Neural Networks: Architectures

Mark Craven and David Page
Computer Sciences 760
Spring 2019

Goals for the lecture

you should understand the following concepts

- parameter tying
- recurrent neural networks
- long short term memory (LSTM) networks
- bidirectional LSTM networks
- convolutional networks

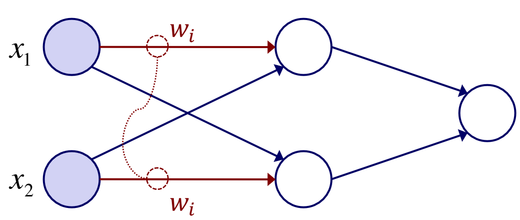
Notation

When describing more complex neural network architectures, it is convenient to have notation for describing manipulations of vectors of values

$\sigma(\boldsymbol{x})$	given vector $\boldsymbol{x}$, return a vector with sigmoid applied to each element of $\boldsymbol{x}$
$\tanh(\boldsymbol{x})$	given vector $\boldsymbol{x}$, return a vector with tanh applied to each element of $\boldsymbol{x}$
$[\boldsymbol{x}, \boldsymbol{z}]$	concatenate the vectors $\boldsymbol{x}$ and $\boldsymbol{z}$
$\boldsymbol{x} \otimes \boldsymbol{z}$	perform element-wise multiplication of vectors $\boldsymbol{x}$ and $\boldsymbol{z}$

Parameter tying

A simple idea that is quite useful in a variety of neural network applications is *parameter tying* (i.e. sharing): certain connections in a network share the same weight



Sequential tasks

Some applications involve processing sequences of input

- natural language
- biological sequences, such as DNA, RNA, protein
- time-series such as financial indicators or weather

Consider the following natural language tasks:

Part-of-speech tagging

$\boldsymbol{y}$: DT NN VBZ DT RB VBN NN
$\boldsymbol{x}$: `The patient has a recently injured knee.`

Sentiment analysis

$\boldsymbol{y}$: positive
$\boldsymbol{x}$: `Great price, fast shipping, great product.`

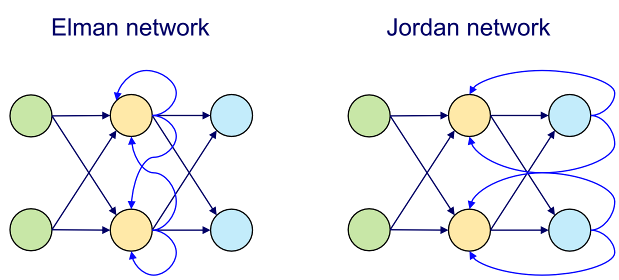
$\boldsymbol{y}$: negative
$\boldsymbol{x}$: `Not such a great product.`

Recurrent neural networks

In sequential tasks, it may be valuable to have the network

- process varying-length input sequences
- "remember" information when processing a sequence

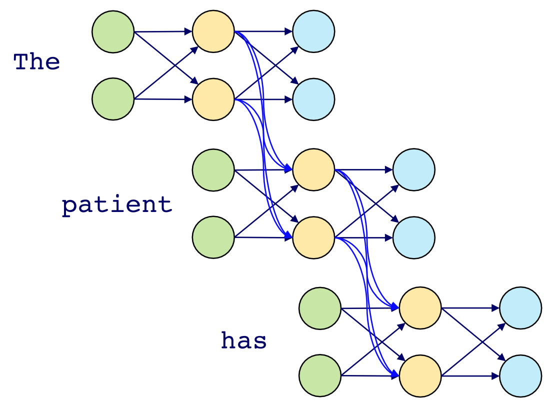
recurrent neural networks enable this



6

Recurrent neural networks

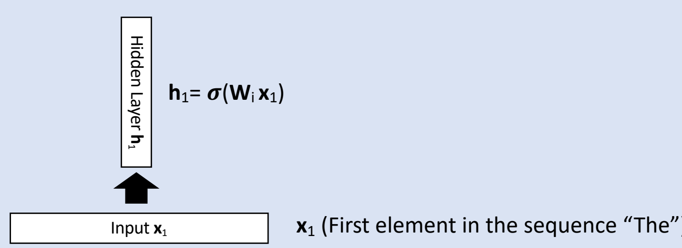
- to consider how the network processes a sequence, we can view it "unrolled"
- the network can be trained by "backpropagation through time" (i.e. running backprop on the unrolled network)



