Introduction to Deep Learning

and its applications to the LHC

David Shih

IBS-CTPU workshop:
"Particle Physics in Computing Frontier"

December 2019

Plan of the lectures

1. Introduction to Deep Learning, part 1
2. Introduction to Deep Learning, part 2
3. Deep Learning at the LHC

I will assume most of you know some collider physics.

I will not assume any familiarity with machine learning or neural networks.

Special thanks to Bryan Ostdiek for letting me draw from his slides
https://github.com/bostdiek/IntroToMachineLearning/tree/master/Slides

Brief (re)fresher on machine learning

"ML is glorified function fitting"

Want to fit a function $f(x; \theta)$ with some parameters θ ("weights") to a collection of examples $\{x_i\}$ in order to achieve some objective.

Brief (re)fresher on machine learning

"ML is glorified function fitting"

Want to fit a function $f(x; \theta)$ with some parameters θ ("weights") to a collection of examples $\{x_i\}$ in order to achieve some objective.

Finally, what functions to use?

Brief (re)fresher on machine learning

"ML is glorified function fitting"

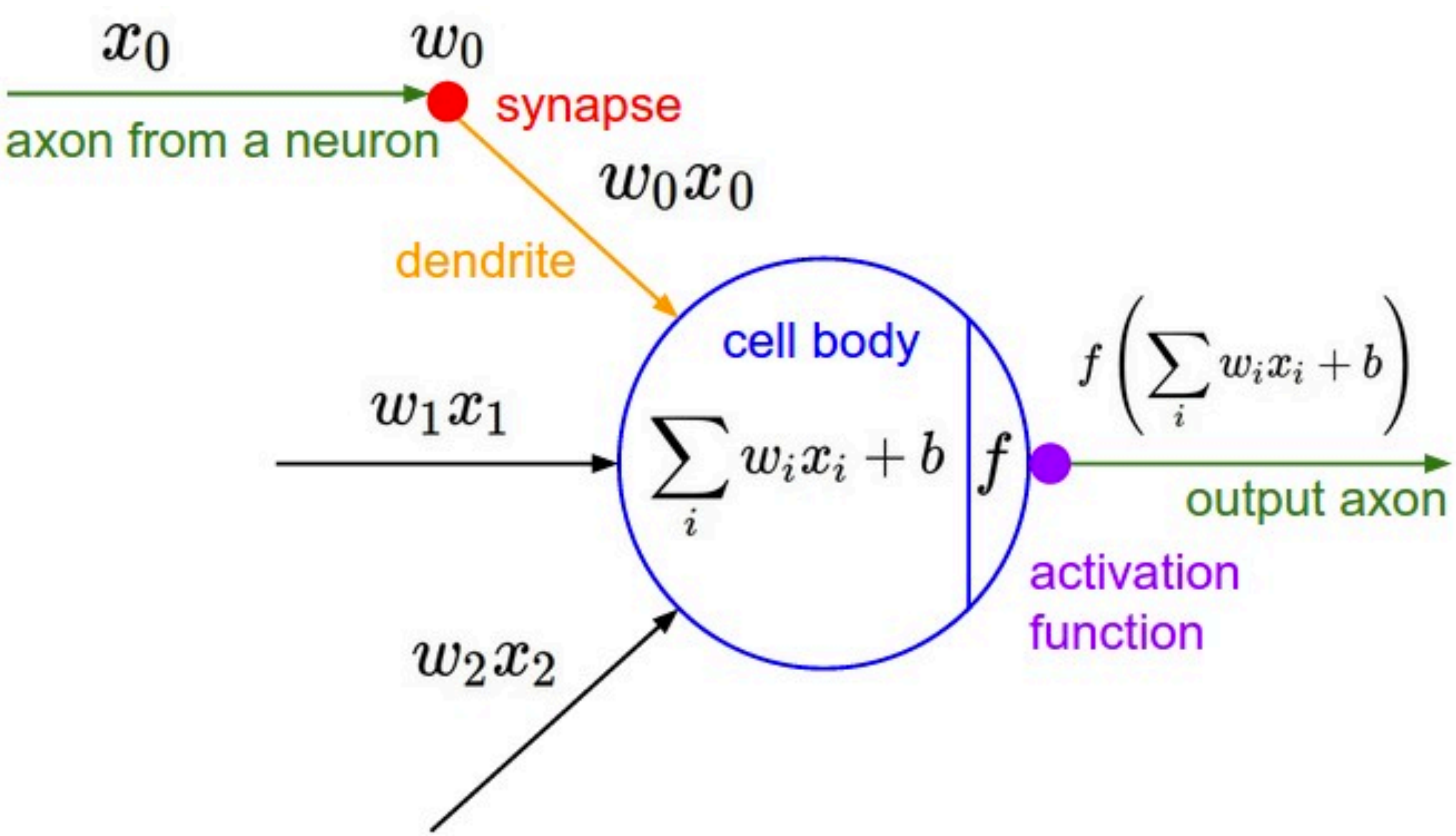
Want to fit a function $f(x; \theta)$ with some parameters θ ("weights") to a collection of examples $\{x_i\}$ in order to achieve some objective.

Finally, what functions to use?

Current trend: deep neural networks!

What is a (deep) neural network?

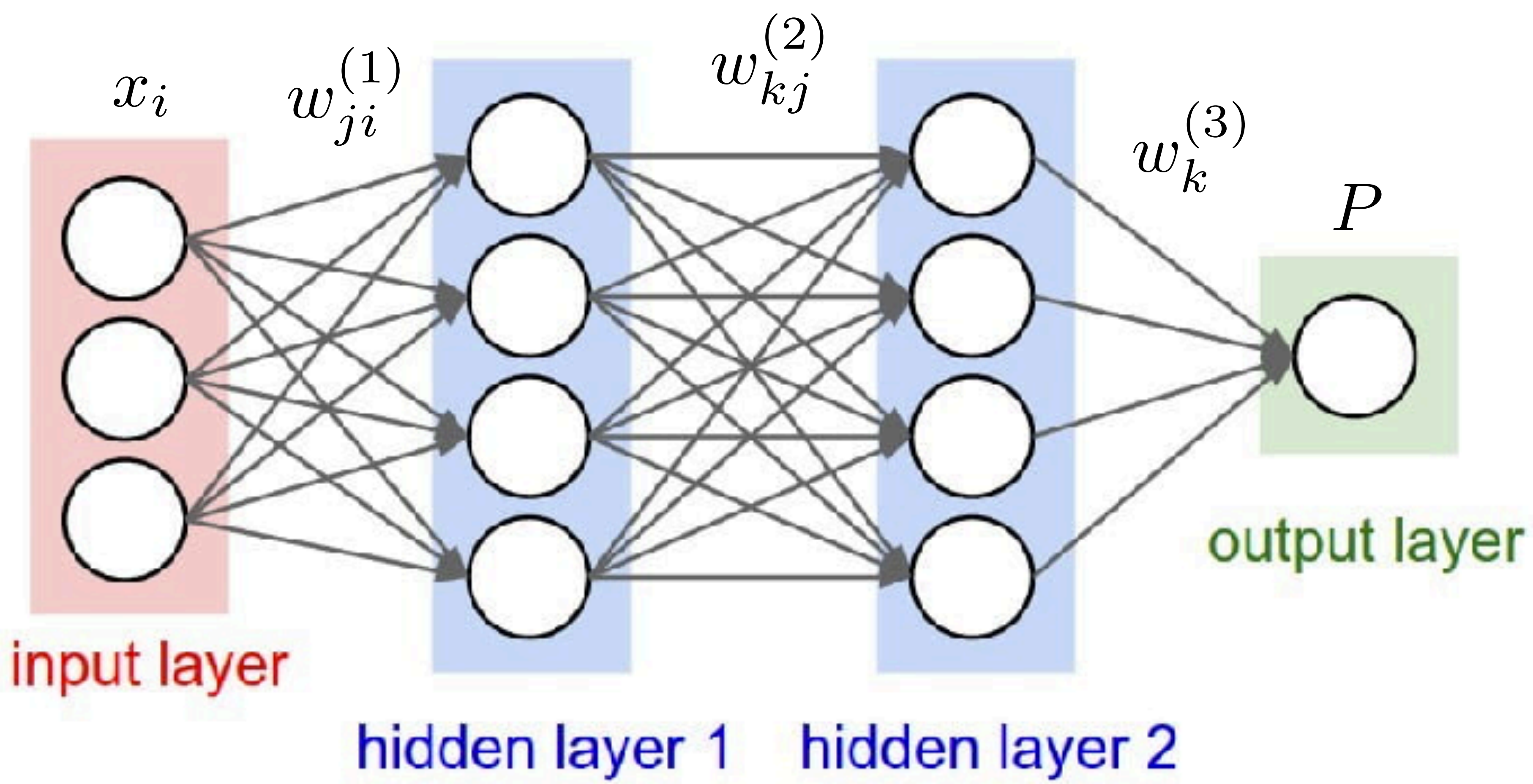
Basic building block of a neural network:



Modeled on biological systems

What is a (deep) neural network?

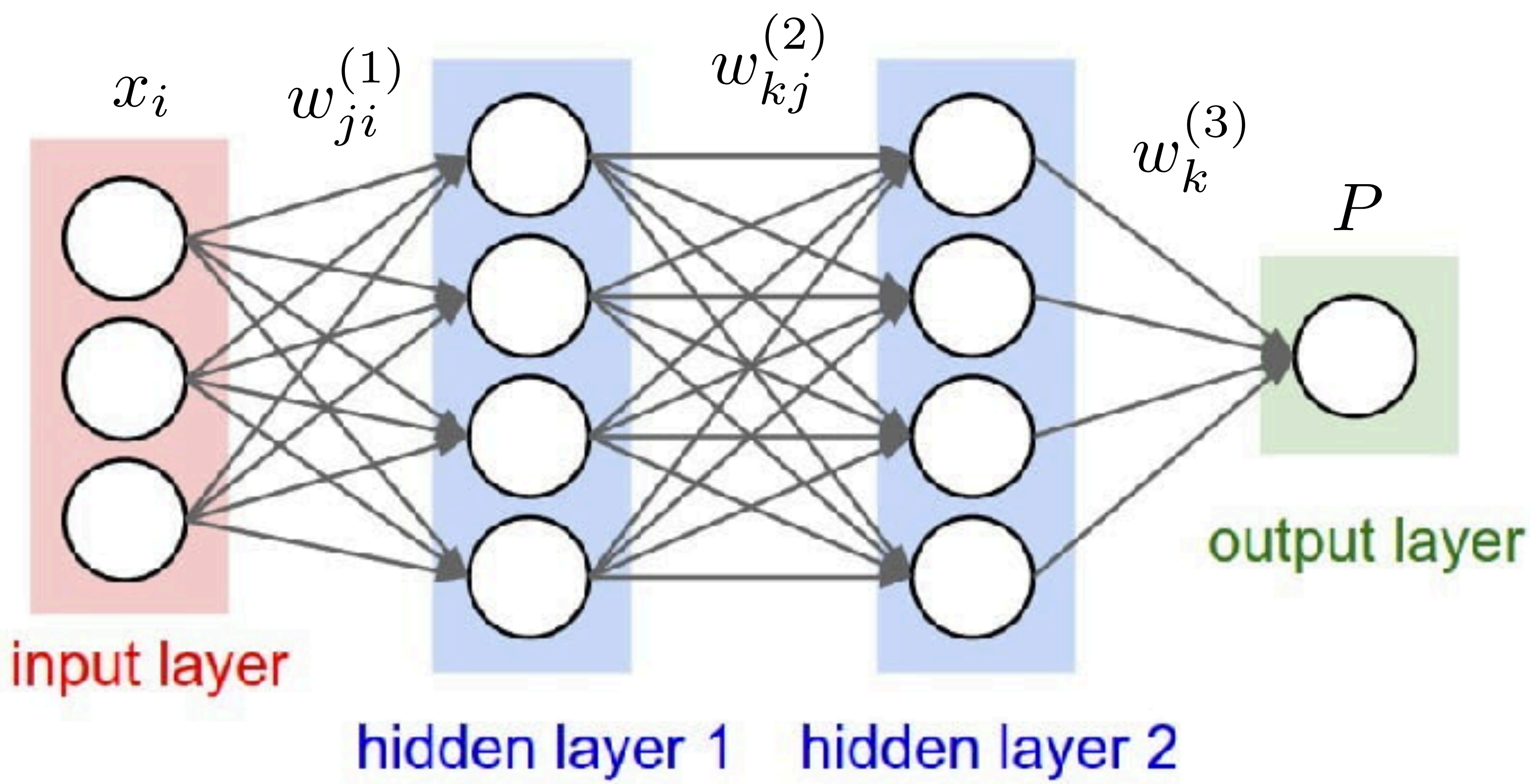
“fully connected” or “dense” NN



$$P = A^{(3)}(w_k^{(3)} A^{(2)}(w_{kj}^{(2)} A^{(1)}(w_{ji}^{(1)} x_i + b_j^{(1)}) + b_k^{(2)}) + b^{(3)})$$

What is a (deep) neural network?

"fully connected" or "dense" NN



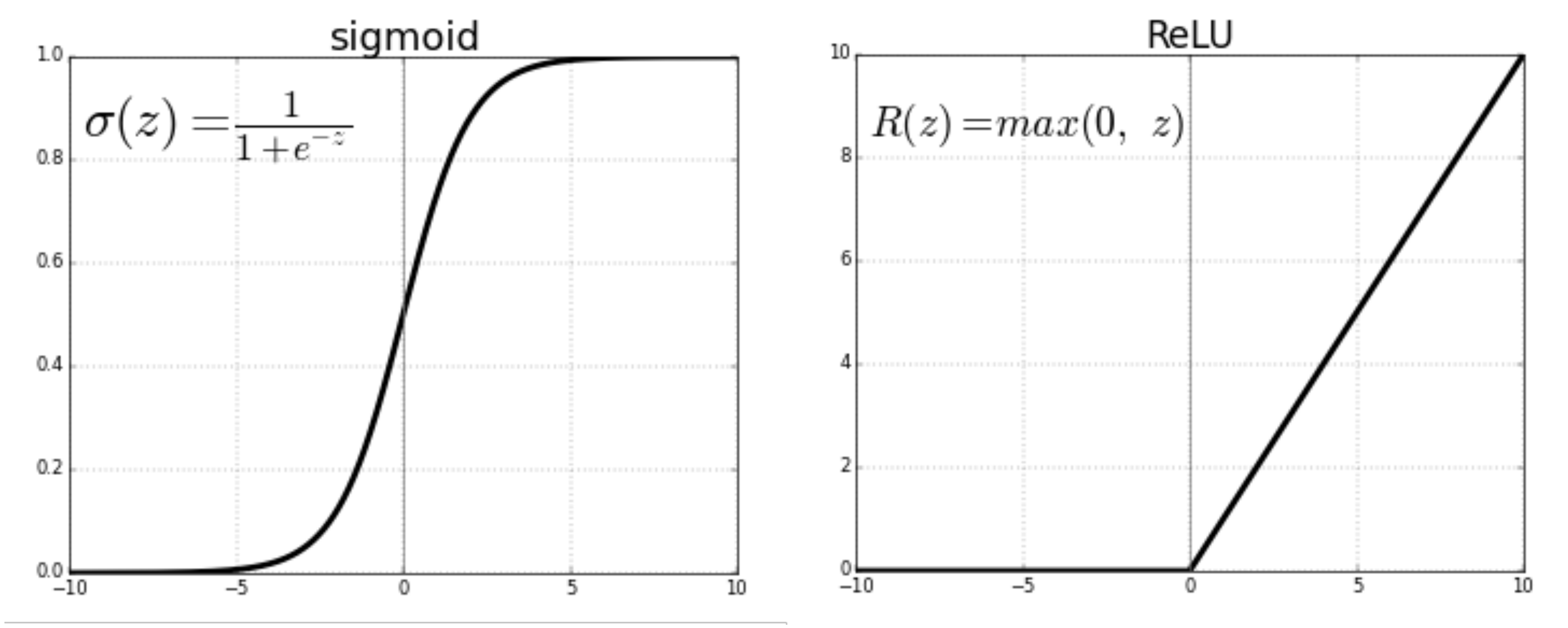
$$P = A^{(3)}(w_k^{(3)}A^{(2)}(w_{kj}^{(2)}A^{(1)}(w_{ji}^{(1)}x_i + b_j^{(1)}) + b_k^{(2)}) + b^{(3)})$$

If f(x; w) involves a multi-layered neural network, it is called *"deep learning"*

What is a (deep) neural network?

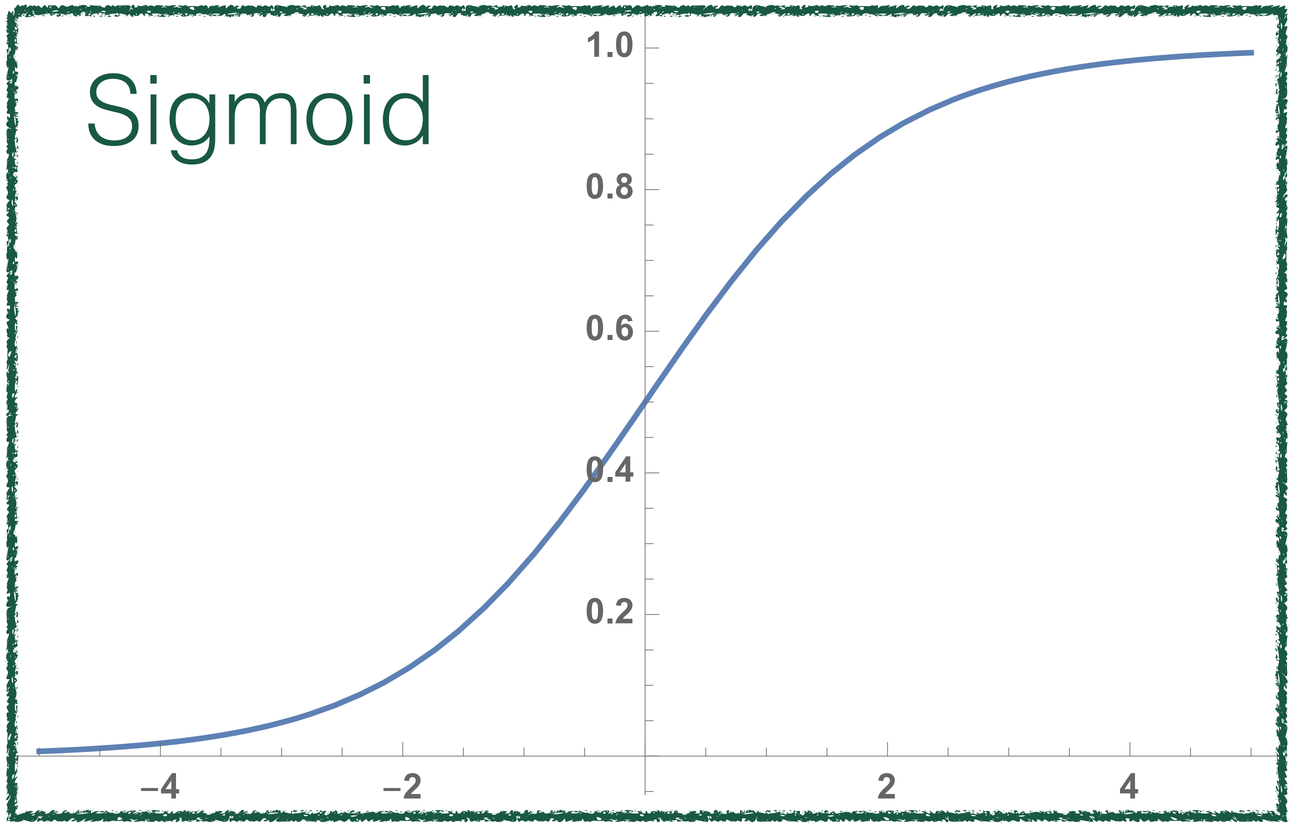
NNs need a source of non-linearity so they can learn general functions.

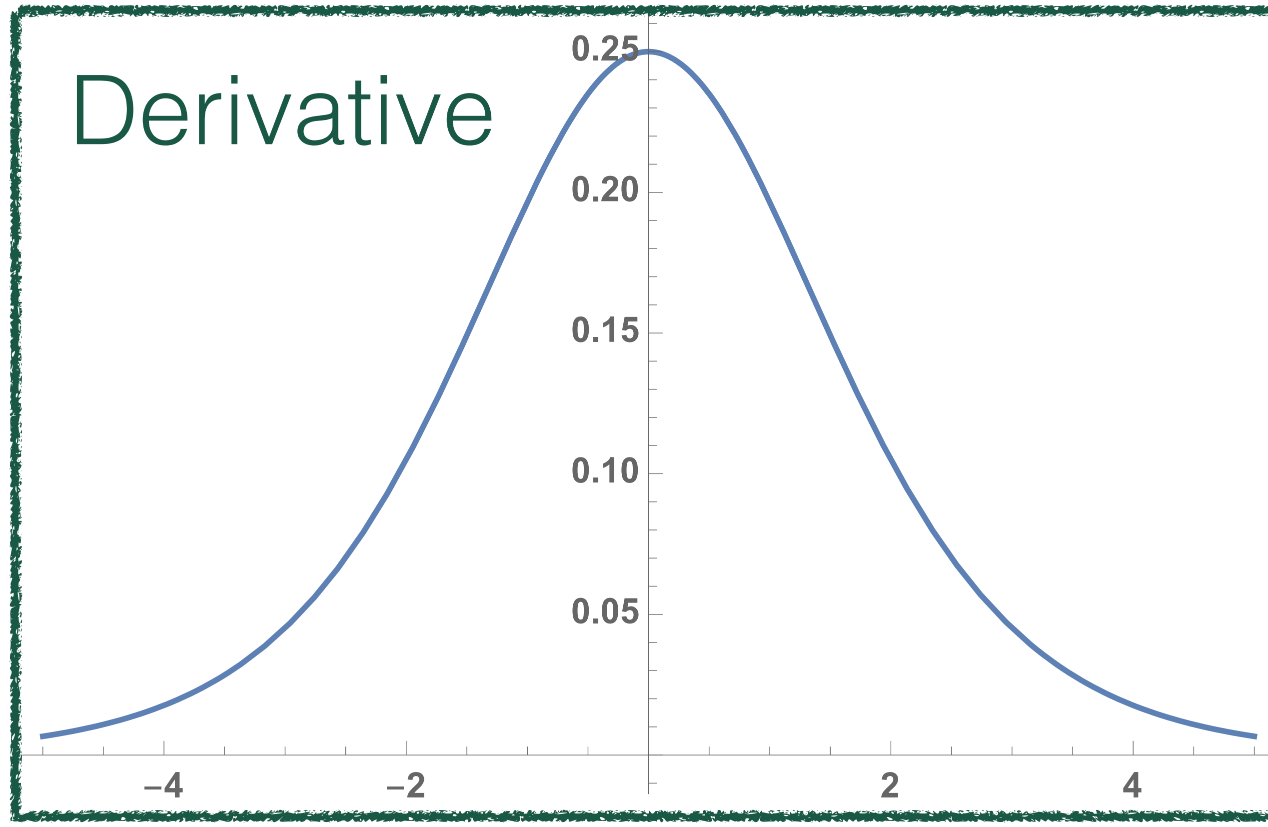
This is usually implemented with the activation function.



Sigmoid used to be standard. But this led to the vanishing gradient problem. The ReLU activation was invented to solve this problem. Now it is the standard.

