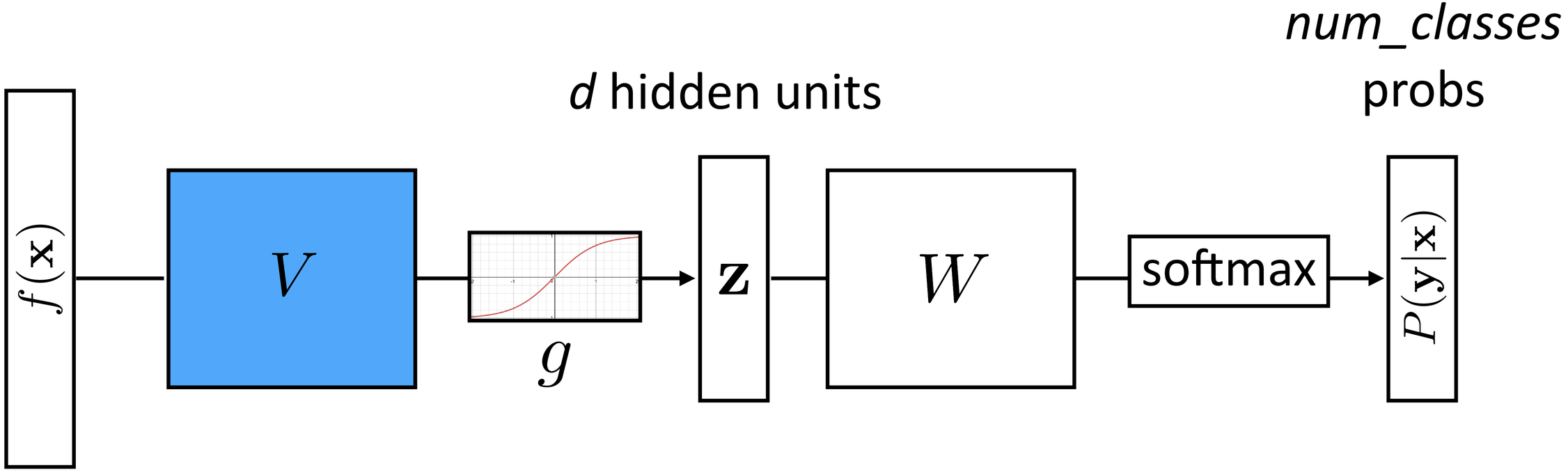
# DS-GA 1003: Machine Learning

Lecture 10: Structured Neural Networks

[Slides courtesy of: Dan Klein, Abigail See, Greg Durrett, Yejin Choi, John DeNero, Eric Wallace, Kevin Lin, Fei-Fei Li, Sergey Levine, Pieter Abbeel, and many others]

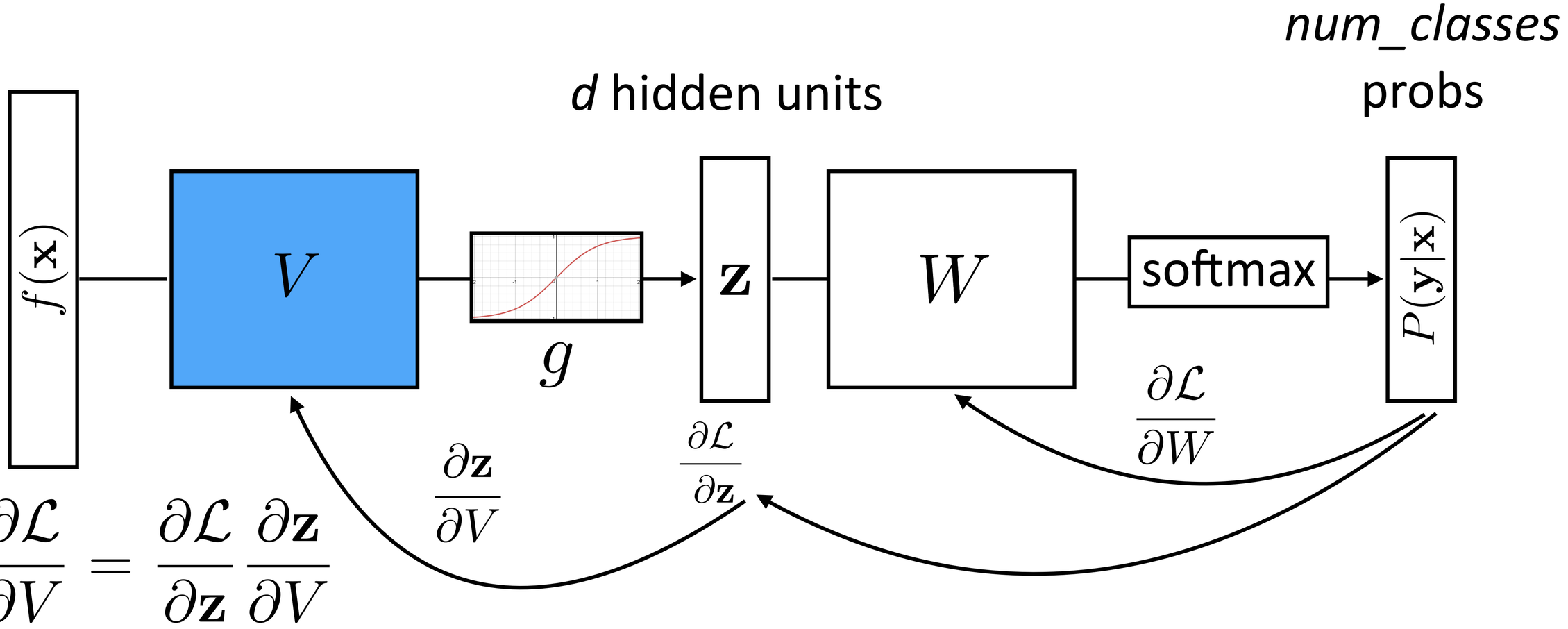
# Recap: Neural Networks

$$p(y \mid x) = \text{softmax}(Wg(Vf(x)))$$



# Recap: Neural Networks

$p(y \mid x) = \text{softmax}(Wg(Vf(x)))$



# Fully Connected Neural Networks

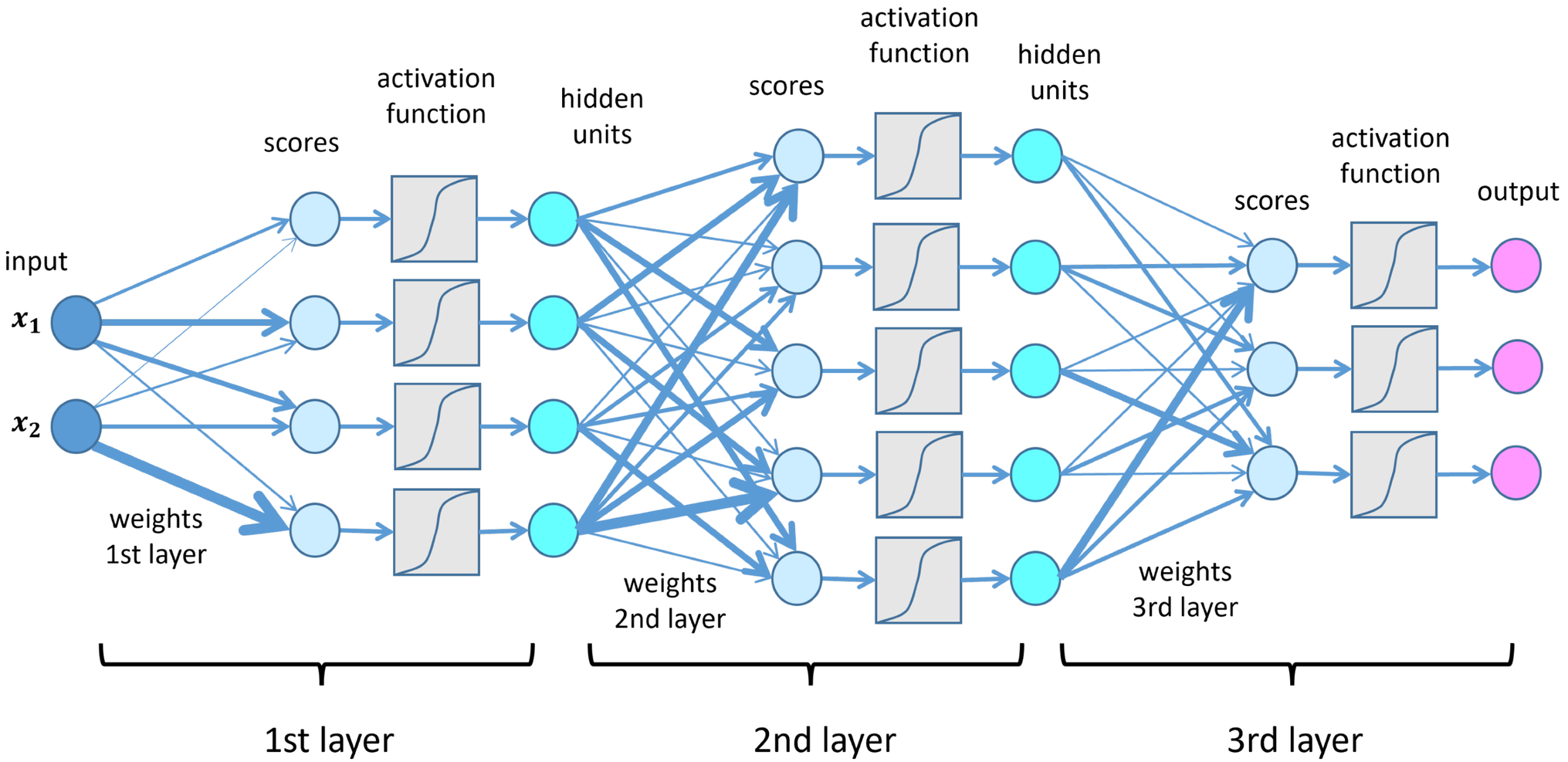



Image from Lamarr Institute

# Fully Connected Neural Networks

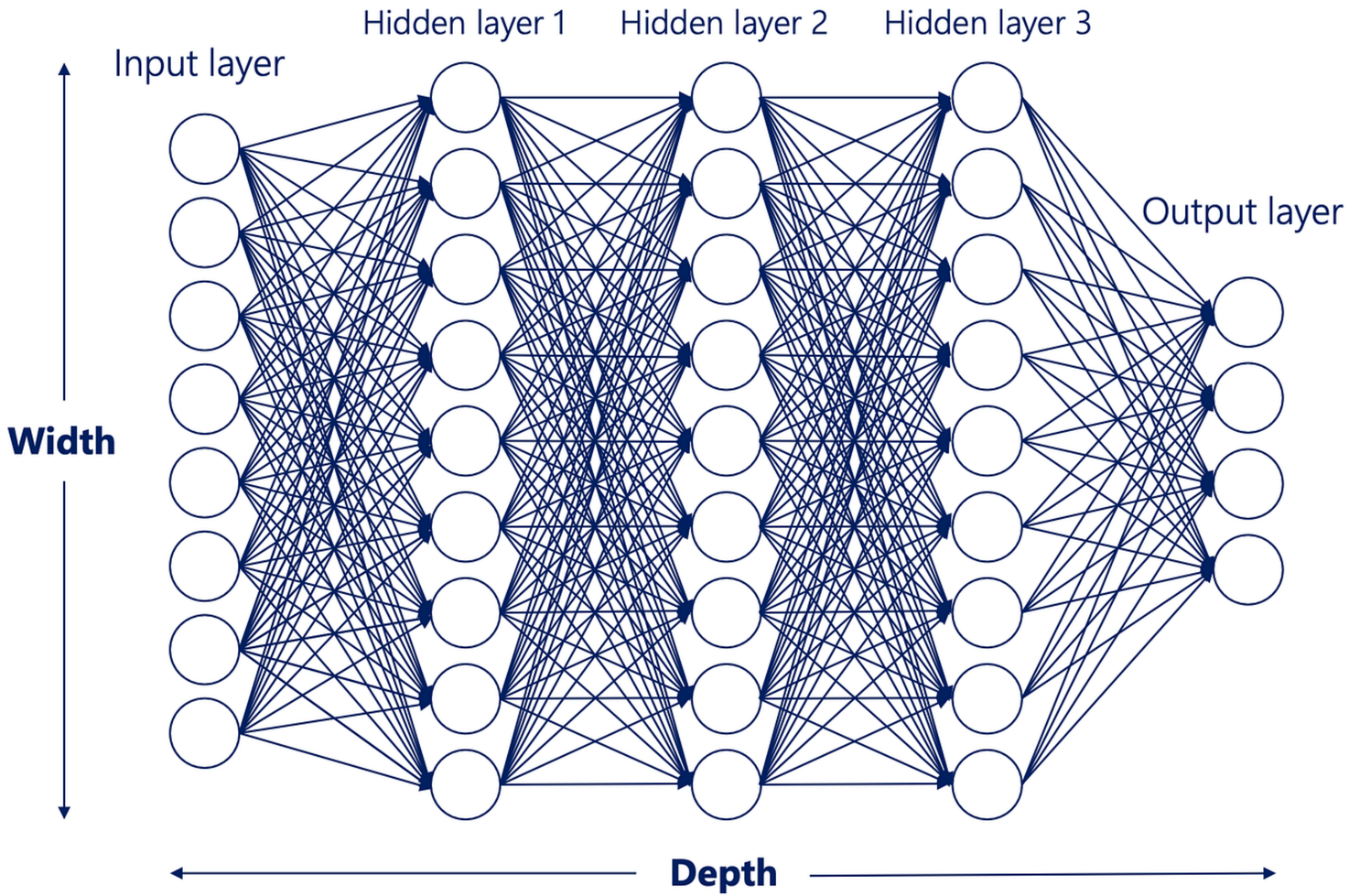



Image from Vijay Gadre

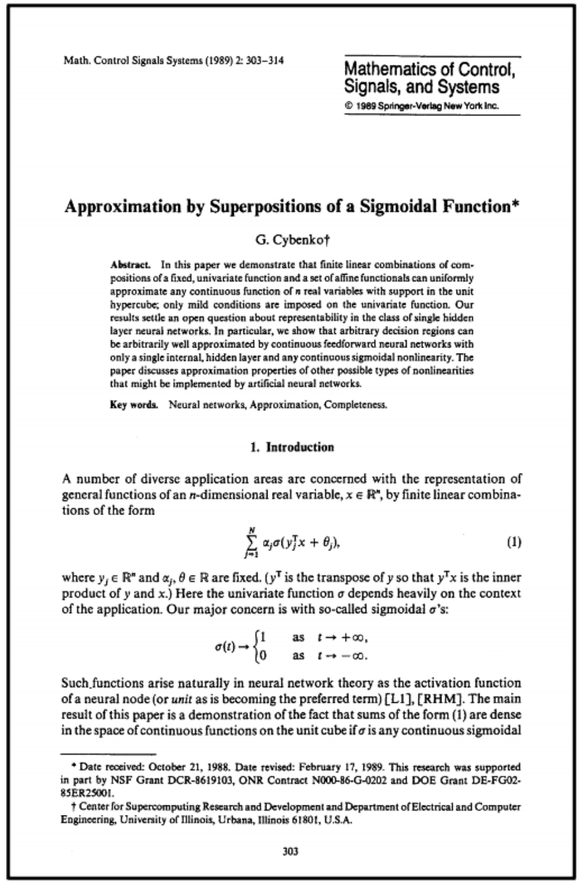
# The Universal Approximation Theorem

**Theorem (Cybenko 1989; Hornik 1991):** Given a continuous function $f$, it is possible to construct a fully connected, single-layer neural network which can closely approximate $f$ on a bounded domain, as long as we can make the hidden dimension arbitrarily large.