7

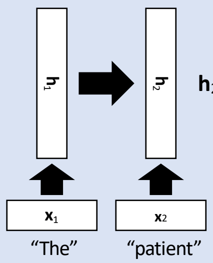
Recurrent Neural Networks (thanks Ed Choi, Jimeng Sun!)

- A recurrent neural network (RNN) for binary classification



Recurrent Neural Networks

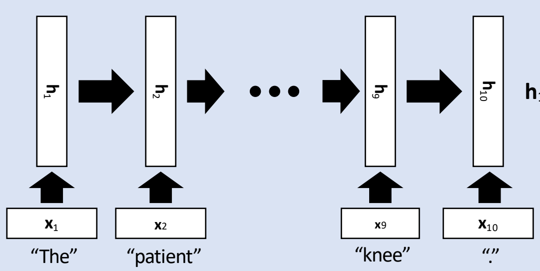
- A recurrent neural network (RNN) for binary classification



$\mathbf{h}_2 = \sigma(\mathbf{W}_h \mathbf{h}_1 + \mathbf{W}_i \mathbf{x}_2)$

Recurrent Neural Networks

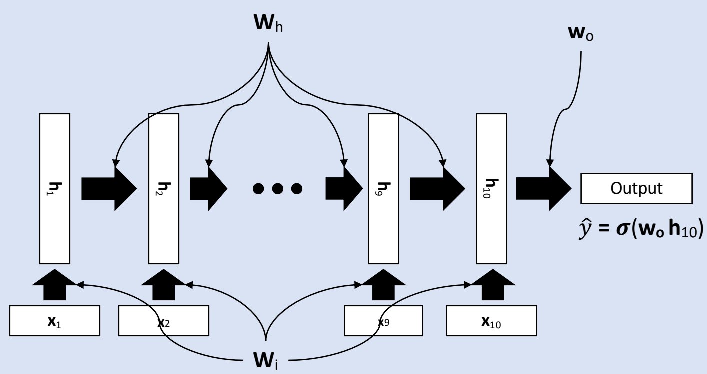
- A recurrent neural network (RNN) for binary classification



$\mathbf{h}_{10} = \sigma(\mathbf{W}_h \mathbf{h}_9 + \mathbf{W}_i \mathbf{x}_{10})$

Recurrent Neural Networks

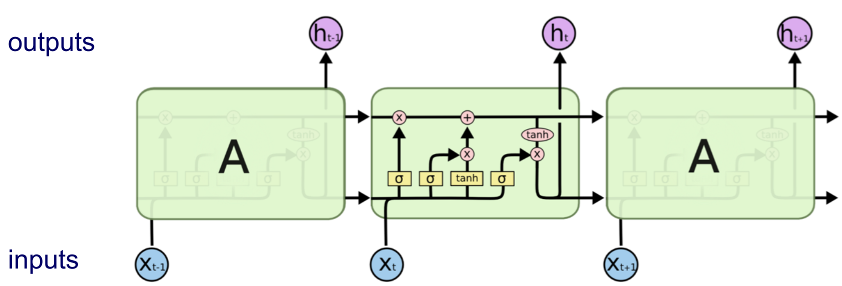
- A recurrent neural network (RNN) for binary classification



Long short term memory (LSTM) networks

[Hochreiter & Schmidhuber 1997]

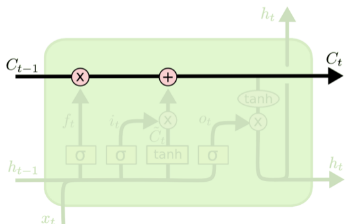
- LSTMs are recurrent networks that have a specialized architecture to enable learning what information is useful to remember/forget
- They are composed of repeating modules



