ECBM E6040 Neural Networks and Deep Learning

Lecture #11: Convolutional Networks (cont'd), and Sequence Modeling: Recurrent and Recursive Nets

Aurel A. Lazar

Columbia University
Department of Electrical Engineering

April 12, 2016

Outline of Part I

2. Summary of the Previous Lecture
   - Topics Covered
   - Learning Objectives

Outline of Part II

3 Variants of the Basic Convolution Function

4 The Neuroscientific Basis for Convolutional Networks

Outline of Part III

5 Sequence Modeling: Recurrent and Recursive Nets
- Unfolding Computational Graphs
- Recurrent Neural Networks

Part I

Review of Previous Lecture

Topics Covered in Optimization for Training Deep Models

- Approximate Second-Order Methods
- Optimization Strategies an Meta-Algorithms

Topics Covered in Convolutional Networks

- The Convolution Operation
- Motivation
- Pooling

Learning Objectives in Optimization for Training Deep Models

- Newtons method for training large neural networks is limited by the significant computational burden it imposes as the inverse Hessian has to be computed at every training iteration.
- Conjugate gradients is a method to efficiently avoid the calculation of the inverse Hessian by iteratively descending conjugate directions.
- Most training algorithms are strongly affected by the choice of initialization.

Learning Objectives in Convolutional Networks

- Convolution leverages three important concepts that substantially improve a machine learning system: sparse connectivity, parameter sharing and equivariant representations.
- Convolution provides a means for working with inputs of variable size.

Part II

Convolutional Networks (cont'd)

Basic Convolution Function

Convolution neural networks consists of many applications of convolution in parallel. Convolution with a single kernel extracts a single feature at many locations. The input is typically a grid of vector-valued observations.

In a multilayer convolutional network, the input to the second layer is the output of the first layer, which usually has the output of many different convolutions at each position.

When working with images, we usually think of the input and output of the convolution as being 3-D tensors, with one index into the different channels and two indices into the spatial coordinates of each channel.

Basic Convolution Function (cont'd)

Assume we have a 4-D kernel tensor K with element $K_{i,j,k,l}$ giving the connection strength between a unit in channel i of the output and a unit in channel j of the input, with an offset of k rows and l columns between the output unit and the input unit.

Assume our input consists of observed data V with element $V_{i,j,k}$ giving the value of the input unit within channel i at row j and column k. Assume our output consists of Z with the same format as V. If Z is produced by convolving K across V without flippping K, then

$$Z_{ijk} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m.n},$$

where the summation over l, m and n is over all values for which the tensor indexing operations inside the summation is valid.

Basic Convolution Function (cont'd)

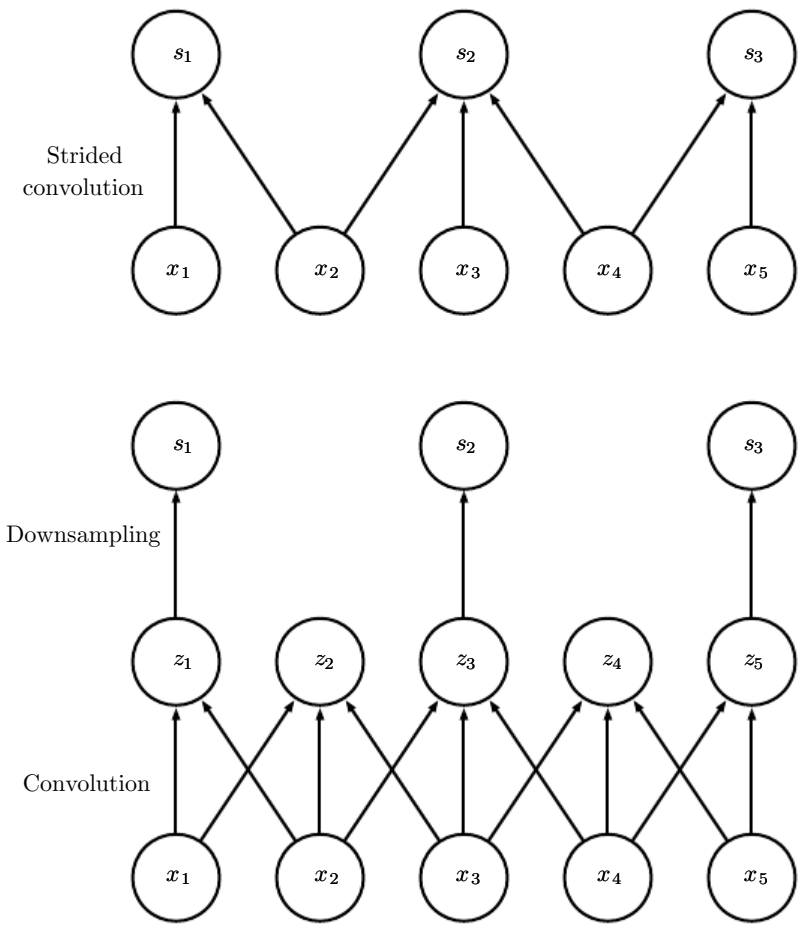
We may want to skip over some positions of the kernel in order to reduce the computational cost (at the expense of not extracting our features as finely). We can think of this as down sampling the output of the full convolution function. If we want to sample only every s pixels in each direction in the output, then we can define a down sampled convolution function c such that

$$Z_{ijk} = c(K, V, s)_{ijk} = \sum_{l,m,n} [V_{l,(j-1)s+m,(k-1)s+n} K_{ilmn}].$$

We refer to s as the stride of this down sampled convolution. It is also possible to define a separate stride for each direction of motion.

Basic Convolution Function (cont'd)

Example: Convolution with a Stride



(Top) Convolution with a stride length of two implemented in a single operation.
(Bottom) Convolution with a stride greater than one pixel is mathematically equivalent to convolution with unit stride followed by downsampling.
The two-step approach involving downsampling is computationally wasteful, because it computes many values that are then simply discarded.

Basic Convolution Function (cont'd)

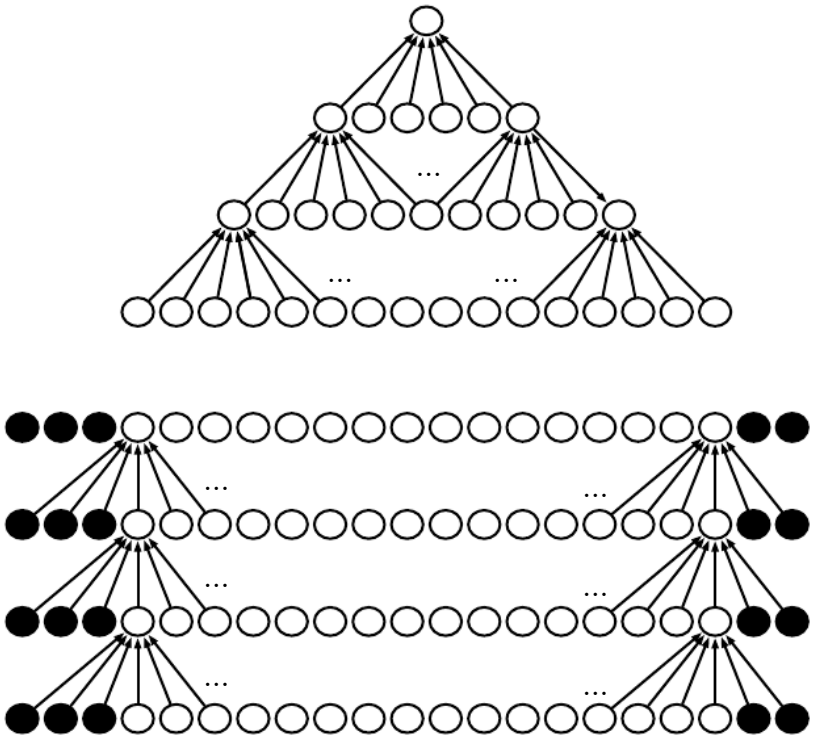
Zero-Padding

One essential feature of any convolutional network implementation is the ability to implicitly zero-pad the input V in order to make it wider. Zero padding the input allows us to control the kernel width and the size of the output independently.
Three special cases of the zero-padding setting are worth mentioning:

- the extreme case in which no zero-padding is used whatsoever, and the convolution kernel is only allowed to visit positions where the entire kernel is contained entirely within the image.
- the case when just enough zero-padding is added to keep the size of the output equal to the size of the input.
- the case in which enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m + k - 1$.

Basic Convolution Function (cont'd)

Example: The Effect of Zero Padding on Network Size



(Top) No implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, there are only three convolutional layers available. (Bottom) By adding five implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

Basic Convolution Function (cont'd)

Unshared Convolution

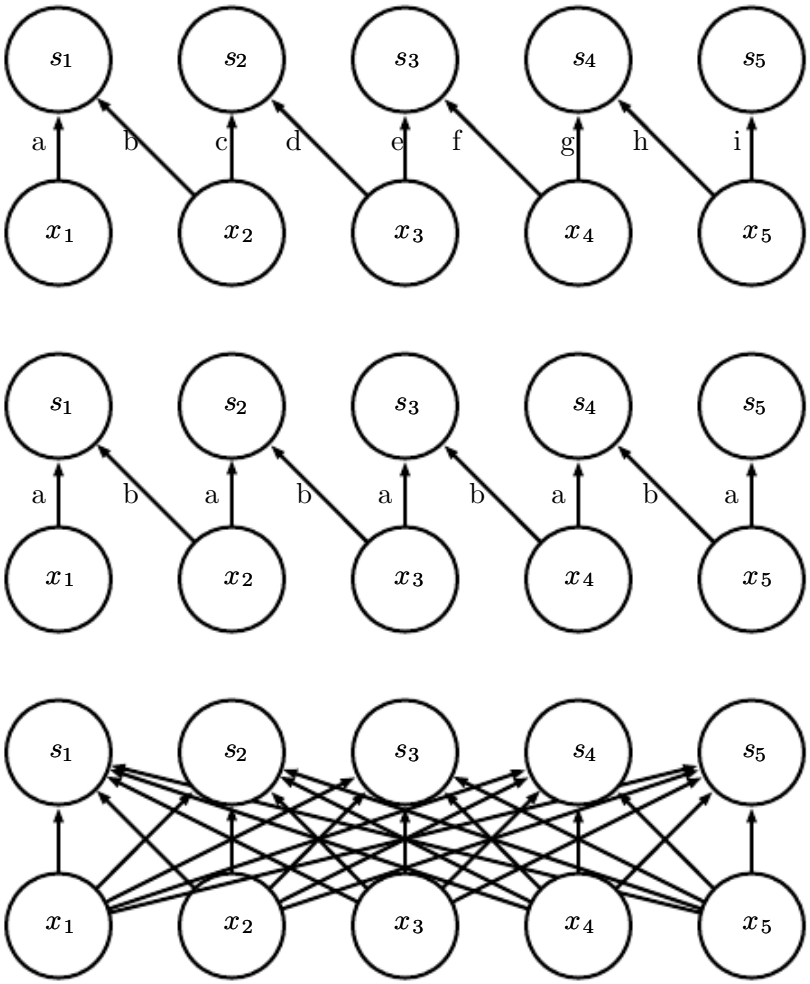
In some cases, we do not actually want to use convolution, but rather locally connected layers. In this case, the adjacency matrix in the graph of our MLP is the same, but every connection has its own weight, specified by a 6-D tensor W.
The indices into W are respectively: i, the output channel, j, the output row, k, the output column, l, the input channel, m, the row offset within the input, and n, the column offset within the input. The linear part of a locally connected layer is then given by

$$Z_{i,j,k} = \sum_{l,m,n} [V_{l,j+m-1,k+n-1} W_{i,j,k,l,m,n}].$$

This is sometimes also called unshared convolution, because it is a similar operation to discrete convolution with a small kernel, but without sharing parameters across locations.

Basic Convolution Function (cont'd)

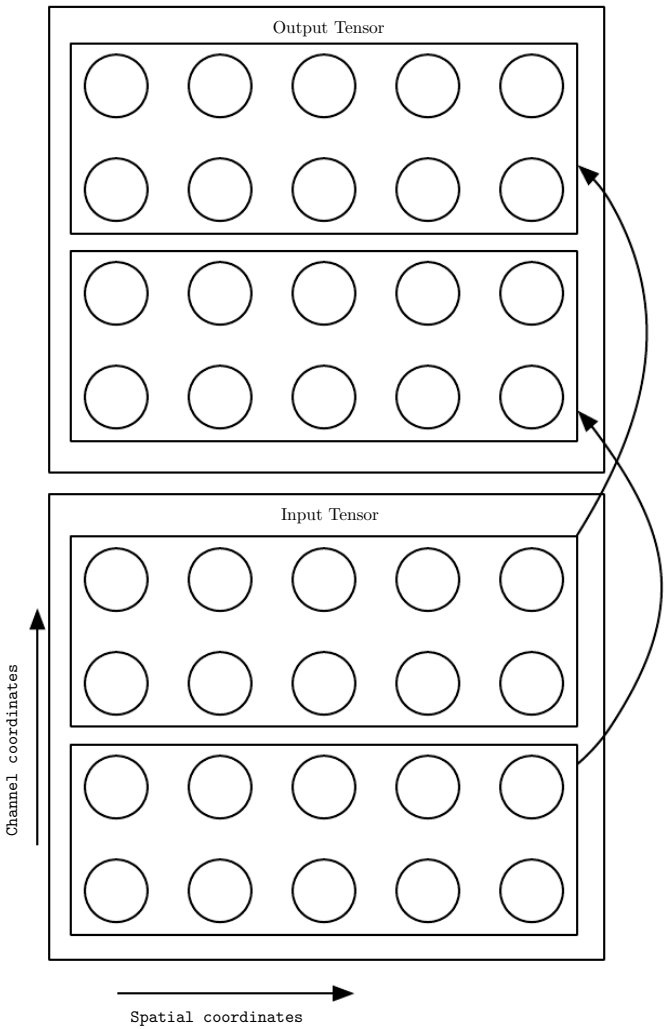
Example: Unshared Convolution



(Top) A locally connected layer with a patch size of two pixels. (Center) A convolutional layer with a kernel width of two pixels. The locally connected layer has no parameter sharing. The conv layer uses the same two weights repeatedly across the entire input, as indicated by the repetition of the letters labeling each edge. (Bottom) A fully connected layer resembles a locally connected layer in the sense that each edge has its own parameter. However, it does not have the restricted connectivity of the locally connected layer

Basic Convolution Function (cont'd)

