University of Colorado **Boulder**

CSCI 5622: Machine Learning

Lecture 7

Overview

- Deep neural networks
  - Motivation & Challenges
  - (Supervised) fine-tuning
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks
  - Encoder-decoder (or sequence-to-sequence) networks

[Contents of the following slides have been summarized from the NIPS 2010 & CVPR 2012 Deep Learning Tutorials, and the Stanford CS231 class by Drs. Li, Johnson, & Yeung]

Overview

- Deep neural networks
  - Motivation & Challenges
  - (Supervised) fine-tuning
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks
  - Encoder-decoder (or sequence-to-sequence) networks

Deep neural networks: Motivation

Traditional recognition

pixels → classifier → "bus"?

edges → classifier → "bus"?

SIFT/HOG

edges → histogram → classifier → "bus"?

edges → histogram → K-means/ sparse code → classifier → "bus"?

But what's next?

shallower

deeper

Deep neural networks: Motivation

Deep Learning

Specialized components, domain knowledge required

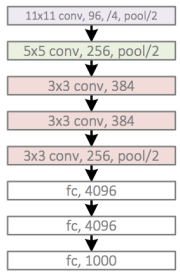
Generic components ("layers"), less domain knowledge

Repeat elementary layers => Going deeper

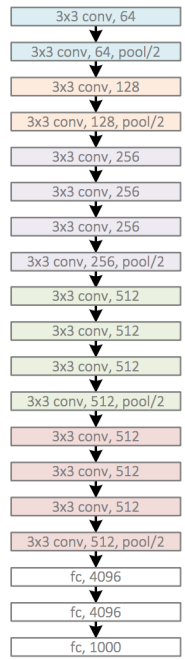
- End-to-end learning
- Richer solution space

Deep neural networks: Motivation

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

Deep neural networks: Motivation

Why many layers?

- Deep Representations might allow for a hierarchy or representation
  - Non-local generalization
  - Comprehensibility
- Multiple levels of latent variables allow combinatorial sharing of statistical strength
- Deep architectures work well (vision, audio, NLP, etc.)!

Deep neural networks: Motivation

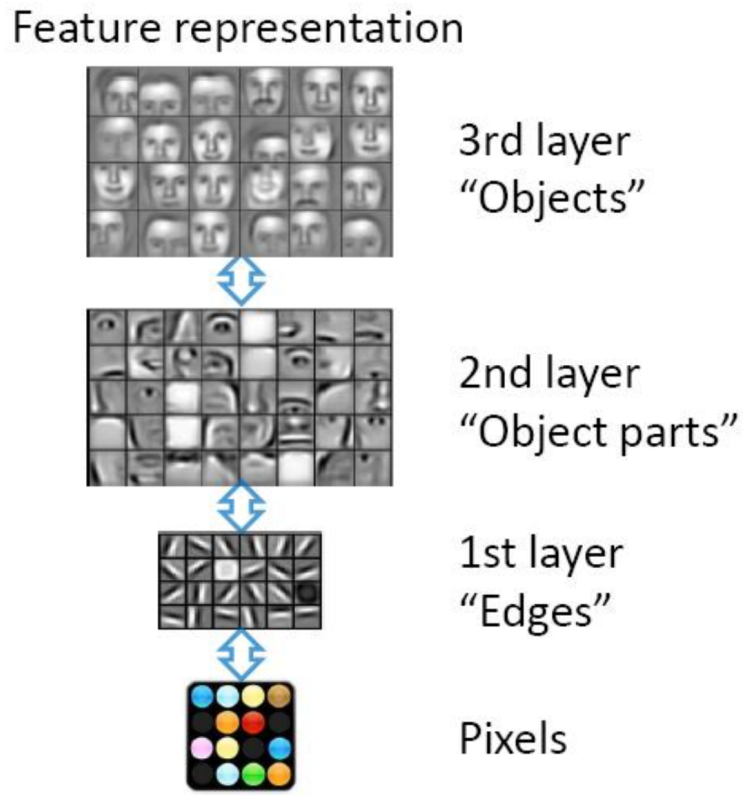
Key ideas of (deep) neural networks

- Learn features from data
- Use differentiable functions that produce features efficiently
- End-to-end learning: no distinction between feature extractor and classifier
- "Deep" architectures: cascade of simpler non-linear modules

Deep neural networks: Motivation

Different levels of abstraction: Hierarchical learning

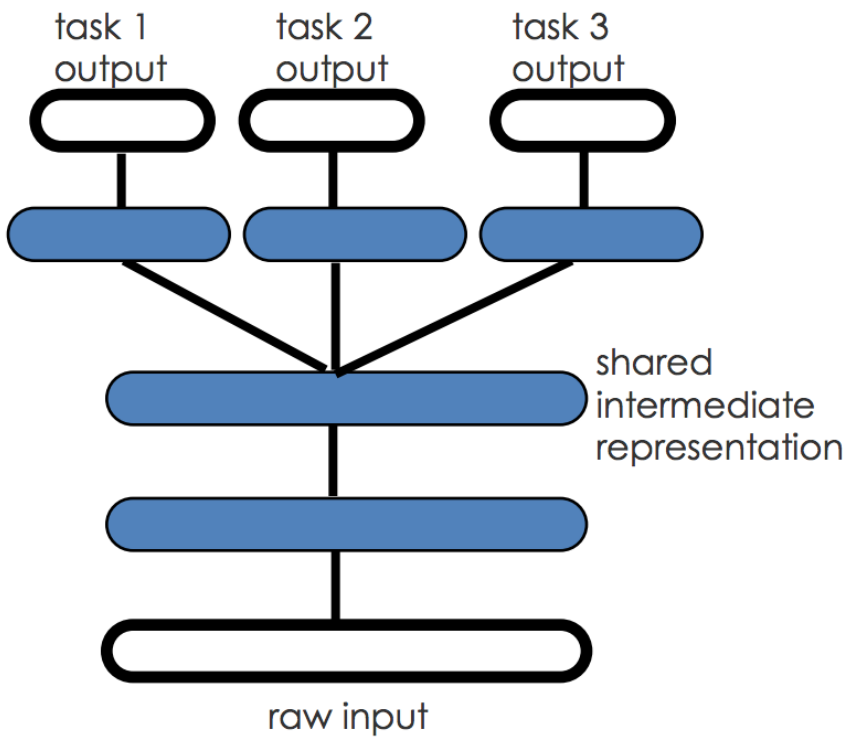
- Natural progression from low level to high level structure as seen in natural complexity
- Easier to monitor what is being learnt and to guide the machine to better subspaces
- A good lower level representation can be used for many distinct tasks



Deep neural networks: Motivation

Shared low-level representations

- Multi-task learning
- Unsupervised training



Deep neural networks: Challenges

High memory requirements

- Memory is used to store input data, weight parameters and activations as an input propagates through the network
- Activations from a forward pass must be retained until they can be used to calculate the error gradients in the backwards pass
- Example: 50-layer neural network
  - 26 million weight parameters, 16 million activations in the forward pass
  - 168MB memory (assuming 32-bit float)

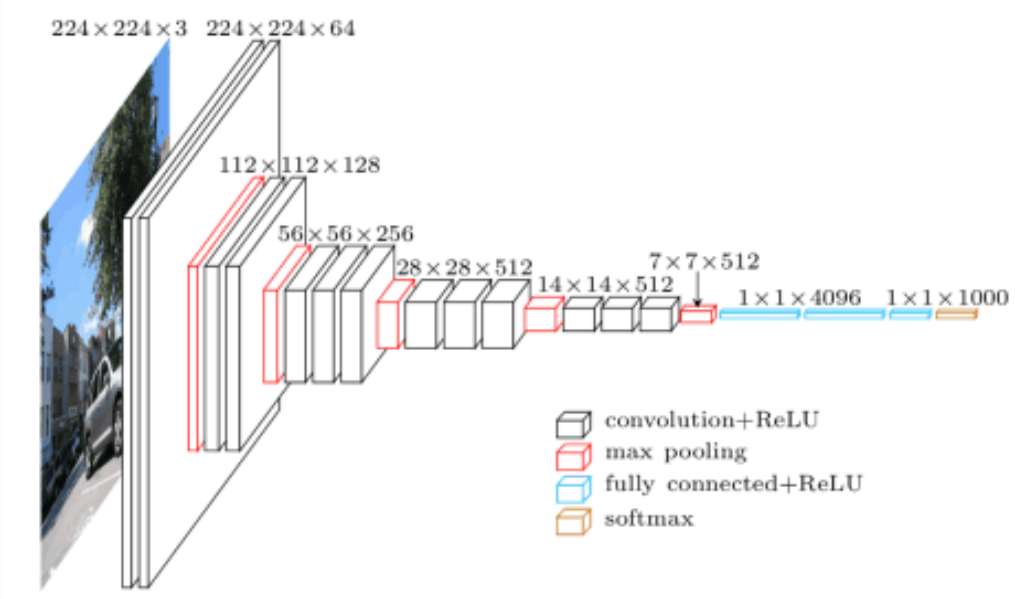
Parallelize computations with GPU (graphics processing units)

Overview

- Deep neural networks
  - Motivation & Challenges
  - (Supervised) fine-tuning
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks
  - Encoder-decoder (or sequence-to-sequence) networks

Deep neural networks: Fine-tuning

- Taking advantage of labelled data from large (publicly available) datasets, e.g., VGG16
- Tweak the parameters of an already trained network so that it adapts to the new task at hand
- Initial layers $\rightarrow$ learn general features
- Last layers $\rightarrow$ learn features more specific to the task of interest
- Fine-tuning freezes the first layers, and relearns weights from the last



Overview

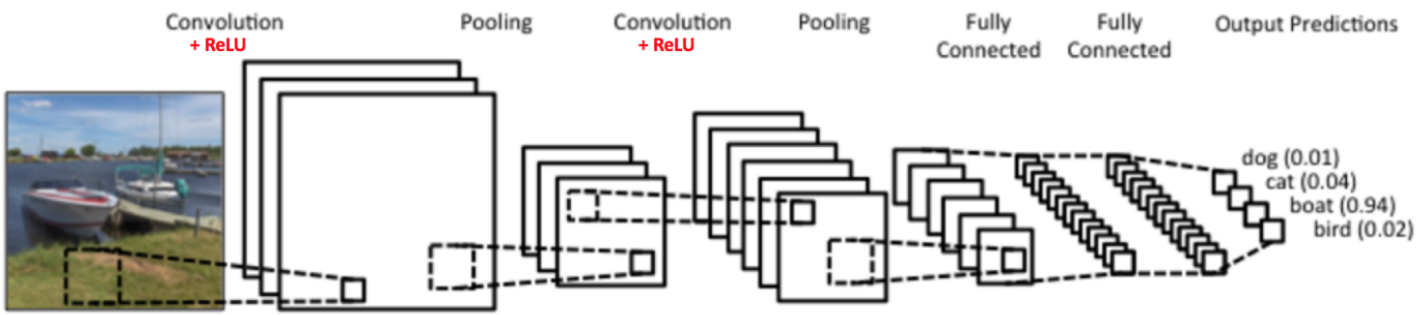
- Deep neural networks
  - Motivation & Challenges
  - Unsupervised pretraining: Deep belief networks & autoencoders
  - (Supervised) fine-tuning
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks
  - Encoder-decoder (or sequence-to-sequence) networks

Overview

- Deep neural networks
  - Motivation & Challenges
  - (Supervised) fine-tuning
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks
  - Encoder-decoder (or sequence-to-sequence) networks

Convolutional neural networks

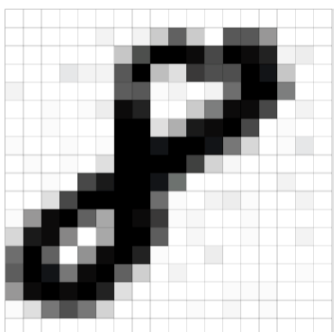
- Similar to regular neural networks
  - made up of neurons, each with an input and an activation function
  - have weights and biases to be learned
  - have a loss function on the last (fully-connected) layer
- Explicit assumption that the inputs are images
  - vastly reduce the amount of parameters in the network



Convolutional neural networks

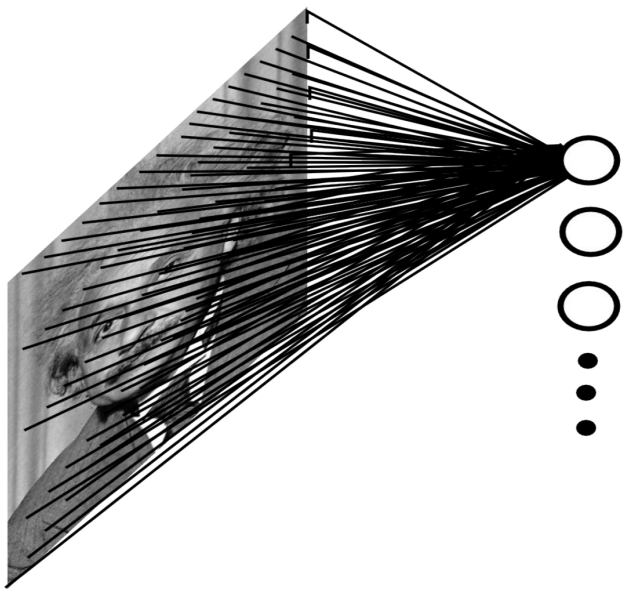
Image representation

- Grayscale image (1-channel)
  - 2d-matrix
  - each pixel ranges from 0 to 255 - 0: white, 255: black
- Color image (3-channel, RGB)
  - three 2d-matrices stacked over each other
  - each with pixel values ranging between 0 and 255



Convolutional neural networks

A **fully** connected neural network with image input

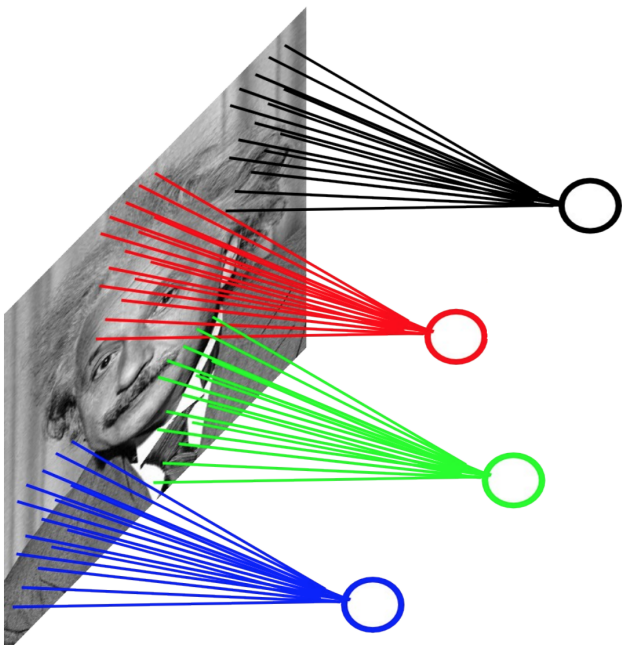


- 1000×1000 image, 1M hidden units $\rightarrow 10^{12}$ parameters
- Since spatial correlation is local, we can significantly simplify this

Convolutional neural networks

Idea 1: Convolution

A **locally** connected neural network with image input

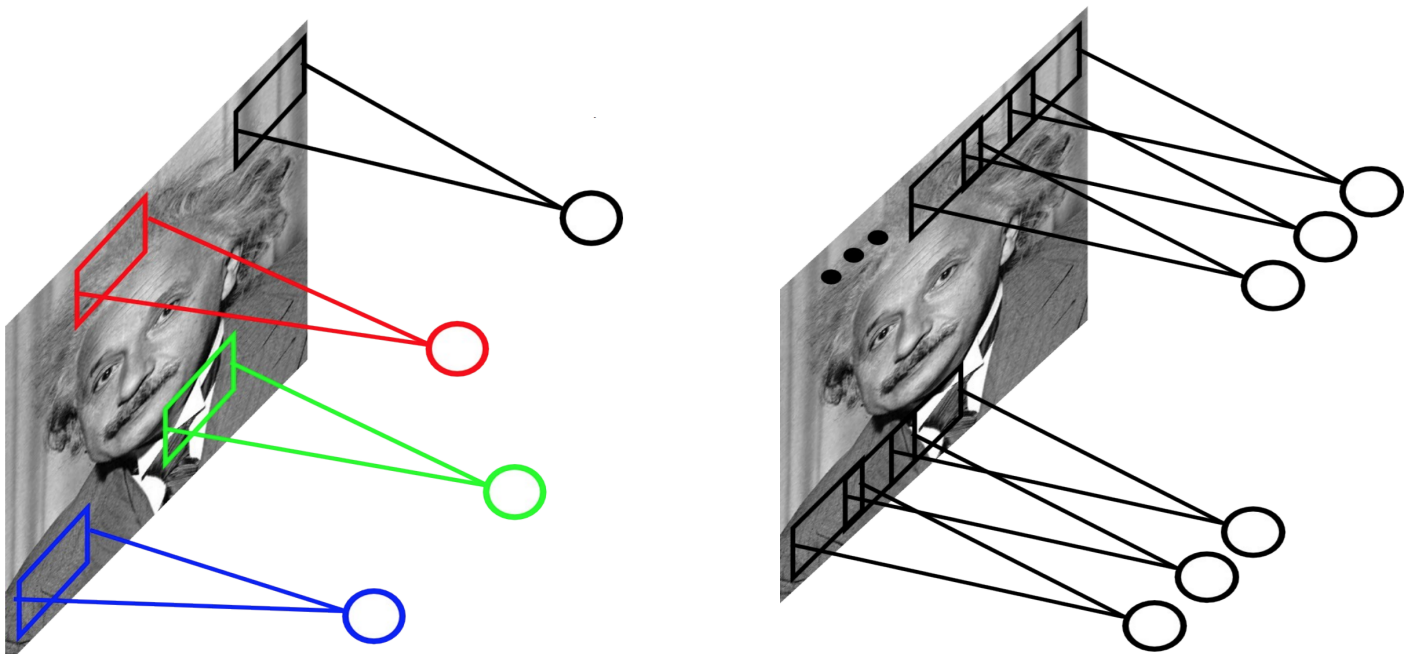


- 1000×1000 image, 1M hidden units, 10×10 filter size $\rightarrow$ 10^8 parameters
- Since spatial correlation is local, we can significantly simplify this

