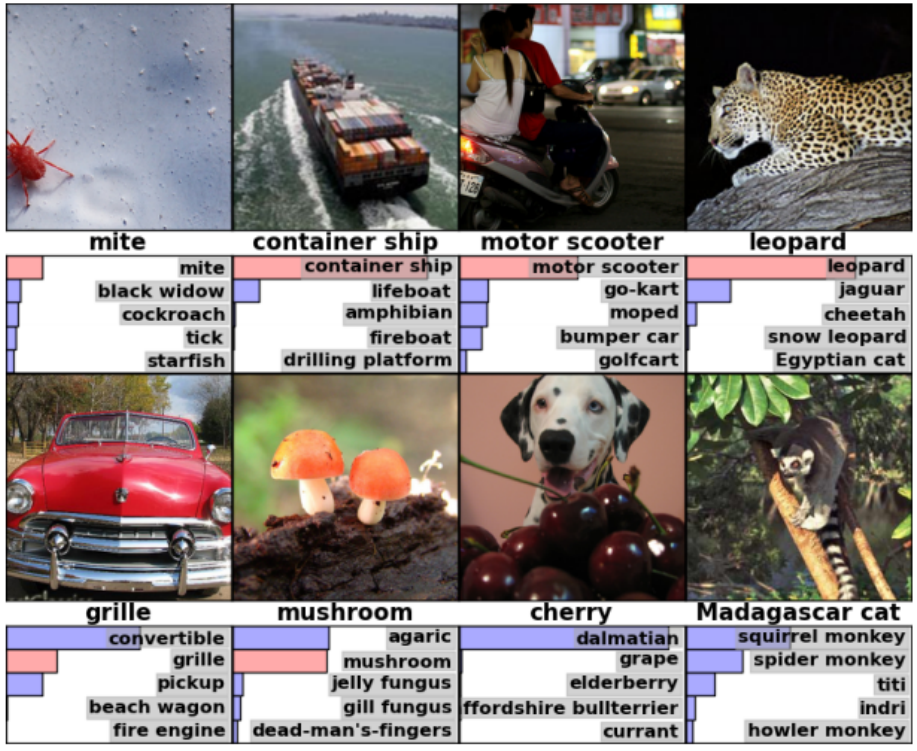
LUND UNIVERSITY

FMAN-45: Machine Learning

Lecture 8: CNNs and RNNs

David Nilsson, Henning Petzka, Cristian Sminchisescu

Image Classification

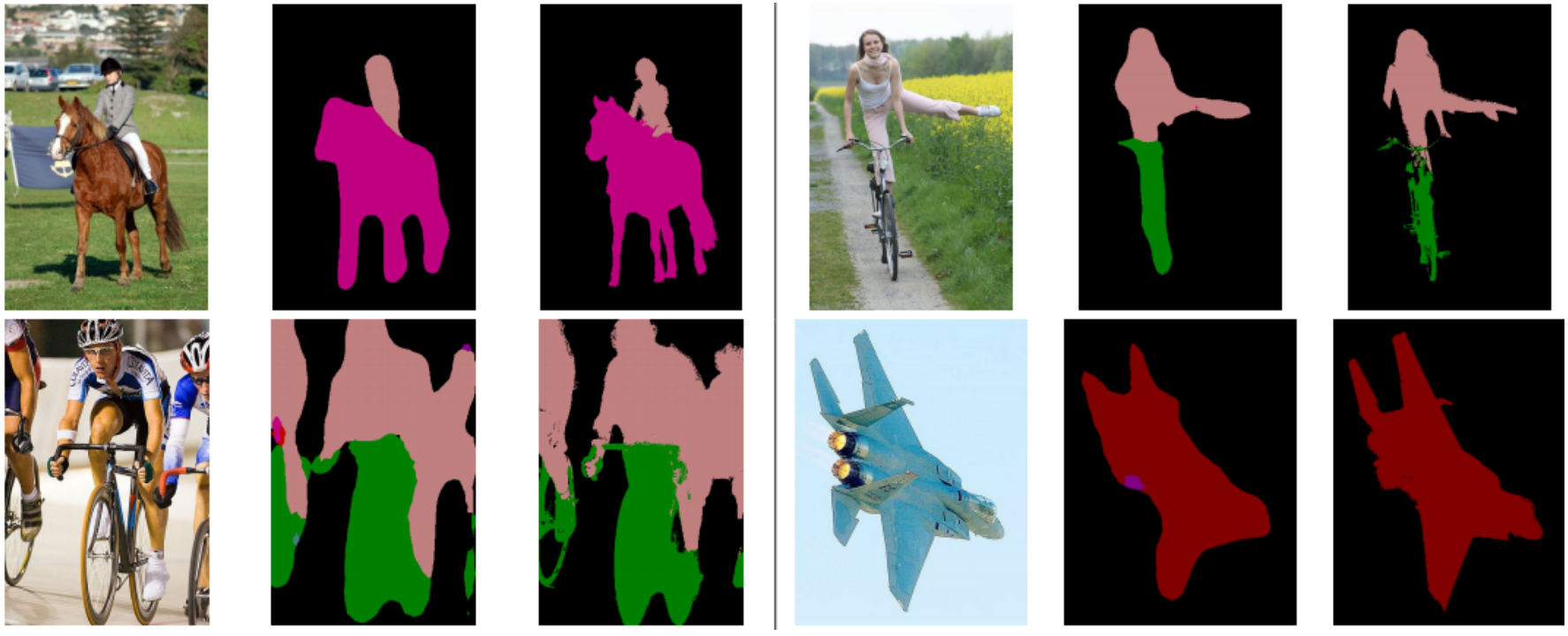


Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

Image Captioning

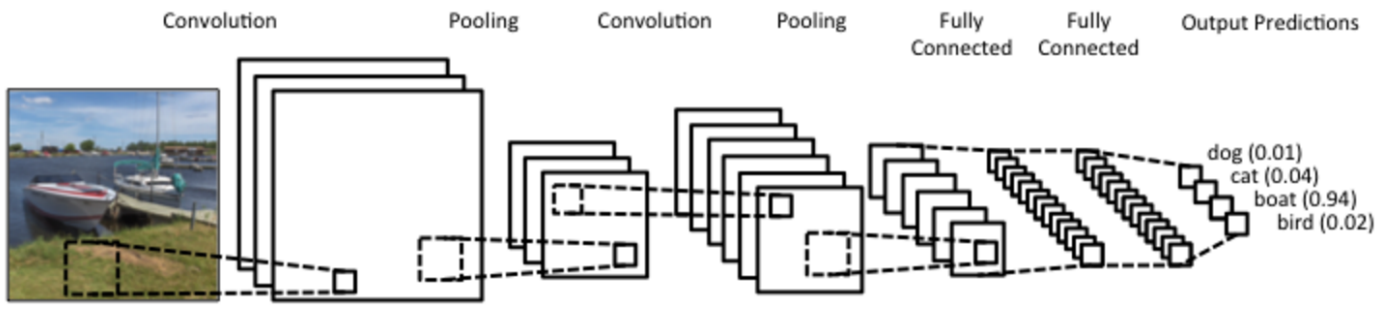
Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

Semantic Segmentation



Chen, Liang-Chieh, et al. "Semantic image segmentation with deep convolutional nets and fully connected crfs." arXiv preprint arXiv:1412.7062 (2014).

Typical CNN


Convolution
Pooling
Convolution
Pooling
Fully Connected
Fully Connected
Output Predictions
dog (0.01)
cat (0.04)
boat (0.94)
bird (0.02)

Representation of images

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

- Colors in RGB (3 dimensions for red, green and blue)

Image processing

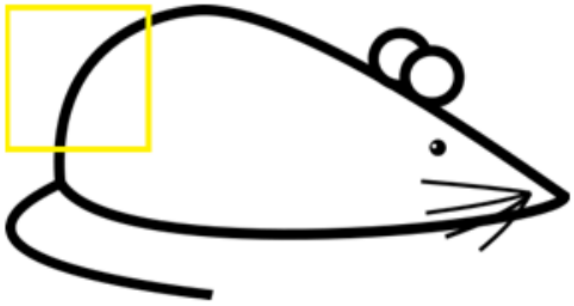
- Image processing has been quite successful before the rise of neural networks
- Experts manually composed smart features
- Features like edge detectors work locally on the natural 2-dimensional structure of images

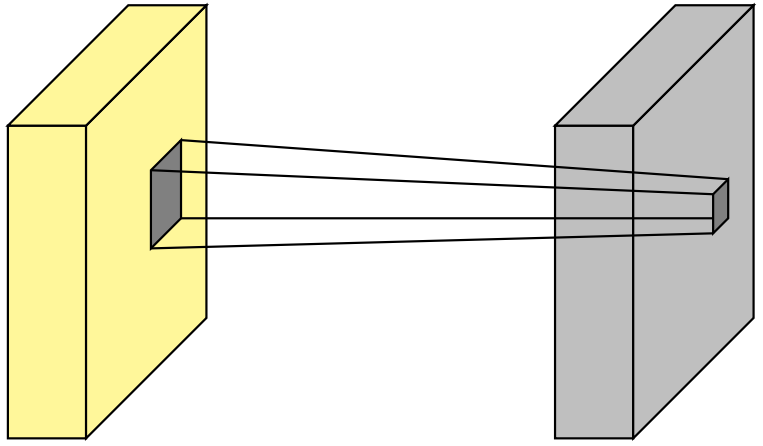
Image processing

- Image processing has been quite successful before the rise of neural networks
- Experts manually composed smart features
- Features like edge detectors work locally on the natural 2-dimensional structure of images

- We have seen that NNs are very successful, because they automatically develop good features
- Can we take advantage of local structures also with NNs?

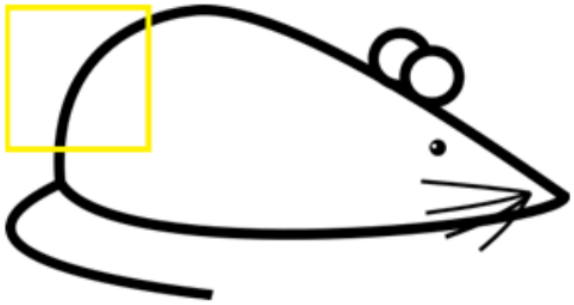
Using local structures

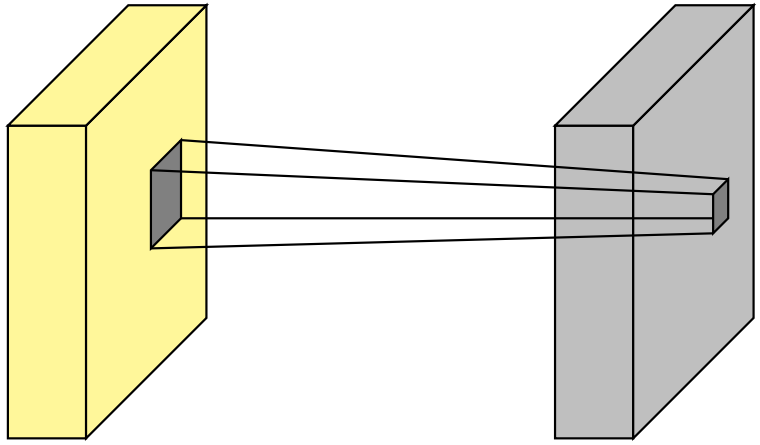




- Slide a filter across the input volume.
- For each subimage (or patch) of the same size as the filter, multiply the two matrices of filter and subimage elementwise and add everything up to a real number.
- The weights $w_{s,t}$ of the filter are parameters that the neural net has to learn.

Using local structures





- Slide a filter across the input volume.
- For each subimage (or patch) of the same size as the filter, multiply the two matrices of filter and subimage elementwise and add everything up to a real number.
- The weights $w_{s,t}$ of the filter are parameters that the neural net has to learn.
- The weight parameters stay equal while the filter is moving over the image (weight sharing).

$$y_{i,j} = \sum_{s=0}^{2k} \sum_{t=0}^{2k} w_{1+s,1+t} \cdot x_{i-k+s,i-k+t}$$

Convolutional Layer - Example

$[x_{i,j}]$

1	-2	0	5
7	0	1	-3
2	-2	2	-2
-5	-4	3	1

$[w_{i,j}]$

1	-2
3	1

$[y_{i,j}]$

- Select a patch in the input and compute the dot product of the patch and the filter.
- Spatially move the patch over the image.

Convolutional Layer - Example

$[x_{i,j}]$

1	-2	0	5
7	0	1	-3
2	-2	2	-2
-5	-4	3	1

$[w_{i,j}]$

1	-2
3	1

$[y_{i,j}]$

26		

- Select a patch in the input and compute the dot product of the patch and the filter.
- Spatially move the patch over the image.

Convolutional Layer - Example

$[x_{i,j}]$

1	-2	0	5
7	0	1	-3
2	-2	2	-2
-5	-4	3	1

$[w_{i,j}]$

1	-2
3	1

$[y_{i,j}]$

26	-1	

- Select a patch in the input and compute the dot product of the patch and the filter.
- Spatially move the patch over the image.

Size of the filter

- The filter must have the same number of channels as the input.
- We sum over the channels to get one real value per patch.
- For images in rgb a filter has dimension $w_{height} \times w_{width} \times 3$
- Each filter produce one output channel.

Convolutional Layer - Example with two channels

$x(i,j,1)$

1	-2	0	5
7	0	1	-3
2	-2	2	-2
-5	-4	3	1

$w(i,j,1)$

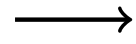
1	-2
3	1

$x(i,j,2)$

0	-1	2	0
-4	9	6	5
3	2	1	0
0	1	3	-3

$w(i,j,2)$

1	-2
3	1



$y(:,:,1)$

25		

Convolutional Layer - Example with two channels

$x(i,j,1)$

1	-2	0	5
7	0	1	-3
2	-2	2	-2
-5	-4	3	1

$w(i,j,1)$

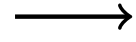
1	-2
3	1

$x(i,j,2)$

0	-1	2	0
-4	9	6	5
3	2	1	0
0	1	3	-3

$w(i,j,2)$

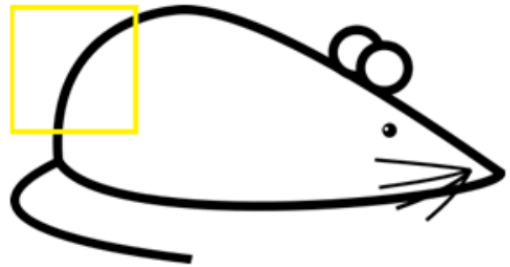
1	-2
3	1



$y(:,:,1)$

25	27	

Filter action



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

The filter detects this curve at this position.

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Filter action

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

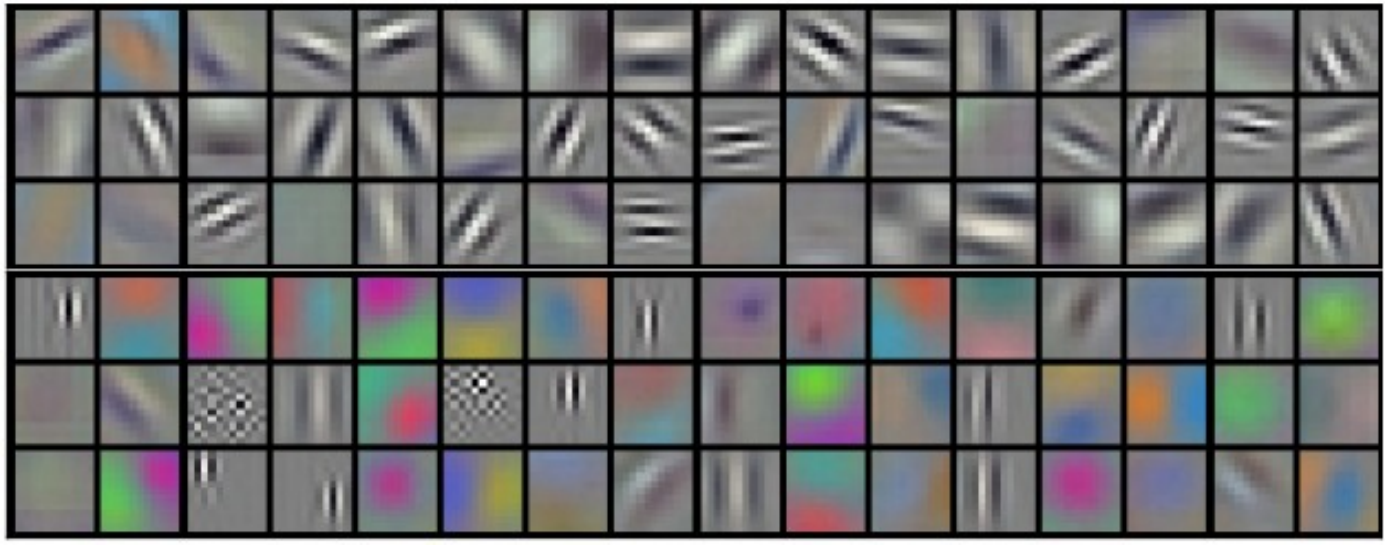
Pixel representation of filter

Multiplication and Summation = 0

The filter does not detect such a curve at this position.