**Hornik theorem 1:** Whenever the activation function is *bounded and nonconstant*, then, for any finite measure $\mu$, standard multilayer feedforward networks can approximate any function in $L^p(\mu)$ (the space of all functions on $R^k$ such that $\int_{R^k} |f(x)|^p d\mu(x) < \infty$) arbitrarily well, provided that sufficiently many hidden units are available.

**Hornik theorem 2:** Whenever the activation function is *continuous, bounded and nonconstant*, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on $X$ arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

# The Universal Approximation Theorem



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.

**1. Introduction**

A number of diverse application areas are concerned with the representation of general functions of an $n$-dimensional real variable, $x \in \mathbb{R}^n$, by finite linear combinations of the form

$$\sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j), \tag{1}$$

where $y_j \in \mathbb{R}^n$ and $\alpha_j, \theta \in \mathbb{R}$ are fixed. ($y^T$ is the transpose of $y$ so that $y^T x$ is the inner product of $y$ and $x$.) Here the univariate function $\sigma$ depends heavily on the context of the application. Our major concern is with so-called sigmoidal $\sigma$'s:

$$\sigma(t) \to \begin{cases} 1 & \text{as} \quad t \to +\infty, \\ 0 & \text{as} \quad t \to -\infty. \end{cases}$$

Such functions arise naturally in neural network theory as the activation function of a neural node (or *unit* as is becoming the preferred term) [L1], [RHM]. The main result of this paper is a demonstration of the fact that sums of the form (1) are dense in the space of continuous functions on the unit cube if $\sigma$ is any continuous sigmoidal

* Date received: October 21, 1988. Date revised: February 17, 1989. This research was supported in part by NSF Grant DCR-8619103, ONR Contract N000-86-G-0202 and DOE Grant DE-FG02-85ER25001.

† Center for Supercomputing Research and Development and Department of Electrical and Computer Engineering, University of Illinois, Urbana, Illinois 61801, U.S.A.



Neural Networks, Vol. 4, pp. 251–257, 1991
Printed in the USA. 

0893-6080/91 $3.00 + .00


*ORIGINAL CONTRIBUTION*

**Approximation Capabilities of Multilayer Feedforward Networks**

KURT HORNIK

Technische Universität Wien, Vienna, Austria

(*Received* 30 *January* 1990; *revised and accepted* 25 *October* 1990)

**Abstract**—*We show that standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and nonconstant activation function are universal approximators with respect to $L^p(\mu)$ performance criteria, for arbitrary finite input environment measures $\mu$, provided only that sufficiently many hidden units are available. If the activation function is continuous, bounded and nonconstant, then continuous mappings can be learned uniformly over compact input sets. We also give very general conditions ensuring that networks with sufficiently smooth activation functions are capable of arbitrarily accurate approximation to a function and its derivatives.*

**Keywords**—Multilayer feedforward networks, Activation function, Universal approximation capabilities, Input environment measure, $L^p(\mu)$ approximation, Uniform approximation, Sobolev spaces, Smooth approximation.

**1. INTRODUCTION**

The approximation capabilities of neural network architectures have recently been investigated by many authors, including Carroll and Dickinson (1989), Cybenko (1989), Funahashi (1989), Gallant and White (1988), Hecht-Nielsen (1989), Hornik, Stinchcombe, and White (1989, 1990), Irie and Miyake (1988), Lapedes and Farber (1988), Stinchcombe and White (1989, 1990). (This list is by no means complete.)

If we think of the network architecture as a rule for computing values at $l$ output units given values at $k$ input units, hence implementing a class of mappings from $R^k$ to $R^l$, we can ask how well arbitrary mappings from $R^k$ to $R^l$ can be approximated by the network, in particular, if as many hidden units as required for internal representation and computation may be employed.

How to measure the accuracy of approximation depends on how we measure closeness between functions, which in turn varies significantly with the specific problem to be dealt with. In many applications, it is necessary to have the network perform *simultaneously* well on all input samples taken from some compact input set $X$ in $R^k$. In this case, closeness is measured by the uniform distance between functions on $X$, that is,

$$\rho_{\infty,X}(f, g) = \sup_{x \in X} |f(x) - g(x)|.$$

In other applications, we think of the inputs as random variables and are interested in the *average performance* where the average is taken with respect to the input environment measure $\mu$, where $\mu(R^k) < \infty$. In this case, closeness is measured by the $L^p(\mu)$ distances

$$\rho_{p,\mu}(f, g) = \left[\int_{R^k} |f(x) - g(x)|^p \, d\mu(x)\right]^{1/p},$$

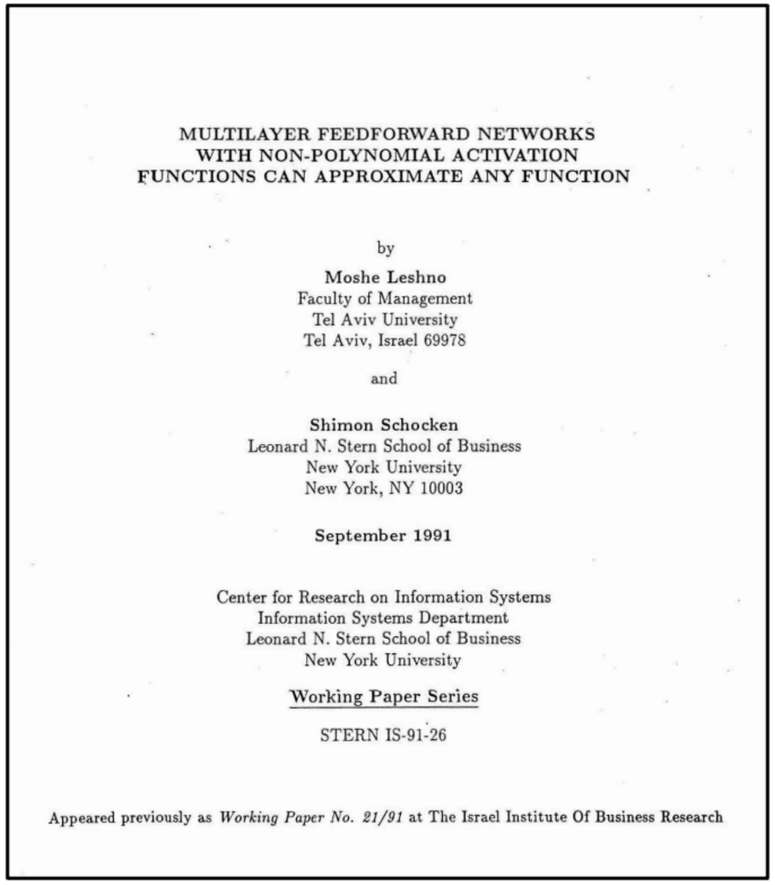
$1 \leq p < \infty$, the most popular choice being $p = 2$, corresponding to mean square error.

Of course, there are many more ways of measuring closeness of functions. In particular, in many applications, it is also necessary that the *derivatives* of the approximating function implemented by the network closely resemble those of the function to be approximated, up to some order. This issue was first taken up in Hornik et al. (1990), who discuss the sources of need of smooth functional approximation in more detail. Typical examples arise in robotics (learning of smooth movements) and signal processing (analysis of chaotic time series); for a recent application to problems of nonparametric inference in statistics and econometrics, see Gallant and White (1989).

All papers establishing certain approximation ca-

Requests for reprints should be sent to Kurt Hornik, Institut für Statistik und Wahrscheinlichkeitstheorie, Technische Universität Wien, Wiedner Hauptstraße 8-10/107, A-1040 Wien, Austria.





MULTILAYER FEEDFORWARD NETWORKS WITH NON-POLYNOMIAL ACTIVATION FUNCTIONS CAN APPROXIMATE ANY FUNCTION

by

**Moshe Leshno**
Faculty of Management
Tel Aviv University
Tel Aviv, Israel 69978

and

**Shimon Schocken**
Leonard N. Stern School of Business
New York University
New York, NY 10003

**September 1991**

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

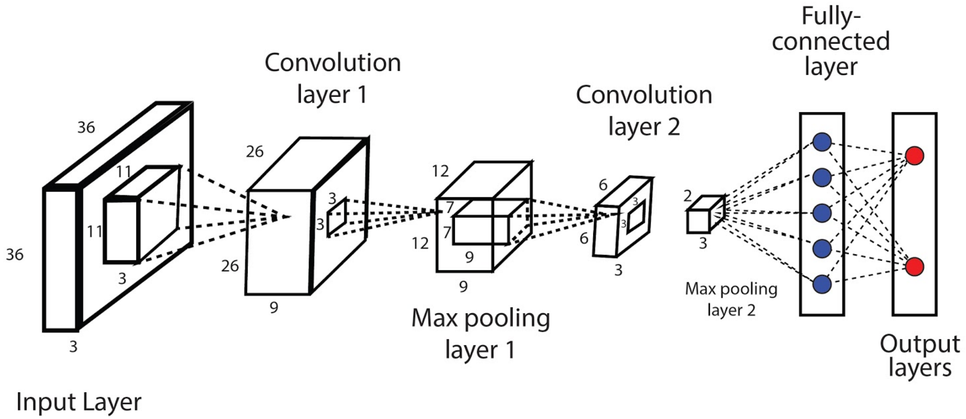
Working Paper Series

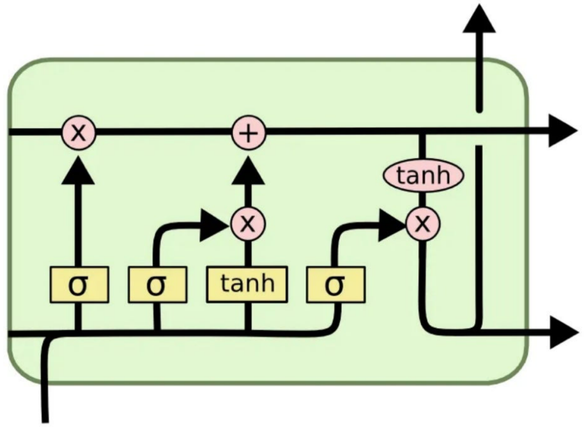
STERN IS-91-26

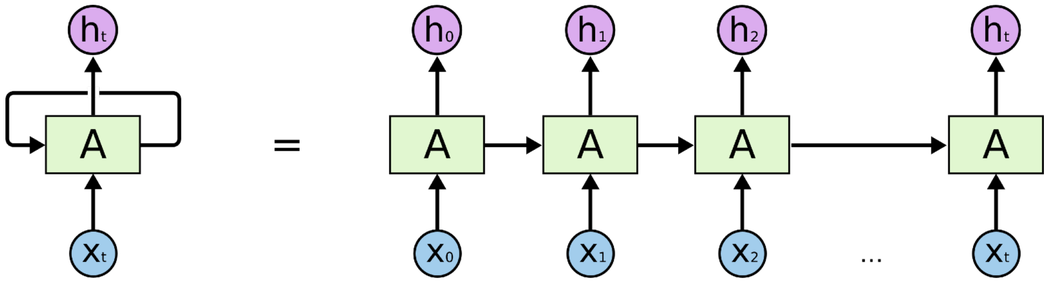
Appeared previously as *Working Paper No. 21/91* at The Israel Institute Of Business Research

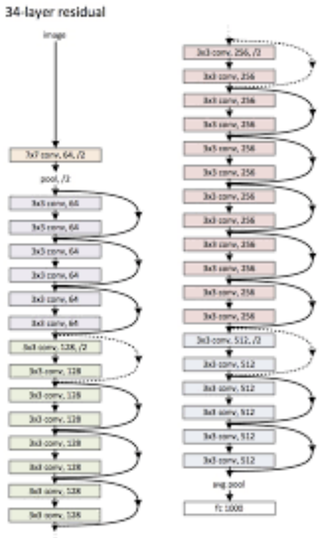
Many different versions of this theorem, but no guarantees on efficiency, learnability, or generalization!

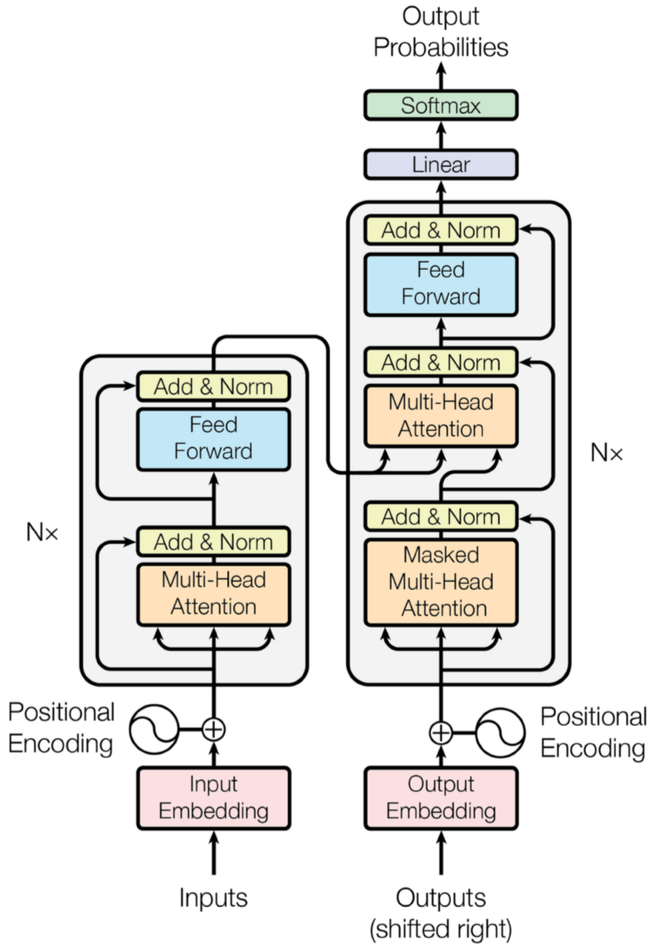
# Today: Other Neural Network Architectures



