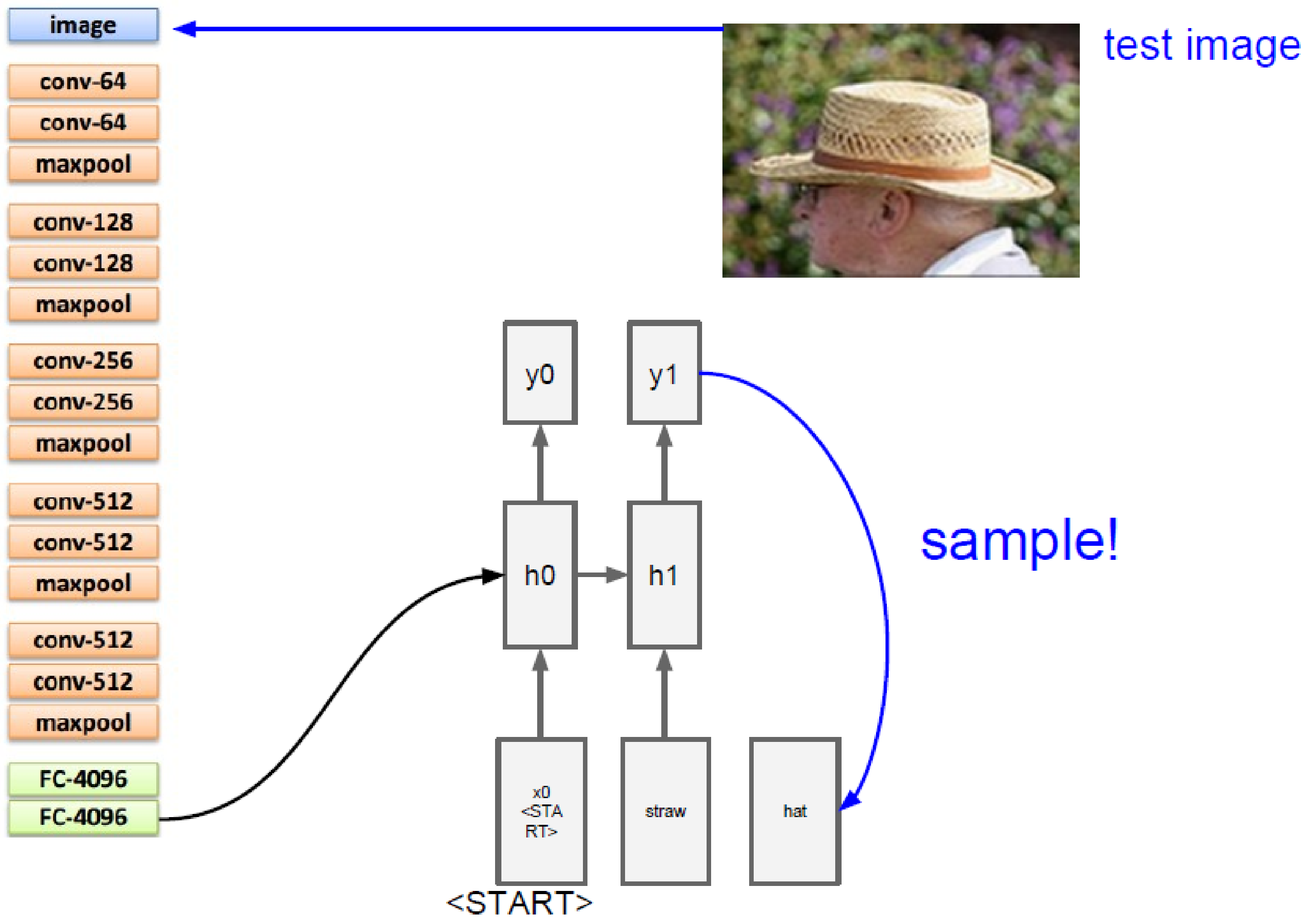


Slide: Karpathy

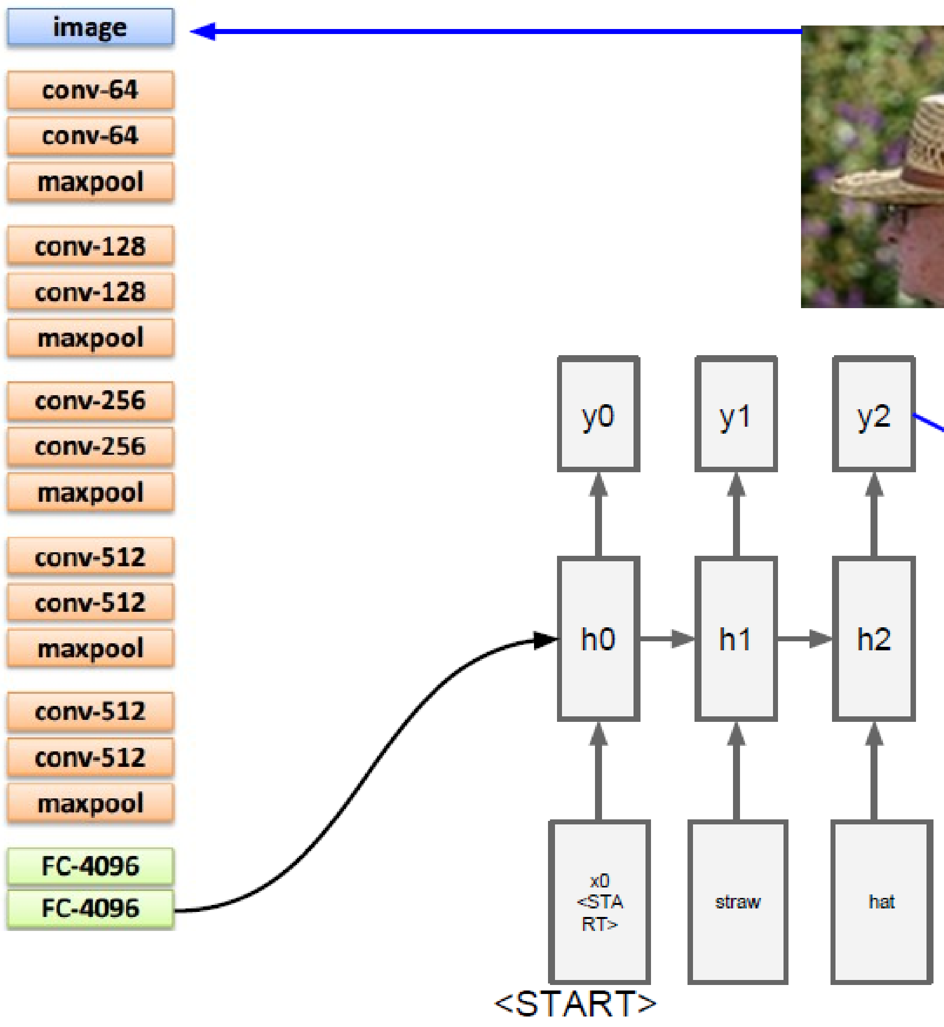


test image

Slide: Karpathy



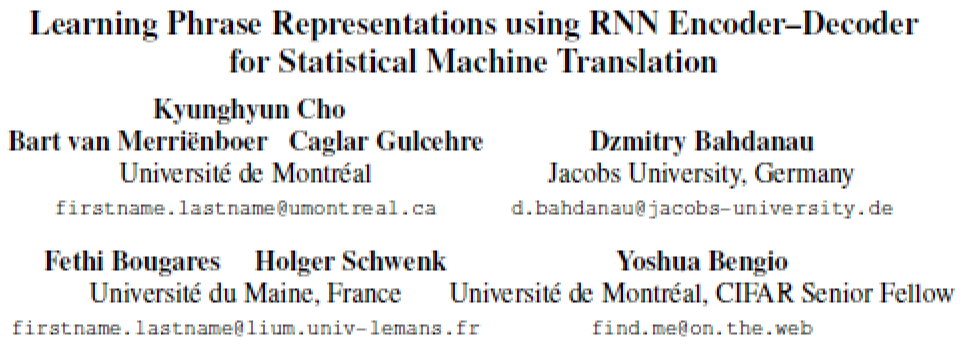
Slide: Karpathy



test image

sample!
<END> token
=> finish.

