

Deep Learning

CS 4300/6300

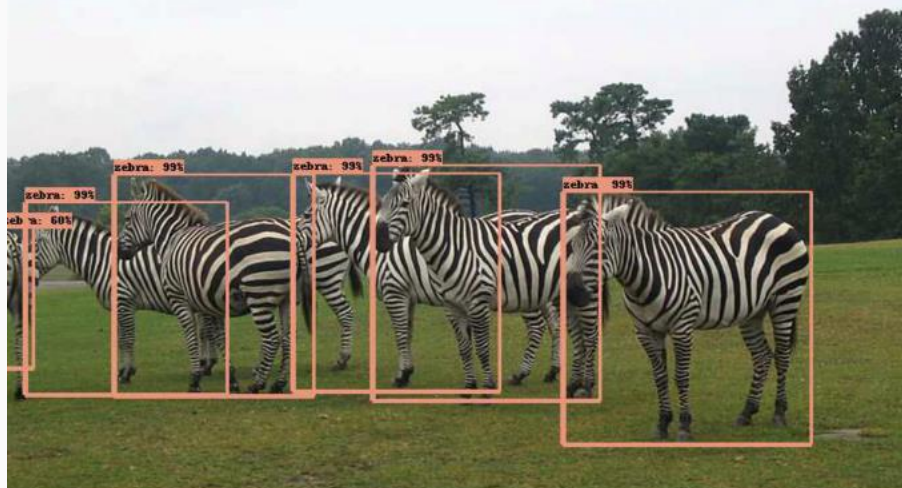
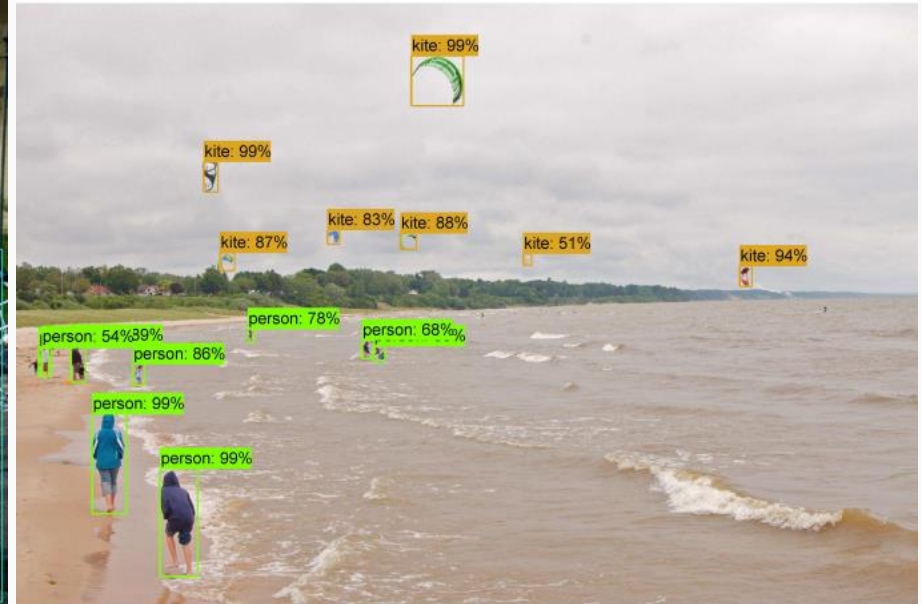
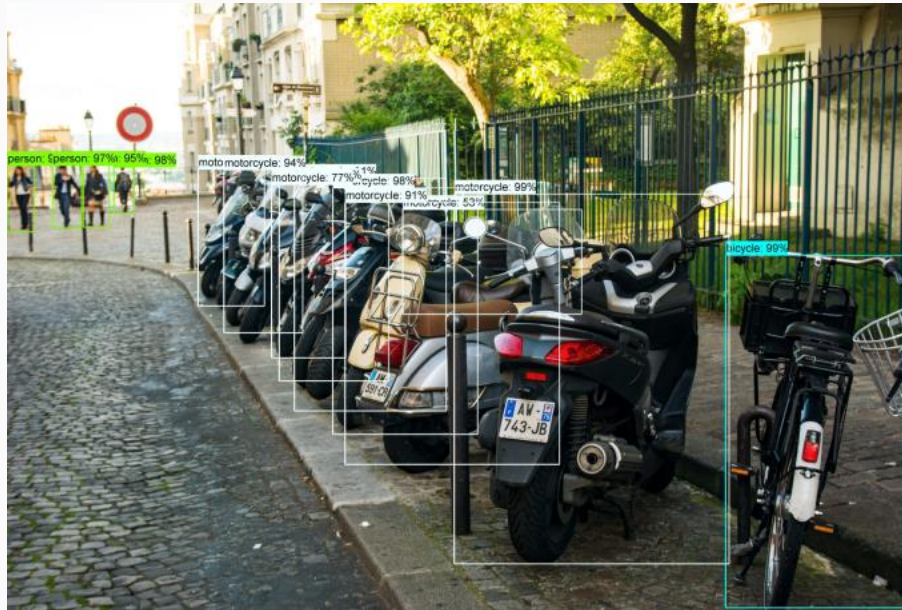
Daniel Brown



[Slide content from Vivek Srikumar]


Success stories

Object recognition




Images from Zoph, Barret, et al. "Learning transferable architectures for scalable image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

Superhuman speech recognition

 Microsoft | The AI Blog Our Company ▾ News and Stories ▾ Press Tools ▾

Historic Achievement: Microsoft researchers reach human parity in conversational speech recognition

October 18, 2016 | [Allison Linn](#)

 NEWS ▾ JOBS EVENTS ▾ LISTS ▾ MEMBERS ▾ STUDIOS ▾ ABOUT ▾

Trending: After \$6.6B round, OpenAI investor says the next step for ChatGPT maker should be to go public

Microsoft claims new speech recognition record, achieving a super-human 5.1% error rate

BY TODD BISHOP on August 20, 2017 at 7:44 pm

Captions generated by a neural network



several people are on a dock in the water .



black and white dog jumps over bar

And many more successes

You have heard about them and likely used them



ChatGPT



Google Translate

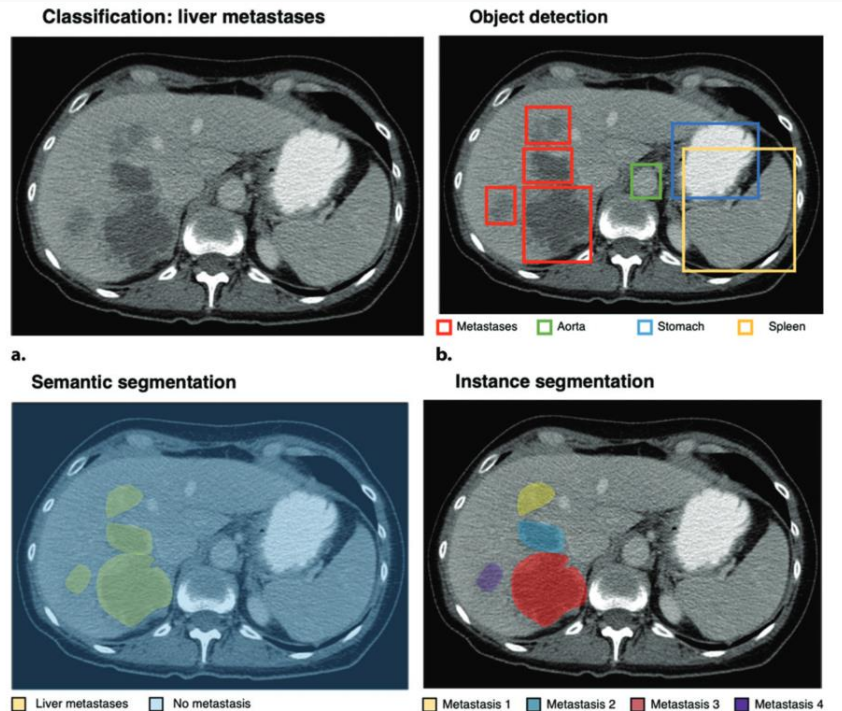


Siri

A class of tools impacting diverse domains

Google DeepMind's new AI system can solve complex geometry problems

Its performance matches the smartest high school mathematicians and is much stronger than the previous state-of-the-art system.



AlphaFold – for predicting protein structures

2023 Albert Lasker Basic Medical Research Award



Demis Hassabis
Google DeepMind



John Jumper
Google DeepMind

Why deep learning?

Machine learning is easy

We just need a lot of data and learning will be successful...

... *provided we have the right features*

Even these functions can be *made* linear

These points are not separable in 1-dimension by a line

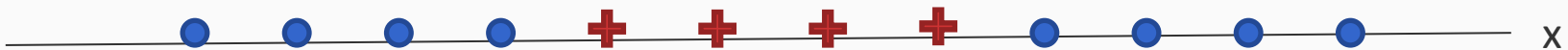
What is a one-dimensional line, by the way?



Even these functions can be *made* linear

These points are not separable in 1-dimension by a line

What is a one-dimensional line, by the way?



The trick: Change the representation

The blown up feature space

The trick: Use feature *conjunctions*

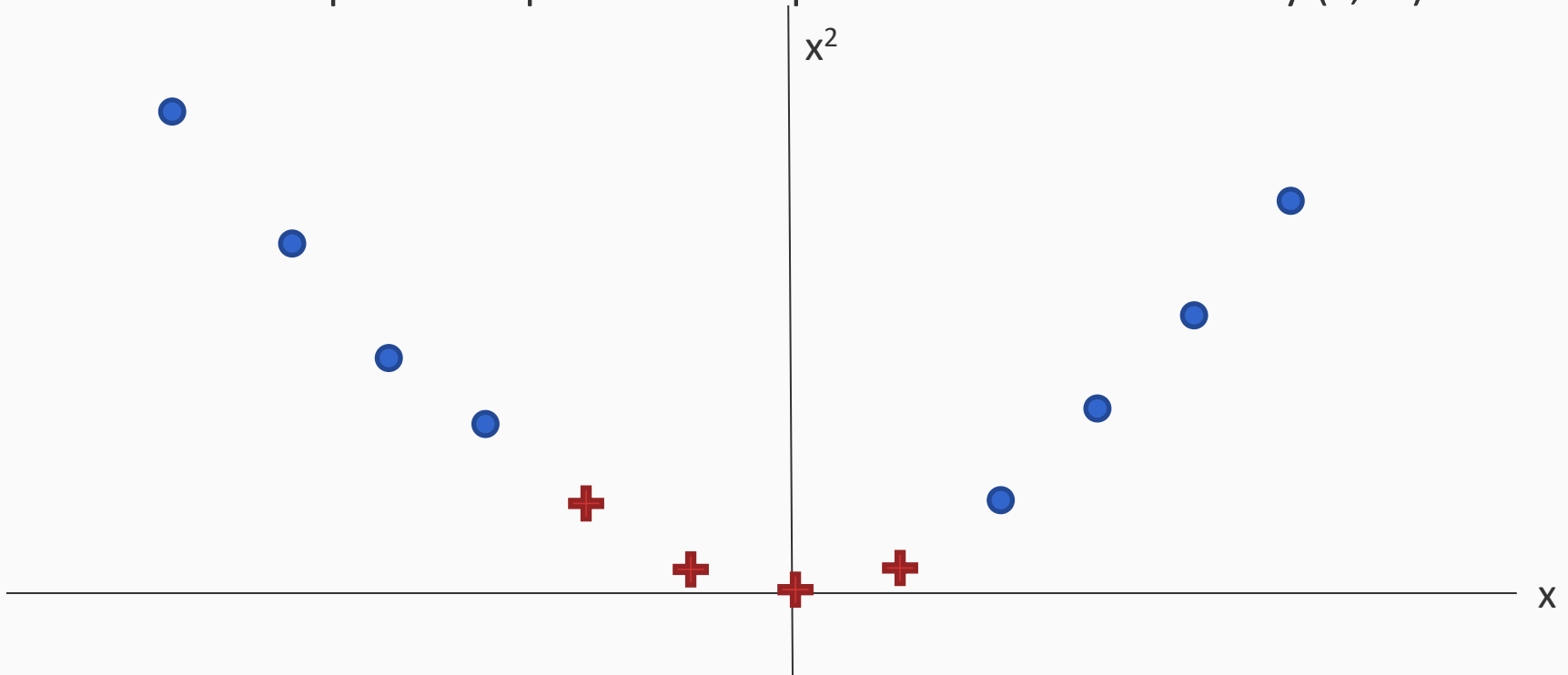
Transform points: Represent each point x in 2 dimensions by (x, x^2)



The blown up feature space

The trick: Use feature *conjunctions*

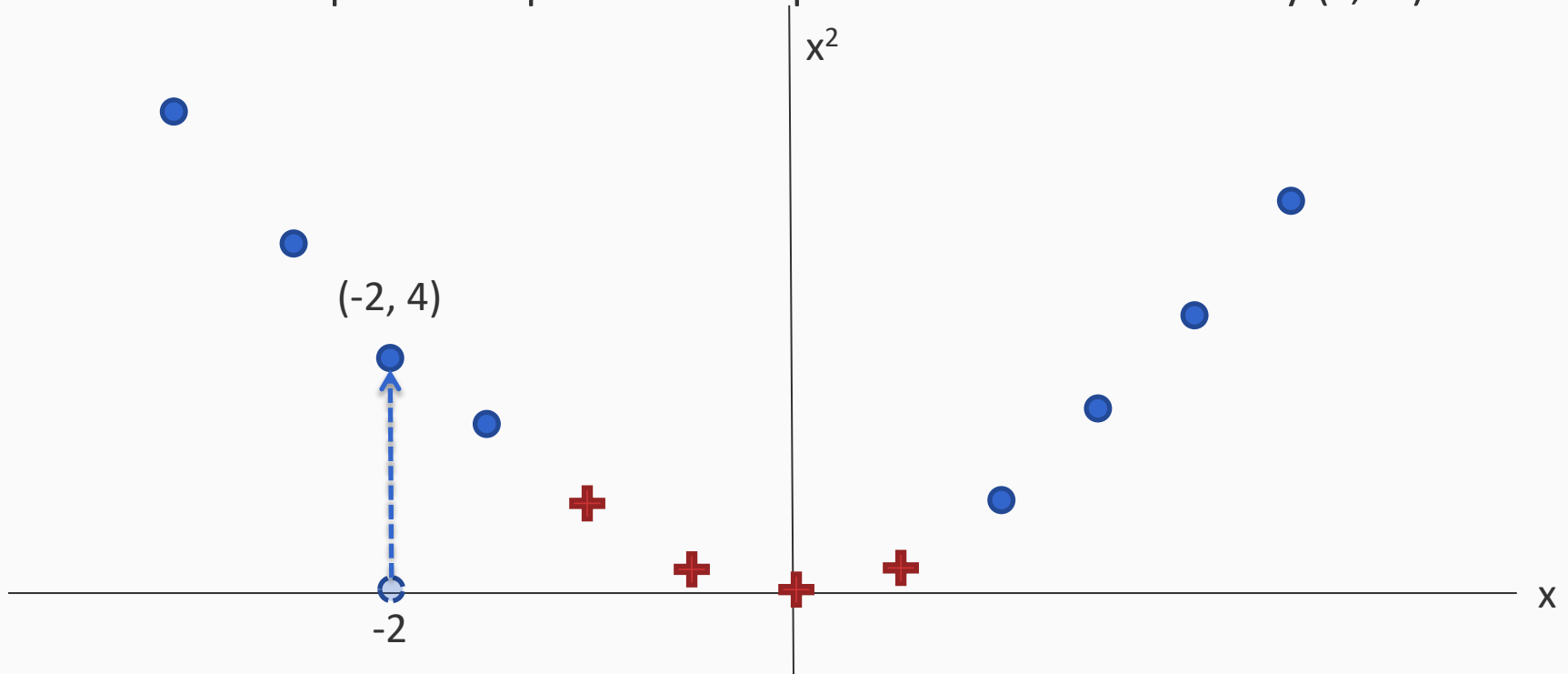
Transform points: Represent each point x in 2 dimensions by (x, x^2)



The blown up feature space

The trick: Use feature *conjunctions*

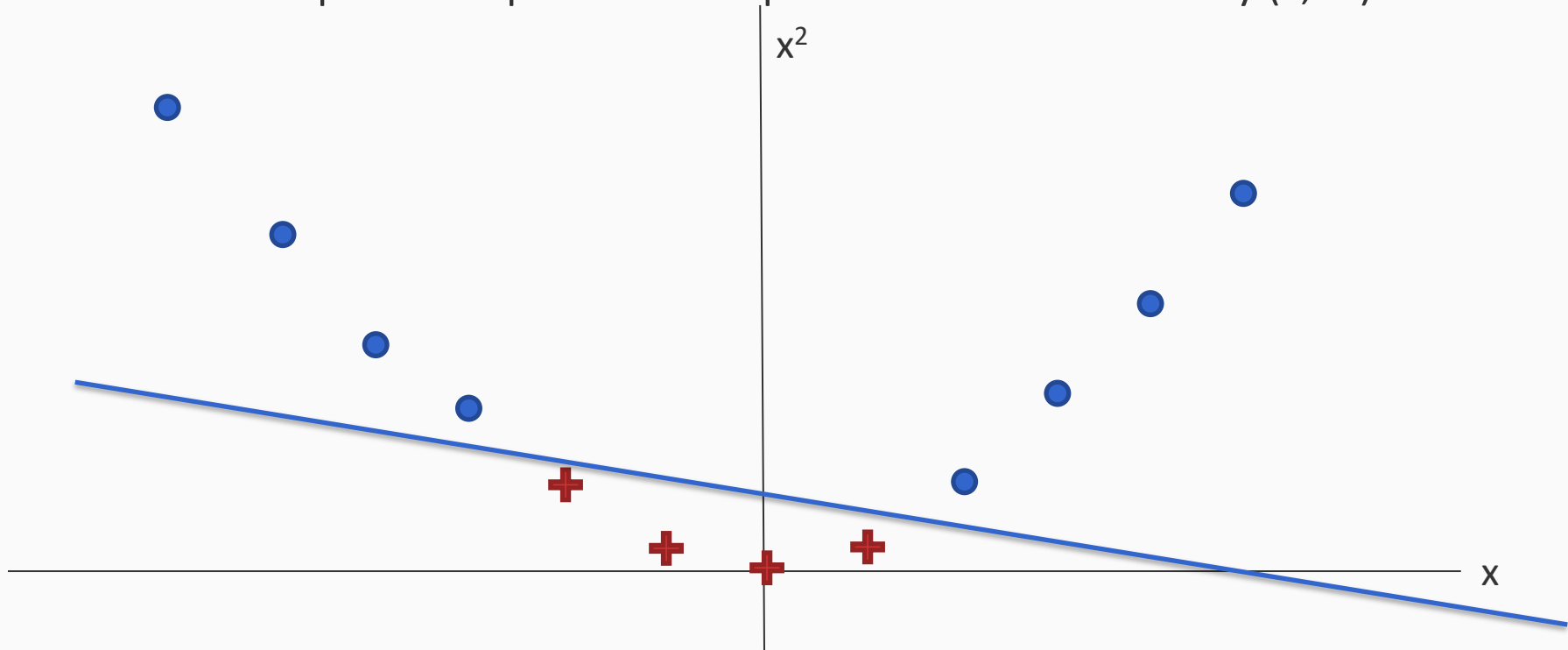
Transform points: Represent each point x in 2 dimensions by (x, x^2)



The blown up feature space

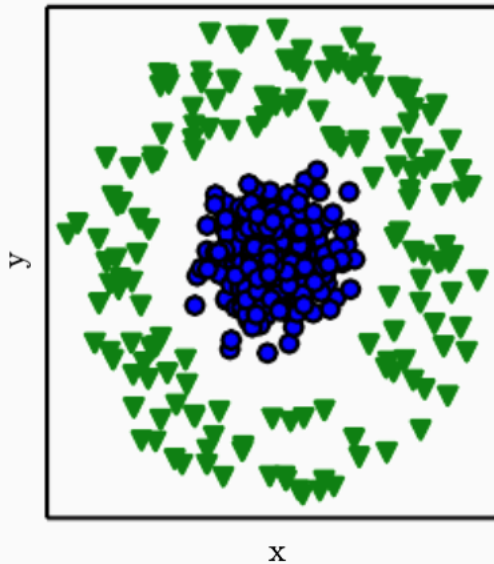
The trick: Use feature *conjunctions*

Transform points: Represent each point x in 2 dimensions by (x, x^2)



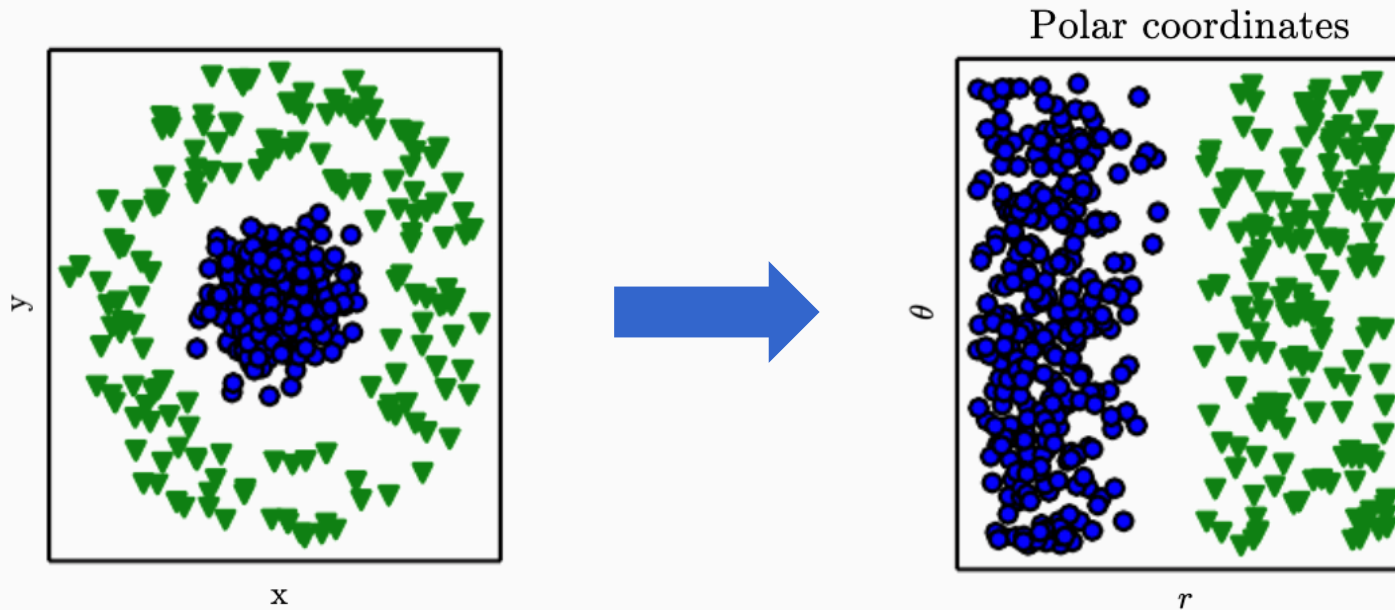
Now the data is linearly separable in this space!