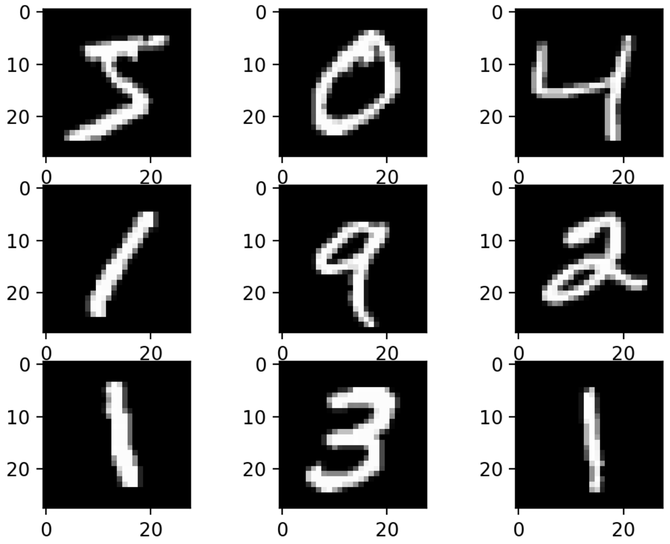



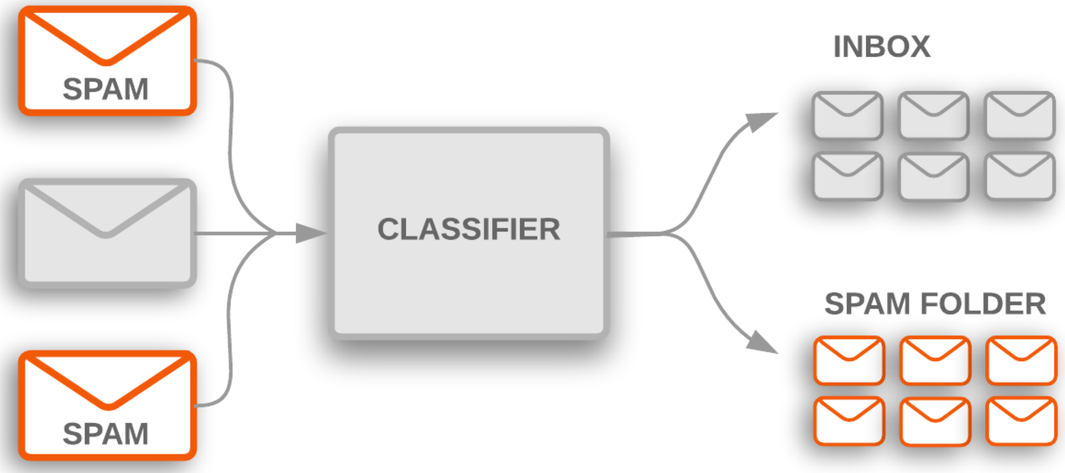



# Many architectural advances have come from two domains

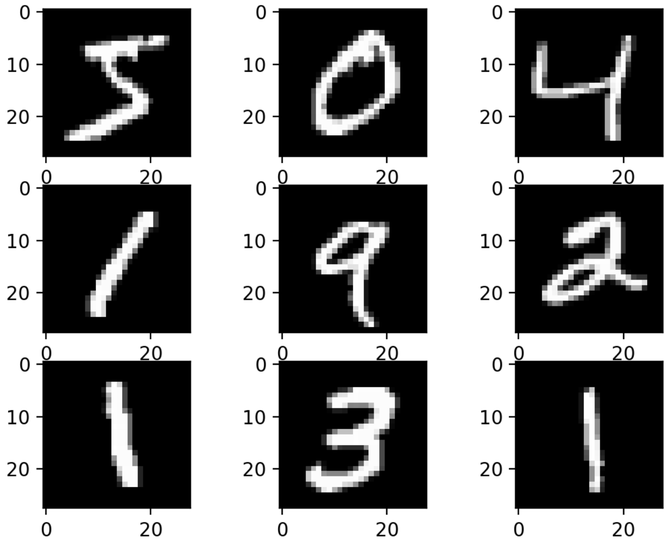


Computer Vision




Natural Language Processing

# Many architectural advances have come from two domains



Computer Vision

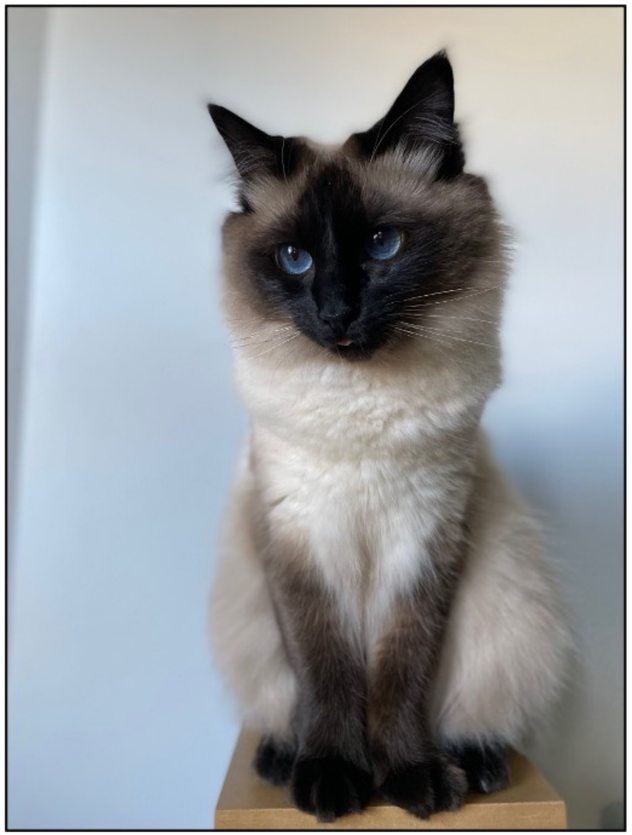



Natural Language Processing

# What tasks does computer vision care about?

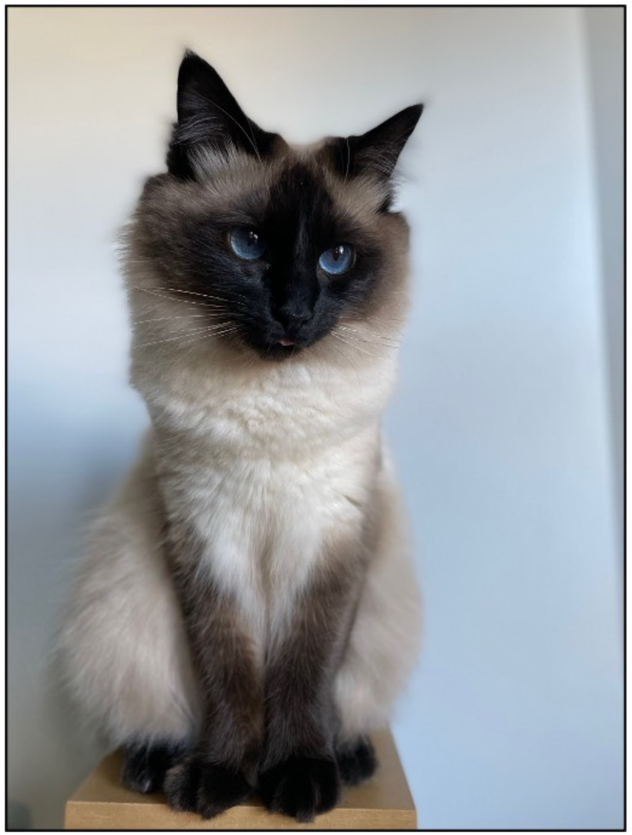
- Object detection and classification
- Semantic segmentation
- Image captioning
- Visual question answering
- Video classification and understanding
- Image classification
- …