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

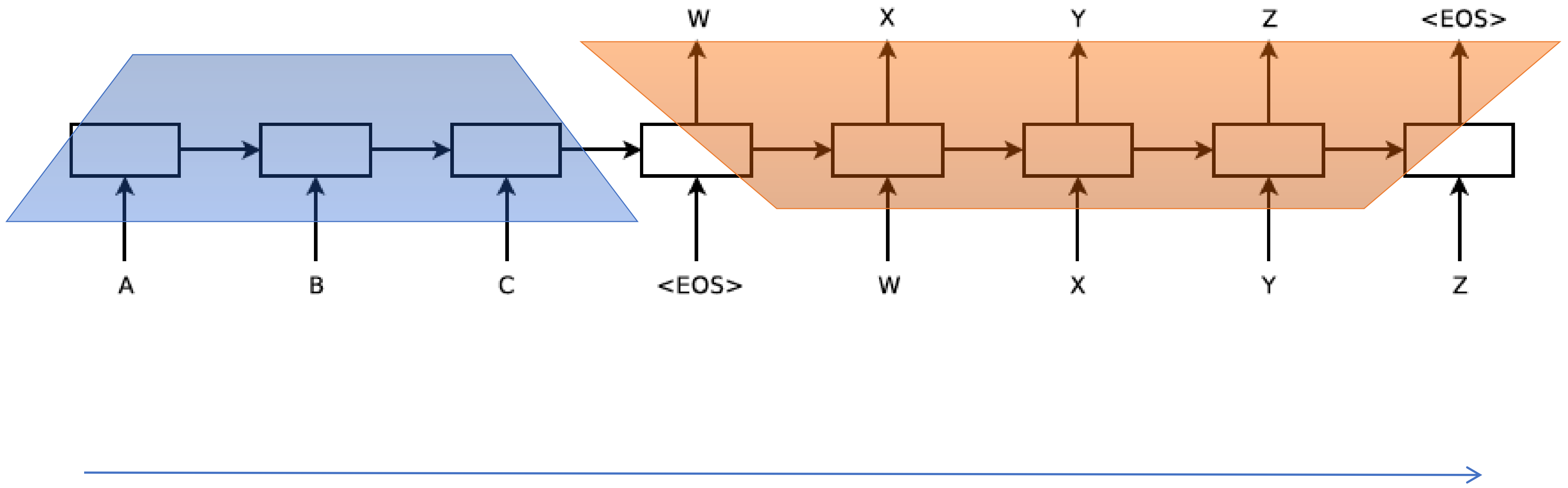
2014

Example: Neural Machine Translation

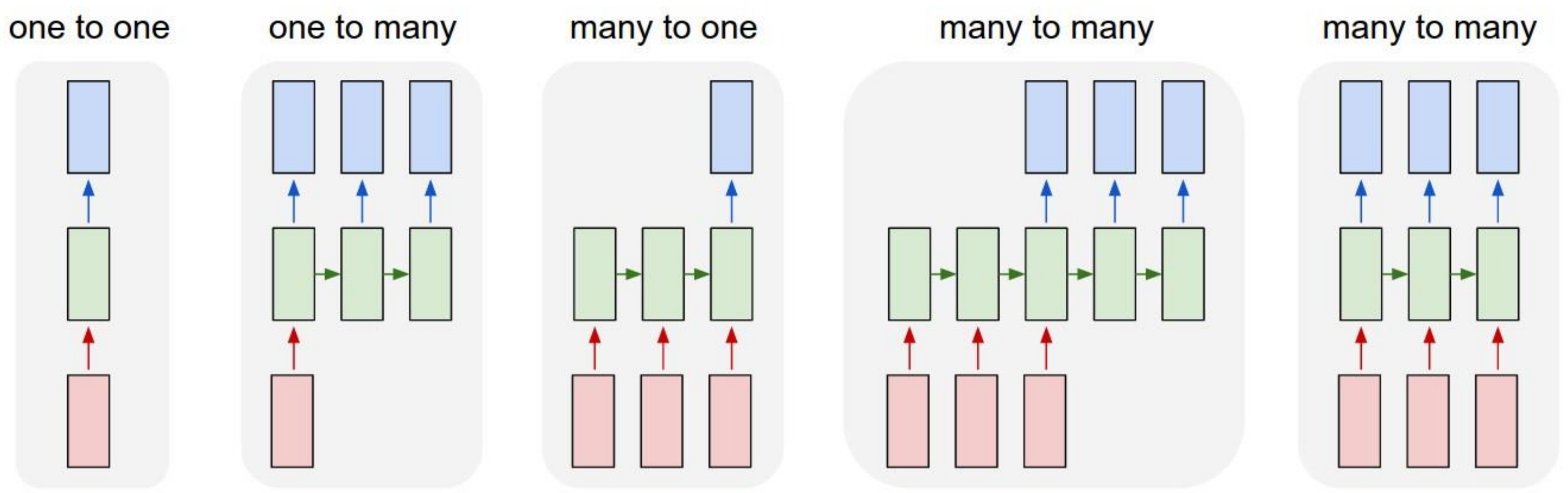
Neural Machine Translation

- Model

Each box is an LSTM or GRU cell.