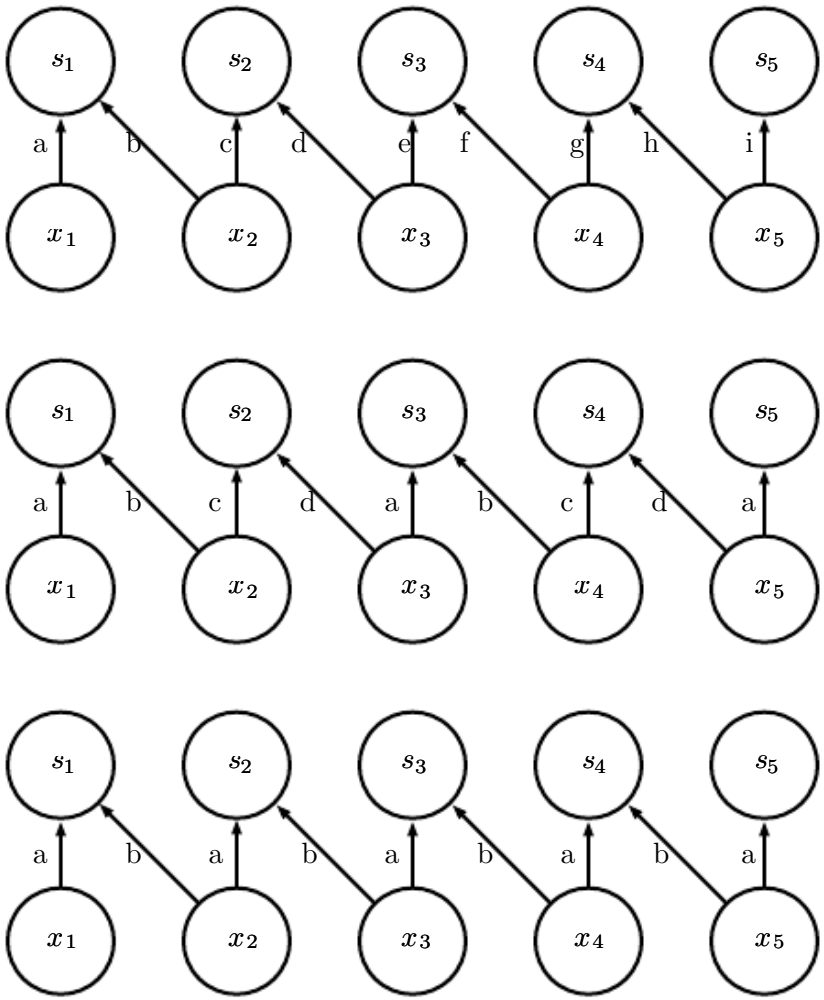
Example: Restricting Connetivity



A convolutional network with the first two output channels connected to only the first two input channels, and the second two output channels connected to only the second two input channels.

Basic Convolution Function (cont'd)

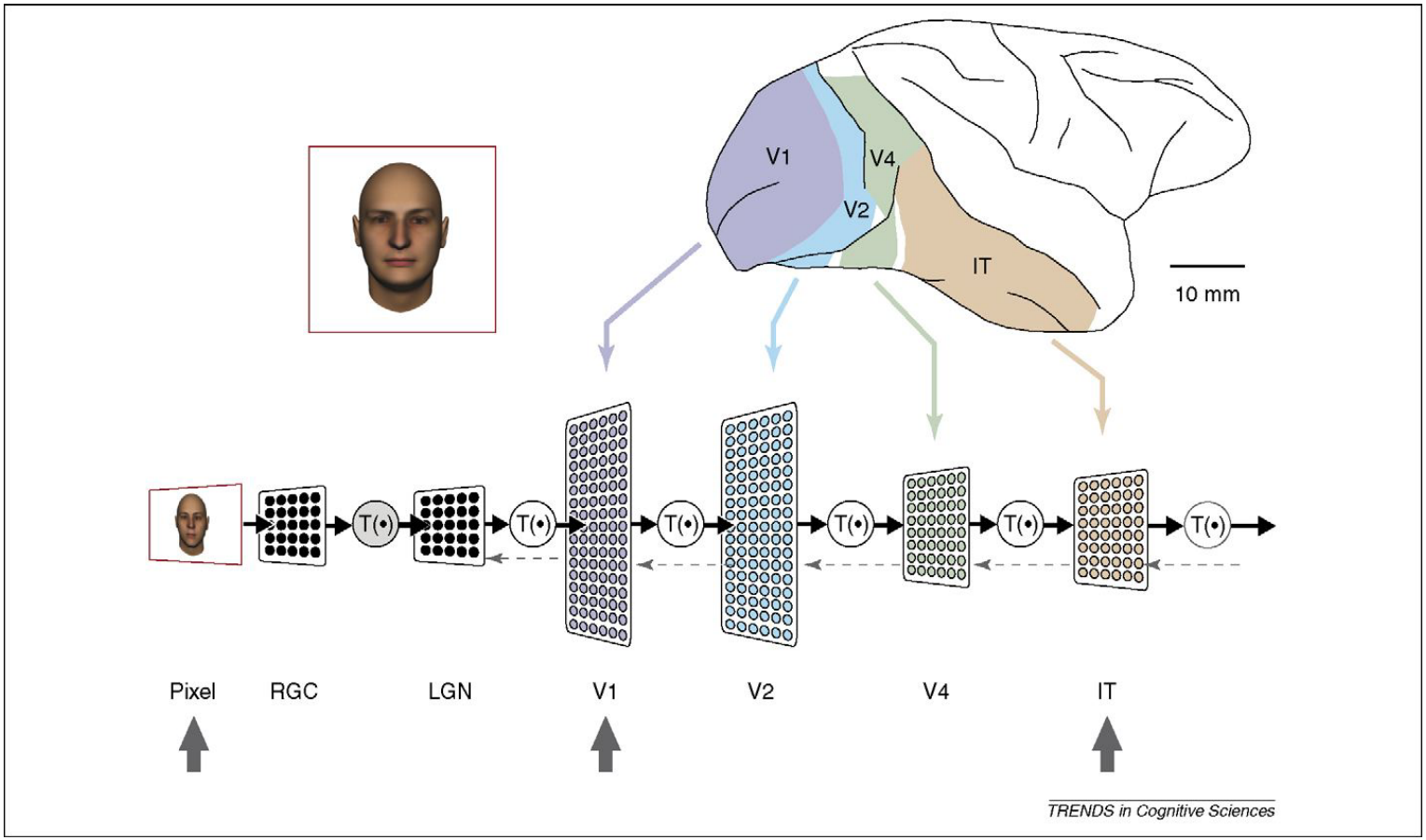
Example: Tiled Convolution



(Top) A locally connected layer has no sharing at all. Each connection has its own weight by labeling each connection with a unique letter. (Center) Tiled convolution with a set of $t = 2$ different kernels. One of these kernels has edges labeled a and b, while the other has edges labeled c and d. If two output units are separated by a multiple of t steps, then they share parameters. (Bottom) Traditional convolution is equivalent to tiled convolution with $t = 1$. There is only one kernel and it is applied everywhere with weights labeled a and b everywhere.

The Brain as an Information Processor

The Basic Building Blocks of Visual Information Processing



RGC: Retinal Ganglion Cells; LGN: Lateral Geniculate Nucleus; Primary Visual Cortex: V1; Visual Areas V2 and V4; IT: Inferior Temporal.