# Image classification



**cat**
dog
horse
person
airplane
house
classroom
computer
water bottle
table
hat
…

# Image classification



**cat**
dog
horse
person
airplane
house
classroom
computer
water bottle
table
hat
…

# Image classification

**cat**
dog
horse
person
airplane
house
classroom
computer
water bottle
table
hat
…

# Image classification

**cat**
dog
horse
person
airplane
house
classroom
computer
water bottle
table
hat
…

# Image classification

**cat**
dog
horse
person
airplane
house
classroom
computer
water bottle
table
hat
…

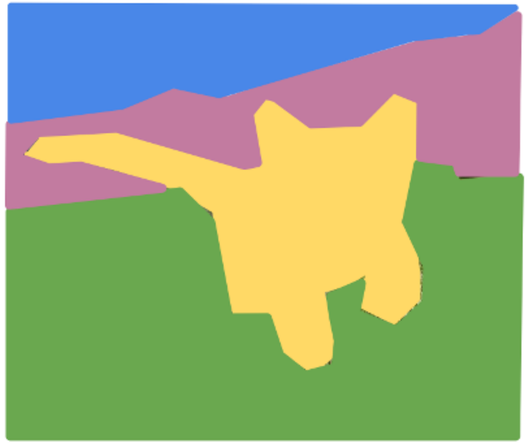
# Beyond Image classification

**Classification**

CAT

No spatial extent

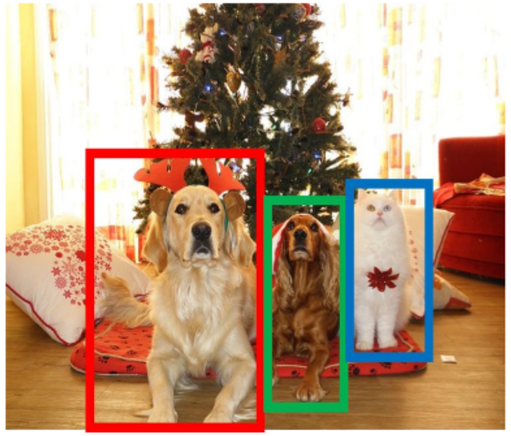
**Semantic Segmentation**



GRASS, CAT, TREE, SKY

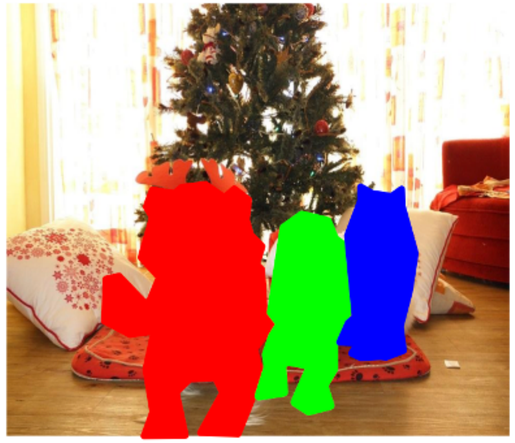
No objects, just pixels

**Object Detection**



DOG, DOG, CAT

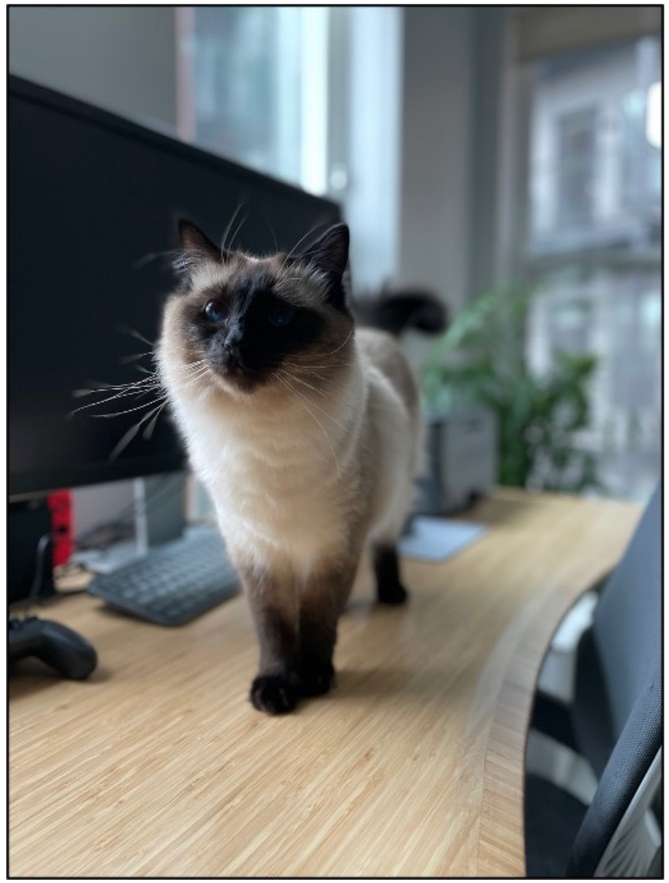
**Instance Segmentation**



DOG, DOG, CAT

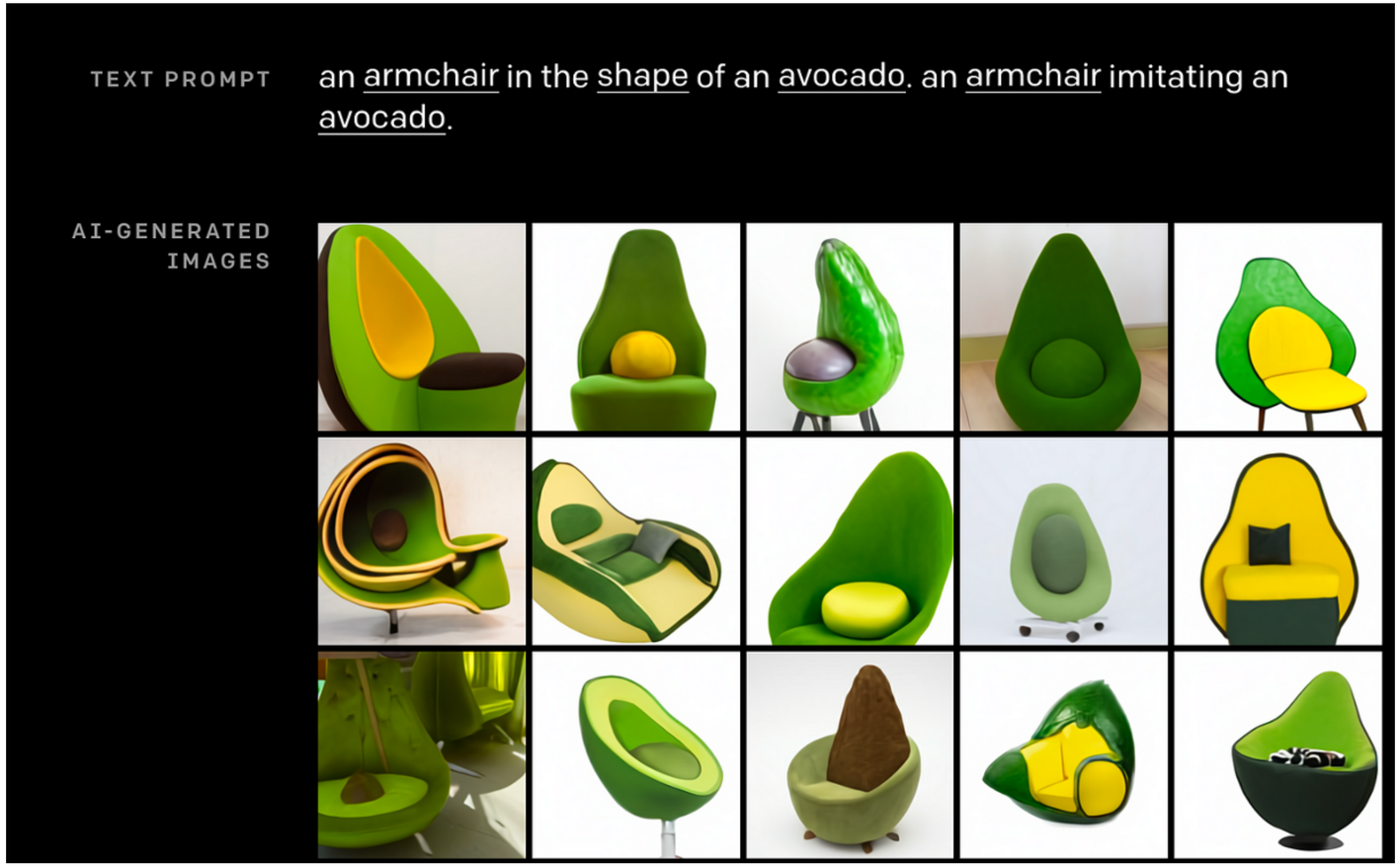
Multiple Object

# Image Captioning

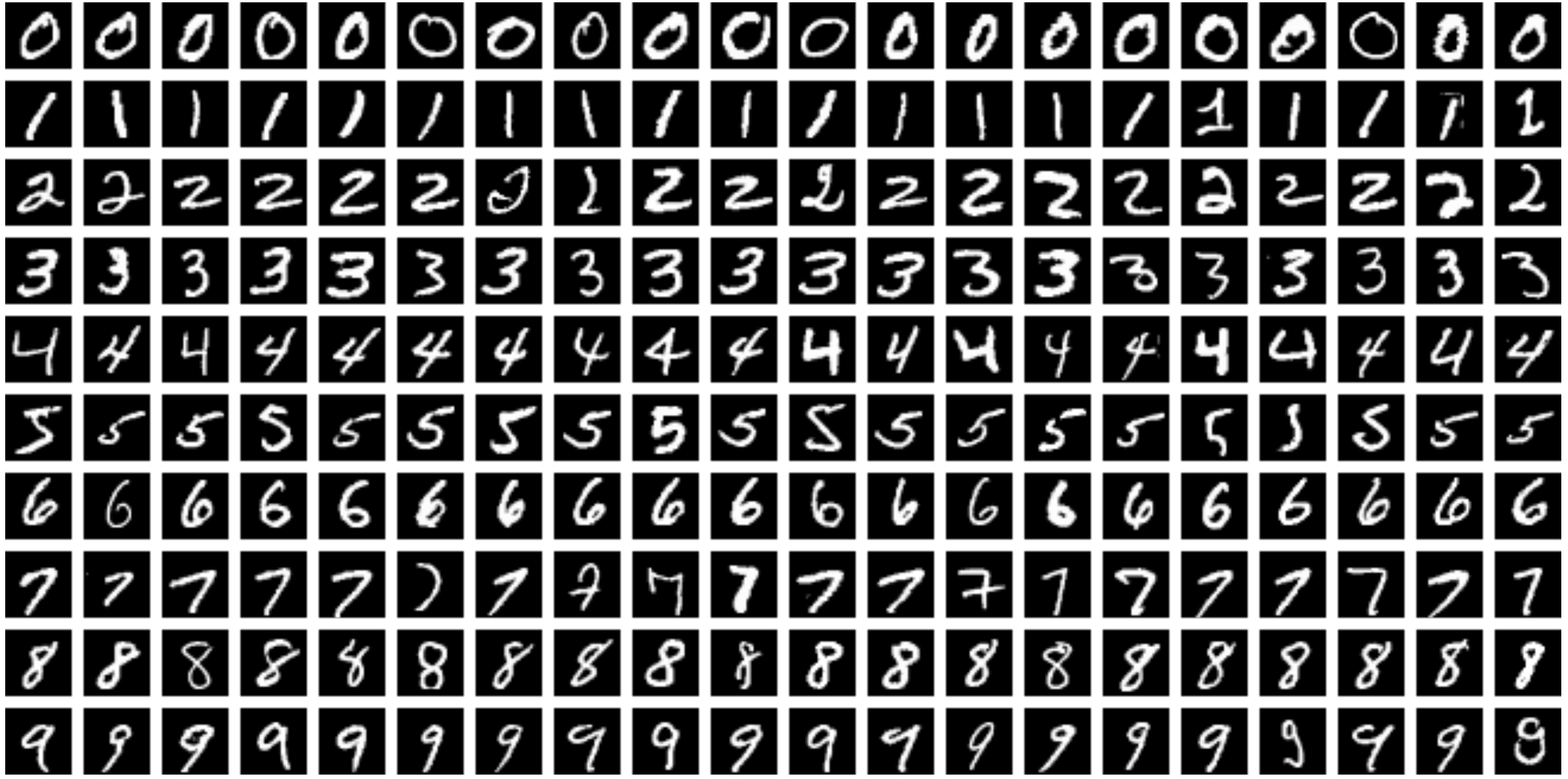


A cat standing on a desk

# Image Generation



# MNIST Digit Classification



# MNIST Digit Classification

0

1

2

1

??

# MNIST Digit Classification

Task specification:

- Input features: binary pixel values (28x28)
- Output: a digit classification (0-9)


0


1


2


1


??

# MNIST Digit Classification

Task specification:

- Input features: binary pixel values (28x28)
- Output: a digit classification (0-9)

What happens if we use a Naive Bayes classifier?


0


1


2


1


??

# MNIST Digit Classification

Task specification:

- Input features: binary pixel values (28x28)
- Output: a digit classification (0-9)

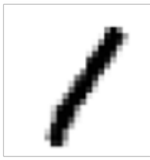
What happens if we use a Naive Bayes classifier?

- Will overfit to individual pixels
- Not robust to scaling, moving left/right, etc.


0


1


2


1


??

# How many parameters in a fully connected network?

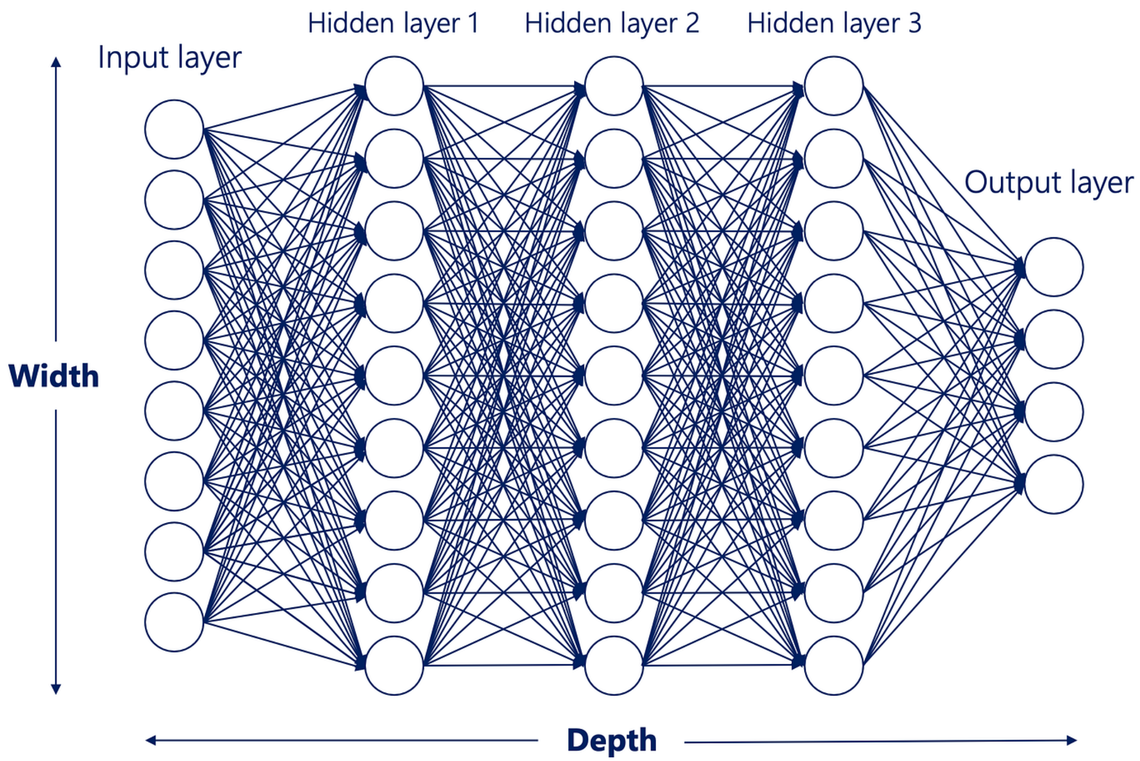



Image from Vijay Gadre

# How many parameters in a fully connected network?

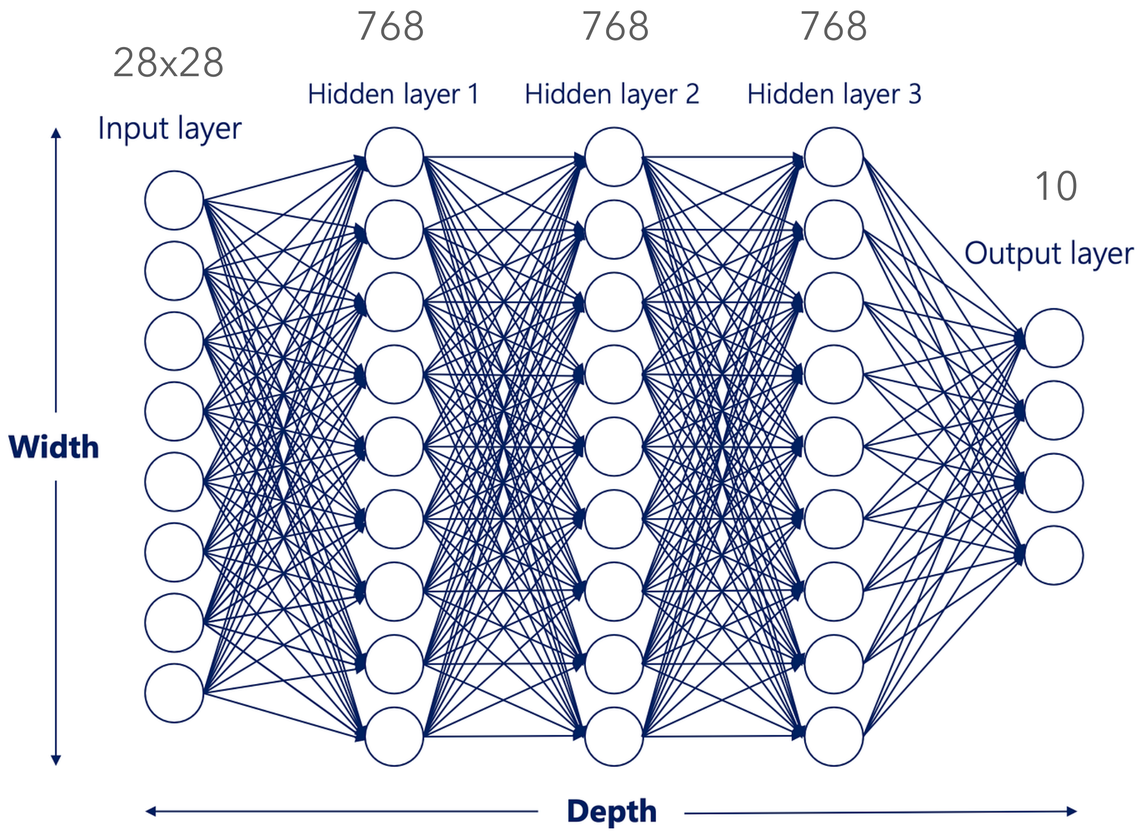



Image from Vijay Gadre

# How many parameters in a fully connected network?

Sum over each layer:

- First layer: 28x28x768
- Second layer: 768x768
- Third layer: 768x768
- Output: 768*10

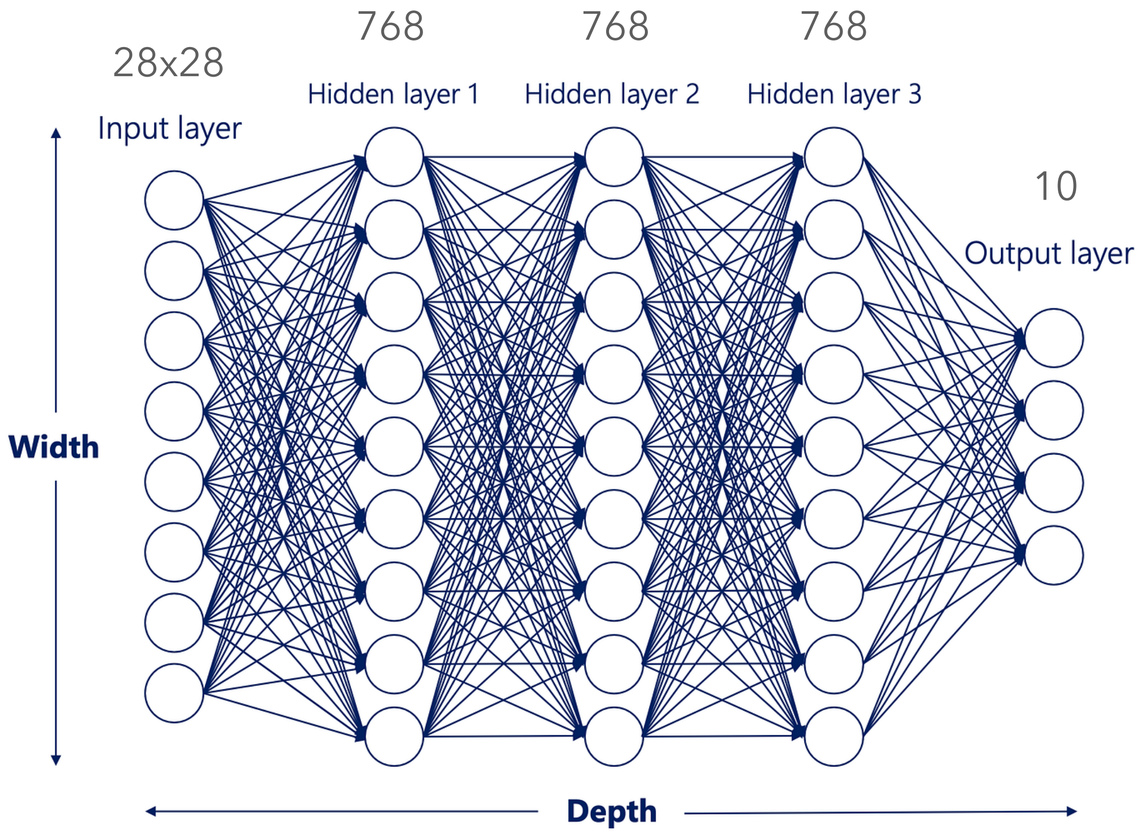



Image from Vijay Gadre

# How many parameters in a fully connected network?

Sum over each layer:

- First layer: 28x28x768
- Second layer: 768x768
- Third layer: 768x768
- Output: 768*10

Total: 1,777,152 parameters

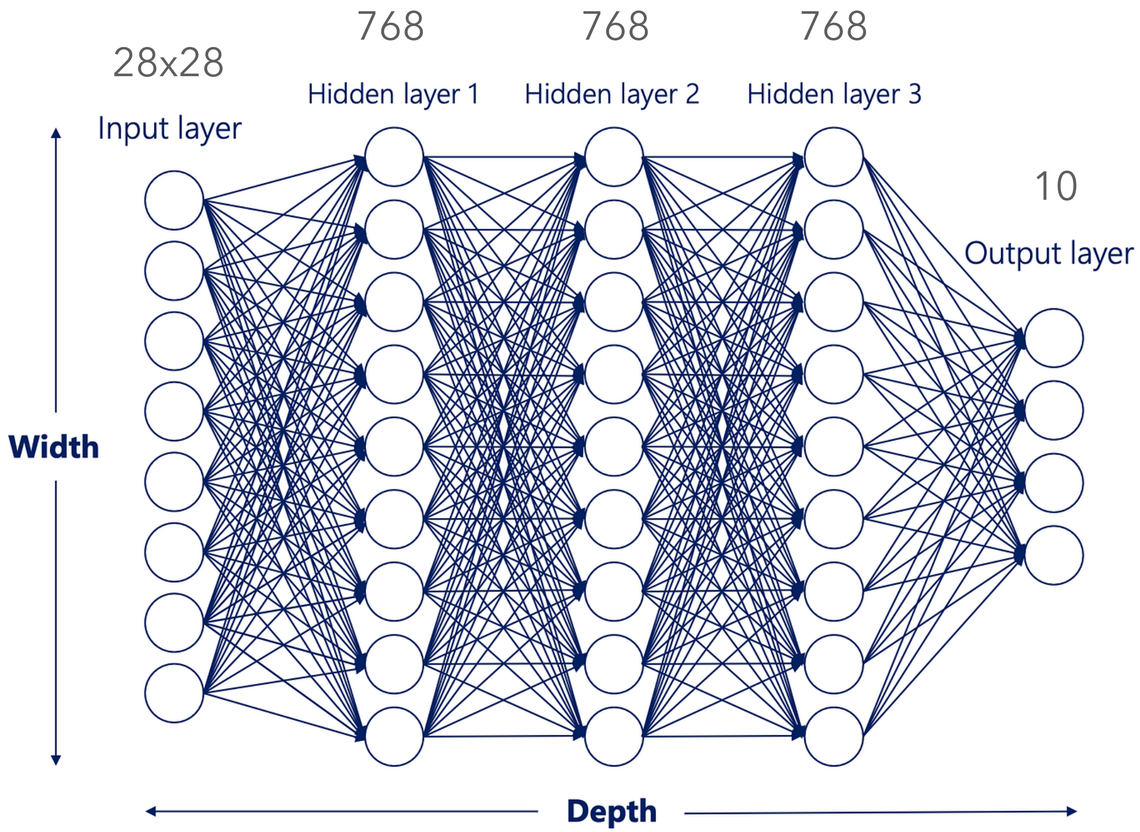
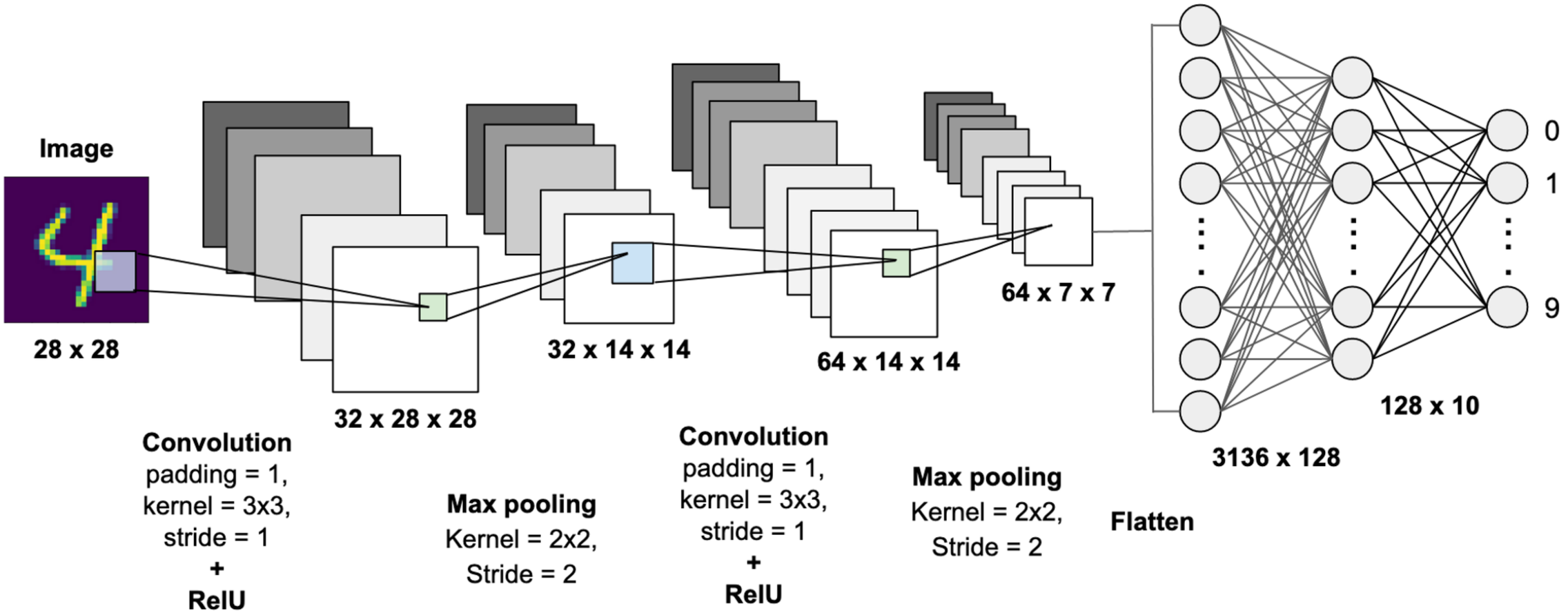