Convolutional neural networks

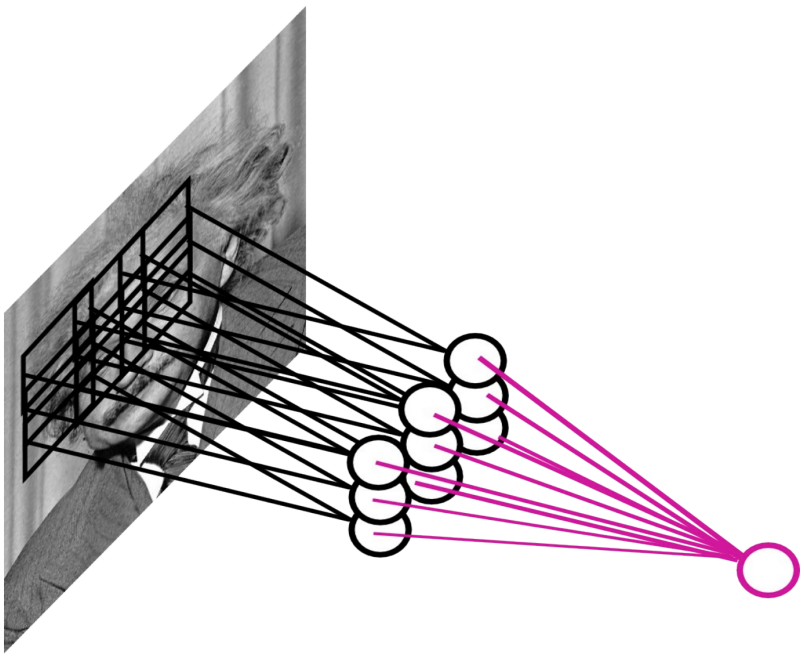
Idea 2: Weight sharing

- Stationarity: Statistics are similar at different locations
- Share the same parameters across different locations



Convolutional neural networks

Idea 3: Max-pooling



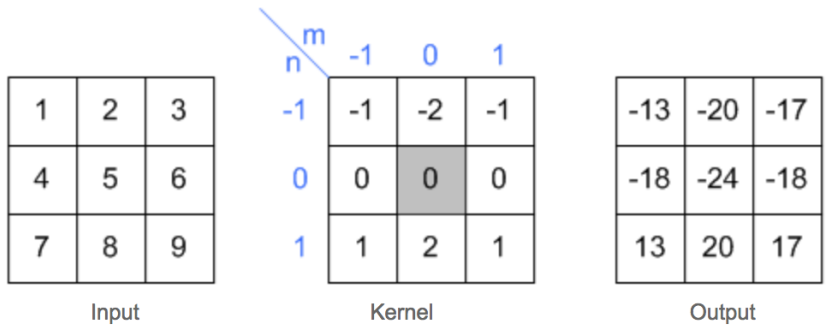
- Let us assume filter is an "eye" detector
- How can we make the detection robust to the exact location of the eye?
- By pooling (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features

Convolutional neural networks: The convolution operation

Example of 2D Convolution

- Convolution is the mathematical operation that implements filtering
- Given an input image $x[m, n]$ and an impulse response $h[m, n]$ (filter or kernel), the convolution output can be written as

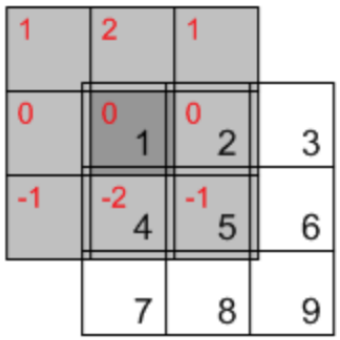
$$y[m, n] = x[m, n] * h[m, n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i, j] h[m - i, n - j]$$



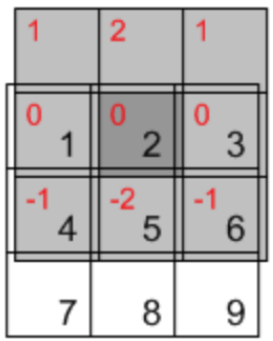
http://www.songho.ca/dsp/convolution/convolution2d_example.html

Convolutional neural networks: The convolution operation

Example of 2D Convolution



$$
\begin{aligned}
y[0,0] = {} & x[-1,-1]\cdot h[1,1] + x[0,-1]\cdot h[0,1] + x[1,-1]\cdot h[-1,1] \\
& + x[-1,0]\cdot h[1,0] + x[0,0]\cdot h[0,0] + x[1,0]\cdot h[-1,0] \\
& + x[-1,1]\cdot h[1,-1] + x[0,1]\cdot h[0,-1] + x[1,1]\cdot h[-1,-1] \\
= {} & 0\cdot 1 + 0\cdot 2 + 0\cdot 1 + 0\cdot 0 + 1\cdot 0 + 2\cdot 0 + 0\cdot(-1) + 4\cdot(-2) + 5\cdot(-1) = -13
\end{aligned}
$$



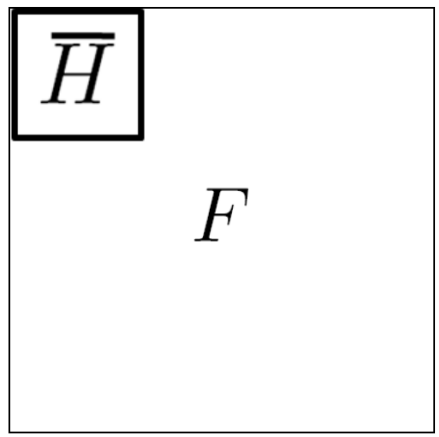
$$
\begin{aligned}
y[1,0] = {} & x[0,-1]\cdot h[1,1] + x[1,-1]\cdot h[0,1] + x[2,-1]\cdot h[-1,1] \\
& + x[0,0]\cdot h[1,0] + x[1,0]\cdot h[0,0] + x[2,0]\cdot h[-1,0] \\
& + x[0,1]\cdot h[1,-1] + x[1,1]\cdot h[0,-1] + x[2,1]\cdot h[-1,-1] \\
= {} & 0\cdot 1 + 0\cdot 2 + 0\cdot 1 + 1\cdot 0 + 2\cdot 0 + 3\cdot 0 + 4\cdot(-1) + 5\cdot(-2) + 6\cdot(-1) = -20
\end{aligned}
$$

`http://www.songho.ca/dsp/convolution/convolution2d_example.html` 3D convolution:

`https://cs231n.github.io/assets/conv-demo/index.html`

Convolutional neural networks: The convolution operation

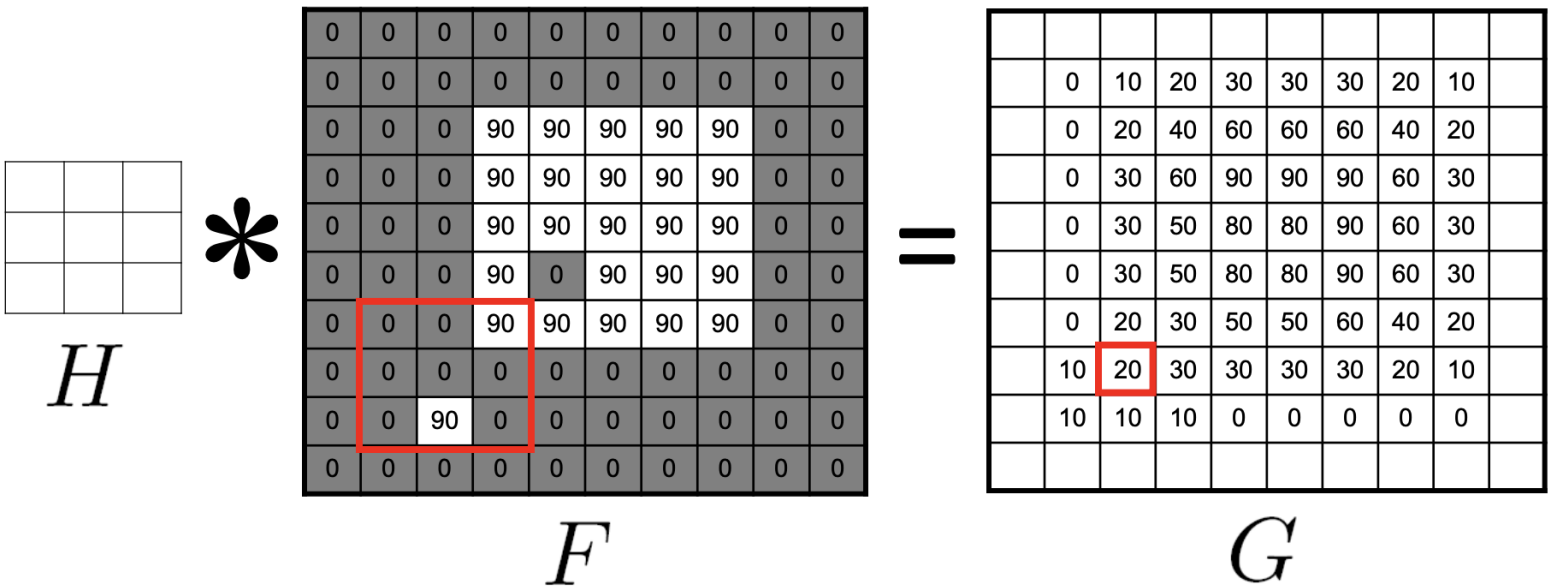
Convolution operation



Convolutional neural networks: Example

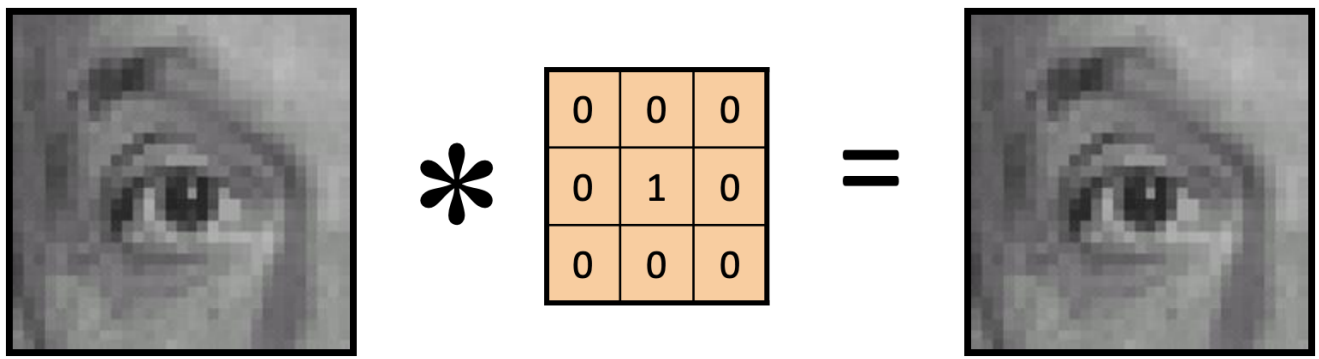
Convolutional neural networks: The convolution operation

Mean filtering



Convolutional neural networks: The convolution operation

Linear filters: Example

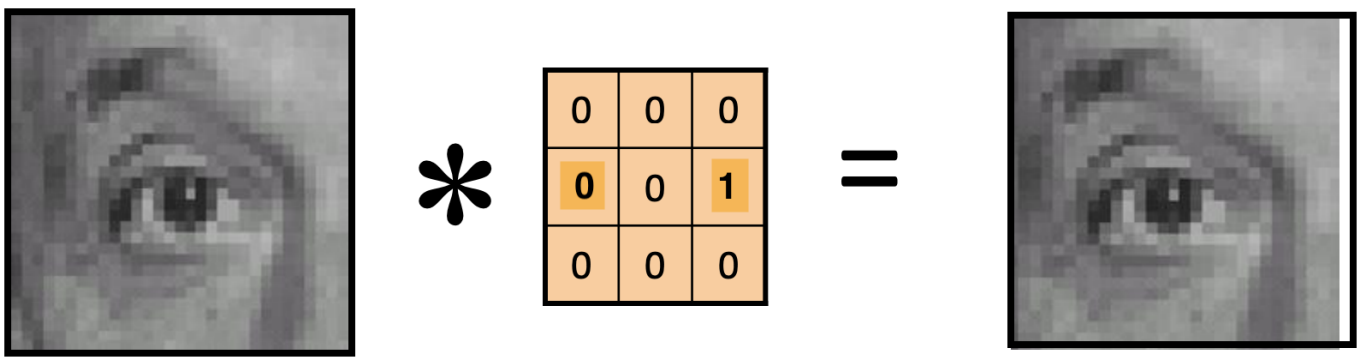


Original

Identical image

Convolutional neural networks: The convolution operation

Linear filters: Example

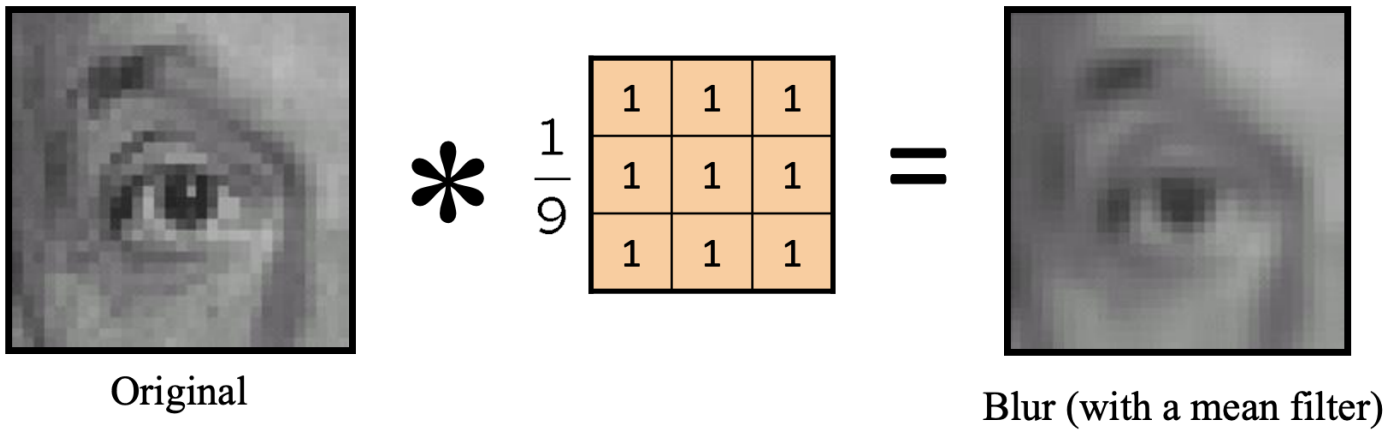


Original

Shifted left
By 1 pixel

Convolutional neural networks: The convolution operation

Linear filters: Example

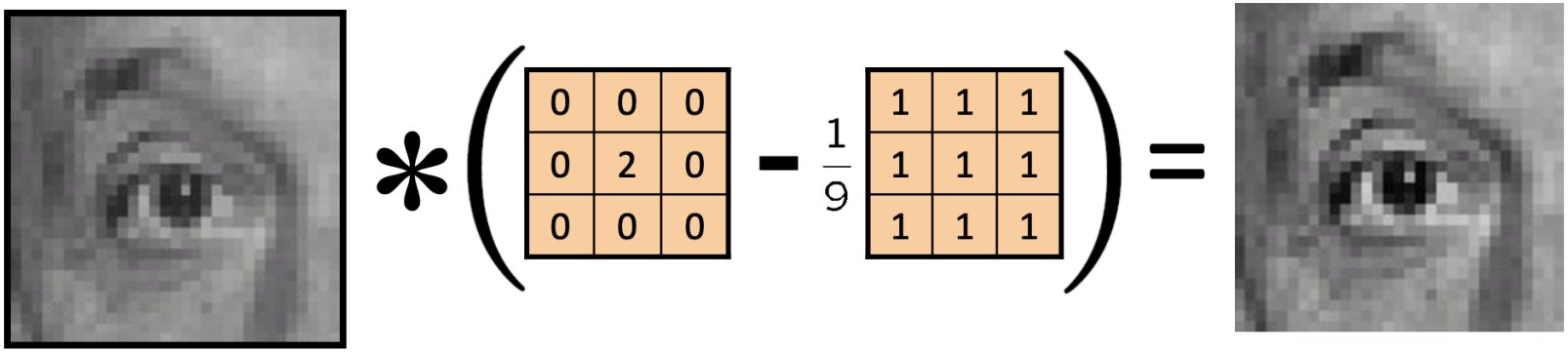


Original

Blur (with a mean filter)

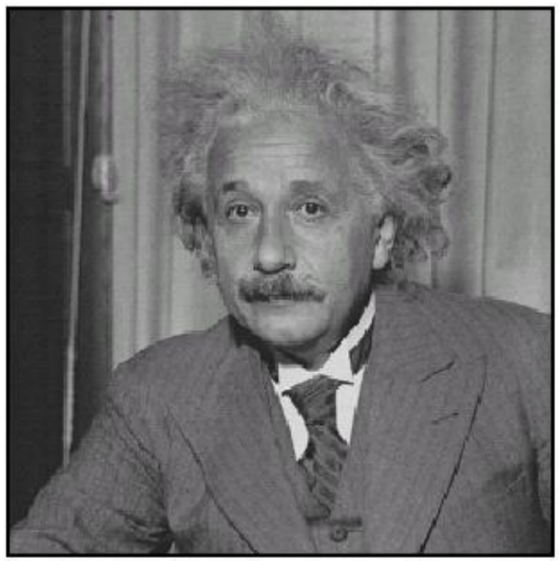
Convolutional neural networks: The convolution operation