Basics of the Early Visual System

Neurophysiologists David Hubel and Torsten Wiesel collaborated for several years in the late 50s and early 60s to determine many of the most basic facts about how the mammalian early visual system works.

Images are formed by light arriving in the eye and stimulating the alertretina, the light-sensitive tissue in the back of the eye.
The neurons in the retina perform preprocessing of the image and the RGCs represent the pre-processed information in the spike domain.
The image then passes through the optic nerve and a brain region called the lateral geniculate nucleus (LGN).
LGN neurons project their axons into the primary visual cortex V1 located at the back of the head.

CNNs Inspired by Basic Elements of the Visual System

A convolutional network layer is designed to capture three properties of the primary visual cortex V1:

- V1 is arranged in a spatial map (retinotopy); its 2-D structure mirrors the structure of the image on the retina.

  Convolutional networks capture this property by having their features defined in terms of 2-D maps.

- V1 contains many simple cells; a simple cell's activity can be characterized by a linear function of the image in a small, spatially localized receptive field.

  The detector units of a convolutional network are designed to emulate the local properties of simple cells.

- V1 also contains many complex cells; complex cells are invariant to small shifts in the position of the feature.

  CNN pooling units attempt to address small shift invariance. Complex cells are also invariant to some changes in lighting that have inspired some of the cross-channel pooling strategies in CNNs, such as maxout units.

CNNs Inspired by Basic Elements ... (dreaming cont'd)

Grandmother Cells

As we repeatedly apply basic strategy of detection followed by pooling and move deeper into the brain, we eventually find cells that respond to some specific concept and are invariant to many transformations of the input. These cells have been nicknamed grandmother cells.

The hypothesis is that a person could have a neuron that activates when seeing an image of their grandmother, regardless of whether she appears in the left or right side of the image, whether the image is a close-up of her face or zoomed out shot of her entire body, whether she is brightly lit, or in shadow, etc.

These grandmother cells have been shown to actually exist in the human brain, in a region called the medial temporal lobe. These medial temporal lobe neurons/circuits are somewhat more general than modern convolutional networks, which would not automatically generalize to identifying a person or object when reading its name.

Differences between CNNs and Mammalian Visual Systems

There are many differences between convolutional networks and the mammalian vision system (too many to enumerate):

- The retina is mostly very low resolution except for a tiny patch called the fovea. Though we feel as if we can see an entire scene in high resolution, this is an illusion created by our visual cortex, as it stitches together several glimpses of small areas.

  Most CNNs receive large full resolution photographs as input.

- The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts.

  Convolutional networks are purely visual.

- The human visual system does much more than just recognize objects. It is able to understand entire scenes including many objects and relationships between objects, and processes rich 3-D geometric information needed for our bodies to interface with the world.

  CNNs have been applied to some of these problems but these applications are in their infancy.

Differences between CNNs and MVSs (cont'd)

Moreover

- V1 is heavily impacted by feedback from higher brain centers. Feedback has been explored extensively in neural network models but has not yet been shown to offer a compelling improvement.
- While feed-forward IT firing rates capture much of the same information as convolutional network features, it's not clear how similar the intermediate computations are. The brain probably uses very different activation and pooling functions.
- An individual neuron's activation probably is not well characterized by a single linear filter response. Our cartoon picture of simple cells and complex cells might create a nonexistent distinction; simple cells and complex cells might both be the same kind of cell but with their "parameters" enabling a continuum of behaviors ranging from what we call "simple" to what we call "complex".

Neuroscience and Learning in CNNs

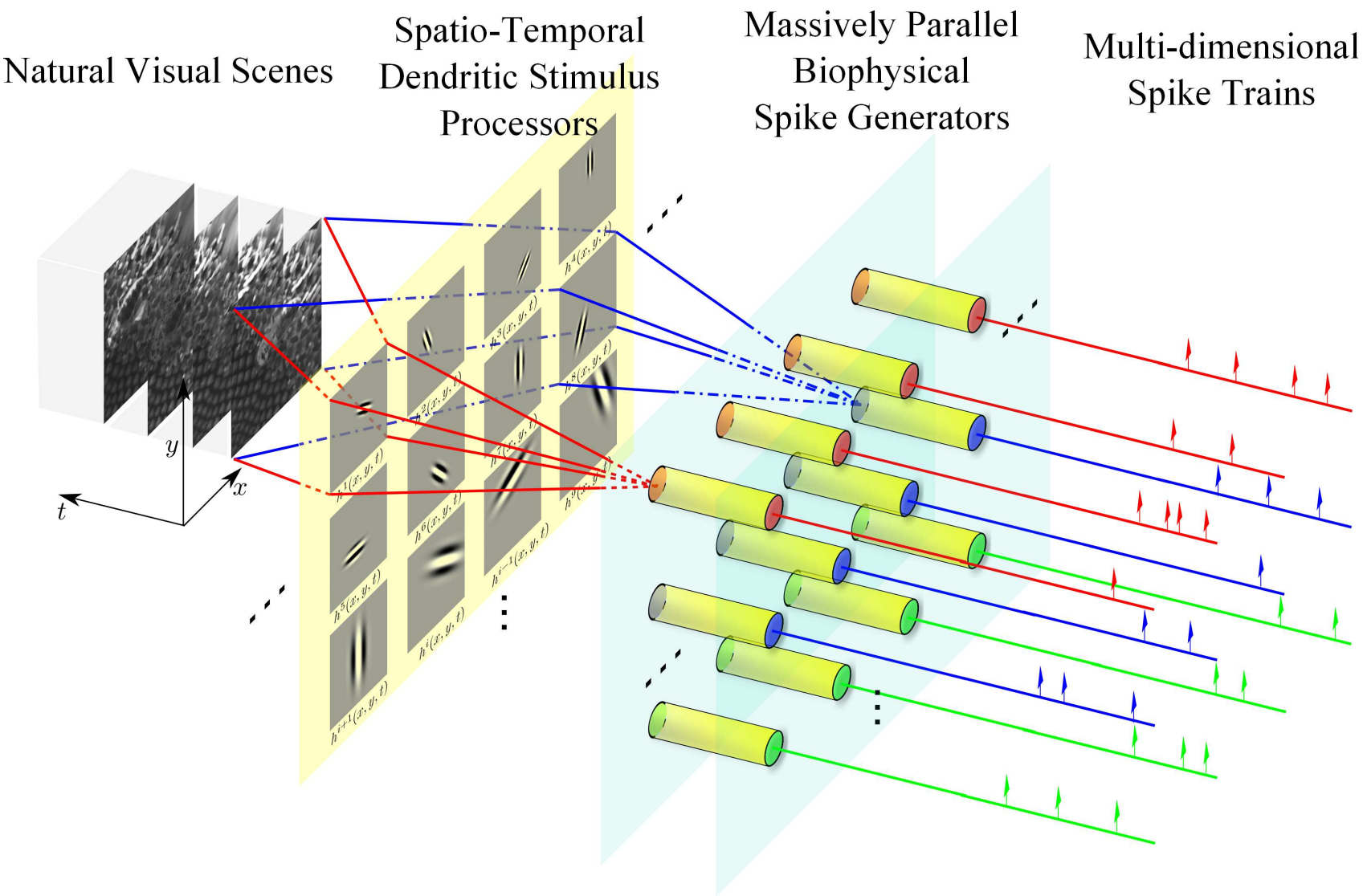
Neuroscience has told us relatively little about how to train convolutional networks.