Disappearing Gradient





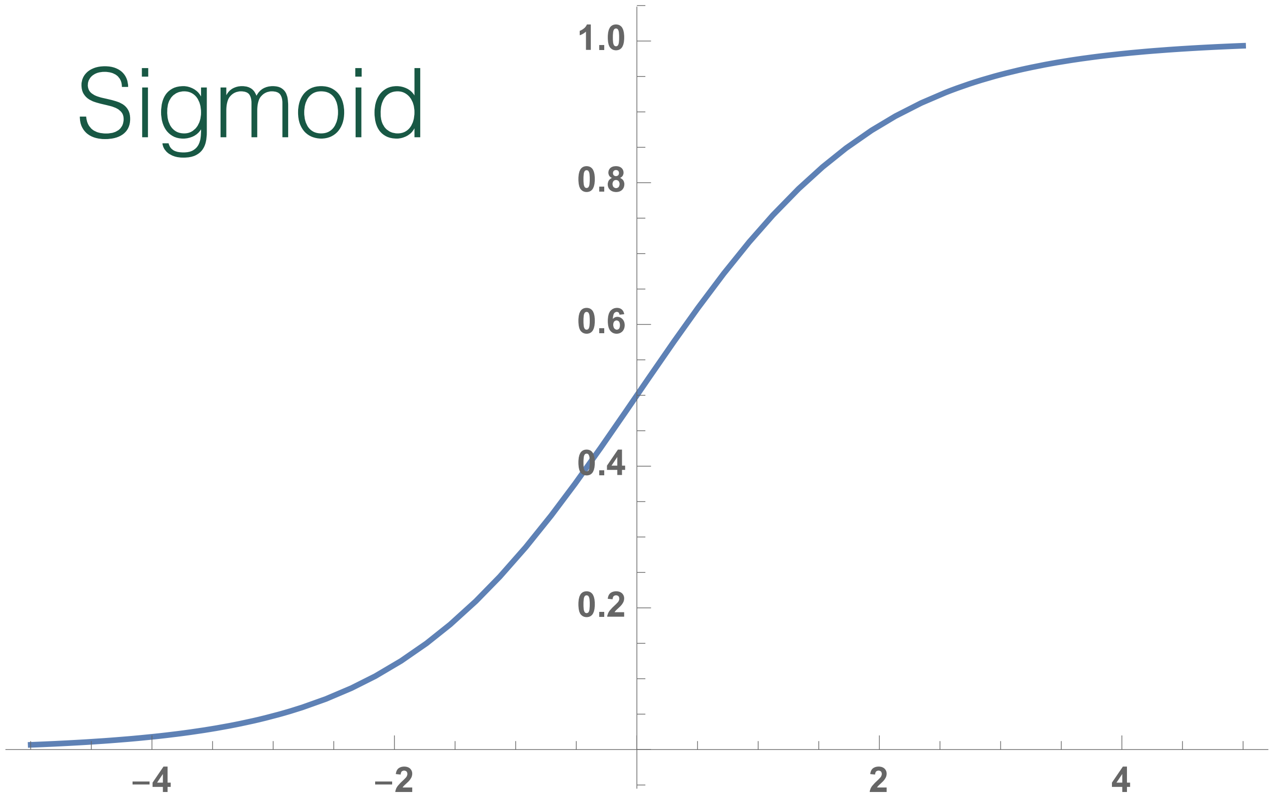
Chain rule for gradient of network involves multiple factors of the derivative multiplied together

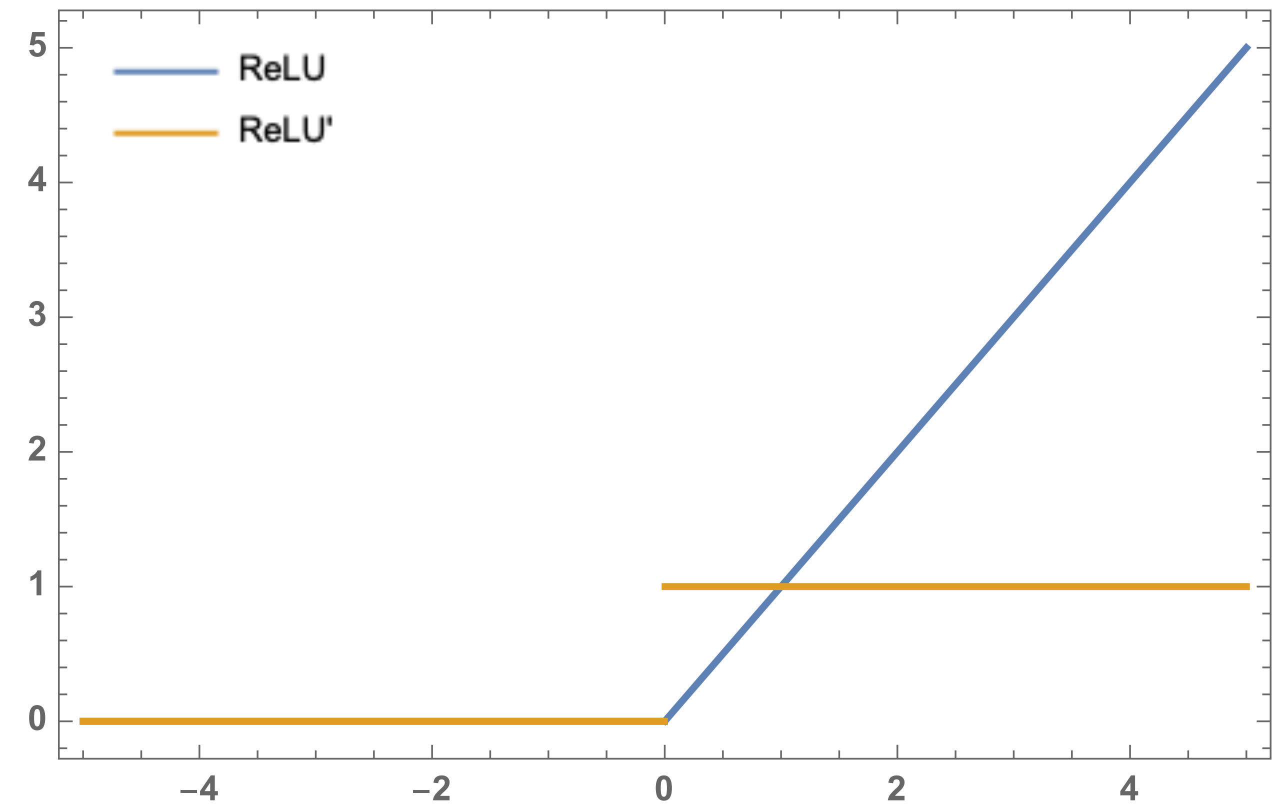
$$\boxed{(0.25)^4 = 0.0039}$$

Deep networks with Sigmoid activations have exponentially hard time training early layers

Disappearing Gradient

Sigmoid





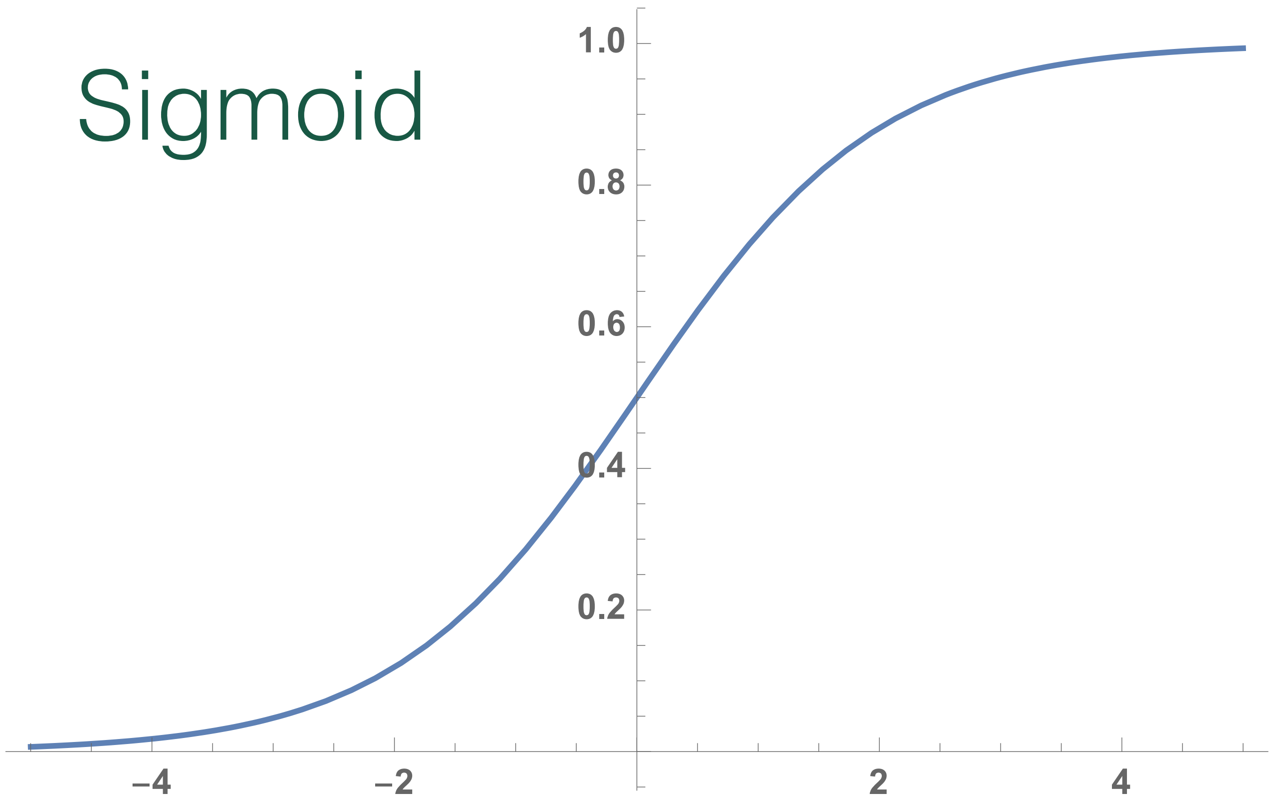
Using the Rectified Linear Unit (ReLU) solves this problem.
ReLU(x) = {0 if x <=0, x if x >0}

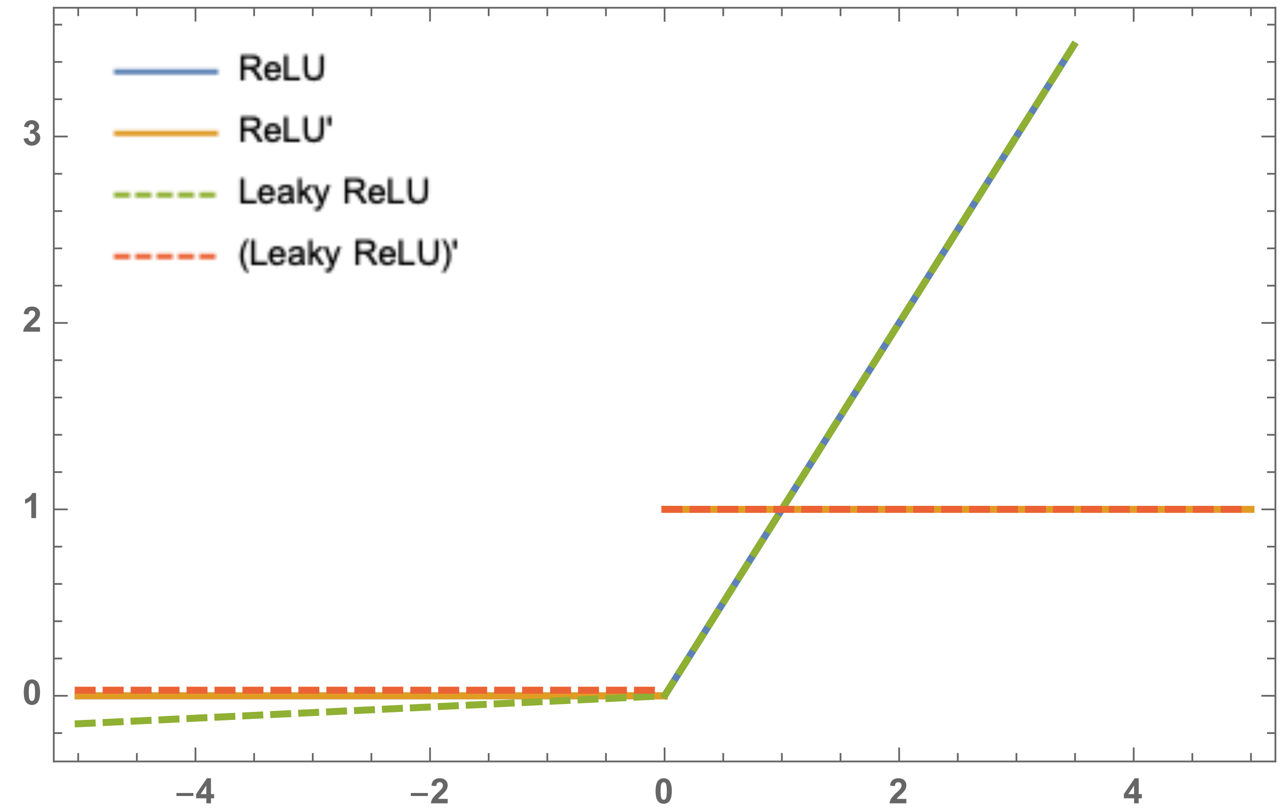
Still has nonlinearity which allows network to learn complicated patterns

Nodes can die (derivative always 0 so cannot update)

Disappearing Gradient

Sigmoid





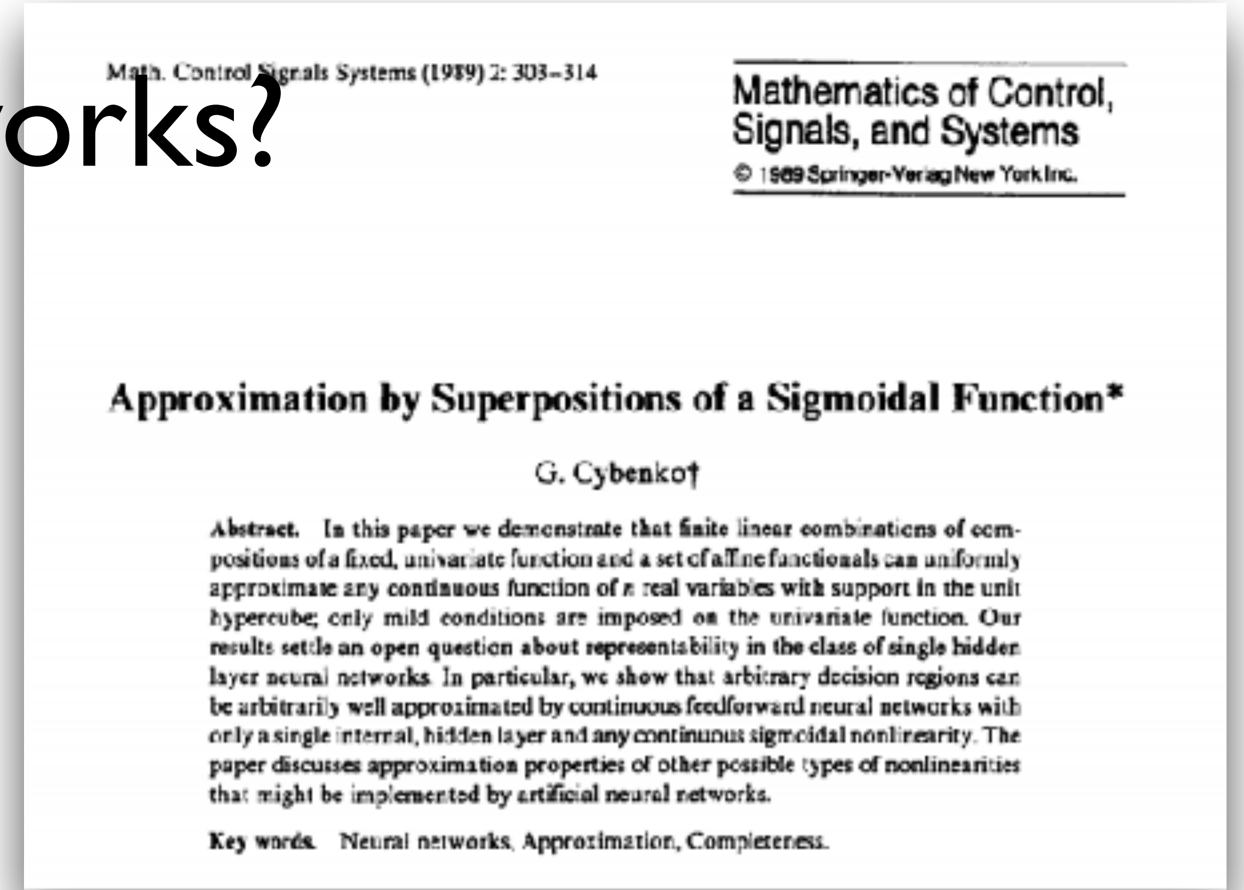
Leaky Rectified Linear Unit (LeakyReLU) solves this problem.

LeakyReLU(x) = {alpha*x if x <=0, x if x >0}

I have never had to use this in practice

Why (deep) neural networks?

One reason why NNs "work" is that they are *universal function approximators* (at least in the infinite limit).


Math. Control Signals Systems (1989) 2: 303–314

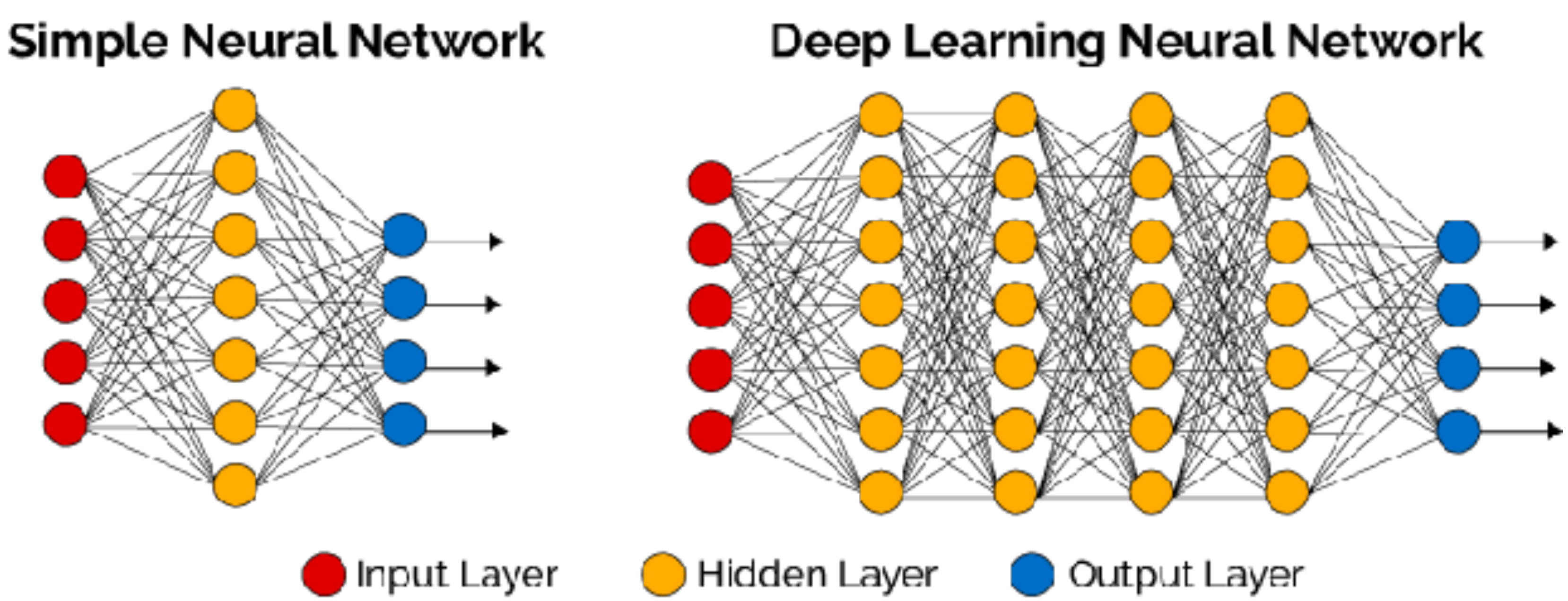
Mathematics of Control, Signals, and Systems

Approximation by Superpositions of a Sigmoidal Function*

G. Cybenko†

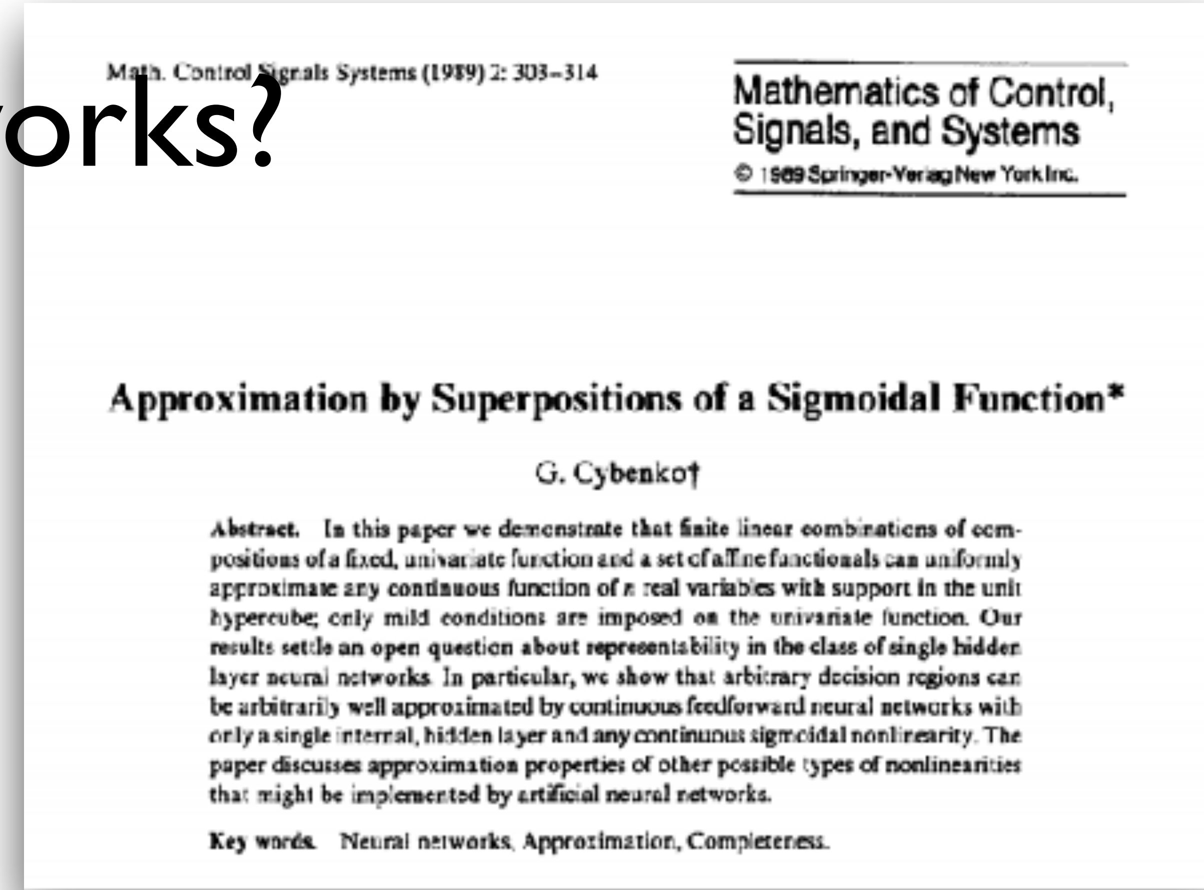
Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single hidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity. The paper discusses approximation properties of other possible types of nonlinearities that might be implemented by artificial neural networks.