Linear filters: Example

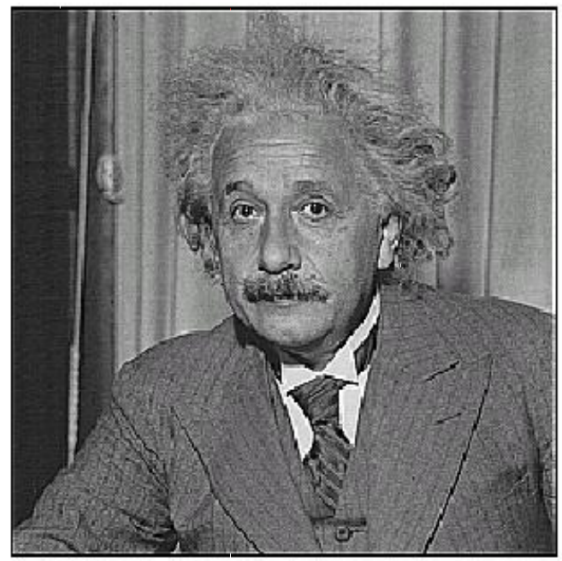


Convolutional neural networks: The convolution operation

Linear filters: Example



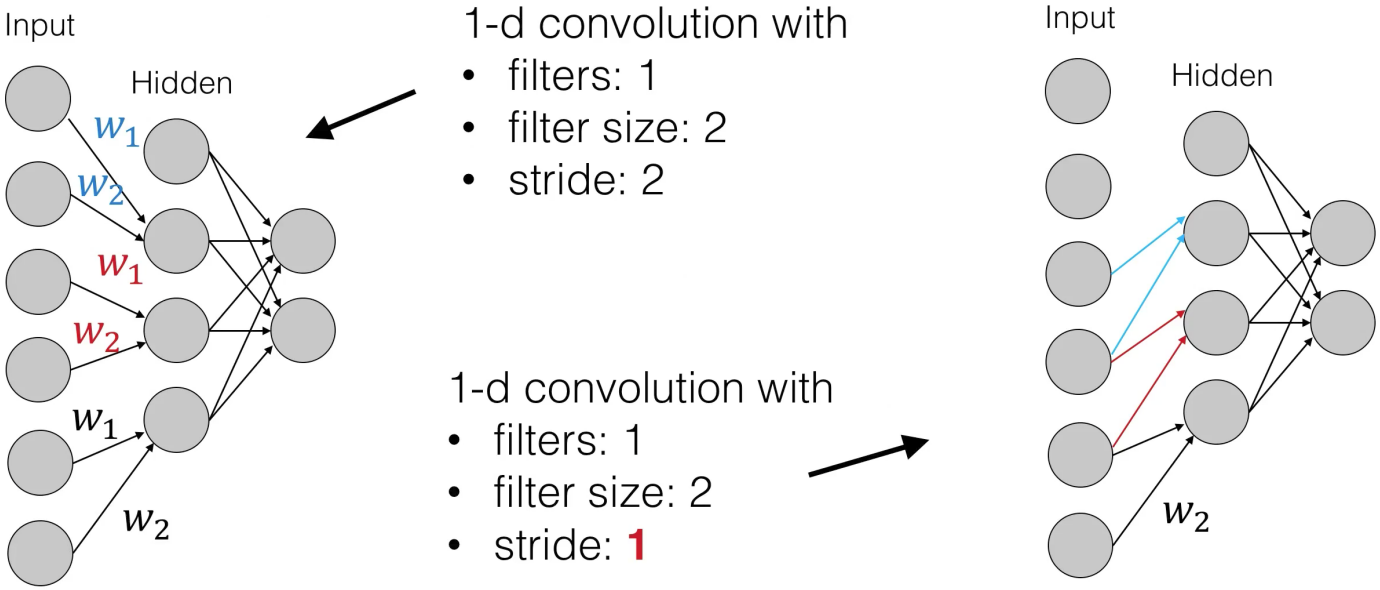
before



after

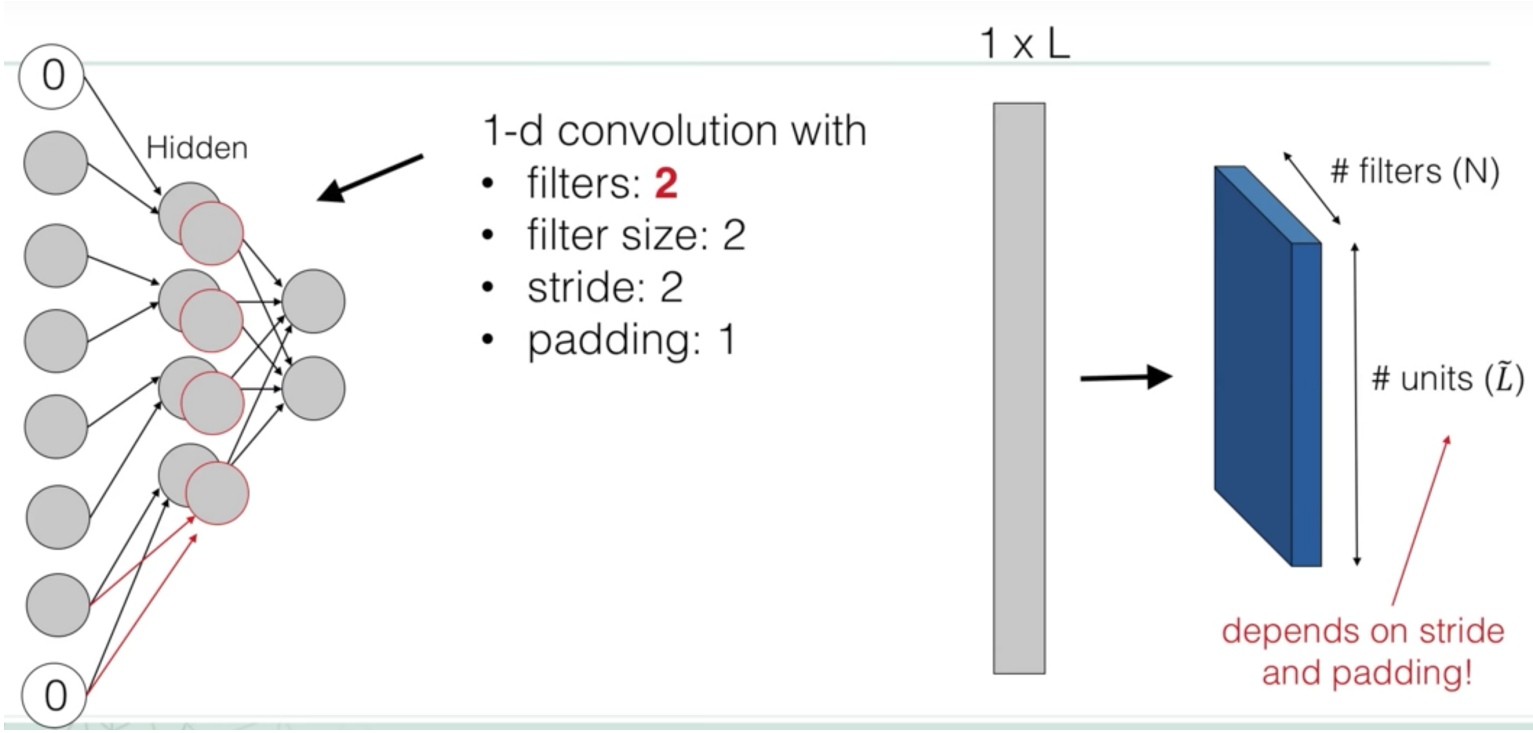
Convolutional neural networks: The convolution operation

Image 1d-convolution



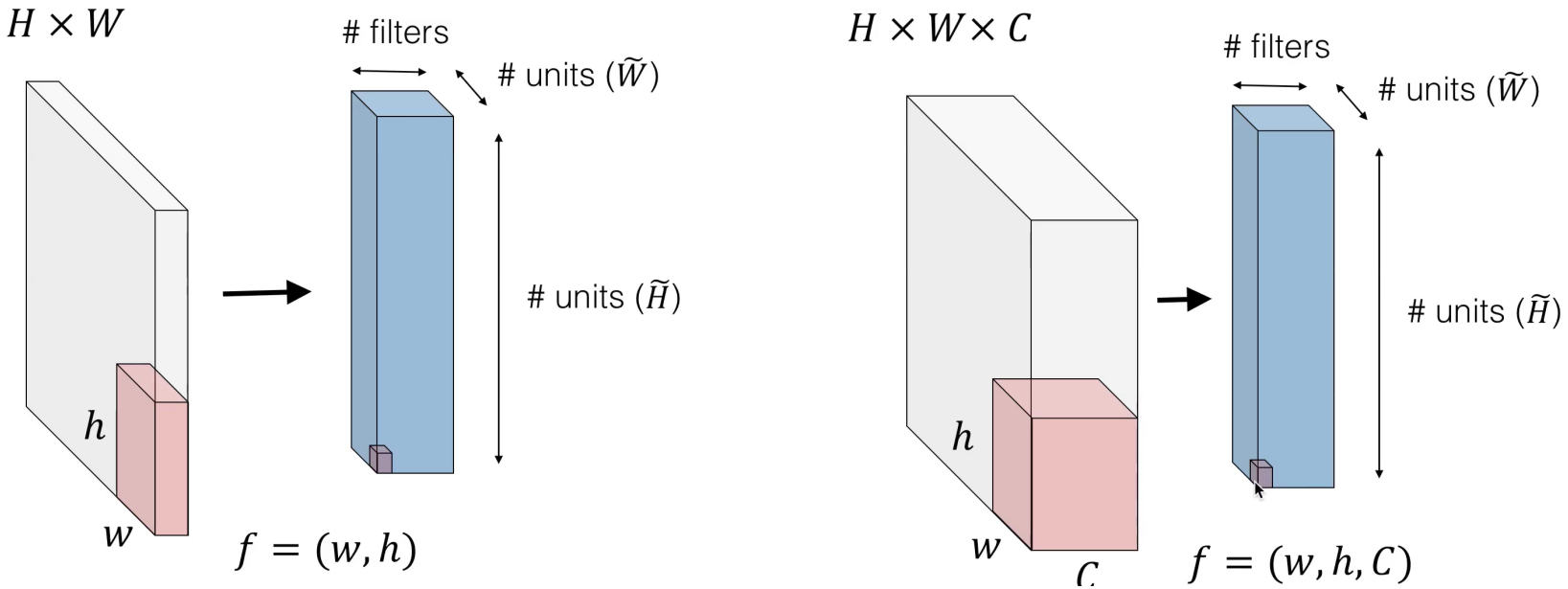
Convolutional neural networks: The convolution operation

Image 1d-convolution



Convolutional neural networks: The convolution operation

Image 2d-convolution



Also check:

`http://cs231n.github.io/assets/conv-demo/index.html`

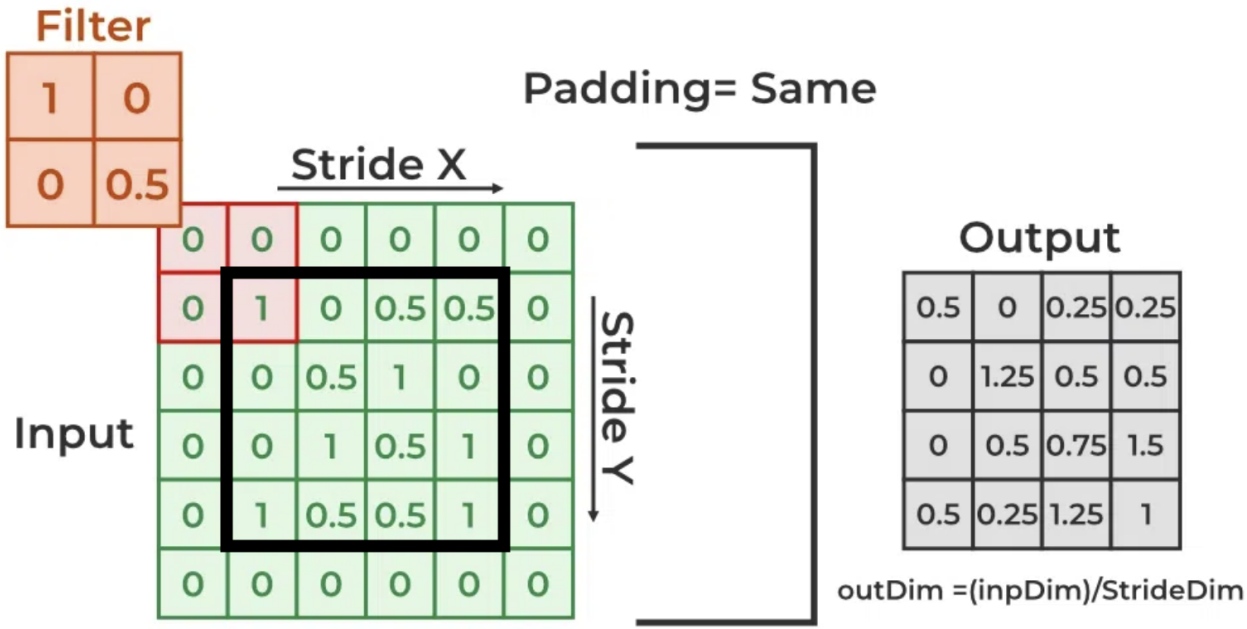
`https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/` (figure 6)

Convolutional neural networks: The convolution operation

Image 2d-convolution hyperparameters

- Depth: the number of filters we use for the convolution operation
- Stride: the number of pixels by which we slide our filter matrix over the input
- Zero-padding: padding the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. **Zero-padding results in an output image of the same size as the input image.**

Convolutional neural networks: Zero-padding



padding in convolutional network

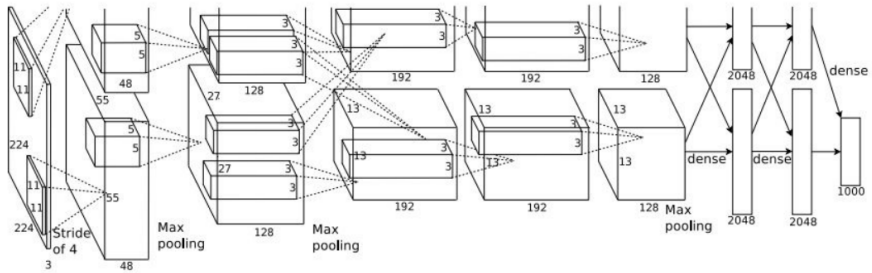
Padding is simply a process of adding layers of zeros to our input images so as to avoid shrinking the image during the convolution operation. When applying zero-padding, the size of the output image is the same as the size of the input image.

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



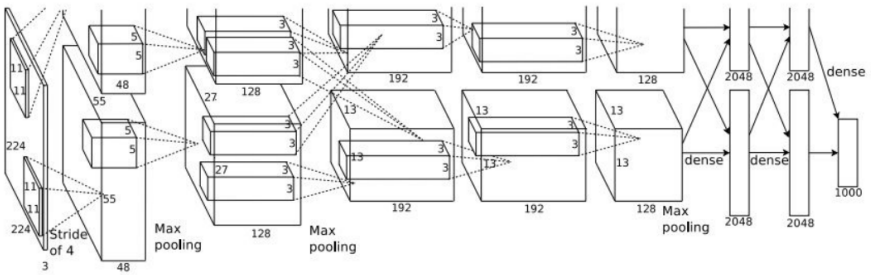
Assuming no zero-padding and weight sharing throughout the entire image

Convolutional neural networks: Example

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

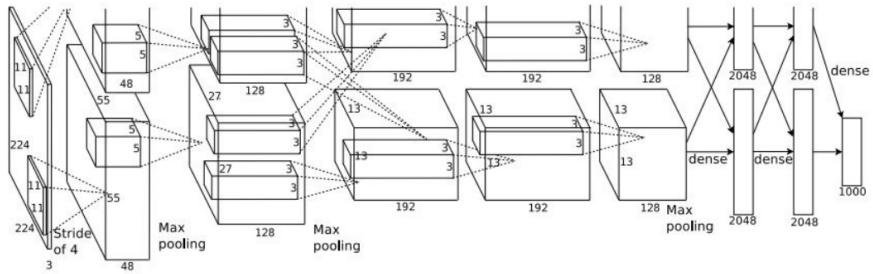
What is the size of the output of the first layer?

Hint: $(227 - 11)/4 + 1 = 55$

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

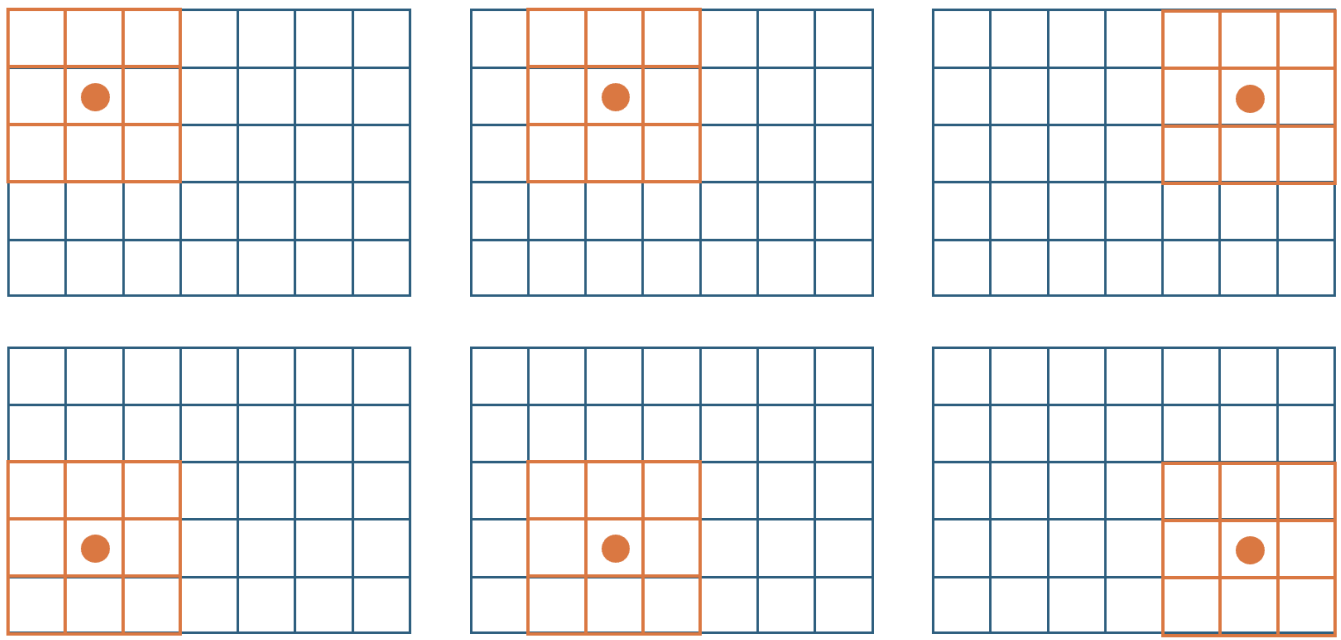
First layer (CONV1): 96 11x11 filters applied at stride 4

What is the size of the output of the first layer?