A toy example



We cannot separate the blue circles from the green triangles

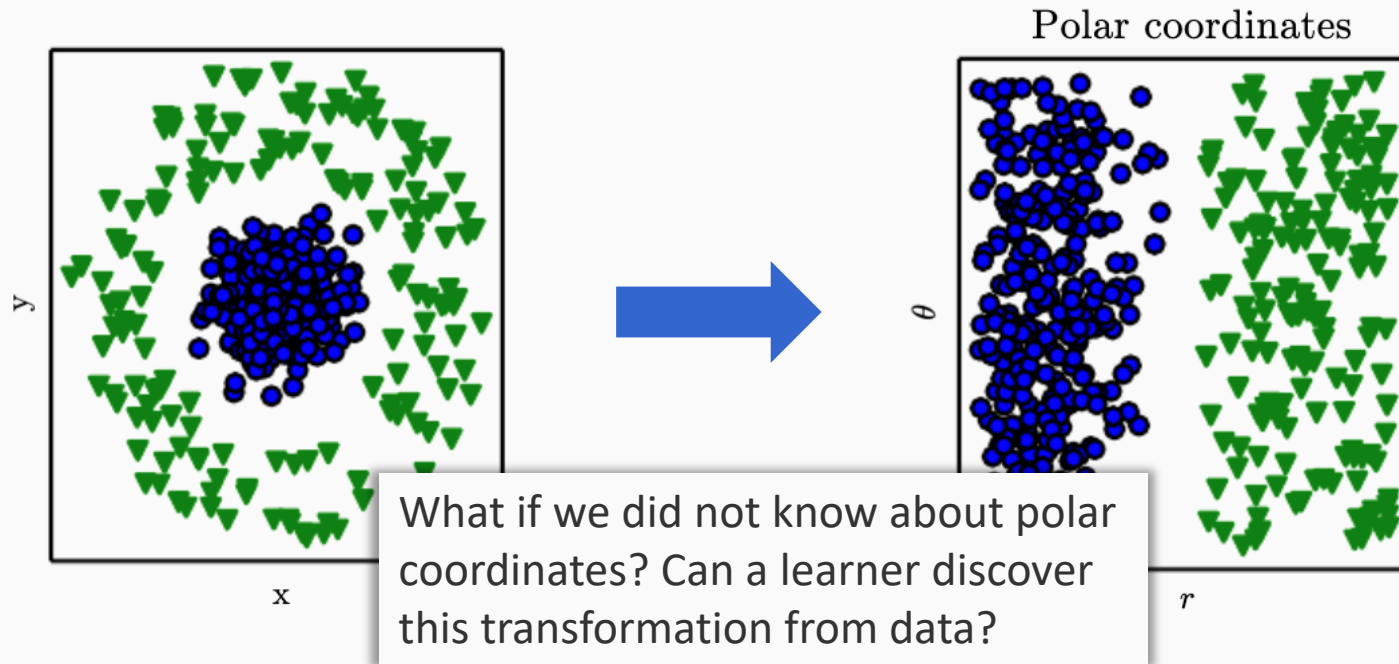
A toy example



We cannot separate the blue circles from the green triangles
in the original representation

But by going to **polar coordinates**, the separator is a line

A toy example



We cannot separate the blue circles from the green triangles
in the original representation

But by going to **polar coordinates**, the separator is a line

What is deep learning?

What is deep learning?

A diverse collection of ideas that are centered around the use of neural networks for machine learning

Some common design patterns

1. Learned distributed representations
2. Composing layers of neurons and training them end-to-end
3. “Differentiable compute”, which allows the use of backpropagation

Hierarchy of learned representations

Rather than hand-crafted representations, make feature extraction into a learning problem

Optimize for the final task and the features together

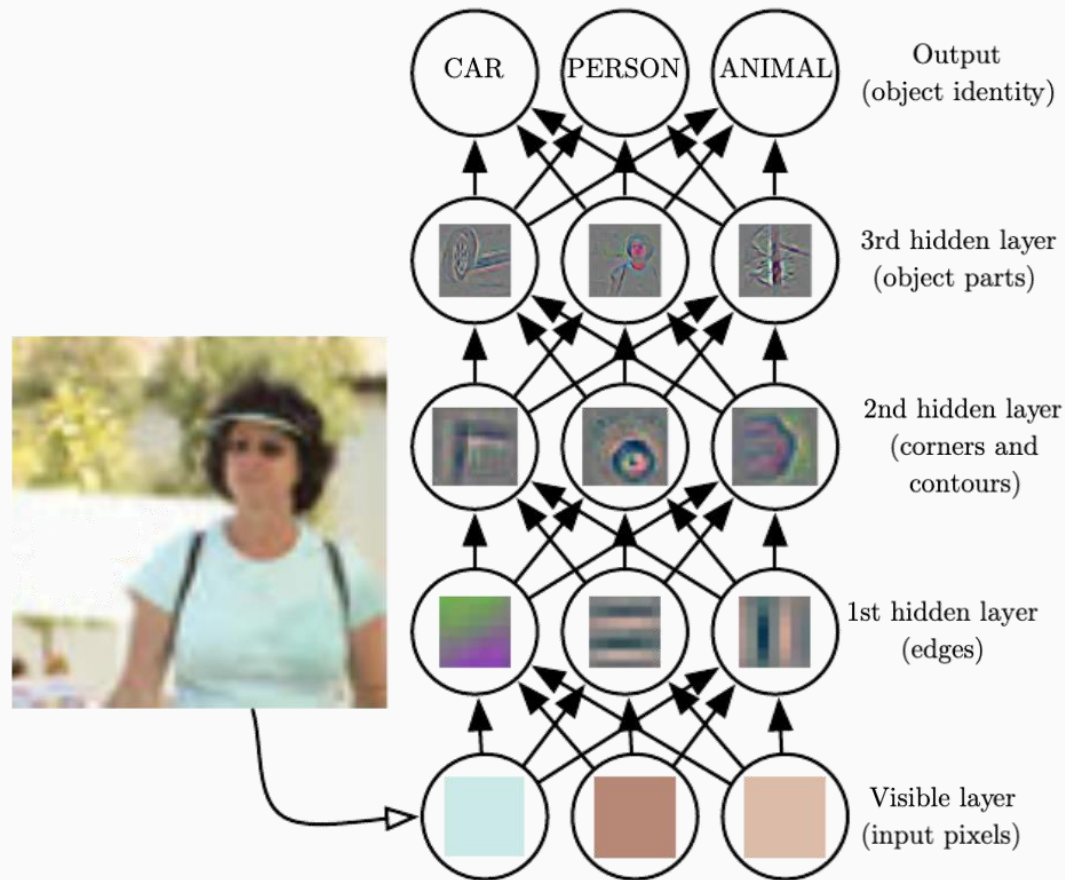
Create *a hierarchy of layers* of increasing abstraction

Why should this work?

“Because gradient descent is better than you”

— A possibly apocryphal quote from Yann Le Cun about not hand-defining features

A deep learning model illustrated

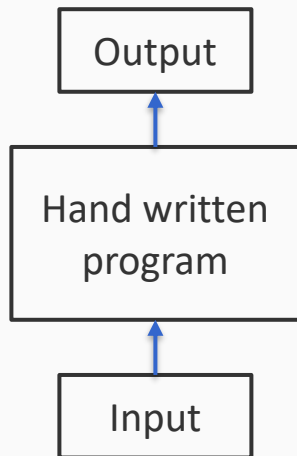


Deep learning in context

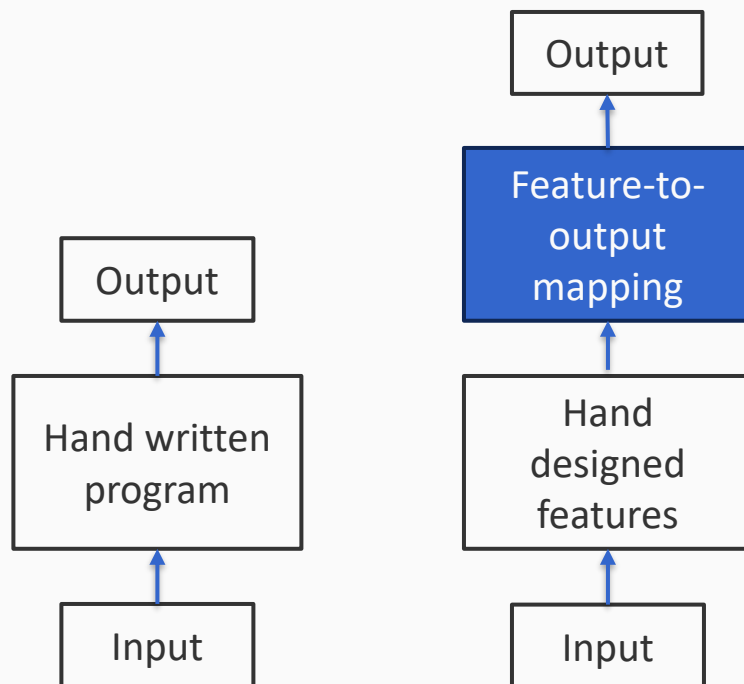
Traditional programs are hand written.

A very successful agenda that gave birth to the computer revolution.

Success stories: Operating systems, the Internet, web browsers, etc



Deep learning in context



Traditional machine learning:

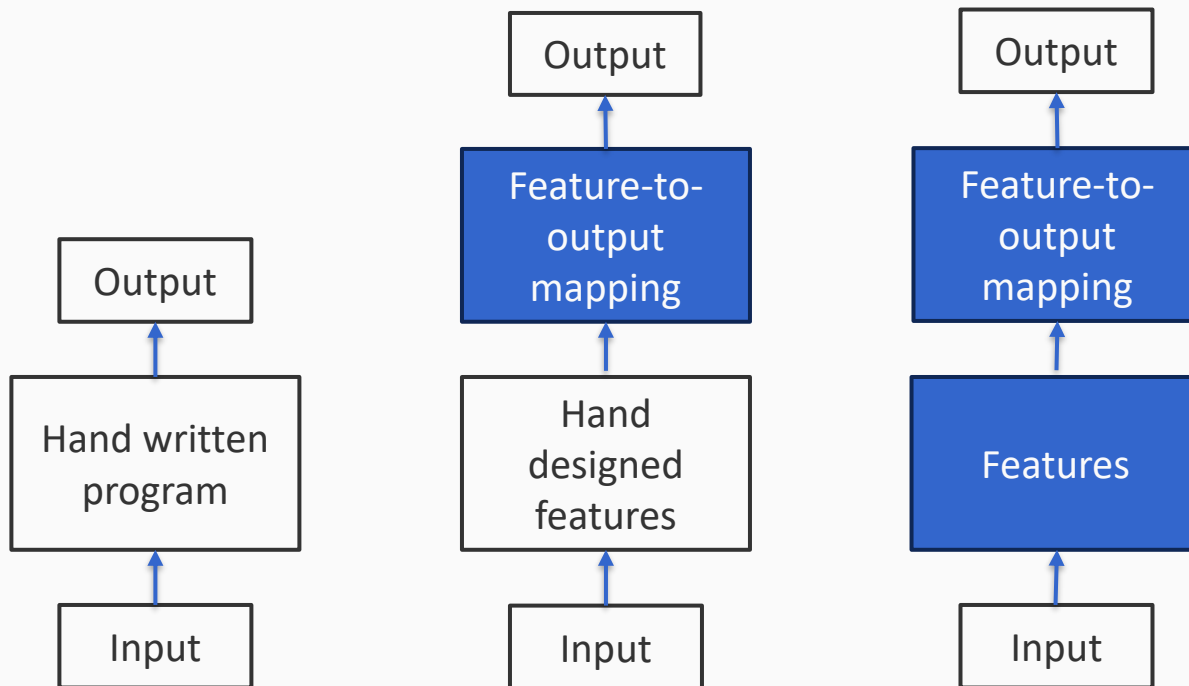
Hand designed features of inputs are given to a learner that learns to map features to inputs

(the blue part is learned)

Deep learning in context

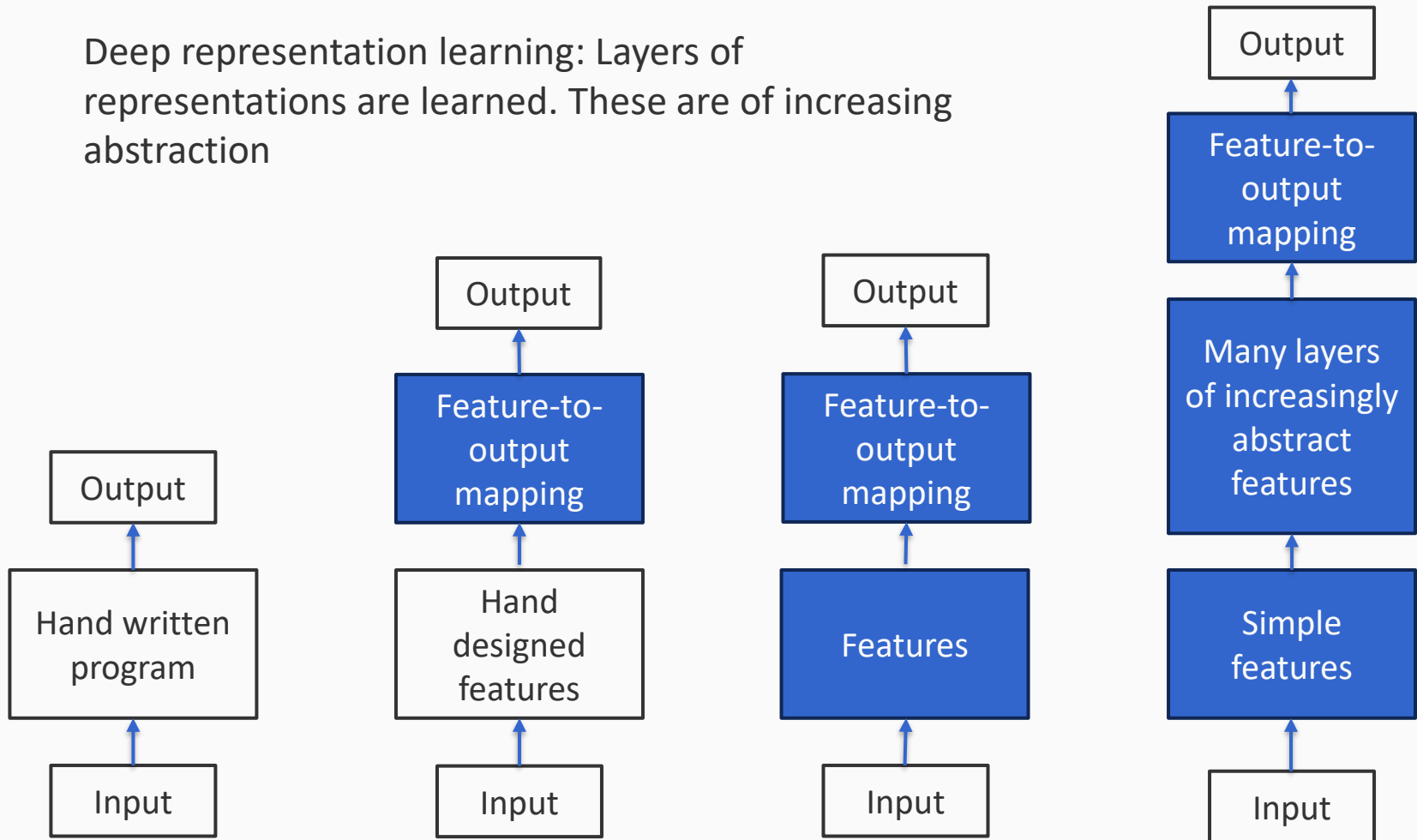
Shallow representation learning:

The hand-designed features are also part of the learned component (blue)



Deep learning in context

Deep representation learning: Layers of representations are learned. These are of increasing abstraction



A neural network

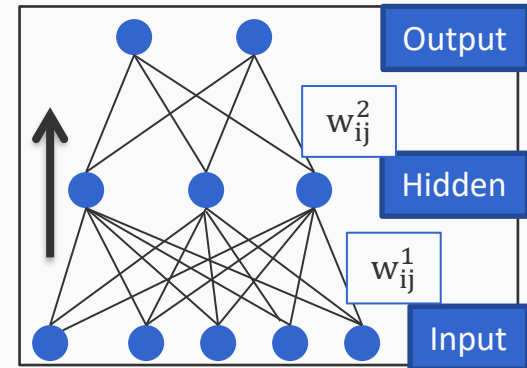
A function that converts inputs to outputs defined by a **directed acyclic graph**

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights

A neural network

A function that converts inputs to outputs defined by a **directed acyclic graph**

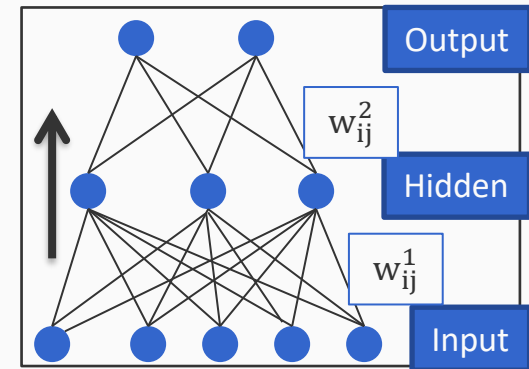
- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



A neural network

A function that converts inputs to outputs defined by a **directed acyclic graph**

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



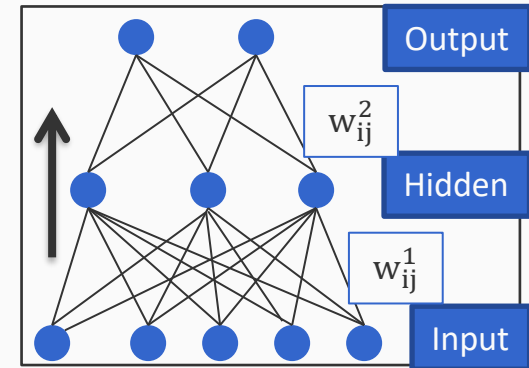
To define a neural network, we need to specify:

- The structure of the graph
 - How many nodes, the connectivity
- The activation function on each node
- The edge weights

A neural network

A function that converts inputs to outputs defined by a **directed acyclic graph**

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



To define a neural network, we need to specify:

- The structure of the graph
 - How many nodes, the connectivity
- The activation function on each node
- The edge weights

Called the **architecture** of the network

Typically predefined, part of the design of the classifier

Learned from data

Computation graphs

A language for constructing deep neural networks and loss functions

- A way to think about *differentiable compute*

Key ideas:

- We can represent functions as graphs
- We can dynamically generate these graphs if necessary
- We can define algorithms over these graphs that map to learning and prediction
 - Prediction via the forward pass
 - Learning via gradients computed using the backward pass

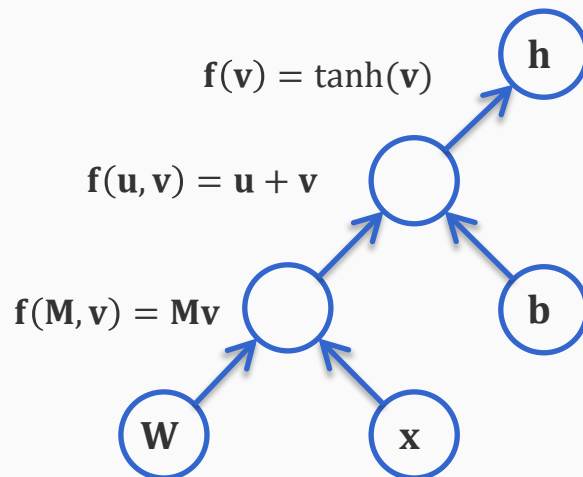
An example two layer neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$