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

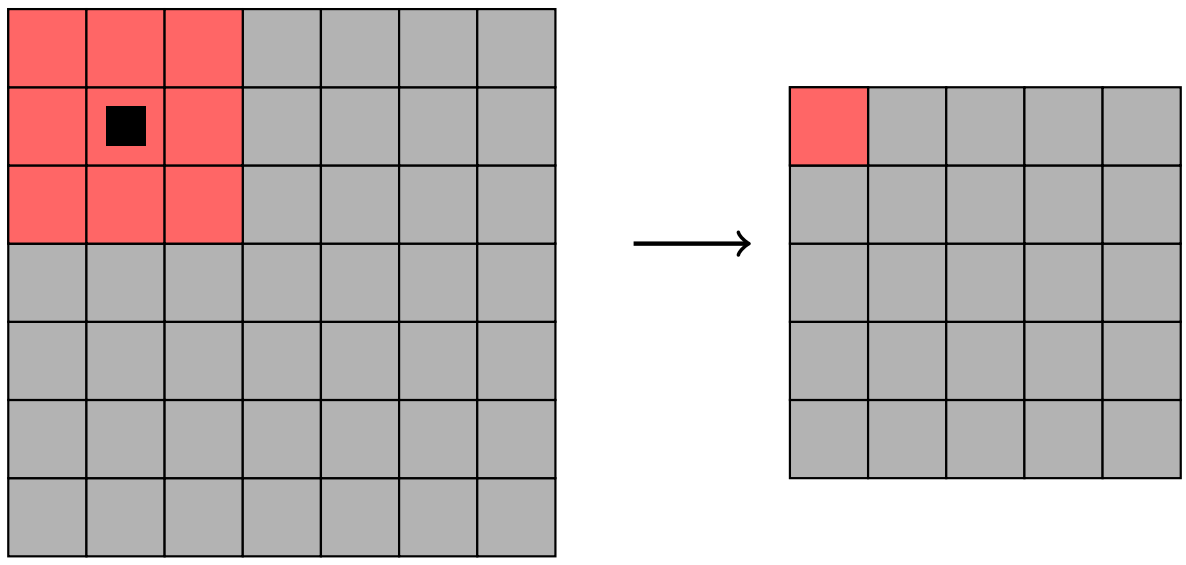
Convolutional Layer



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

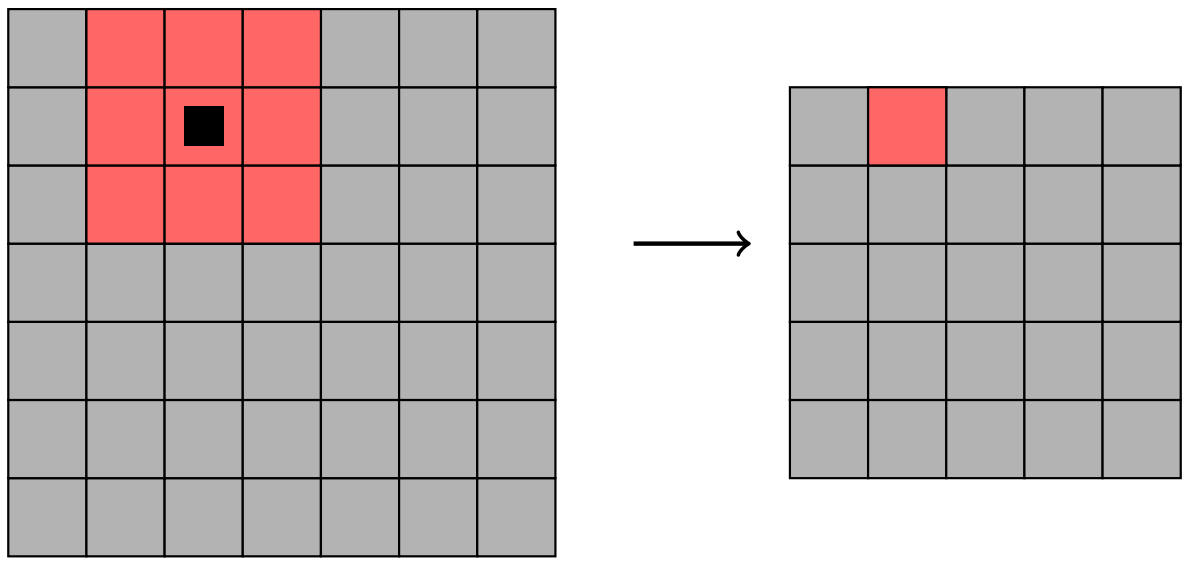
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



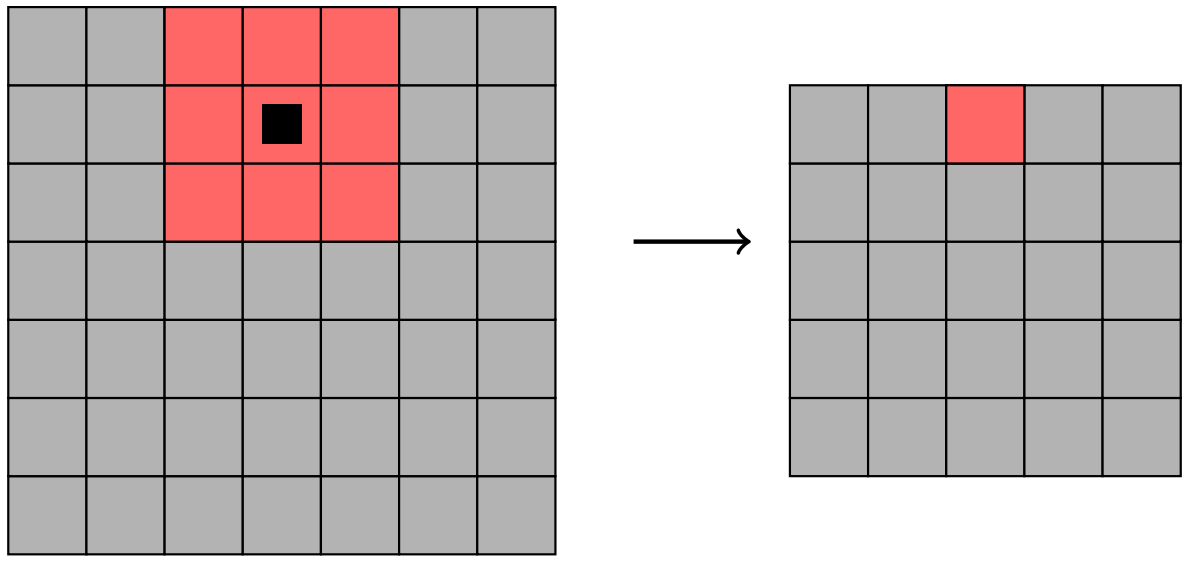
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



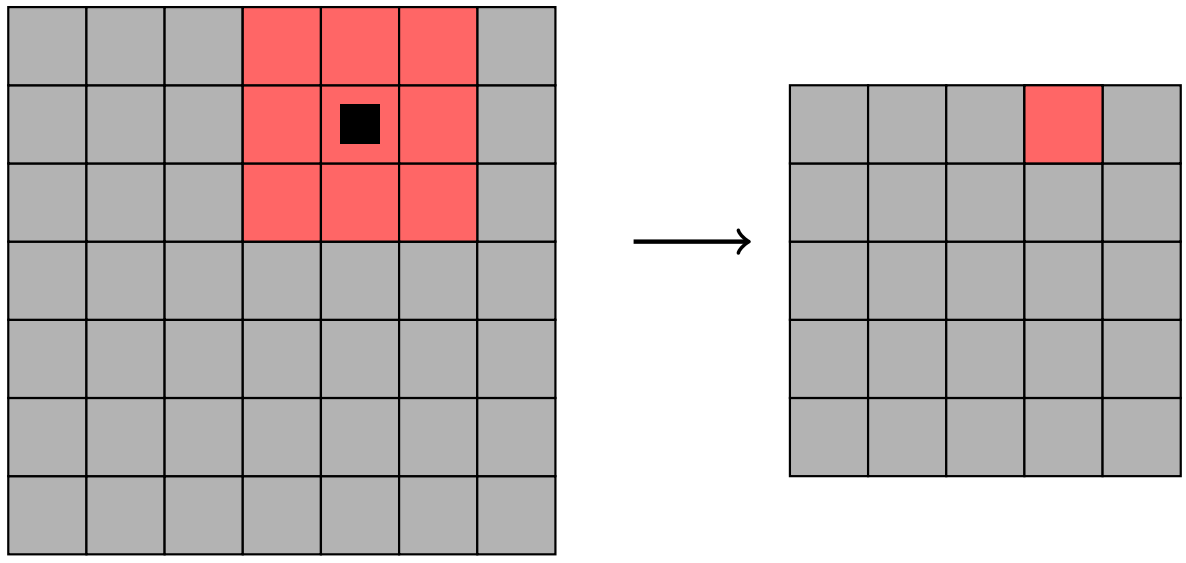
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



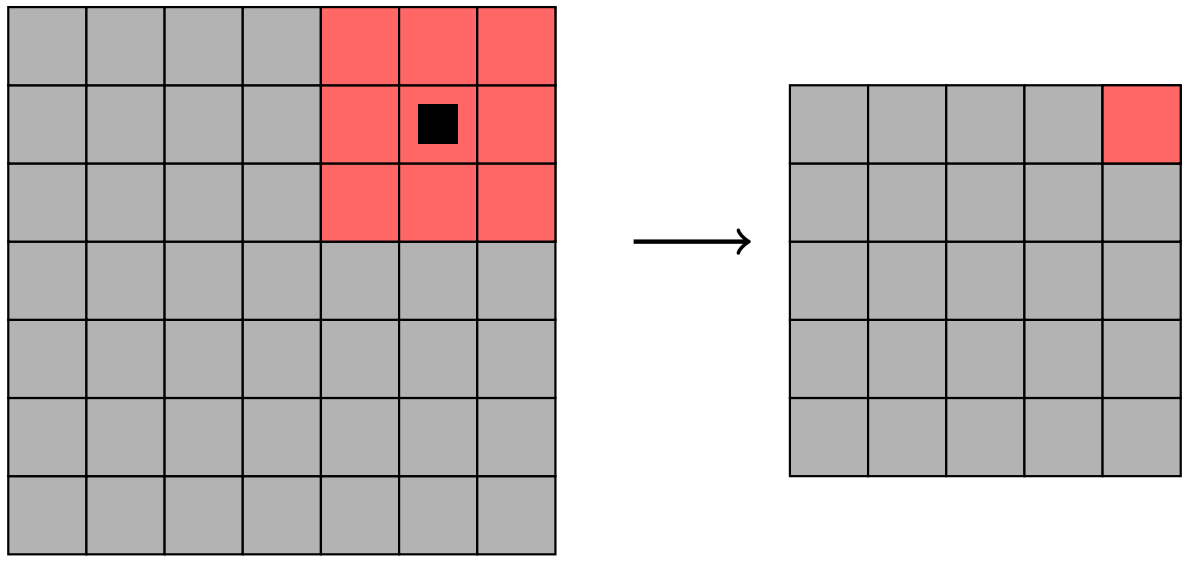
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



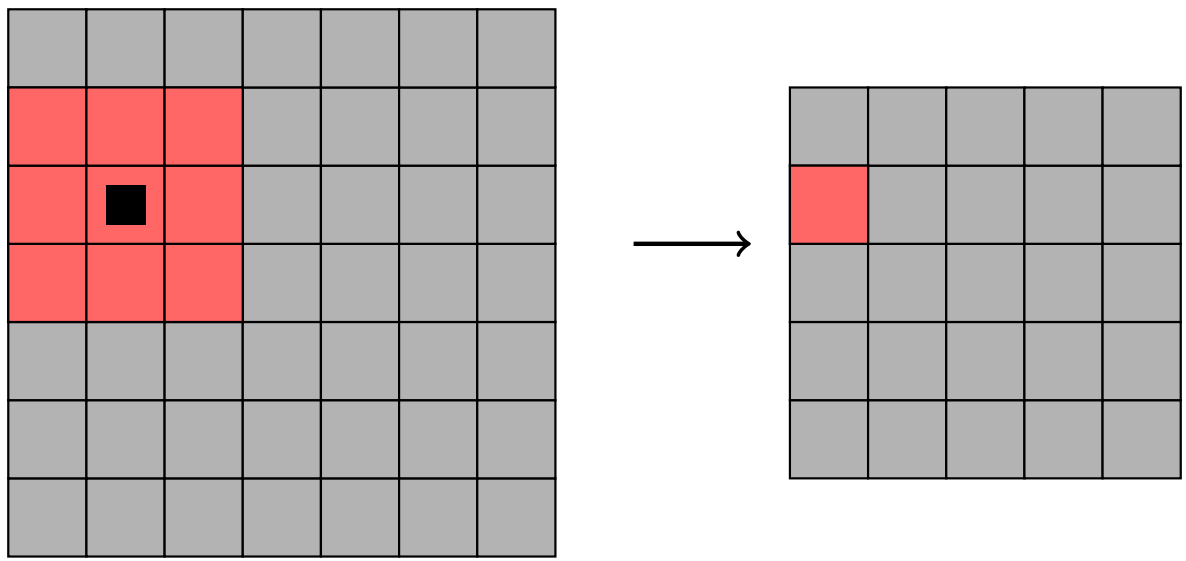
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



Convolutional Layer

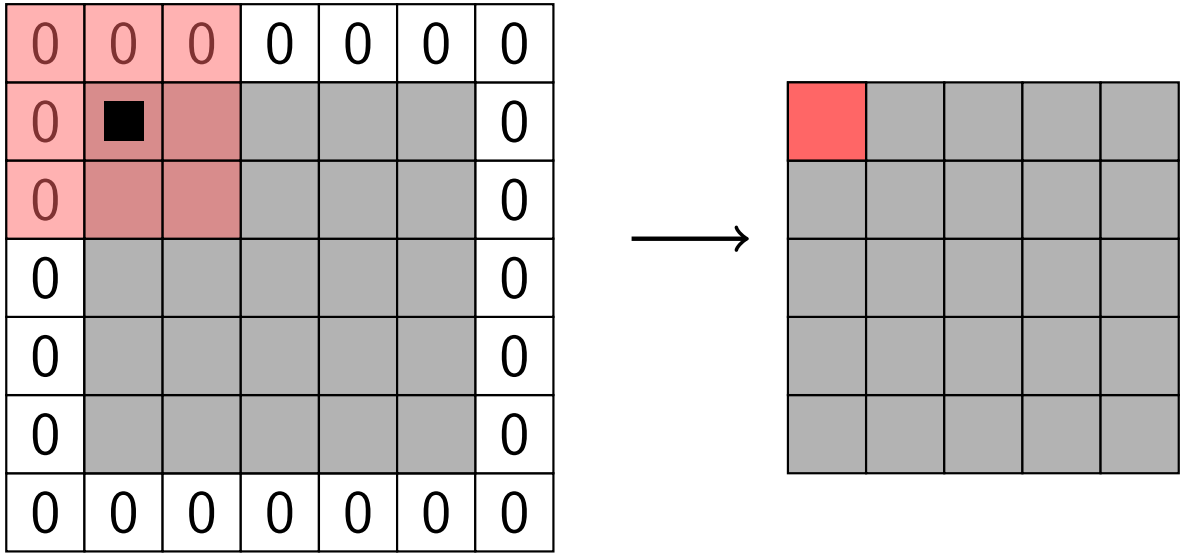
- We convolve a 7x7 image with a 3x3 filter. What is the output size?