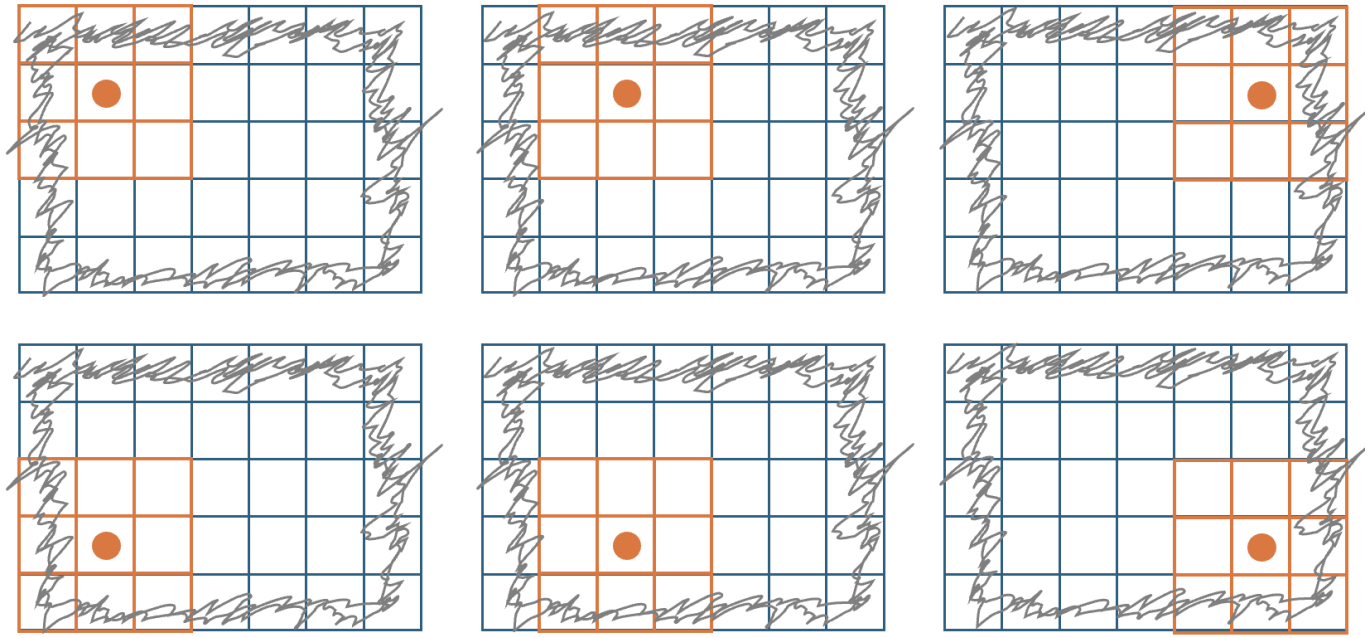
Hint: $(227 - 11)/4 + 1 = 55$

Answer: $55 \times 55 \times 96$

Convolutional neural networks: Example



Convolutional neural networks: Example



$N \times N$ image, $K \times K$ filter, stride of 1, no zero-padding

On each side of the image, $\frac{K-1}{2}$ pixels are disregarded

This results in $N - \frac{K-1}{2} - \frac{K-1}{2} = N - K + 1$ pixels on each side, thus an output image of $(N - K + 1) \times (N - K + 1)$

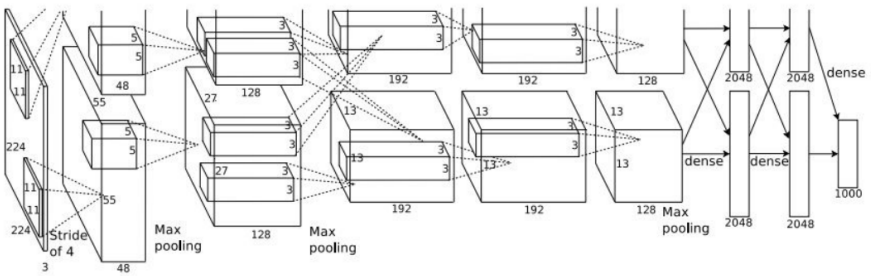
For a stride S, the resulting output image would be $(\frac{N-K}{S} + 1) \times (\frac{N-K}{S} + 1)$

If we assume zero-padding, then the size of the output image is the same as the size of the input $(N \times N)$

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

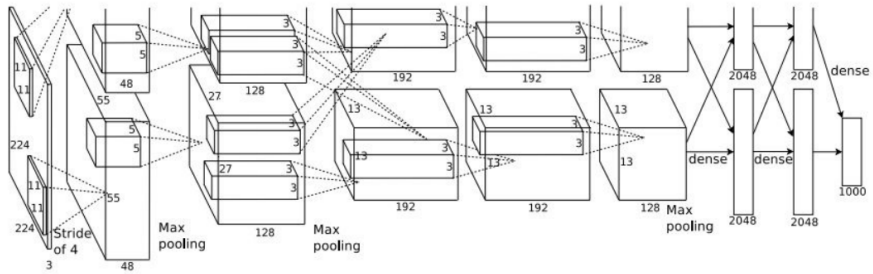
First layer (CONV1): 96 11x11 filters applied at stride 4

What is the total number of parameters that need to be learned in the first layer?

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

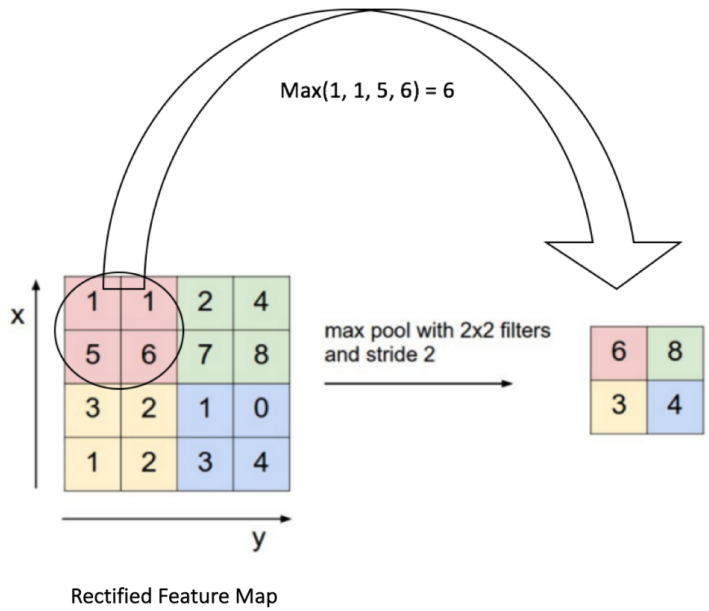
What is the total number of parameters that need to be learned in the first layer?

Answer: $11 \times 11 \times 3 \times 96 = 35K$

Convolutional neural networks: Max-pooling

Spatial Pooling (also called subsampling or downsampling)

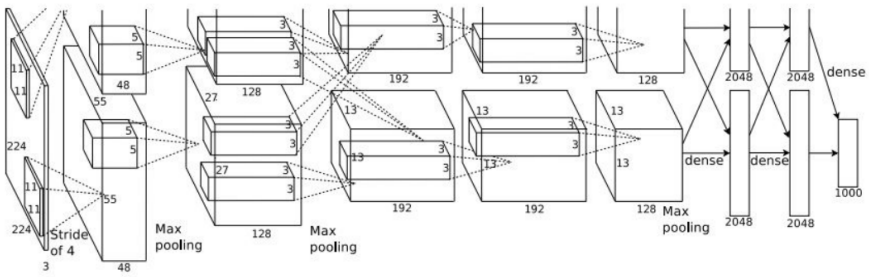
- Reduces the dimensionality of each feature map but retains the most important information
- Can be of different types: Max, Average, Sum etc.
- Makes the input representations (feature dimension) smaller and more manageable
- Promotes an almost scale invariant representation of the image



Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

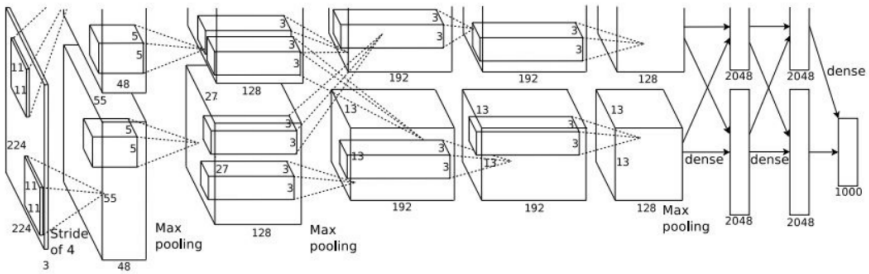
Second layer (POOL1): 3x3 filters applied at stride 2

What is the size of the output of the second layer?
Hint: $(55 - 3)/2 + 1 = 27$

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

What is the size of the output of the second layer?

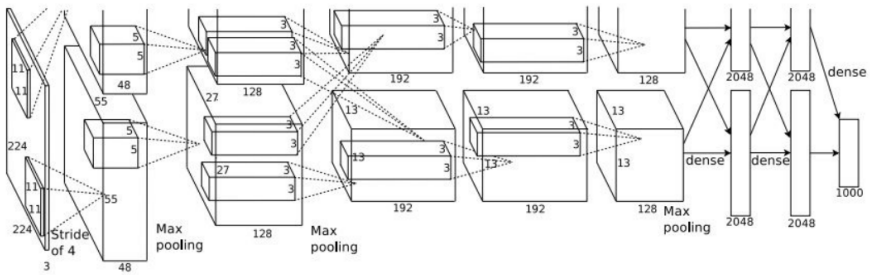
Hint: $(55 - 3)/2 + 1 = 27$

Answer: $27 \times 27 \times 96$

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

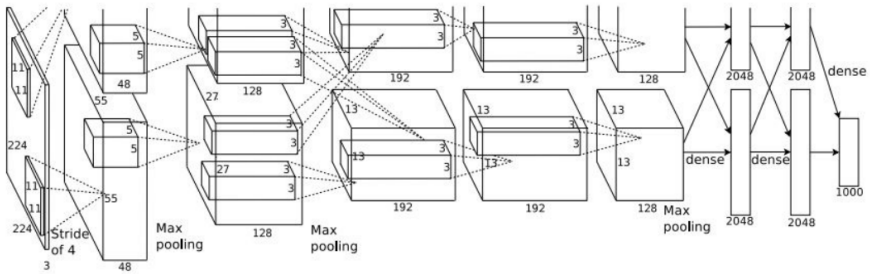
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

What is the number of parameters that need to be learned in second layer?

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

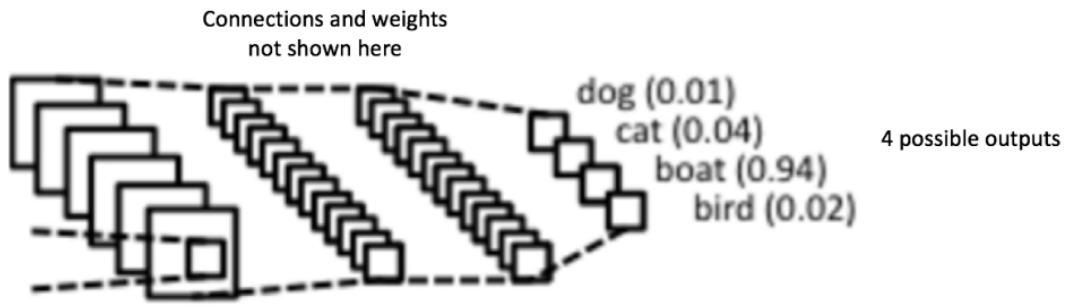
What is the number of parameters that need to be learned in second layer?

Answer: 0!

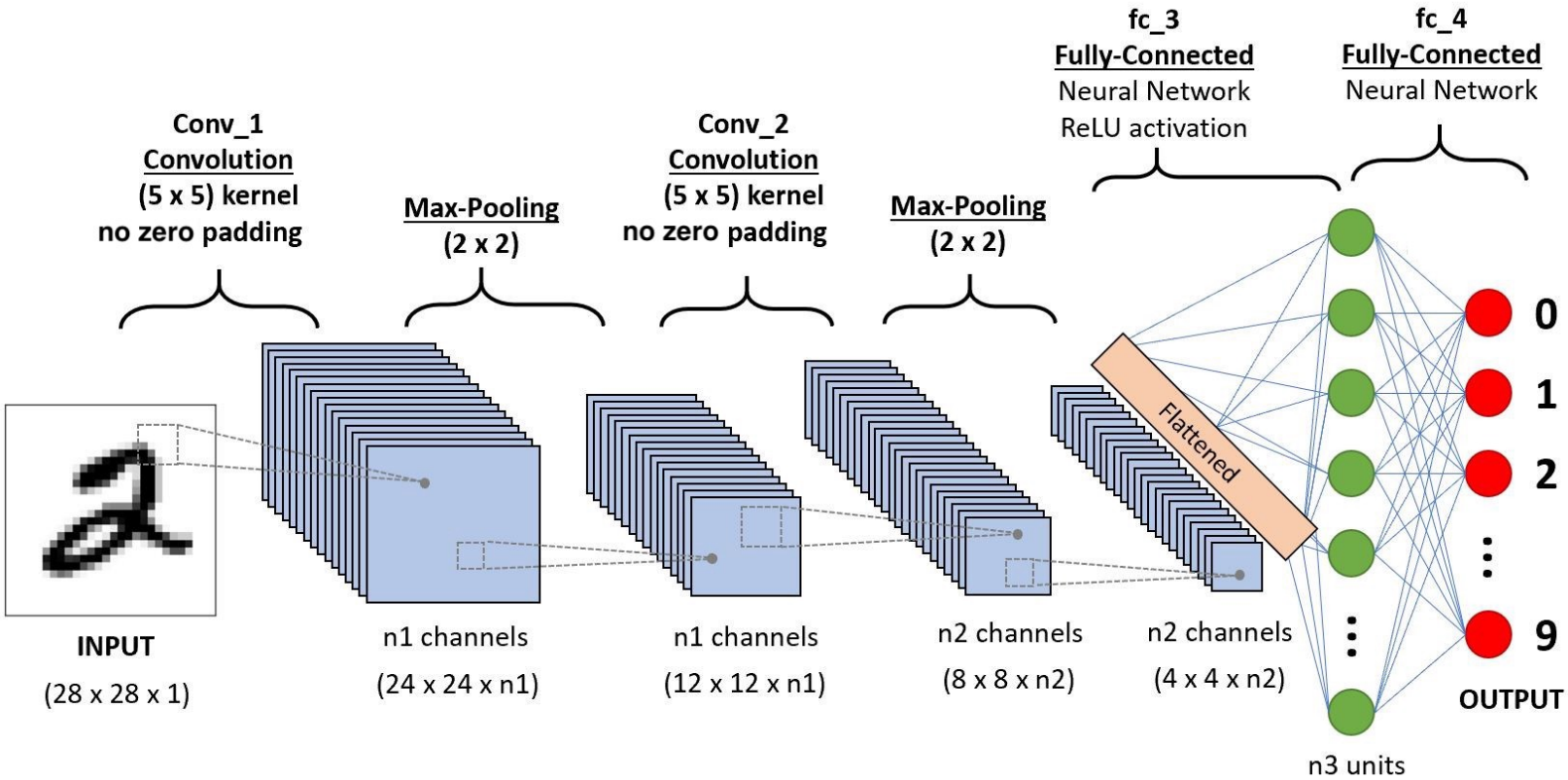
Convolutional neural networks: Final fully connected layer(s)

Final fully connected layer

- "Flattening" the output from the last convolutional layer, i.e., stacking all elements of the output into a single vector
- Using a multilayer perceptron to associate the flattened image representation with the outcome of interest
- Results in the classification/regression outcome

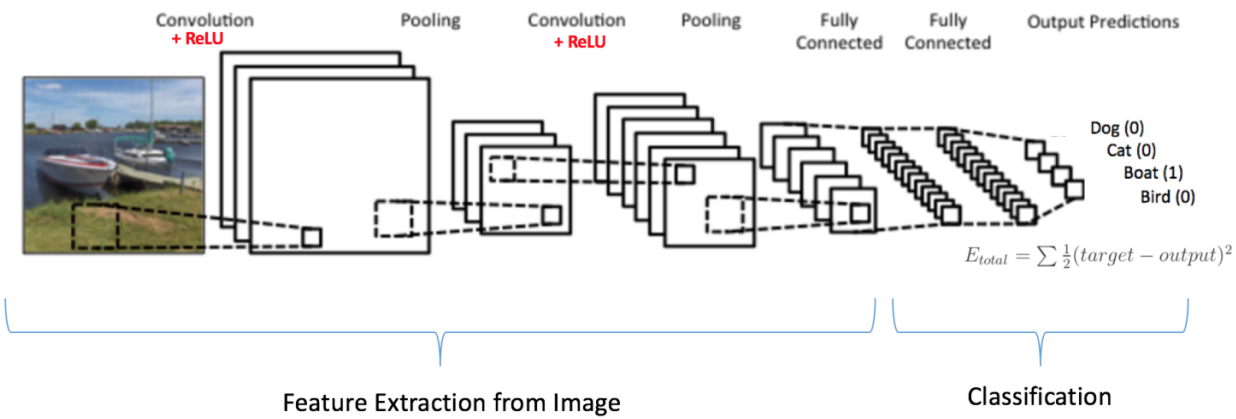


Convolutional neural networks: Putting it all together

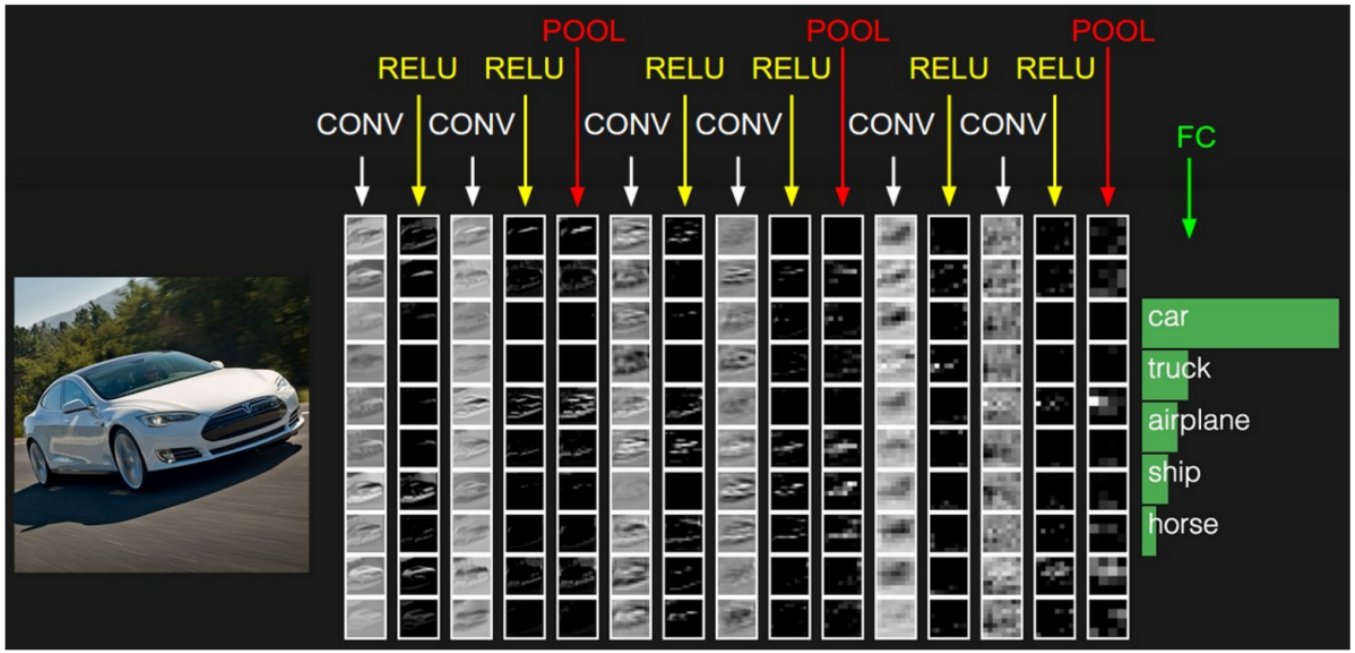


Convolutional neural networks: Putting it all together

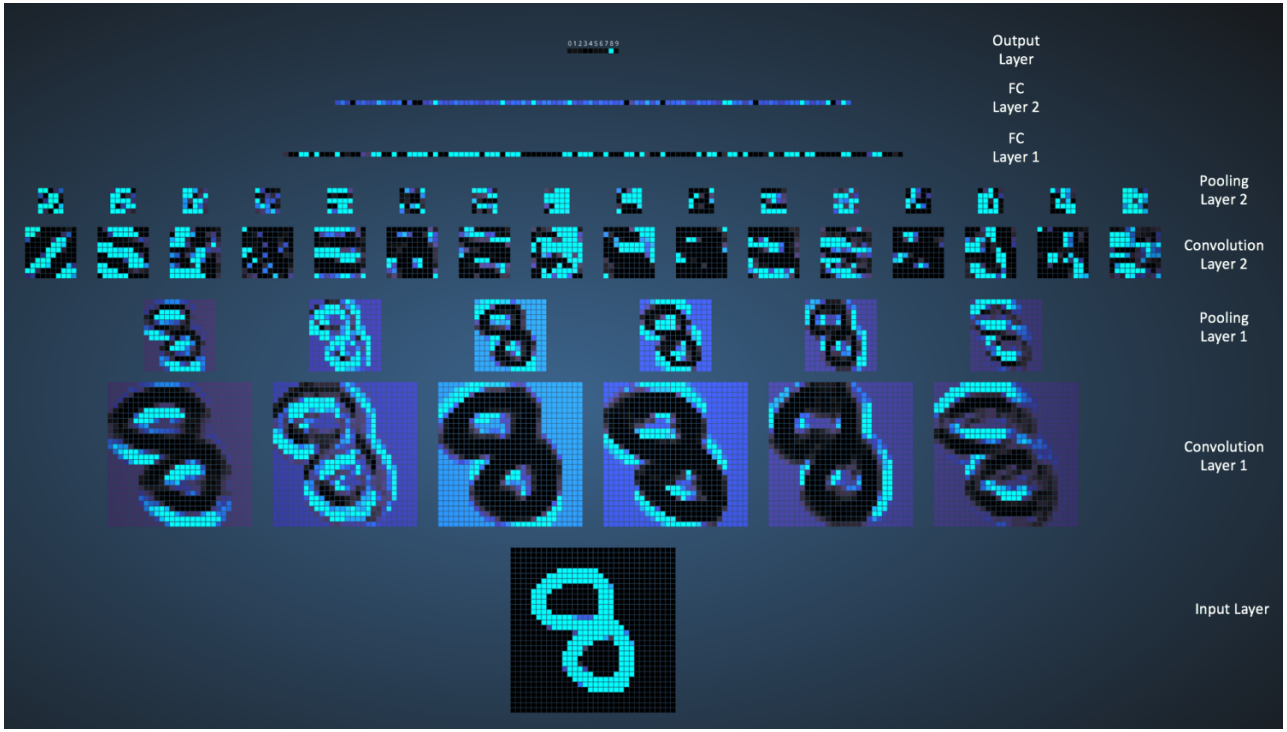
- **Step 1:** Initialize weights
- **Step 2:** Take first image as input and go through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the fully connected layer) and finds the output probabilities for each class
- **Step 3:** Calculate the total error at the output layer
- **Step 4:** Use backpropagation to update the weights, which are adjusted in proportion to their contribution to the total error
- **Step 5:** Repeat Steps 1-4 for all train images