Key words. Neural networks, Approximation, Completeness.



slide credit: Bryan Ostdiek

Why (deep) neural networks?



Math. Control Signals Systems (1989) 2: 303–314

Mathematics of Control, Signals, and Systems

Approximation by Superpositions of a Sigmoidal Function*

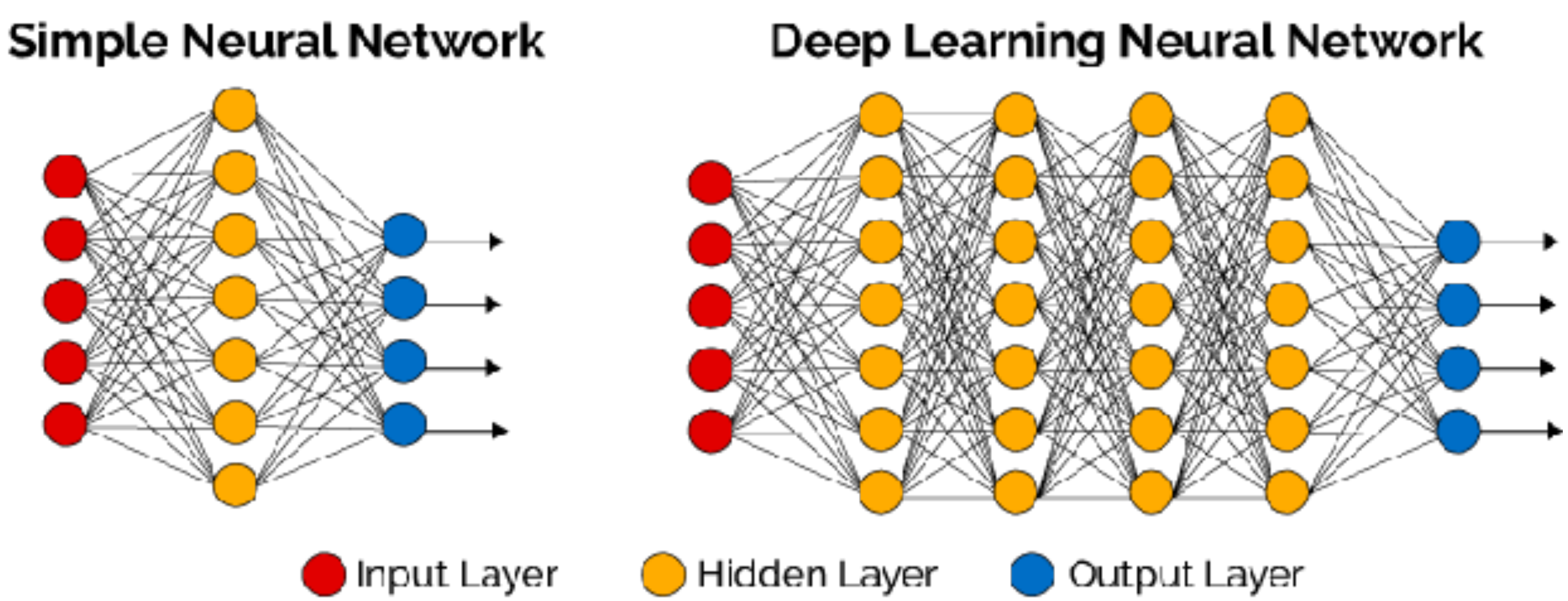
G. Cybenko†

Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single hidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity. The paper discusses approximation properties of other possible types of nonlinearities that might be implemented by artificial neural networks.

Key words. Neural networks, Approximation, Completeness.

One reason why NNs "work" is that they are *universal function approximators* (at least in the infinite limit).

Can fit any function with infinite data and infinite nodes (1 hidden layer)

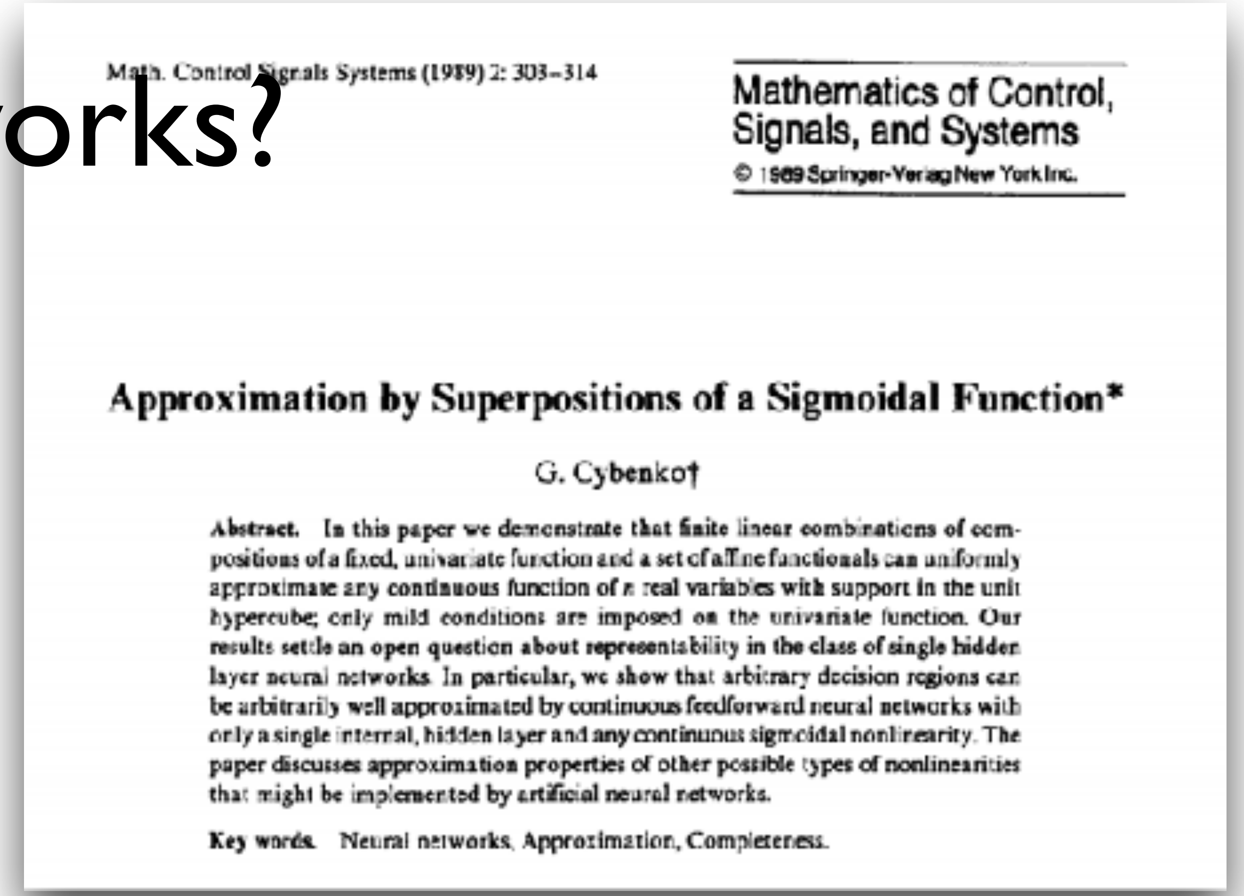


slide credit: Bryan Ostdiek

Why (deep) neural networks?

One reason why NNs "work" is that they are *universal function approximators* (at least in the infinite limit).

Can fit any function with infinite data and infinite nodes (1 hidden layer)


Math. Control Signals Systems (1989) 2: 303–314

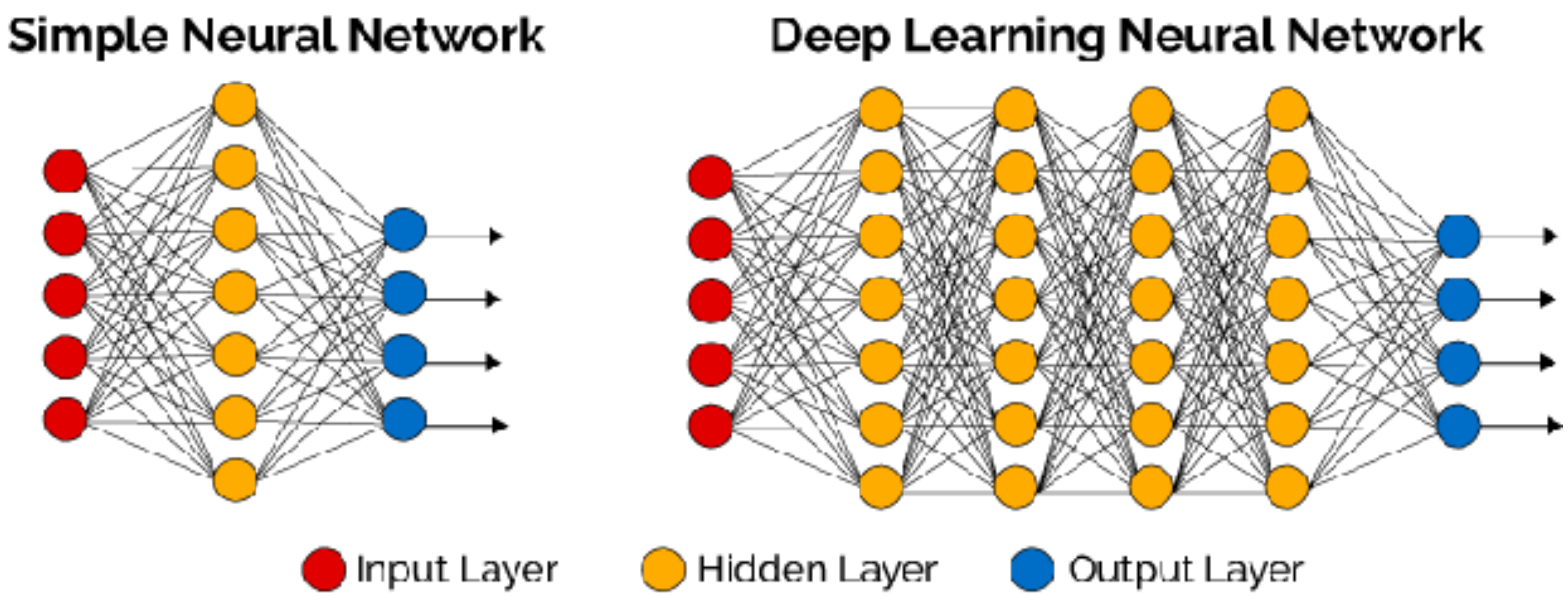
Mathematics of Control, Signals, and Systems
© 1989 Springer-Verlag New York Inc.

Approximation by Superpositions of a Sigmoidal Function*

G. Cybenko†

Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single hidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity. The paper discusses approximation properties of other possible types of nonlinearities that might be implemented by artificial neural networks.

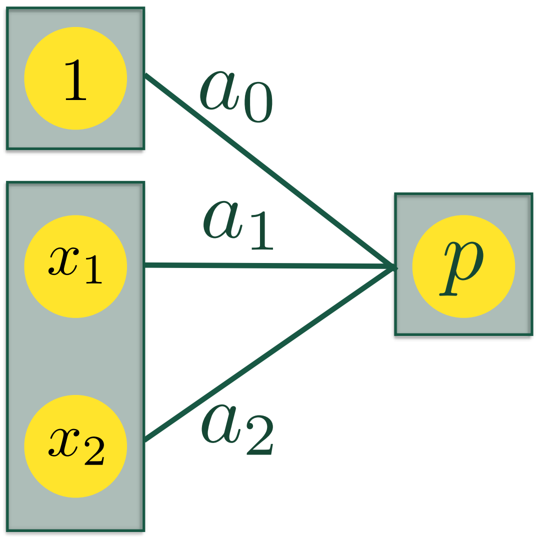
Key words. Neural networks, Approximation, Completeness.



Going deeper rather than wider learns non-linearities with fewer parameters

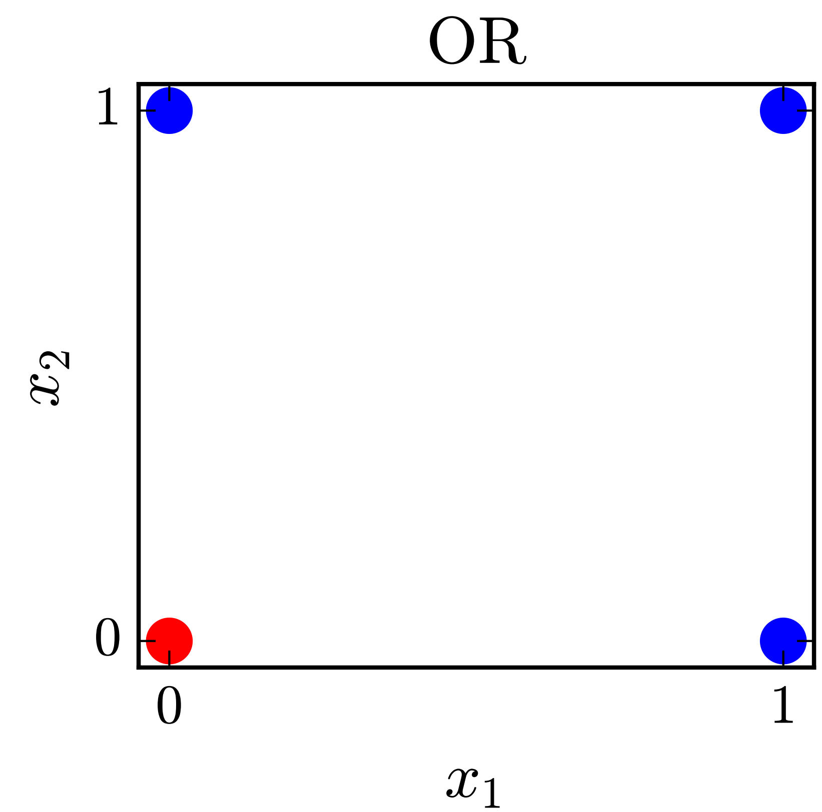
slide credit: Bryan Ostdiek

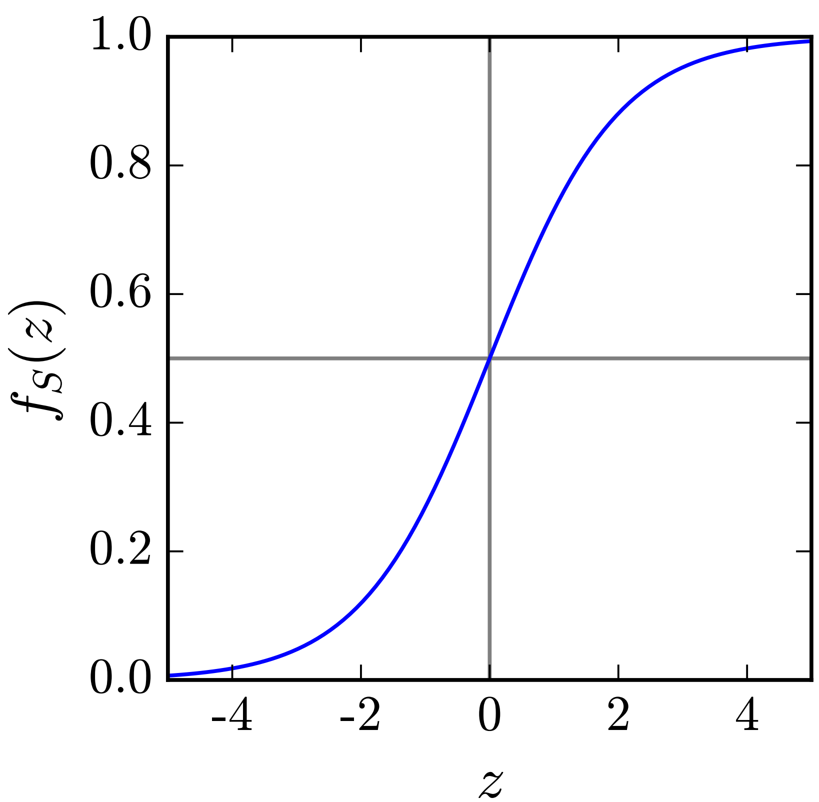
A Simple Example



OR

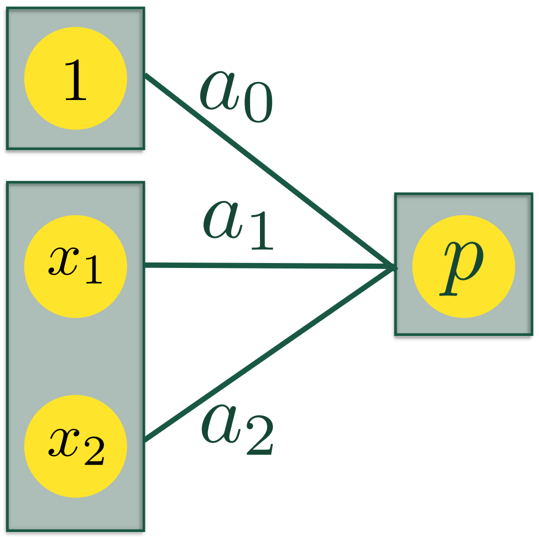
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1