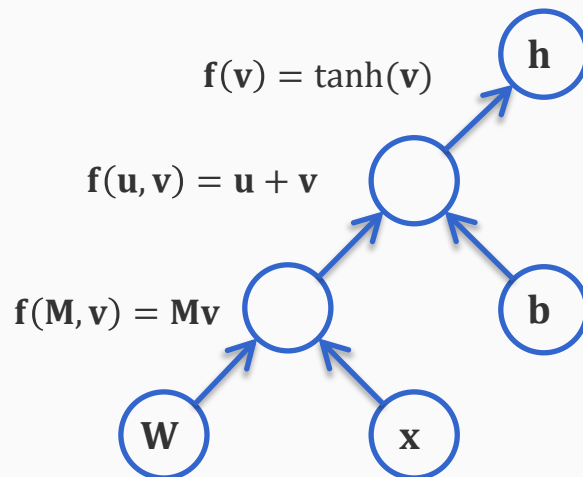
An example two layer neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$



An example two layer neural network

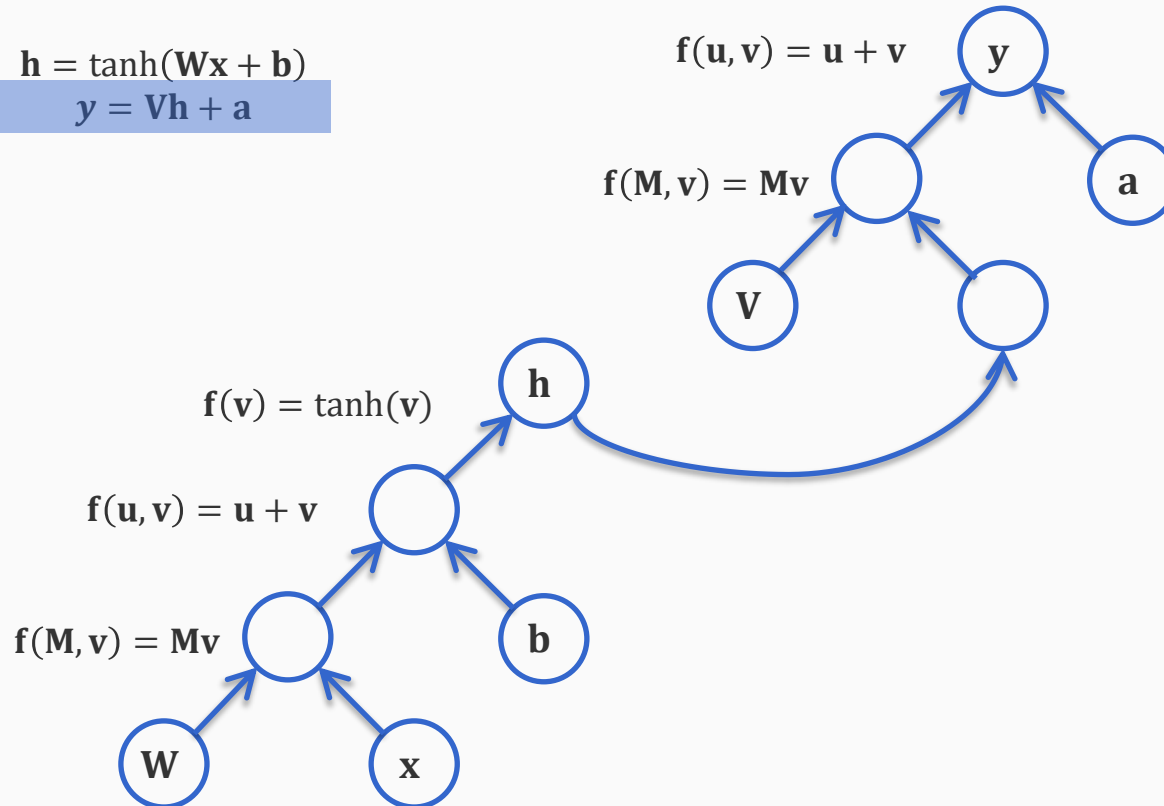
$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$



An example two layer neural network

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$



Two algorithmic questions

1. Forward propagation

- Given inputs to the graph, compute the value of the function expressed by the graph
- Something to think about: Given a node, can we say which nodes are inputs? Which nodes are outputs?

2. Backpropagation

- After computing the function value for an input, compute the gradient of the function at that input
- Or equivalently: *How does the output change if I make a small change to the input?*

Forward propagation

Given a computation graph G and values of its input nodes:

For each node in the graph, in **topological order**:

 Compute the value of that node

Why topological order: Ensures that children are computed before parents.

Two algorithmic questions

1. Forward propagation

- Given inputs to the graph, compute the value of the function expressed by the graph
- Something to think about: Given a node, can we say which nodes are inputs? Which nodes are outputs?

2. Backpropagation

- After computing the function value for an input, compute the gradient of the function at that input
- Or equivalently: *How does the output change if I make a small change to the input?*

Backpropagation, in general

After we have done the forward propagation,

Loop over the nodes in **reverse topological order** starting with a final goal node

- Compute derivatives of final goal node value with respect to each edge's tail node
 - If there are multiple outgoing edges from a node, sum up all the derivatives for the edges
- Save the computed gradient so that the next time we need it, it doesn't need to be re-computed

The standard process of training neural networks

1. Design the graph that defines the computation you want
2. Initialize the graph

The standard process of training neural networks

1. Design the graph that defines the computation you want
2. Initialize the graph
Either randomly, or with pre-trained parameters
3. Iterate over example (or mini-batches of examples):

The standard process of training neural networks

1. Design the graph that defines the computation you want
2. Initialize the graph
Either randomly, or with pre-trained parameters
3. Iterate over example (or mini-batches of examples):
 1. Run the forward pass to calculate the result of the computation using the current parameters

The standard process of training neural networks

1. Design the graph that defines the computation you want
2. Initialize the graph
Either randomly, or with pre-trained parameters
3. Iterate over example (or mini-batches of examples):
 1. Run the forward pass to calculate the result of the computation using the current parameters
 2. Define the loss for the network over the current example
Characterizes the idea of “how bad is the result that was just computed”

The standard process of training neural networks

1. Design the graph that defines the computation you want
2. Initialize the graph
Either randomly, or with pre-trained parameters
3. Iterate over example (or mini-batches of examples):
 1. Run the forward pass to calculate the result of the computation using the current parameters
 2. Define the loss for the network over the current example
Characterizes the idea of “how bad is the result that was just computed”
 3. Compute the gradient of the loss using backpropagation

The standard process of training neural networks

1. Design the graph that defines the computation you want
2. Initialize the graph
Either randomly, or with pre-trained parameters
3. Iterate over example (or mini-batches of examples):
 1. Run the forward pass to calculate the result of the computation using the current parameters
 2. Define the loss for the network over the current example
Characterizes the idea of “how bad is the result that was just computed”
 3. Compute the gradient of the loss using backpropagation
 4. Update the parameters

Neural networks are data-driven programs

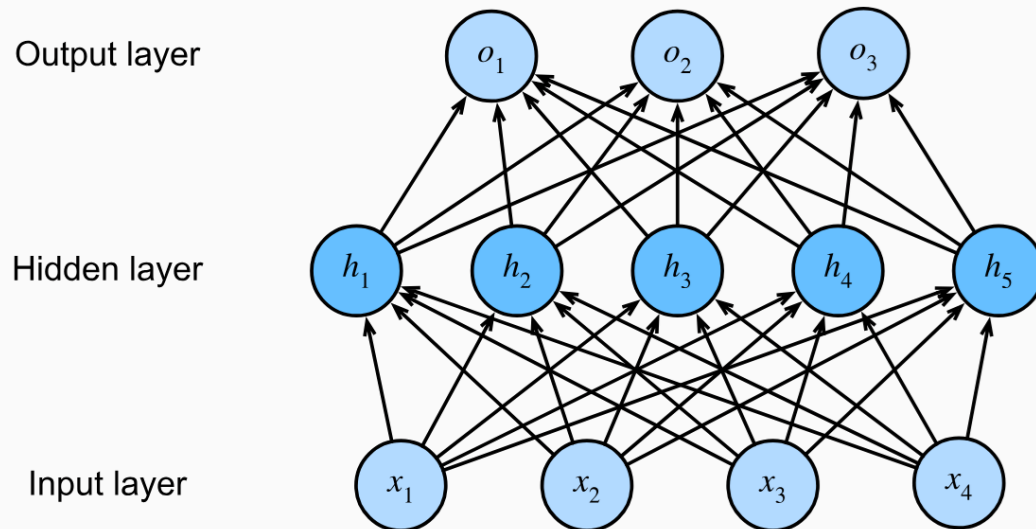
The forward pass allows us to compute the result of computations on examples

The backward pass over loss functions allows us to compute the update to the parameters that produced the loss

Both loss functions and neural networks are computation graphs

This abstraction allows us to think of neural networks as functions (in a programming sense) that will be “filled in” by data

Refresher: Multi-Layer Perceptron (MLP)



Stack many fully connected layers on top of each other

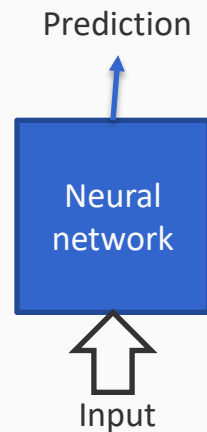
Each layer feeds into the next one

If the total number of layers is L (here $L=2$), then the first $L-1$ layers construct the representation and the last one is the linear predictor over it

Recurrent Neural Networks



Neural networks are prediction machines



We can assign labels to inputs

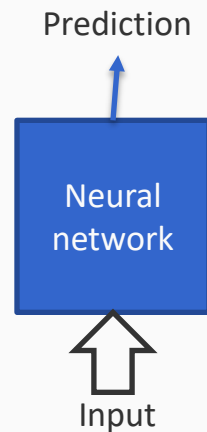


cat



burrito

Neural networks are prediction machines



We can assign labels to inputs



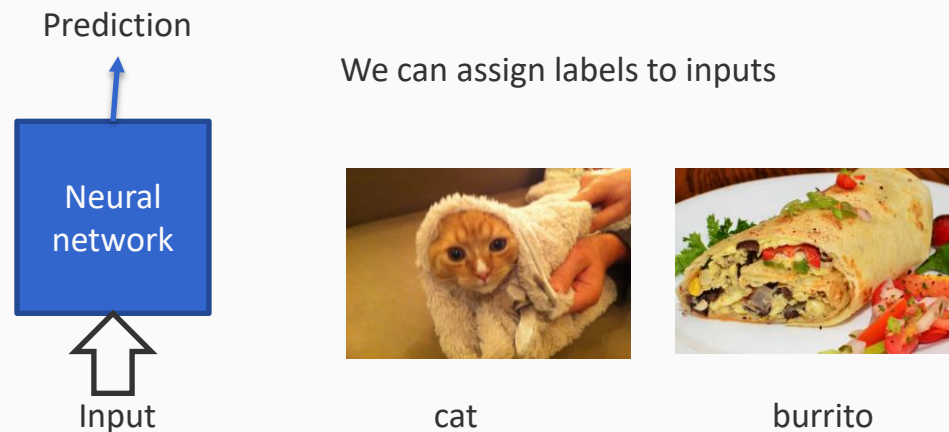
cat



burrito

But what if the label to an input depends on a previous state of the network?

Neural networks are prediction machines



But what if the label to an input depends on a previous state of the network?

Vanilla neural networks

1. Do not have persistent memory
2. Can not deal with varying sized inputs

Sequential prediction: Examples

- Language models: “It was a dark and stormy _____”
 - Constructing sentences automatically requires us to remember what we constructed before
- Speech recognition
 - Convert a sequence of audio signals to words
 - The word at time t may depend on what word was predicted at time $(t-1)$
- Event extraction from movies
 - Watch a movie and predict what events are happening
 - The events at a particular scene probably depends on both the video signal ***and*** the events that were predicted in the previous scene
- Many more examples

What does it mean to model a sequence?

Some questions:

- Given a sequence of inputs (words, stock prices, etc), predict the next item in the sequence
- Can we represent arbitrarily long sequences as fixed sized vectors?
 - Perhaps to provide features for subsequent classification

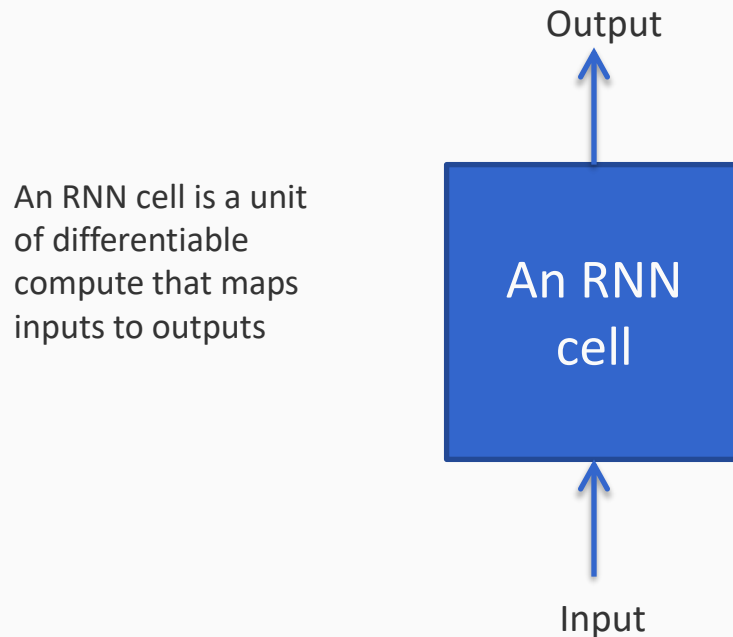
Answer: Recurrent neural networks (RNNs)

Recurrent neural networks

- First introduced by Elman 1990
- Provides a mechanism for representing sequences of arbitrary length into vectors that encode the sequential information
- A useful design abstraction if you'd like to work with sequential data
 - Till transformers came along, for a few years, RNNs were the best tools for representing text sequences

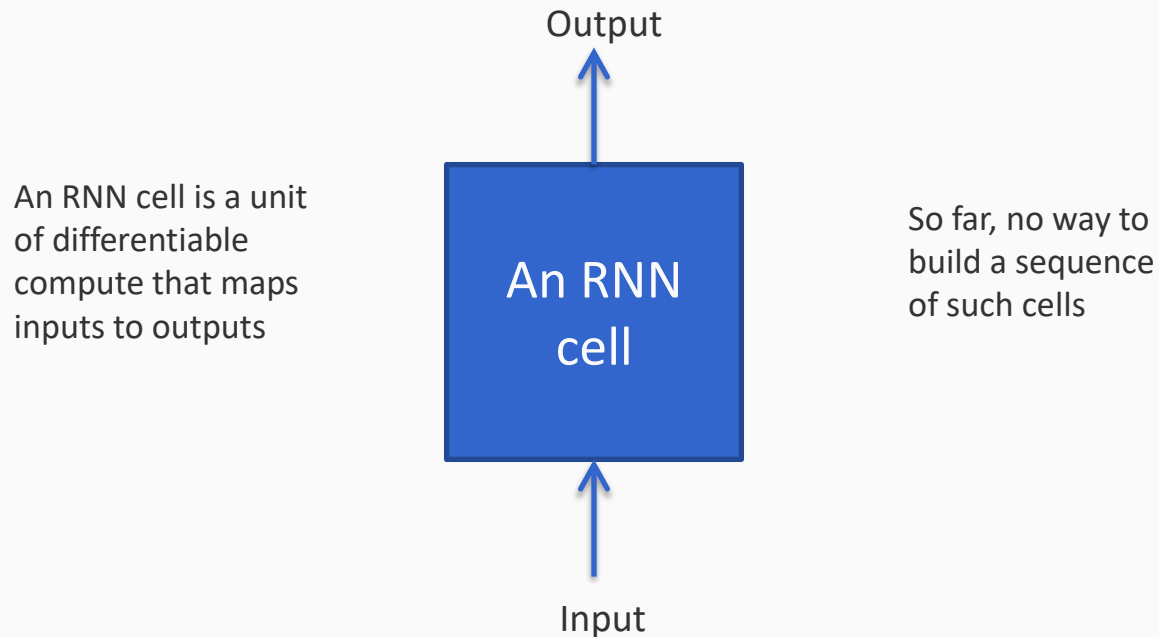
The RNN abstraction

A high level overview that doesn't go into details



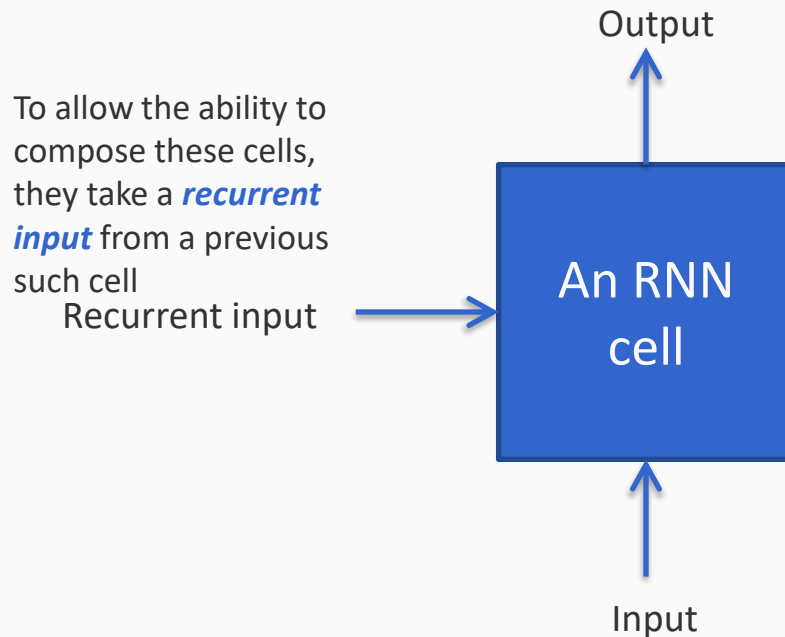
The RNN abstraction

A high level overview that doesn't go into details



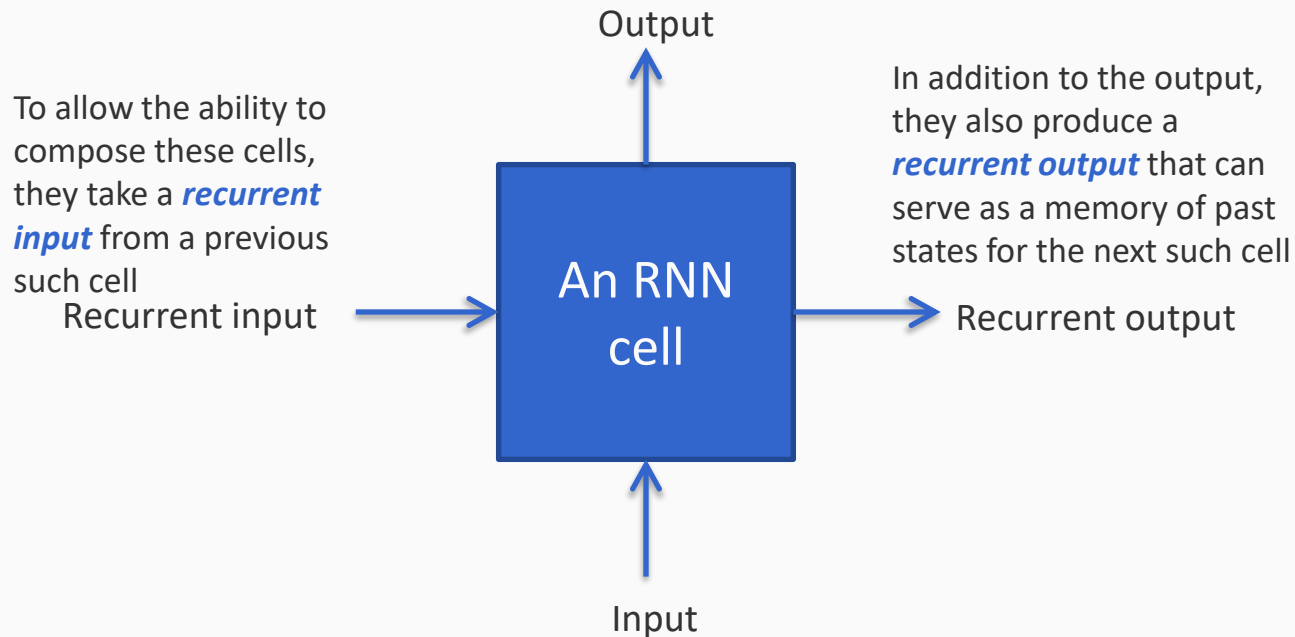
The RNN abstraction

A high level overview that doesn't go into details



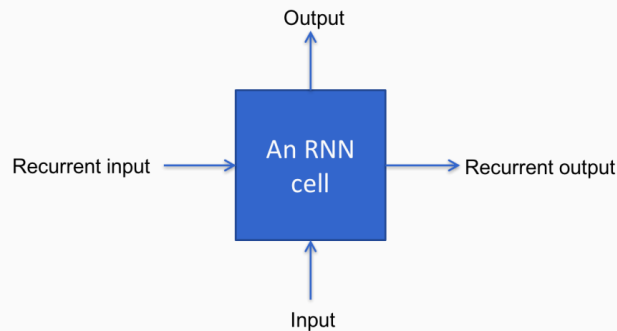
The RNN abstraction

A high level overview that doesn't go into details



The RNN abstraction

A high level overview that doesn't go into details

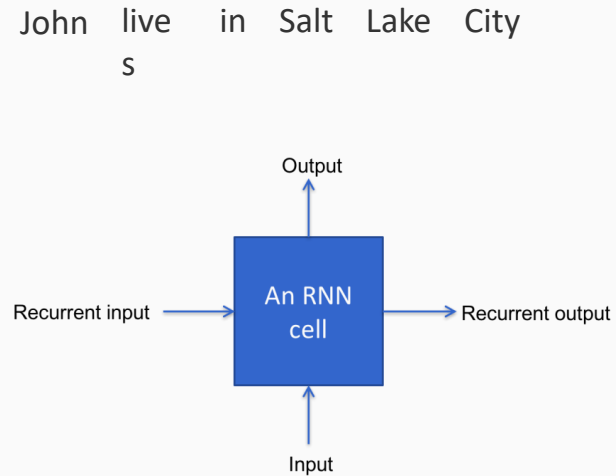


Conceptually two operations

Using the input and the recurrent input (also called the previous cell state), compute