In general: Convolving a $n \times m$ image with a $f_h \times f_w$ filter will result in size $(n - f_h + 1) \times (m - f_w + 1)$.

Convolutional layer

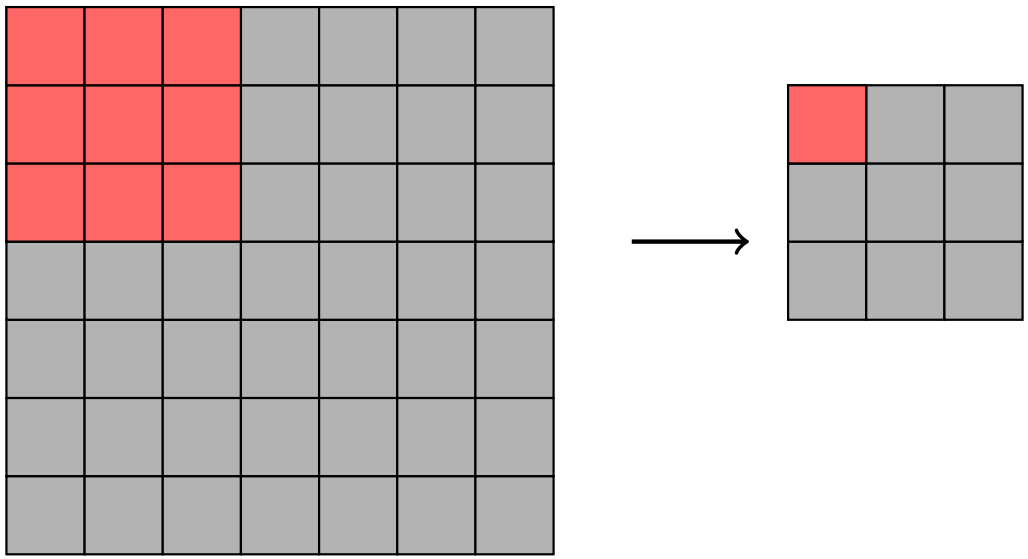
- What if we want the same output and input sizes?
- Pad with zeros!



- The result is also 5×5.
- Pad with $(f-1)/2$ if you want to retain the size.

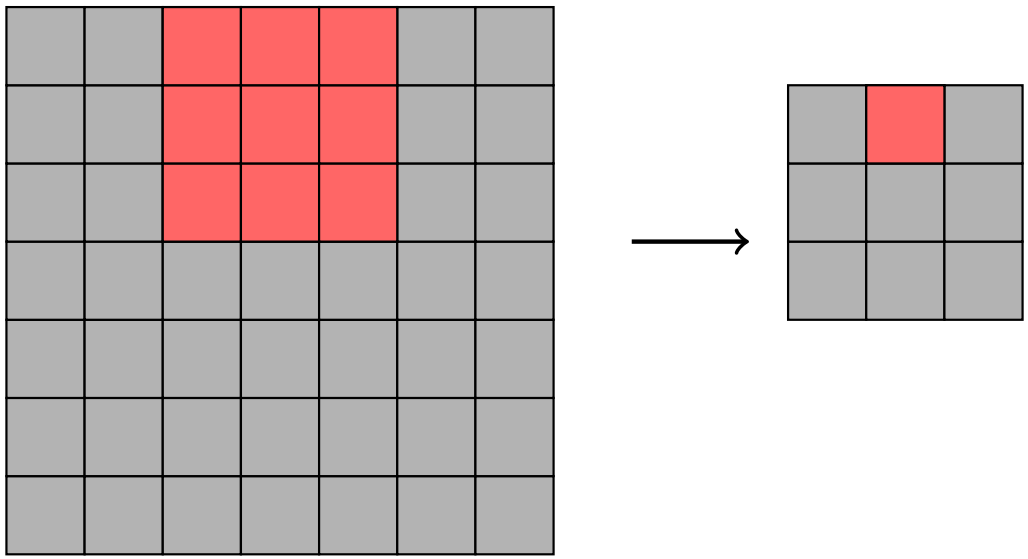
Strides

- Longer strides are possible.



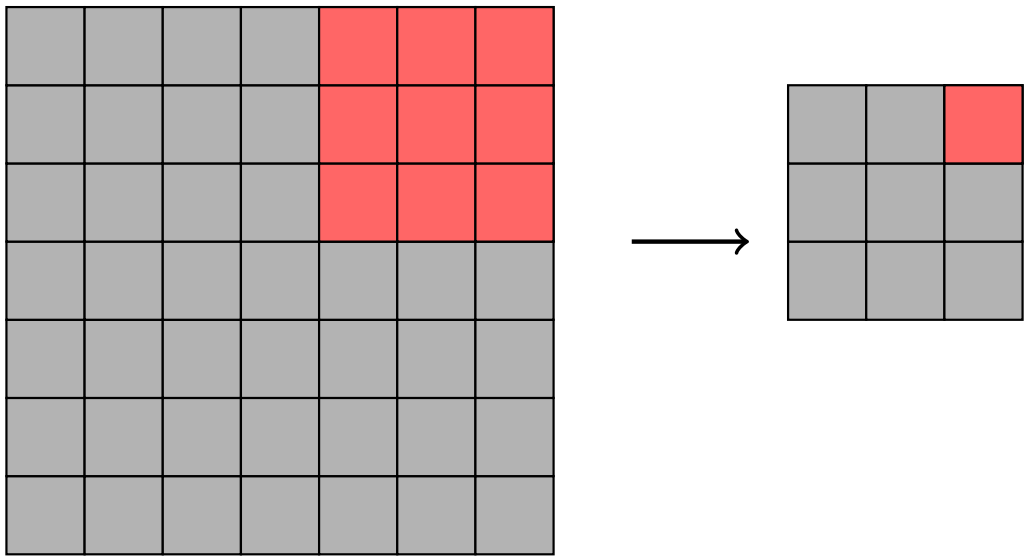
Strides

- Longer strides are possible.



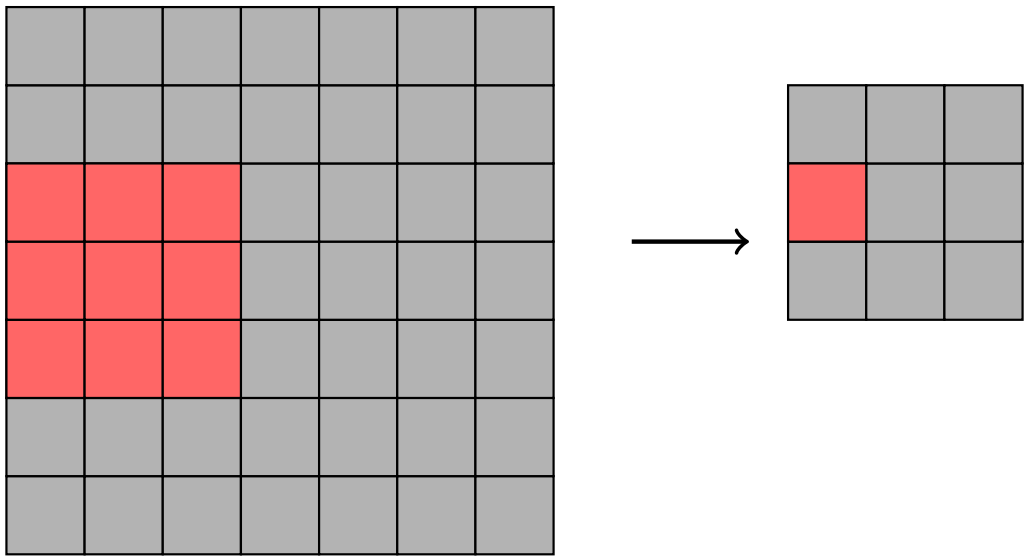
Strides

- Longer strides are possible.



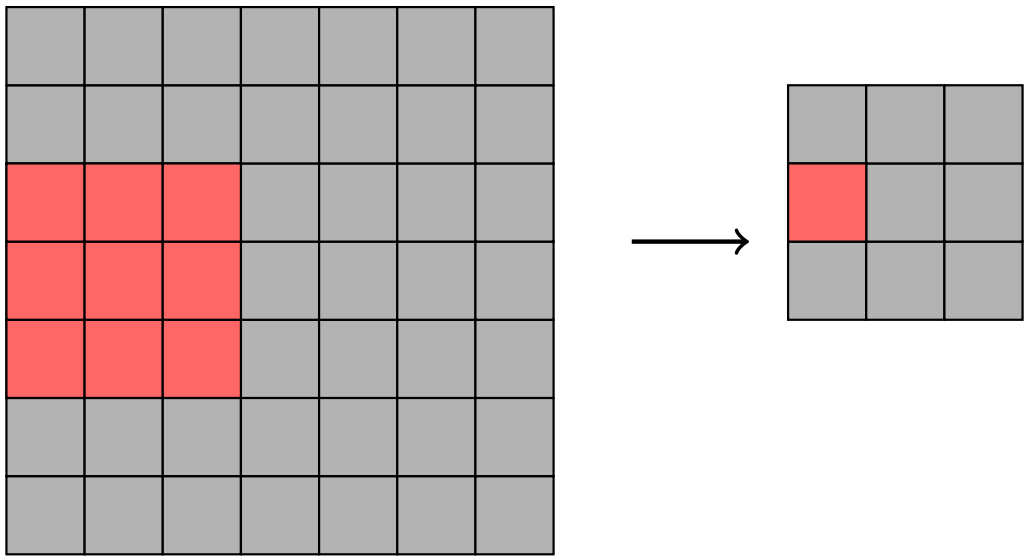
Strides

- Longer strides are possible.



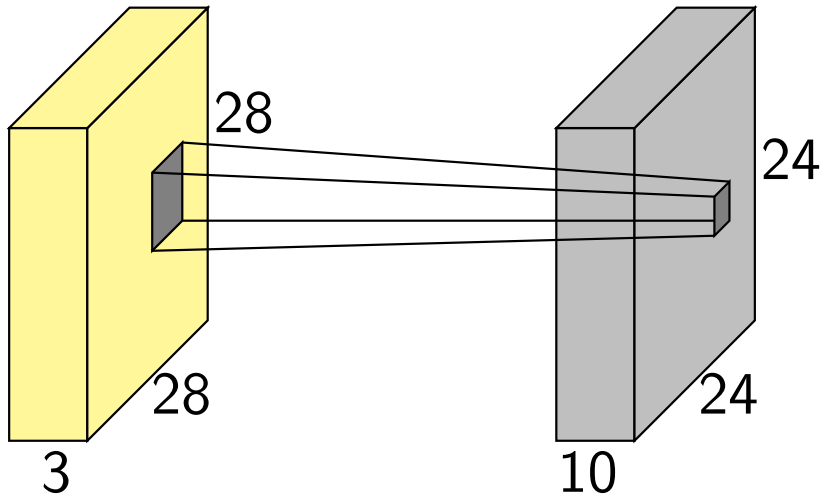
Strides

- Longer strides are possible.



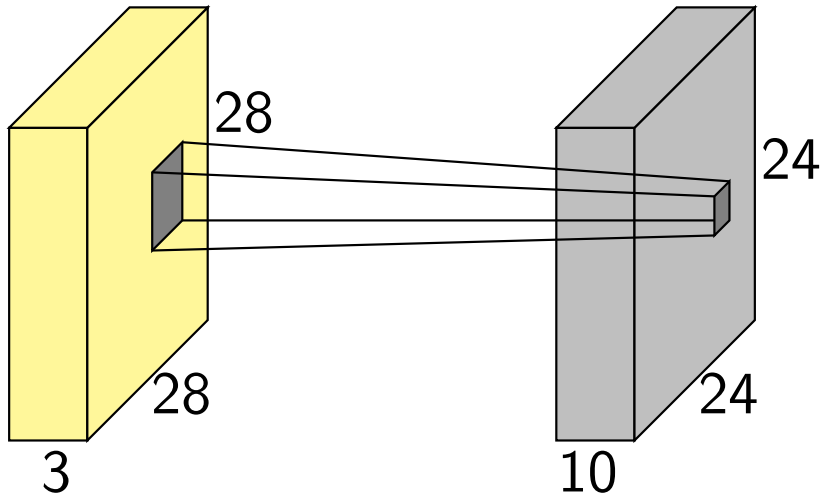
- Convolving a $n \times m$ image with a $f_h \times f_w$ filter and a stride of k will result in size $(n - f_h)/k + 1 \times (m - f_w)/k + 1$

Input and output shapes



- In the assignment, filter weights have 4 dimensions. E.g. the shape (5, 5, 3, 10) represents a 5×5 filter mapping 3 input channels to 10 output channels.
- There are 10 $(5, 5, 3)$ filters. Each filter gets one output channel.
- Output channel 1 is computed by convolving x with $w(:, :, :, 1)$, channel 2 by convolving x with $w(:, :, :, 2)$, etc.

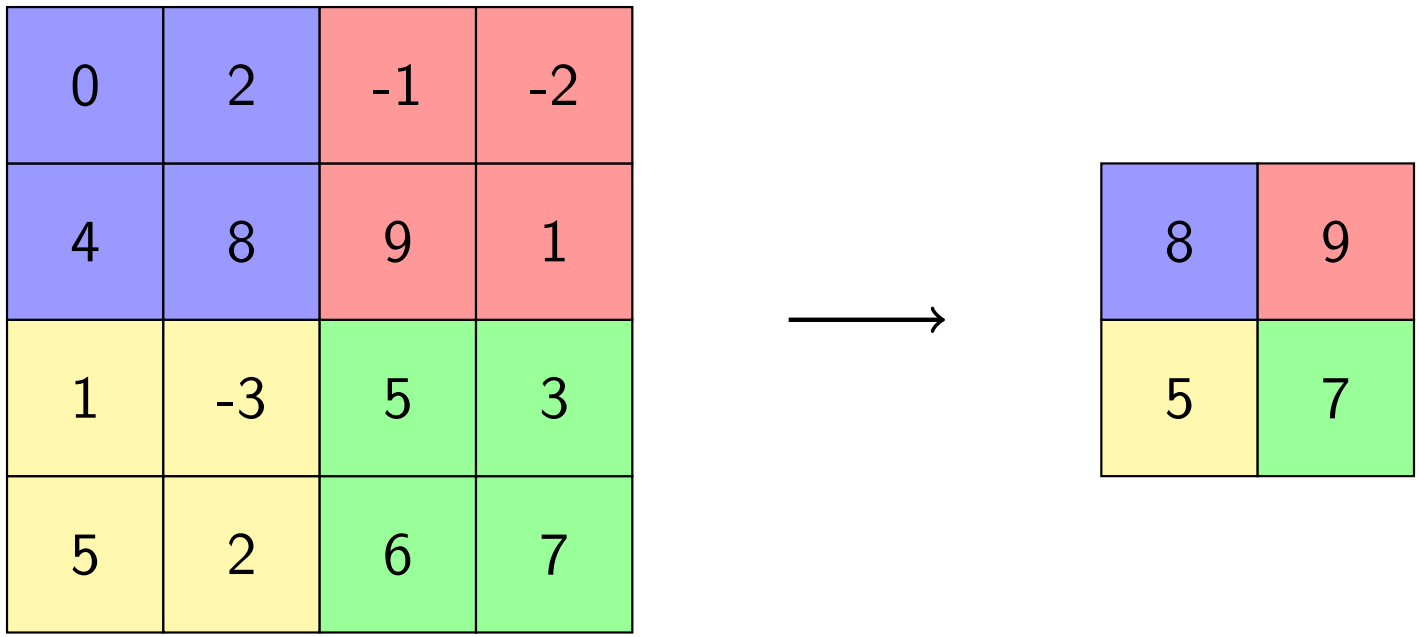
Input and output shapes



- The number of filter channels must match the number of input channels!
- Each output channel has a bias element. In total 10 such parameters for the layer in the figure. Note that there are $5 \cdot 5 \cdot 3 \cdot 10$ filter weights.

Maxpooling Layer

To reduce dimensions and check whether the feature is present in a patch, we can perform maxpooling.