Convolutional neural networks: Examples



Convolutional neural networks: Examples



Convolutional neural networks: Hyperparameter tuning

- Learning rate: how much to update the weight during optimization
- Number of epochs: # times the entire training set passes through the network
- Batch size: the number of samples in the training set for weight update
- Activation function: the function that introduces non-linearity to the model (e.g. sigmoid, tanh, ReLU, etc.)
- Number of convolution and max-pooling layers
- # Filters: the number of filters we use for the convolution operation
- Filter size: the number of filters we use for the convolution operation
- Stride size: the number of pixels by which we slide our filter matrix over the input
- Zero-padding: whether or not we pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. **Zero-padding results in an output image of the same size as the input image.**
- Dropout for regularization: probability of dropping a unit. **Dropout is applied to the fully connected layers at the end and to entire convolutional filters (rather than a random weight of the convolution operation in order to preserve the spatial structure of the input)**
- Optimization method: optimization method for learning the weights (e.g., Adam, RMSProp)

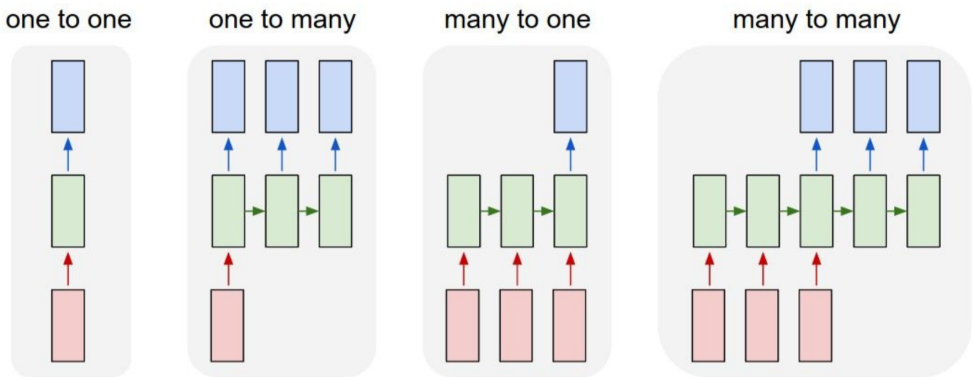
Overview

- Deep neural networks
  - Motivation & Challenges
  - (Supervised) fine-tuning
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks
  - Encoder-decoder (or sequence-to-sequence) networks

[The content for the following slides has been summarized from Li, Johnson, & Yeung, Stanford CSCE 231]

Recurrent neural networks: Motivation

- Networks with feedback loops (recurrent edges)
- Output at current time step depends on current input as well as previous state (via recurrent edges)



- one-to-one: e.g., image classification (image $\rightarrow$ user ID)
- one-to-many: e.g., image captioning (image $\rightarrow$ sequence of words)
- many-to-one: e.g., sentiment classification (sequence of words $\rightarrow$ emotion)
- many-to-many: e.g., machine translation (e.g., sequence of words $\rightarrow$ sequence of words)

Recurrent neural networks: Representation

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

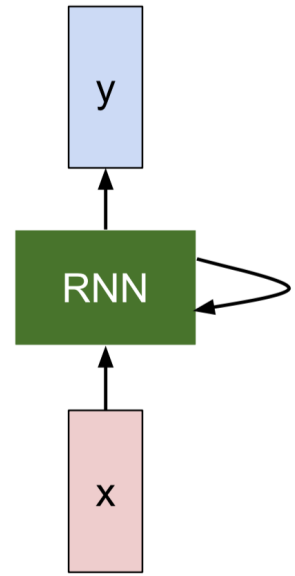
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

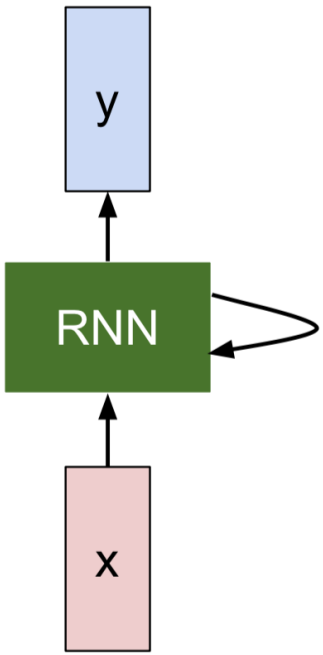
some function with parameters W



The same function and the same set of parameters are used at every time step

Recurrent neural networks: Representation

The state consists of a single *"hidden"* vector **h**:



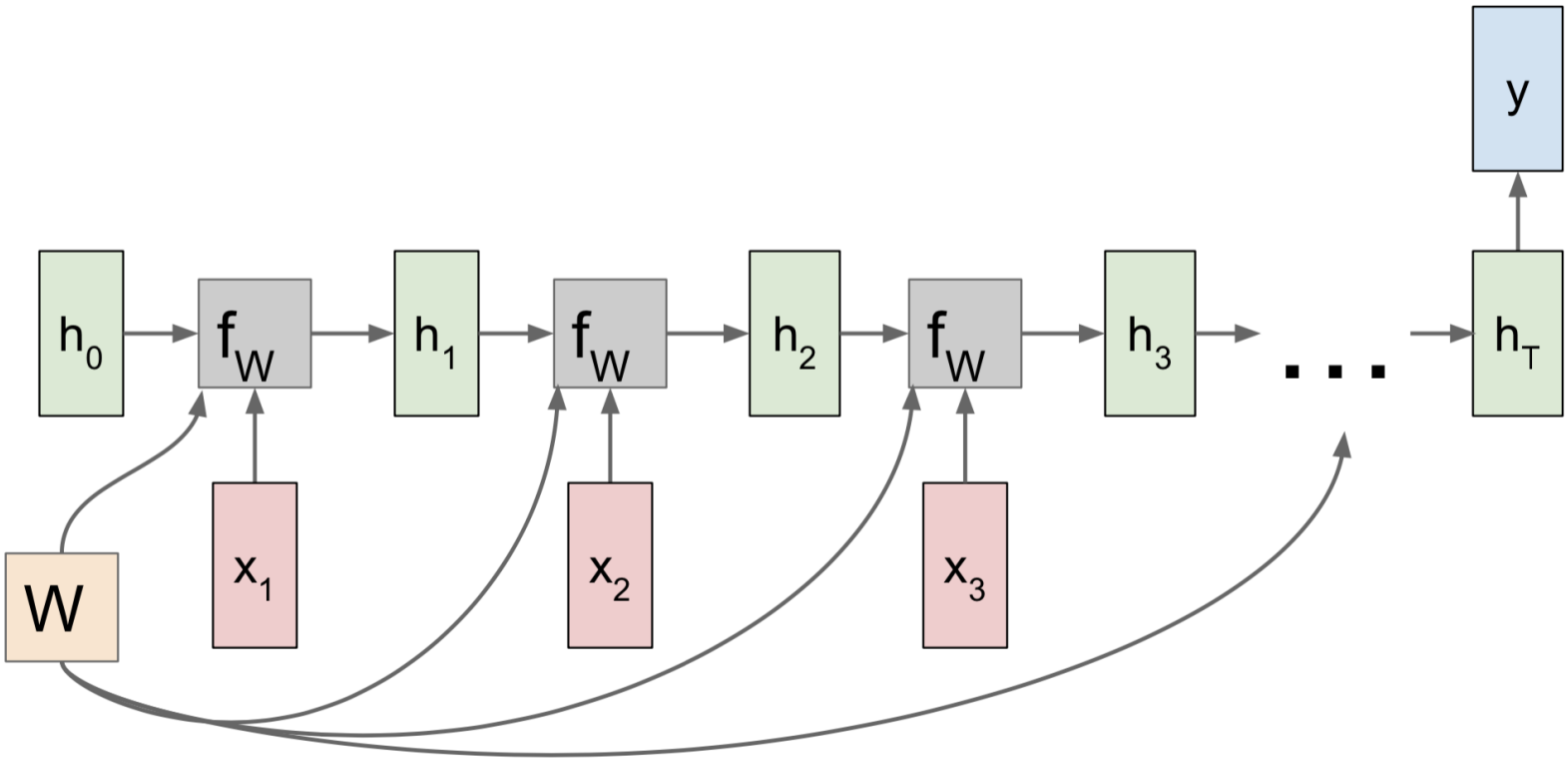
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Recurrent neural networks: Representation

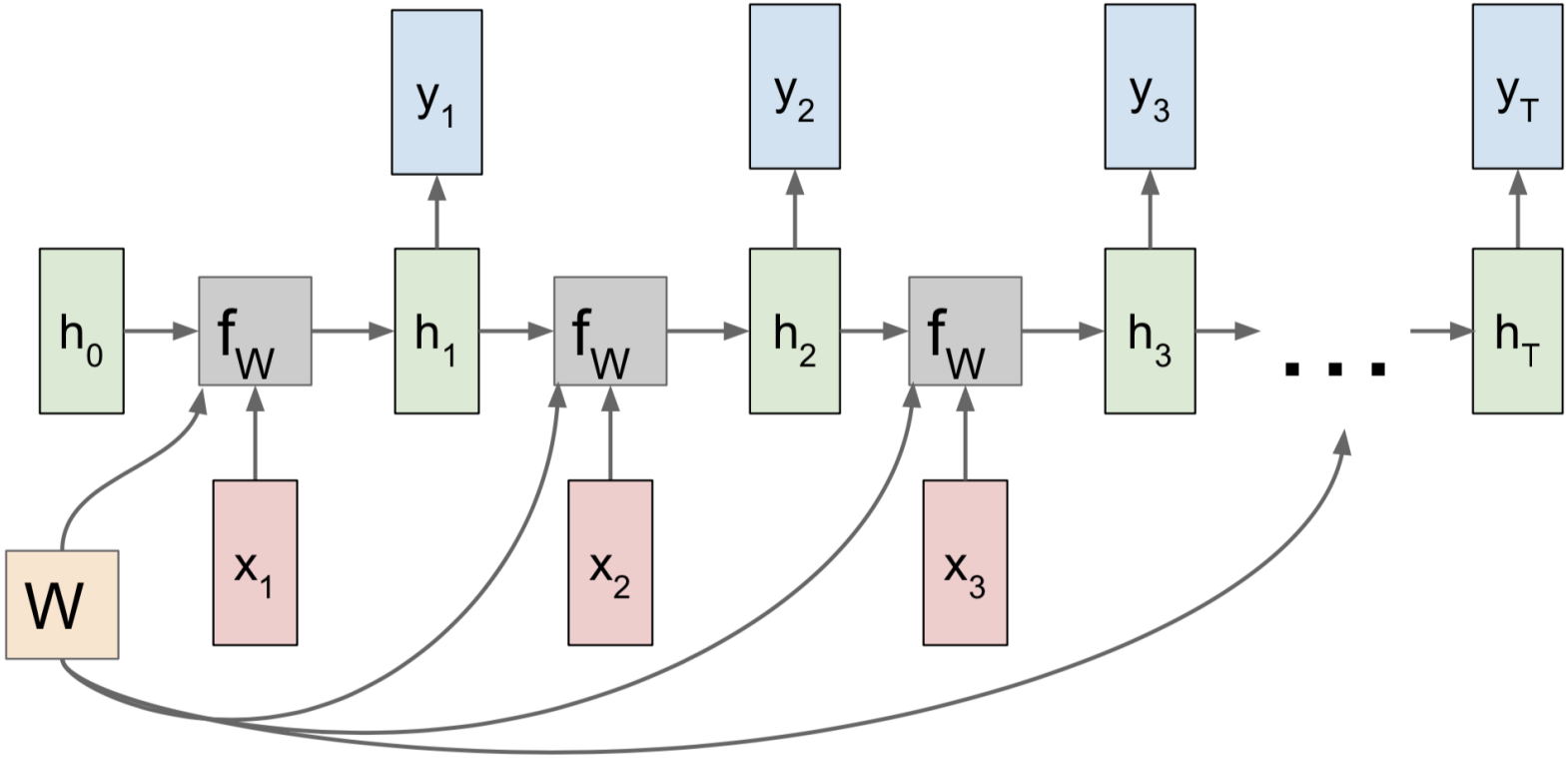
Many-to-one representation



The same function and the same set of parameters are used at every time step.

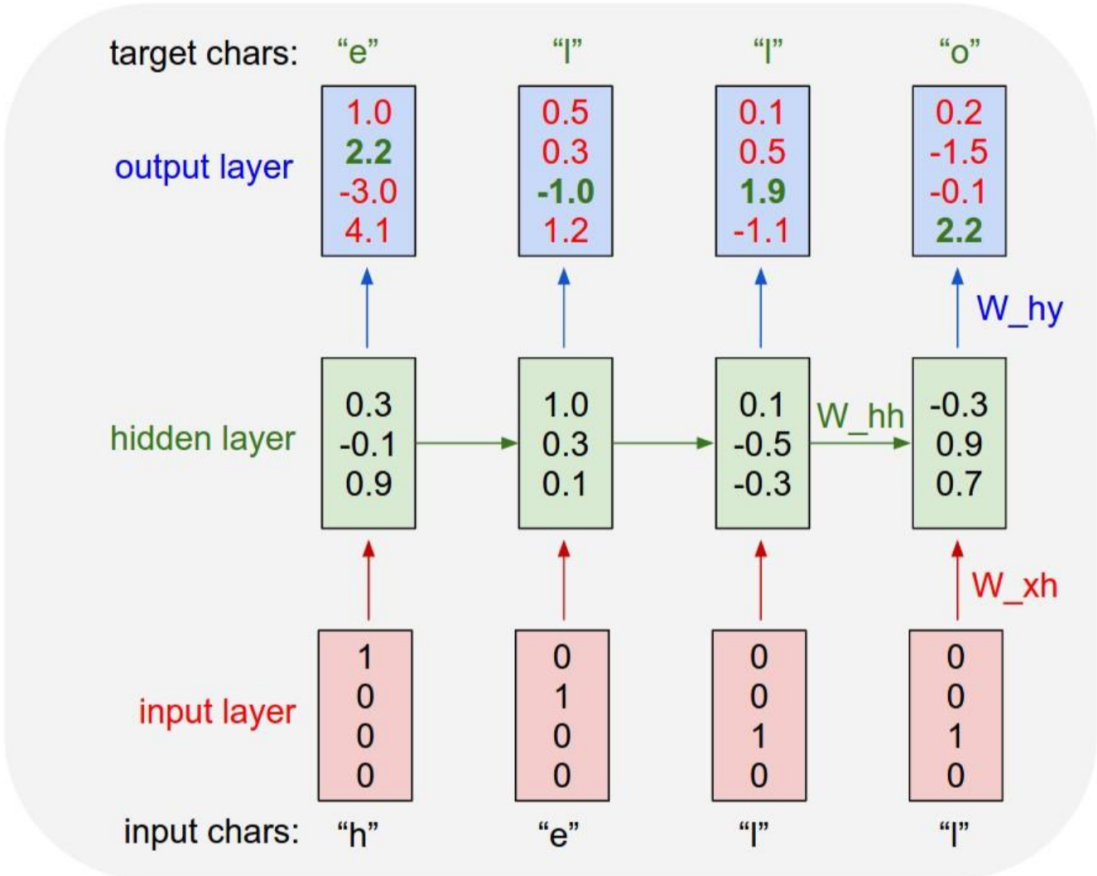
Recurrent neural networks: Representation

Many-to-many representation



Recurrent neural networks: Representation

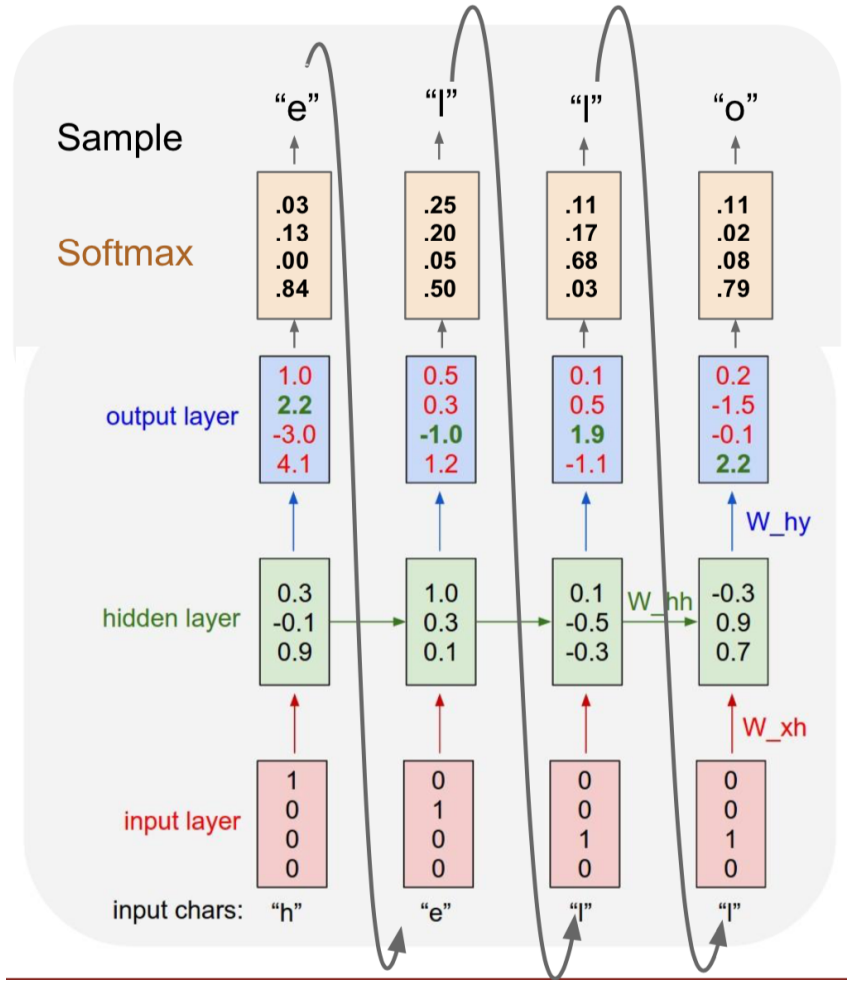
Example: Character-level language model
During training: learning sequence of characters