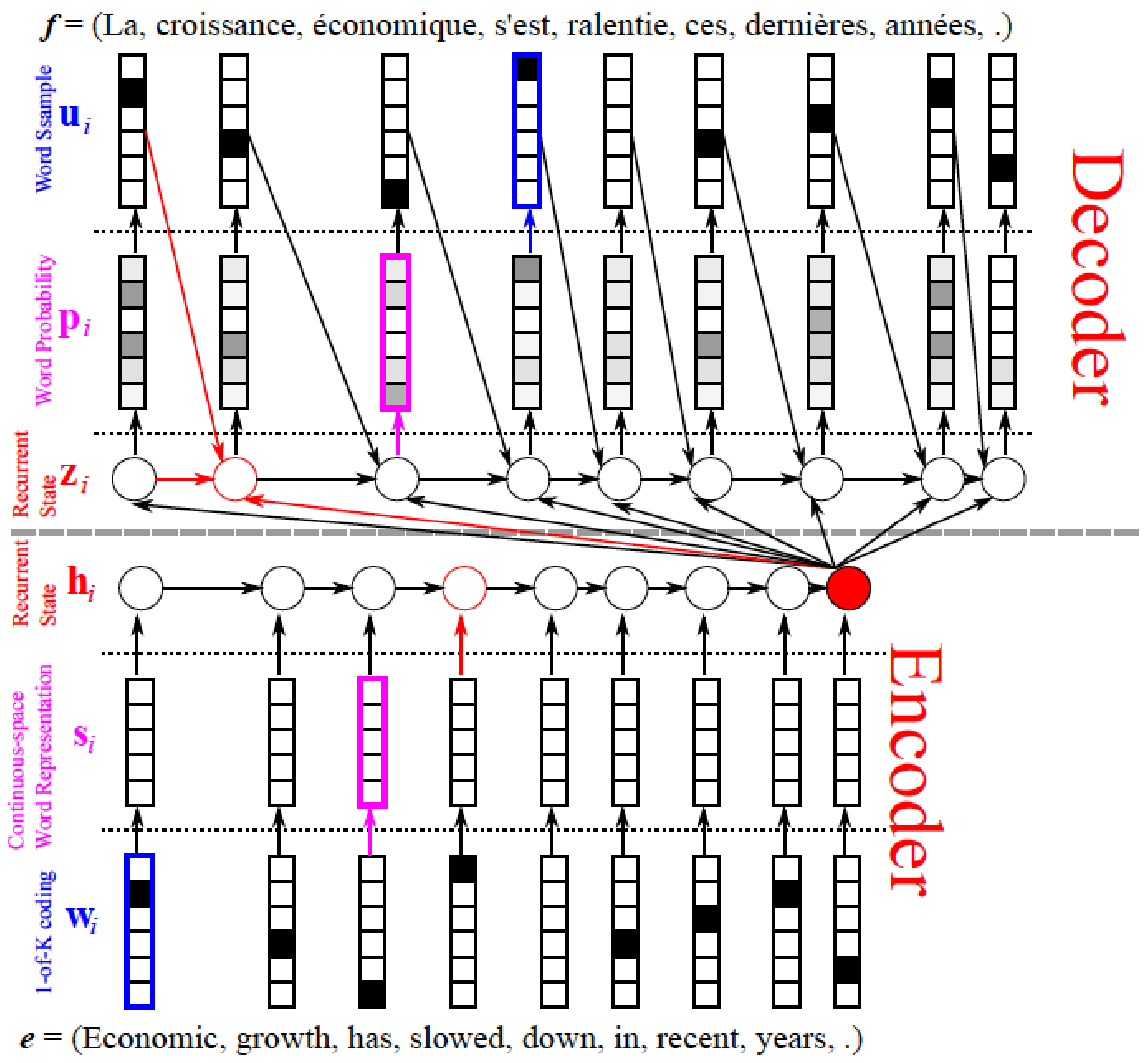


Sutskever et al. 2014



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

Neural Machine Translation

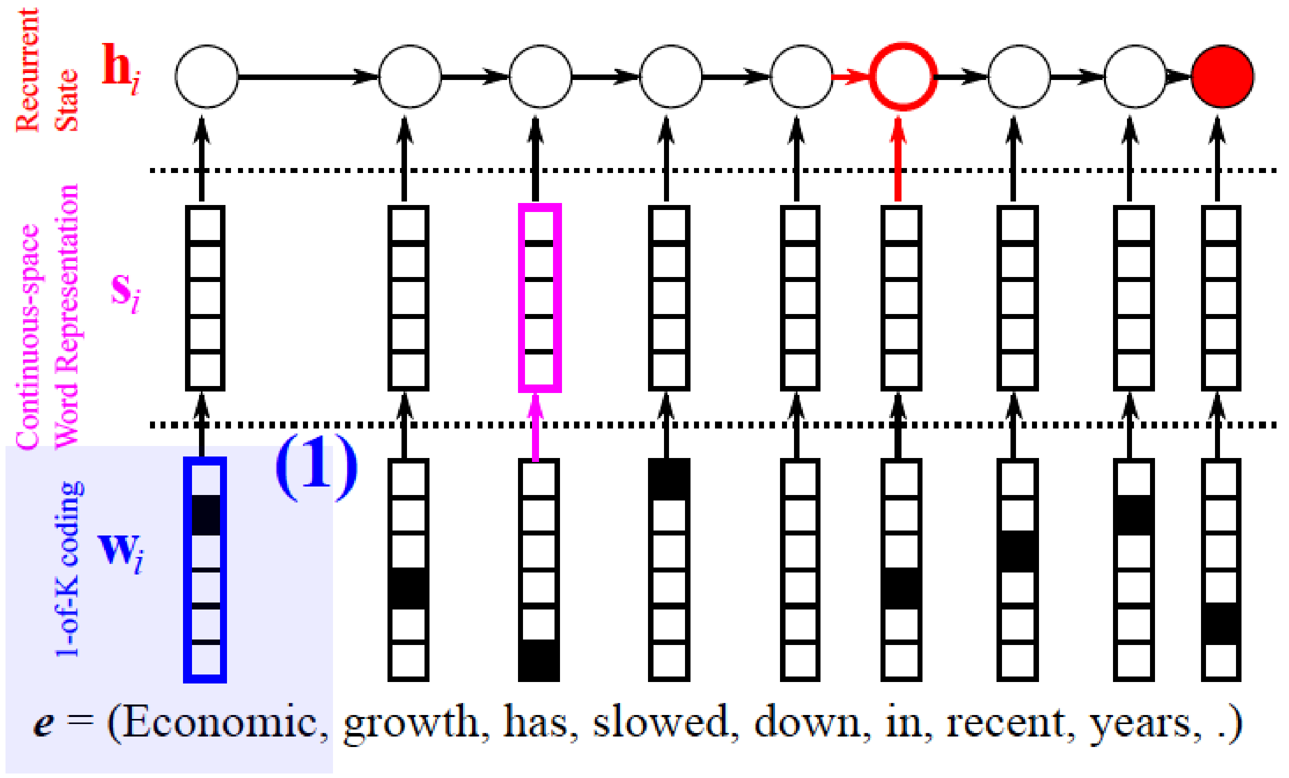


f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Word Probability
$\mathbf{p}_i$
Recurrent State
$\mathbf{z}_i$
Decoder
Recurrent State
$\mathbf{h}_i$
Continuous-space Word Representation
$\mathbf{s}_i$
1-of-K coding
$\mathbf{w}_i$
Encoder
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Cho: From Sequence Modeling to Translation