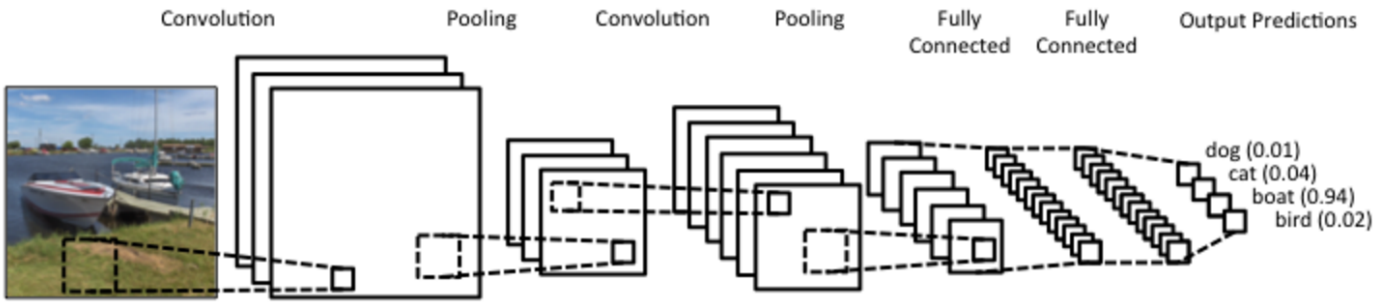


- 2x2 maxpooling

$$y_{ijc} = \max\{x_{2i,2j,c}, x_{2i,2j+1,c}, x_{2i+1,2j,c}, x_{2i+1,2j+1,c}\}$$

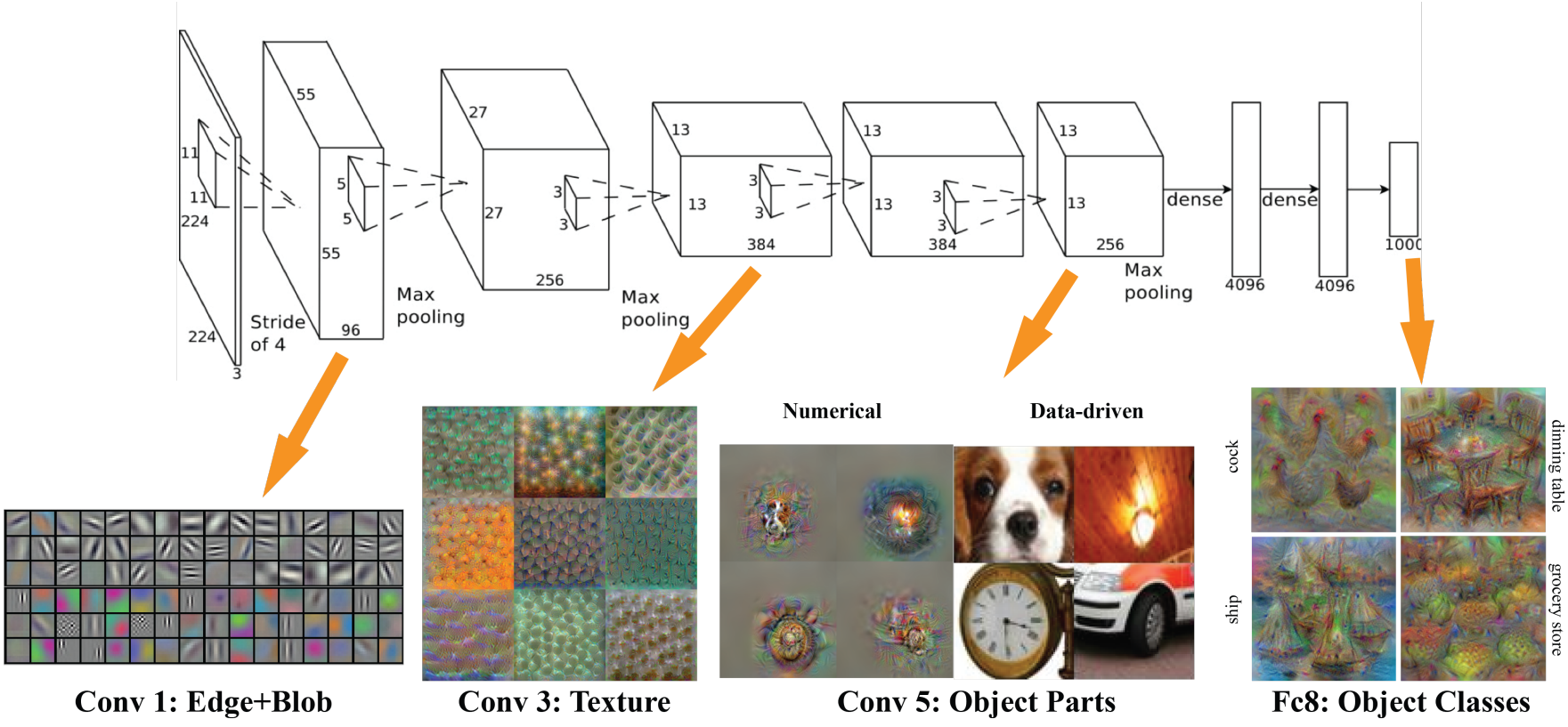
- Subsample and retain the strongest activations.
- Computed for each channel independently.

Typical CNN

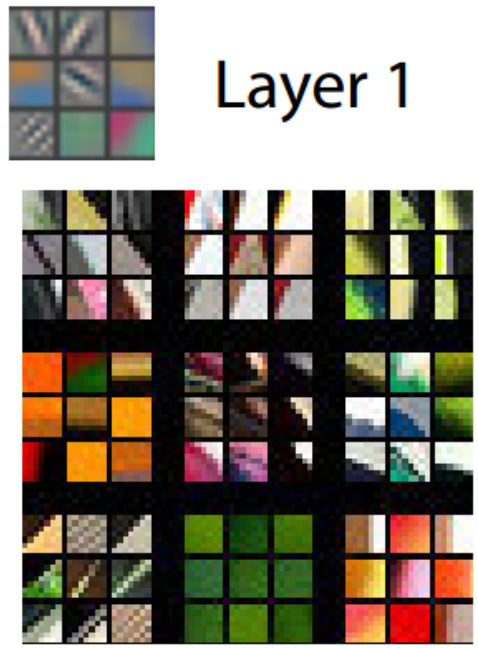


- Use the layers as building blocks in a large network.
- Convolution - relu - max-pooling
- Often a few conv-relu pairs between the max-pooling layers.
- One or more fully connected layers at the very end.

Hierarchical Features

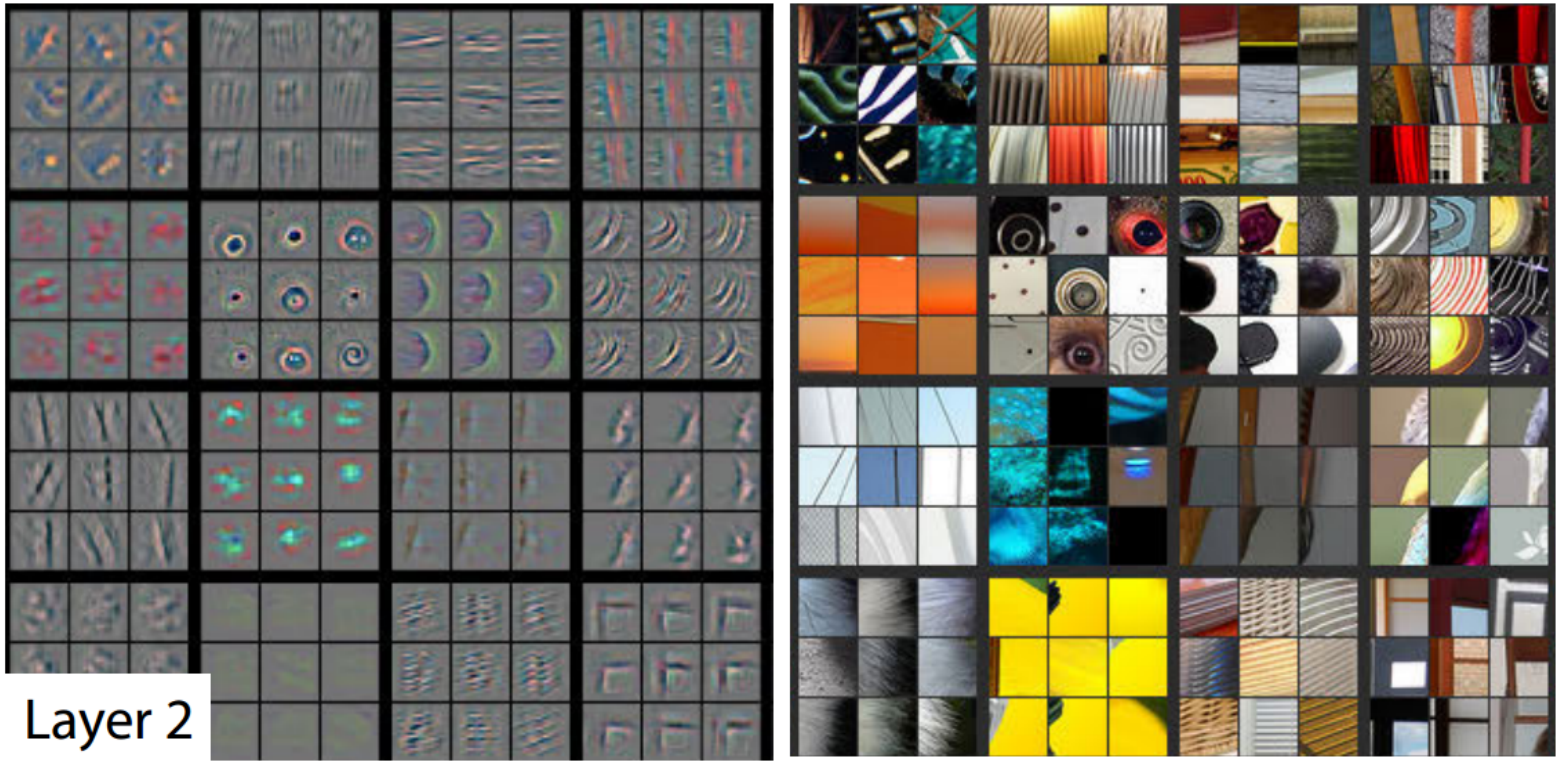


Hierarchical Features



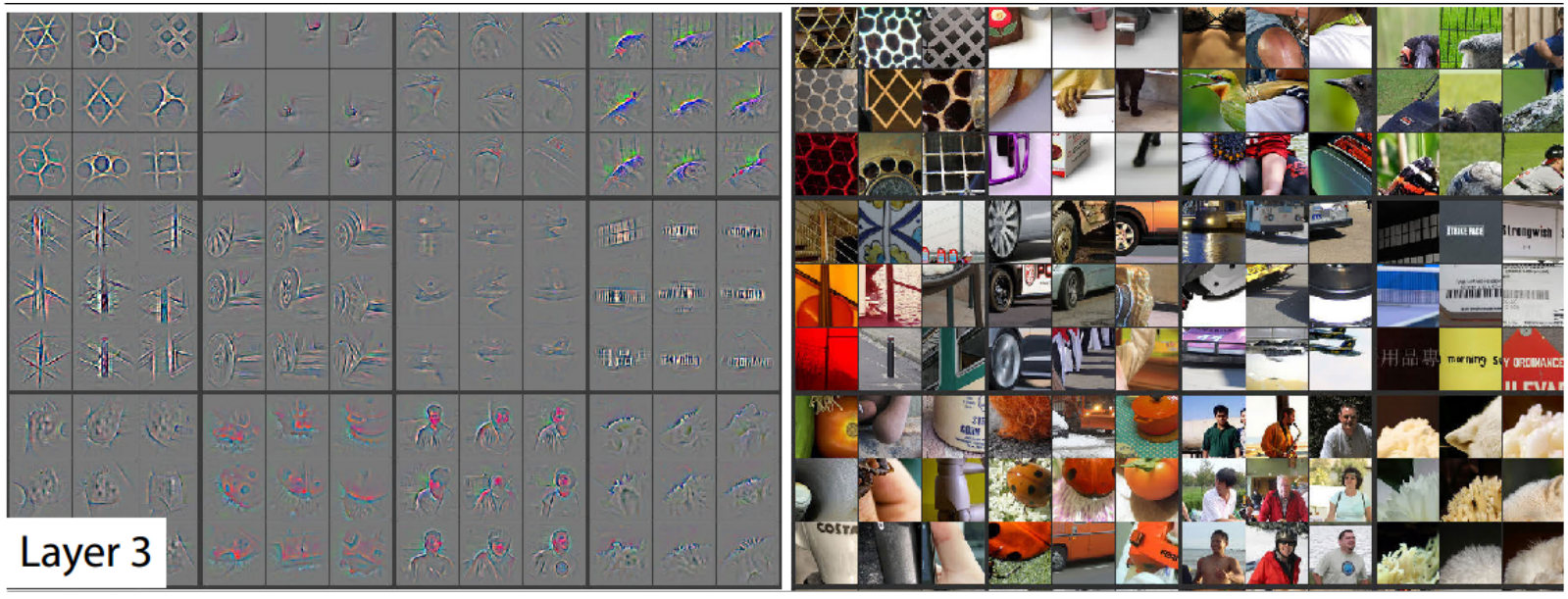
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Hierarchical Features



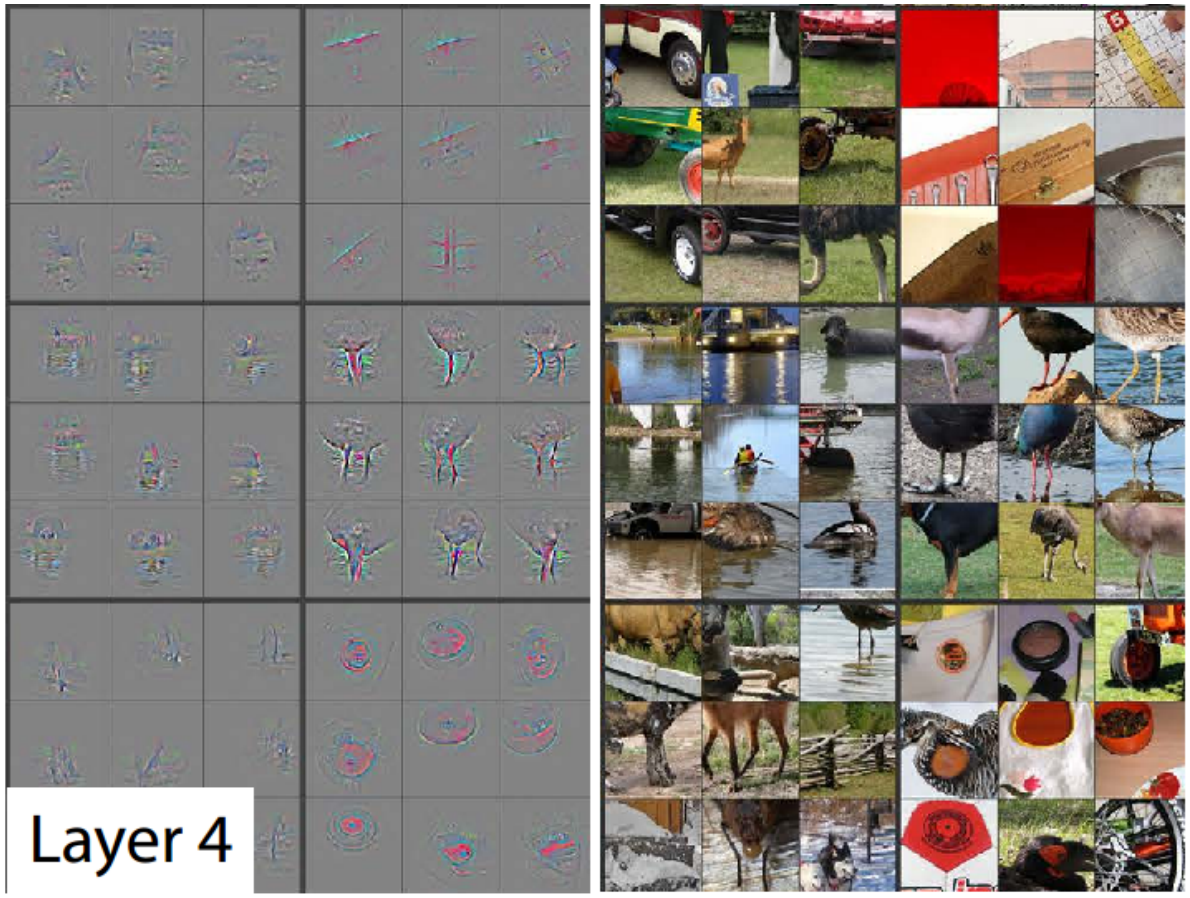
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Hierarchical Features