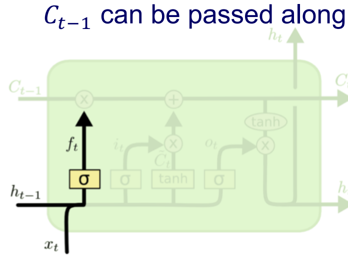
LSTM figures from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM networks

each module passes a *cell state* C_t to the next module



the *forget gate layer* determines which part of C_{t-1} can be passed along



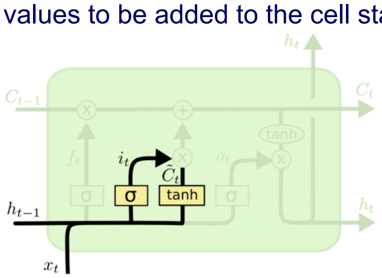
$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f)$$

vector of values in [0,1]

LSTM networks

the *input gate layer determines which part of* C_{t-1} will be updated

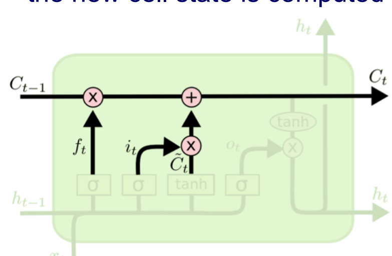
$\tilde{C}_t$ represents new candidate values to be added to the cell state



$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i)$$

$$\widetilde{\boldsymbol{C}}_t = tanh(\boldsymbol{W}_C[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_C)$$

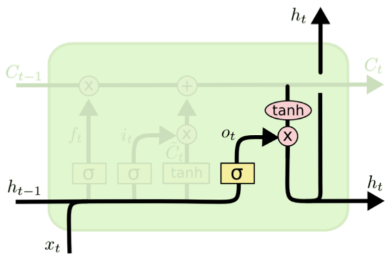
the new cell state is computed



$$\boldsymbol{C}_t = \boldsymbol{f}_t \otimes \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \otimes \tilde{C}_t$$

LSTM networks

the output h_t is computed as a function of the updated cell state

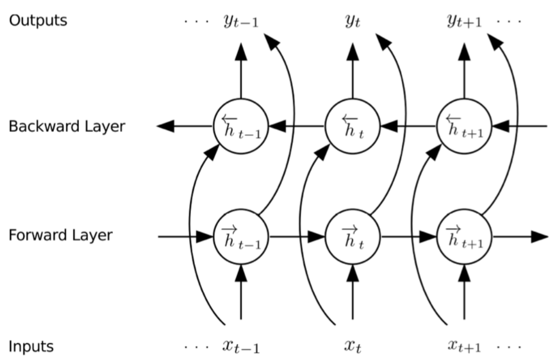


$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o)$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \otimes tanh(\boldsymbol{C}_t)$$

Bidirectional LSTM networks

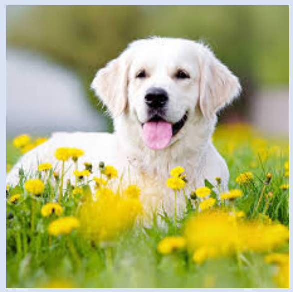
- there are many variations on the LSTM architecture, including the bidirectional LSTM (BLSTM)
- BLSTM models have provided state of the art accuracy in many NLP tasks



Here $\vec{h}_t$ represents a forward cell state, $\overleftarrow{h}_t$represents a backward cell state, and y_t represents an output

Convolutional Neural Networks (CNNs)

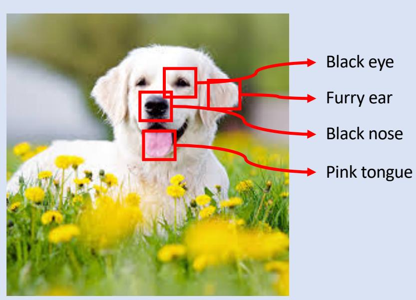
What makes a dog a dog?



Convolutional Neural Networks (CNNs)

What makes a dog a dog?