Recurrent neural networks: Representation

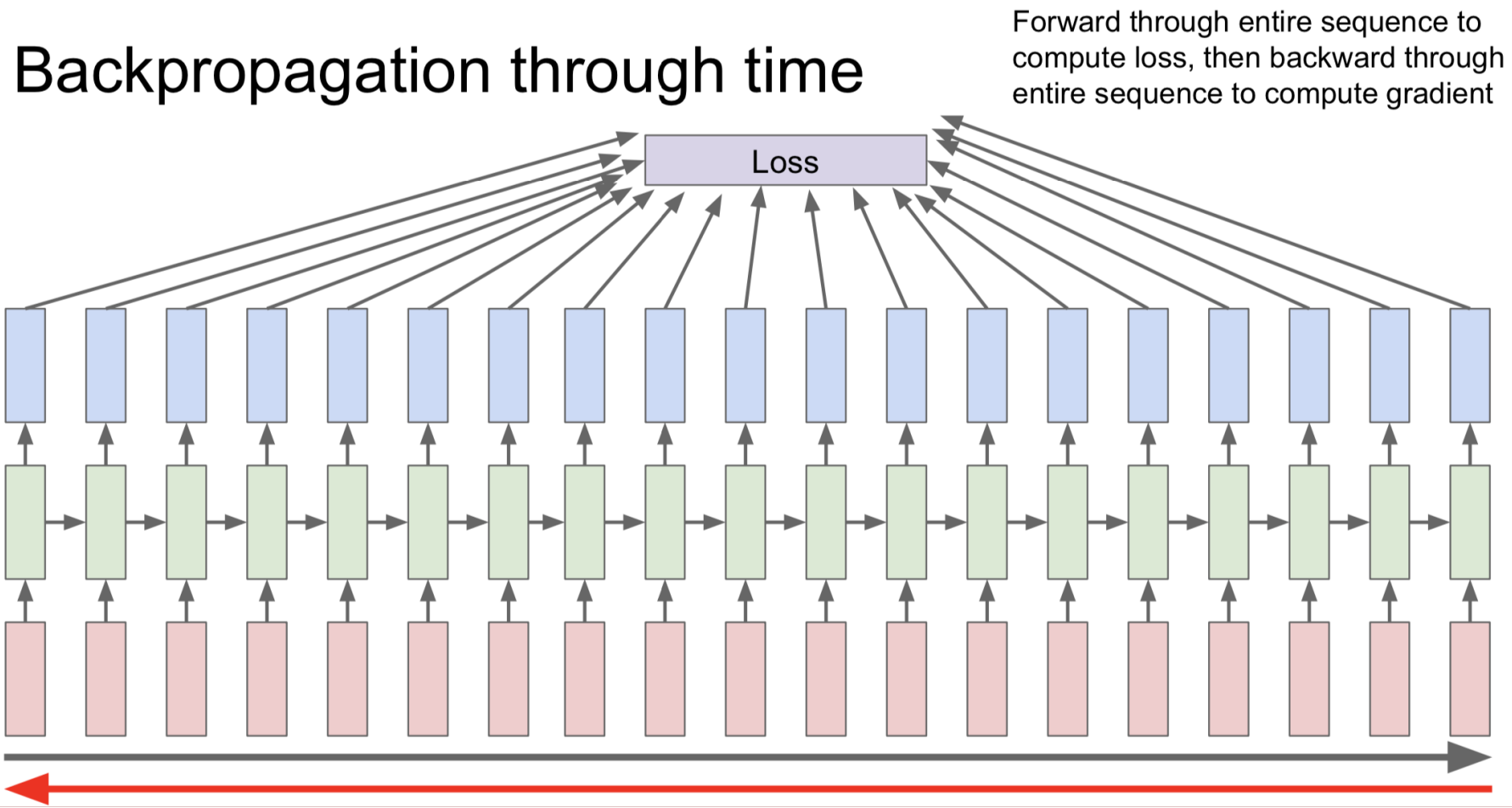
Example: Character-level language model

During testing: sample characters feed back to model one at a time



*note (3/12): the indices of the output layer softmax don't match up to the output, but the general premise of the figure holds

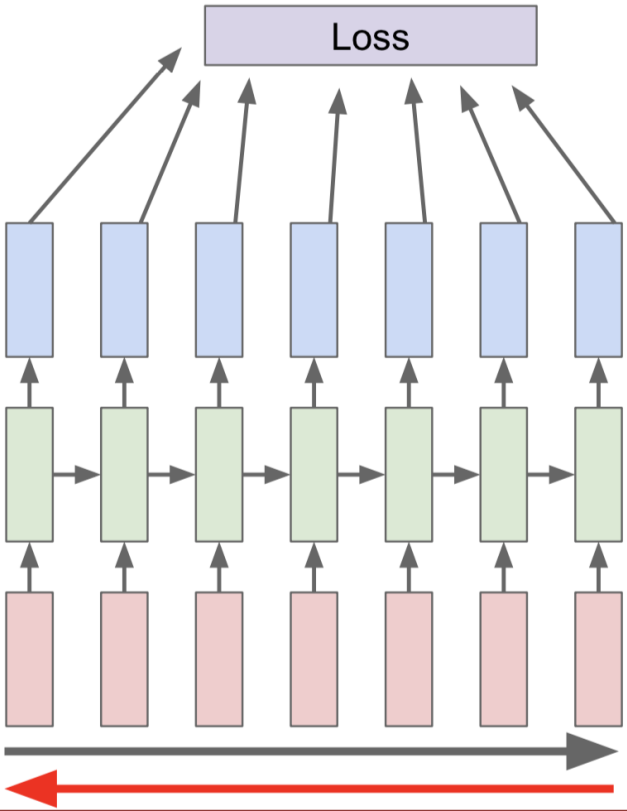
Recurrent neural networks: Learning



Recurrent neural networks: Learning

Recurrent neural networks: Learning

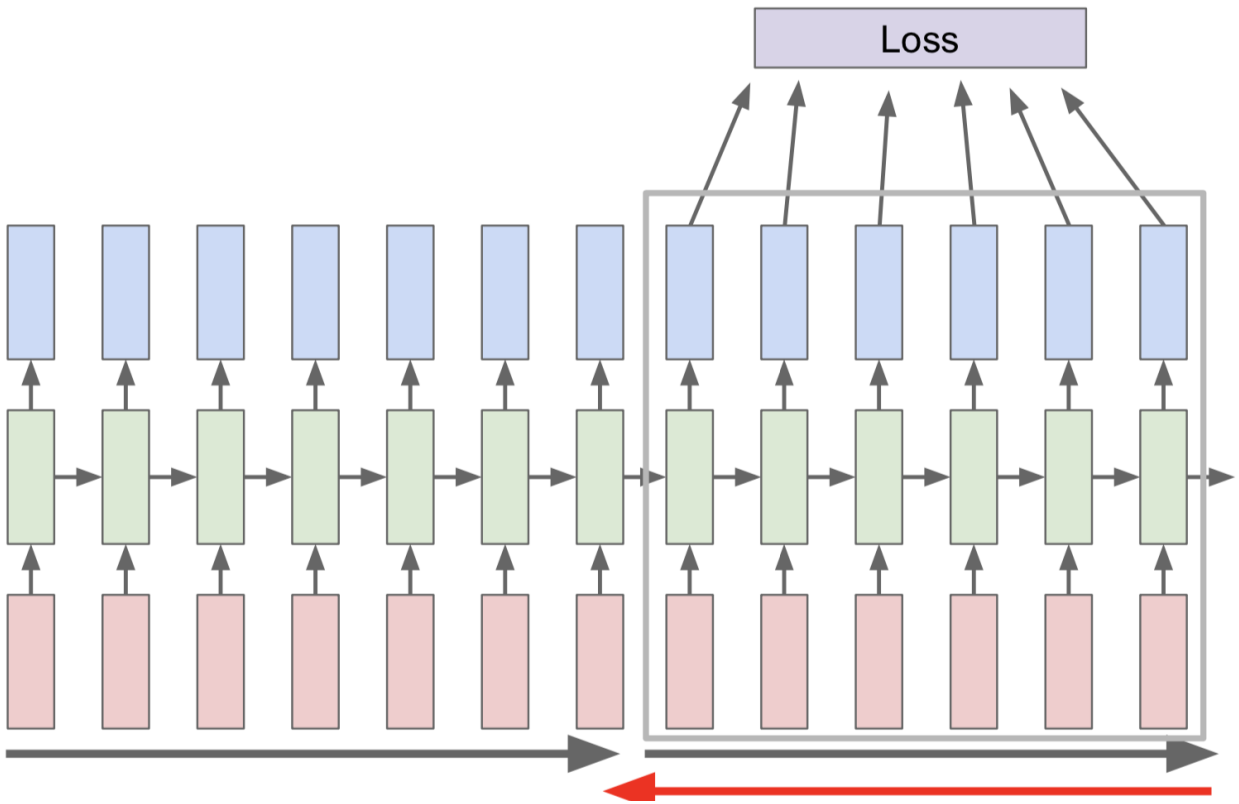
Truncated Backpropagation through time



Run forward and backward through chunks of the sequence instead of whole sequence

Recurrent neural networks: Learning

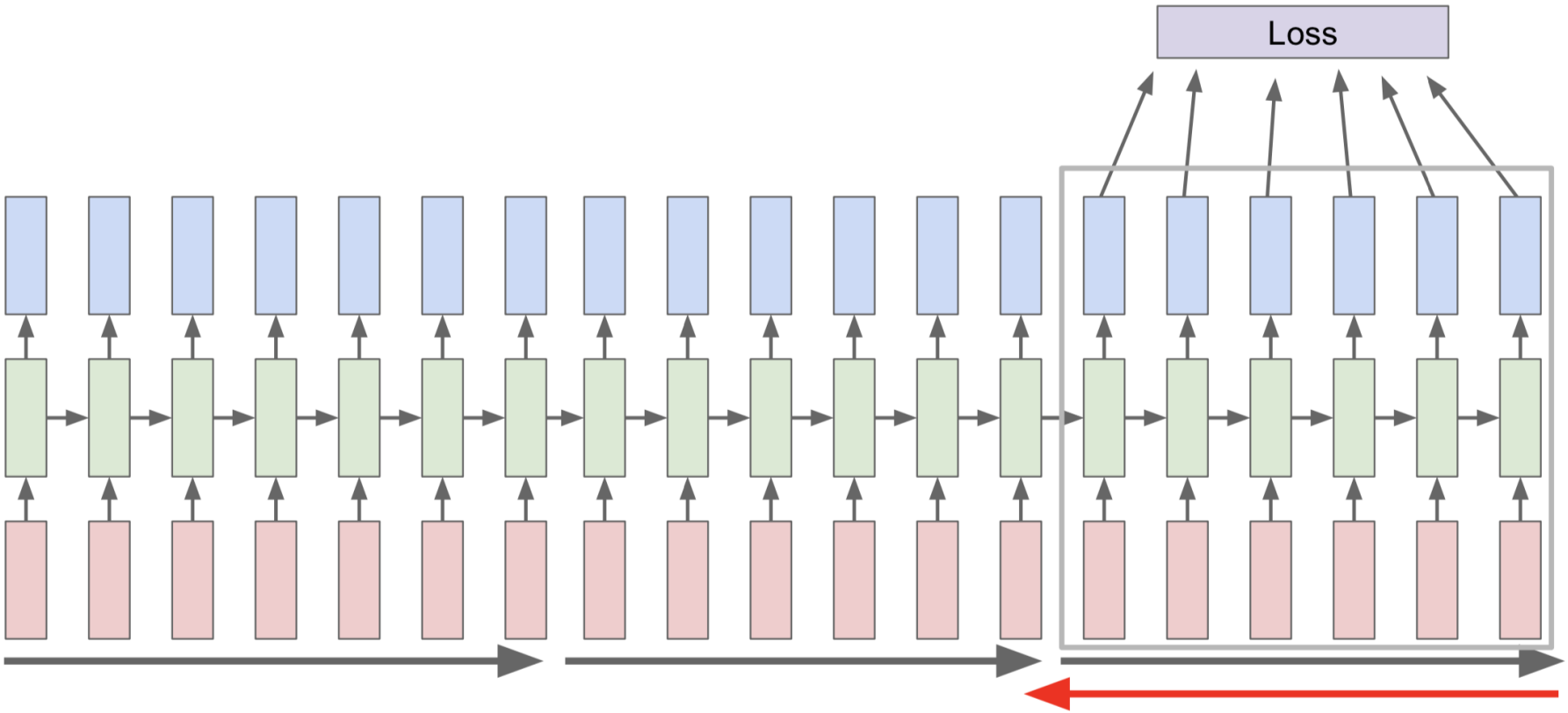
Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Recurrent neural networks: Learning

Truncated Backpropagation through time



Recurrent neural networks: Learning

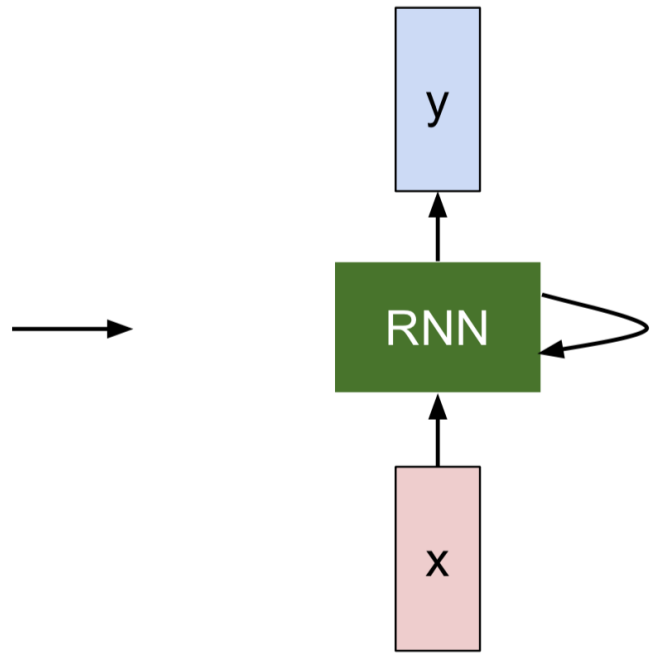
Example: Text generation

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.



Recurrent neural networks: Learning

Example: Text generation

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

↓ train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

↓ train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

↓ train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Recurrent neural networks: Learning

Example: Music generation

Music and Art Generation using Machine Learning | Curtis Hawthorne | TEDxMountainViewHighSchool

`https://www.youtube.com/watch?v=Q-Qq8ipUHEI`

Recurrent neural networks: Learning

Example: Image captioning

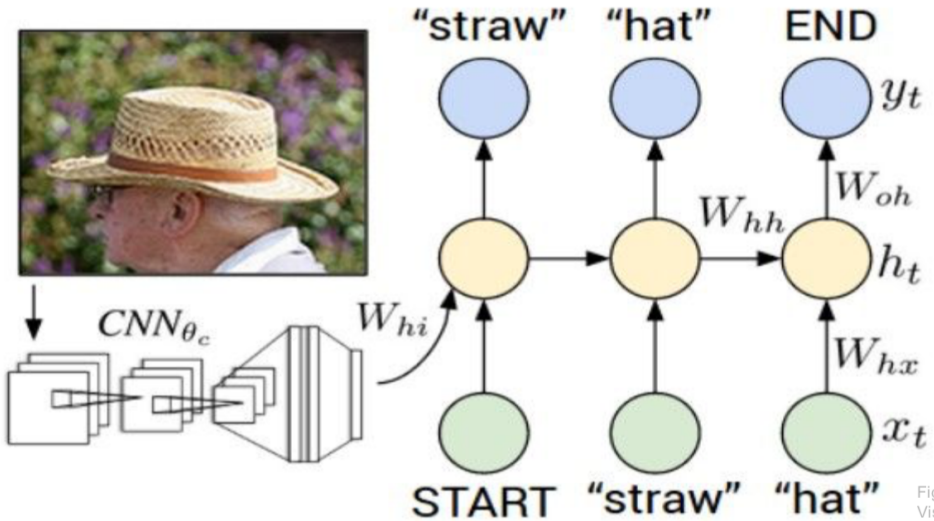
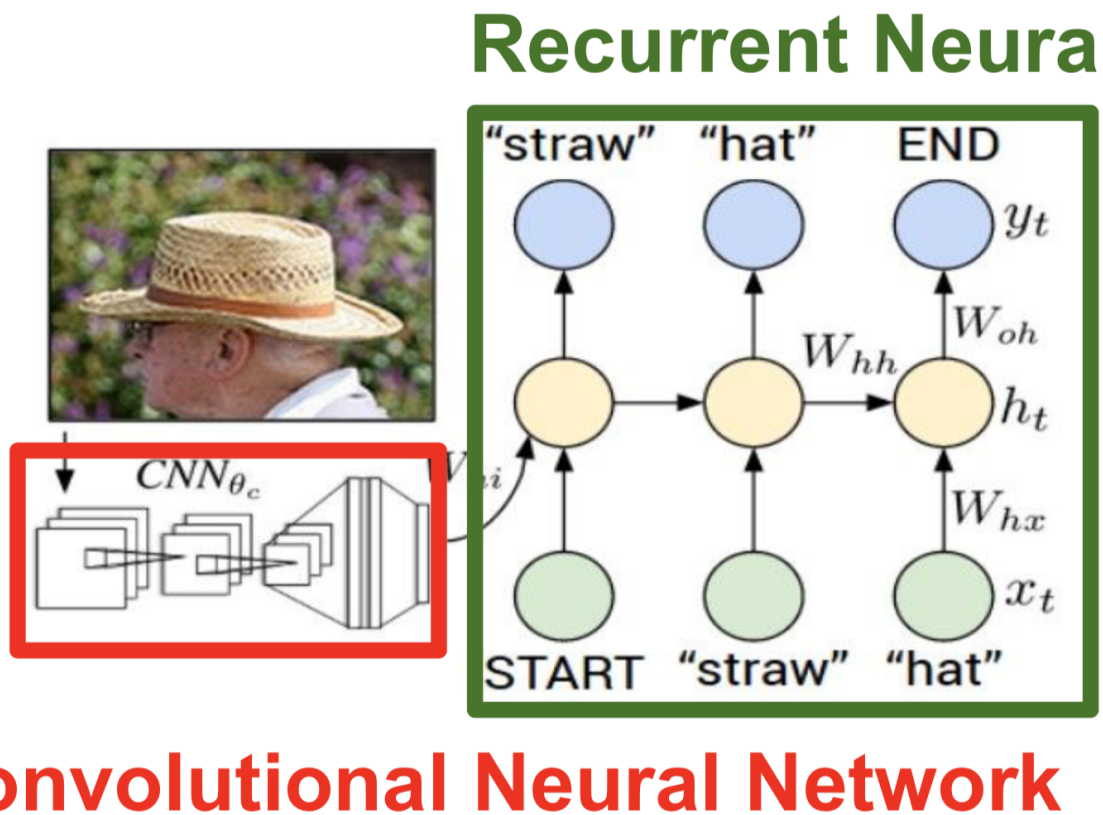