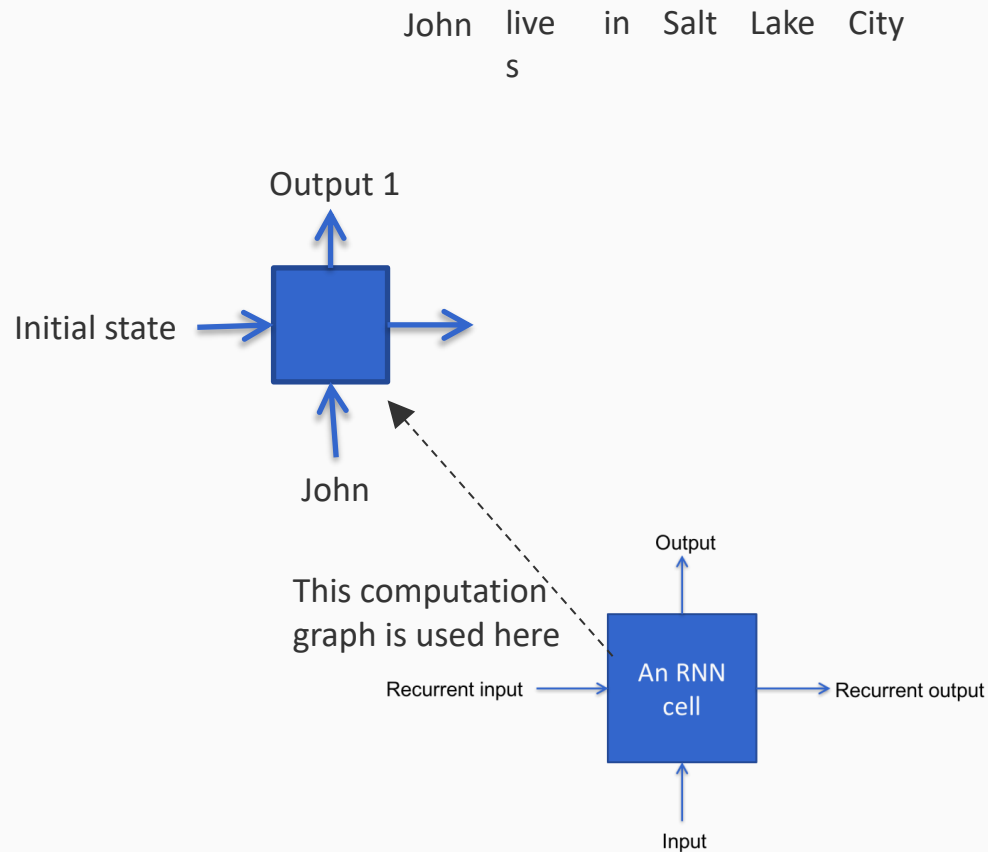
1. The next cell state
2. The output

The RNN abstraction: A simple example

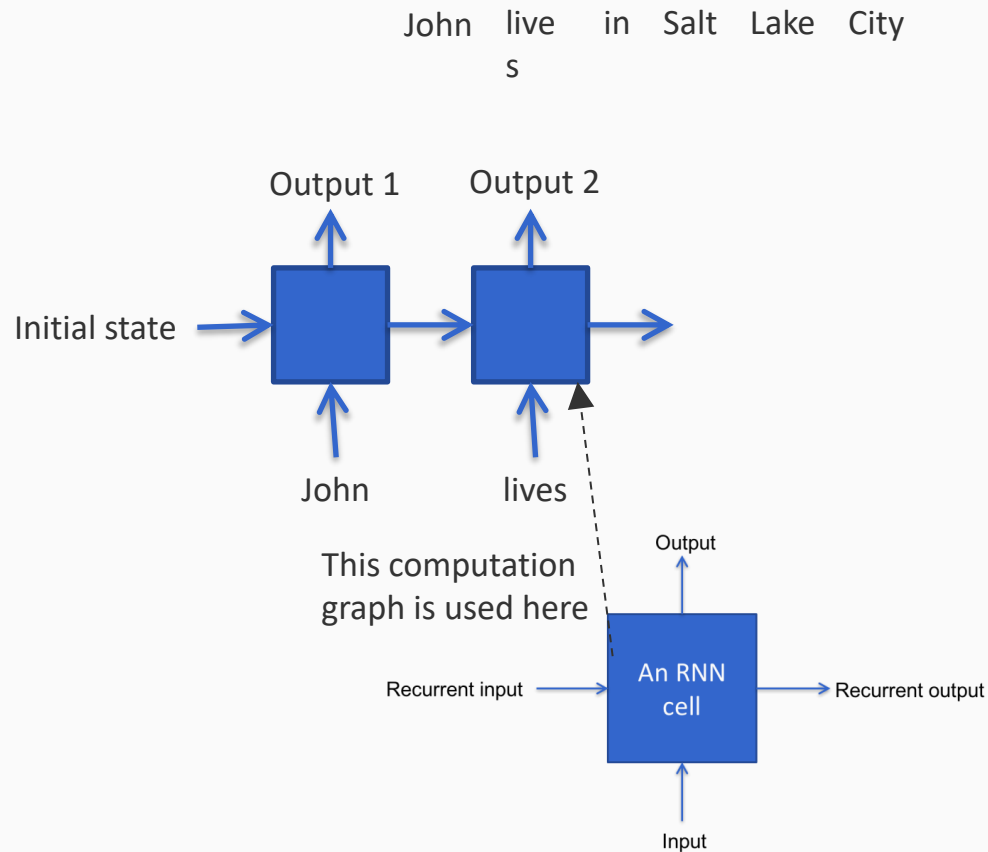


This template is **unrolled** for each input

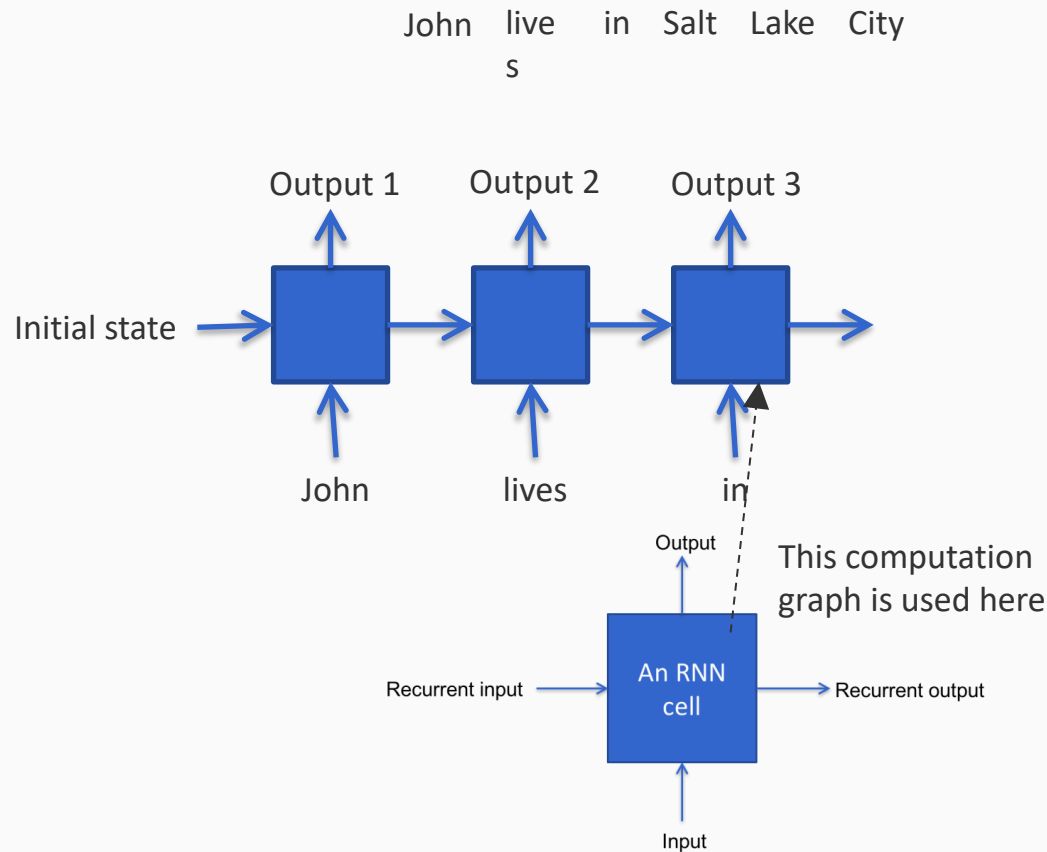
The RNN abstraction: A simple example



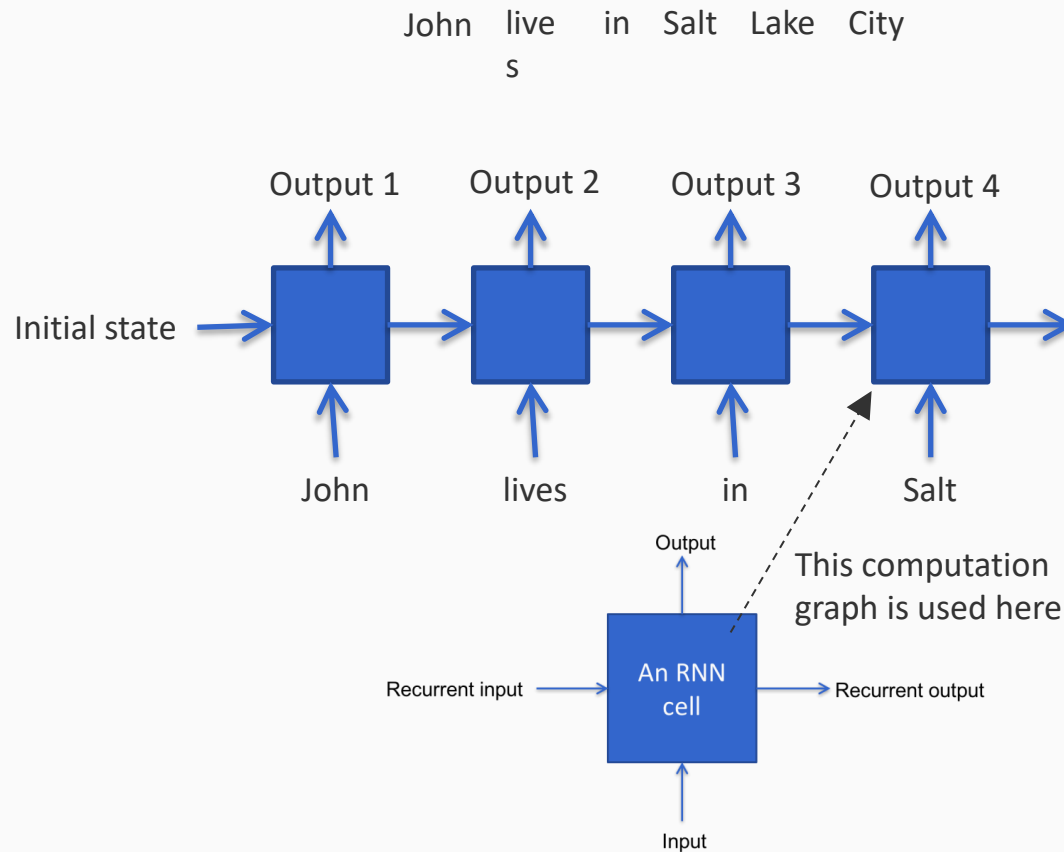
The RNN abstraction: A simple example



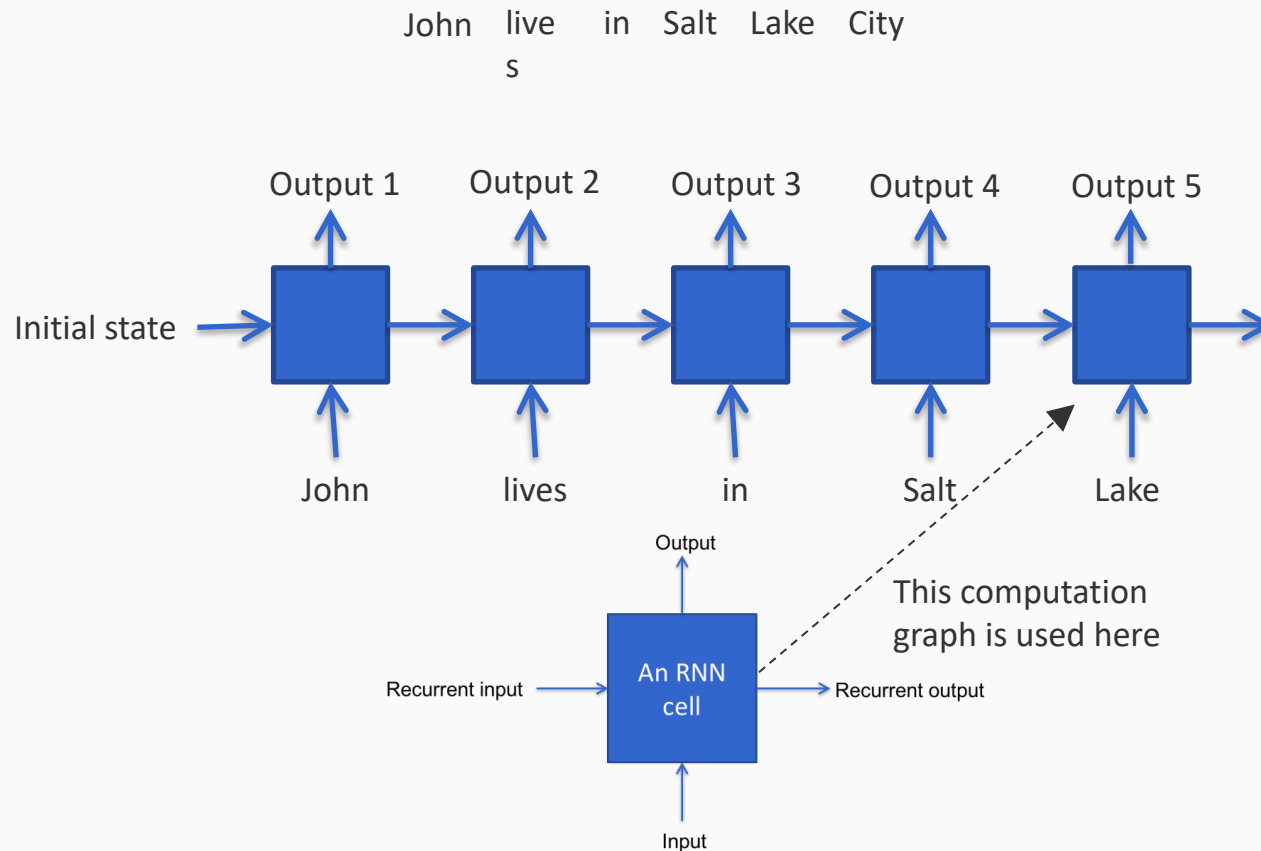
The RNN abstraction: A simple example



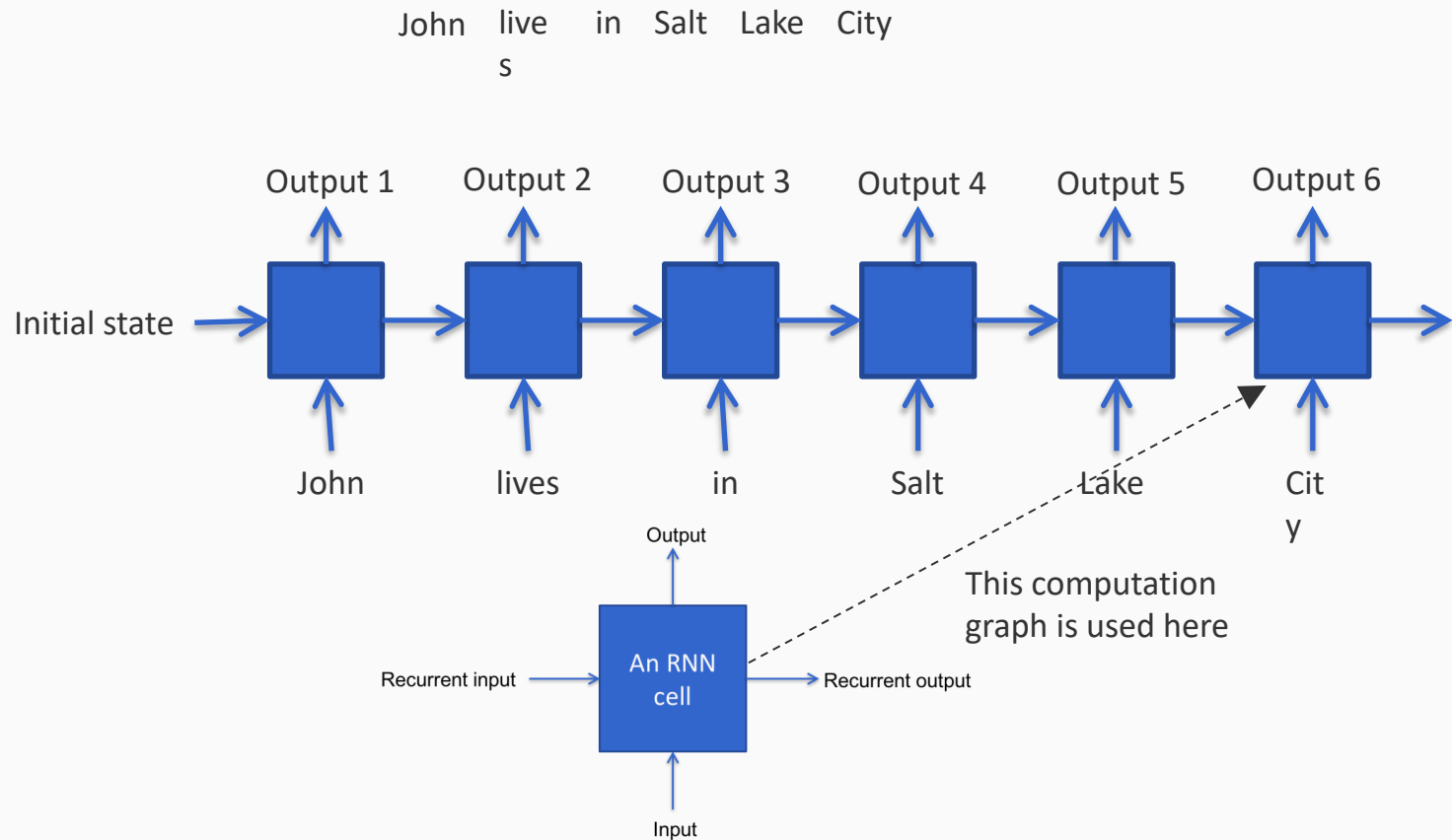
The RNN abstraction: A simple example



The RNN abstraction: A simple example



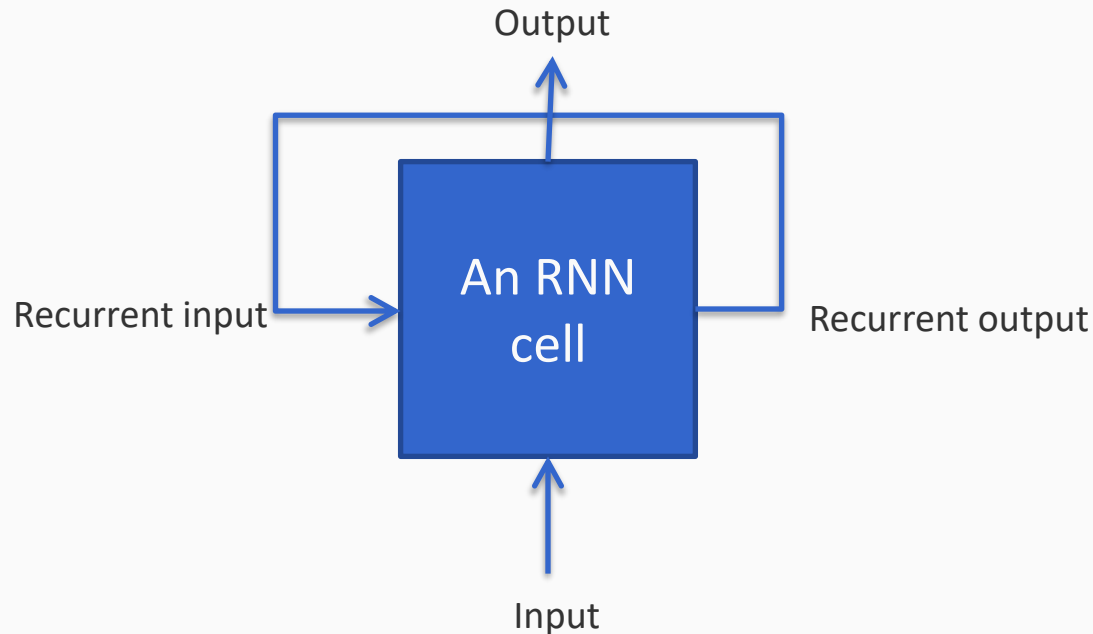
The RNN abstraction: A simple example



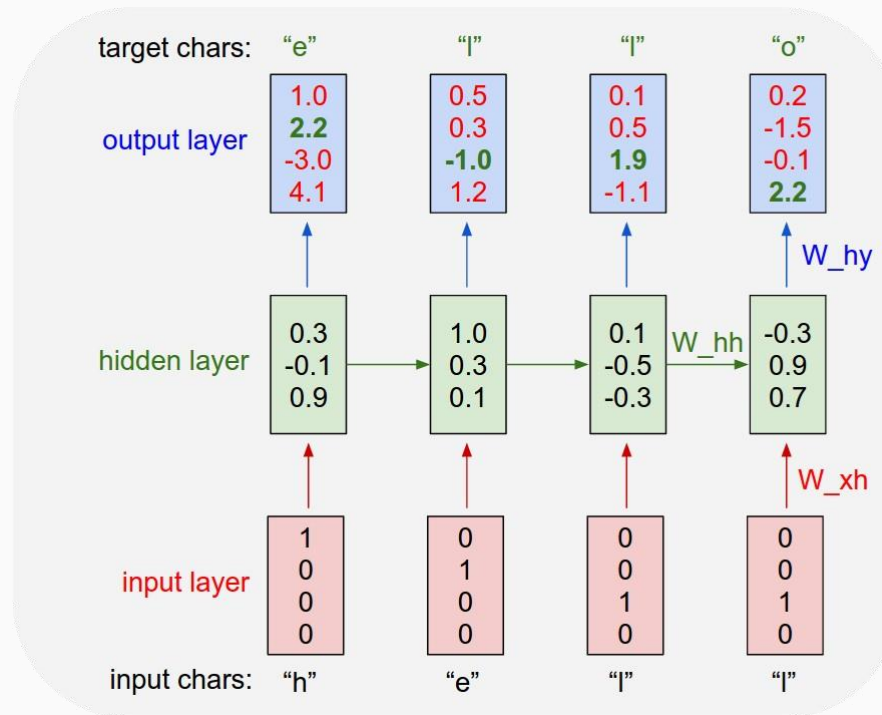
The RNN abstraction

Sometimes this is represented as a “neural network with a loop”.