- Focus on the local features, build up global features



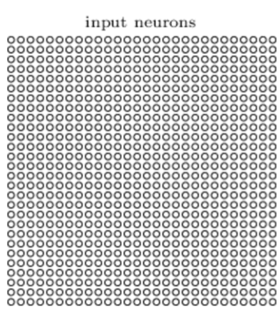
Convolutional neural nets

- well suited to tasks in which the input has spatial structure, such as images or sequences

- based on four key ideas
  - local receptive fields
  - weight sharing (parameter tying)
  - pooling
  - multiple layers of hidden units

Convolutional neural nets

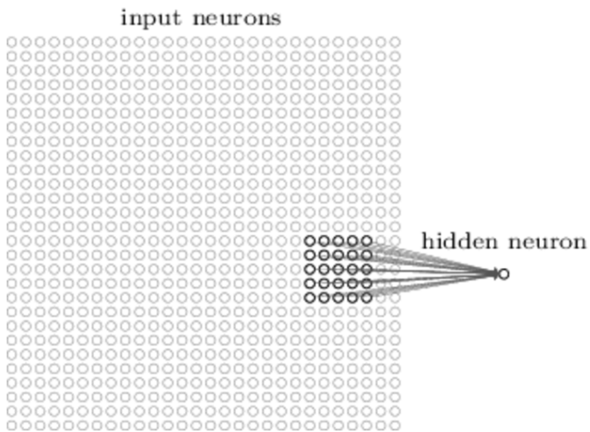
- suppose we have a task in which are instances are 28×28 pixel images
- we can represent each using 28×28 input units



[Figure from neuralnetworksanddeeplearning.com]

Convolutional neural nets

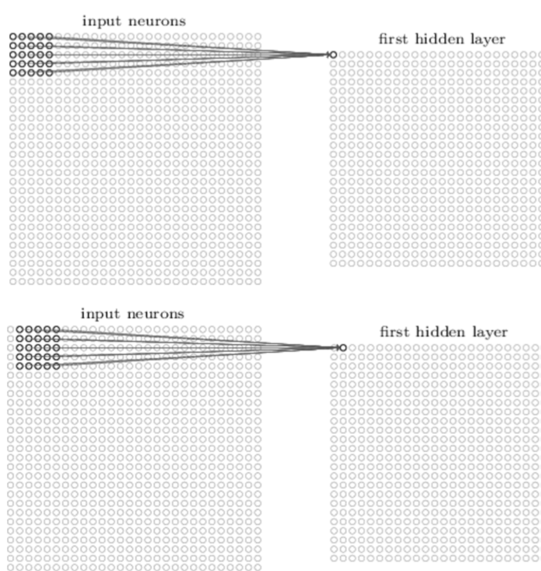
- we can connect hidden units so that each has a *local receptive field* (e.g. a 5 × 5 patch of the image).



[Figure from neuralnetworksanddeeplearning.com]

Convolutional neural nets

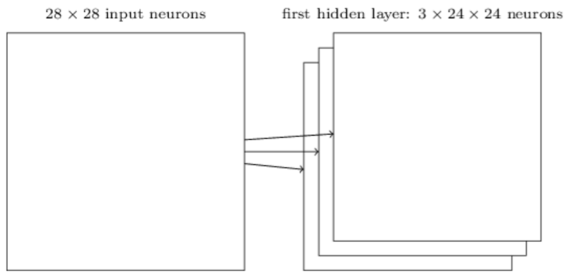
- we can have a set of these units that differ in their local receptive field
- all of the units share the same set of weights
- so the units detect same "feature" in the image, but at different locations



[Figure from neuralnetworksanddeeplearning.com]

Convolutional neural nets

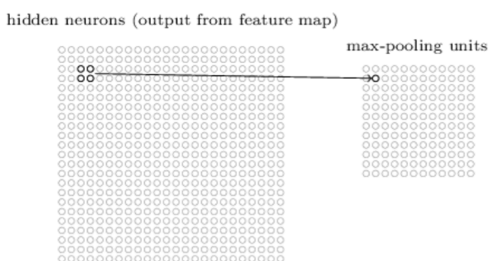
- a set of units that detect the same "feature" is called a *feature map*
- typically we'll have multiple feature maps in each layer