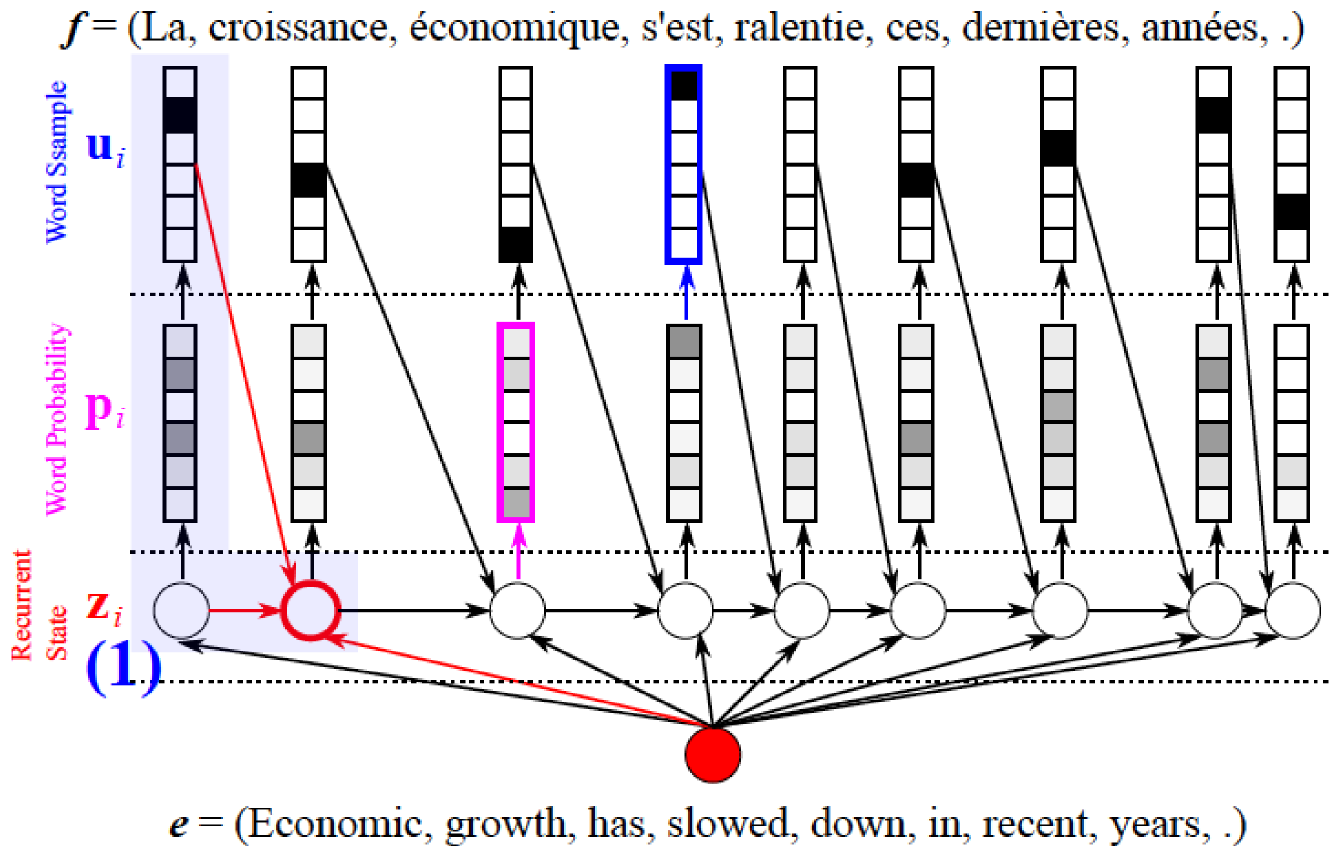
Neural Machine Translation

- Model- *encoder*


Recurrent State
$\mathbf{h}_i$
Continuous-space Word Representation
$\mathbf{s}_i$
(1)
1-of-K coding
$\mathbf{w}_i$
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Neural Machine Translation