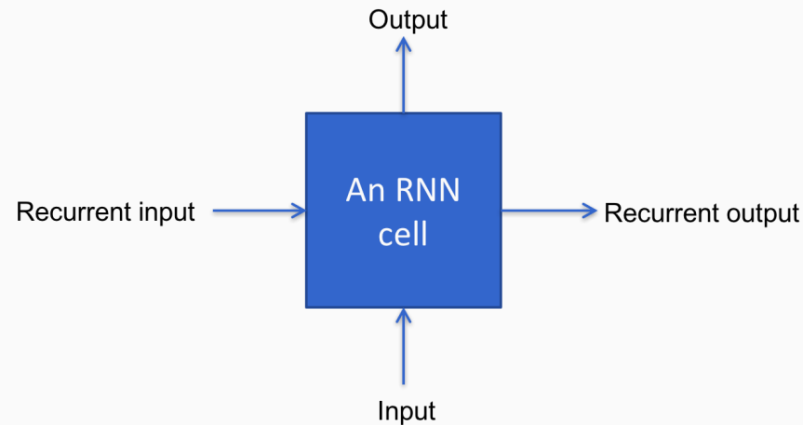
But really, when unrolled, there are no loops. Just a big feedforward network.



An example: Character level language model

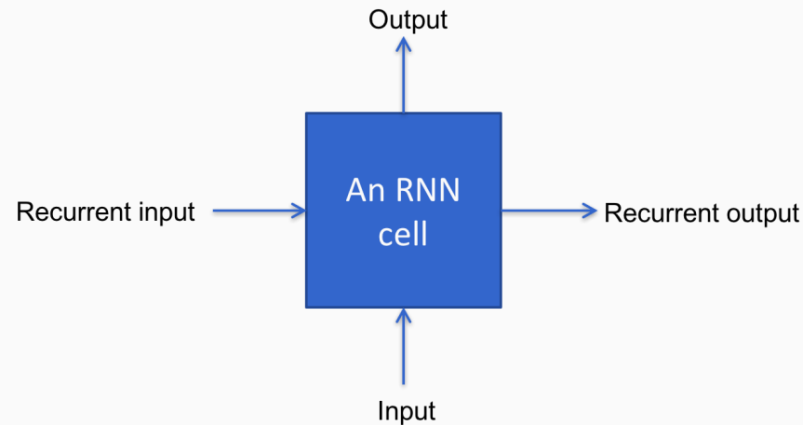


What can we do with such an abstraction?



1. The **encoder**: Convert a sequence into a feature vector for subsequent classification
2. A **generator**: Produce a sequence using an initial state
3. A **transducer**: Convert a sequence into another sequence
4. A **conditional generator** (or an **encoder-decoder**): Combine 1 and 2

What can we do with such an abstraction?

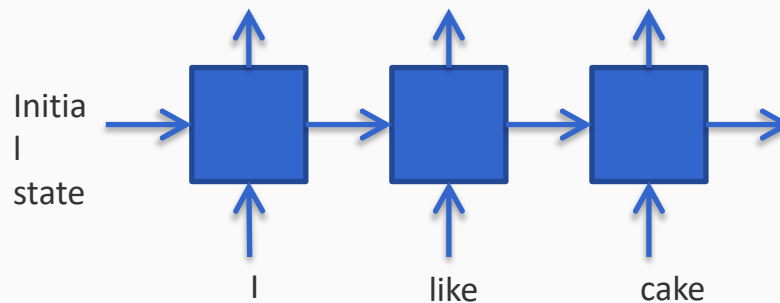


1. The **encoder**: Convert a sequence into a feature vector for subsequent classification
2. A **generator**: Produce a sequence using an initial state
3. A **transducer**: Convert a sequence into another sequence
4. A **conditional generator** (or an **encoder-decoder**): Combine 1 and 2

This set of operations also applies to other models for sequences. In particular, transformers.

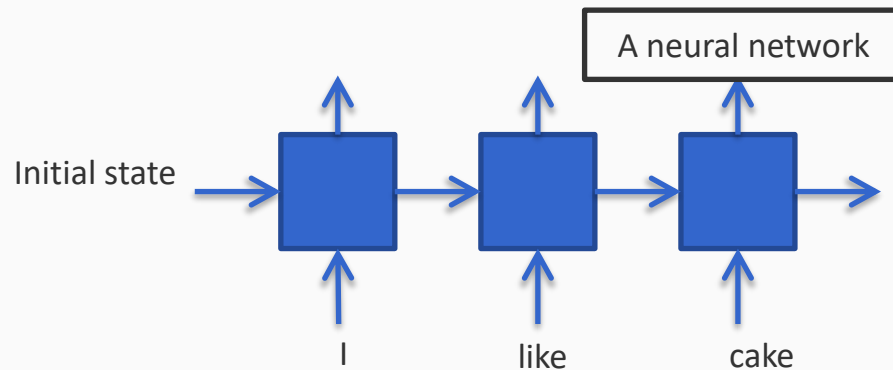
1. An Encoder

Convert a sequence into a feature vector for subsequent classification



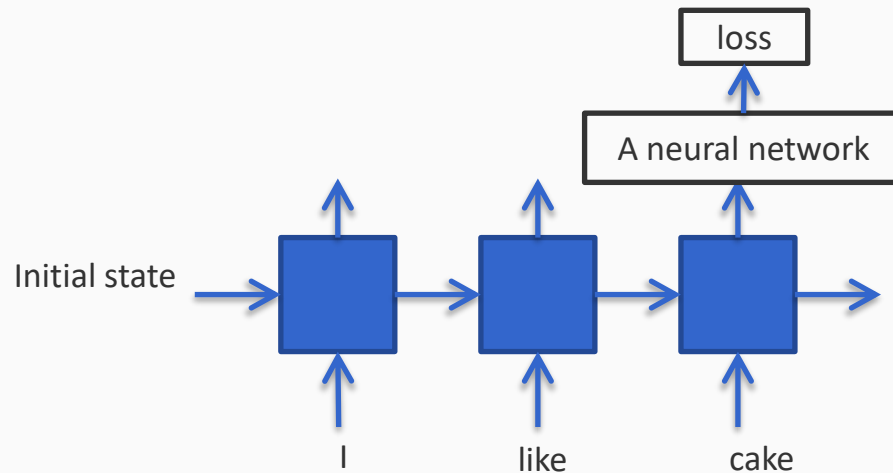
1. An Encoder

Convert a sequence into a feature vector for subsequent classification



1. An Encoder

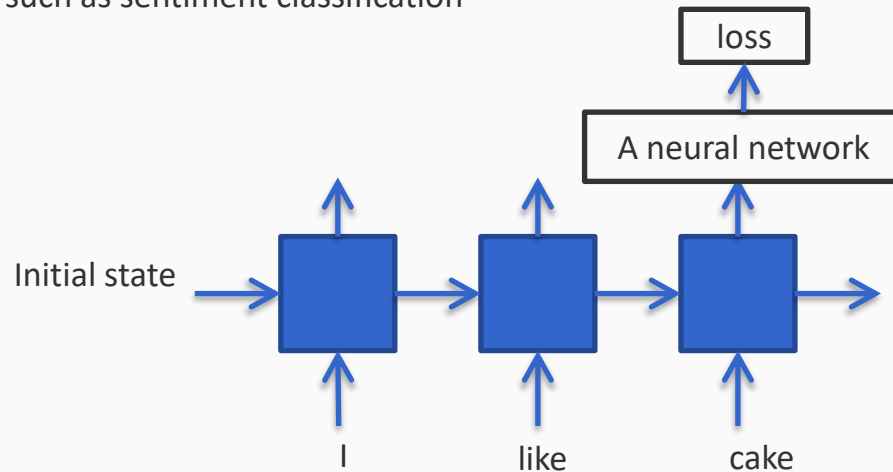
Convert a sequence into a feature vector for subsequent classification



1. An Encoder

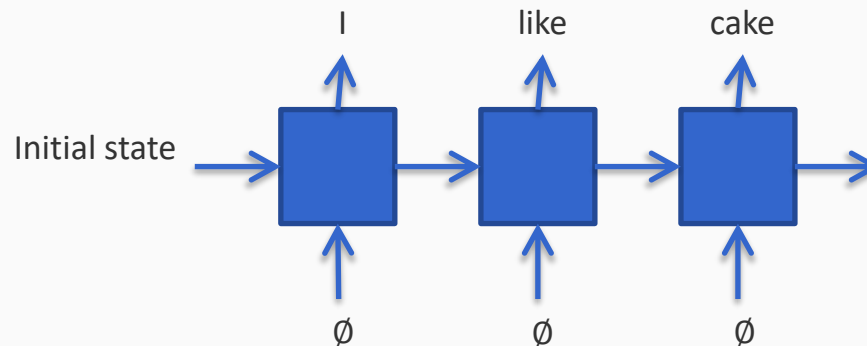
Convert a sequence into a feature vector for subsequent classification

Example: Encode a sentence or a phrase into a feature vector for a classification task such as sentiment classification



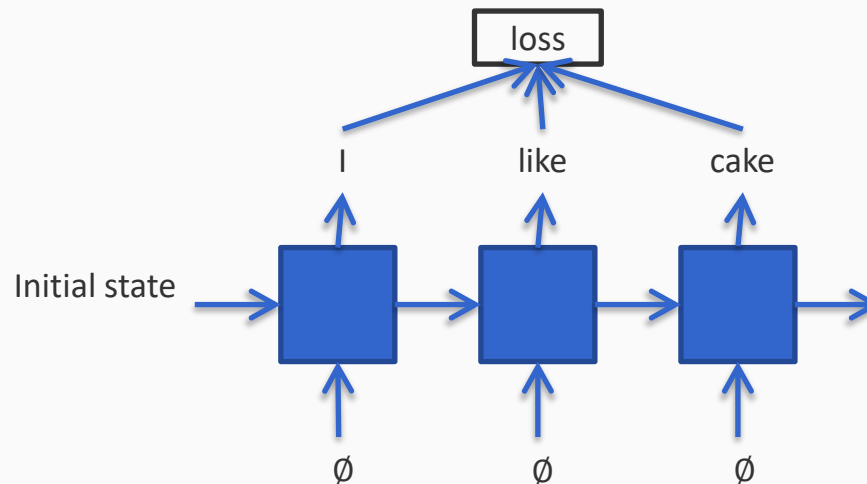
2. A Generator

Produce a sequence using an initial state



2. A Generator

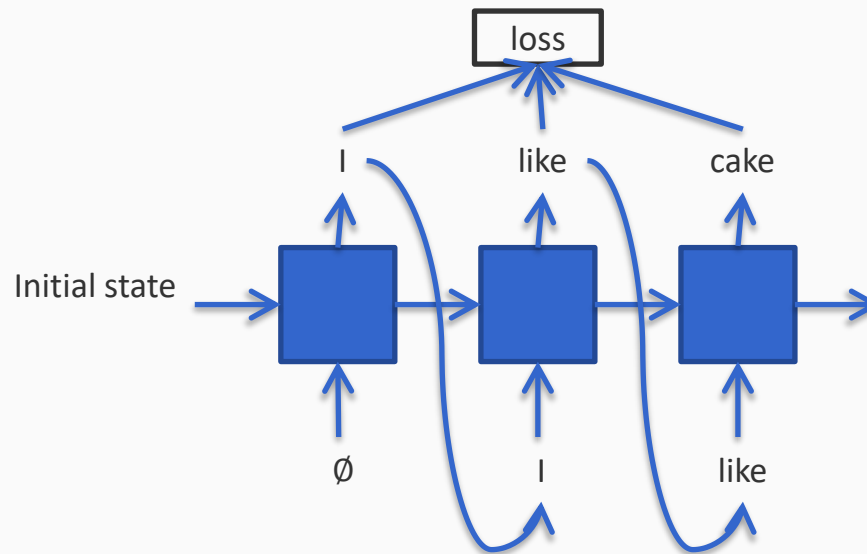
Produce a sequence using an initial state



2. A Generator

Produce a sequence using an initial state

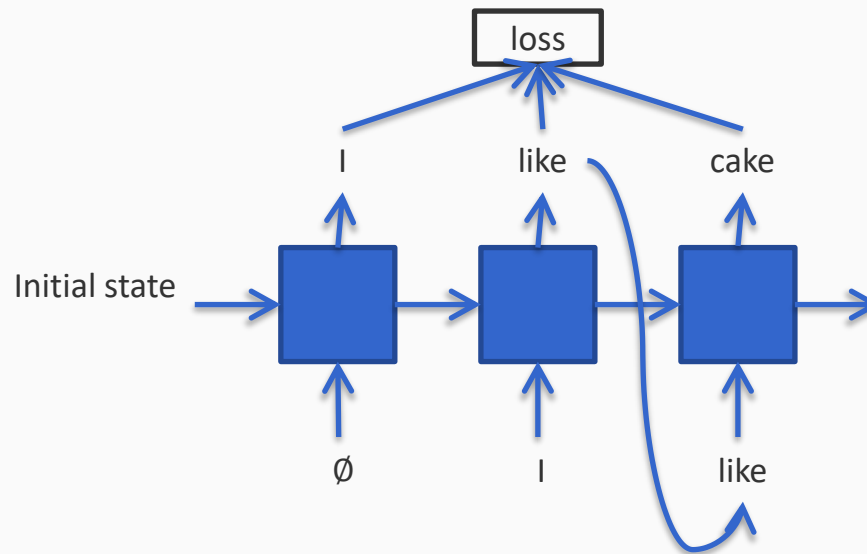
Maybe the previous output becomes the current input



2. A Generator

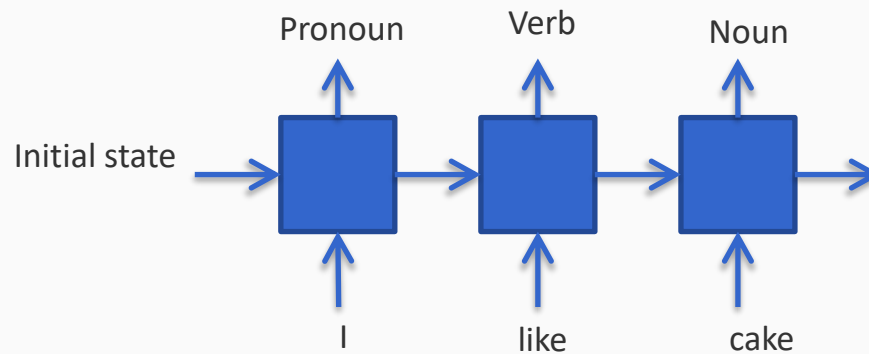
Produce a sequence using an initial state

Examples: Text generation tasks



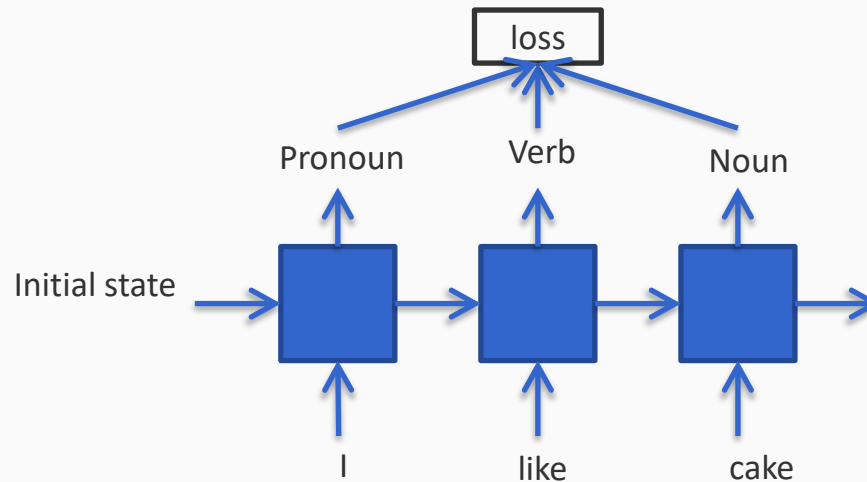
3. A Transducer

Convert a sequence into another sequence



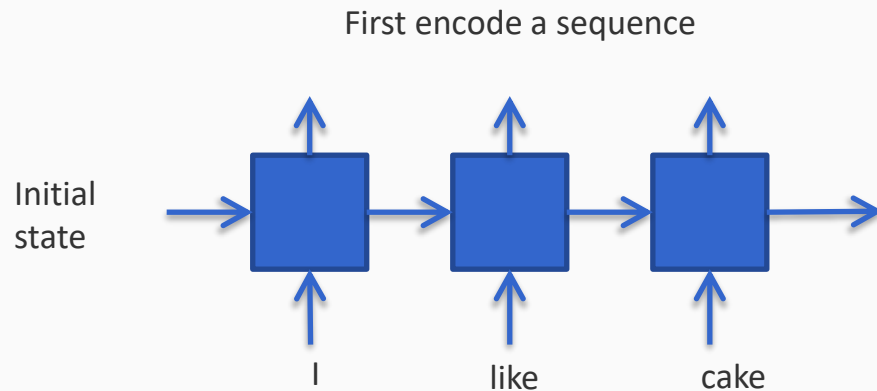
3. A Transducer

Convert a sequence into another sequence



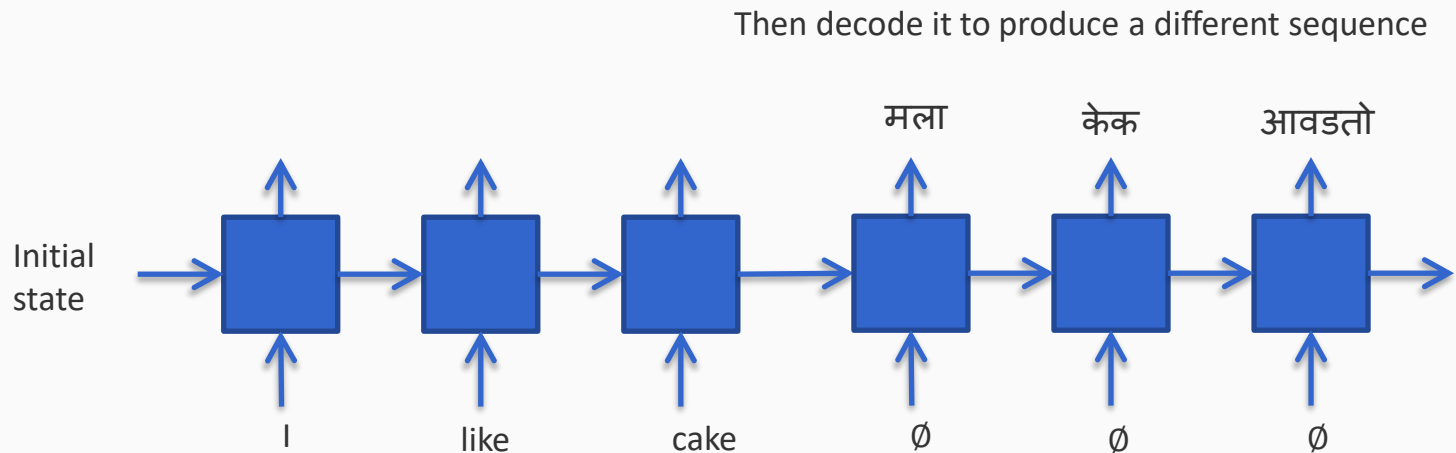
4. Conditioned generator

Or an encoder-decoder: First encode a sequence, then generate another one



4. Conditioned generator

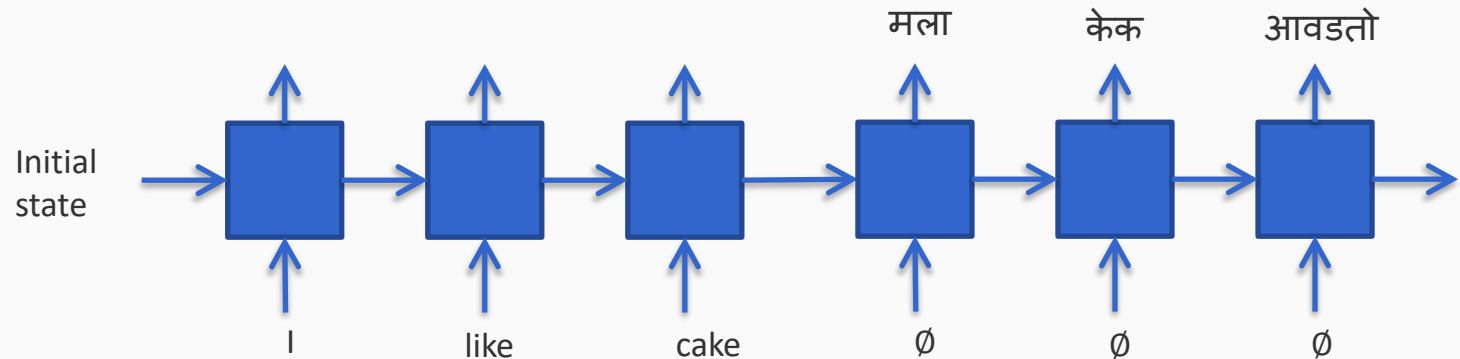
Or an encoder-decoder: First encode a sequence, then generate another one



4. Conditioned generator

Or an encoder-decoder: First encode a sequence, then generate another one

Example: This used to be a building block for neural machine translation



A simple RNN

At each step, an RNN:

- Computes the next hidden state: $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$
- (Optional) Computes the output: $\mathbf{y}_t = g(\mathbf{h}_t)$

Need to specify two functions:

1. How to generate the current state using the previous hidden state and the current input?
2. How to generate the current output using the current hidden state?