Back-propagation to train 1-D convolutional networks applied to time series was first introduced in 1988 (Hinton).

Back-propagation applied to these models was not inspired by any neuroscientific observation and is considered to be biologically implausible.

In 1989 (LeCun) developed the modern convolutional network by applying the same training algorithm to 2-D convolution applied to images.

A Model of Neural Encoding in the Primary Visual Cortex



Gabor Filters as Convolution Operators

We can think of an image as being a function of 2-D coordinates, $I(x, y)$. Likewise, we can think of a simple cell as sampling the image at a set of locations, defined by a set of x coordinates $\mathbb{X}$ and a set of y coordinates, $\mathbb{Y}$, and applying weights that are also a function of the location, $w(x, y)$. From this point of view, the response of a simple cell to an image is given by

$$s(I) = \sum_{x \in \mathbb{X}} \sum_{y \in \mathbb{Y}} w(x, y) I(x, y).$$

Here $w(x, y)$ takes the form of a Gabor function:

$$w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp(-\beta_x x^{'2} - -\beta_y y^{'2}) \cos(f x^{'} + \phi),$$

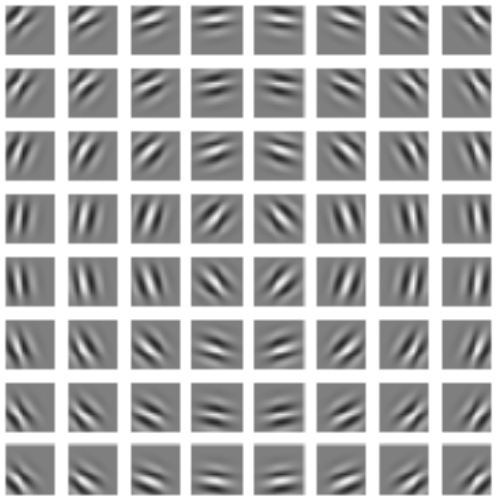
where

$$x^{'} = (x - x_0) \cos(\tau) + (y - y_0) \sin(\tau)$$

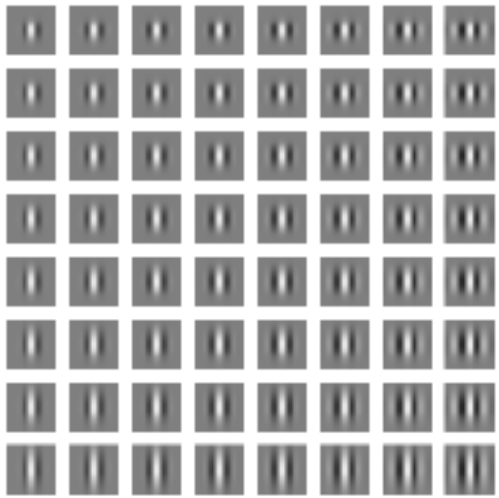
and

$$y^{'} = -(x - y_0) \sin(\tau) + (y - y_0) \cos(\tau).$$

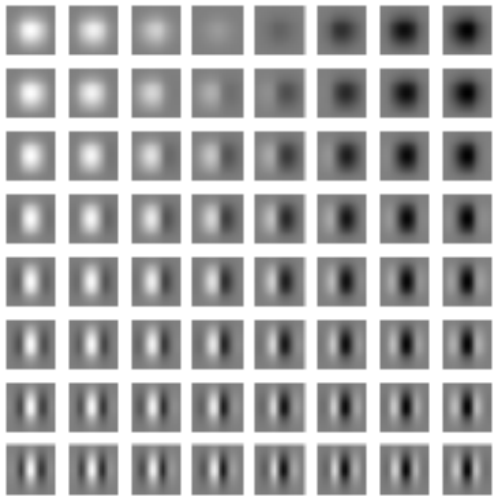
Gabor Filters/Kernels/Functions



Gabor functions with different parameter values (x_0, y_0) and τ.

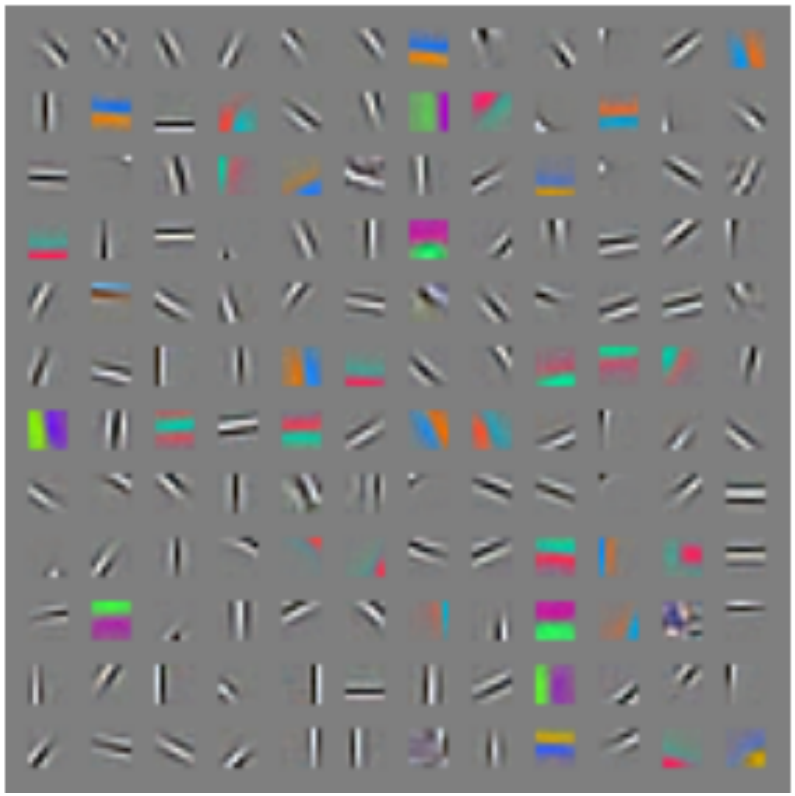


Gabor functions with different Gaussian scale parameters β_x and β_y.

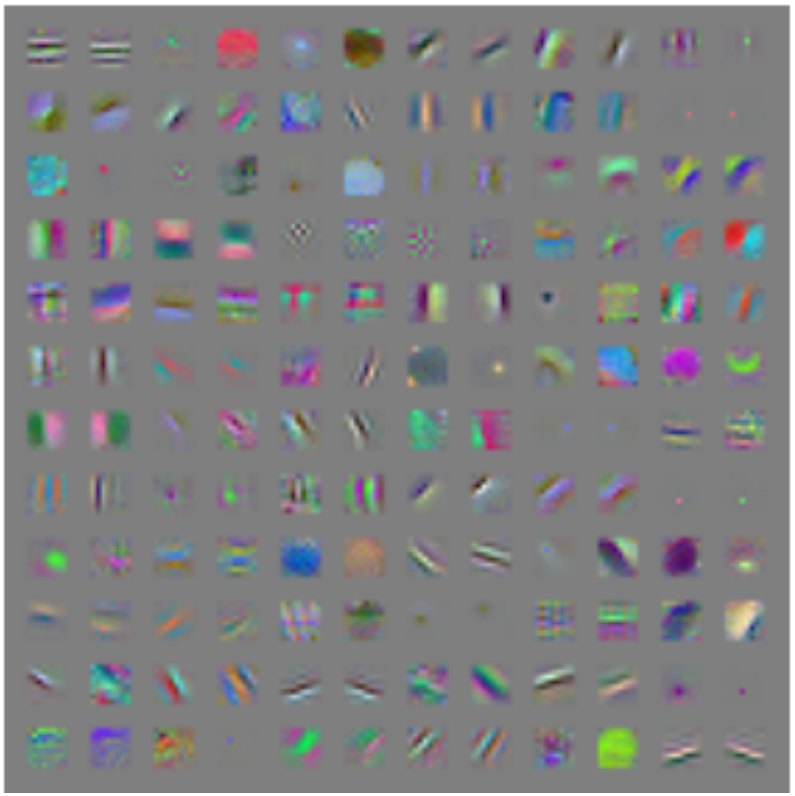


Gabor functions with different sinusoid parameters f and ϕ.