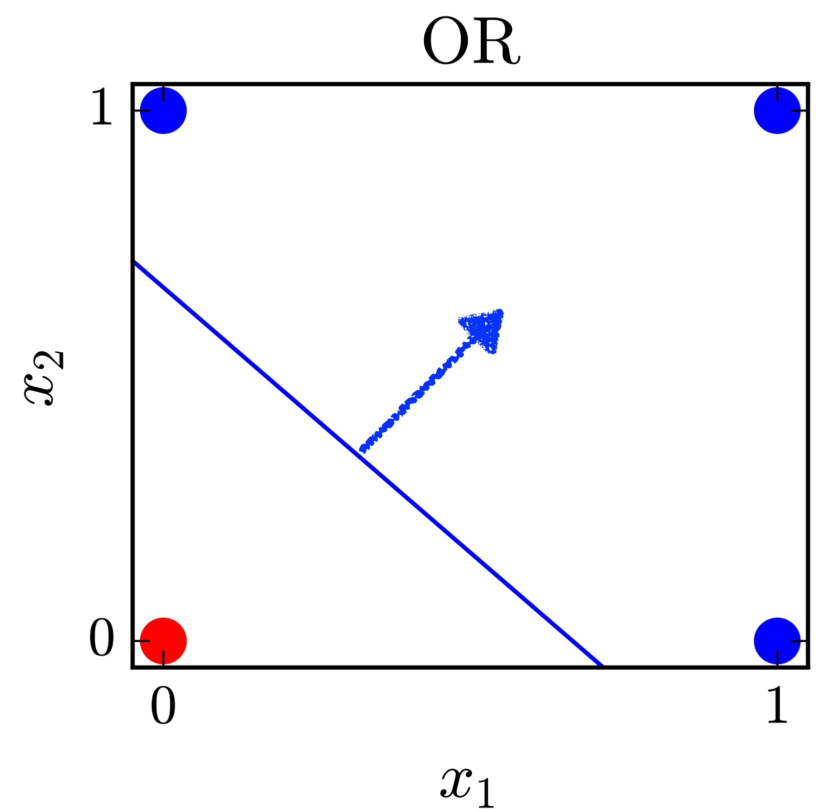
slide credit: Bryan Ostdiek

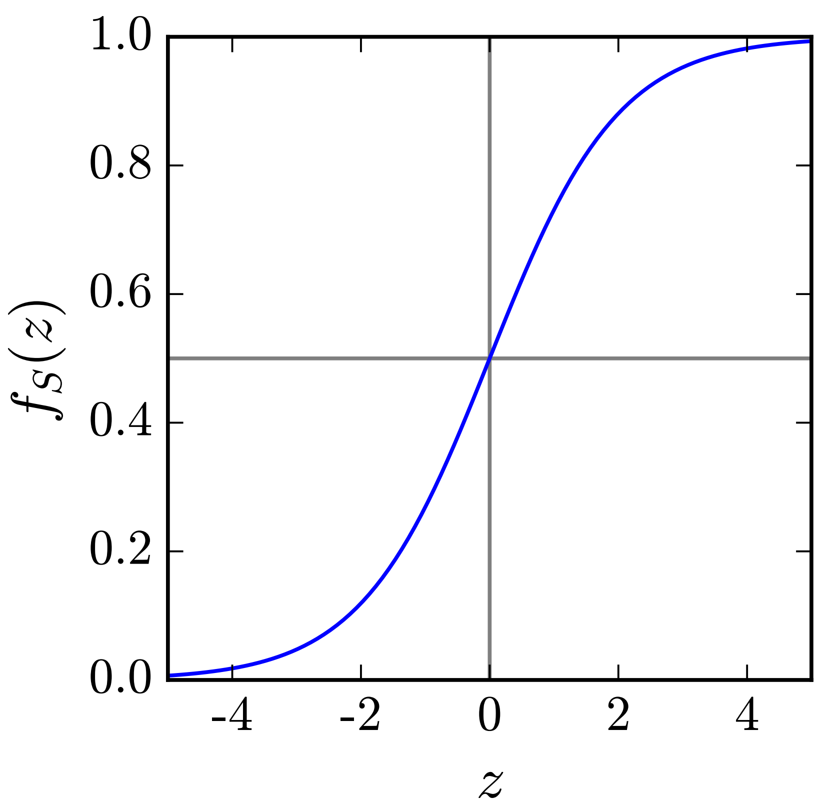
A Simple Example



OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

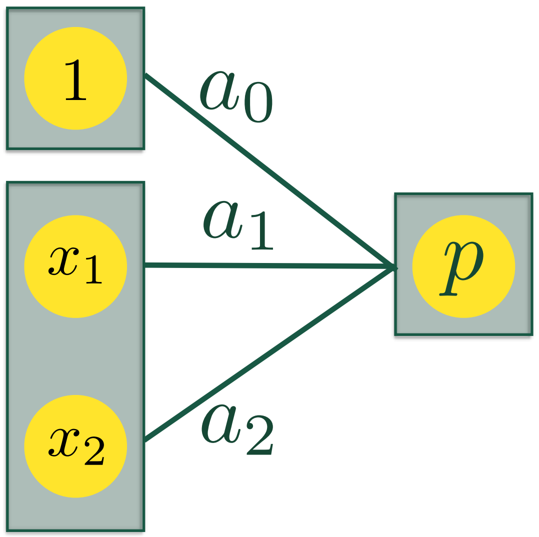


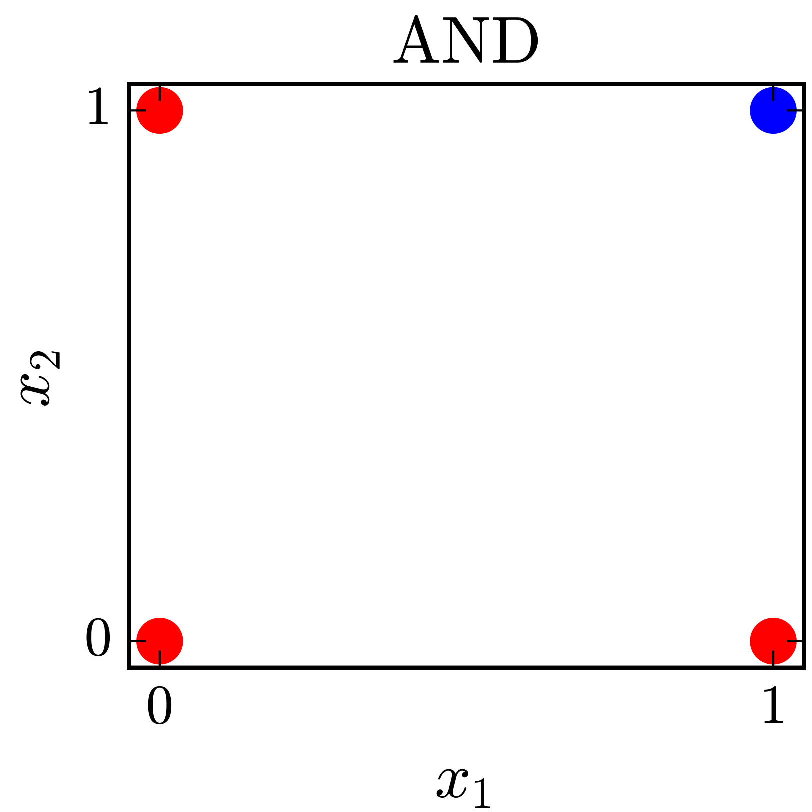


$$a_0 = -10, \quad a_1 = 15, \quad a_2 = 15$$

slide credit: Bryan Ostdiek

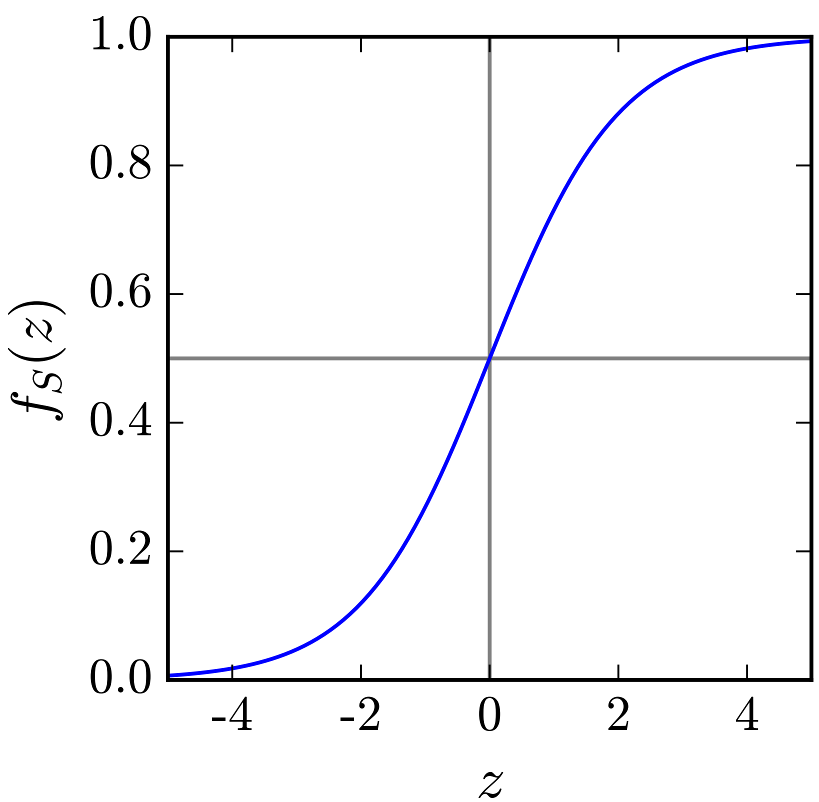
A Simple Example





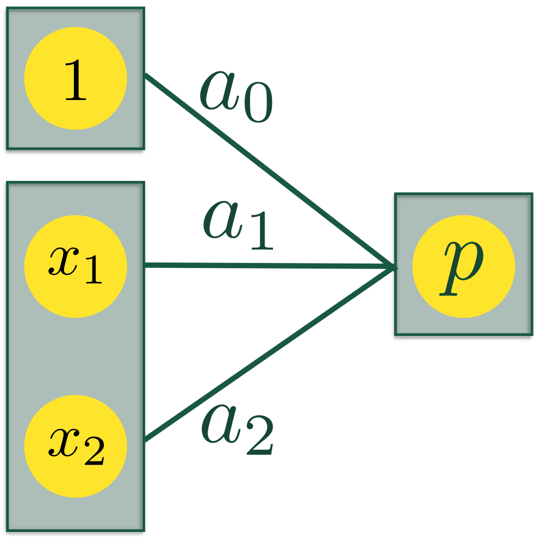
AND

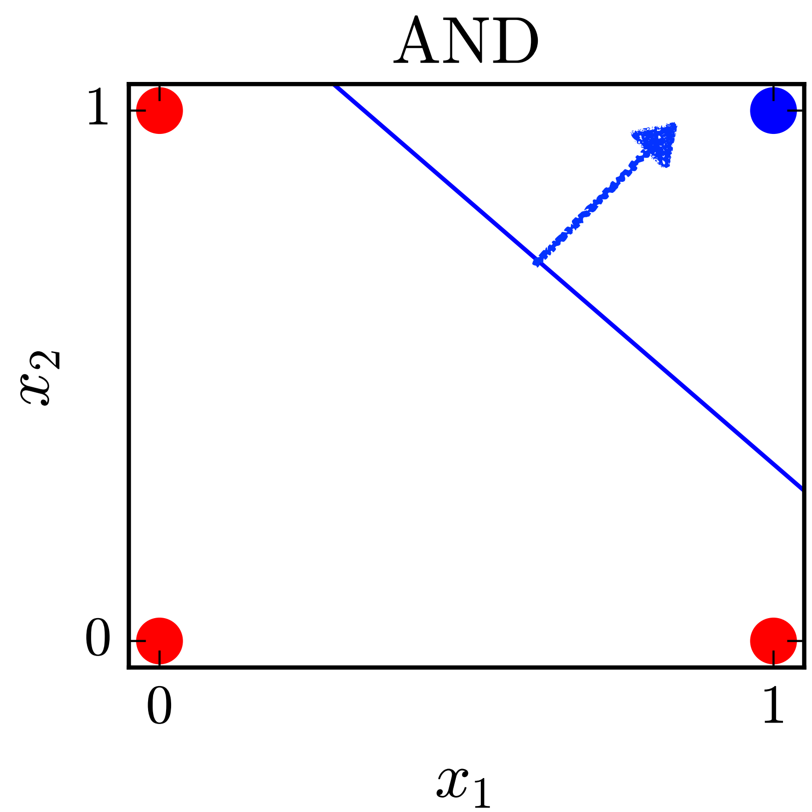
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



slide credit: Bryan Ostdiek

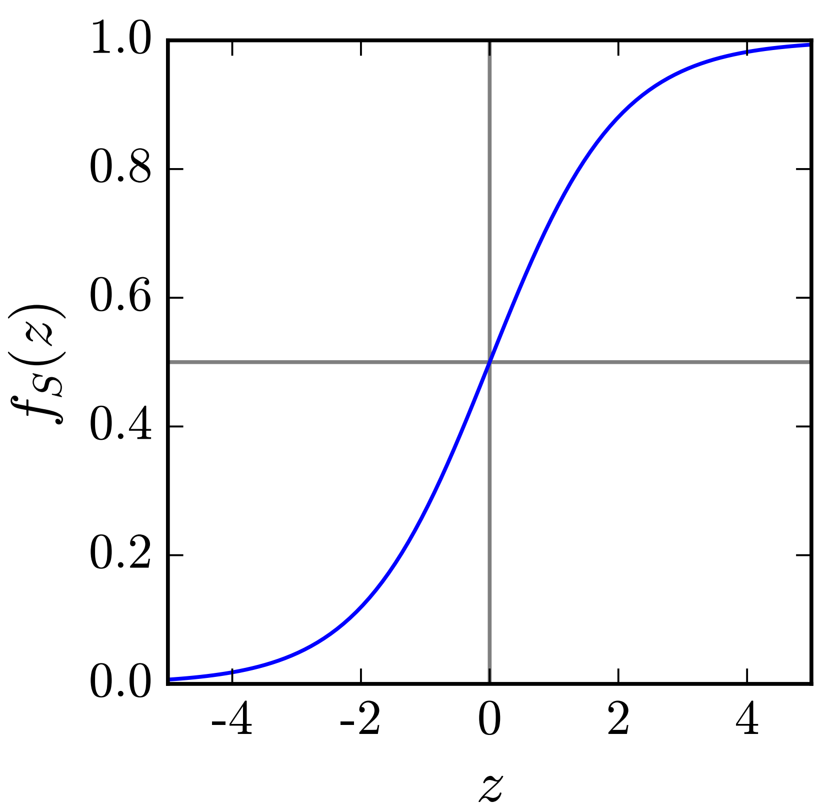
A Simple Example





AND

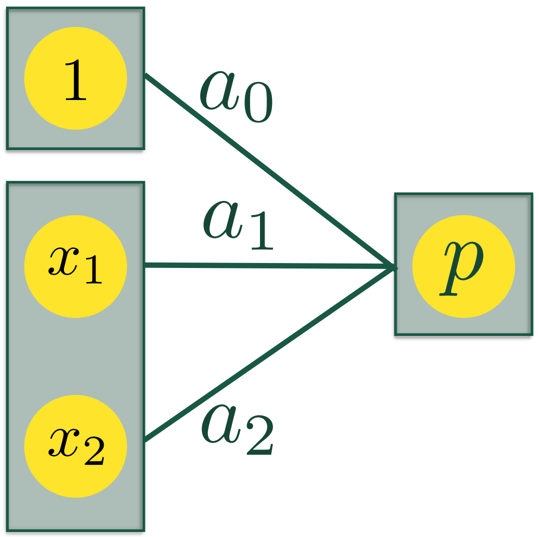
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



$$a_0 = -20, \quad a_1 = 15, \quad a_2 = 15$$

slide credit: Bryan Ostdiek

A Simple Example



OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

$$a_0 = -10, \quad a_1 = 15, \quad a_2 = 15$$

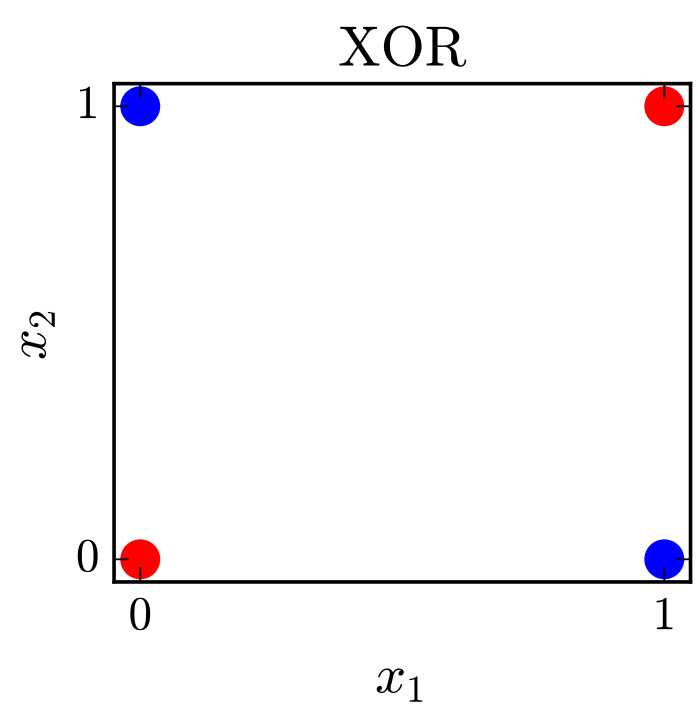
AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$$a_0 = -20, \quad a_1 = 15, \quad a_2 = 15$$

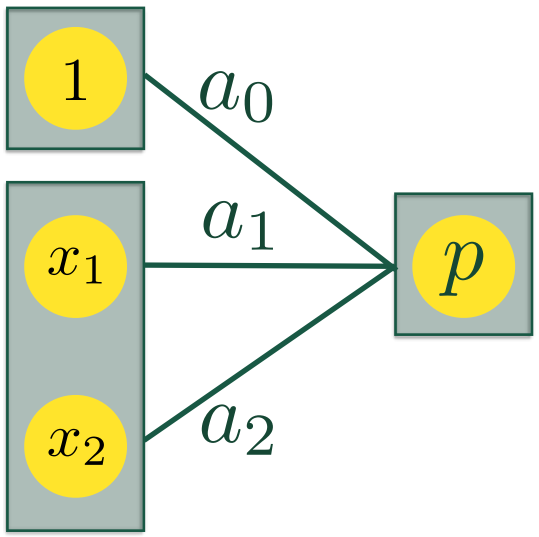
XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



slide credit: Bryan Ostdiek

A Simple Example



OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

$a_0 = -10, \quad a_1 = 15, \quad a_2 = 15$

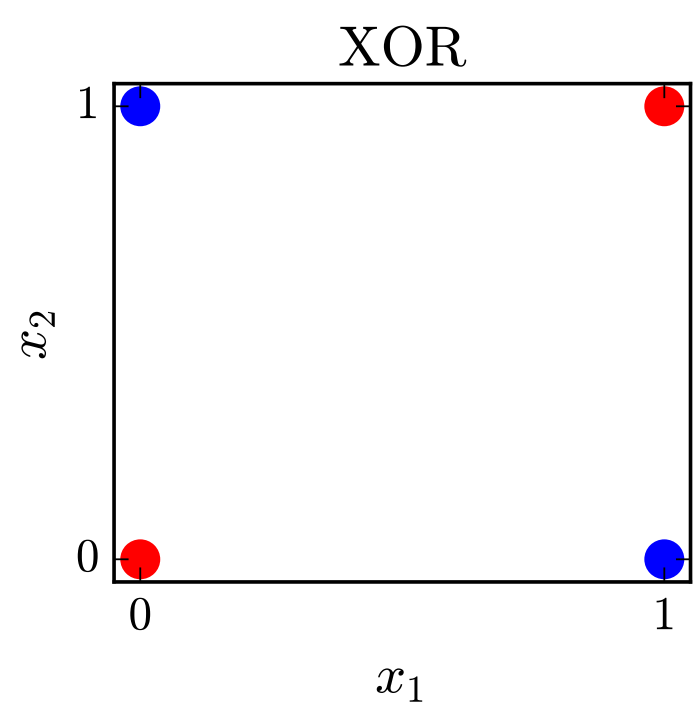
AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$a_0 = -20, \quad a_1 = 15, \quad a_2 = 15$

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



This system cannot produce XOR

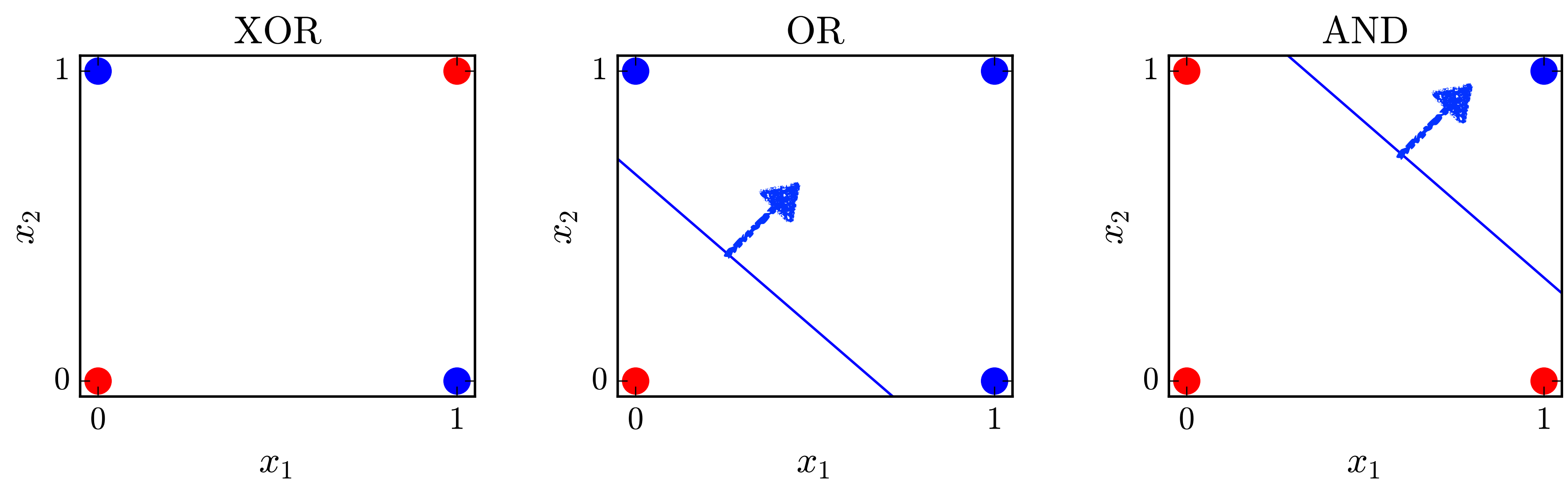
(cannot make a two sided cut)