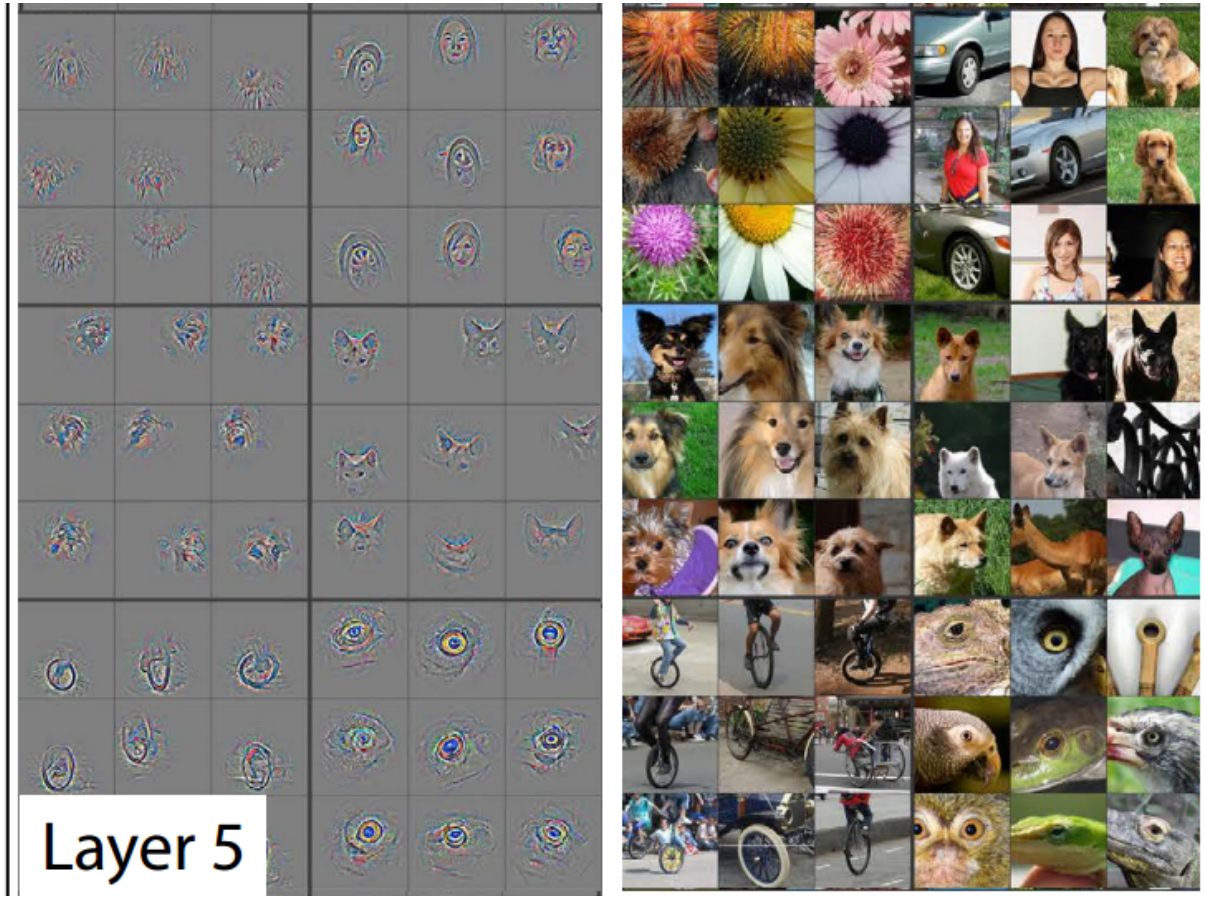
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Hierarchical Features



Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

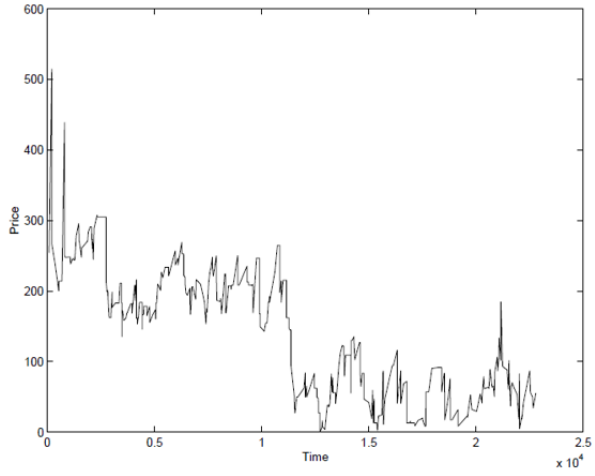
Hierarchical Features



Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Sequential data

- Images come with a natural 2-dimensional structure that we could exploit by using CNNs
- Other data like language has a natural sequential (or temporal) structure.



AAGTCAAGCTGCTCTGTGGGCTGTGATCTGCCTCAAACCCACAGCCTGGGTAGCAGG
AGGACCTTGATGCTCCTGGCACAGATGAGGAGAATCTCTCTTTTCTCCTGCTTGAAG
GACAGACATGACTTTGGATTTCCCCAGGAGGAGTTTGGCAACCAGTTCCAAAAGGCT
GAAACCATCCCTGTCCTCCATGAGATGATCCAGCAGATCTTCAATCTCTTCAGCACA
AAGGACTCATCTGCTGCTTGGGATGAGACCCTCCTAGACAAATTCTACACTGAACTC
TACCAGCAGCTGAATGACCTGGAAGCCTGTGTGATACAGGGGGTGGGGGTGACAGAG
ACTCCCCTGATGAAGGAGGACTCCATTCTGGCTGTGAGGAAATACTTCCAAAGAATC
ACTCTCTATCTGAAAGAGAAGAAATACAGCCCTTGTGCCTGGGAGGTTGTCAGAGCA
GAAATCATGAGATCTTTTTCTTTGTCAACAAACTTGCAAGAAAGTTTAAGAAGTAAG
GAATGA, TGTGATCTGCCTCAAACCCACAGCCTGGGTAGCAGGAGGACCTTGATGC
TCCTGGCACAGATGAGGAGAATCTCTCTTTTCTCCTGCTTGAAGGACAGACATGACT
TTGGATTTCCCCAGGAGGAGTTTGGCAACCAGTTCCAAAAGGCTGAAACCATCCCTG
TCCTCCATGAGATGATCCAGCAGATCTTCAATCTCTTCAGCACAAAGGACTCATCTG
CTGCTTGGGATGAGACCCTCCTAGACAAATTCTACACTGAACTCTACCAGCAGCTGA
ATGACCTGGAAGCCTGTGTGATACAGGGGGTGGGGGTGACAGAGACTCCCCTGATGA
AGGAGGACTCCATTCTGGCTGTGAGGAAATACTTCCAAAGAATCACTCTCTATCTGA
AAGAGAAGAAATACAGCCCTTGTGCCTGGGAGGTTGTCAGAGCAGAAATCATGAGAT
CTTTTTCTTTGTCAACAAACTTGCAAGAAAGTTTAAGAAGTAAGGAATGA and

this deep learning course is great!

this deep learning course is great?

this deep learning course is great!

Problems involving sequential data

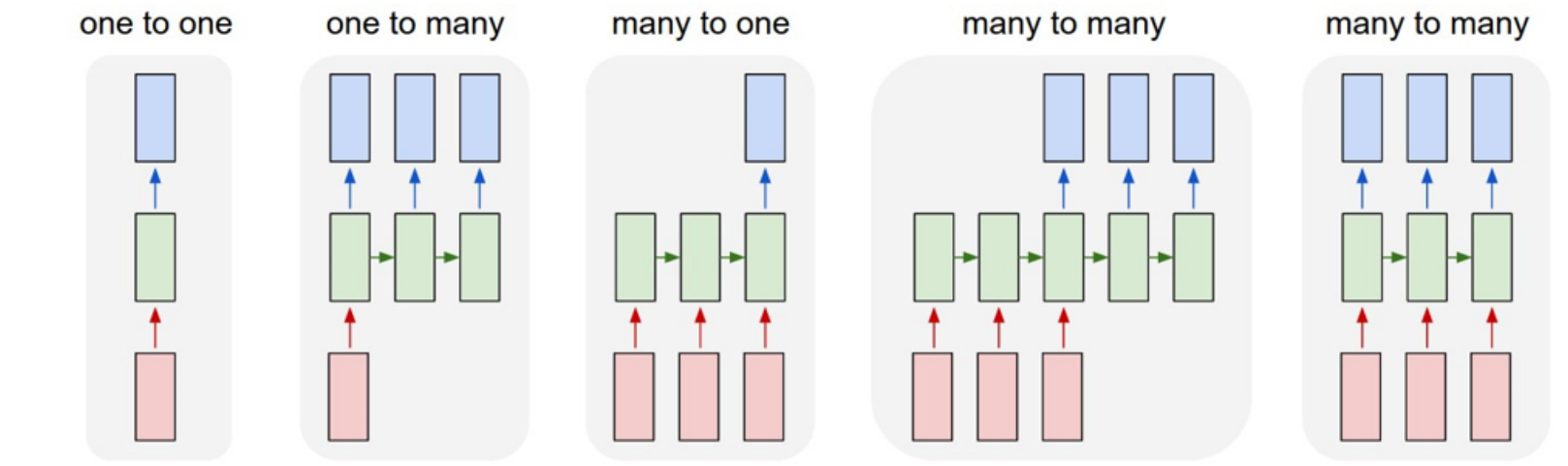
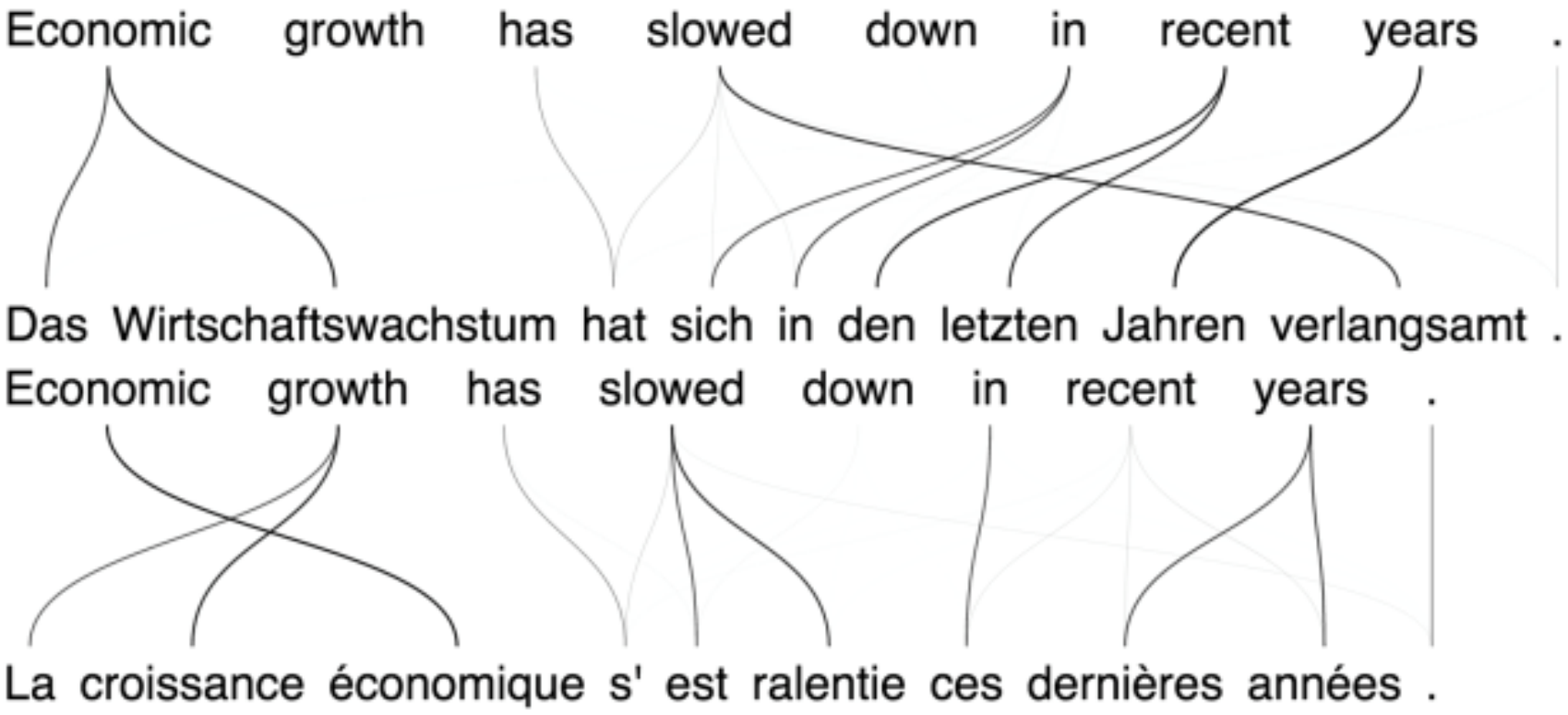


Image Captioning

Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

Machine Translation



Sequential data

How can we exploit sequential data?

$\overbrace{\text{In deep learning, a}}^{\textit{slide window}}$ convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

Sequential data

How can we exploit sequential data?

- Idea 1: Use CNN's again.

$\overbrace{\text{learning, a convolutional neural}}^{\textit{slide window}}$

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

Sequential data

How can we exploit sequential data?

- Idea 1: Use CNN's again.

slide window

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

Sequential data

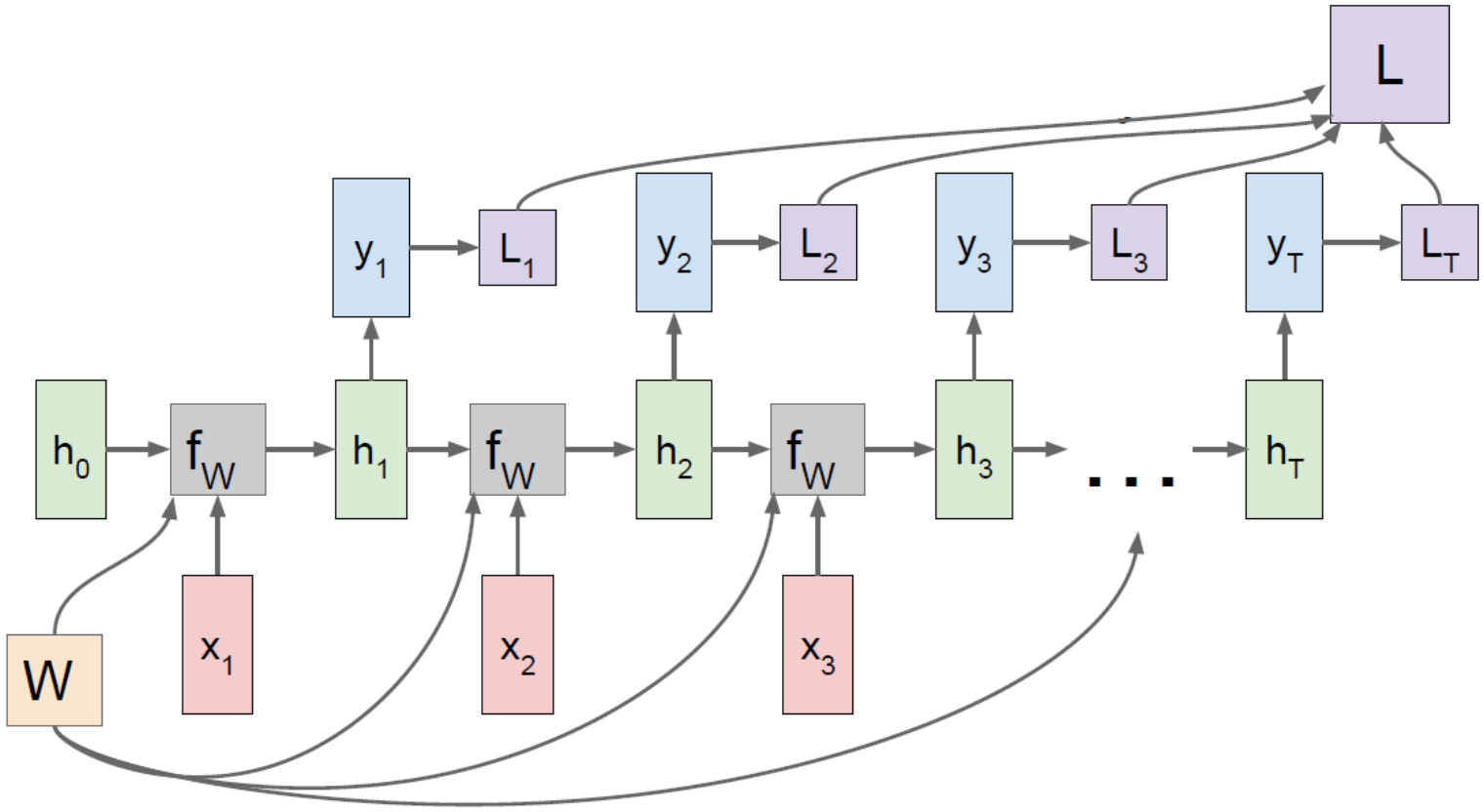
There are some disadvantages to taking CNNs.