Computing the value of a state

1. How to generate the current state using the previous state and the current input?

$$\mathbf{h}_{t-1}$$

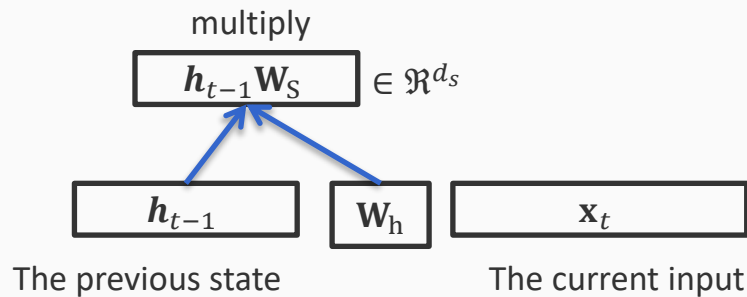
The previous state
A vector in \mathfrak{R}^{d_h}

$$\mathbf{x}_t$$

The current input
A vector in \mathfrak{R}^d

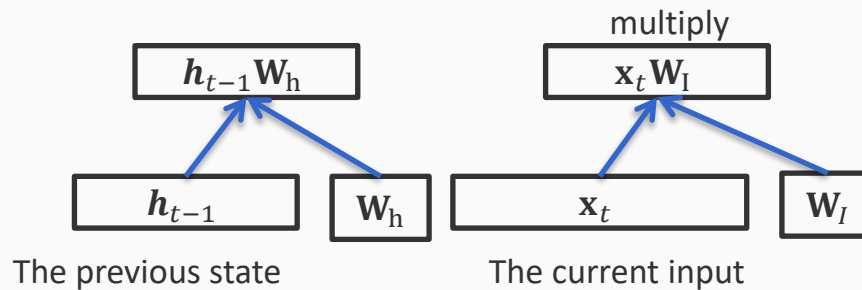
Computing the value of a state

1. How to generate the current state using the previous state and the current input?



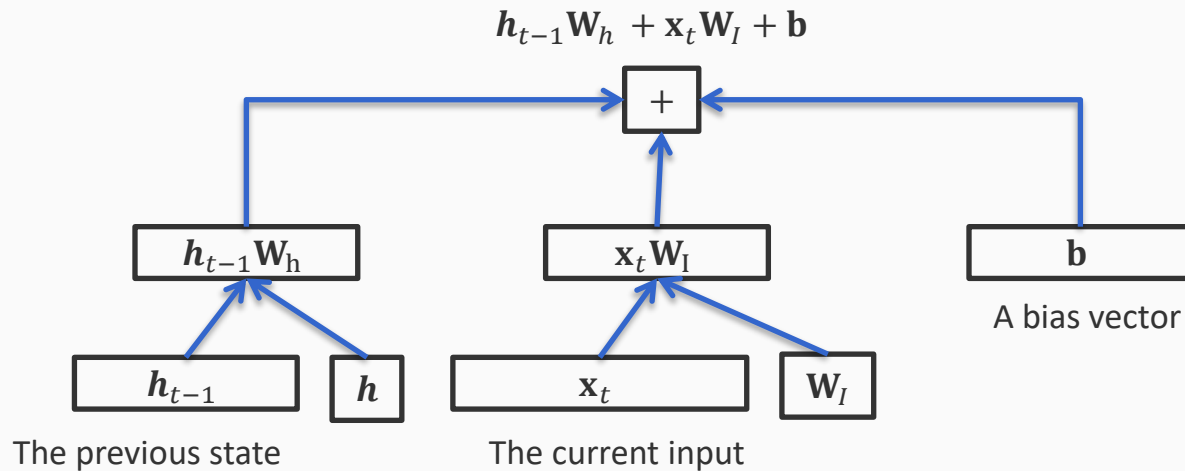
Computing the value of a state

1. How to generate the current state using the previous state and the current input?



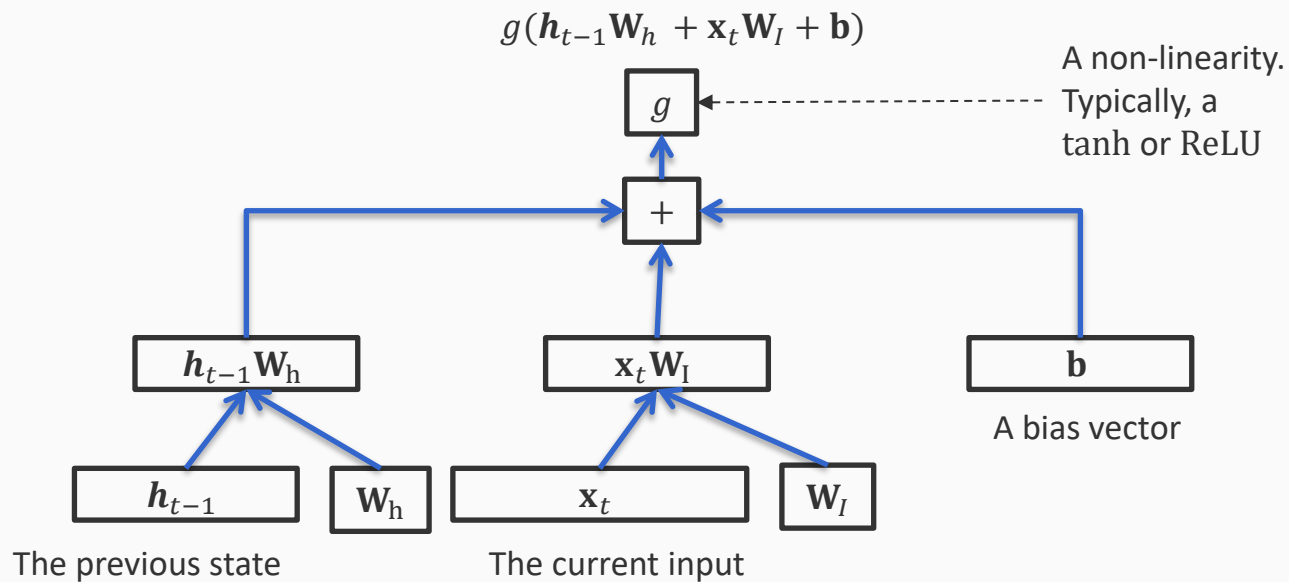
Computing the value of a state

1. How to generate the current state using the previous state and the current input?



Computing the value of a state

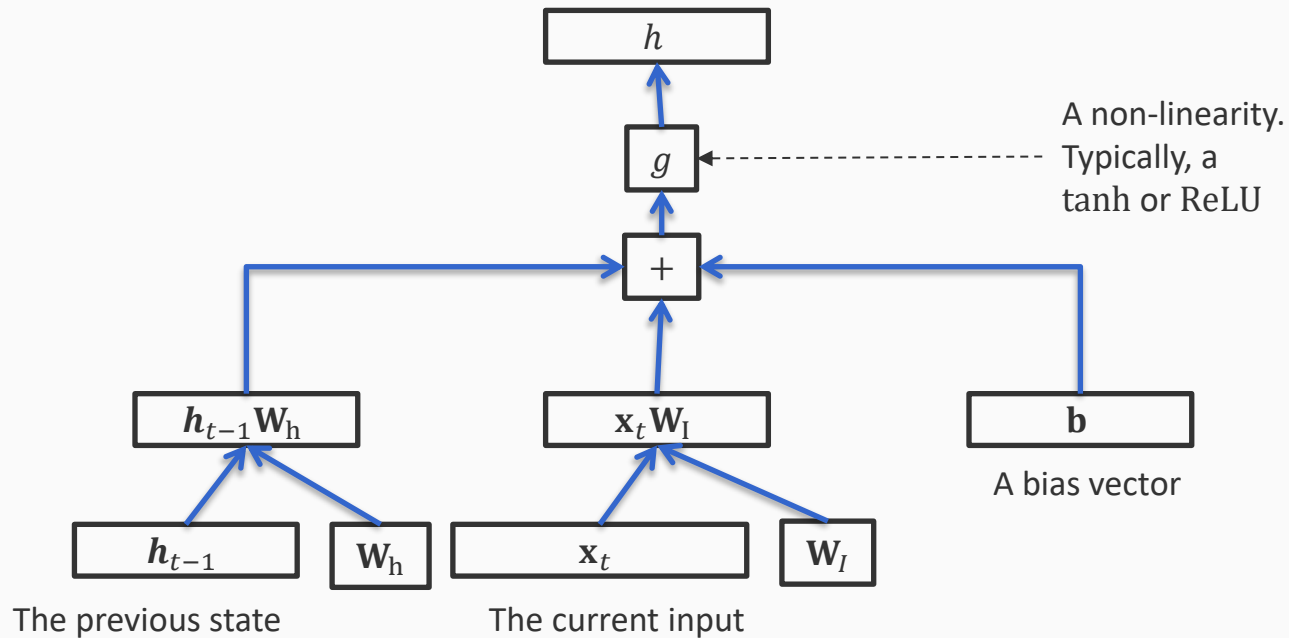
1. How to generate the current state using the previous state and the current input?



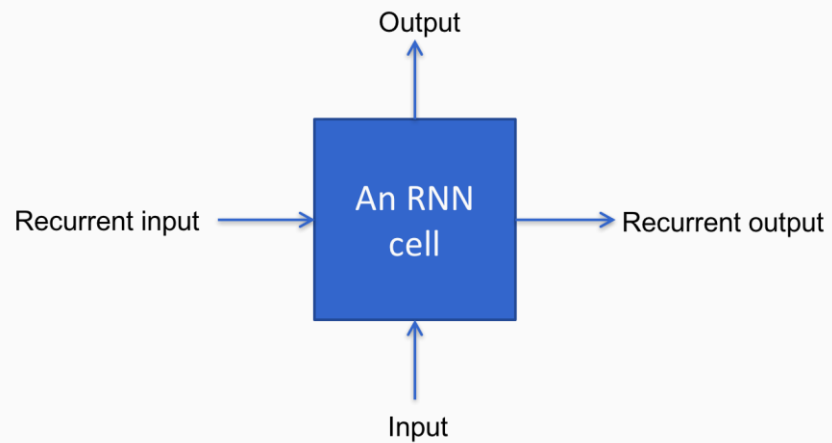
Computing the value of a state

1. How to generate the current state using the previous state and the current input?

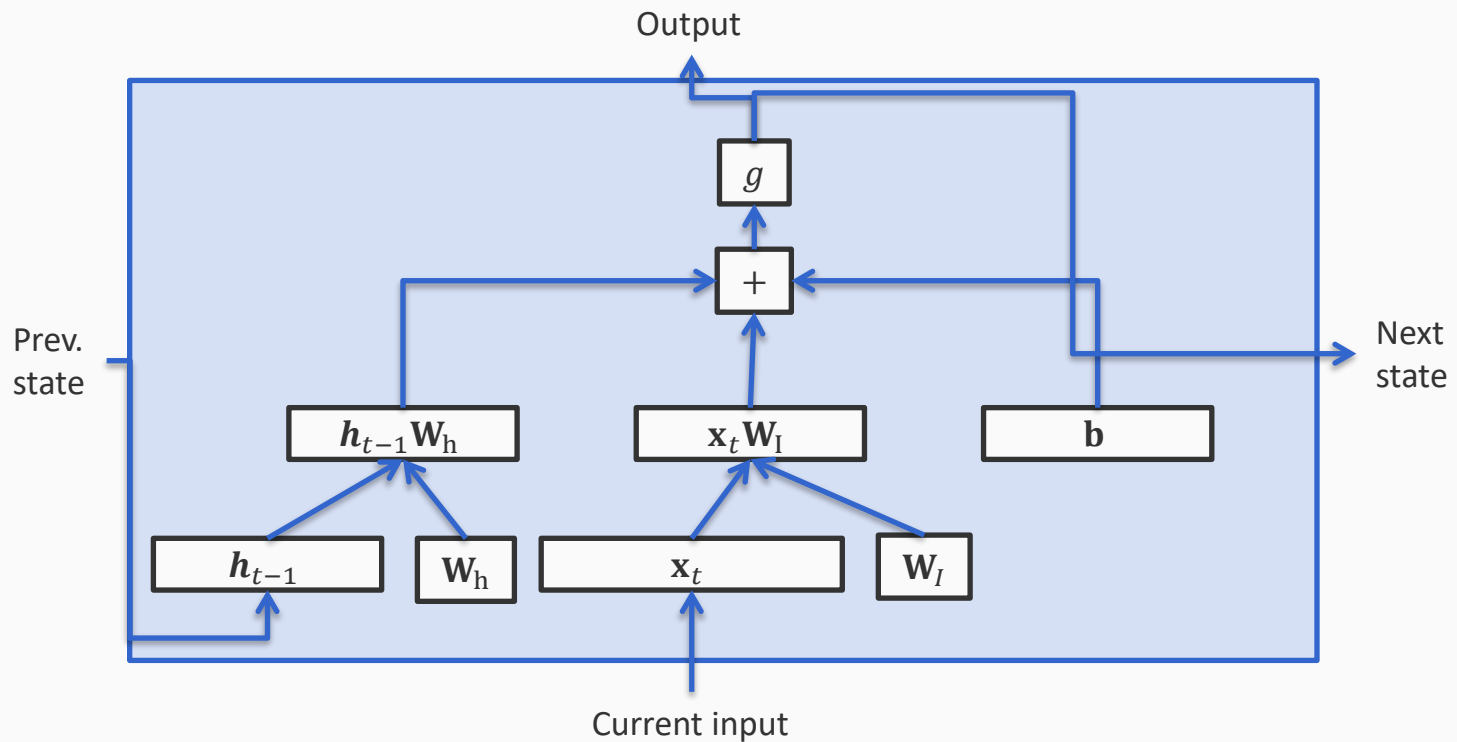
$$\text{Next state } \mathbf{h}_t = g(\mathbf{h}_{t-1}\mathbf{W}_h + \mathbf{x}_t\mathbf{W}_l + \mathbf{b})$$



The Elman RNN



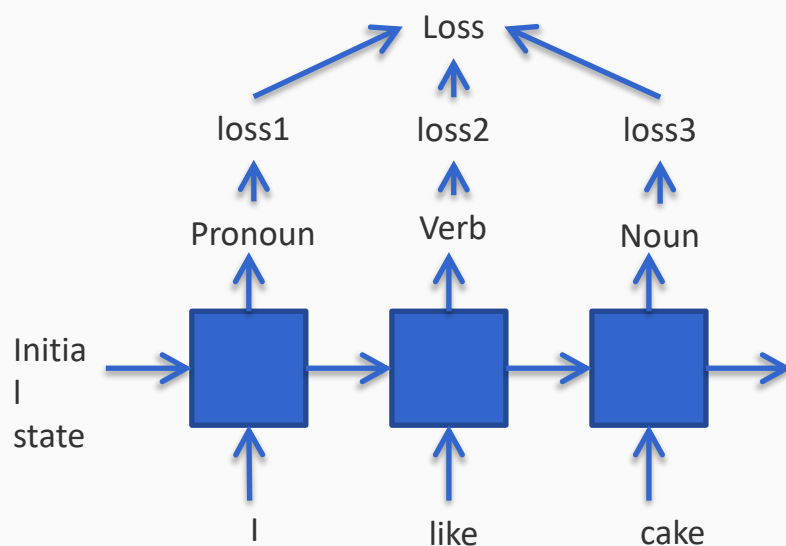
The Elman RNN



How do we train a recurrent network?

We need to specify a problem first. Let's take an example.

- Inputs are sequences (say, of words)
- The outputs are labels associated with each word
- Losses for each word are added up



The vanishing gradient problem

- As the length of the sequence grows, the impact of the far away inputs diminishes because the gradient vanishes [Hochreiter 1991, Bengio et al 1994]
- Applicable not only to recurrent networks, but to any case where we have a long chain of such activations (i.e. in a deep network): **Layers closer to the loss will get larger updates**

The vanishing gradient problem

Why is this a problem?

The vanishing gradient problem

Why is this a problem?

I have a banana and an apple. My friend ate the banana and I ate the _____?

The vanishing gradient problem

Why is this a problem?

I have a banana and an apple. My friend ate the banana. I was hungry and wanted a fruit. So I ate the _____?

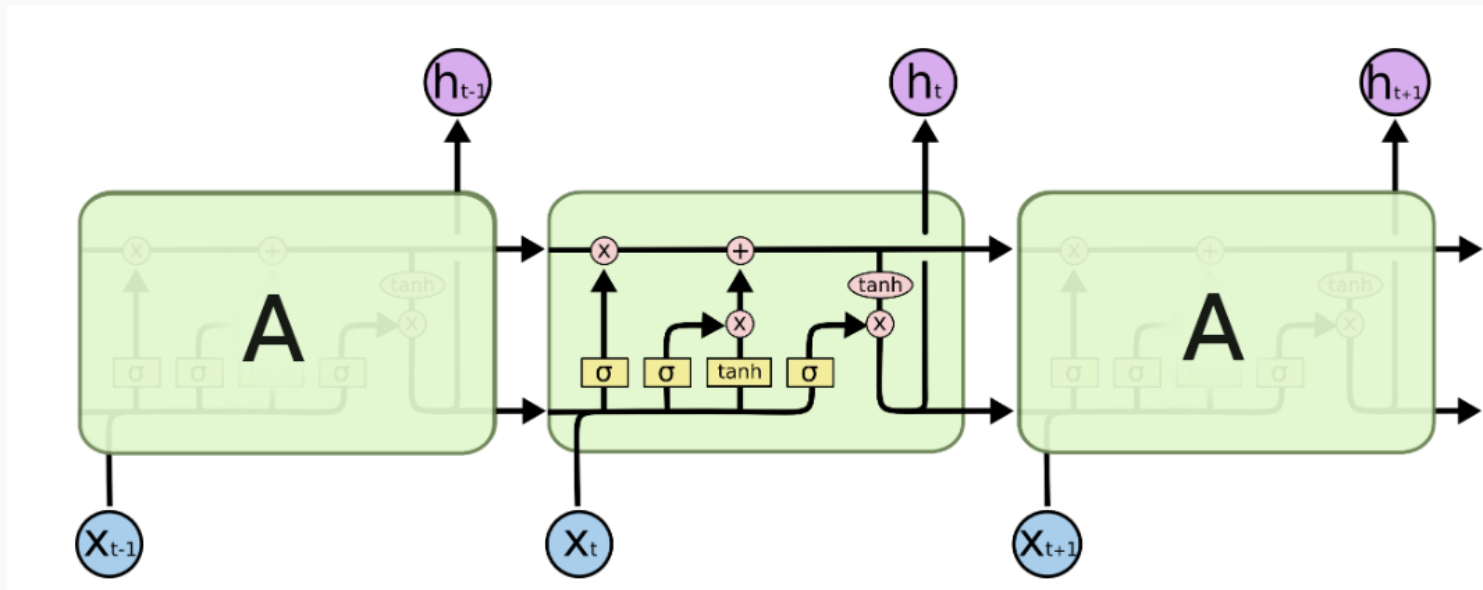
The vanishing gradient problem

Why is this a problem?

I have a banana and an apple. My friend ate the banana. I was hungry and wanted a fruit. I really wished I had a banana as well, but we were all out. So I ate the _____?