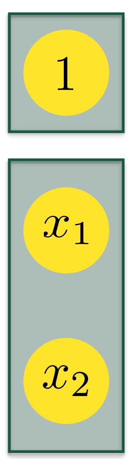
slide credit: Bryan Ostdiek

A Simple Example

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

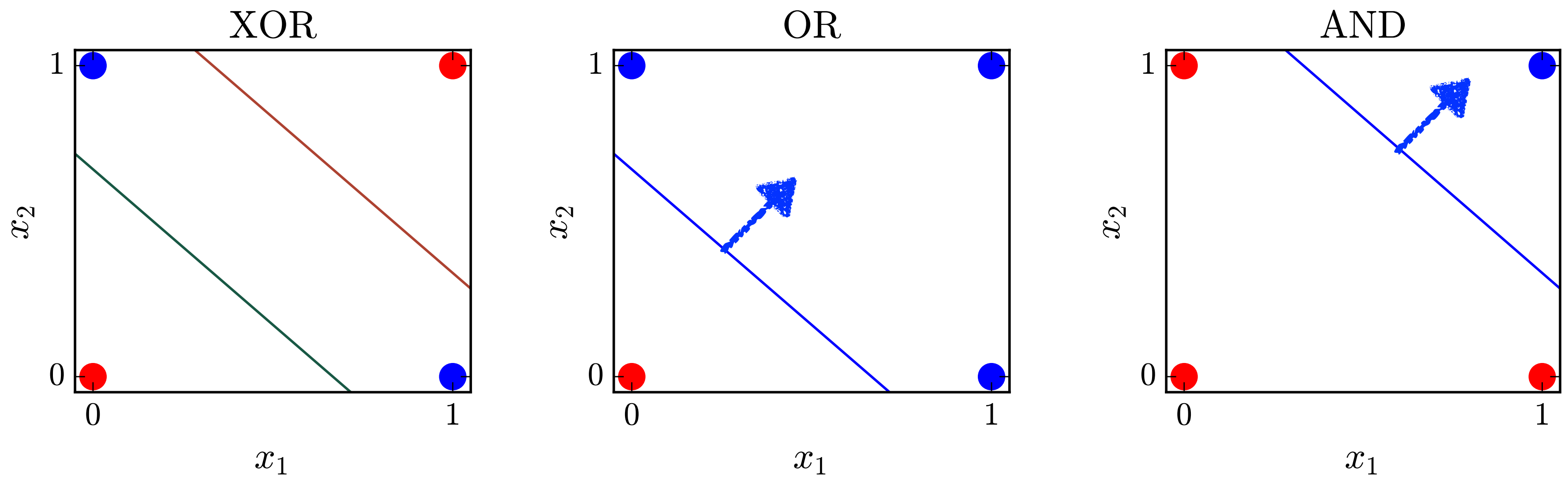


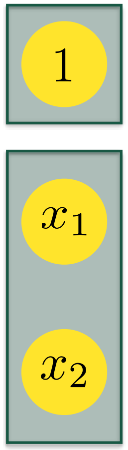


slide credit: Bryan Ostdiek

A Simple Example

XOR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



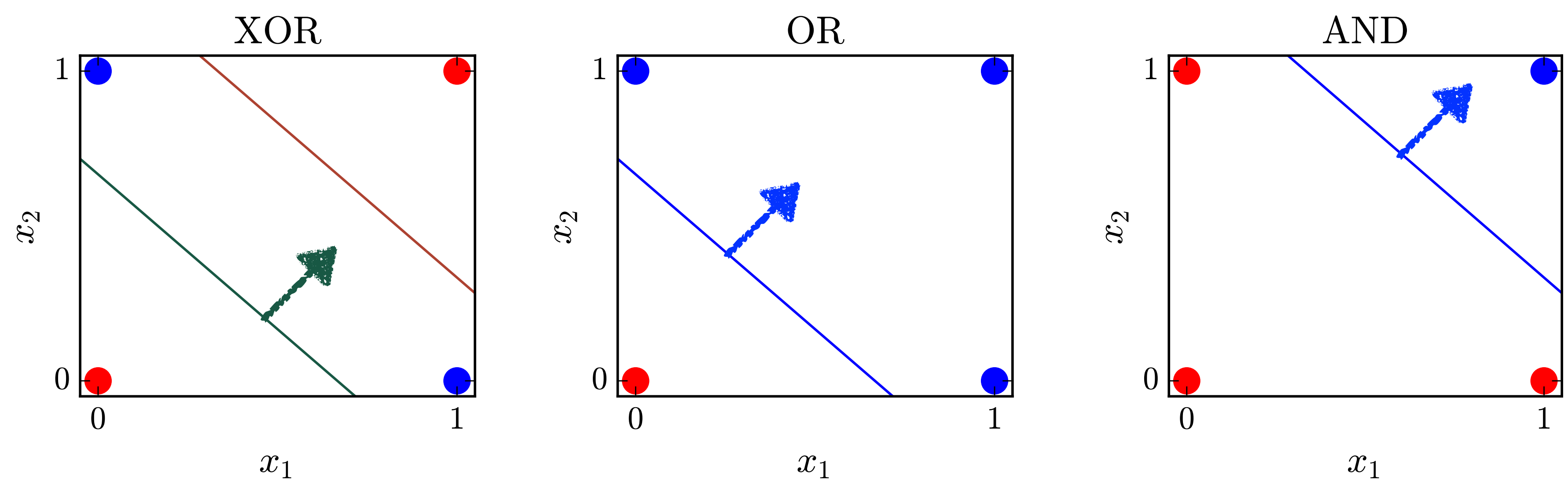


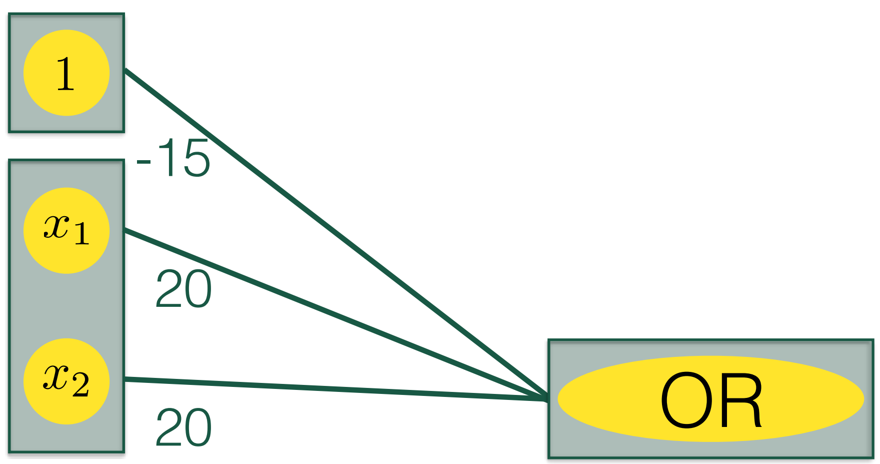
slide credit: Bryan Ostdiek

A Simple Example

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



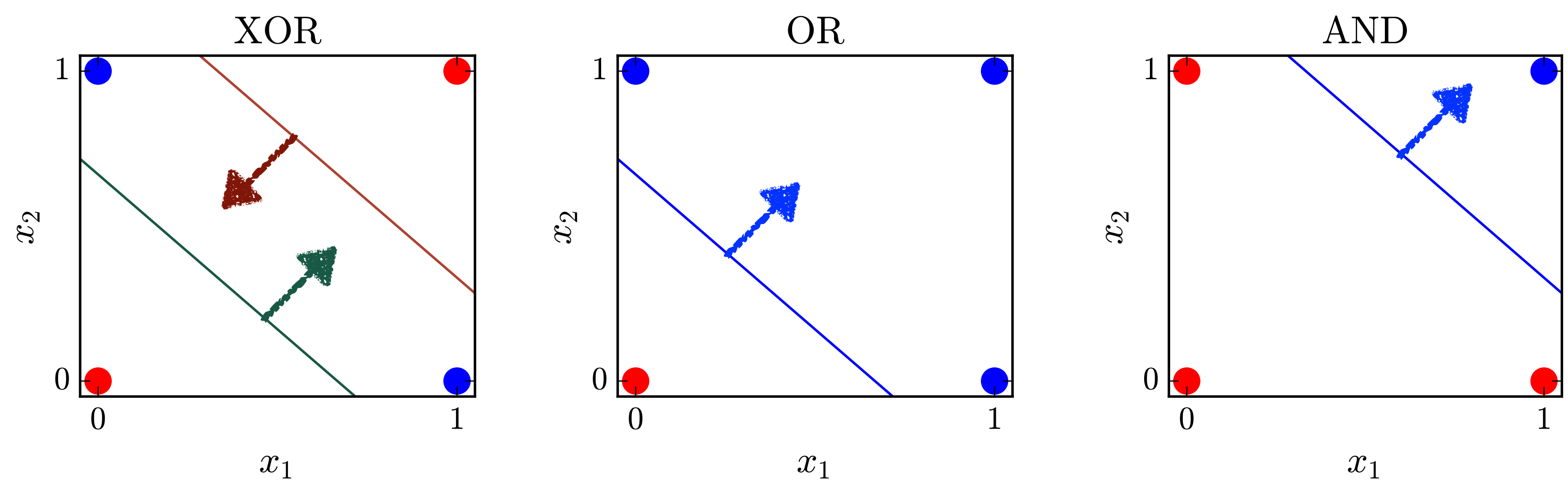


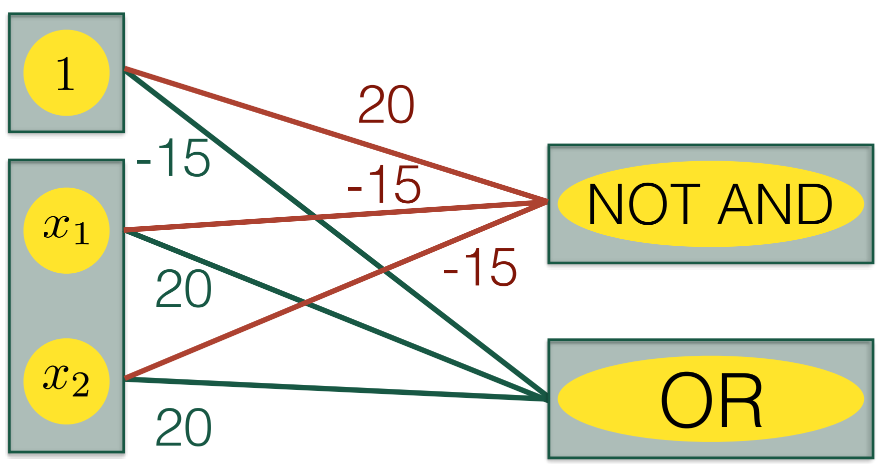
slide credit: Bryan Ostdiek

A Simple Example

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



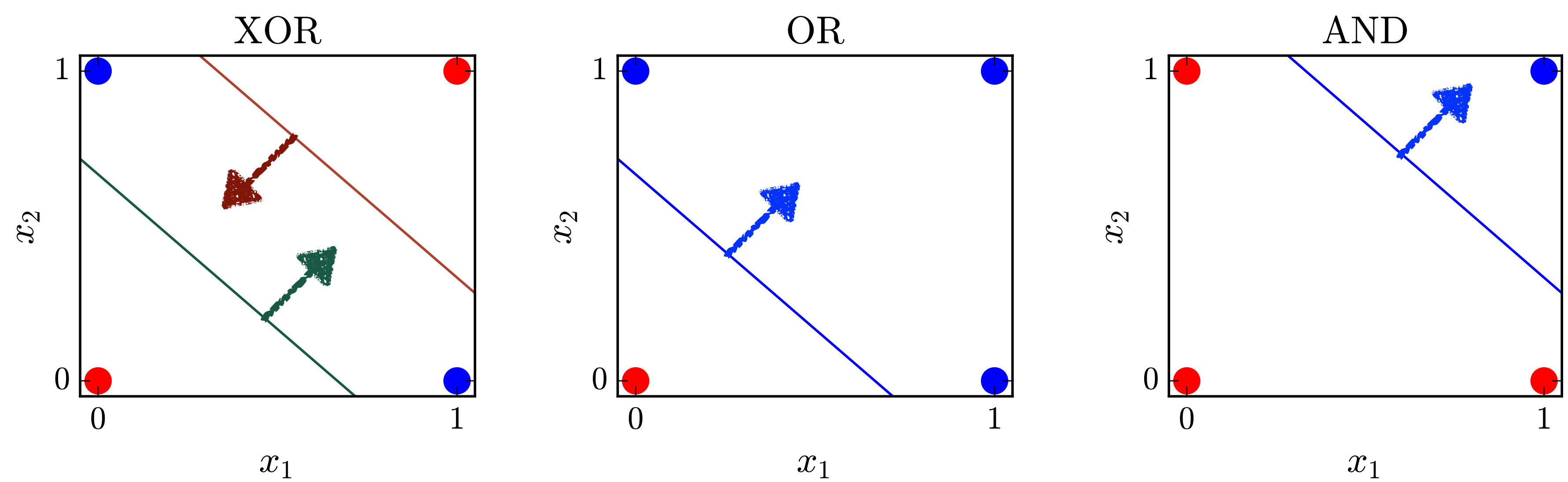


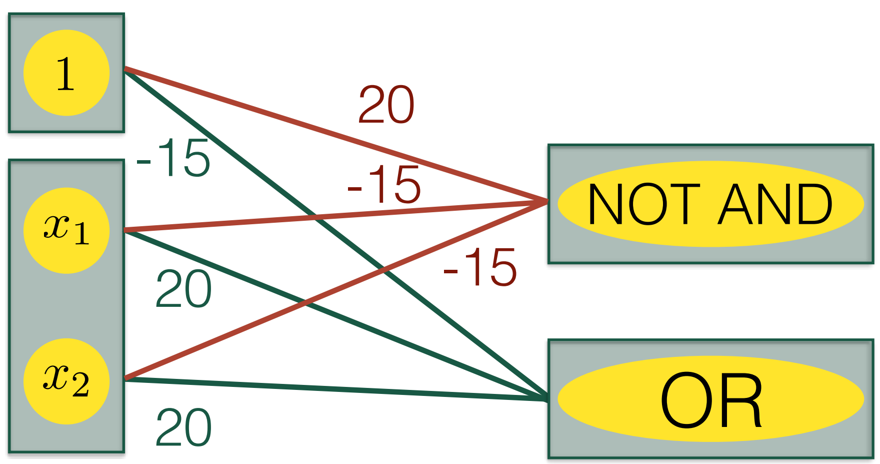
slide credit: Bryan Ostdiek

A Simple Example

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





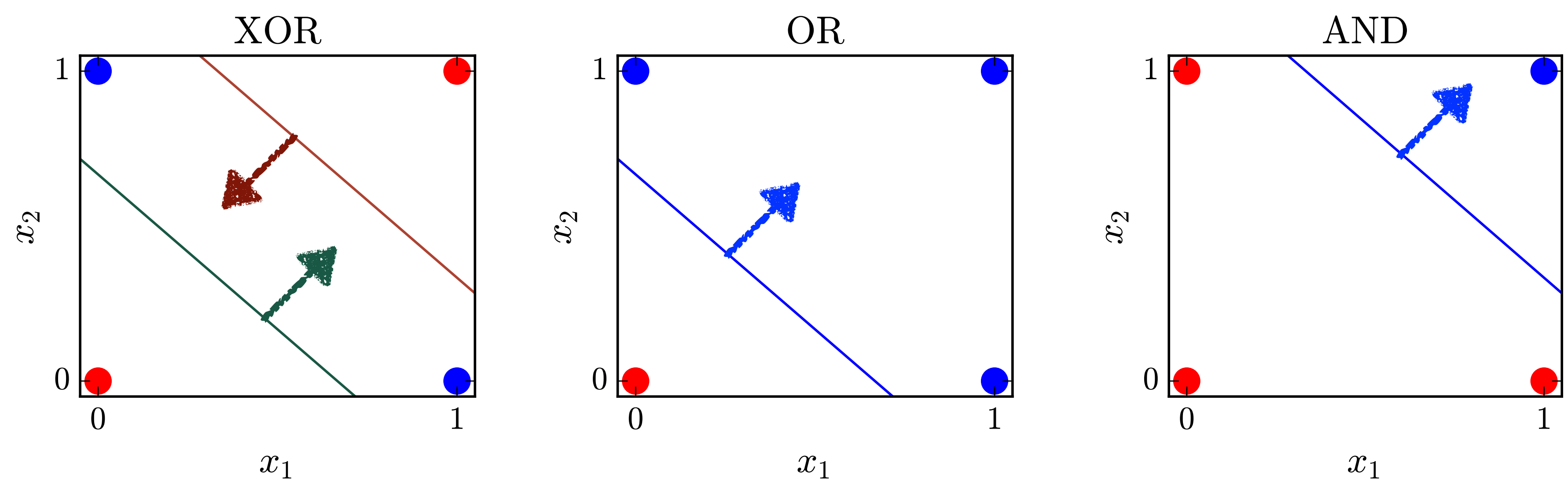
x_1	x_2	OR	NOT AND	XOR
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

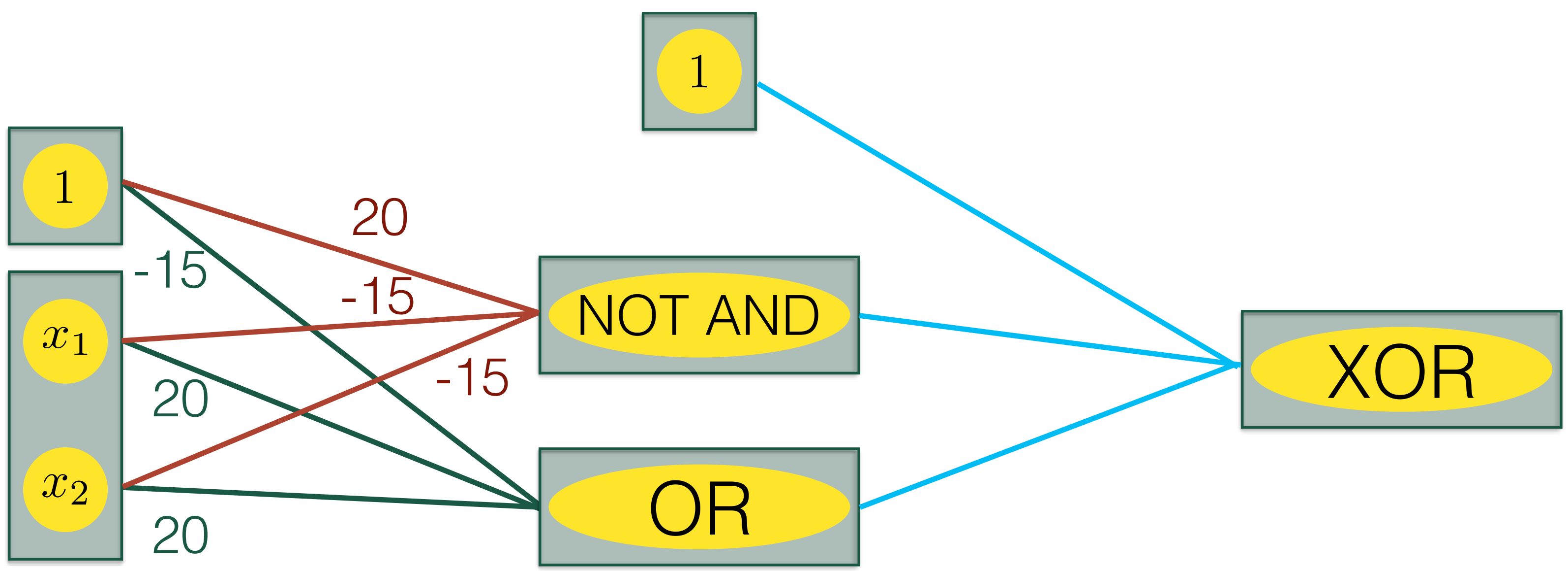
slide credit: Bryan Ostdiek

A Simple Example

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





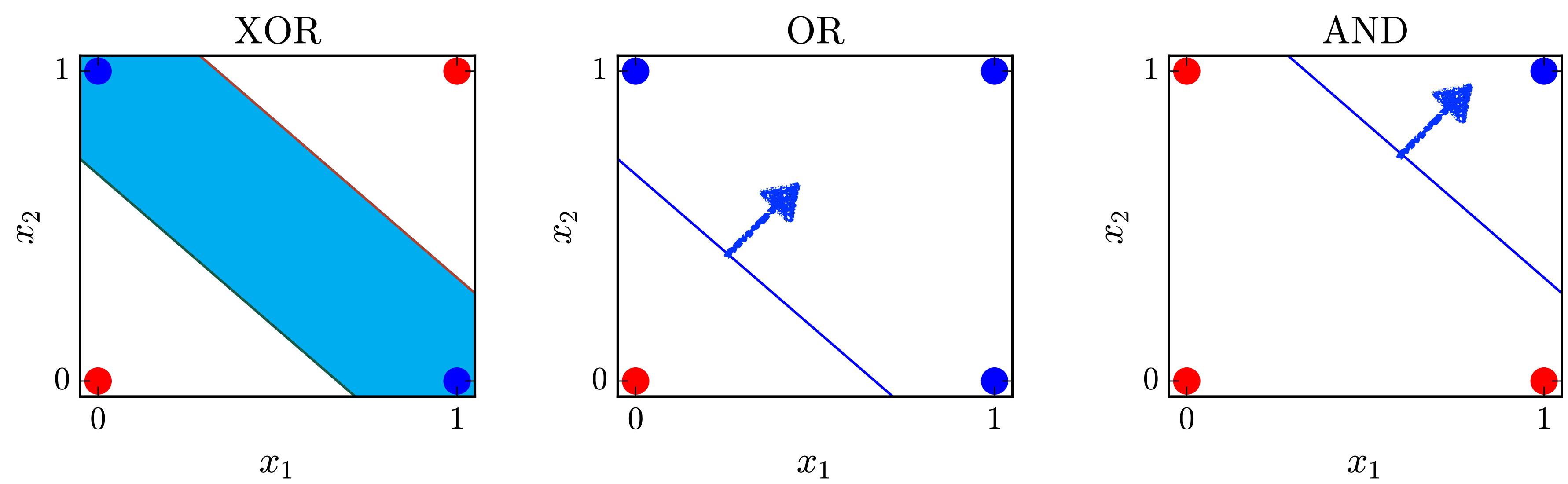
x_1	x_2	OR	NOT AND	XOR
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

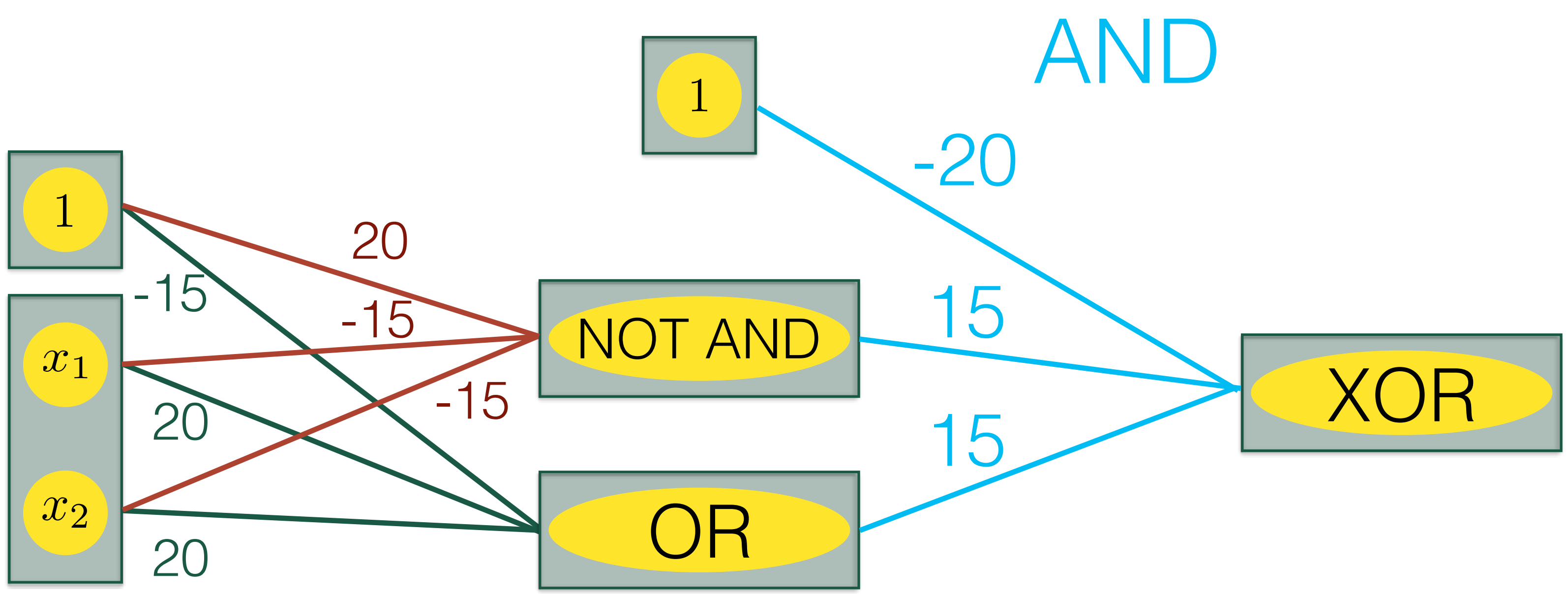
slide credit: Bryan Ostdiek

A Simple Example

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





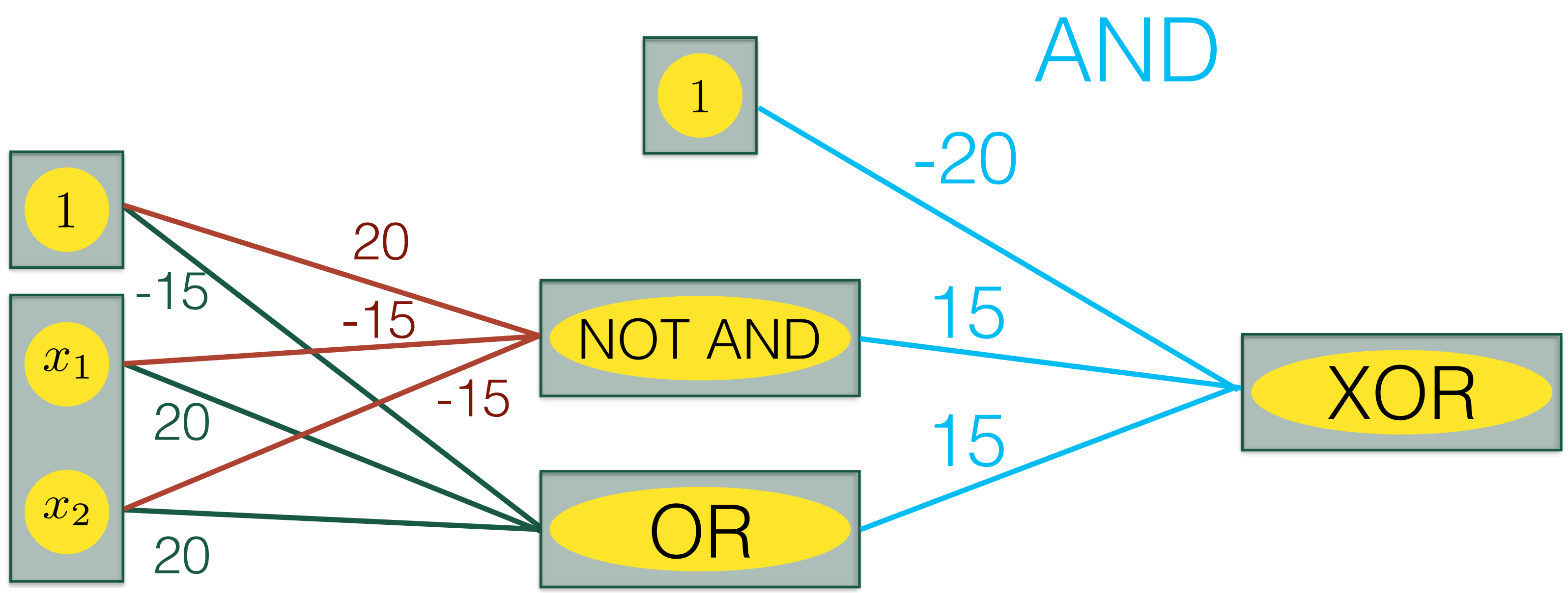
x_1	x_2	OR	NOT AND	XOR
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

slide credit: Bryan Ostdiek

A Simple Example

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

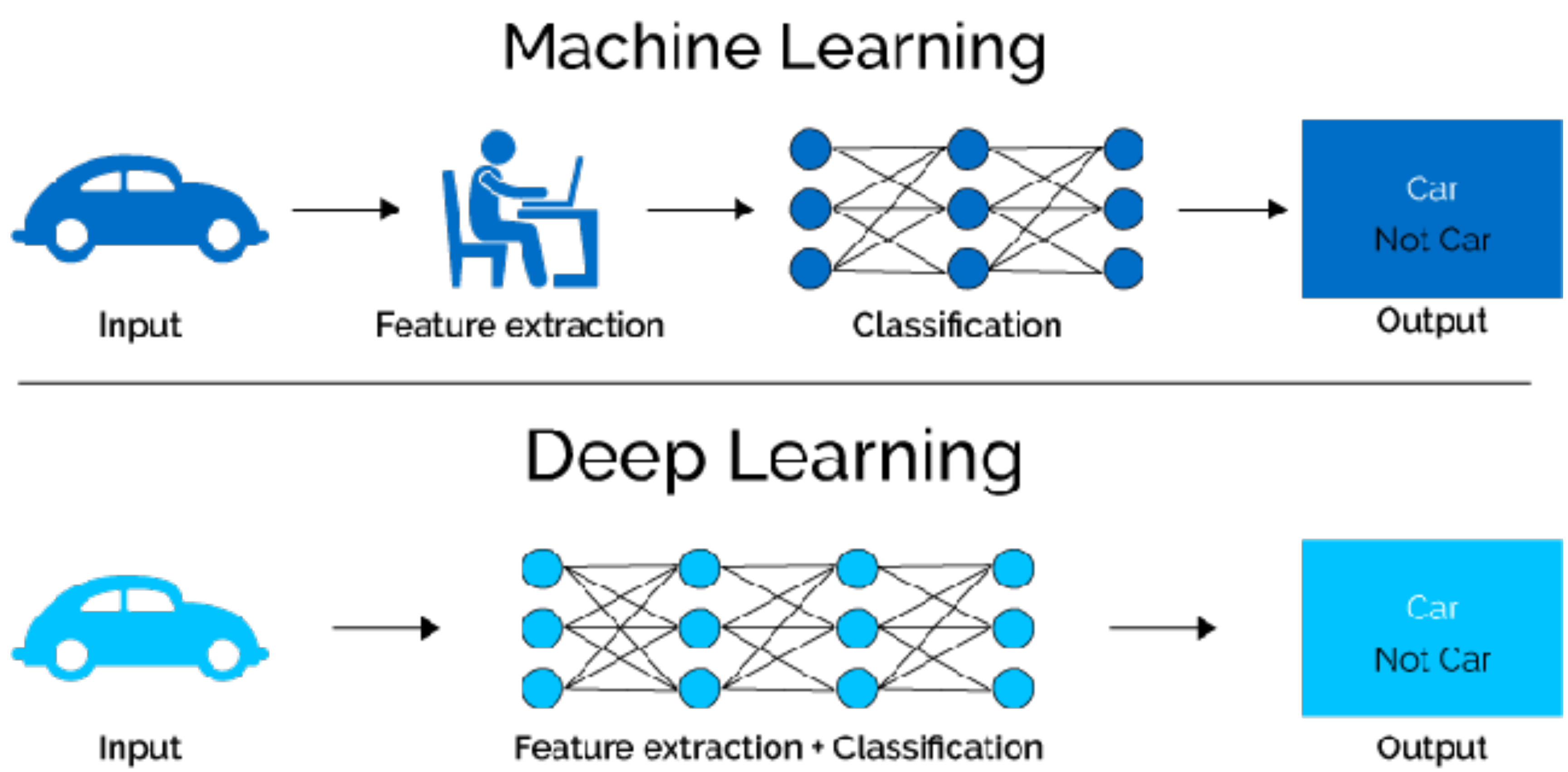


Simple example showing that neural network can access 'high-level' functions

To learn weights, need large training set and CPU time

slide credit: Bryan Ostdiek

Why (deep) neural networks?