- The window considers local features.
- The sentences could be of variable length. How to deal with that?

We should work with models tailored for sequential data.

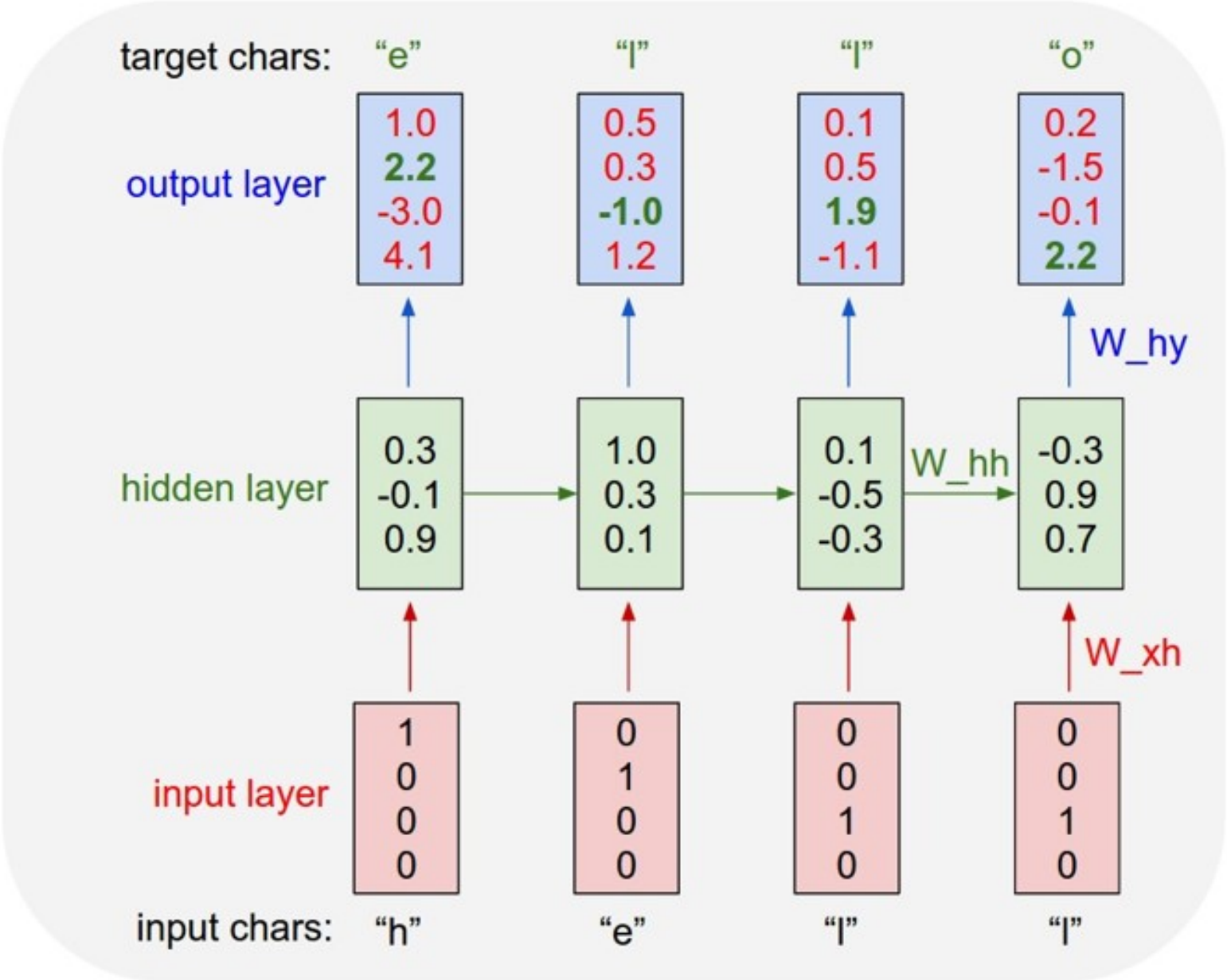
Recurrent Neural Networks

RNNs give model for tasks with sequential input and sequential output and use hidden/latent vectors.



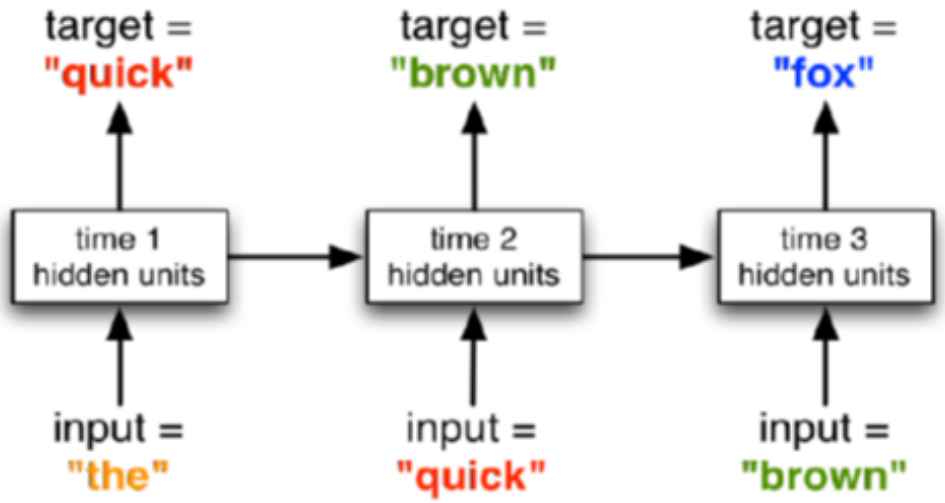
Li , Johnson , Yeung (cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf)

Recurrent Neural Networks



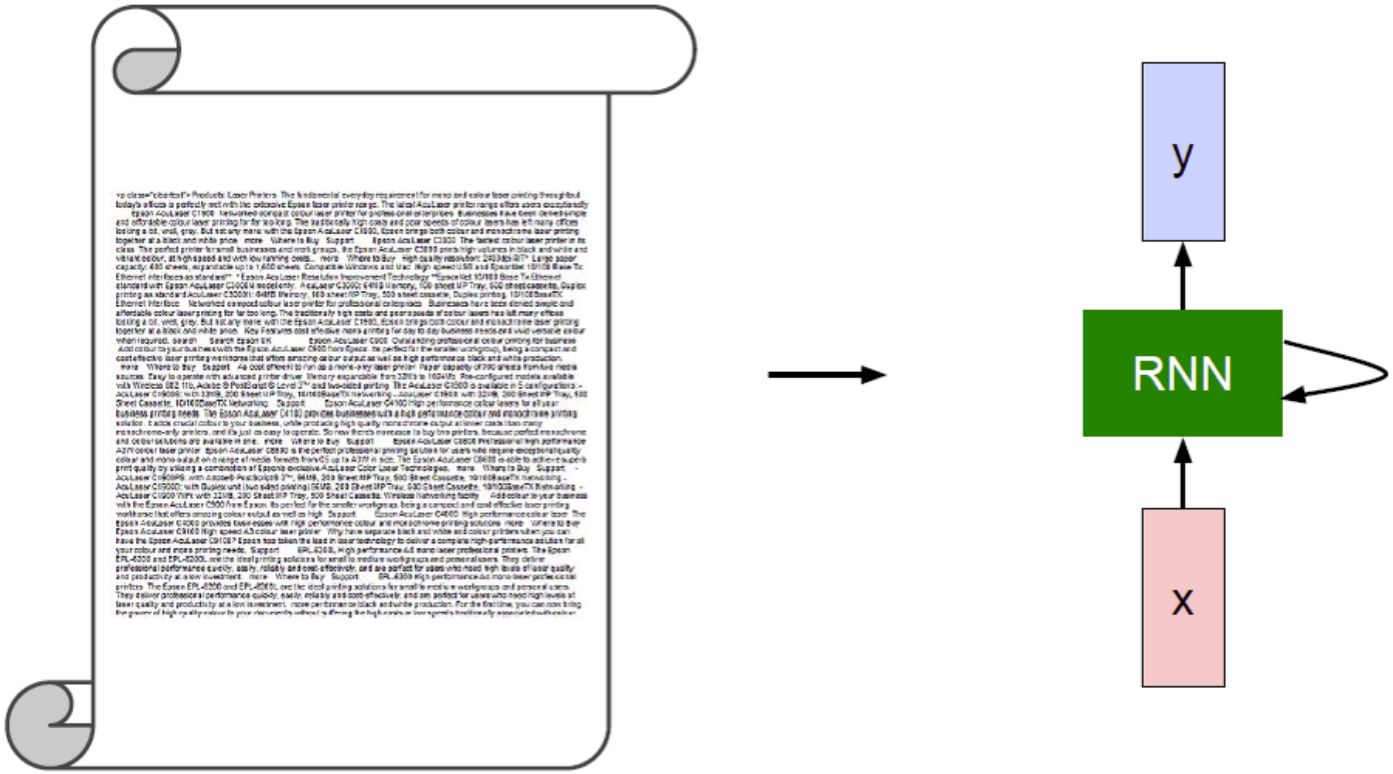
Karpathy http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Text generation with RNNs



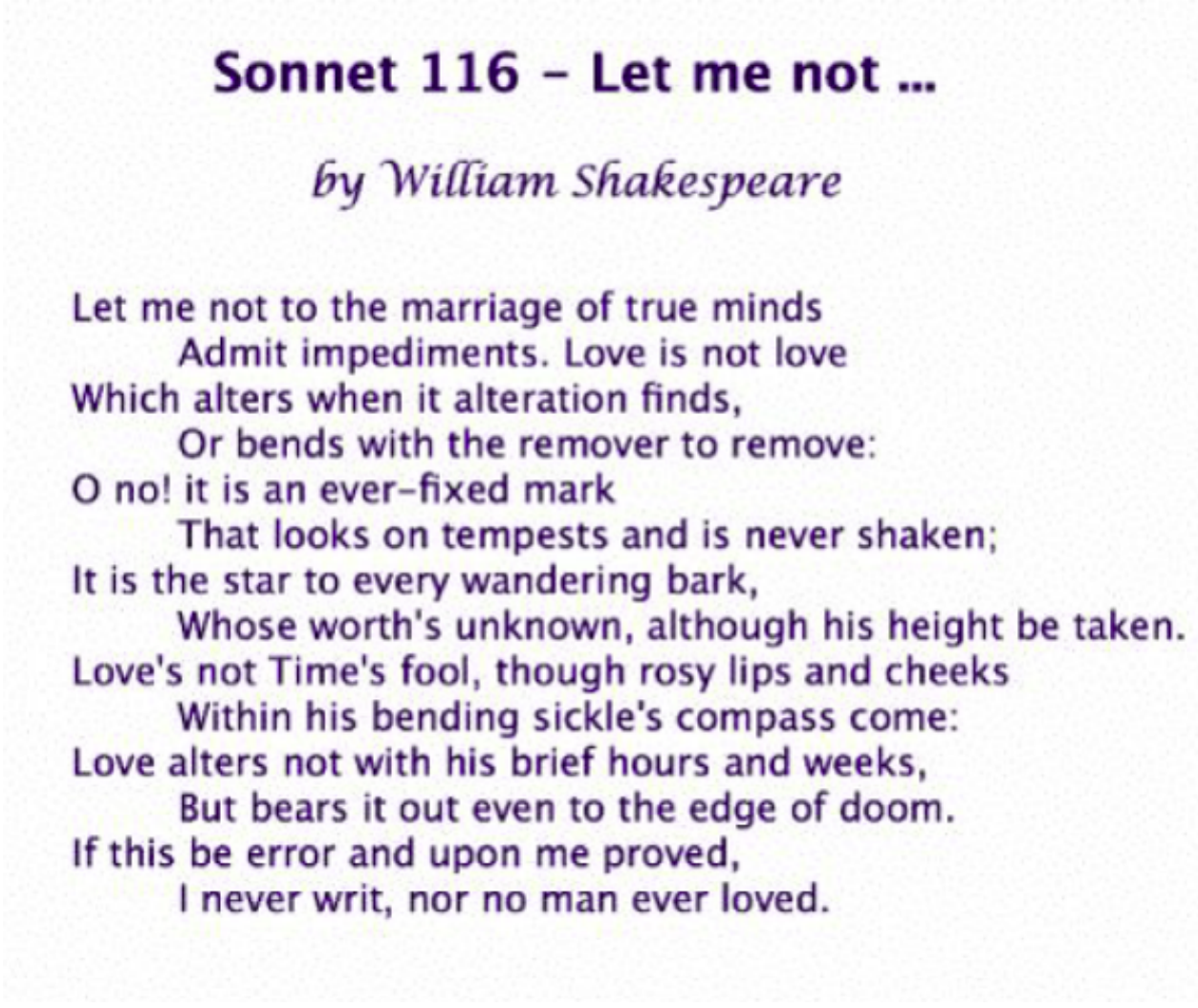
Learning to generate Shakespeare

We feed a lot of Shakespeare's work into an RNN for text generation.



Learning to generate Shakespeare

This is how Shakespeare sounds like.