Great LSTM Blog

- <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



Predicting sequences? Consider an LSTM

RNNs and especially LSTMs are a fundamental unit of recurrent neural networks

- Useful building block for modeling sequences
- Several variants exist, but all have a similar flavor
 - Eg: The gated recurrent unit is a simpler variant

Convolutional Neural Networks



Convolutional Neural Networks

Designed to

1. Identify local predictors in a larger input
2. Pool them together to create a feature representation
3. And possibly repeat this in a hierarchical fashion

CNN terminology

Shows its computer visions and signal processing origins

CNN terminology

Shows its computer visions and signal processing origins

- *Filter*
 - A function that transforms in input matrix/vector into a scalar feature
 - A filter is a learned feature detector (also called a feature map)

CNN terminology

Shows its computer visions and signal processing origins

- *Filter*
 - A function that transforms in input matrix/vector into a scalar feature
 - A filter is a learned feature detector (also called a feature map)
- *Channel*
 - In computer vision, color images have red, blue and green channels
 - In general, a channel represents a “view of the input” that captures information about an input independent of other channels
 - For example, different kinds of word embeddings could be different channels
 - Channels could themselves be produced by previous convolutional layers

CNN terminology

Shows its computer visions and signal processing origins

- *Filter*
 - A function that transforms in input matrix/vector into a scalar feature
 - A filter is a learned feature detector (also called a feature map)
- *Channel*
 - In computer vision, color images have red, blue and green channels
 - In general, a channel represents a “view of the input” that captures information about an input independent of other channels
 - Channels could themselves be produced by previous convolutional layers
- *Receptive field*
 - The region of the input that a filter currently focuses on

What is a convolution?

Let's see this using an example for vectors.

We will generalize this to matrices and beyond, but the general idea remains the same.

What is a convolution?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

What is a convolution?

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size n

1	2	1
---	---	---

Here, the filter size is 3

What is a convolution?

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---

The output is also a vector

What is a convolution?

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---

The output is also a vector

The filter moves across the vector.

At each position, the output is the dot product of the filter with a slice of the vector of that size.

What is a convolution?

Padding at the beginning



A vector \mathbf{x}

0	2	3	1	3	2	1
---	---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---

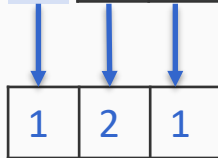
What is a convolution?

Padding at the beginning

A vector \mathbf{x}



Filter \mathbf{f} of size
 n



The output is also a vector



What is a convolution?

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---

The output is also a vector

7	9				
---	---	--	--	--	--

What is a convolution?

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---

The output is also a vector

7	9	8			
---	---	---	--	--	--

What is a convolution?

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---

The output is also a vector

7	9	8	9		
---	---	---	---	--	--

What is a convolution?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---

The output is also a vector

7	9	8	9	8	
---	---	---	---	---	--

What is a convolution?

An example using vectors

Padding at the end

A vector \mathbf{x}

2	3	1	3	2	1	0
---	---	---	---	---	---	---



Filter \mathbf{f} of size
 n

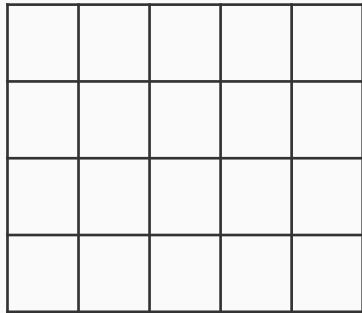
1	2	1
---	---	---

The output is also a vector

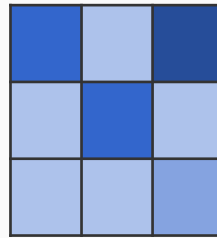
7	9	8	9	8	4
---	---	---	---	---	---

What is a convolution?

The same idea applies to matrices as well



An input matrix



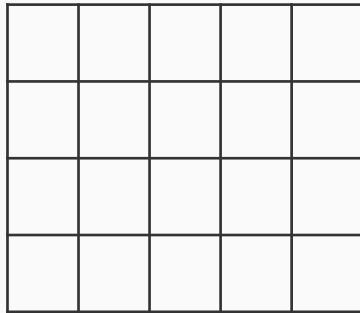
A filter

The filter moves across the matrix.

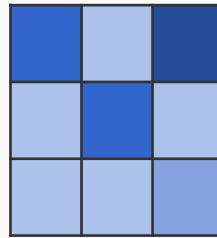
At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

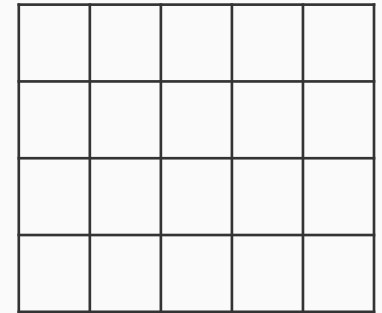
The same idea applies to matrices as well



An input matrix



A filter



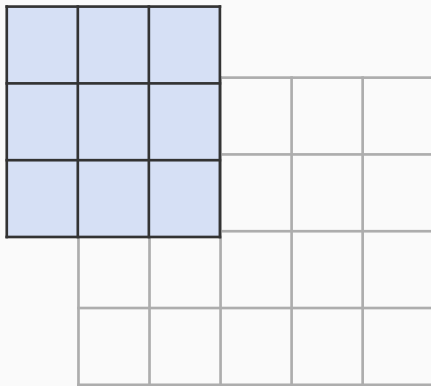
The result
of
convolution

The filter moves across the matrix.

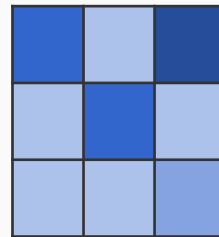
At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

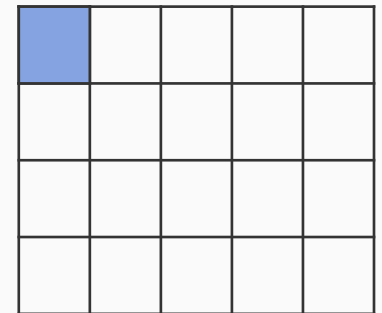
The same idea applies to matrices as well



An input matrix



A filter



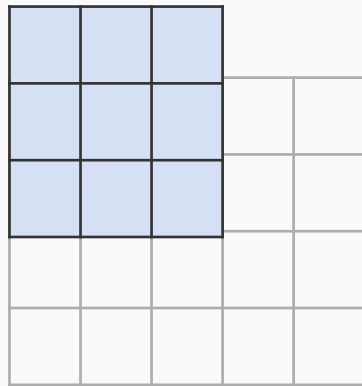
The result
of
convolution

The filter moves across the matrix.

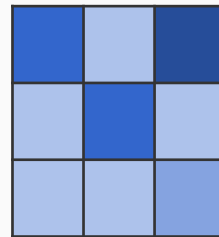
At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

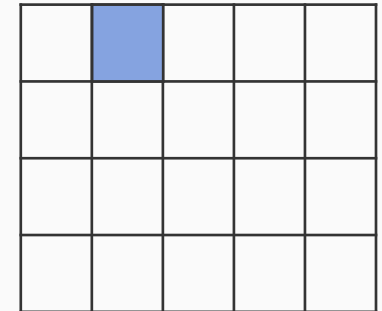
The same idea applies to matrices as well



An input matrix



A filter



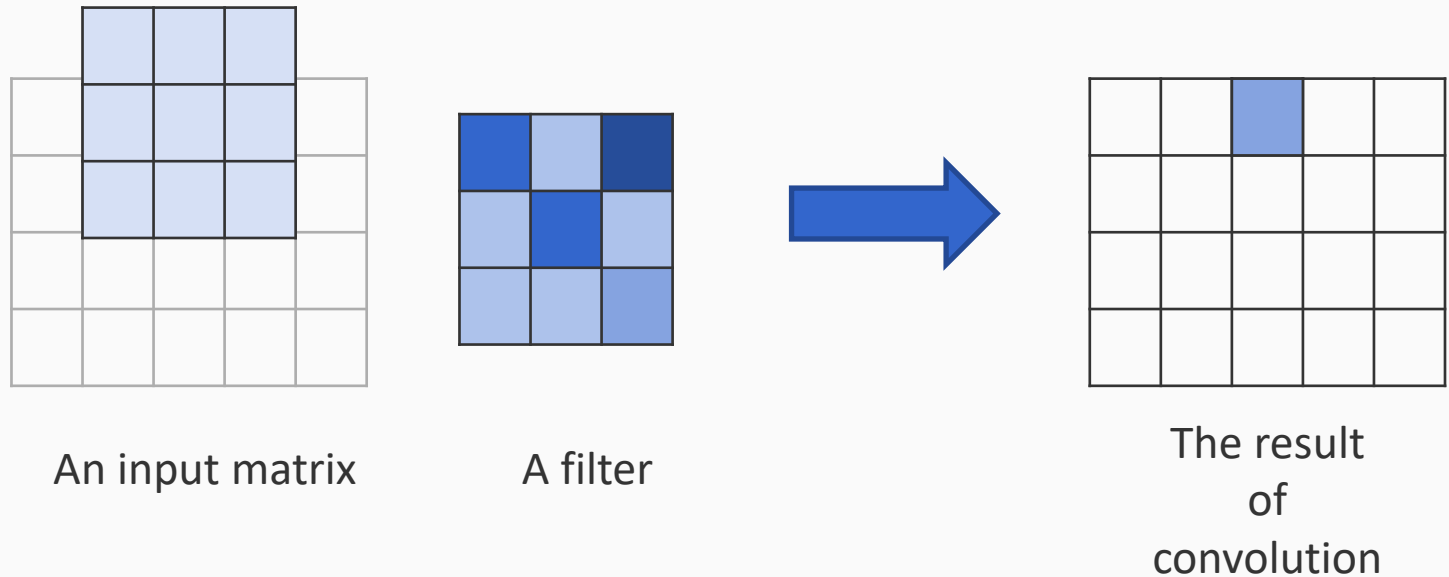
The result
of
convolution

The filter moves across the matrix.

At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

The same idea applies to matrices as well

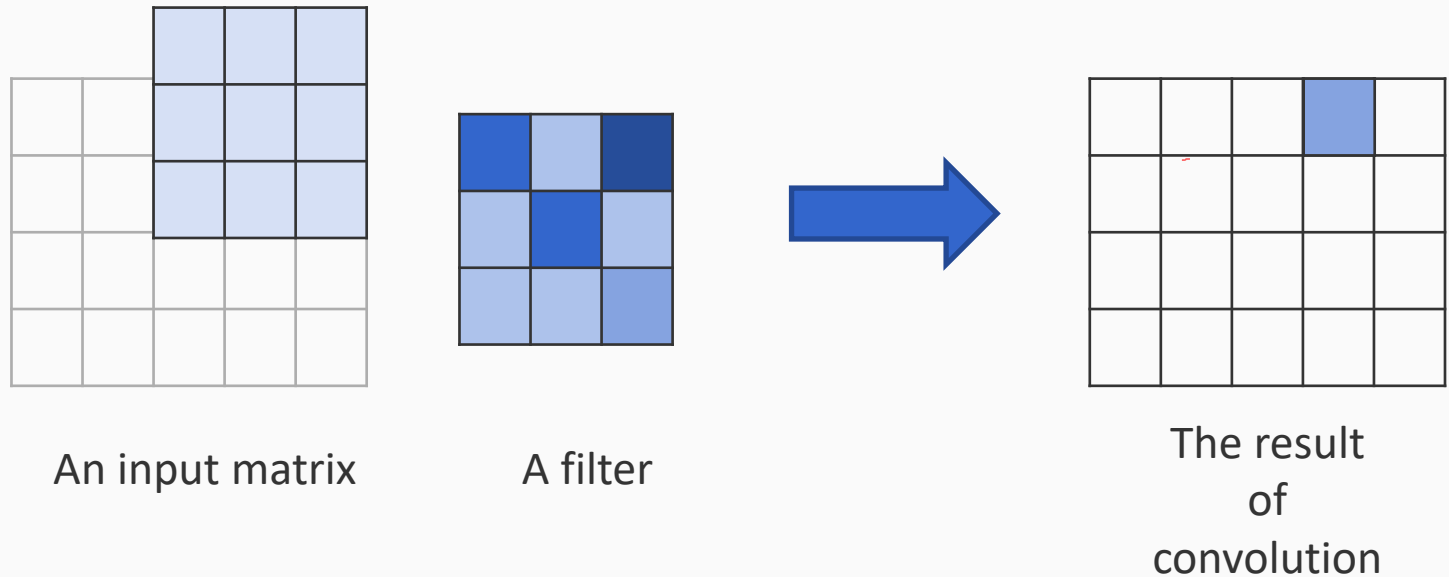


The filter moves across the matrix.

At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

The same idea applies to matrices as well

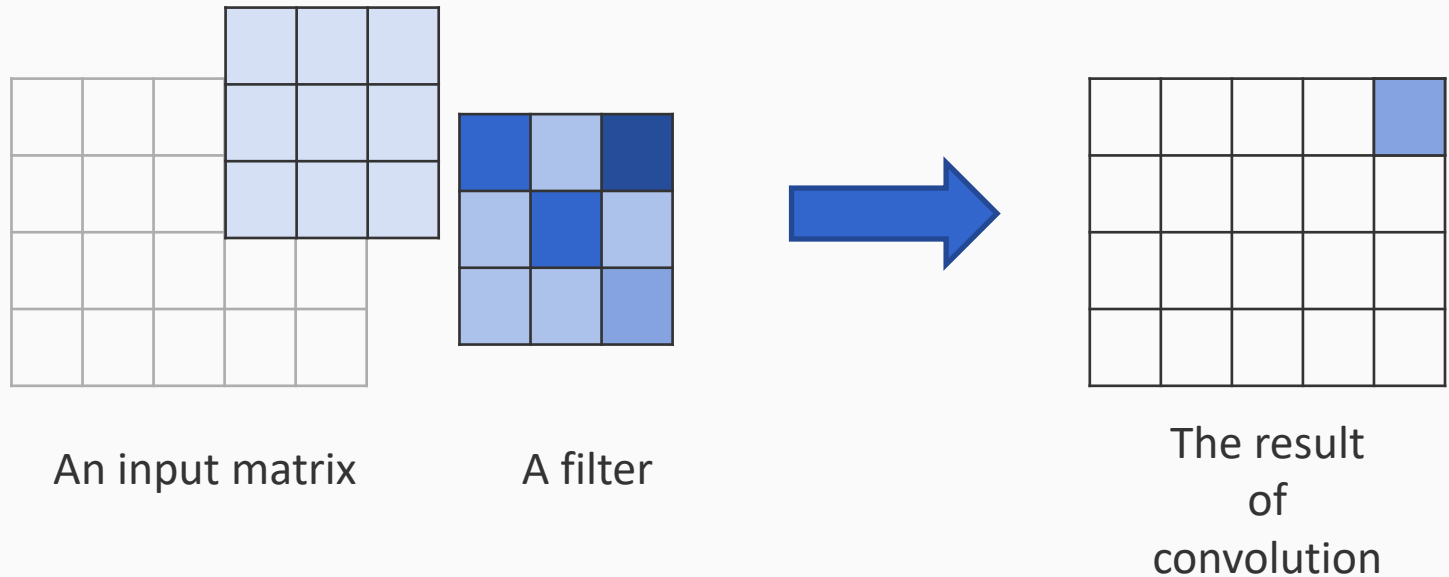


The filter moves across the matrix.

At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

The same idea applies to matrices as well

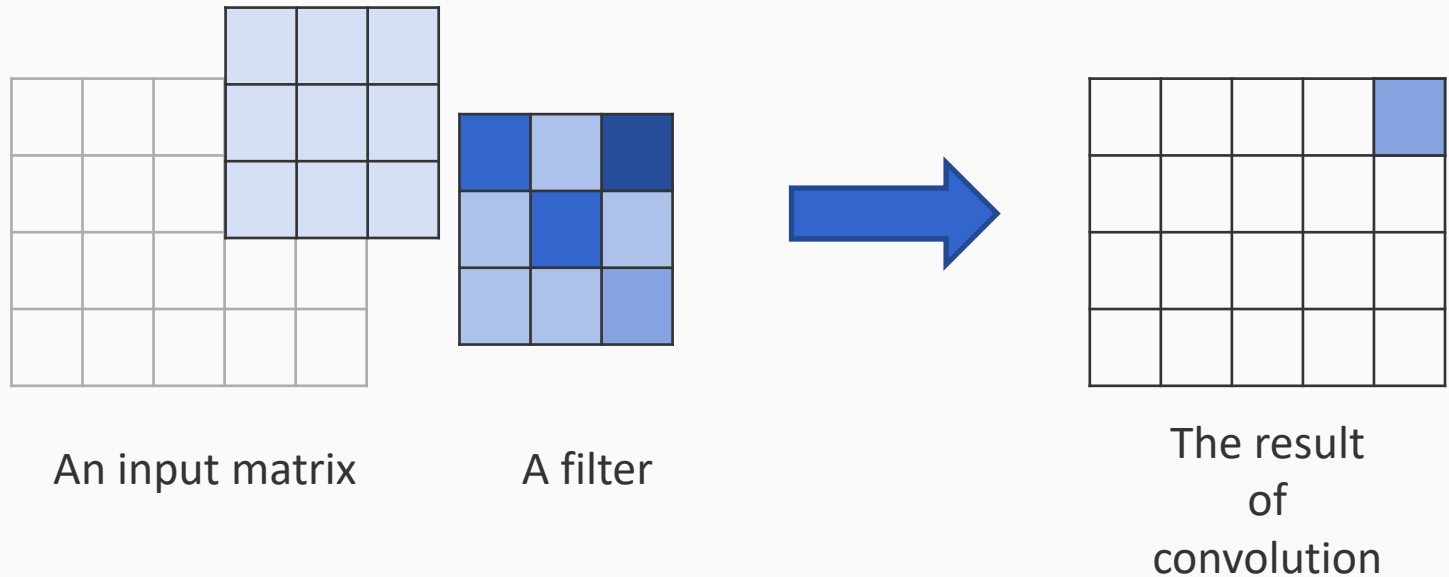


The filter moves across the matrix.

At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

The same idea applies to matrices as well



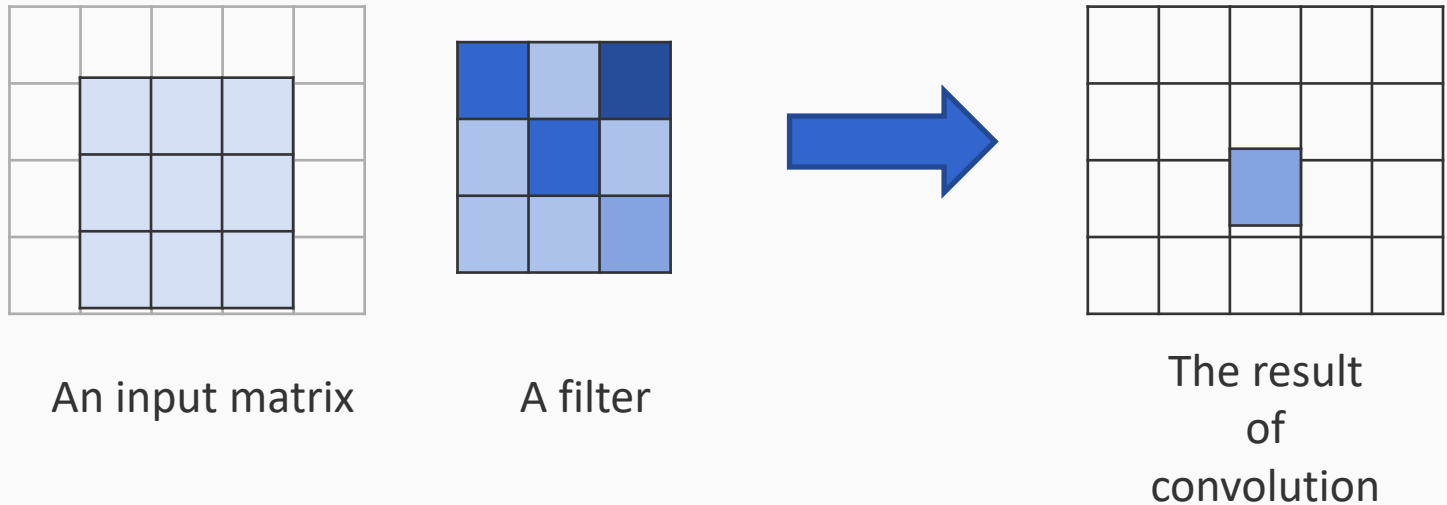
And so on...

The filter moves across the matrix.

At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

The same idea applies to matrices as well

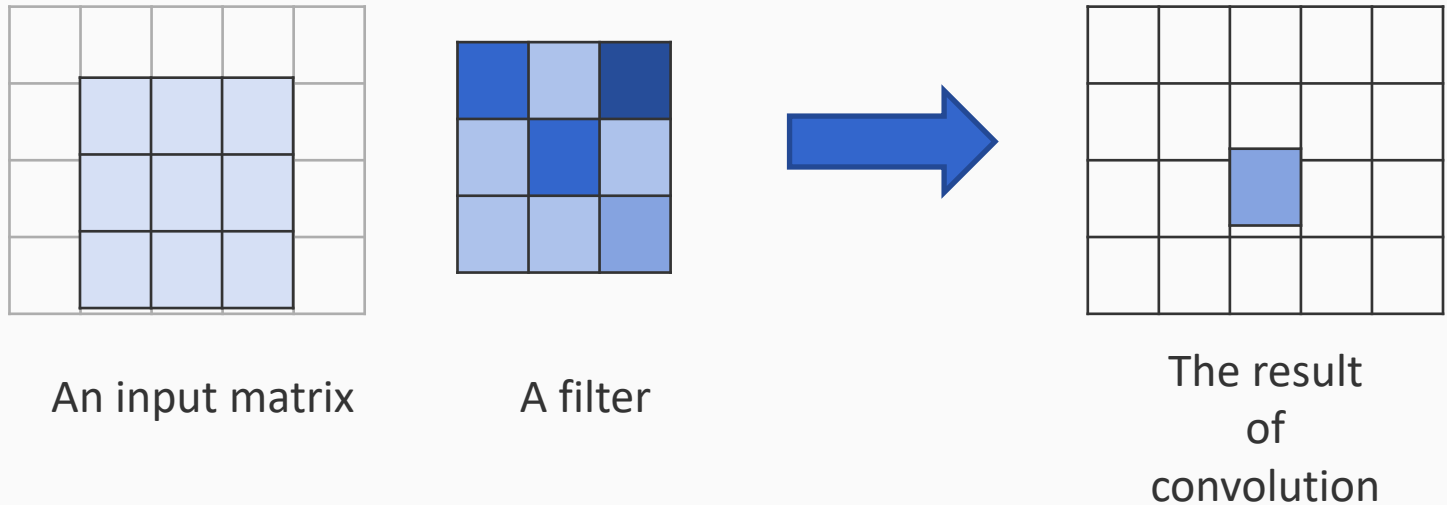


The filter moves across the matrix.