From towardsdatascience.com

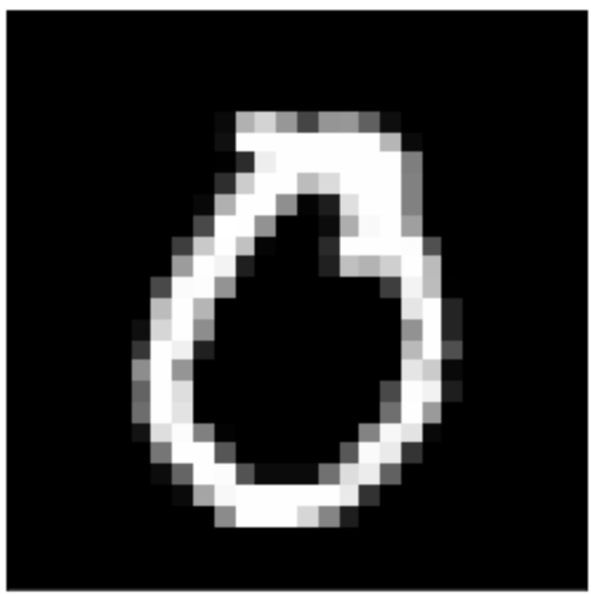
Universal function approximation means that deep NNs can learn high-level concepts from low-level, high-dimensional inputs

“Automated feature engineering”

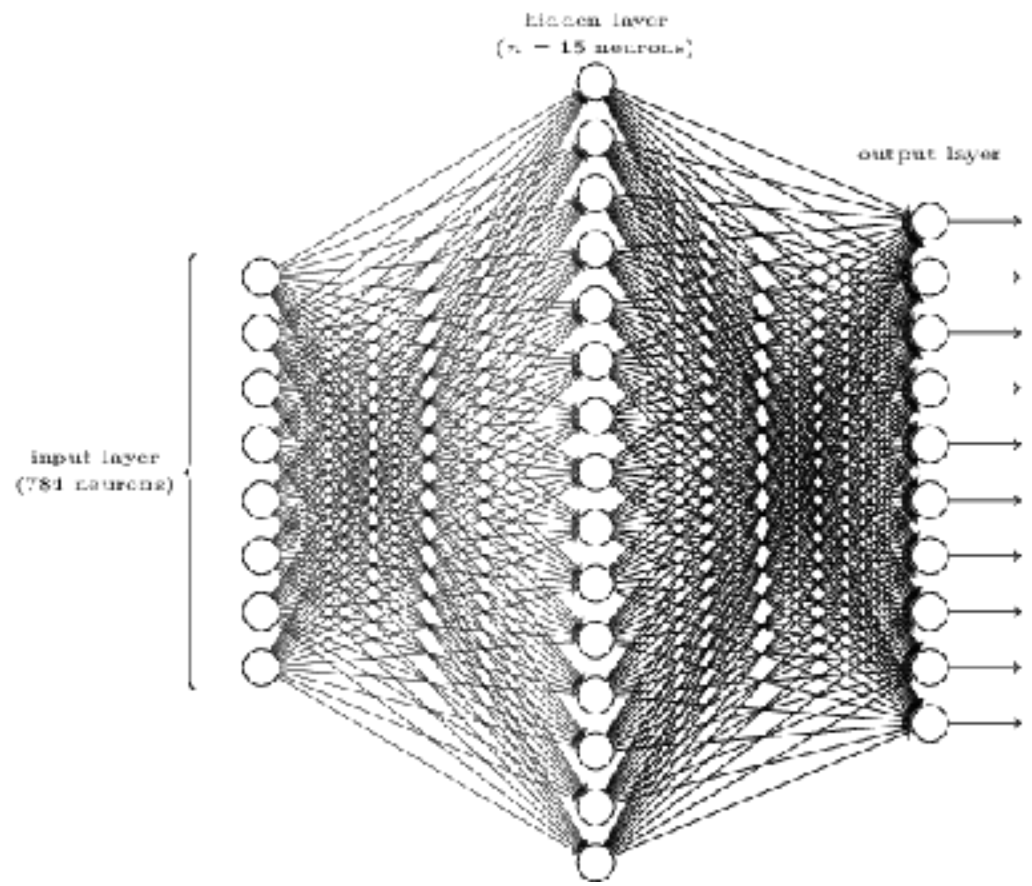
“End-to-end learning”

Example of end-to-end learning:

MNIST in more detail



x



$$\begin{pmatrix} P_0(x;\theta) \\ P_1(x;\theta) \\ \vdots \\ P_9(x;\theta) \end{pmatrix}$$

$$\sum_i P_i(x;\theta) = 1$$

Output: probability it's a 0, 1, …,9

Example of end-to-end learning:

MNIST in more detail

Using Keras framework…See Felipe's tutorial for more info!