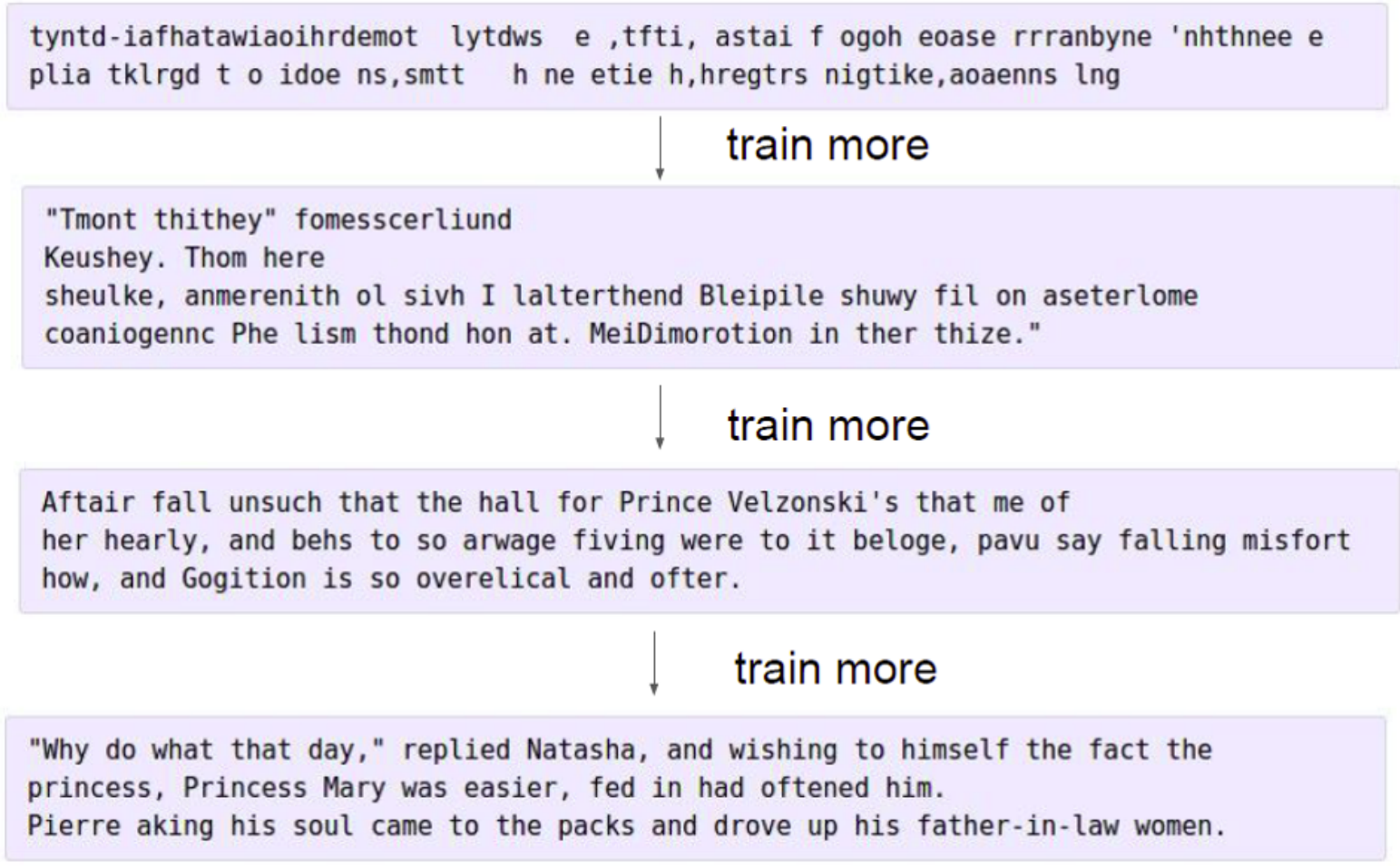
Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.

Learning to generate Shakespeare

While learning the RNN improves its text generation.



Karpathy (2016)

Learning to generate Shakespeare

Finally, it the learned model does sound like Shakespeare.

```
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.
```