At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

The same idea applies to matrices as well



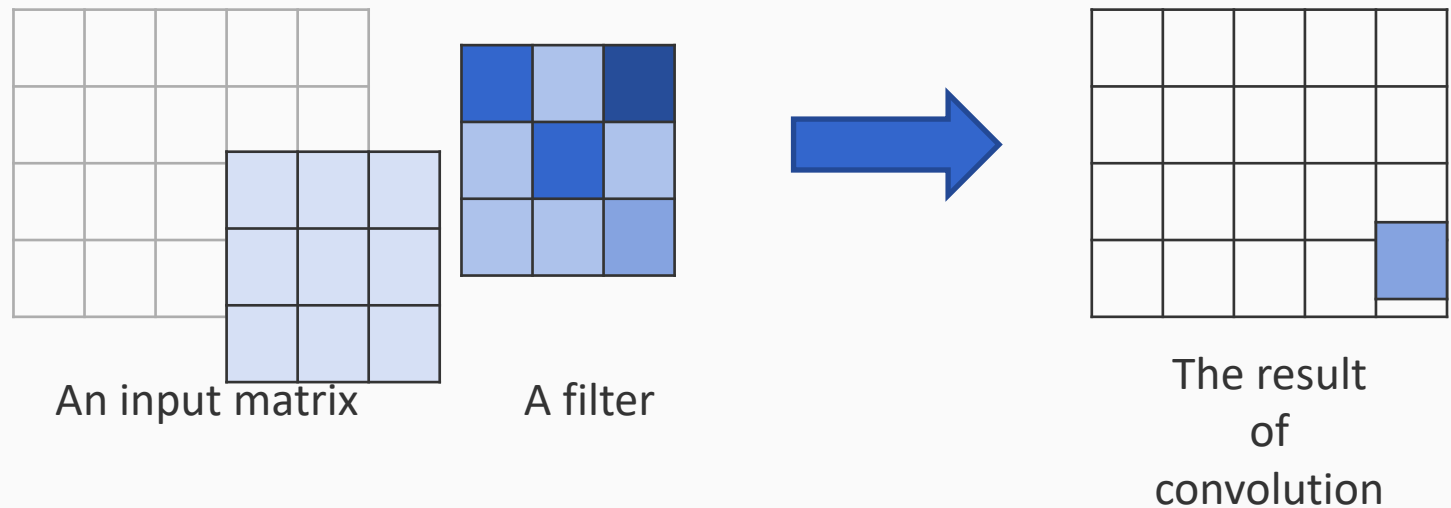
And so on...

The filter moves across the matrix.

At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

The same idea applies to matrices as well



The filter moves across the matrix.

At each position, the output is the dot product of the filter with a slice of the matrix of that size.

What is a convolution?

What would this kind of filter do?

An input matrix

0	1	0
0	1	0
0	1	0

A filter



The result
of
convolution

What is a convolution?

What would this kind of filter do?

An input matrix

$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$

A filter



The result
of
convolution

Pooling: An aggregation operation

- A convolution produces a vector/matrix that captures properties of each window
- Pooling combines this information to produce a down-sampled version vector/matrix
 - Typically using the maximum or the average value within a window
 - Tries to find most important or salient features within a window

Pooling: An aggregation operation

- A convolution produces a vector/matrix that captures properties of each window
- Pooling combines this information to produce a down-sampled version vector/matrix
 - Typically using the maximum or the average value within a window
- Intuition
 - A filter is a feature detector that discovers how well each window matches a feature of interest
 - The most important features should be recognized regardless of their location
 - Answer: Pool the information from different windows together

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---



The pooling operation can
be applied using a window
as well

The output is also a vector

7	9	8	9	8	4
---	---	---	---	---	---

Example 1: Max pooling with window size 3

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size n

1	2	1
---	---	---



7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 1: Max pooling with window size 3

9			
---	--	--	--

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size n

1	2	1
---	---	---



7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 1: Max pooling with window size 3

9	9		
---	---	--	--

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size n

1	2	1
---	---	---



7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 1: Max pooling with window size 3

9	9	9	
---	---	---	--

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size n

1	2	1
---	---	---



The output is also a vector

7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 1: Max pooling with window size 3

9	9	9	9
---	---	---	---

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---



7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 2: Average pooling with window size 3

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size n

1	2	1
---	---	---



7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 2: Average pooling with window size 3

8			
---	--	--	--

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size n

1	2	1
---	---	---



7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 2: Average pooling with window size 3

8	8.6		
---	-----	--	--

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size n

1	2	1
---	---	---



7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 2: Average pooling with window size 3

8	8.6	8.3	
---	-----	-----	--

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size n

1	2	1
---	---	---



7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 2: Average pooling with window size 3

8	8.6	8.3	7
---	-----	-----	---

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---



7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

Example 3: Max pooling with window size = length of the vector

9

What is pooling?

An example using vectors

A vector \mathbf{x}

2	3	1	3	2	1
---	---	---	---	---	---

Filter \mathbf{f} of size
 n

1	2	1
---	---	---



The output is also a vector

7	9	8	9	8	4
---	---	---	---	---	---

The pooling operation can be applied using a window as well

ReLU
↓
max/ave

Important note

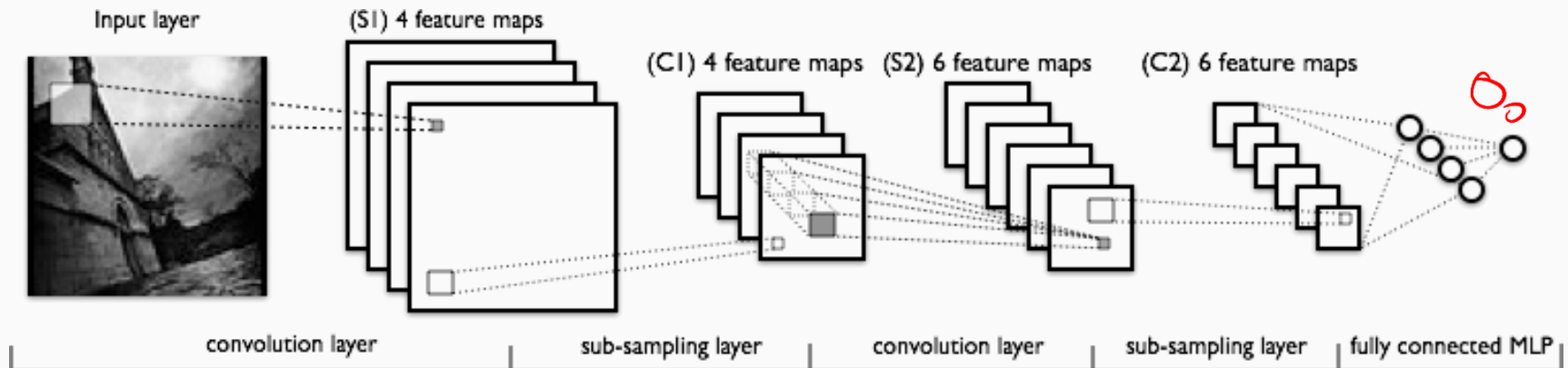
There are no learned parameters for the pooling operation. It is a deterministic operation.

Convolution + Pooling = one layer

- Input: a matrix. Convolution will operate over windows of this matrix.
- The window size defines the *receptive field*
 - We will refer to the window as x_i
- A filter is defined by some parameters (that will be learned)
 - In general, a matrix u of the same shape as a the window and a bias b
- Convolution: Iterate over all windows and apply the filter
 - Typically has a non-linearity (e.g. ReLU)
$$p_i = g(u \cdot x_i + b)$$
- Pooling: Aggregate the p_i 's into a down-sampled version, sometimes a single number
- Typically, there are many filters, each of which are pooled independently

Example: LeNet

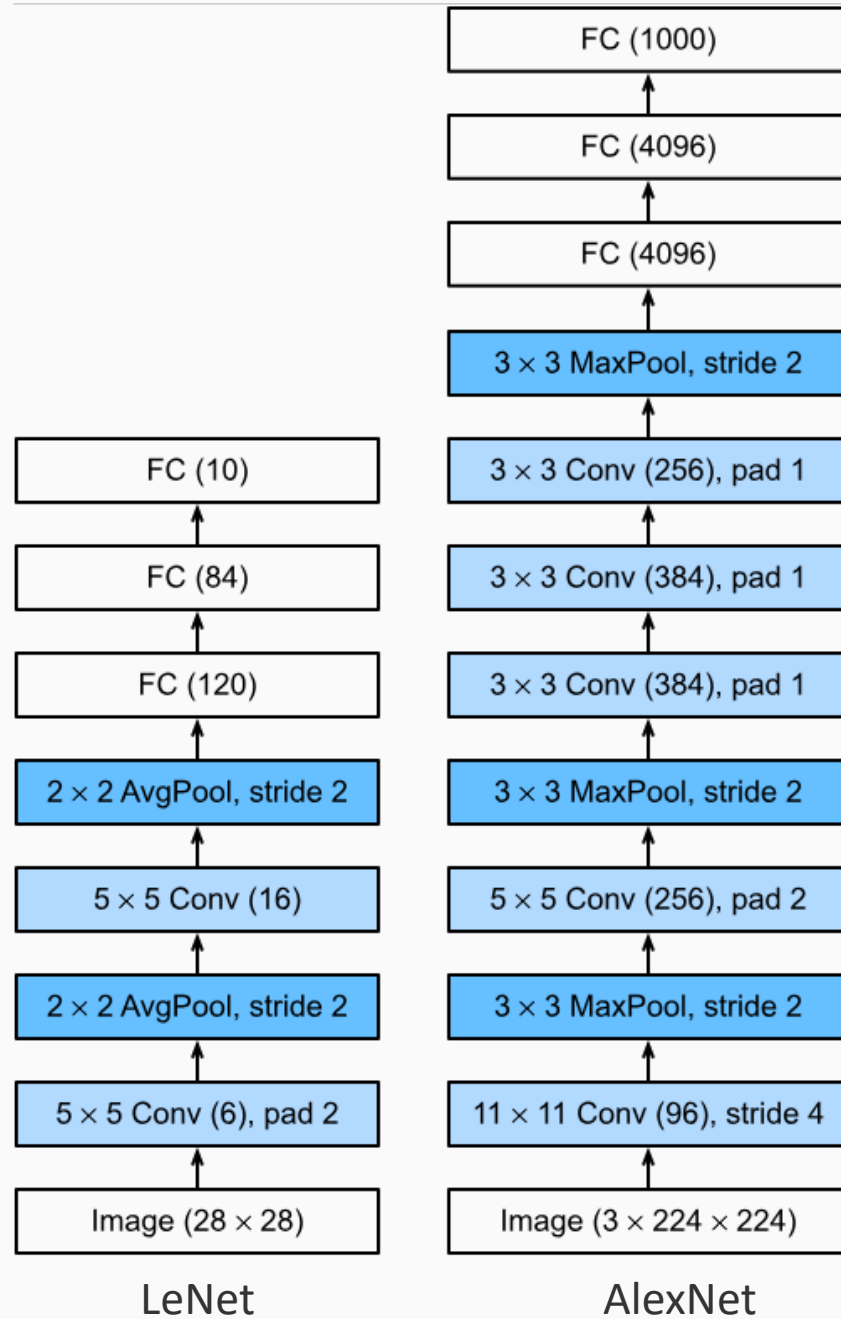
An example network uses these building block



LeNet-5 was proposed by Yann LeCun for handwriting recognition
Had several levels of convolution-pooling

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

AlexNet



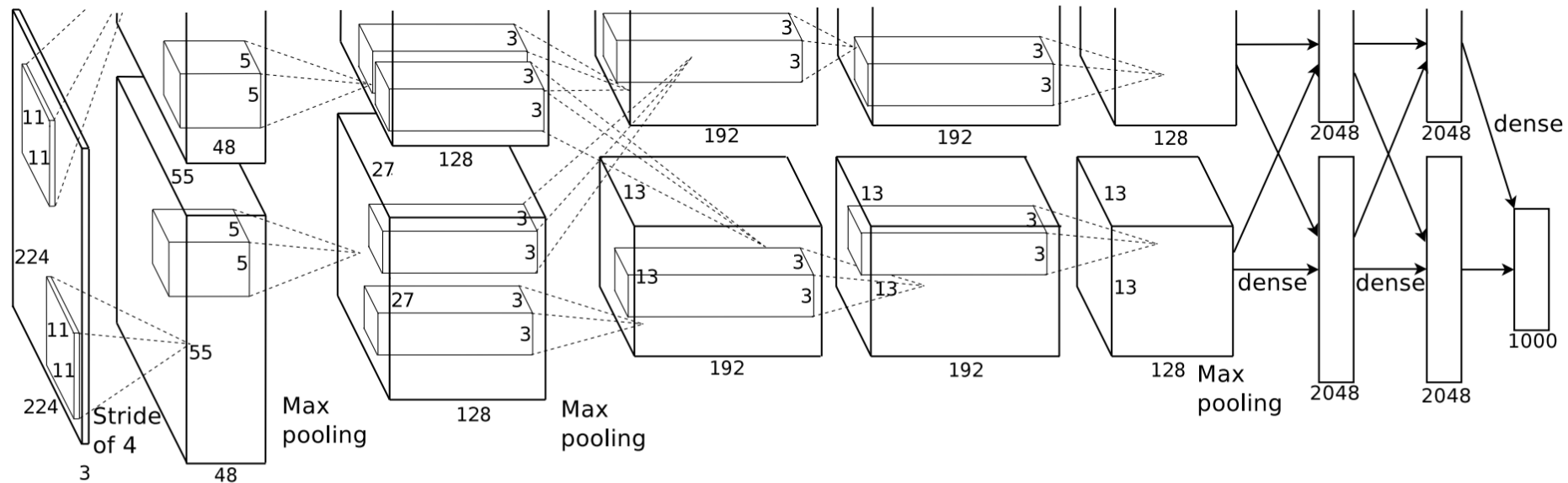
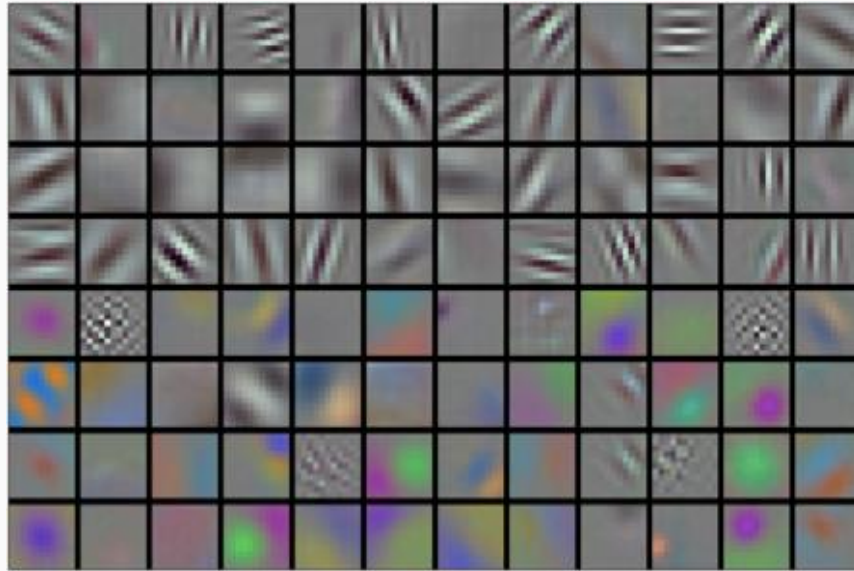


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Image filters learned by the first layer of AlexNet



The filters from the first layer of AlexNet

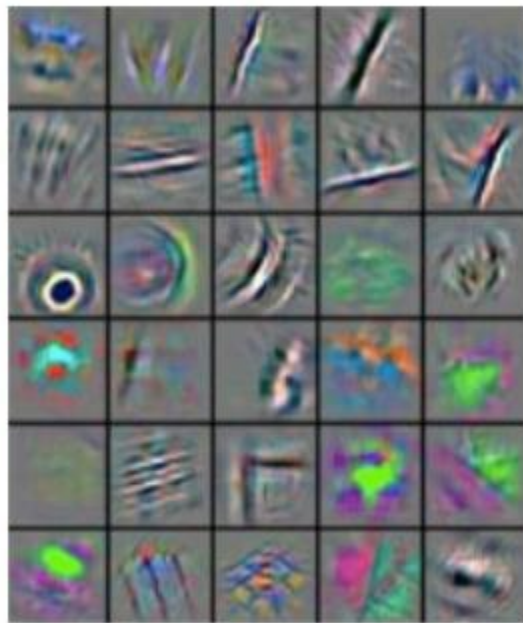
Lower layers resembled traditional filters from computer vision like edge detectors

Higher layers did indeed represent composite objects like noses, eyes, people, dogs, etc

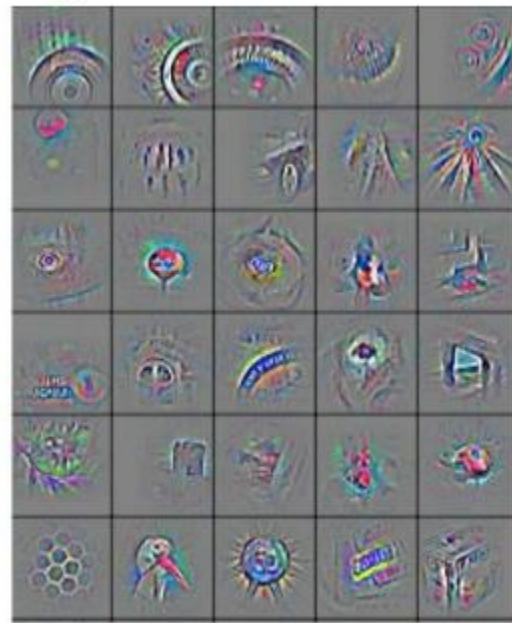
low-level features



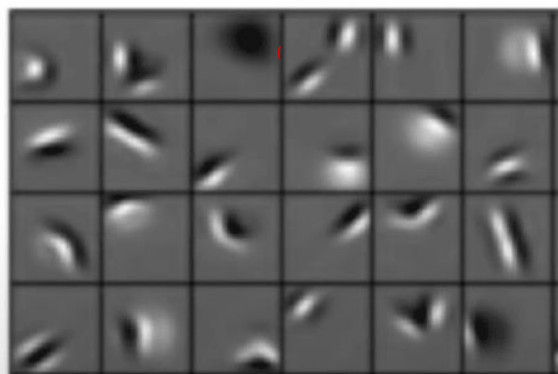
mid-level features



high-level features

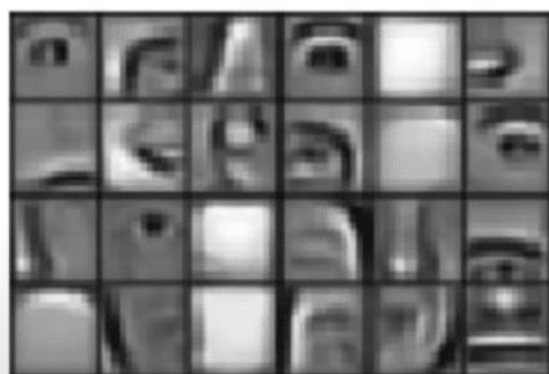


Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

High level features



Facial structure

Summary: Convolutional Neural Networks

Default building block for images and image-like data

A typical architecture involves a set of CNN layers followed by an MLP for classification

After AlexNet, there were more named networks like VGGNet, ResNet, and GoogLeNet which became standard building blocks for handling images

(Recently though, image transformers seem to be gaining popularity)

Summary: Optimizers

- Standard libraries offer standard optimizers
- The most basic optimizer is stochastic gradient descent, which we saw
- In practice, use Adam optimizer or one of its variants
 - Has many hyperparameters
 - Lots of folk knowledge about them 😞

Summary: Hyperparameters

- Larger models + more complicated optimizers = profusion of hyperparameters
- How do we find good hyperparameters?
 - Lots of experimentation with held out data
 - Often libraries come with good defaults, but not all of them will work everywhere
 - Yet, there is a bit of an art to this

Tutorials and references

- Basic Pytorch Tutorials
 - https://pytorch.org/tutorials/beginner/deep_learning_60min_boltz.html
 - https://clemsonciti.github.io/rcde_workshops/pytorch/03-regression_and_classification.html
 - <https://machinelearningmastery.com/building-a-regression-model-in-pytorch/>
 - <https://www.geeksforgeeks.org/classification-using-pytorch-linear-function/>
- Free, interactive Deep Learning textbook
 - <https://d2l.ai/>