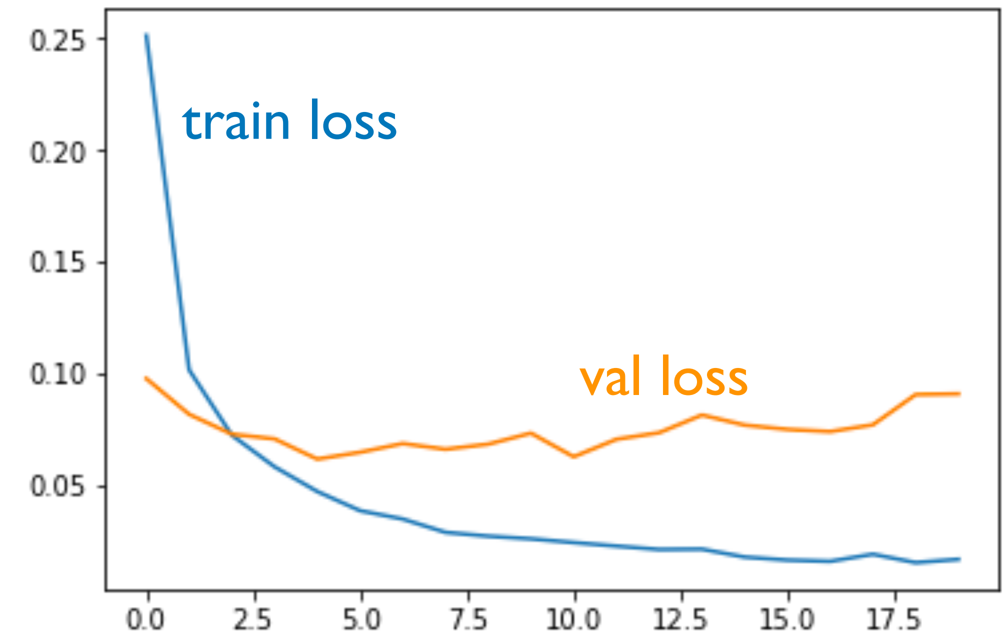
```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

Softmax activation: $P_i(x) = \frac{e^{x_i}}{\sum_i e^{x_i}}$

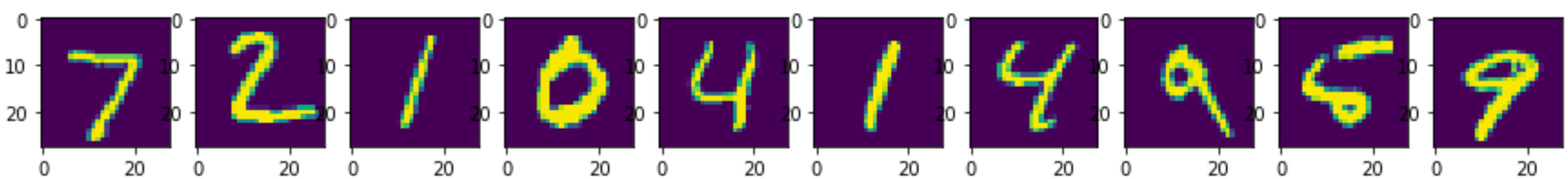
Example of end-to-end learning:

MNIST in more detail

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

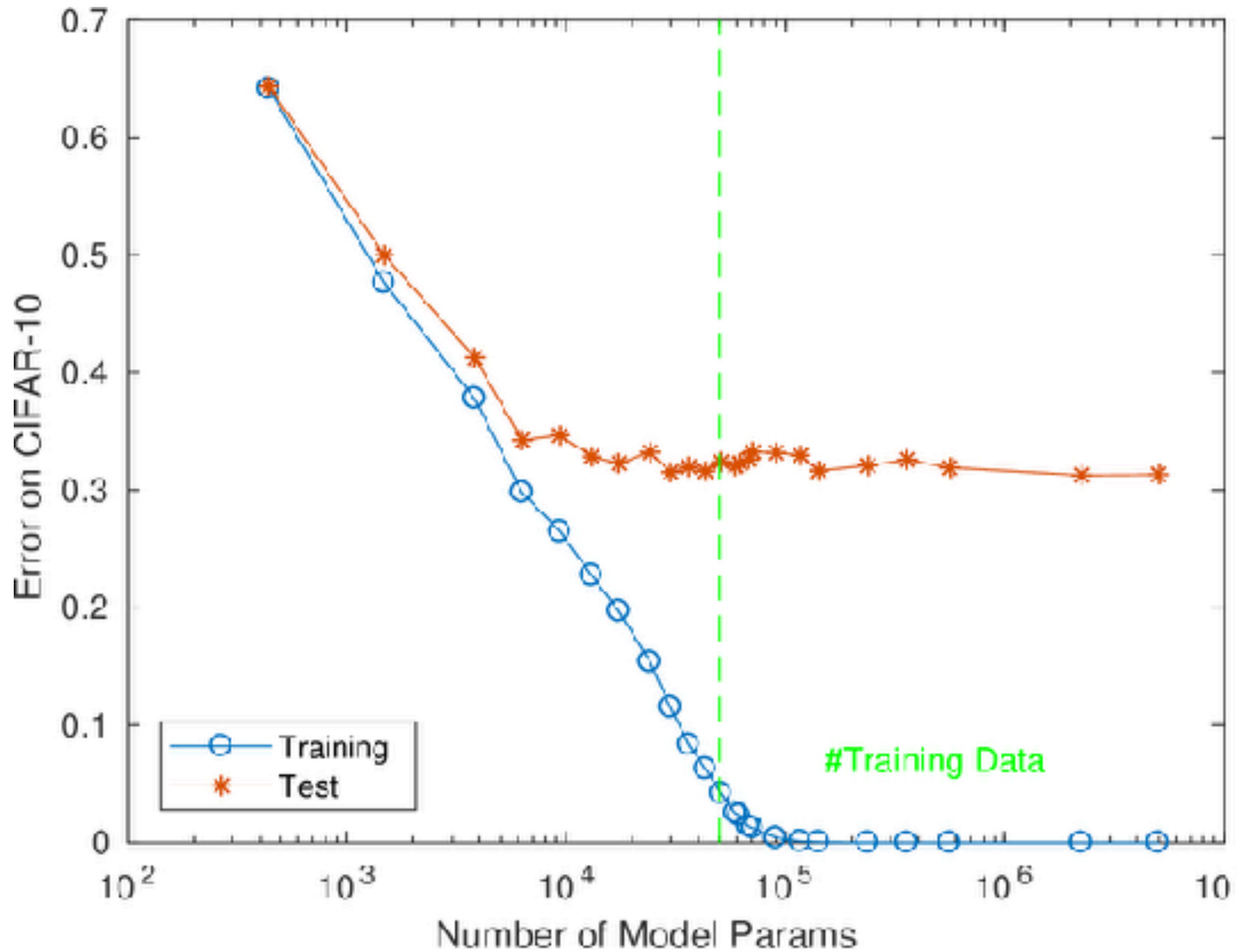


`Test accuracy: 0.9831`



prediction: 7 2 1 0 4 1 4 9 6 9

Why (deep) neural networks?

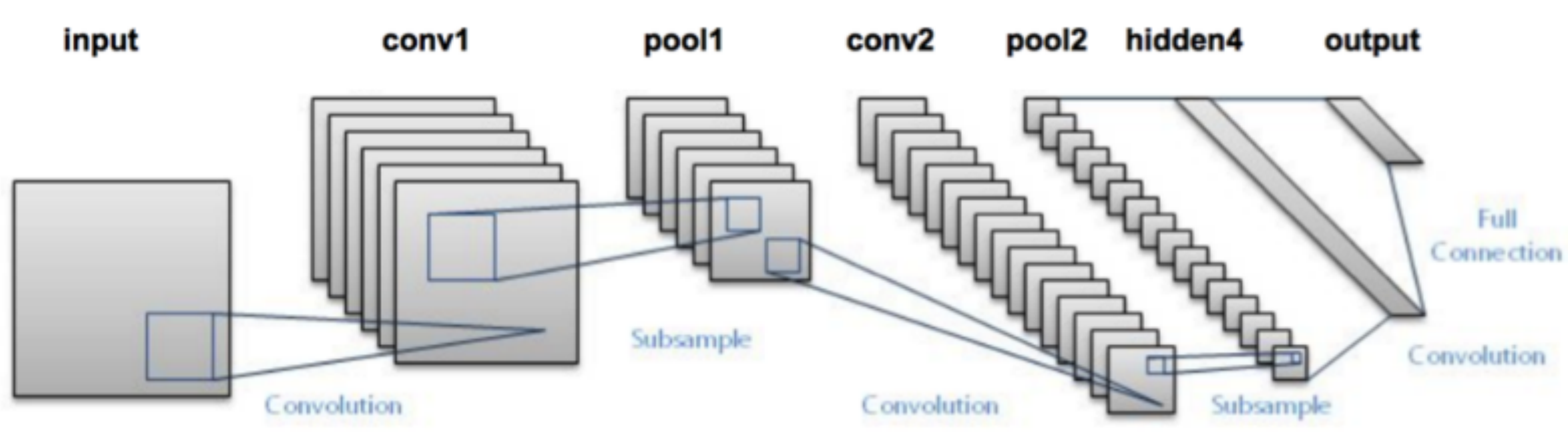


Deep neural networks (trained with SGD) are also unreasonably robust against overfitting. People still don't understand how/why…

"Generalization puzzle"

More examples of deep neural networks

Convolutional neural network (CNN)



Principal neural network architecture for image recognition.
Invented in 1998 (LeCun, Bottou, Bengio, Haffner)

Achieved 99% accuracy on MNIST!

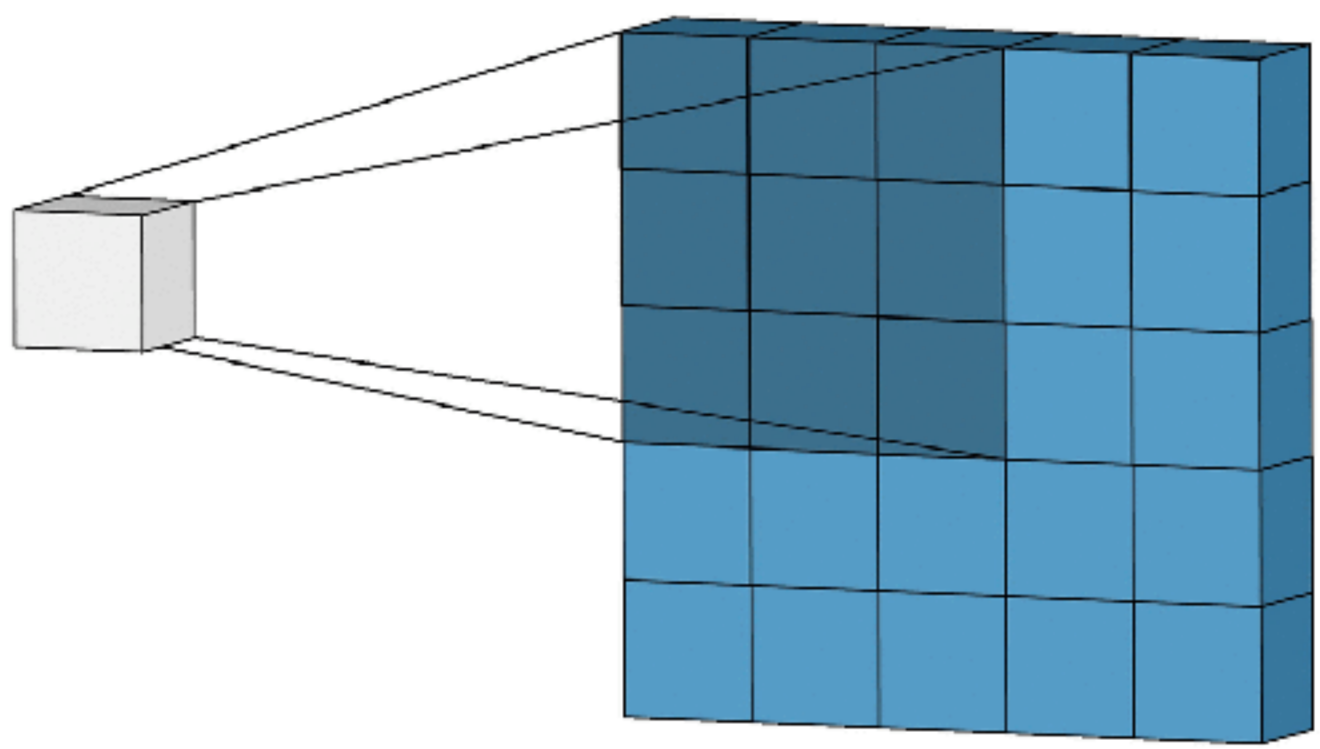
However, CNNs fell out of favor (until 2012) when they did not immediately generalize well to more complex image recognition tasks such as ImageNet.

Convolutional neural network (CNN)

Main idea: features in an image (edges, curves, corners,…eyes, noses,…) are the same no matter where they occur.

Goal: Want to find these features in a translationally invariant way.

Solution: Drag or convolve a "filter" across the image that selects out interesting features.

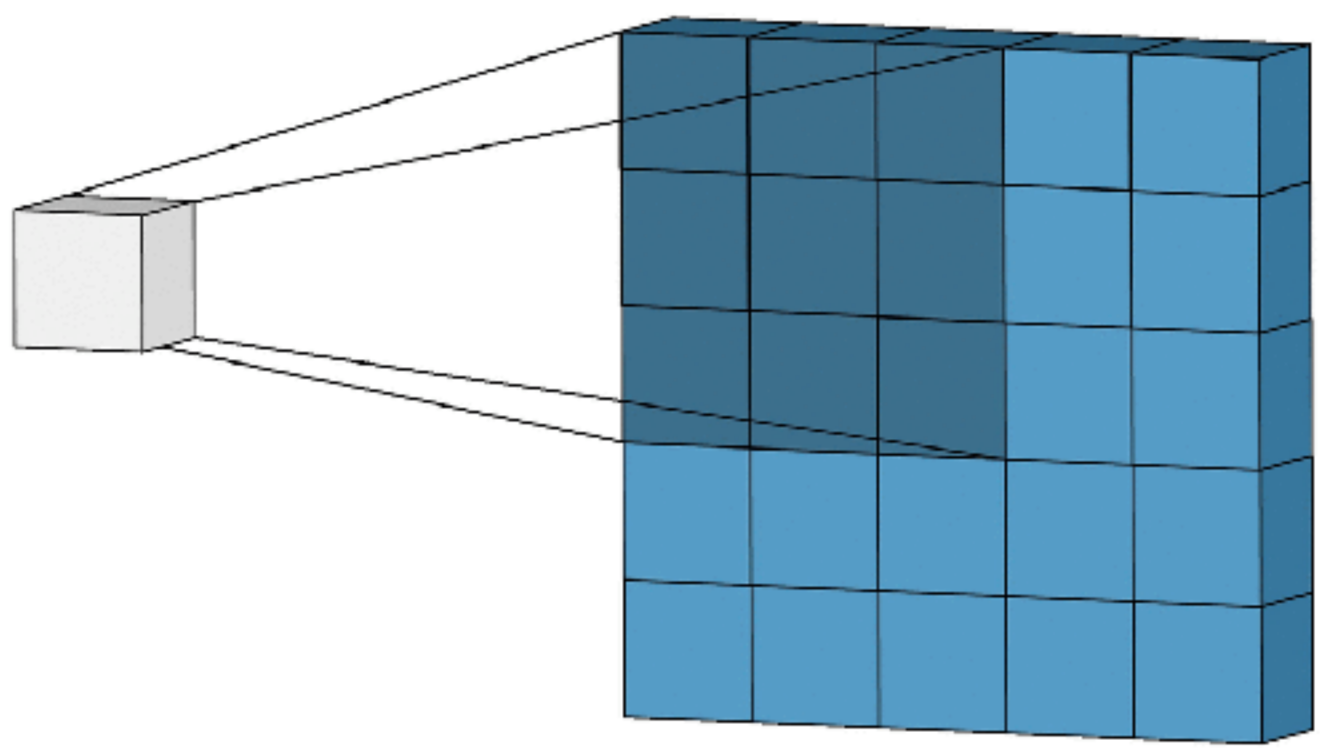


Convolutional neural network (CNN)

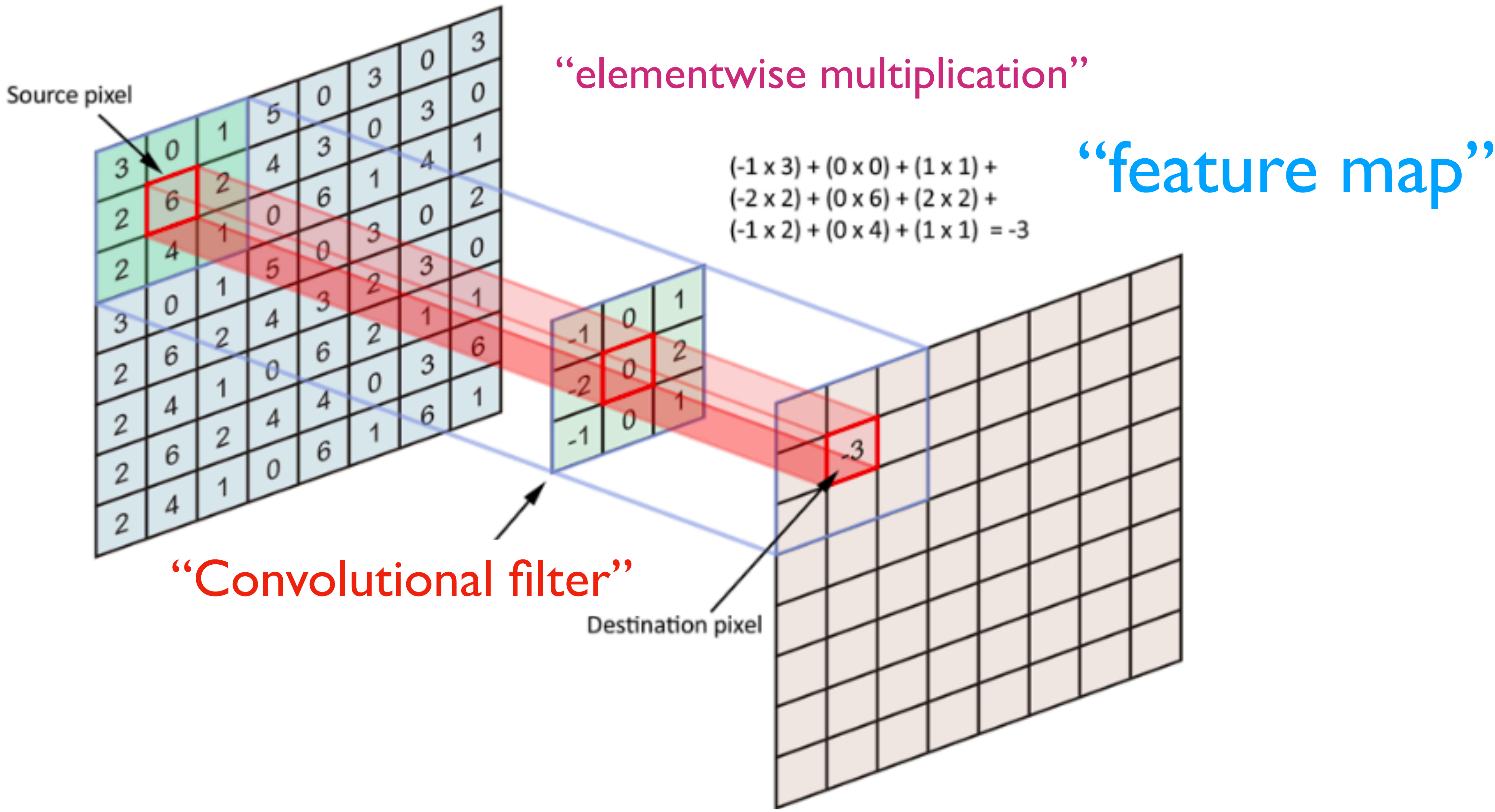
Main idea: features in an image (edges, curves, corners,…eyes, noses,…) are the same no matter where they occur.

Goal: Want to find these features in a translationally invariant way.

Solution: Drag or convolve a "filter" across the image that selects out interesting features.



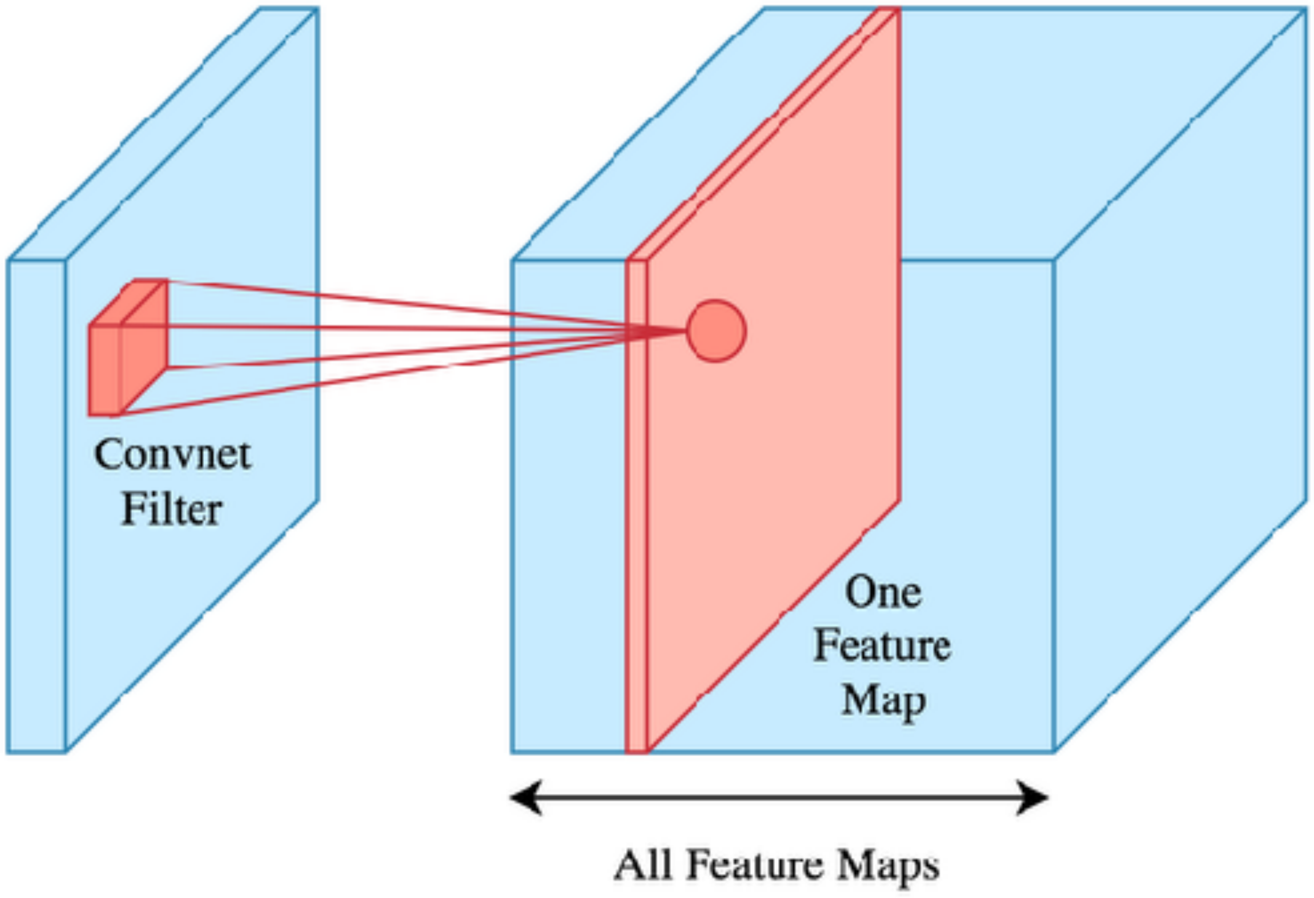
Convolutional neural network (CNN)



Finds features in the image in a translation invariant way

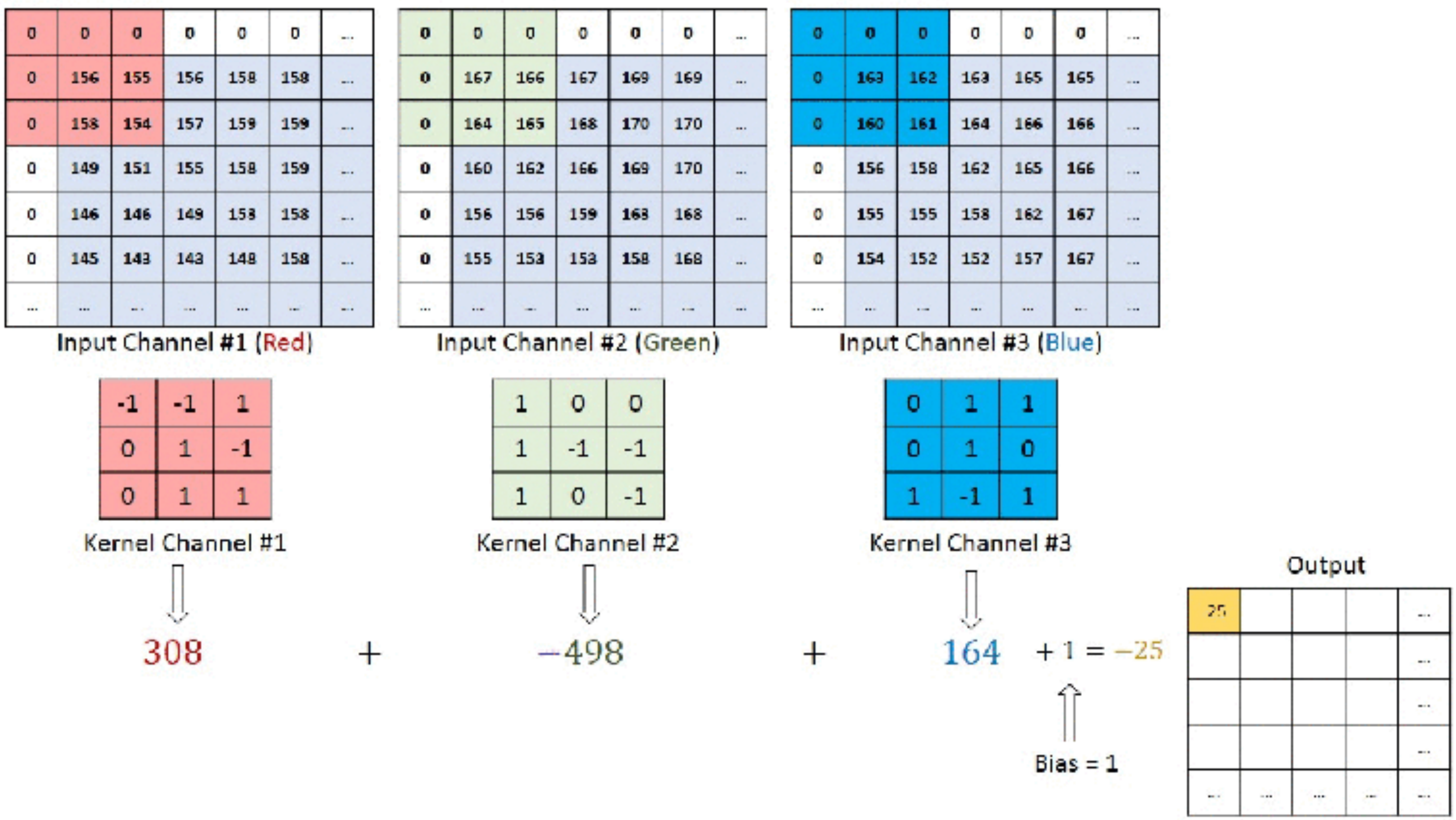
Convolutional neural network (CNN)

Can apply multiple filters to image to produce a stack of feature maps



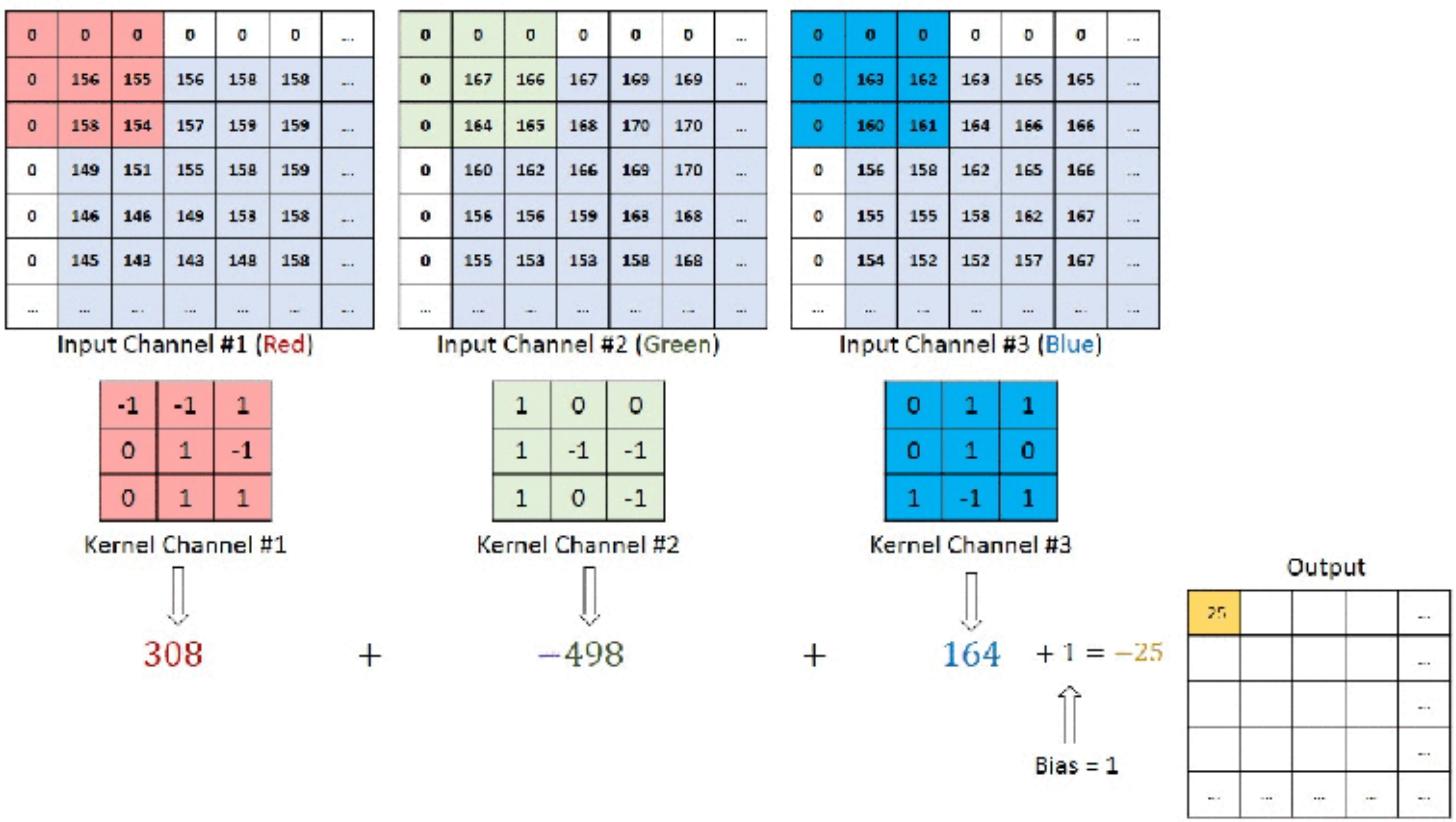
Convolutional neural network (CNN)

Dealing with multiple channels is straightforward — just enlarge filter to include channel dimension (3d filter) and perform element-wise multiplication along channel dimension as well.



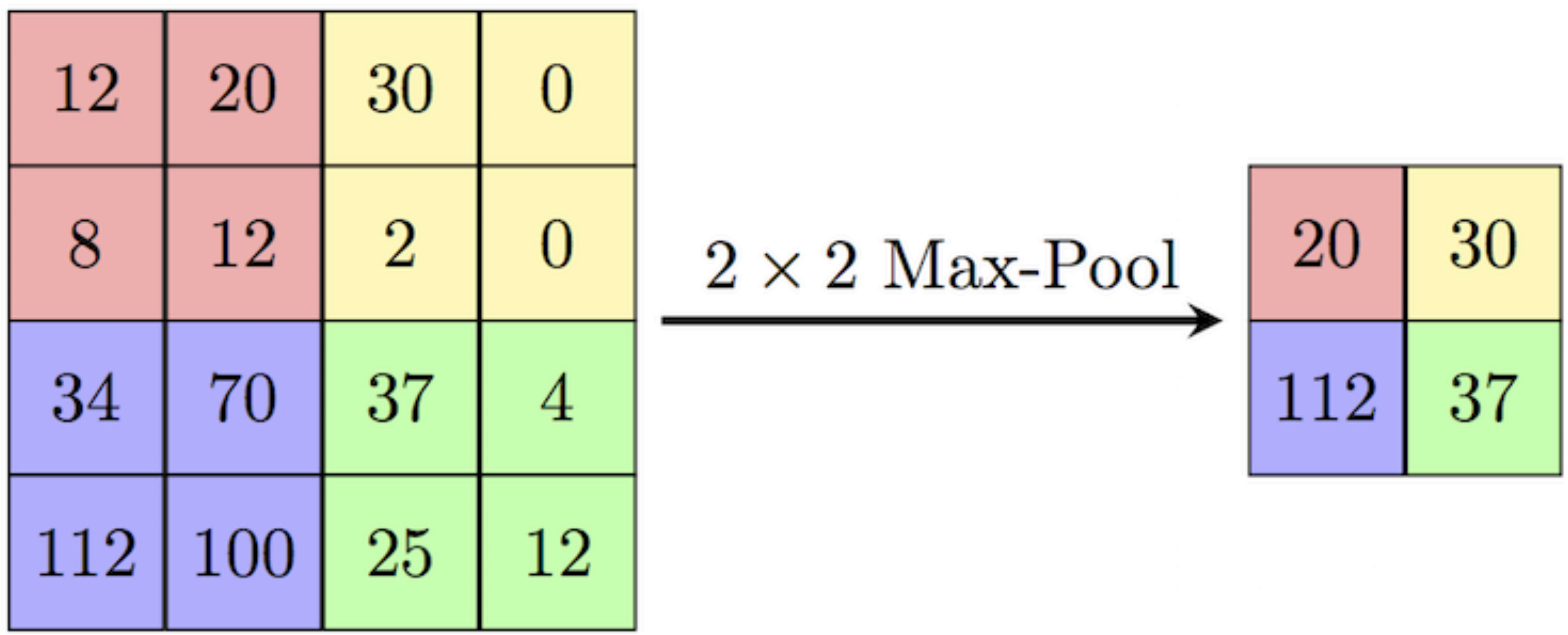
Convolutional neural network (CNN)

Dealing with multiple channels is straightforward — just enlarge filter to include channel dimension (3d filter) and perform element-wise multiplication along channel dimension as well.



Convolutional neural network (CNN)

“Max pooling”

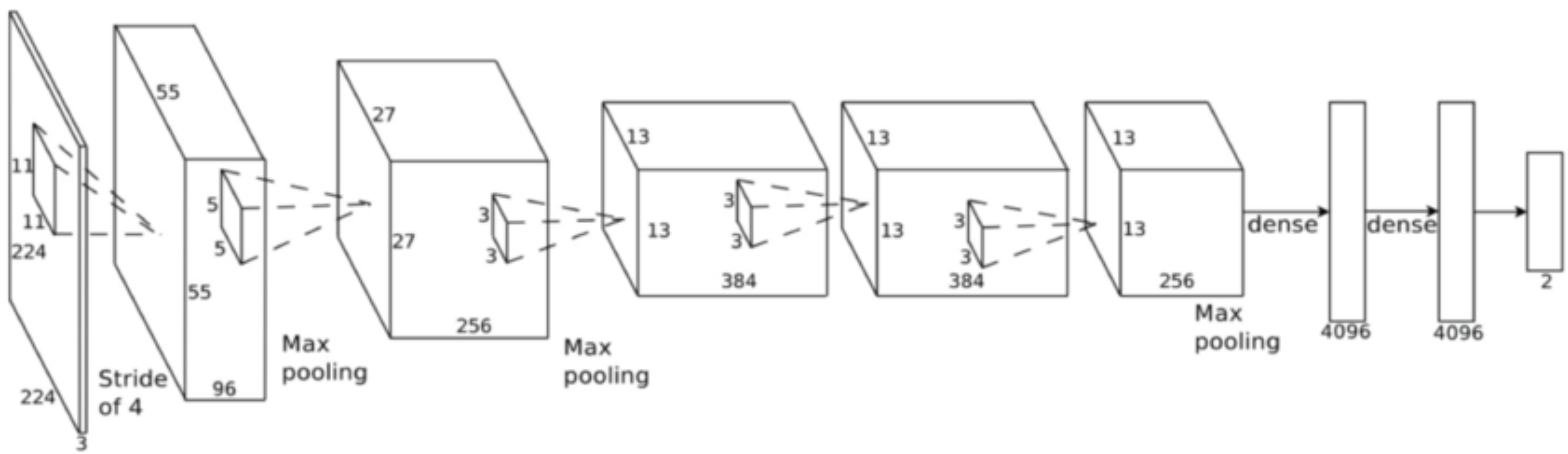


Reduces image size, reducing parameters and mitigating overfitting

Allows NN to find spatially larger, often higher-level features

Convolutional neural network (CNN)

Putting it all together

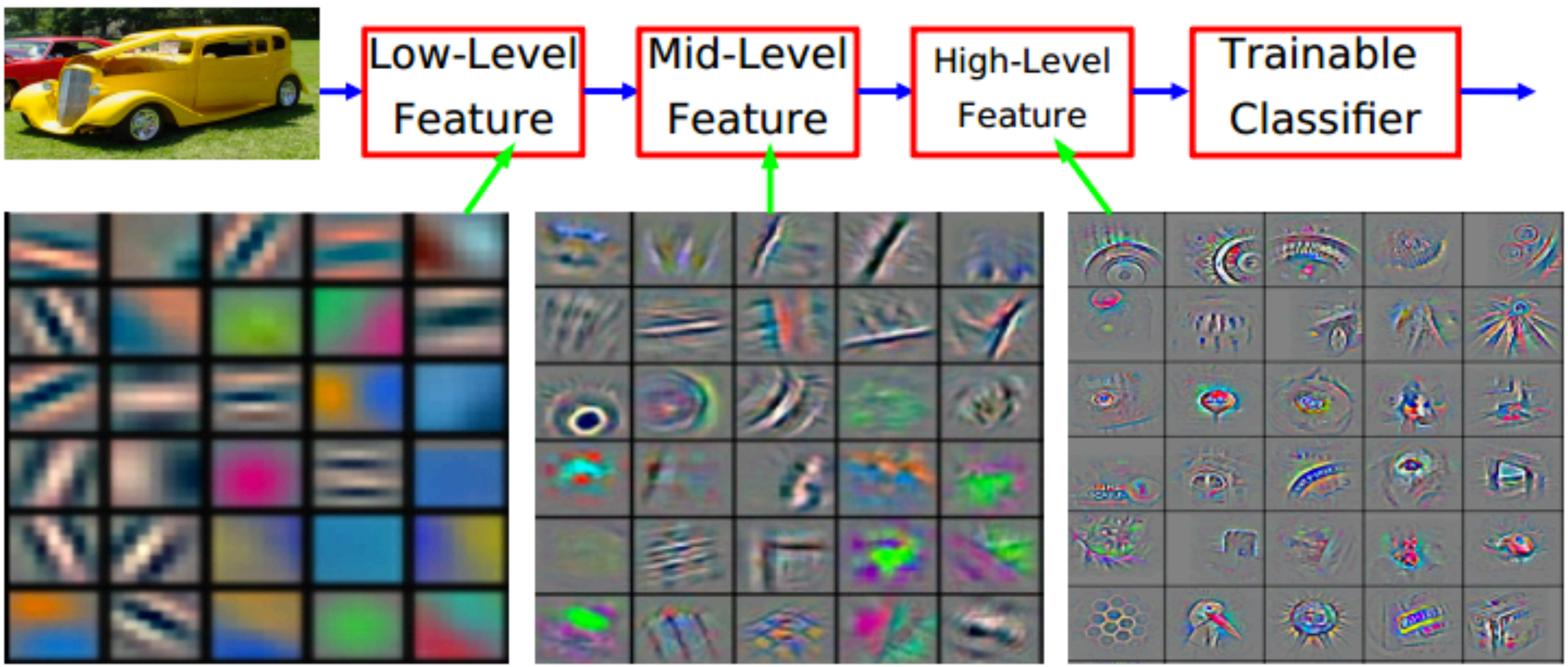


CNNs typically end with fully connected layers.

These are thought to extract the highest level information and to perform the actual classification task on the feature maps found by the convolutional layers.