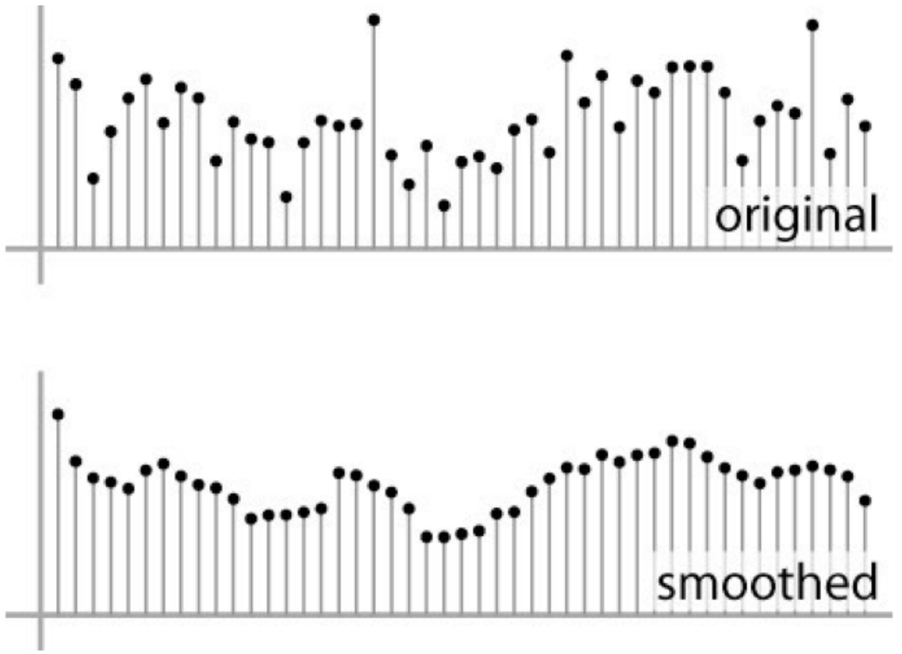

Image from Vijay Gadre

# Convolutional Neural Networks



# What’s a convolution?

- Basic idea: define a function by averaging over a sliding window
- Example in one dimension: smoothing



# Convolutions in 1D

- Moving average:

$$c[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} a[j]$$

# Convolutions in 1D

- Moving average:

$$c[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} a[j]$$

- Convolution: same idea but with weighted average

$$(a \star b)[i] = \sum_{j} a[j] \cdot b[i-j]$$

# Convolutions in 1D

- Moving average:

$$c[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} a[j]$$

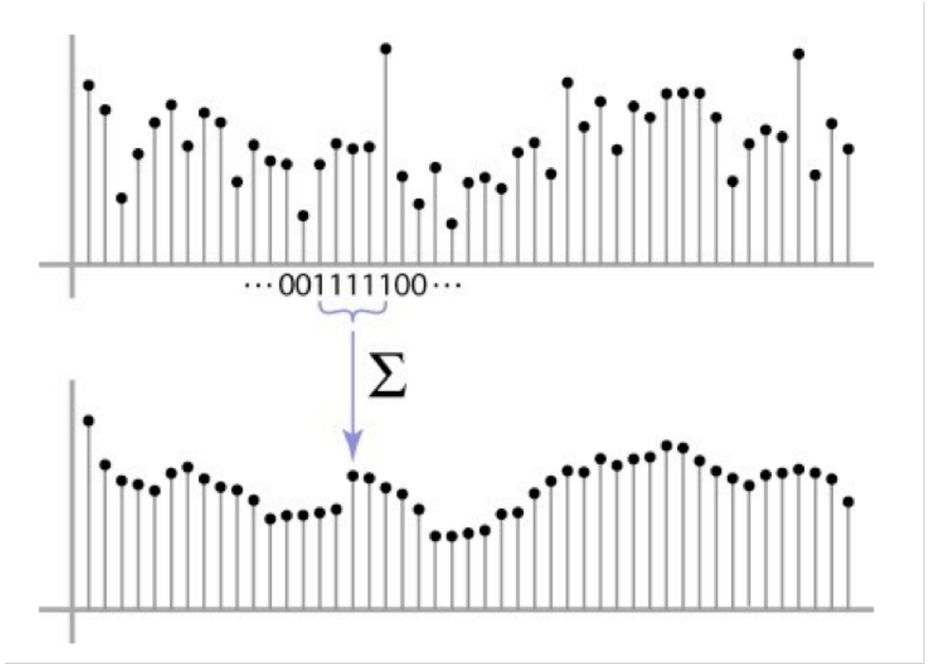
- Convolution: same idea but with weighted average

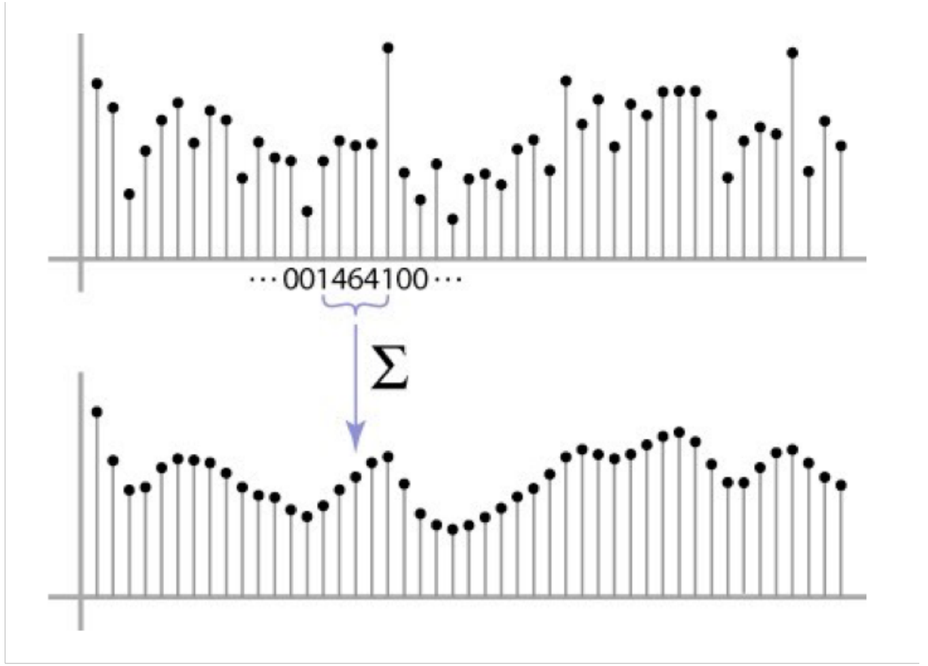
$$(a \star b)[i] = \sum_{j} a[j] \cdot b[i-j]$$

Called a “filter”

# Convolutions in 1D

- Filters in one dimension:
  - Box filter: […, 0, 0, 1, 1, 1, 1, 1, 0, 0,…]/5
  - Gaussian filter: […, 0, 0, 1, 4, 6, 4, 1, 0, 0,…]/16



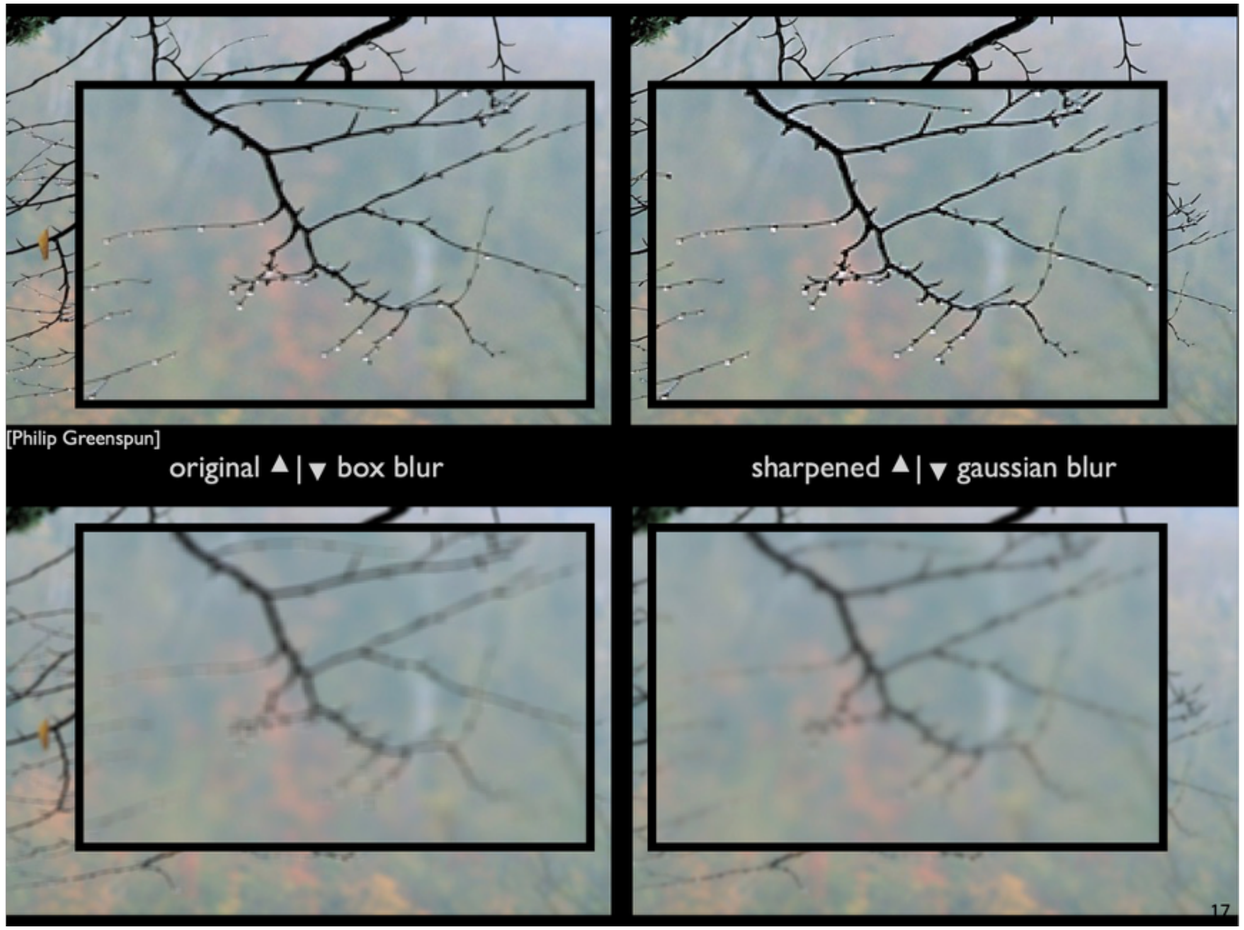



# Convolutions in 2D

Filters in two dimensions: same idea but apply over a square patch of inputs (often 3x3 or 5x5)

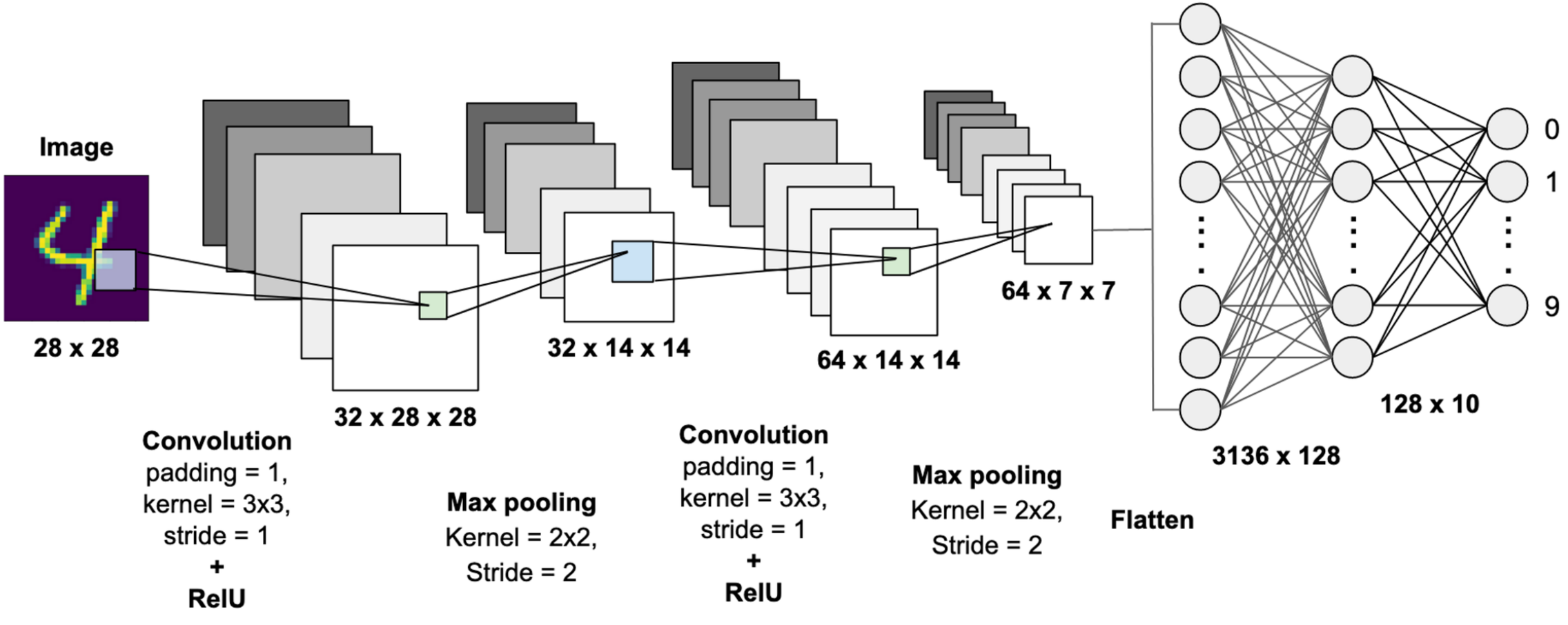
Applications:

- Blurring
- Sharpening
- Edge detection



# Convolutional Neural Networks

Key idea: learn the filter weights via backprop



# Stride and Padding

What does a convolution do to the size of the input?