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015. Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

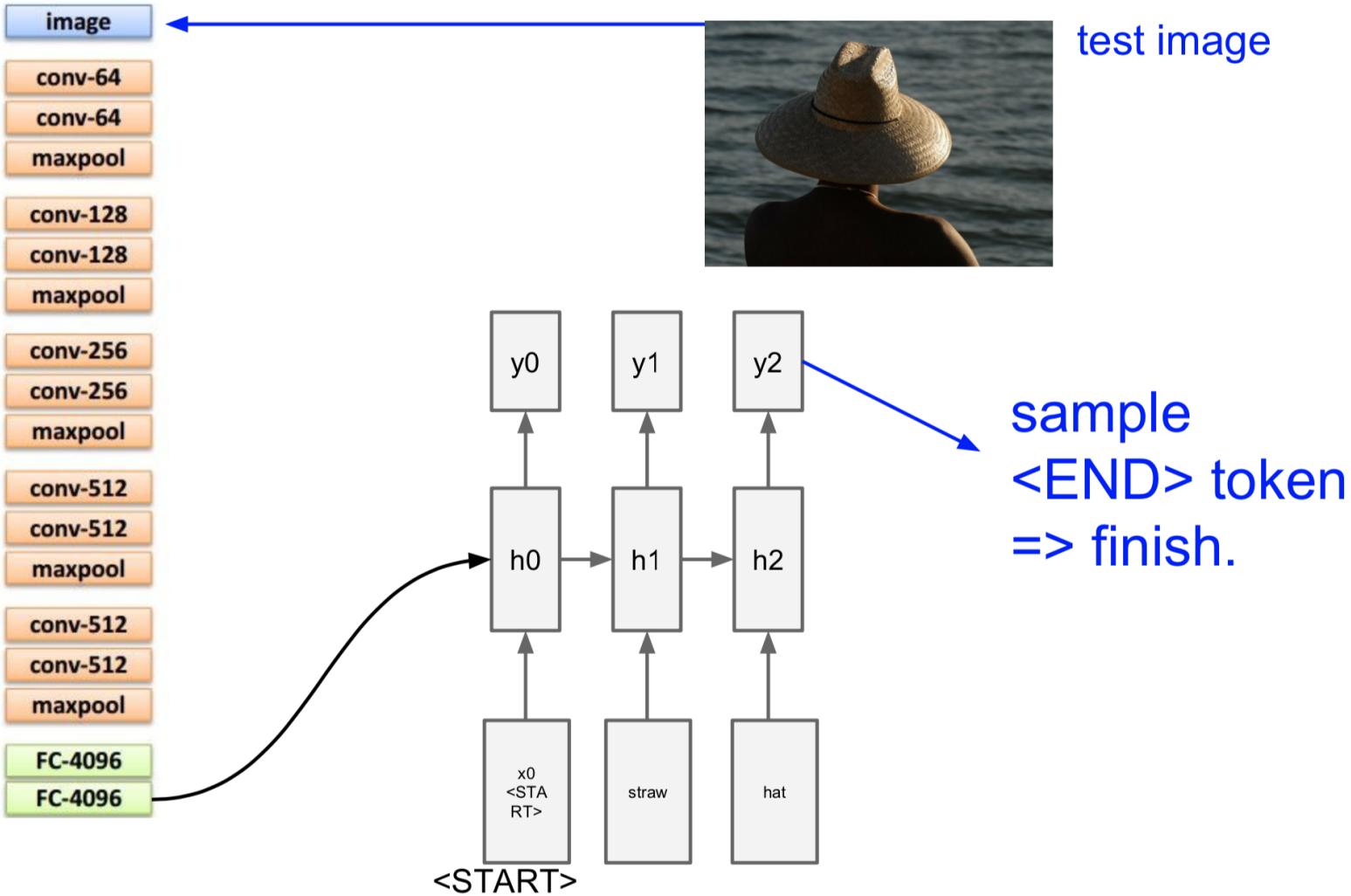
Recurrent neural networks: Learning

Example: Image captioning



Recurrent neural networks: Learning

Example: Image captioning



Recurrent neural networks: Learning

Example: Image captioning

Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain:
cat suitcase, cat tree, dog, bear,
surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

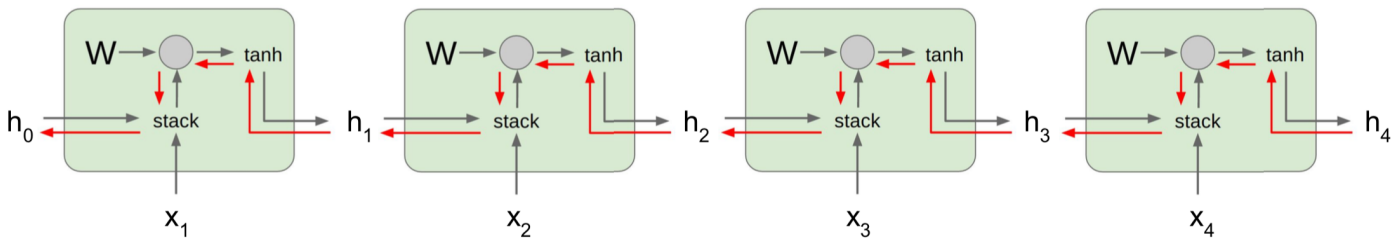
Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Recurrent neural networks: Learning

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients** → **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

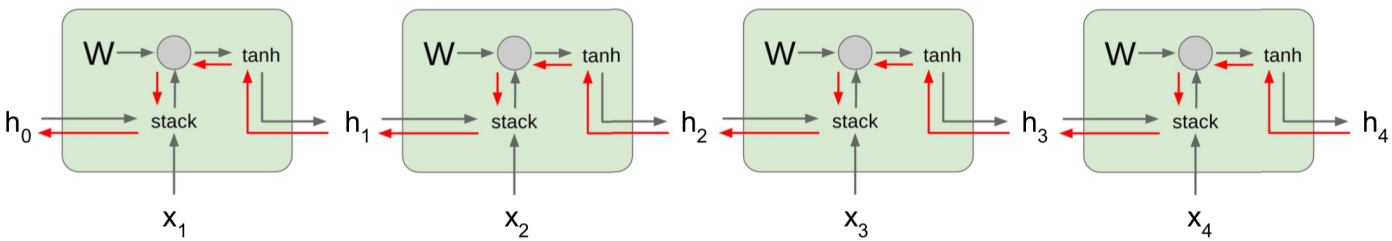
Largest singular value < 1: **Vanishing gradients**

$$\mathbf{h}_t = tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t), \quad \frac{\vartheta L}{\vartheta W_{hh}}$$

Recurrent neural networks: Learning

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients**

Largest singular value < 1: **Vanishing gradients** → Change RNN architecture

Overview

- Deep neural networks
  - Motivation & Challenges
  - (Supervised) fine-tuning
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks
  - Encoder-decoder (or sequence-to-sequence) networks

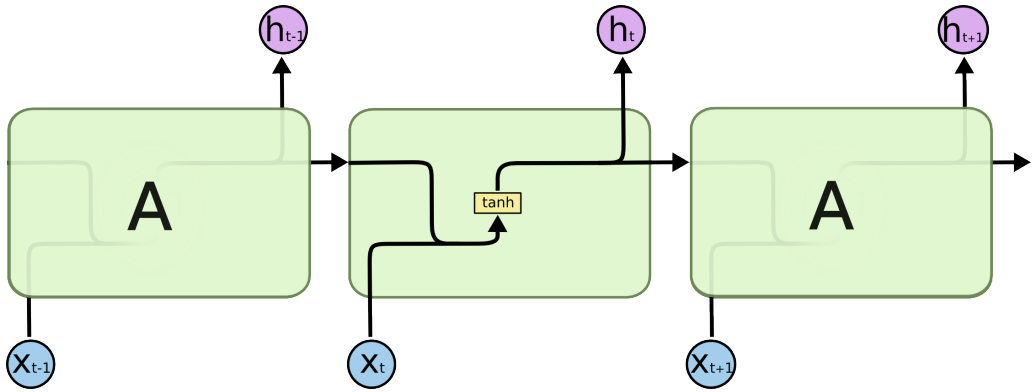
[The content for the following slides has been summarized from

`https://colah.github.io/posts/2015-08-Understanding-LSTMs/`]

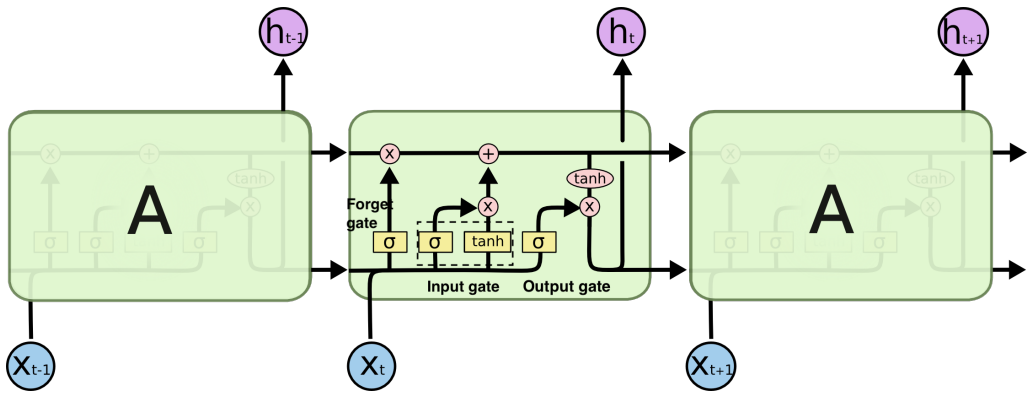
Long short term memory neural networks: Representation

Central Idea: A memory consists of an explicit memory and gating units which regulate the information flow into and out of the memory. Remove information no longer needed. Add information likely to be needed for later.

RNN:

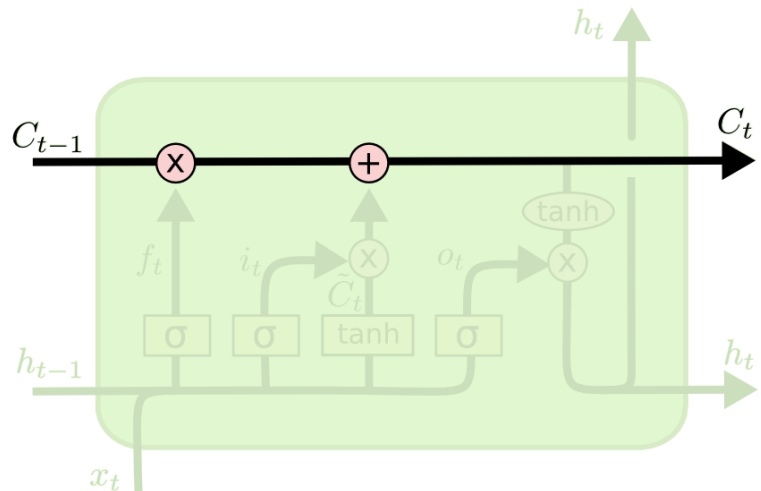


LSTM:



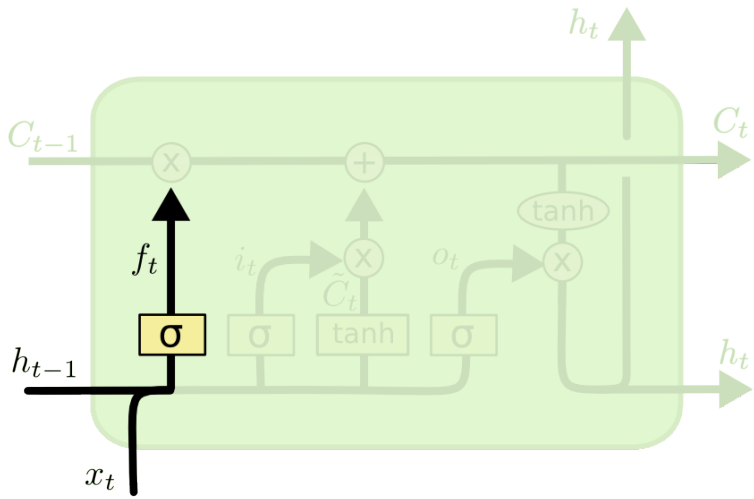
Long short term memory neural networks: Representation

The cell state represents the memory of the network. The LSTM removes or adds information to the cell state, regulated by structures called gates.



Long short term memory neural networks: Representation

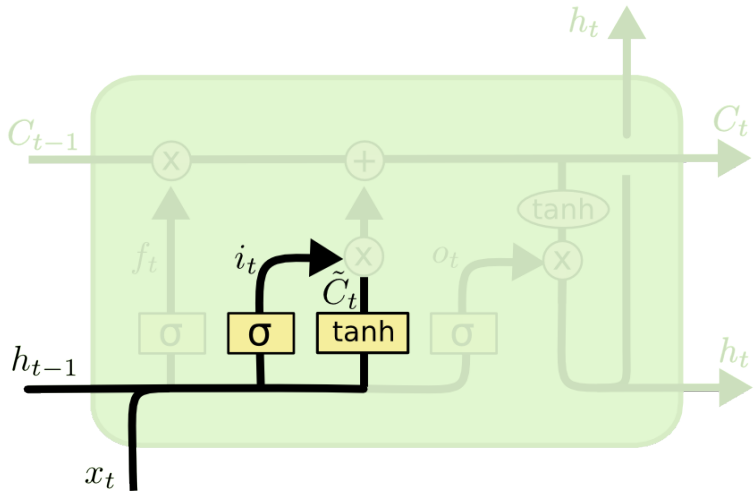
Forget gate layer: Decides what information we will throw away from the previous cell state via a sigmoid function.



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

Long short term memory neural networks: Representation

Input gate layer: Decides what information from the current state we will store to the cell state. The sigmoid determines which input elements will be updated. The *tanh* determines the new candidate values.

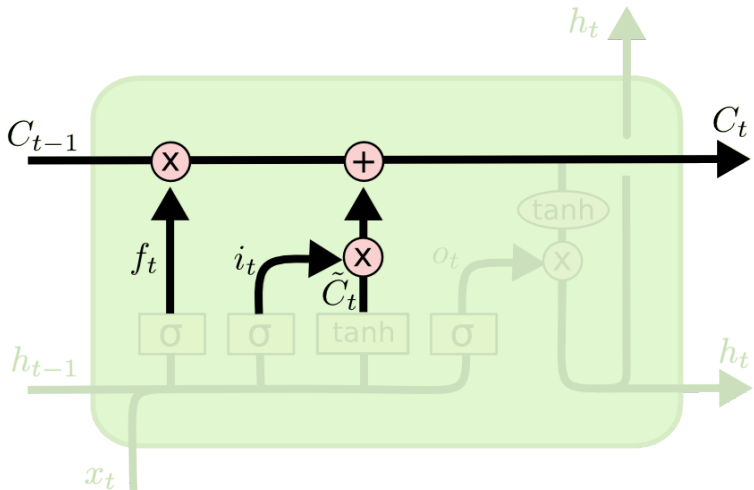


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Long short term memory neural networks: Representation

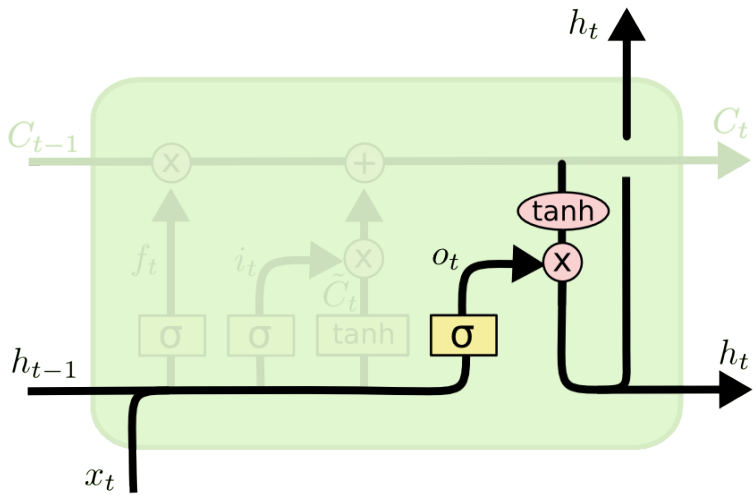
Update cell state: Update cell state based on the forget gate and input gate layers.



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

Long short term memory neural networks: Representation

Output: Provides an output based on the updated cell state and the current input.



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

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

Long short term memory neural networks: Representation

Central Idea: A memory consists of an explicit memory and gating units which regulate the information flow into and out of the memory.

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

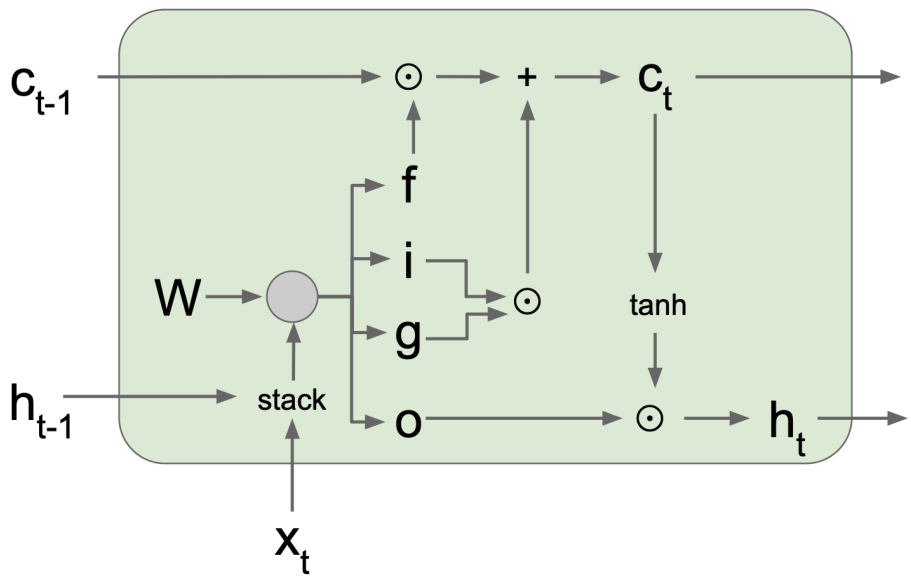
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long short term memory neural networks: Representation

Central Idea: A memory consists of an explicit memory and gating units which regulate the information flow into and out of the memory.