Learning Convolution Kernels in the First Layer



Weights learned by an unsupervised learning algorithm (spike and slab sparse coding) applied to small image patches.



Convolution kernels learned by the first layer of a fully supervised convolutional maxout network. Neighboring pairs of filters drive the same maxout unit.

Part III

Sequence Modeling: Recurrent and Recursive Nets

Recurrent Neural Networks - Overview

Recall that a convolutional network is a neural network that

- is specialized for processing a grid of values X such as an image,
- can readily scale to images with large width and height, and some convolutional networks can process images of variable size.

Recurrent neural networks (RNNs) are a family of neural networks for processing sequential data that:

- specialized for processing a sequence of values $(\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3, ..., \boldsymbol{x}^{\tau-1}, \boldsymbol{x}^{\tau})$.
- can scale to much longer sequences than would be practical for networks without sequence-based specialization. Most recurrent networks can also process sequences of variable length.

Unfolding a Computational Graph

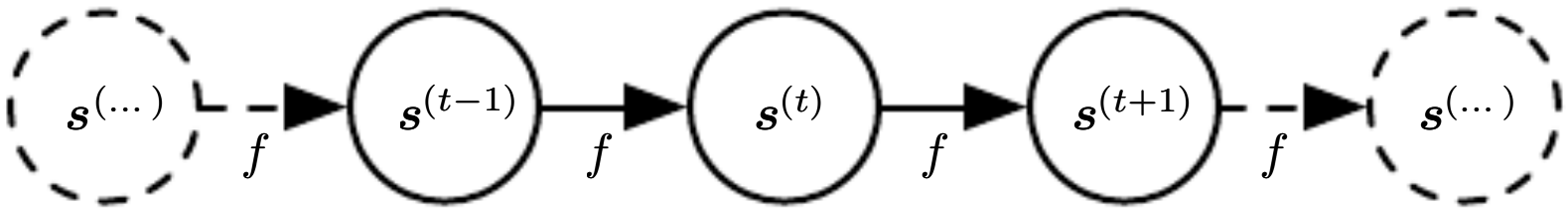
We consider the classical form of a dynamical system

$$\boldsymbol{s}^t = f(\boldsymbol{s}^{t-1}; \boldsymbol{\theta}),$$

where $\boldsymbol{s}^t$ is called the state of the system at time t. For $t = 3$ we have

$$\boldsymbol{s}^3 = f(\boldsymbol{s}^2; \boldsymbol{\theta}) = f(f(\boldsymbol{s}^1; \boldsymbol{\theta}); \boldsymbol{\theta}) = f(f(f(\boldsymbol{s}^0; \boldsymbol{\theta}); \boldsymbol{\theta}); \boldsymbol{\theta}).$$

Unfolding this equation by repeatedly applying the definition in this way yields an expression that does not involve recurrence and can be represented by a traditional directed acyclic computational graph:



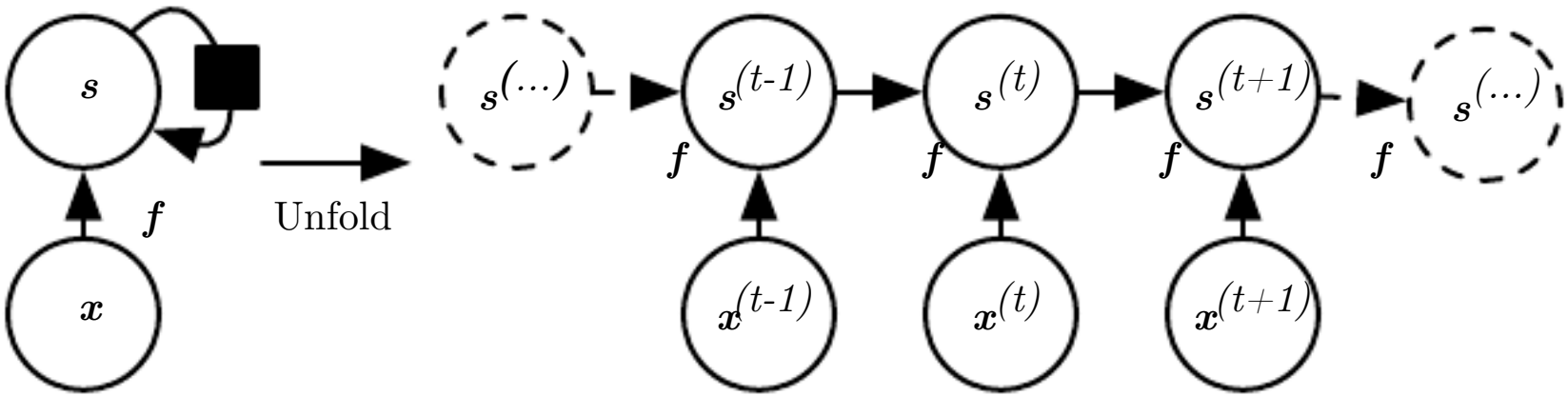
Unfolding a Computational Graph with an External Signal

Dynamical System Driven by an External Signal

Consider a dynamical system driven by an external signal $\boldsymbol{x}^t$,

$$\boldsymbol{s}^t = f(\boldsymbol{s}^{t-1}, \boldsymbol{x}^t; \boldsymbol{\theta}).$$

Note that the state contains information about the whole past sequence.



(Left) Circuit diagram. The black square indicates a delay of 1 time step. (Right) Same network as an unfolded computational graph; each node is now associated with one particular time instance