# Stride and Padding

What does a convolution do to the size of the input?

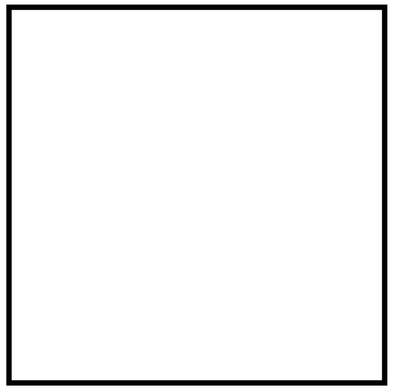


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

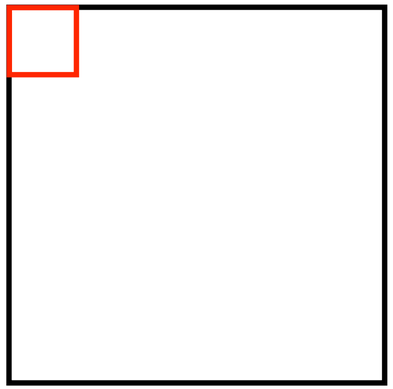


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

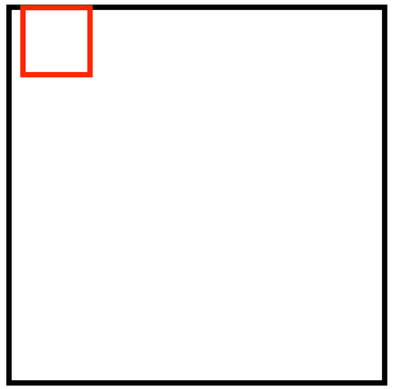


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

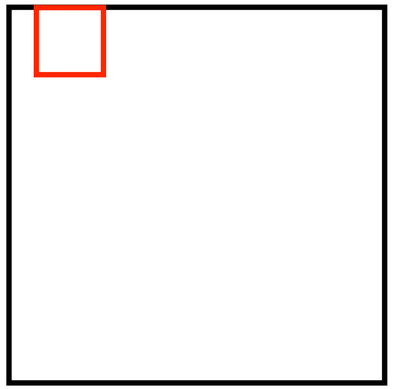


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

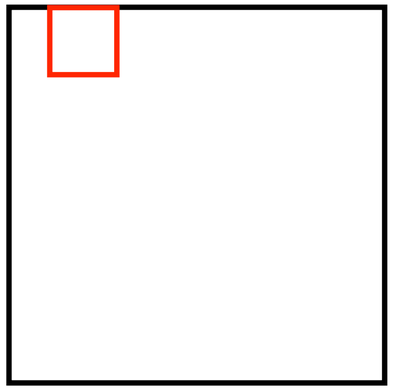


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

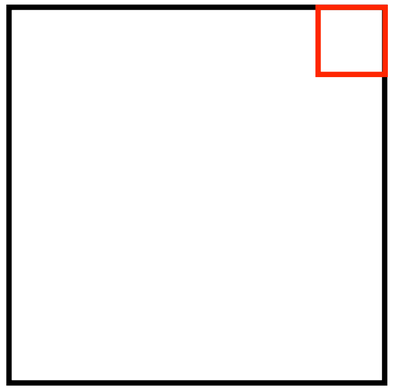


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

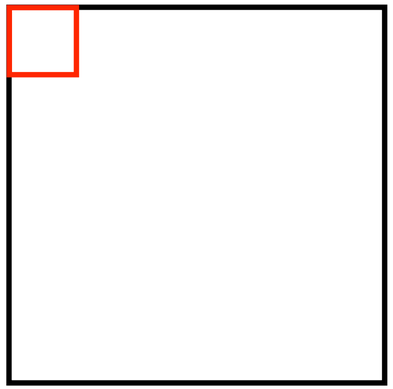


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

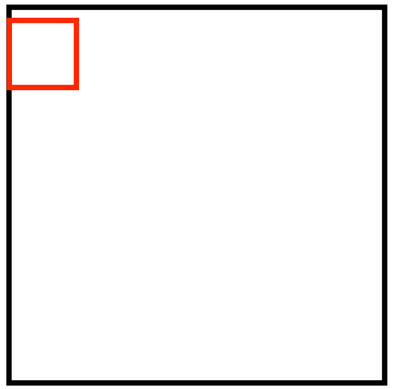


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

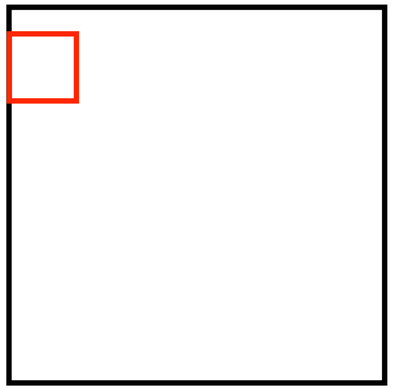


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

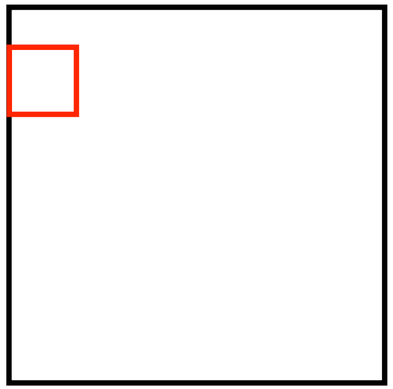


Image: 28x28

Filter: 5x5

# Stride and Padding

What does a convolution do to the size of the input?

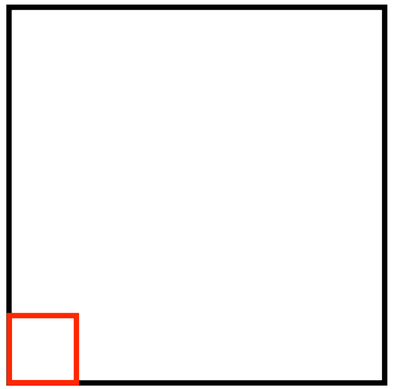
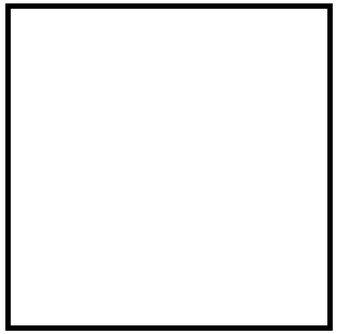


Image: 28x28

Filter: 5x5



Output: 24x24

# Stride and Padding

What if we make the stride larger? (i.e., less overlap between filters)

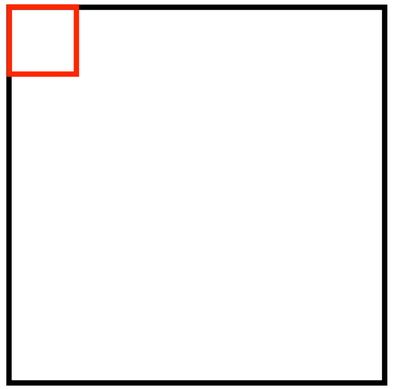


Image: 28x28

Filter: 5x5

# Stride and Padding

What if we make the stride larger? (i.e., less overlap between filters)

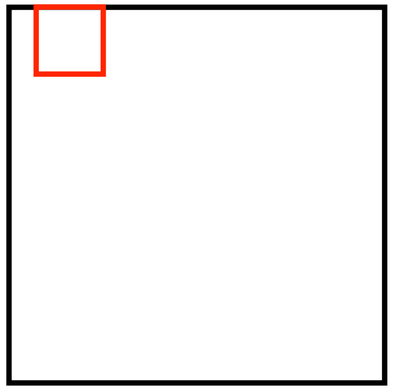


Image: 28x28

Filter: 5x5

# Stride and Padding

What if we make the stride larger? (i.e., less overlap between filters)

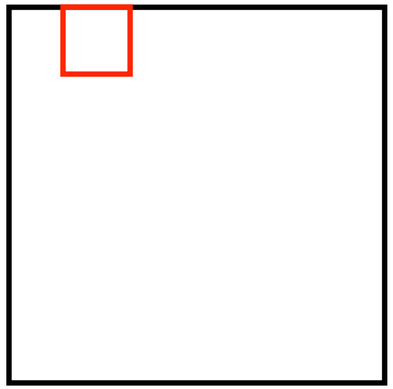


Image: 28x28

Filter: 5x5

# Stride and Padding

What if we make the stride larger? (i.e., less overlap between filters)

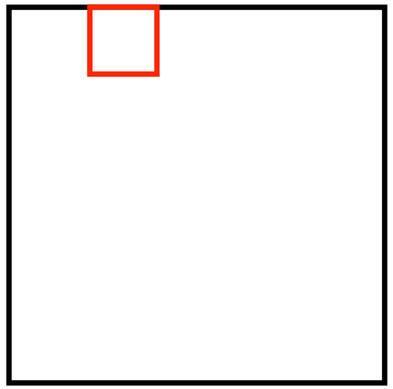


Image: 28x28

Filter: 5x5

# Stride and Padding

What if we make the stride larger? (i.e., less overlap between filters)

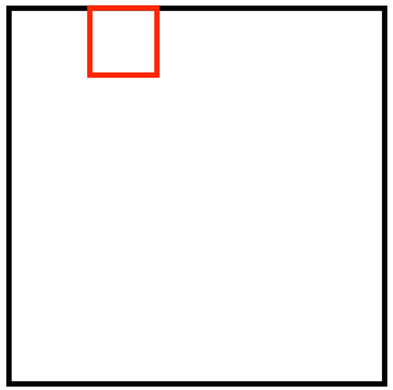
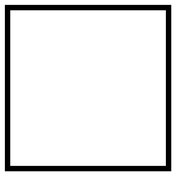


Image: 28x28

Filter: 5x5



Output: 12x12

# Stride and Padding

What if we want the output size to be the same shape as the output?

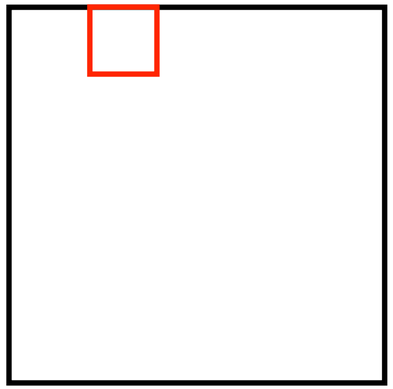
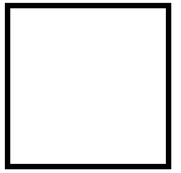


Image: 28x28

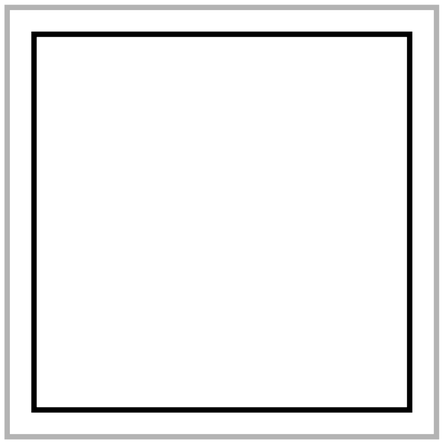
Filter: 5x5



Output: 12x12

# Stride and Padding

What if we want the output size to be the same shape as the output?

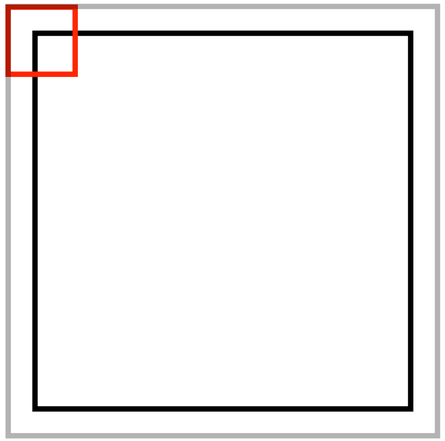


Padded Image: 32x32

Filter: 5x5

# Stride and Padding

What if we want the output size to be the same shape as the output?

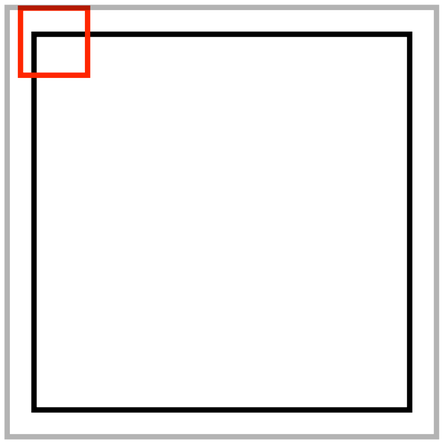


Padded Image: 32x32

Filter: 5x5

# Stride and Padding

What if we want the output size to be the same shape as the output?

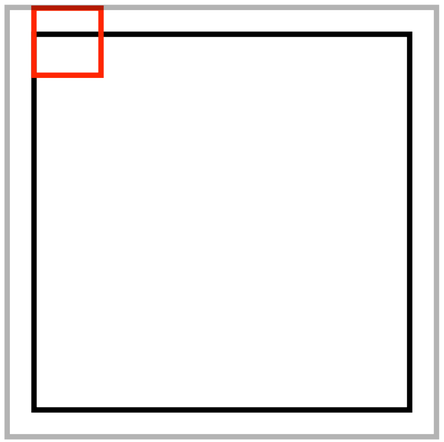


Padded Image: 32x32

Filter: 5x5

# Stride and Padding

What if we want the output size to be the same shape as the output?

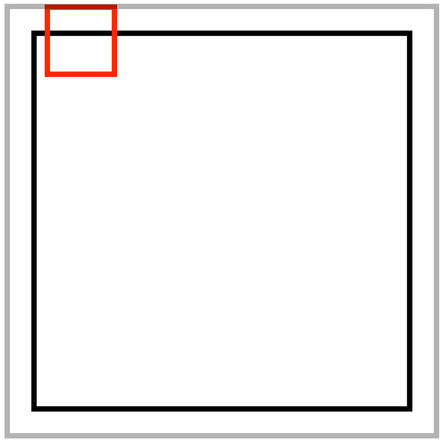


Padded Image: 32x32

Filter: 5x5

# Stride and Padding

What if we want the output size to be the same shape as the output?

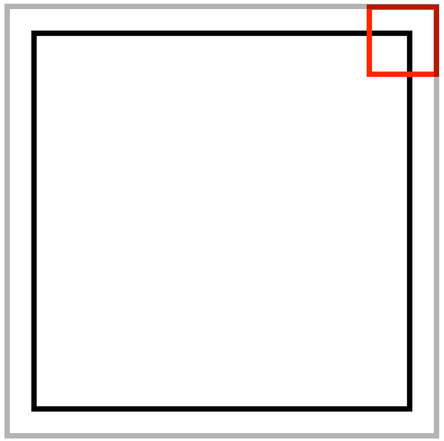


Padded Image: 32x32

Filter: 5x5

# Stride and Padding

What if we want the output size to be the same shape as the output?