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

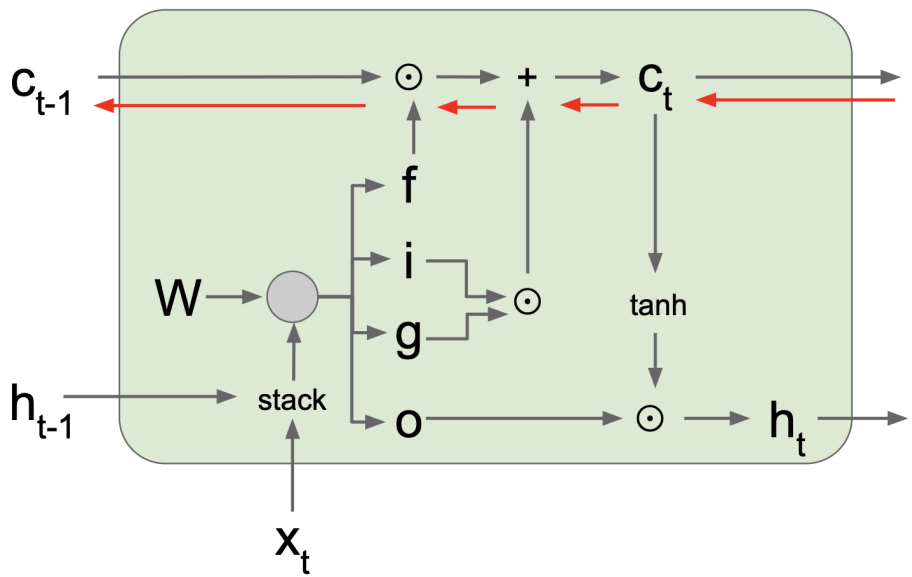
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long short term memory neural networks: Representation

The cell state in the LSTM results from element-wise multiplications with gated inputs and previous cell states without relying on activation functions, which are more susceptible to vanishing gradients

[Hochreiter et al., 1997]



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Overview

- Deep neural networks
  - Motivation & Challenges
  - (Supervised) fine-tuning
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks
  - Encoder-decoder (or sequence-to-sequence) networks

[Contents of the following slides have been summarized from: Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. https://web.stanford.edu/ jurafsky/slp3]

Encoder-Decoder (or Sequence-to-Sequence) Network

- Models capable of generating contextually appropriate, arbitrary length, output sequences
- Applied to various applications, e.g., machine translation, summarization, question answering
- **Encoder:** takes an input sequence $\{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ and generates a sequence of contextualized representations $\{\mathbf{h_1}, \ldots, \mathbf{h_n}\}$
- **Context vector:** conveys the essence of the input to the decoder
- **Decoder:** generates a task-specific output sequence $\{\mathbf{y_1}, \ldots, \mathbf{y_m}\}$

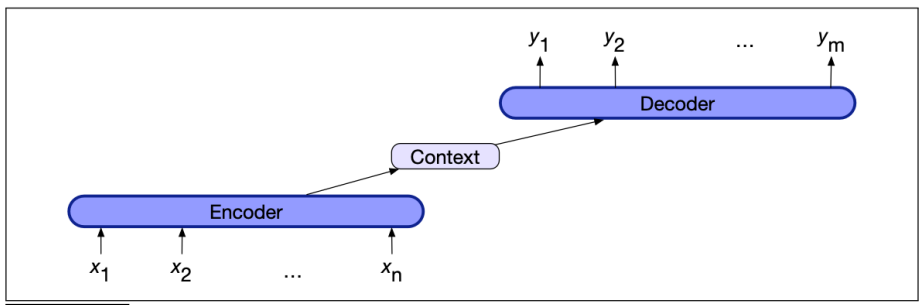


Figure 10.3 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

Encoder-Decoder (or Sequence-to-Sequence) Network

Language translation

- Languages are structured differently, thus depicting differences in their word order
- This can cause problems for translation, requiring the system to do significant structural reorderings as it generates the output

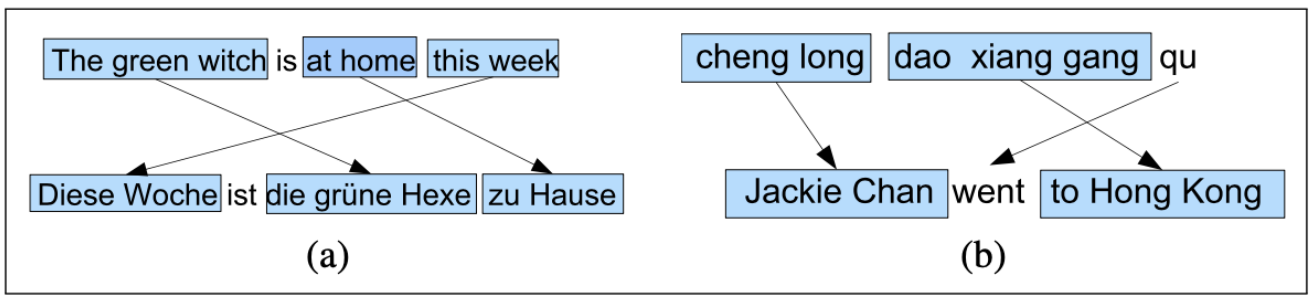
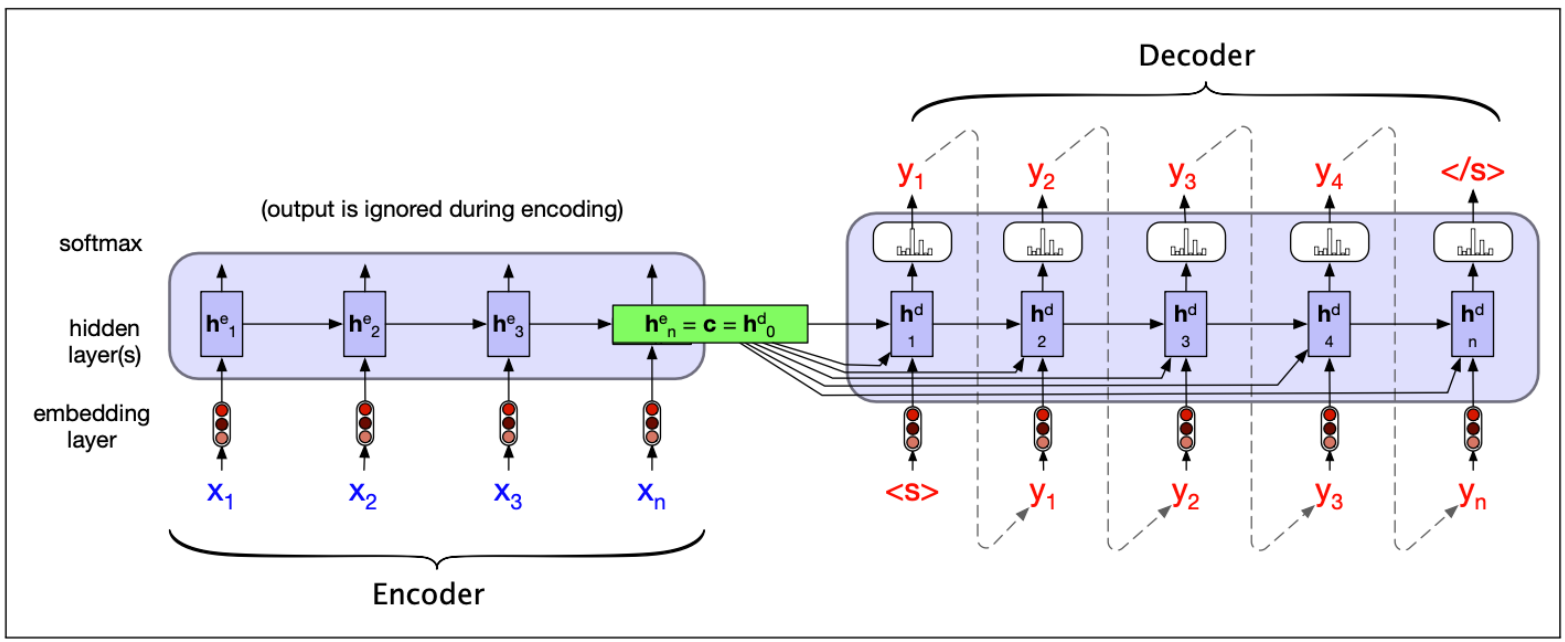


Figure 10.1 Examples of other word order differences: (a) In German, adverbs occur in initial position that in English are more natural later, and tensed verbs occur in second position. (b) In Mandarin, preposition phrases expressing goals often occur pre-verbally, unlike in English.

Encoder-Decoder (or Sequence-to-Sequence) Network

Encoder-Decoder implemented with RNN

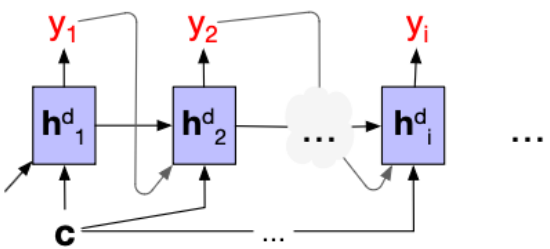
- The encoder generates a contextualized representation of the input
- The final hidden state of the encoder RNN, $\mathbf{h}_n^e$, serves as the context for the decoder in its role as $\mathbf{h}_d^0$ in the decoder RNN $\mathbf{h}_n^e = \mathbf{c} = \mathbf{h}_d^0$
- Subsequent words are conditioned on the previous hidden state and the embedding for the last word generated



Encoder-Decoder (or Sequence-to-Sequence) Network

Encoder-Decoder implemented with RNN

- One weakness of the approach described so far is that the influence of the context vector **c** will wane as the output sequence is generated
- A solution is to make the context vector c available at each step in the decoding process by adding is as a parameter to the computation of the current hidden state



$$\mathbf{c} = \mathbf{h}^{\mathbf{e}}_{\mathbf{n}}$$

$$\mathbf{h}^{\mathbf{d}}_{\mathbf{0}} = \mathbf{c}$$

$$\mathbf{h}^{\mathbf{d}}_{\mathbf{t}} = g(\hat{y}_{t-1}, \mathbf{h}^{\mathbf{d}}_{\mathbf{t-1}}, \mathbf{c})$$

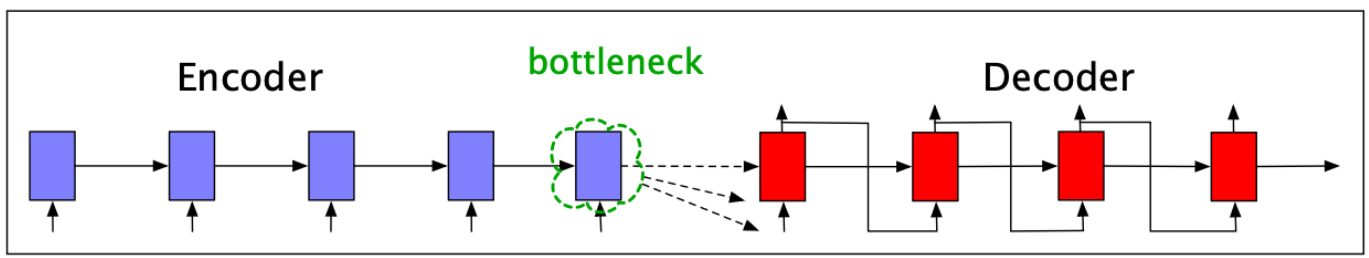
$$\mathbf{z}^{\mathbf{t}} = f(\mathbf{h}^{\mathbf{d}}_{\mathbf{t}})$$

$$y_t = \text{softmax}(\mathbf{z}^{\mathbf{t}})$$

Encoder-Decoder (or Sequence-to-Sequence) Network

Encoder-Decoder implemented with RNN: The bottleneck problem

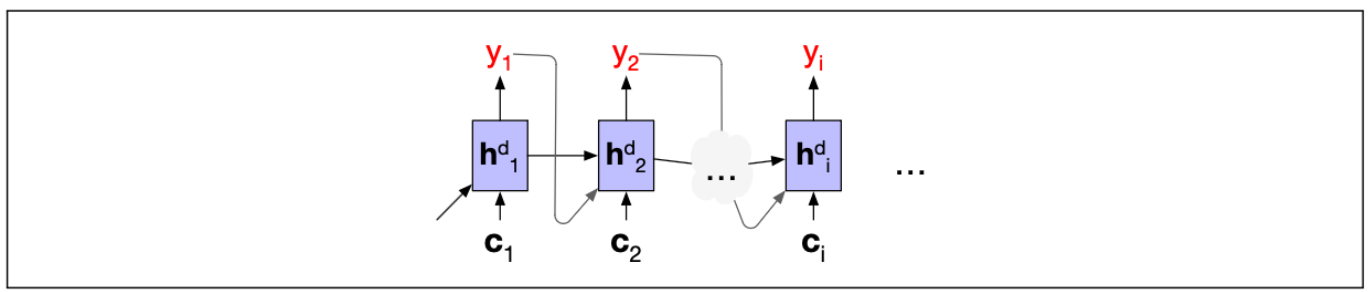
- This final hidden state is acting as a bottleneck and represents everything about the meaning of the source text, since the only thing the decoder knows about the source text is what's in this context vector
- Information at the beginning of the input sentence, especially for long sentences, may not be equally well represented in the context vector



Encoder-Decoder (or Sequence-to-Sequence) Network

Attention

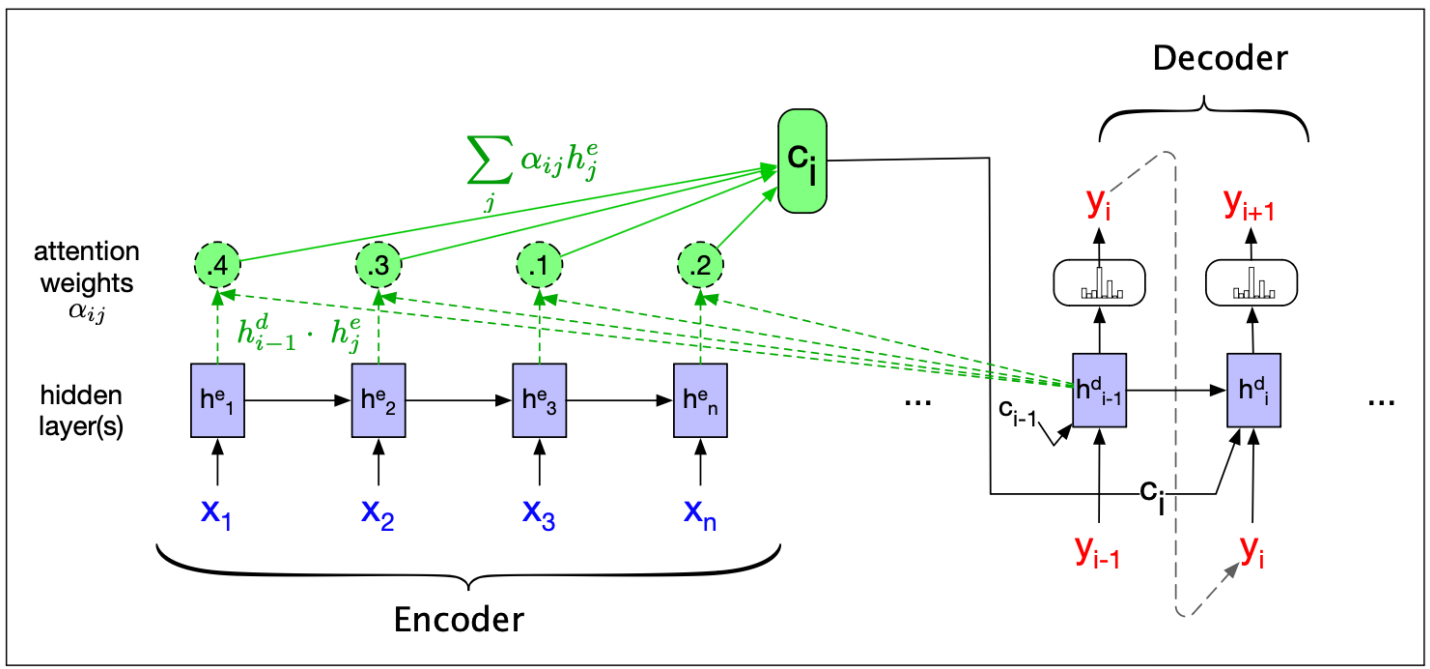
- A solution to the bottleneck problem
- Allows the decoder to get information from all the hidden states of the encoder, not just the last hidden state
- Replaces the static context vector with one that is dynamically derived from the encoder hidden states, different for each token in decoding
- Creates a fixed-length context vector **c** by taking a weighted sum of all the encoder hidden states. The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing



Encoder-Decoder (or Sequence-to-Sequence) Network

Attention: Implementation

- Measure how relevant each encoder state $\mathbf{h}_j^e$ is to the decoder state captured in $\mathbf{h}_{i-1}^d$
- The simplest similarity score is the dot-product, i.e., **dot-product attention**: $score(\mathbf{h}_j^e, \mathbf{h}_{i-1}^d) = \mathbf{h}_j^e \cdot \mathbf{h}_{i-1}^d$
- Weighted average over all the encoder hidden states: $\mathbf{c_i} = \sum_j \alpha_{ij} \mathbf{h}_j^e$, where $\alpha_{ij} = softmax(score(\mathbf{h}_j^e, \mathbf{h}_{i-1}^d)) = \frac{exp(score(\mathbf{h}_j^e, \mathbf{h}_{i-1}^d))}{\sum_k exp(score(\mathbf{h}_k^e, \mathbf{h}_{i-1}^d))}$



Encoder-Decoder (or Sequence-to-Sequence) Network

Attention: Implementation

- More sophisticated scoring functions, e.g., $score(\mathbf{h}_j^e, \mathbf{h}_{i-1}^d) = \mathbf{h}_j^e \mathbf{W_s} \mathbf{h}_{i-1}^d$
- The weights $\mathbf{W_s}$ are trained during normal end-to-end training, giving the network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application

What have we learnt so far

- DNNs allow hierarchical representations learned from raw data
- Challenges in terms of training $\rightarrow$ pretraining, alternative optimization methods
- Convolutional neural networks $\rightarrow$ image
  - convolution: local image properties
  - weight sharing: stationarity
  - max-pooling: robustness in the representation and reduced cost
- Residual Neural Networks and LSTMs $\rightarrow$ sequence data
- Encoder-decoder networks: learn latent/hidden context
- Attention mechanism (underlies transformer architectures)