- Model- *decoder*



f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Word Probability
$\mathbf{p}_i$
Recurrent State
$\mathbf{z}_i$
(1)
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Decoder in more detail

Given

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

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

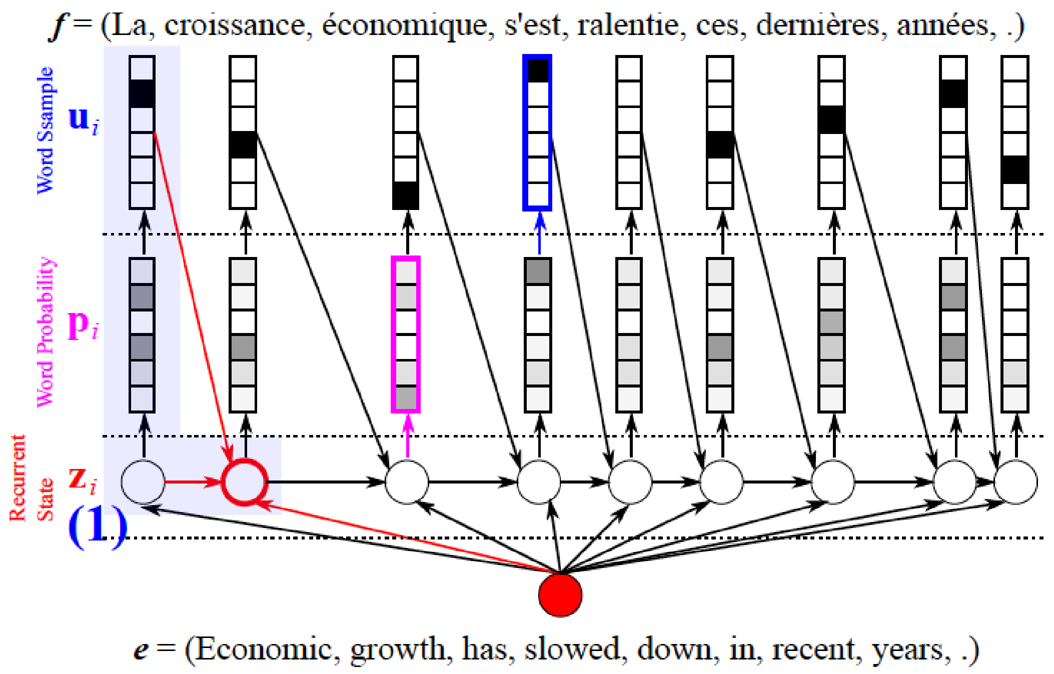
(iii) the previous state ($\mathbf{z}_{t-1}$)

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:

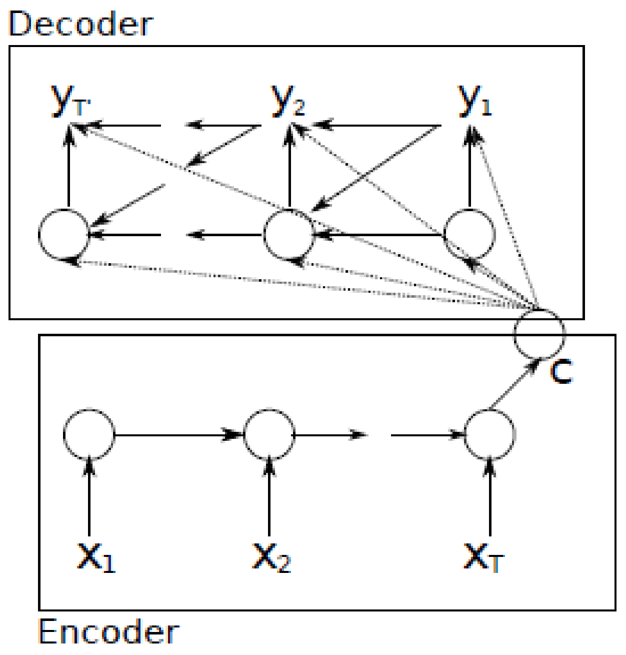
$$P(f_t \mid f_{t-1}, f_{t-2}, \dots, \mathbf{h}) = p(f_t \mid \mathbf{z}_t, f_{t-1}, \mathbf{h})$$