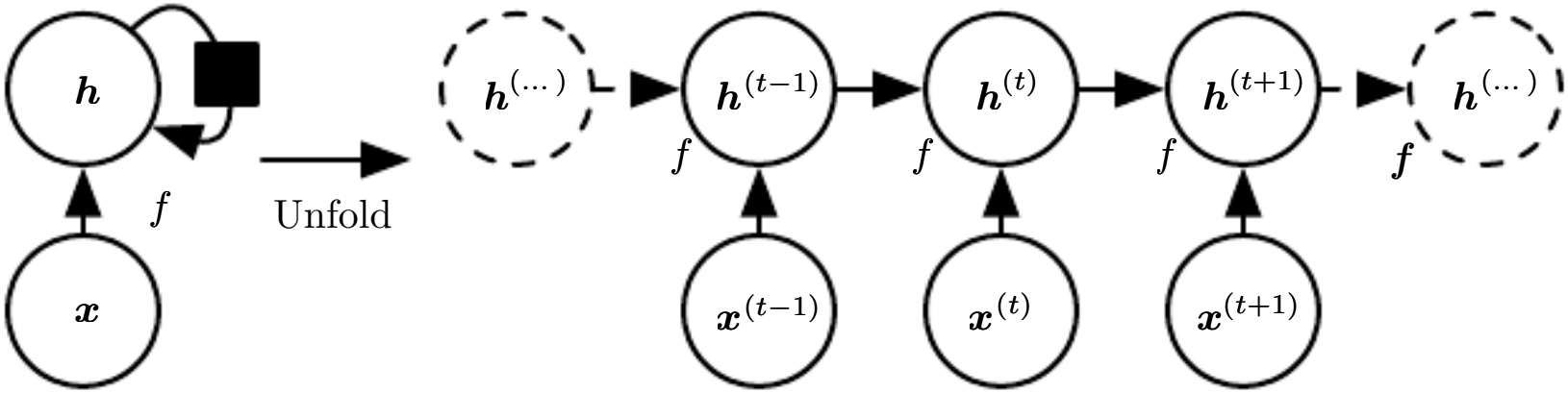
Unfolding the Computational Graph

Notation for Hidden Units

To indicate that the state is the hidden units of the network, we now use the variable $\boldsymbol{h}$ to represent the state:

$$\boldsymbol{h}^t = f(\boldsymbol{h}^{t-1}, \boldsymbol{x}^t; \boldsymbol{\theta}).$$

Typical RNNs will add extra architectural features such as output layers that read information out of the state $\boldsymbol{h}$ to make predictions.



A recurrent network with no outputs. This recurrent network just processes information from the input $\boldsymbol{x}$ by incorporating it into the state $\boldsymbol{h}$ that is passed forward through time.

Representing Unfolded Recurrence

We can represent the unfolded recurrence after t steps with a function g^t:

$$\boldsymbol{h}^t = g^t(\boldsymbol{x}^t, \boldsymbol{x}^{t-1}, \boldsymbol{x}^{t-2}, ..., \boldsymbol{x}^2, \boldsymbol{x}^1) = f(\boldsymbol{h}^{t-1}, \boldsymbol{x}^t; \boldsymbol{\theta})$$

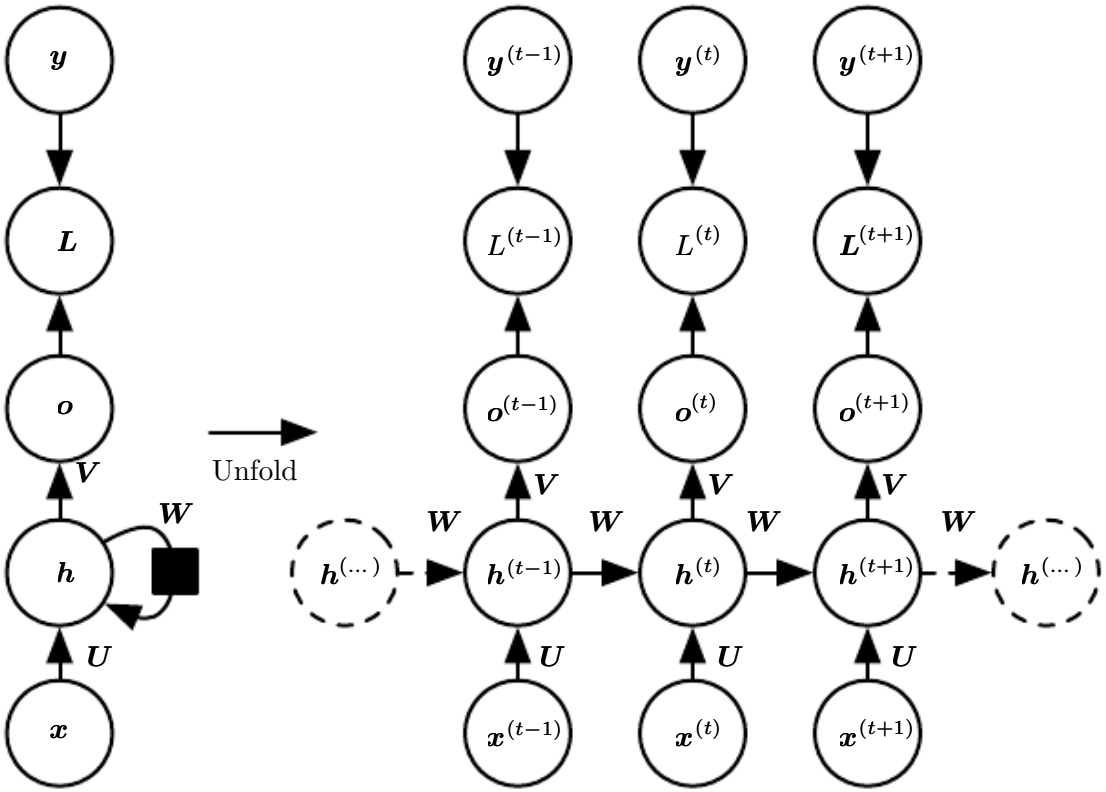
The function g^t takes as input the past sequence in its entirety $(\boldsymbol{x}^t, \boldsymbol{x}^{t-1}, \boldsymbol{x}^{t-2}, ..., \boldsymbol{x}^2, \boldsymbol{x}^1)$ and produces the current state. The unfolded recurrent structure allows us to factorize g^t into repeated application of the function f.

The unfolding process introduces two major advantages:

- regardless of the sequence length, the learned model always has the same input size, because it is specified in terms of transition from one state to another state, rather than specified in terms of a variable-length history of states;
- it is possible to use the same transition function f with the same parameters at every time step.

Building Universal RNNs

Any Function Computable by a Turing Machine Can Be Computed by a Finte Size RNN



The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$.

Specifying an RNN

To instantiate the computational graph we assume here

- a hyperbolic tangent activation function,
- the output $\boldsymbol{o}$ as giving the unnormalized log probabilities of each possible value of a discrete variable.
- the softmax operation as a post-processing step to obtain a vector $\hat{\boldsymbol{y}}$ of normalized probabilities over the output.

Forward propagation begins with the initial state $\boldsymbol{h}^0$:

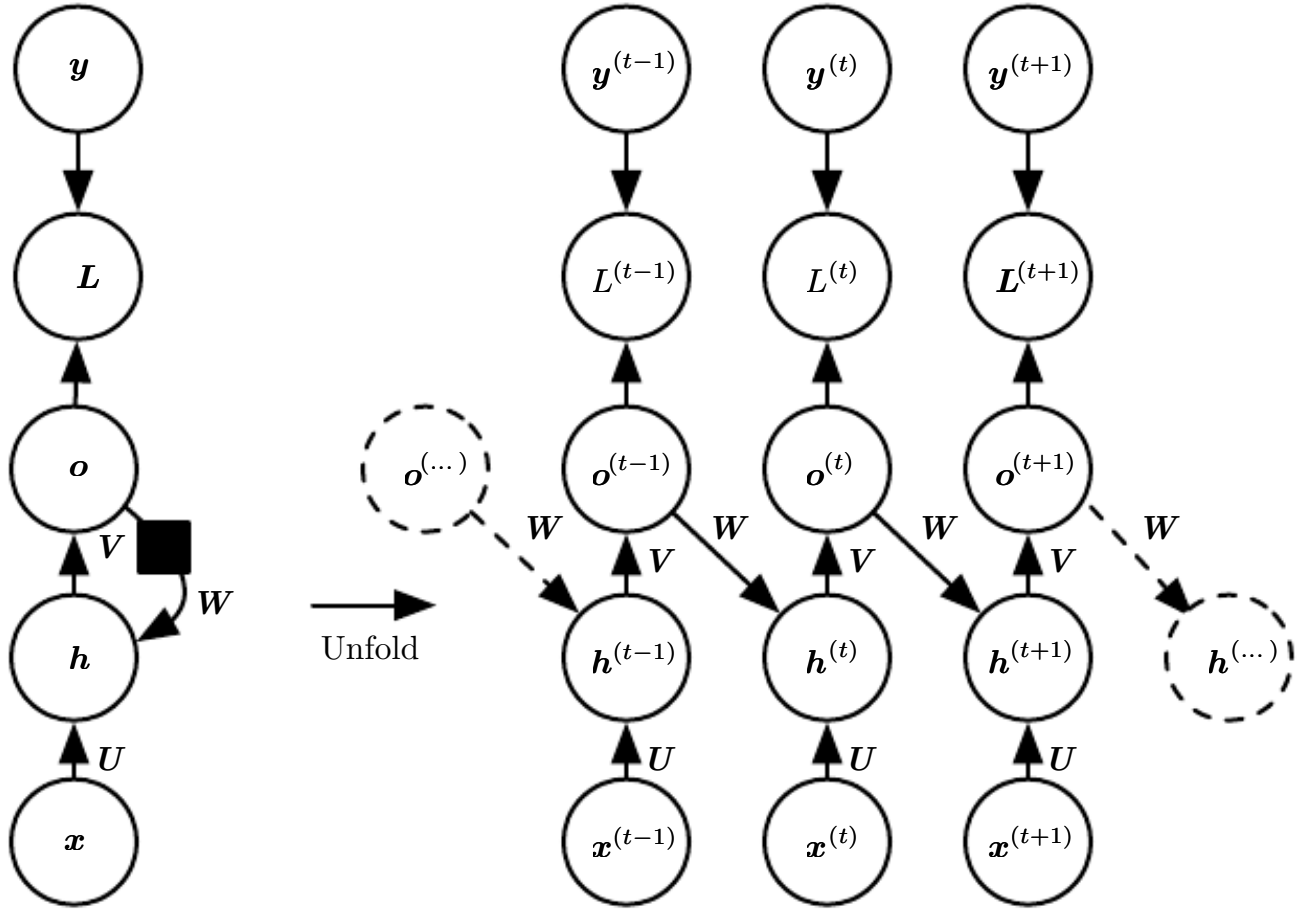
$$\begin{aligned}
\boldsymbol{a}^t &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{t-1} + \boldsymbol{U}\boldsymbol{x}^t \\
\boldsymbol{h}^t &= \tanh(\boldsymbol{a}^t) \\
\boldsymbol{o}^t &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^t \\
\hat{\boldsymbol{y}}^t &= \text{softmax}(\boldsymbol{o}^t).
\end{aligned}$$

Design Patterns for Building RNNs

Examples of important design patterns for recurrent neural networks include the following:

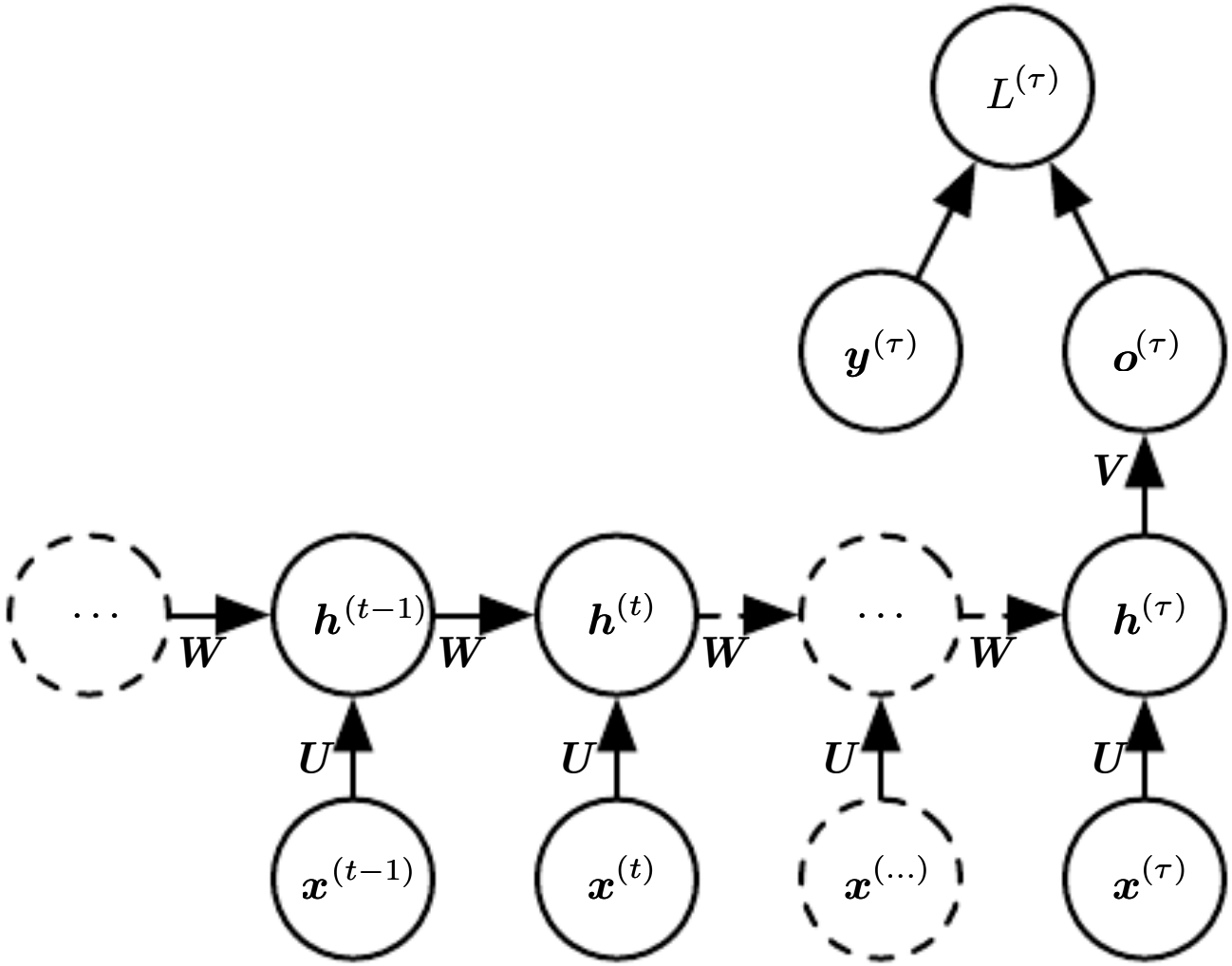
- RNNs that produce an output at each time step and have recurrent connections between hidden units (already discussed);
- RNNs that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step;
- RNNs with recurrent connections between hidden units, that read an entire sequence and then produce a single output.

Design Patterns for Building RNNs (cont'd)



An RNN whose only recurrence is the feedback connection from the output to the hidden layer.

Design Patterns for Building RNNs (cont'd)



Time-unfolded recurrent neural network with a single output at the end of the sequence.