[Figure from neuralnetworksanddeeplearning.com]

Convolutional neural nets

- feature-map layers are typically alternated with pooling layers
- each unit in a pooling layer outputs a max, or similar function, of a subset of the units in the previous layer



$$f(\boldsymbol{x}) = max(x_1 \dots x_i \dots)$$

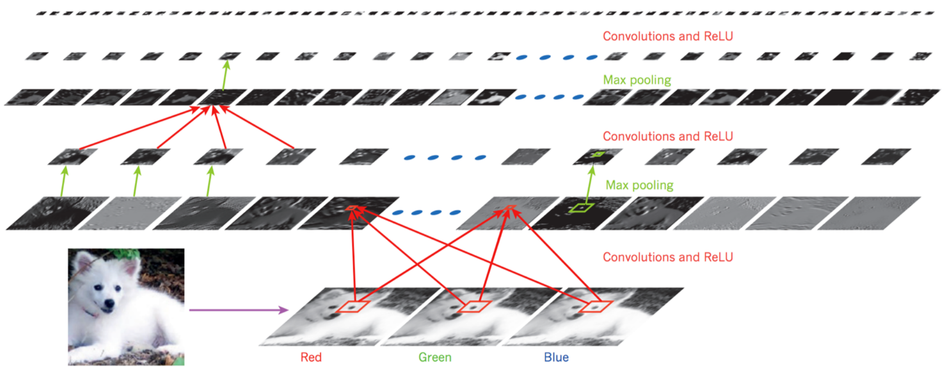
$$f(\boldsymbol{x}) = log \sum_i e^{x_i}$$

$$f(\boldsymbol{x}) = \sqrt{\sum_i x_i^2}$$

[Figure from neuralnetworksanddeeplearning.com]

Convolutional neural nets

- alternating layers of convolutional and pooling layers can be stacked



[Figure from LeCun et al., *Nature* 2015]

Convolution Operator (Already Learned)

u:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

w:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

To detect a given type of feature in an image, we can use a convolution operator, as represented by **w**

Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

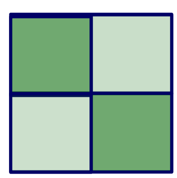
w:

1	0
0	1

Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1



Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

2		

Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

2	1	

Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

2	1	2

31

Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

2	1	2
1		

32

Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

2	1	2
1	2	

Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

2	1	2
1	2	0

Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

2	1	2
1	2	0
0		

35

Convolution Operator (Already Learned)

u:

1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

2	1	2
1	2	0
0	1	

36

Convolution Operator (Already Learned)

u:

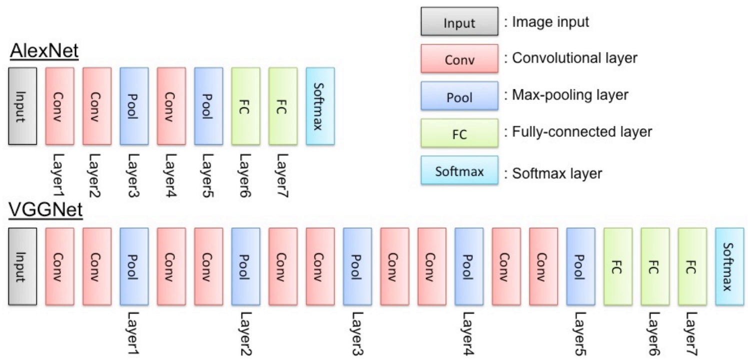
1	1	1	0
0	1	0	1
0	1	1	0
1	0	0	1

2	1	2
1	2	0
0	1	2

Notes on convolution example

- Repeating twice more and max pooling would yield an 8, which tells us there is a continuous diagonal edge (if we assume “1” is a darkened pixel)
- Many ways to define how to handle boundary case (e.g. by padding the image); we ignored this issue
- We used a 2x2 *span* and a *stride* of 1 (overlapping fields)

Convolutional neural networks



AlexNet: Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. NIPS 2012
VGGNet: Karen Simonyan, and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. ICLR 2015