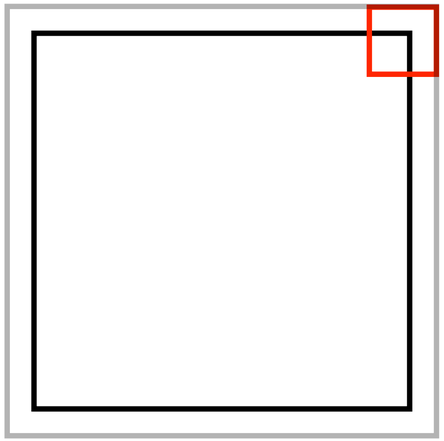


Padded Image: 32x32

Filter: 5x5

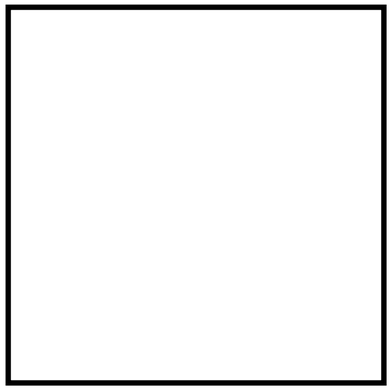
# Stride and Padding

What if we want the output size to be the same shape as the output?



Padded Image: 32x32

Filter: 5x5



Output: 28x28

# Convolutions in 2D

**Single Channel Padded Image**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | **5** | **0** | **8** | **7** | **8** | **1** | 0 |
| 0 | **1** | **9** | **5** | **0** | **7** | **7** | 0 |
| 0 | **6** | **0** | **2** | **4** | **6** | **6** | 0 |
| 0 | **9** | **7** | **6** | **6** | **8** | **4** | 0 |
| 0 | **8** | **3** | **8** | **5** | **1** | **3** | 0 |
| 0 | **7** | **2** | **7** | **0** | **1** | **0** | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*1 x 1 x (6+2) x (6+2)*

*

**Filter**

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

*1 x 1 x 3 x 3*

=

**Result**

| | | | | | |
|---|---|---|---|---|---|
| -19 | 22 | -20 | -12 | -17 | 11 |
| 16 | -30 | -1 | 23 | -7 | -14 |
| -14 | 24 | 7 | -2 | 1 | -7 |
| -15 | -10 | -1 | -1 | -15 | 1 |
| -13 | 13 | -11 | -5 | 13 | -7 |
| -18 | 9 | -18 | 13 | -3 | 4 |

*1 x 1 x 6 x6*

**Region**

| | | |
|---|---|---|
| **9** | **5** | **0** |
| **0** | **2** | **4** |
| **7** | **6** | **6** |

x

**Filter**

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

=

**Element-wise Multiplication**

| | | |
|---|---|---|
| 0 | 5 | 0 |
| 0 | -8 | 4 |
| 0 | 6 | 0 |

**adds up to 7**

# Increasing and Decreasing Dimensionality

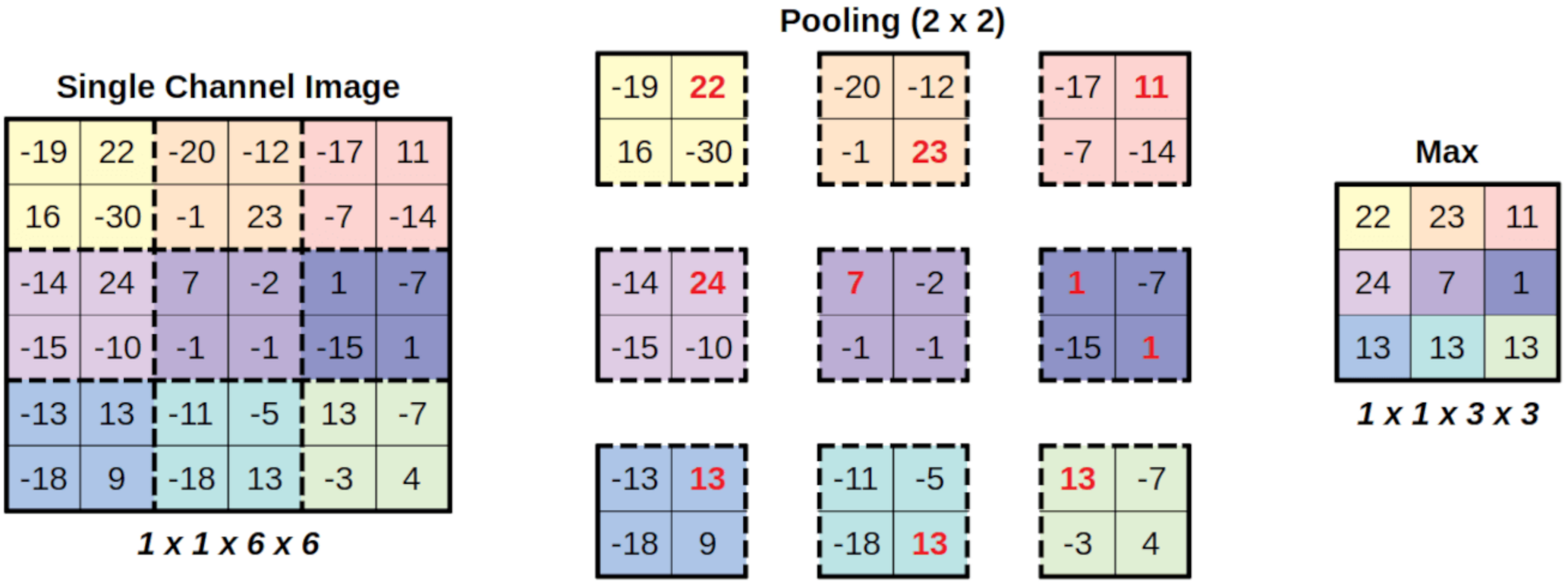
To increase dimensionality:

- Multiple filters at each layer
- This corresponds to extracting multiple types of features/patterns

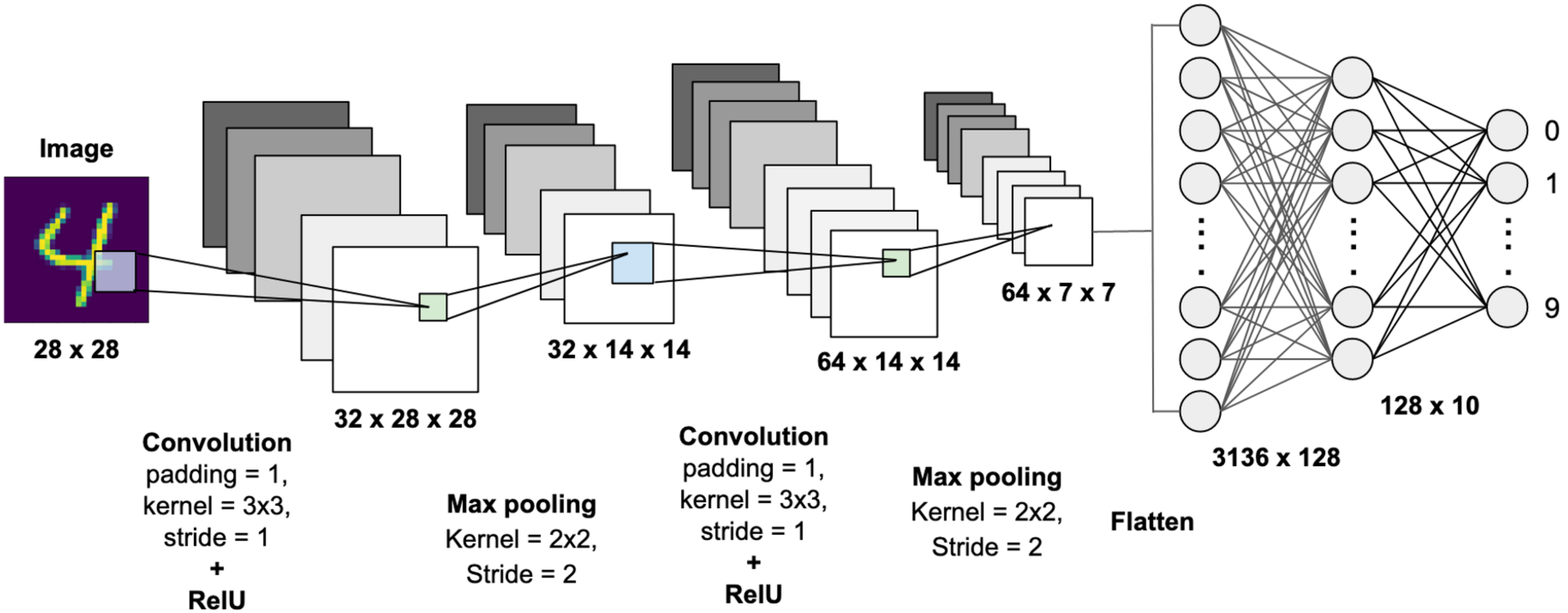
To decrease dimensionality:

- Interleave with max pooling operations
- This checks for the presence of features in a position-invariant way

# Max Pooling

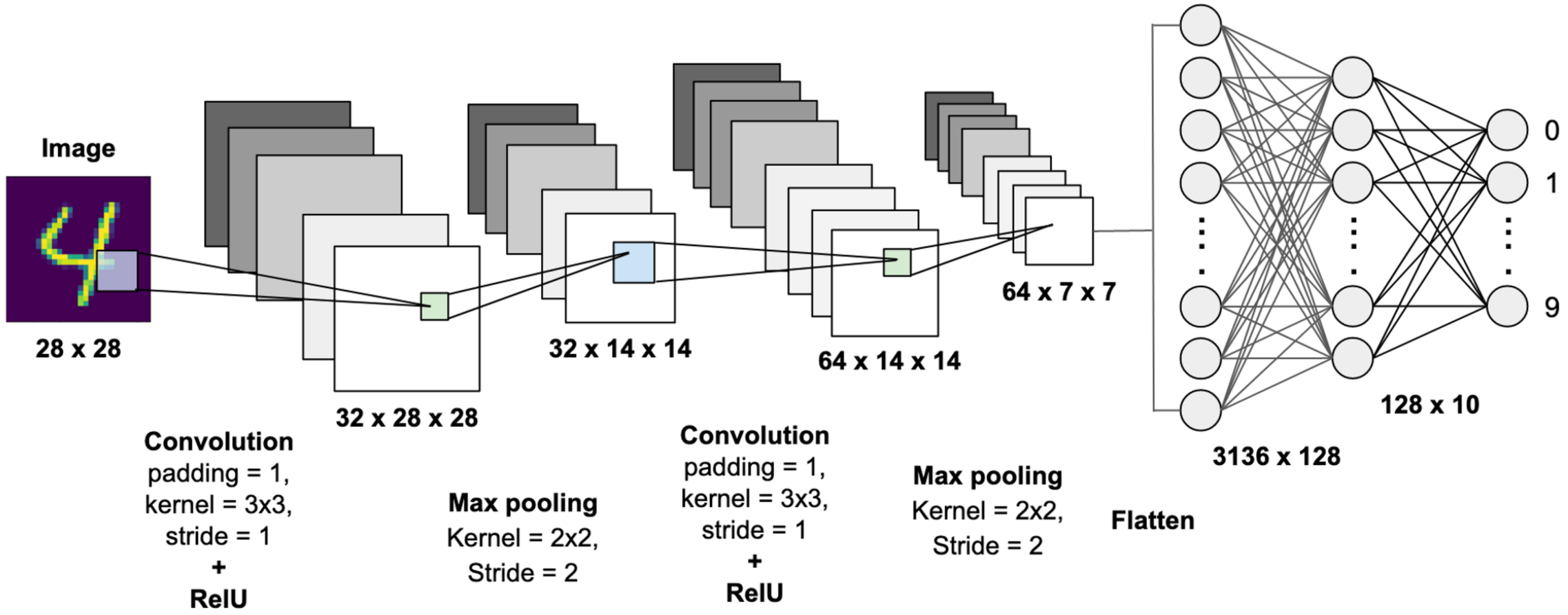

Single Channel Image
-19 22 -20 -12 -17 11
16 -30 -1 23 -7 -14
-14 24 7 -2 1 -7
-15 -10 -1 -1 -15 1
-13 13 -11 -5 13 -7
-18 9 -18 13 -3 4
1 x 1 x 6 x 6
Pooling (2 x 2)
-19 22
16 -30
-20 -12
-1 23
-17 11
-7 -14
-14 24
-15 -10
7 -2
-1 -1
1 -7
-15 1
-13 13
-18 9
-11 -5
-18 13
13 -7
-3 4
Max
22 23 11
24 7 1
13 13 13
1 x 1 x 3 x 3

# Convolutional Neural Networks



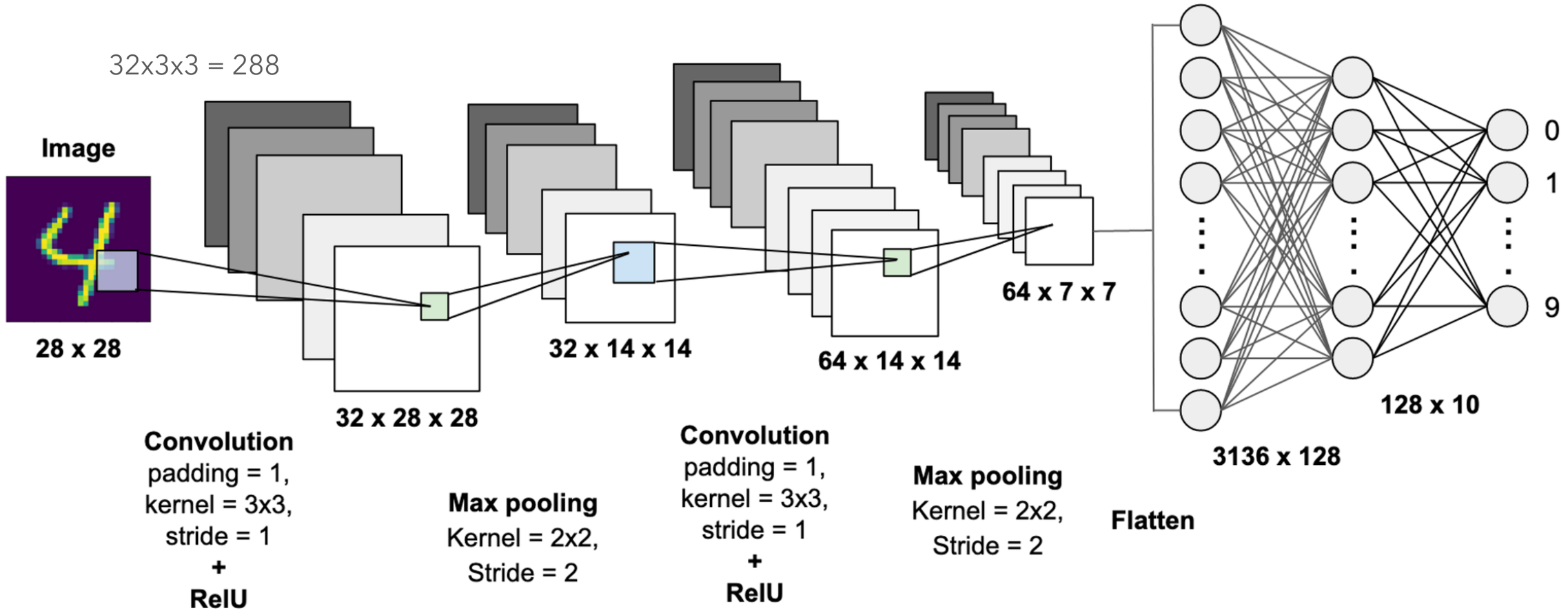
# Convolutional Neural Networks

How many learnable parameters at each step?