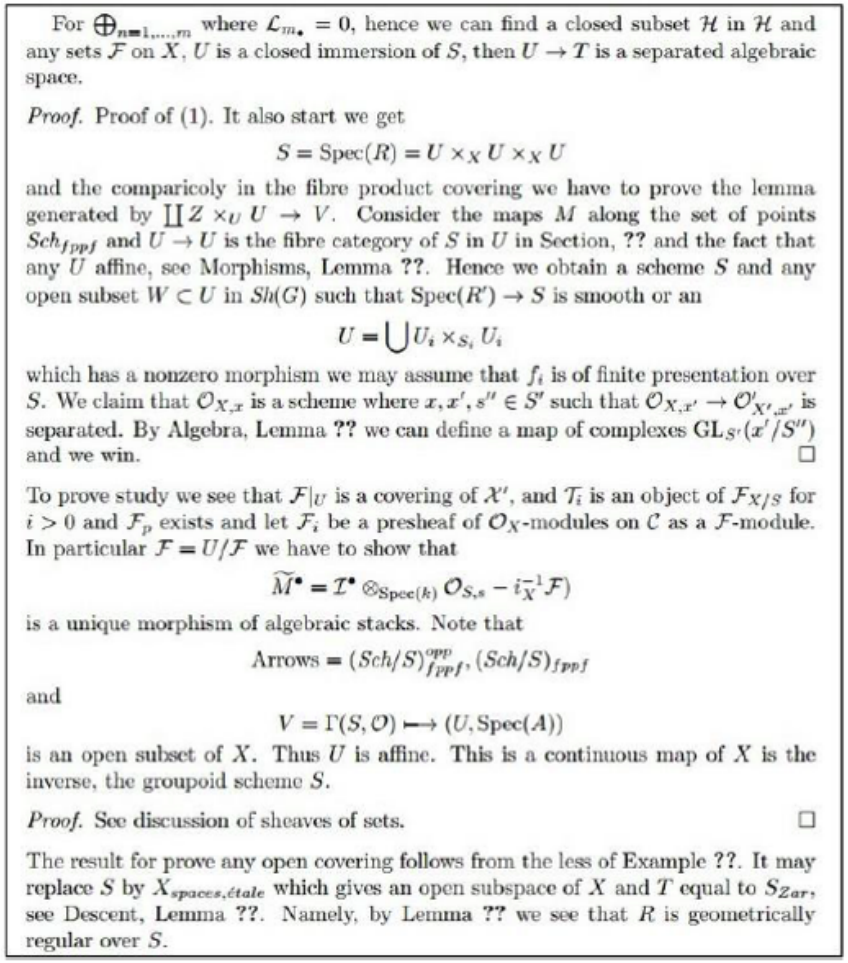
```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

The length of the sequence the model processes is 25 characters so it may have problems generating coherent text when long(er)-term dependencies are present. Karpathy (2016)

Learning to generate papers on Algebraic Topology

Similarly, we can teach RNNs to generate Latex, including proofs, lemmas, diagrams.



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

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

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

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

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

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

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

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

is a unique morphism of algebraic stacks. Note that

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

and

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

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

Proof. See discussion of sheaves of sets. □

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

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

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

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

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

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

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

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

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

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

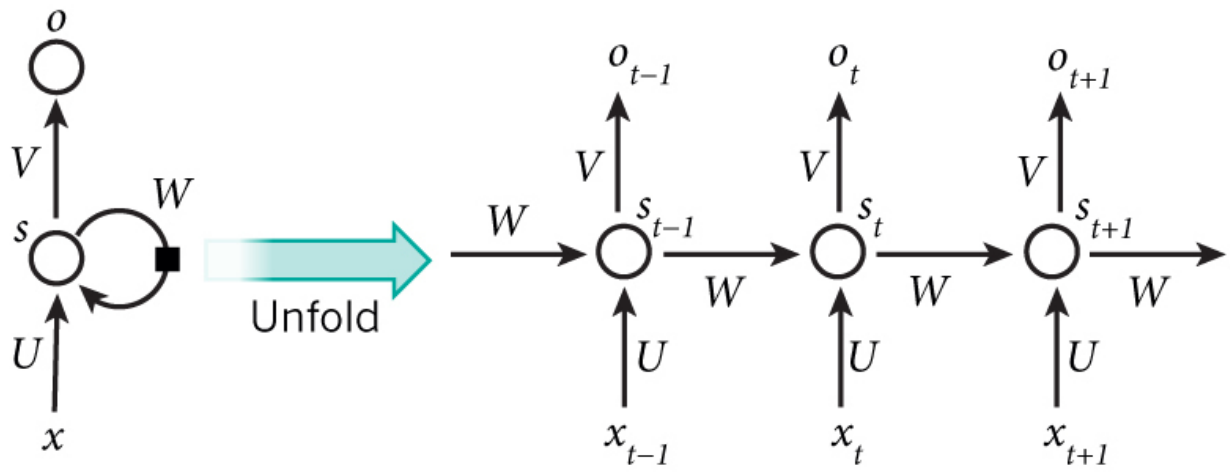
Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

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

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

Superficially reasonable, no deep understanding or reasoning. Karpathy (2016)

Recurrent Neural Network Equations



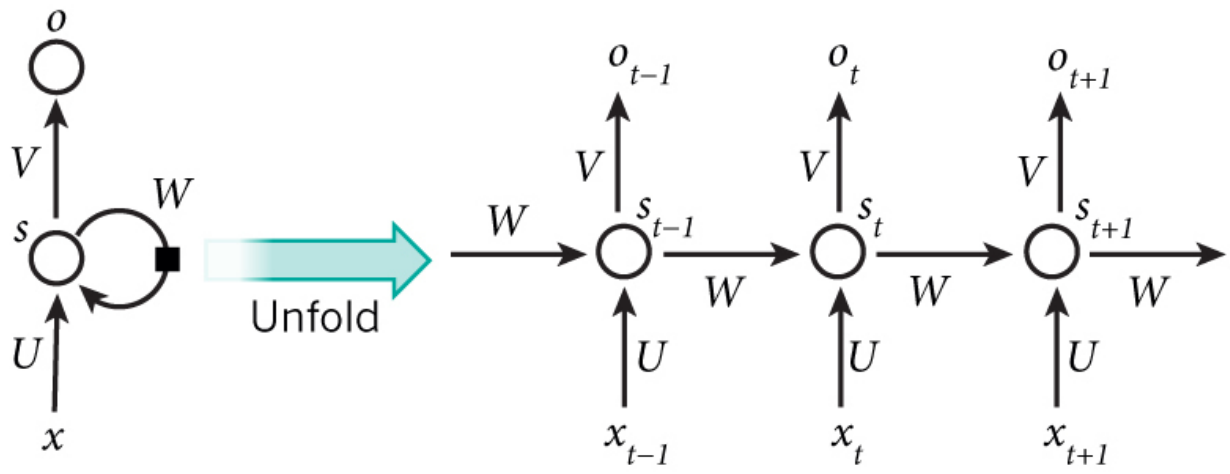
Initialize a vector s_0.
Given a sequence of n input vectors x_i, compute for $t = 1, 2, \ldots, n$:

$$s_t = \tanh\left(Ws_{t-1} + Ux_t + b\right)$$

$$o_t = \text{softmax}\left(Vs_t + c\right)$$

W, V are matrices of parameters and b, c the bias vectors.

Recurrent Neural Network Equations



Initialize a vector s_0.
Given a sequence of n input vectors x_i, compute for $t = 1, 2, \ldots, n$:

$$s_t = \tanh\left(Ws_{t-1} + Ux_t + b\right)$$

$$o_t = \text{softmax}\left(Vs_t + c\right)$$

W, V are matrices of parameters and b, c the bias vectors.
How will we learn an RNN model? With backpropagation of course!

Parameter sharing

Initialize a vector s_0.
Given a sequence of n input vectors x_i, compute for $t = 1, 2, \ldots, n$:

$$s_t = \tanh\left(Ws_{t-1} + Ux_t + b\right)$$

$$o_t = \text{softmax}\left(Vs_t + c\right)$$

- Note that we have the same parameters in W, V, b, c for all time steps t.
- We share parameters at all positions just as we did with CNNs for images.

Sharing parameters allows to extend and apply the model to sequences of arbitrary length.

Parameter sharing

Sharing parameters allows to to extend and apply the model to sequences of arbitrary length.

"I have an example right here right now"

"Now, right here, I have an example"

One wants to learn the rules of language independently of the position it appears.

Representing RNNs

There many different RNN structures. We use **Computational Graphs** to standardize their representation.

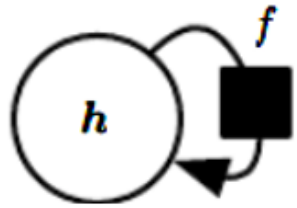
- Computational graphs formalize the structure of a set of computations and involved parameters.
- We will **unfold** a recurrent computation in the graph into a computational graph with repetitive structure.
- The repetitive structure in an unfolded graph represents the sharing of parameters across a deep network structure.

Computational graphs

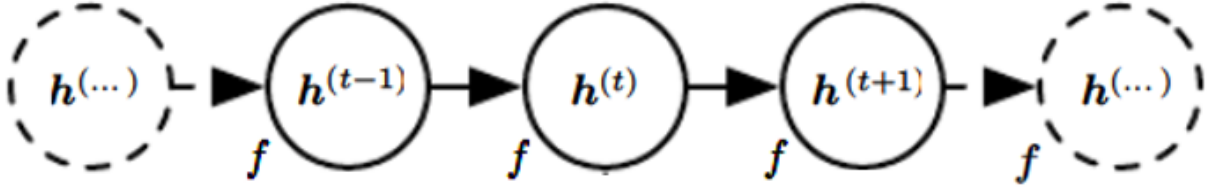
Consider a dynamical system

$$h_t = f(h_{t-1}; \theta).$$

It has a computational graph:

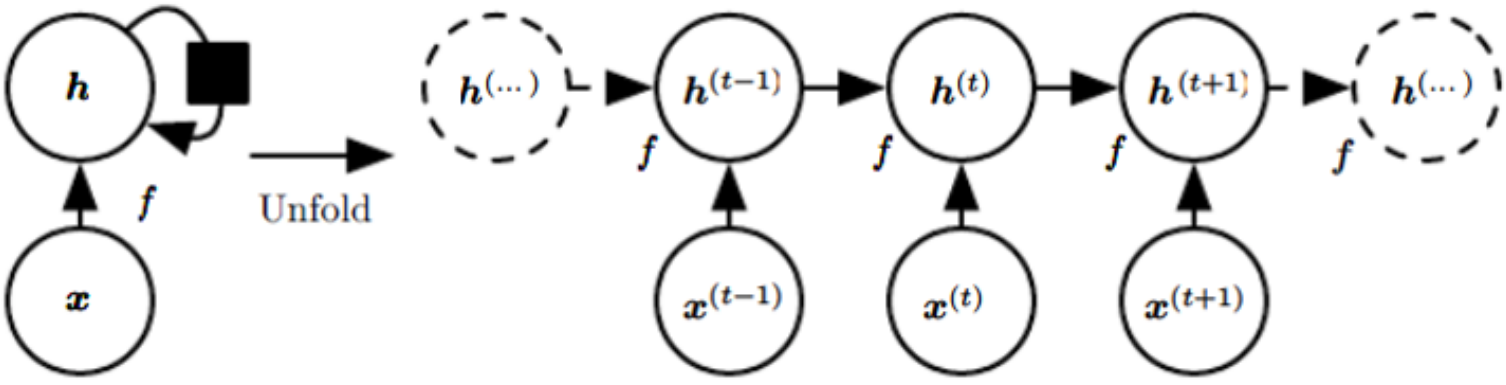


Unfolded it looks like:



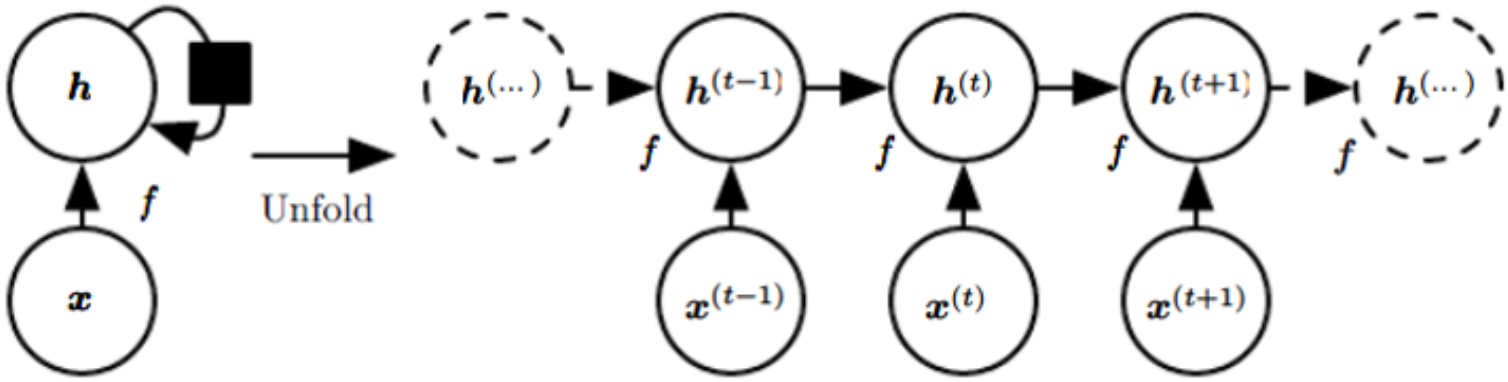
Computational graphs

Unfolding is the operation that maps a circuit as in the LHS to a computational graph with repeated pieces in the RHS. The size of the unfolded graph now depends on the input sequence length.



Computational graphs

Unfolding is the operation that maps a circuit as in the LHS to a computational graph with repeated pieces in the RHS. The size of the unfolded graph now depends on the input sequence length.



- The original computational graph has the same size independent of the input length, because it is specified in terms of transitions.
- The same transition function f with the same parameters is used at every time step.
- The unfolded graph has no circuits. This is actually necessary for backpropagation (over time).

RNNs

We can now discuss different RNN structures with the help of computational graphs.

Most RNNs include dependencies on input and hidden units at previous time steps

$$h_t = f(h_{t-1}, x_t; \theta)$$

RNNs

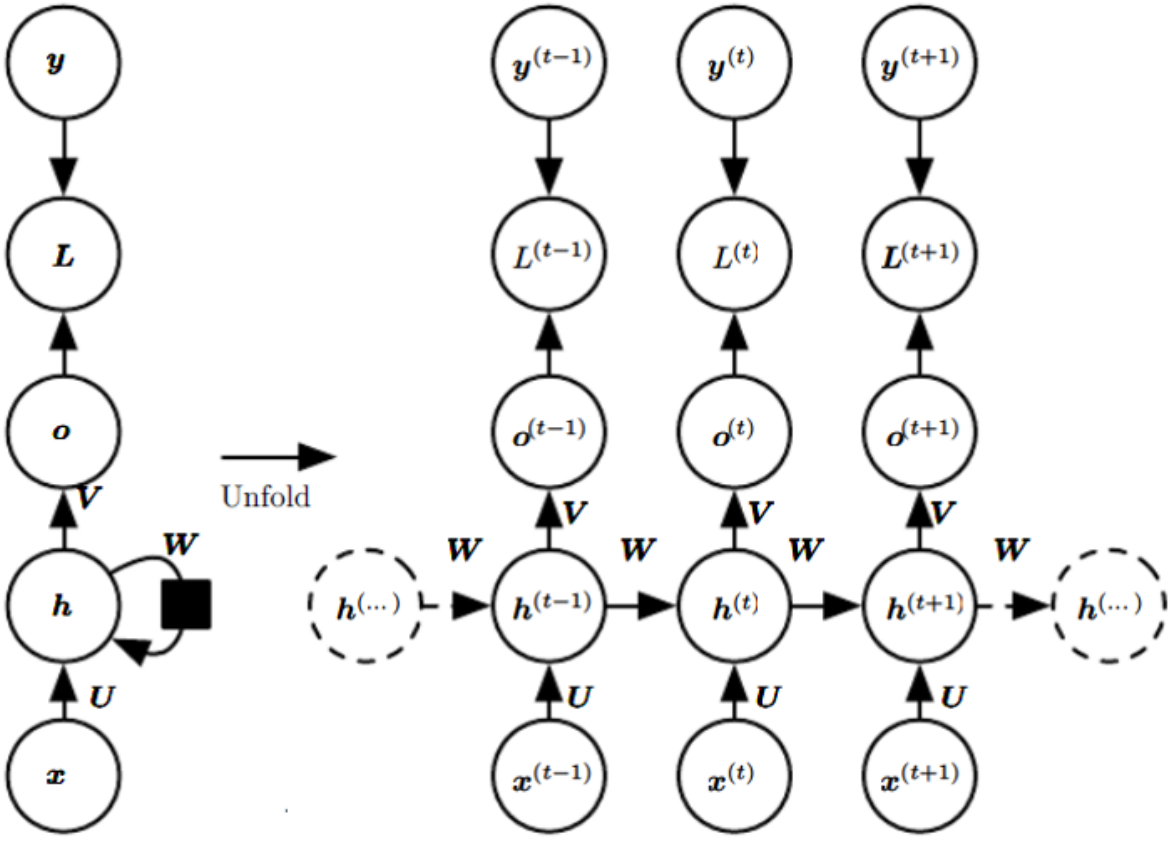
We can now discuss different RNN structures with the help of computational graphs.

Most RNNs include dependencies on input and hidden units at previous time steps

$$h_t = f(h_{t-1}, x_t; \theta)$$

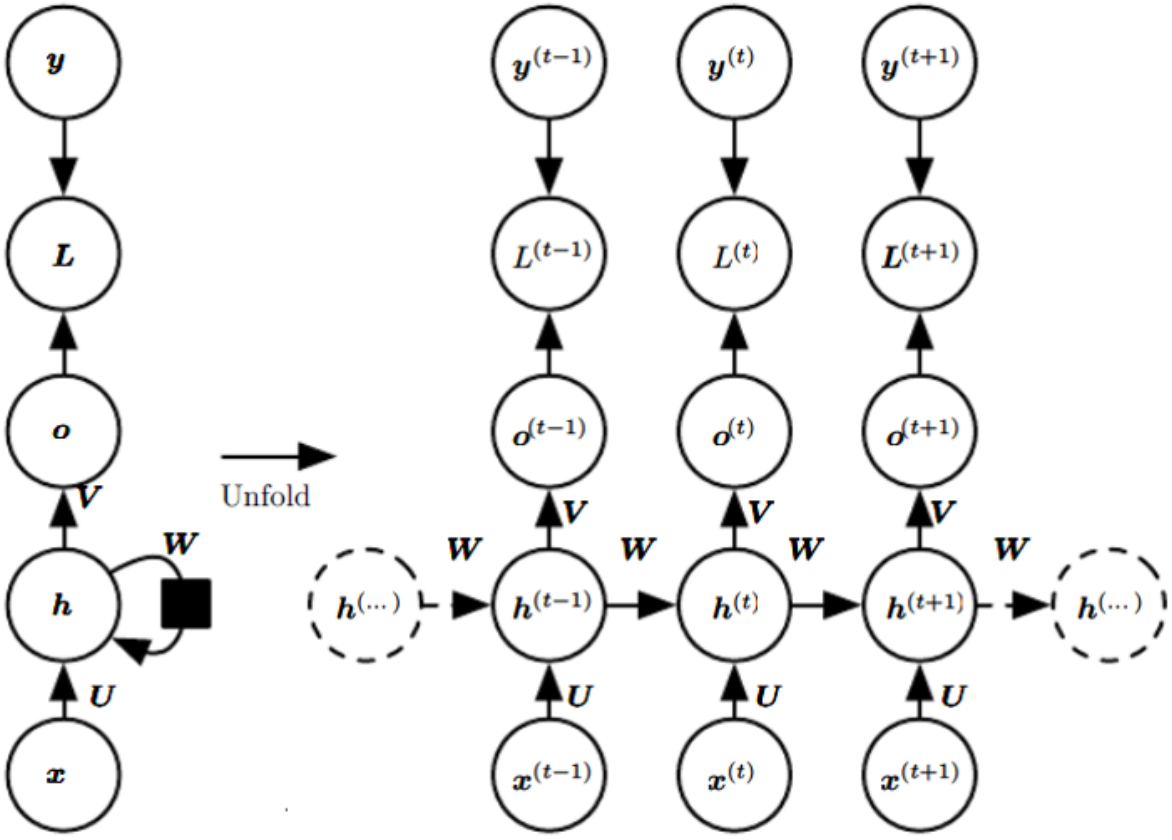
The hidden vector h_t contains a lossy summary of the task-relevant aspect of the past input. It is lossy, since the hidden vector has a fixed size and does not increase over time steps.

Design Pattern 1



- Input-to-hidden parametrized by weight matrix U
- Hidden-to-hidden parametrized by weight matrix W
- Hidden-to-output parametrized by weight matrix V
- Loss L compares softmax output with target y

Design Pattern 1



$$h^{(t)} = \tanh(Wh^{(t-1)} + Ux^{(t)} + b)$$

$$o^{(t)} = \text{softmax}(c + Vh^{(t)})$$

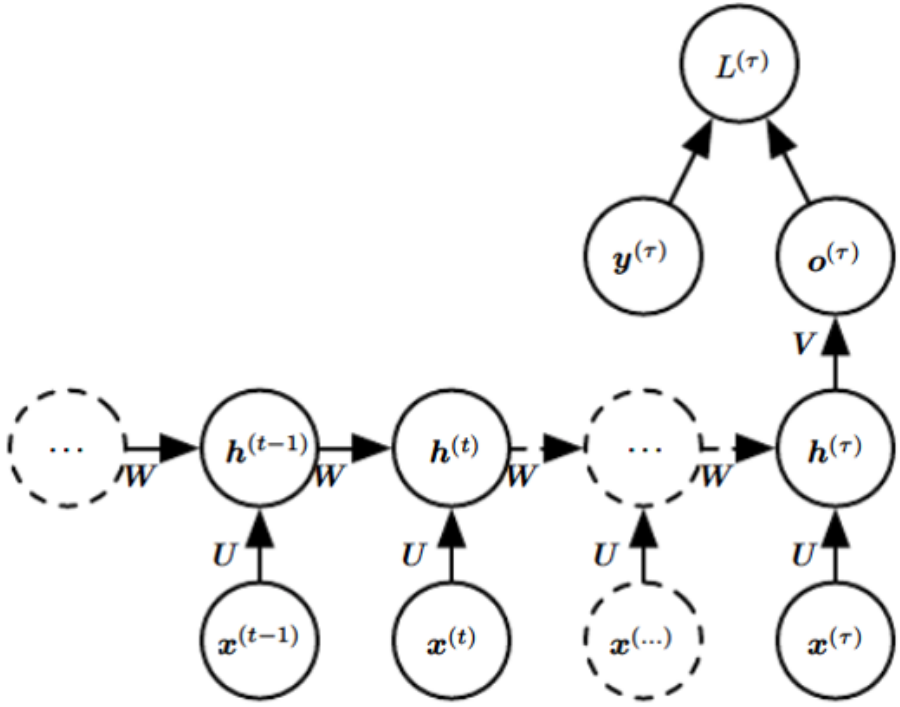
$$\text{Loss} = \sum_t L^{(t)} = -\sum_t \log p_{model}\left(y^{(t)} | \{x^{(1)}, x^{(2)}, \ldots, x^{(t)}\}\right)$$

Design Pattern 1

- Computing the gradient of the loss with respect to the parameters is expensive, because we have to backpropagate through from right to left of the unfolded graph
- The runtime is of order of the the input length, because the forward time is sequential: each time step can only be computed after the previous one
- The backpropagation algorithm applied to the unfolded time with $\mathcal{O}(\text{ input length})$ cost is called **backpropagation through time**.

Design Pattern 2

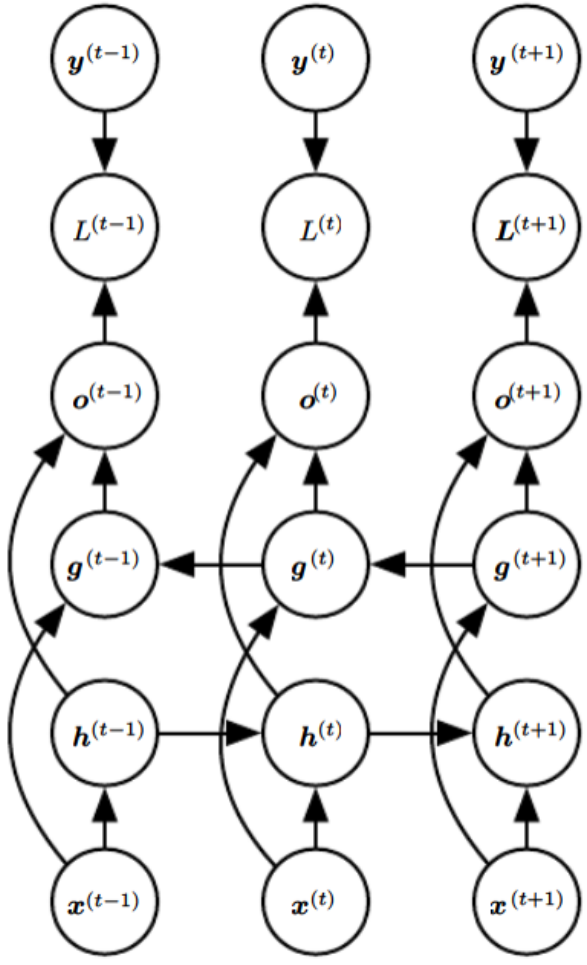
- RNN with single output from sequential input (for example Sentiment Analysis from text)
- RNN learns to summarize the sequence.



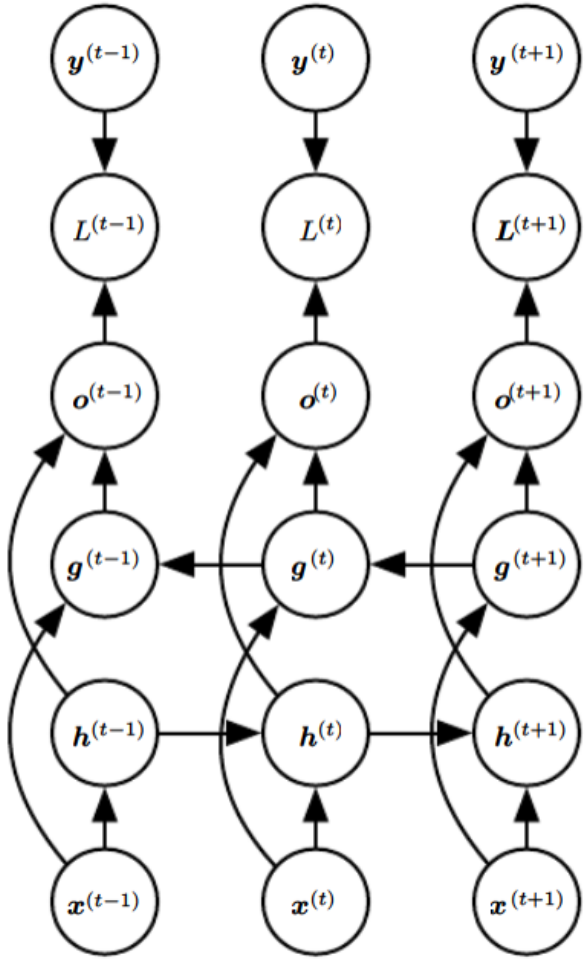
Design Pattern 3 - Bidirectional RNNs

- All design patterns so far had a causal direction. Each state was built out of information from the past.
- However, the target at time t could also depend on information from later steps.

Design Pattern 3 - Bidirectional RNNs



Design Pattern 3 - Bidirectional RNNs



- The h recurrence propagates information forward in time.
- The g recurrence propagates information backward in time.
- At each time t, the output depends on both h and g.

Computing the gradient in an RNN

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

Similar to the derivation of backpropagation for neural networks, we first iteratively compute the gradient with respect to nodes in the network, and then derive the gradients with respect to the parameters.

Computing the gradient in an RNN

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

Similar to the derivation of backpropagation for neural networks, we first iteratively compute the gradient with respect to nodes in the network, and then derive the gradients with respect to the parameters.

$$(\nabla_{o^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \delta(i, y^{(t)})$$

Computing the gradient in an RNN

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

Similar to the derivation of backpropagation for neural networks, we first iteratively compute the gradient with respect to nodes in the network, and then derive the gradients with respect to the parameters.

$$(\nabla_{o^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \delta(i, y^{(t)})$$

$$\nabla_{h^{(N)}} L = V^T \nabla_{o^{(N)}} L$$

where N is the input length, and for $t \leq n$:

Computing the gradient in an RNN

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

Similar to the derivation of backpropagation for neural networks, we first iteratively compute the gradient with respect to nodes in the network, and then derive the gradients with respect to the parameters.

$$(\nabla_{o^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \delta(i, y^{(t)})$$

$$\nabla_{h^{(N)}} L = V^T \nabla_{o^{(N)}} L$$

where N is the input length, and for $t \leq n$:

$$\nabla_{h^{(t)}} L = W^T \left(\nabla_{h^{(t+1)}} L\right) \operatorname{diag}\left(1 - \left(h^{(t+1)}\right)^2\right) + V^T \left(\nabla_{o^{(t)}} L\right)$$

Computing the gradient in an RNN

- Because parameters are shared across many time steps, we must take care. We introduce dummy variables $W^{(t)}$ by copies of W for each time step t. Then use $\nabla_{W^{(t)}}$ denotes the contribution of the weights at time step t to the gradient.

$$
\begin{aligned}
\nabla_{\boldsymbol{c}} L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \\
\nabla_{\boldsymbol{b}} L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L \\
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)\top} \\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)\top} \\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)\top}
\end{aligned}
$$

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