f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Word Probability
$\mathbf{p}_i$
Recurrent State
$\mathbf{z}_i$
(1)
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Encoder-decoder

- Jointly trained to maximize

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



NMT can be done at char-level too

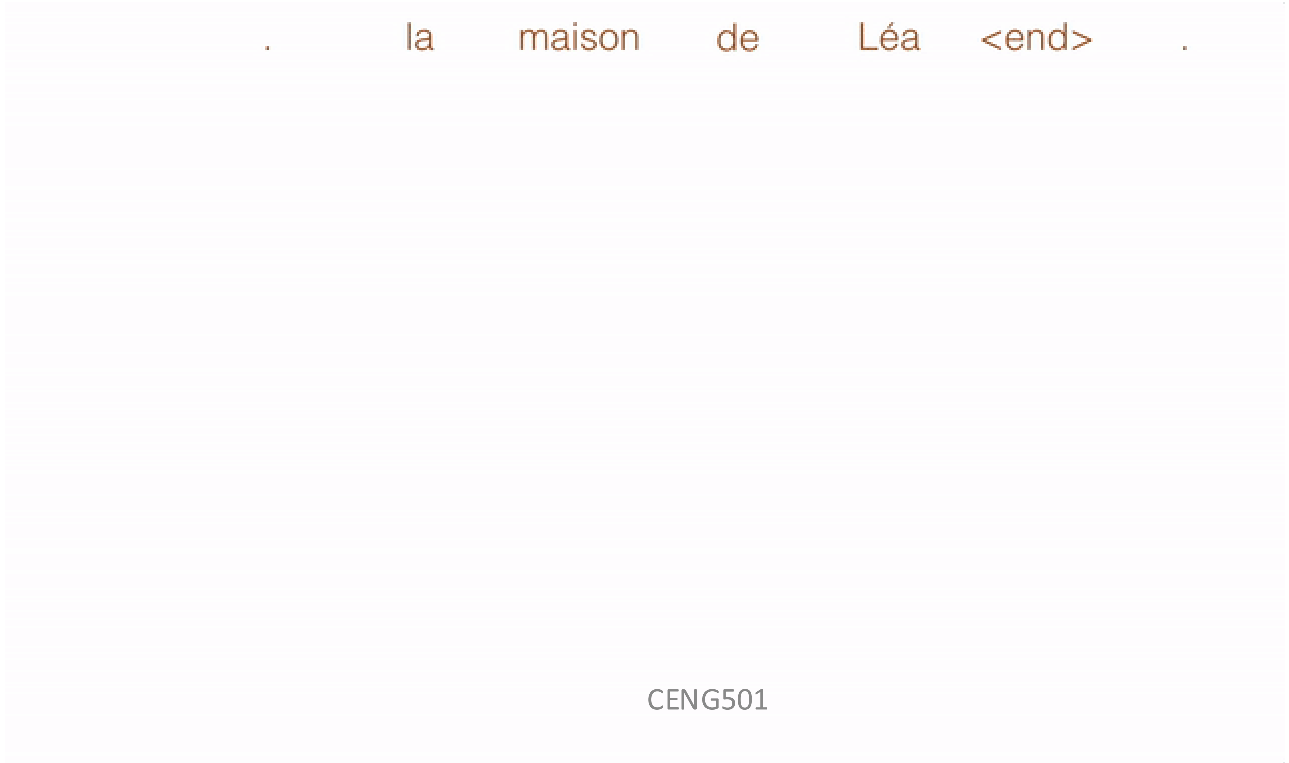
- http://arxiv.org/abs/1603.06147

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