Convolutional neural network (CNN)

What does the machine learn?



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Recurrent Neural Networks (RNNs)

Popular architecture for natural language processing (sentence completion, autocorrect, translation, speech recognition…)

Starting point: sequence of numbers $x_1, x_2, x_3, \ldots$

Suppose we want to predict the next number in the sequence?

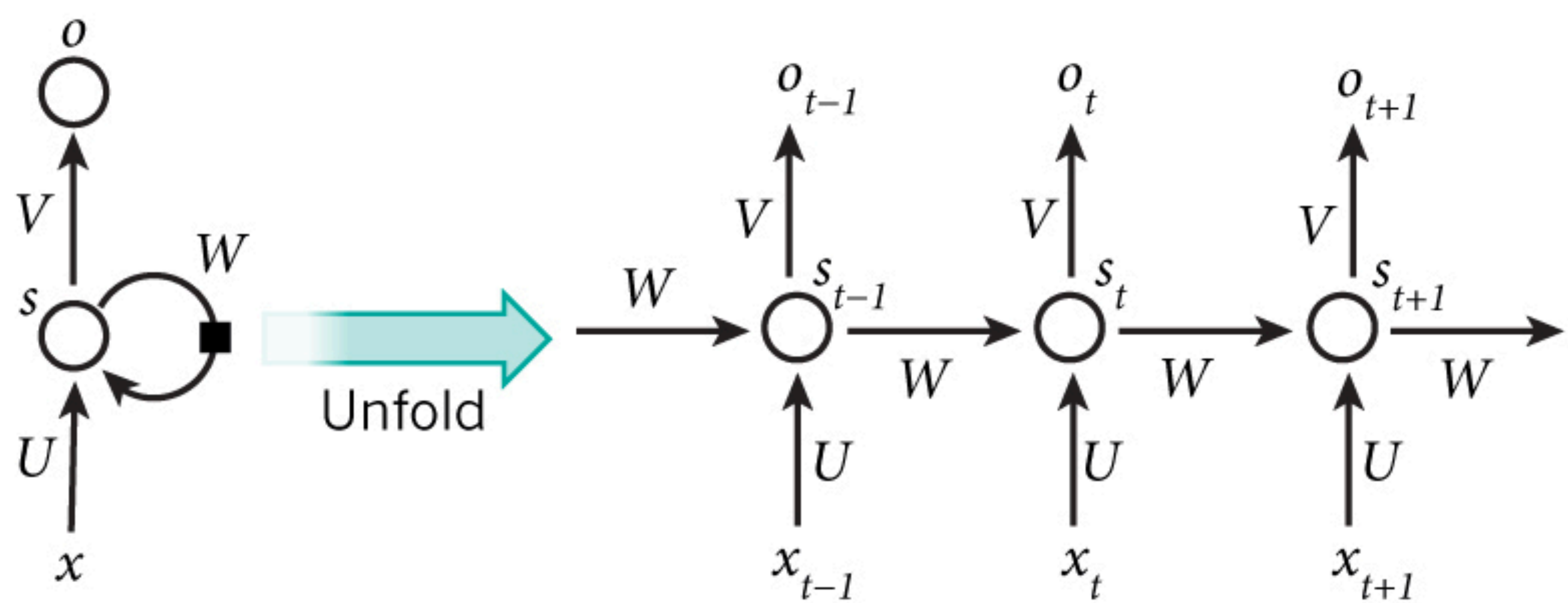
Idea of RNN:

- feed data sequentially to NN
- after each time step update *hidden state.* Hidden state encodes "memory" of sequence.
- Use hidden state to make predictions.

Recurrent Neural Networks (RNNs)

Basic RNN architecture

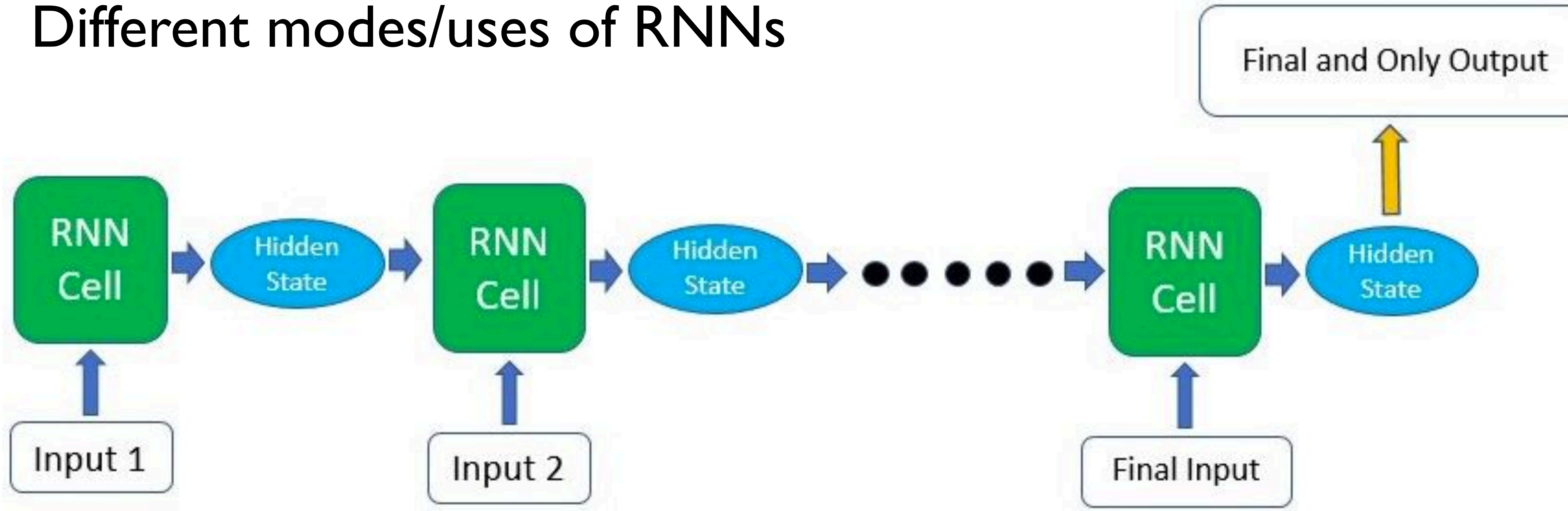
predictions, e.g. $o_t = \mathrm{softmax}(V s_t)$



$s_t = f(U x_t + W s_{t-1})$ hidden state after time step t

Recurrent Neural Networks (RNNs)

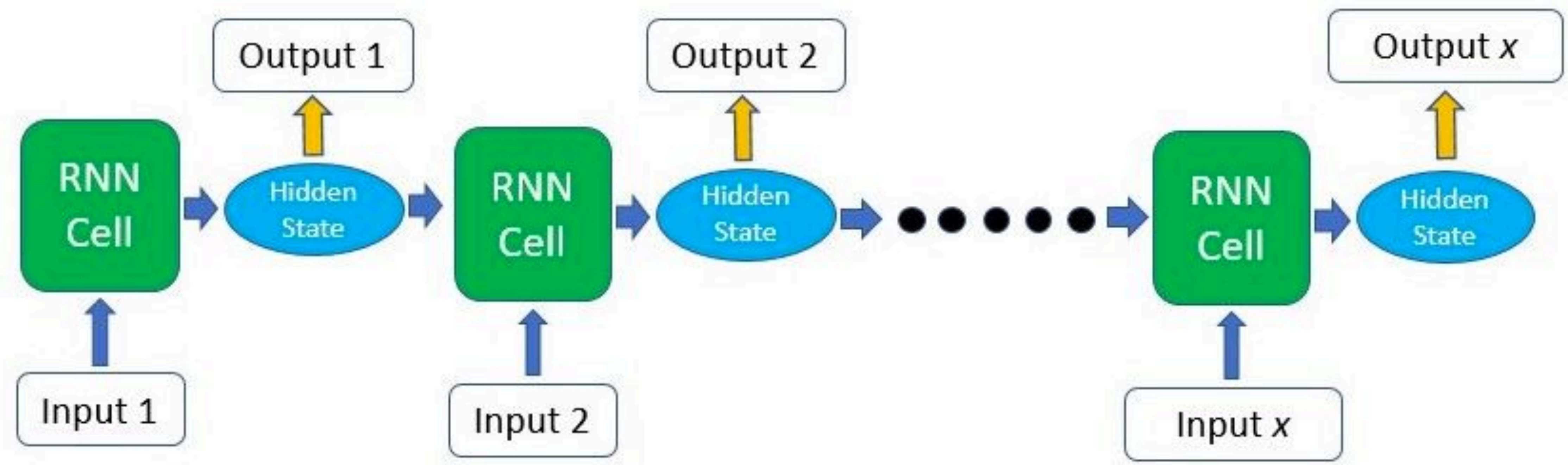
Different modes/uses of RNNs



sequence classification, regression

Recurrent Neural Networks (RNNs)

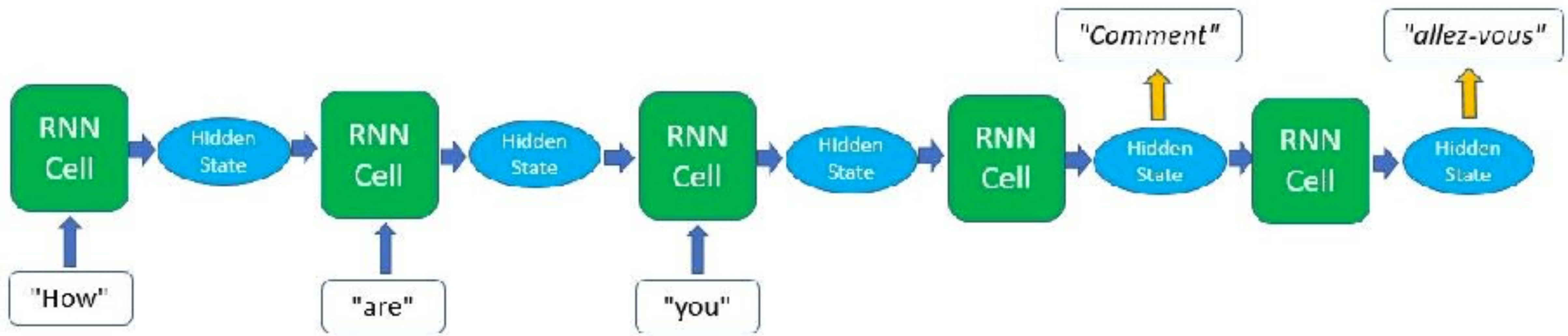
Different modes/uses of RNNs



real-time prediction

Recurrent Neural Networks (RNNs)

Different modes/uses of RNNs

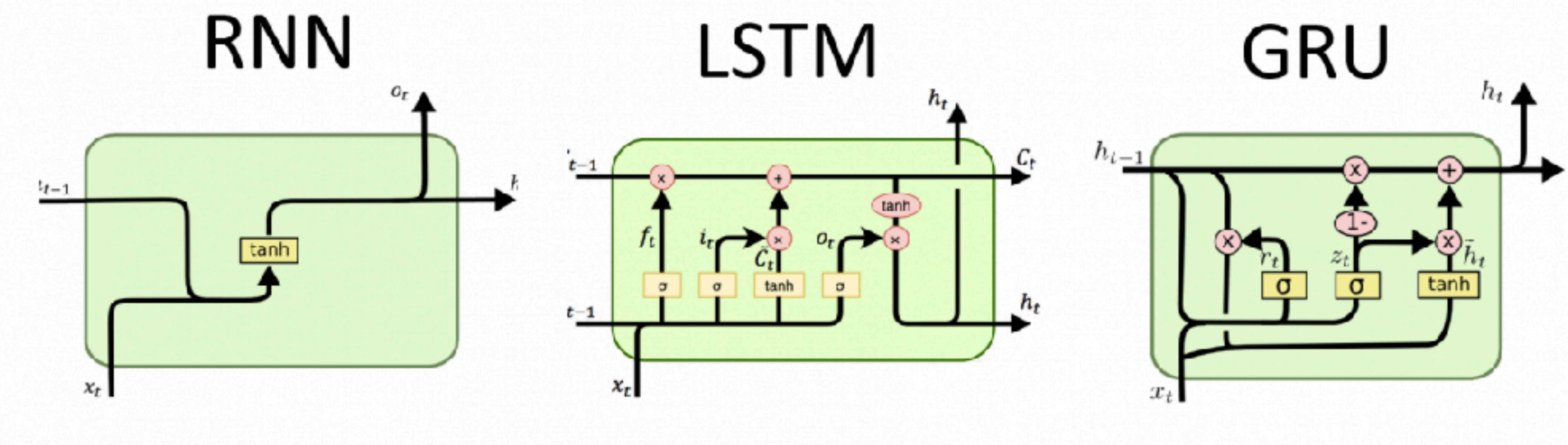


sequence-to-sequence

Recurrent Neural Networks (RNNs)

Simple RNNs applied to long sequences have a very serious exploding/ vanishing gradient problem.

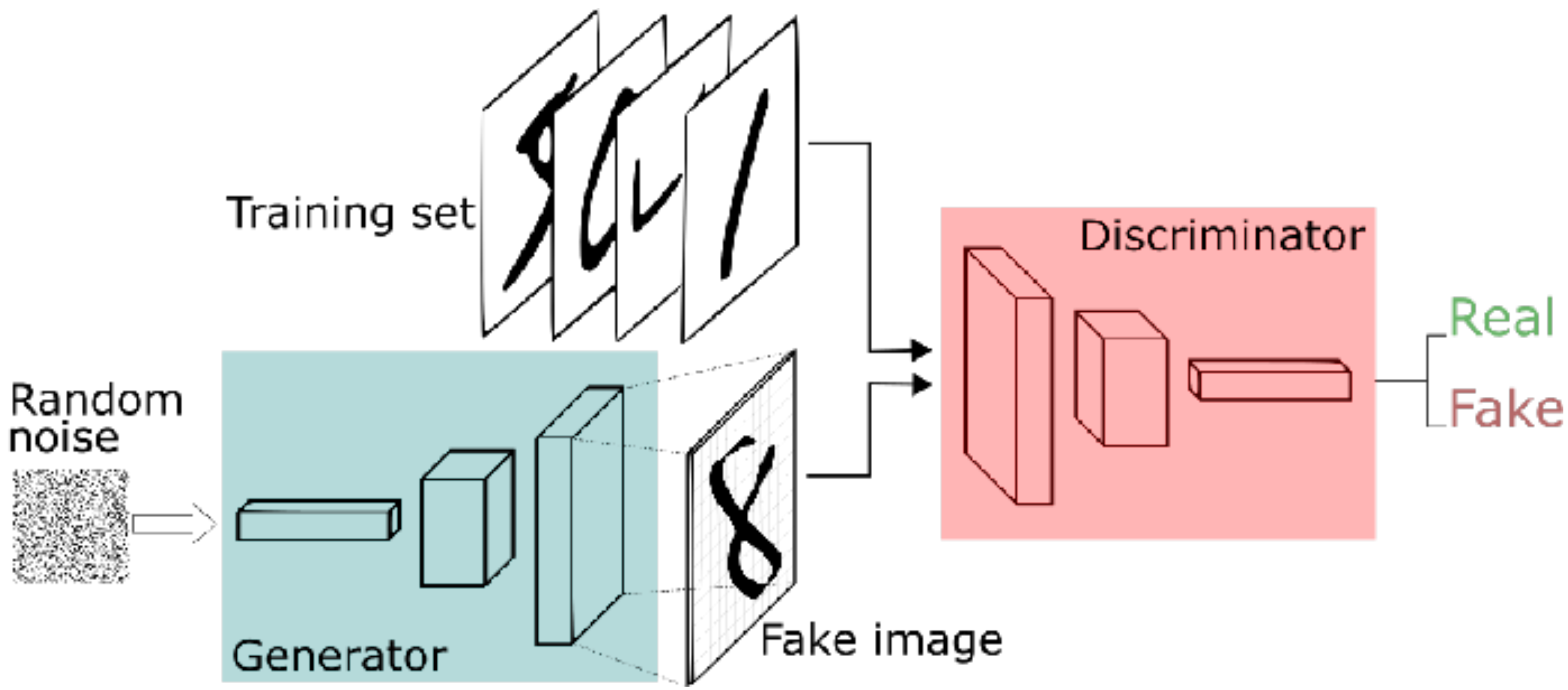
Prevents them from "remembering" relevant information from earlier in the sequence.



"Long-short term memory" and "Gated recurrent units" are two methods commonly used to solve the gradient problem and improve performance.

Generative Adversarial Networks (GANs)

Breakthrough method in generative modeling and unsupervised learning (Goodfellow et al. 2014)



Idea: train two neural networks: a "generator" that attempts to generate fake, random images, and a "discriminator" that tries to tell them apart from a database of real images.

Generative Adversarial Networks (GANs)

$$L_{GAN} = \sum_{x \in \text{real}} \log D(x) + \sum_{z \in \text{random}} \log(1 - D(G(z)))$$

Training is performed "adversarially"

- Discriminator tries to minimize loss
- Generator tries to ***maximize*** loss
- Take turns training discriminator and generator to optimize fake image generator

Generative Adversarial Networks (GANs)

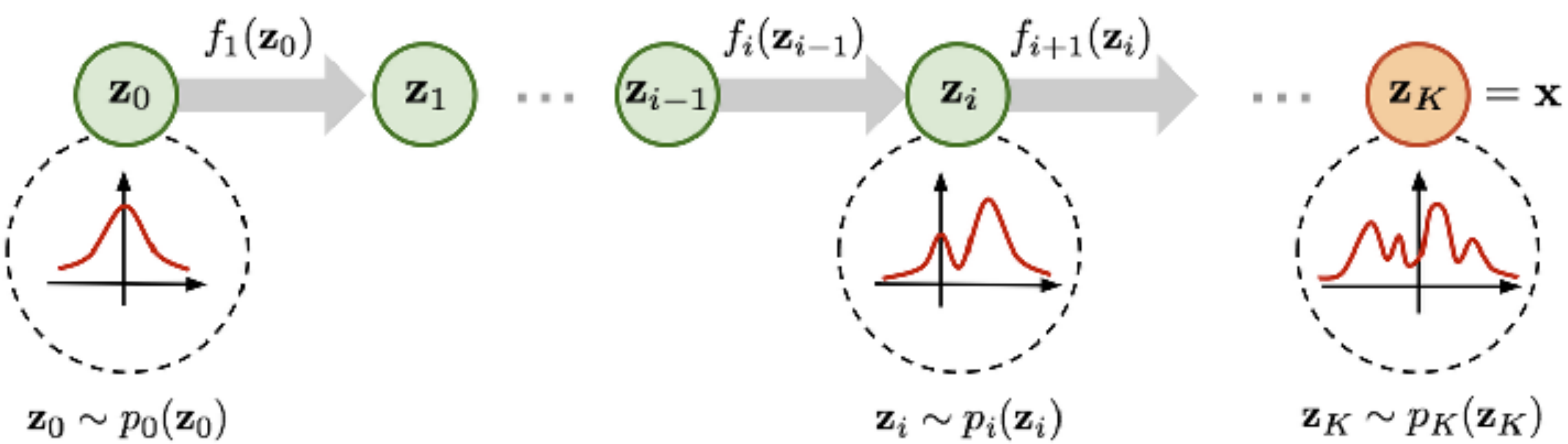
Real or fake?

Normalizing flows

Rezende & Mohamed 1505.05770

Recently a lot of excitement and progress in the problem of density estimation with neural networks.

Idea: map original distribution to normal distribution through series of invertible transformations.



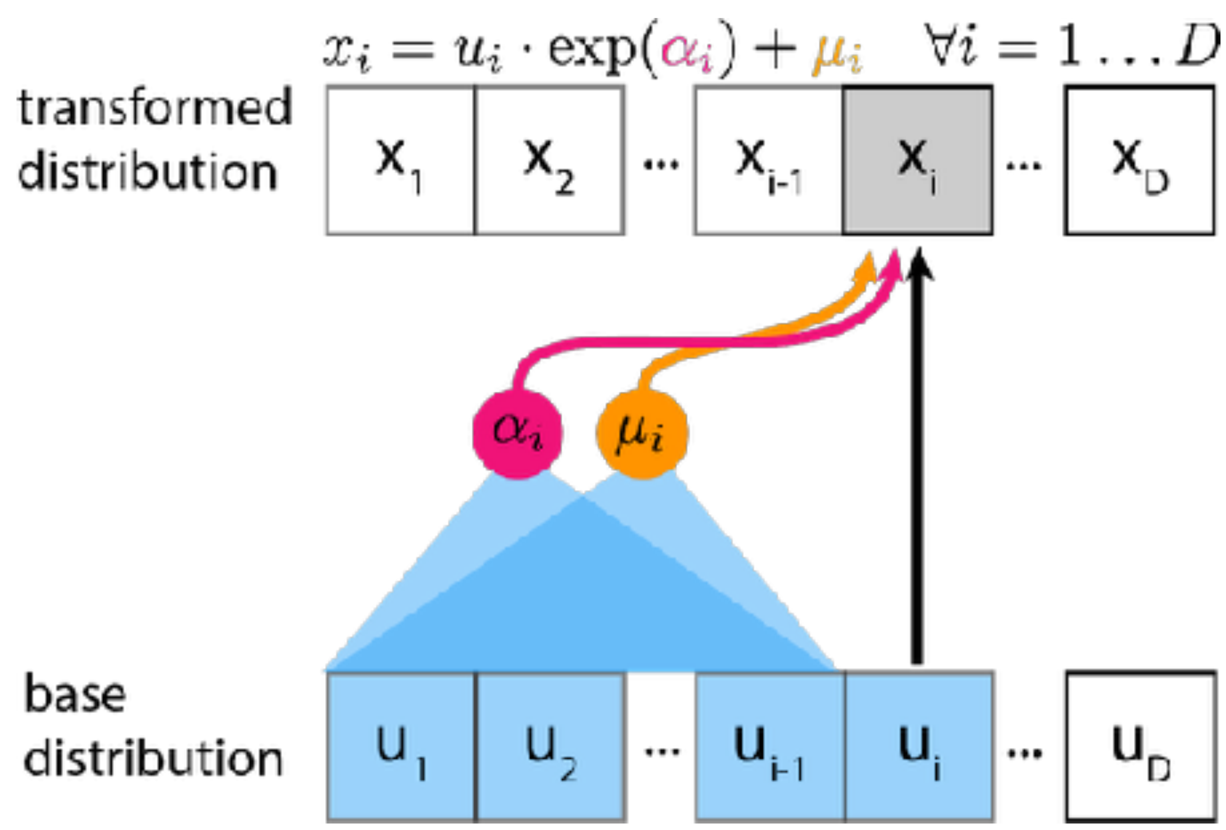
Examples: RealNVP, NICE, Glow, …

Autoregressive flows

$$p(x) = \prod_i p(x_i \mid x_{1:i-1})$$

Special type of normalizing flows. Learn probability density of each coordinate conditioned on previous coordinates.

Transformation upper triangular — automatically invertible. Allows for more expressive transformations.



Examples: MADE, MAF, IAF, NAF, PixelRNN, Wavenet, …