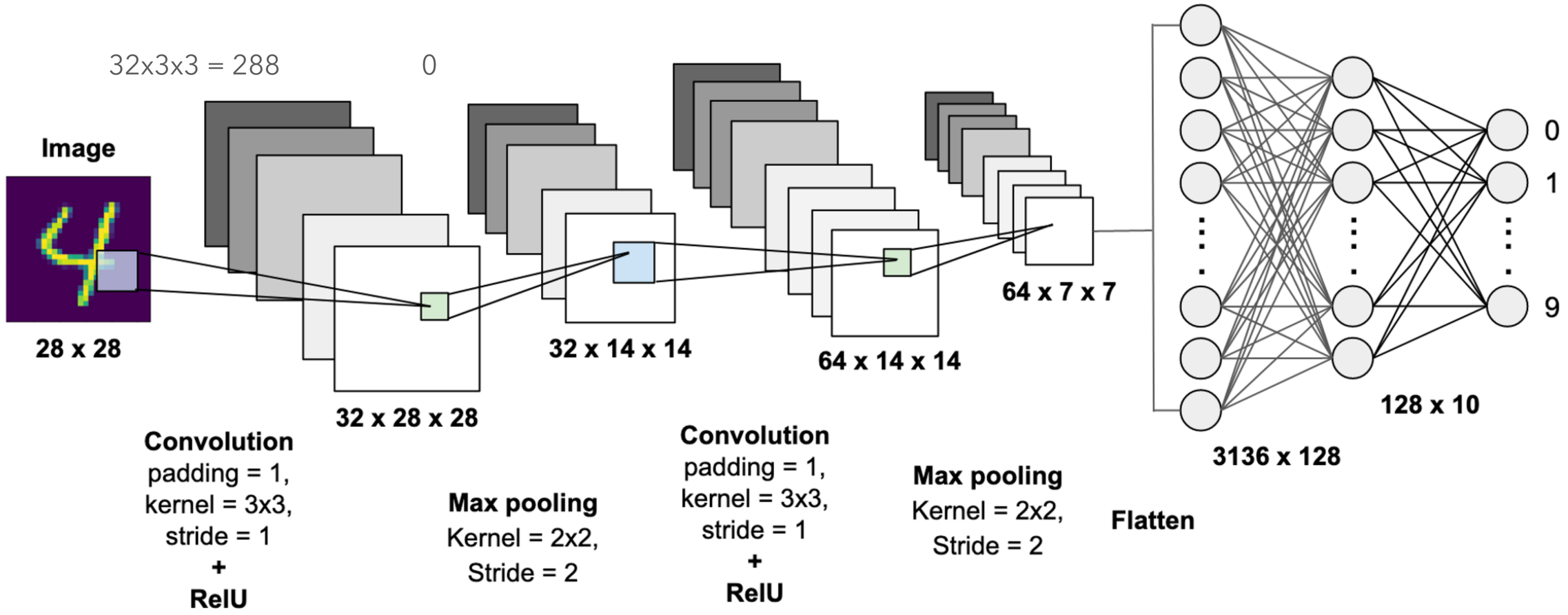
# Convolutional Neural Networks

How many learnable parameters at each step?



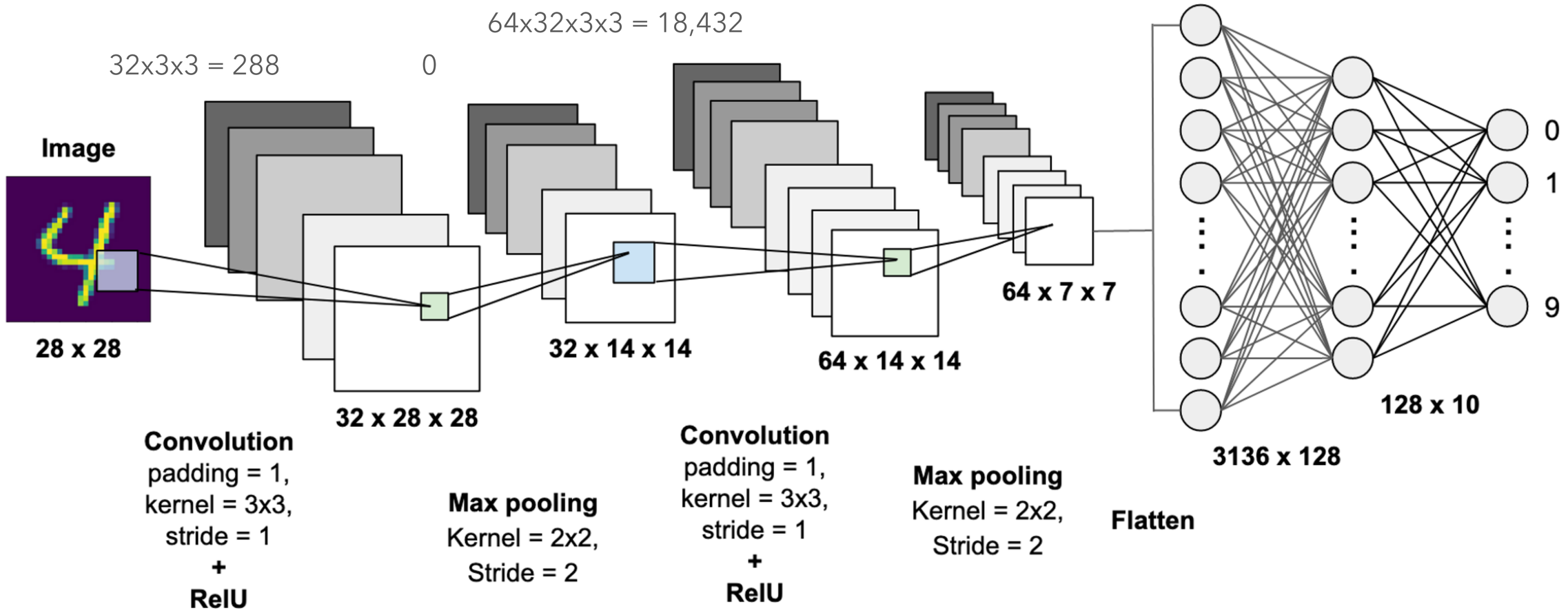
# Convolutional Neural Networks

How many learnable parameters at each step?



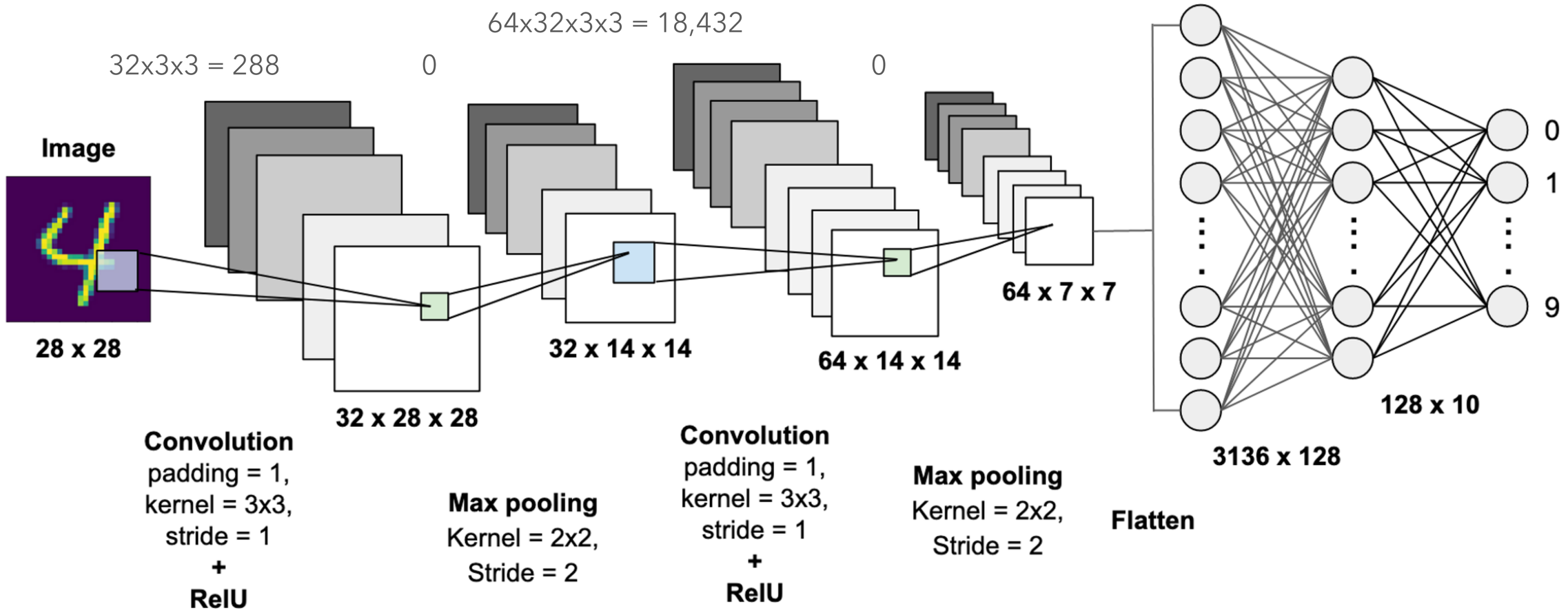
# Convolutional Neural Networks

How many learnable parameters at each step?



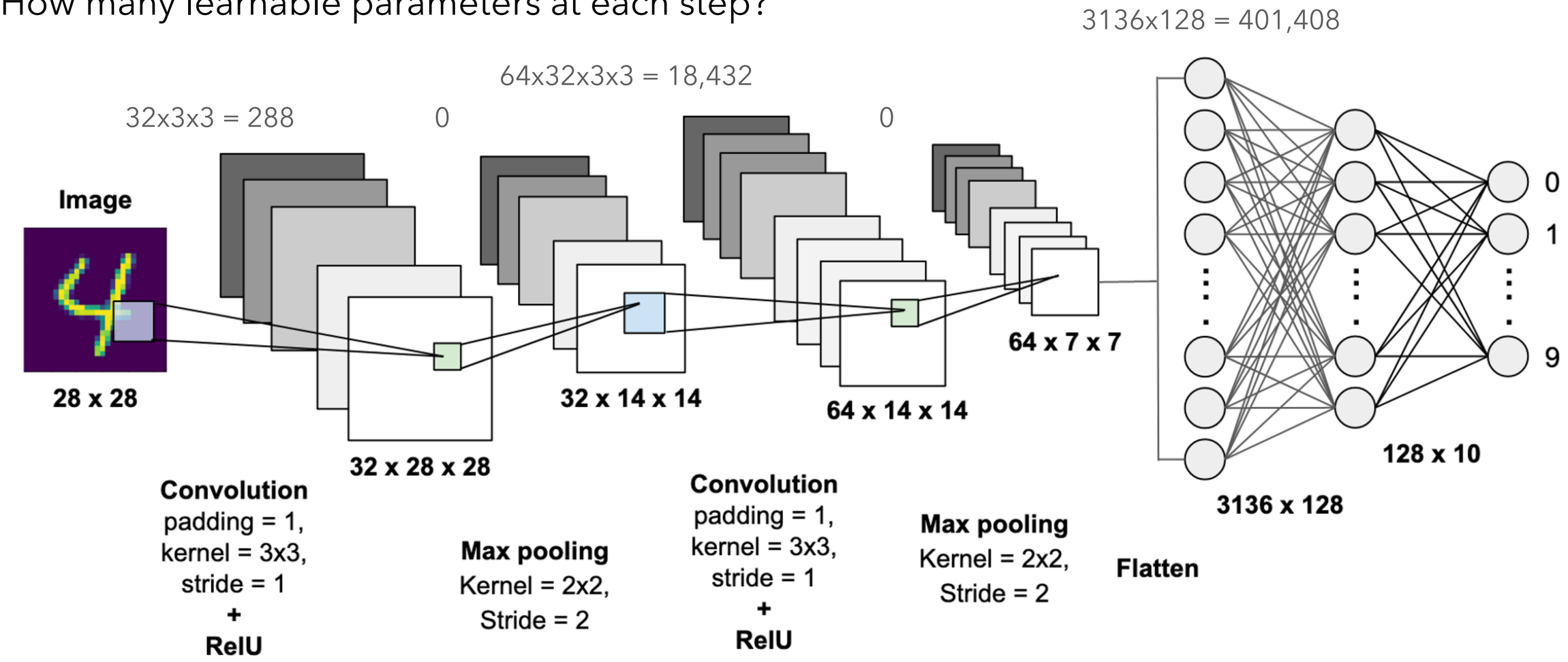
# Convolutional Neural Networks

How many learnable parameters at each step?



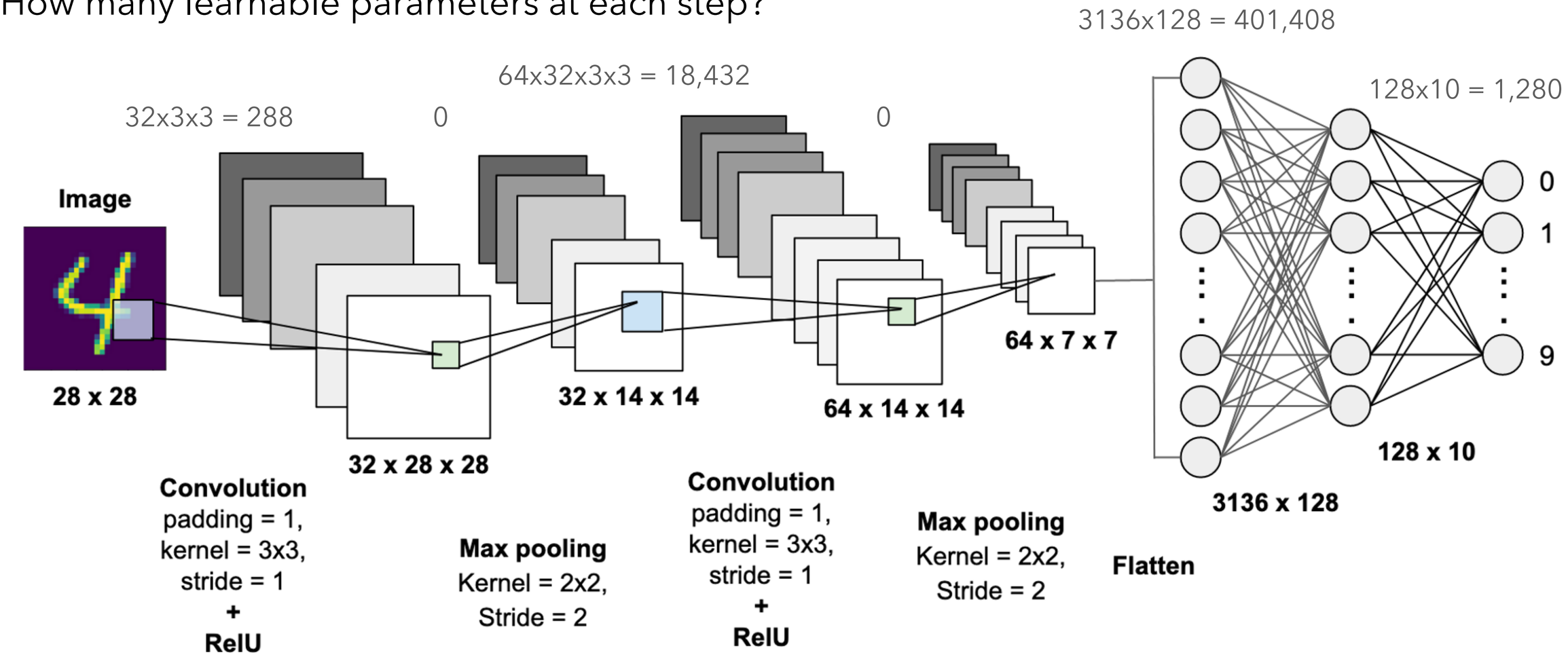
# Convolutional Neural Networks

How many learnable parameters at each step?



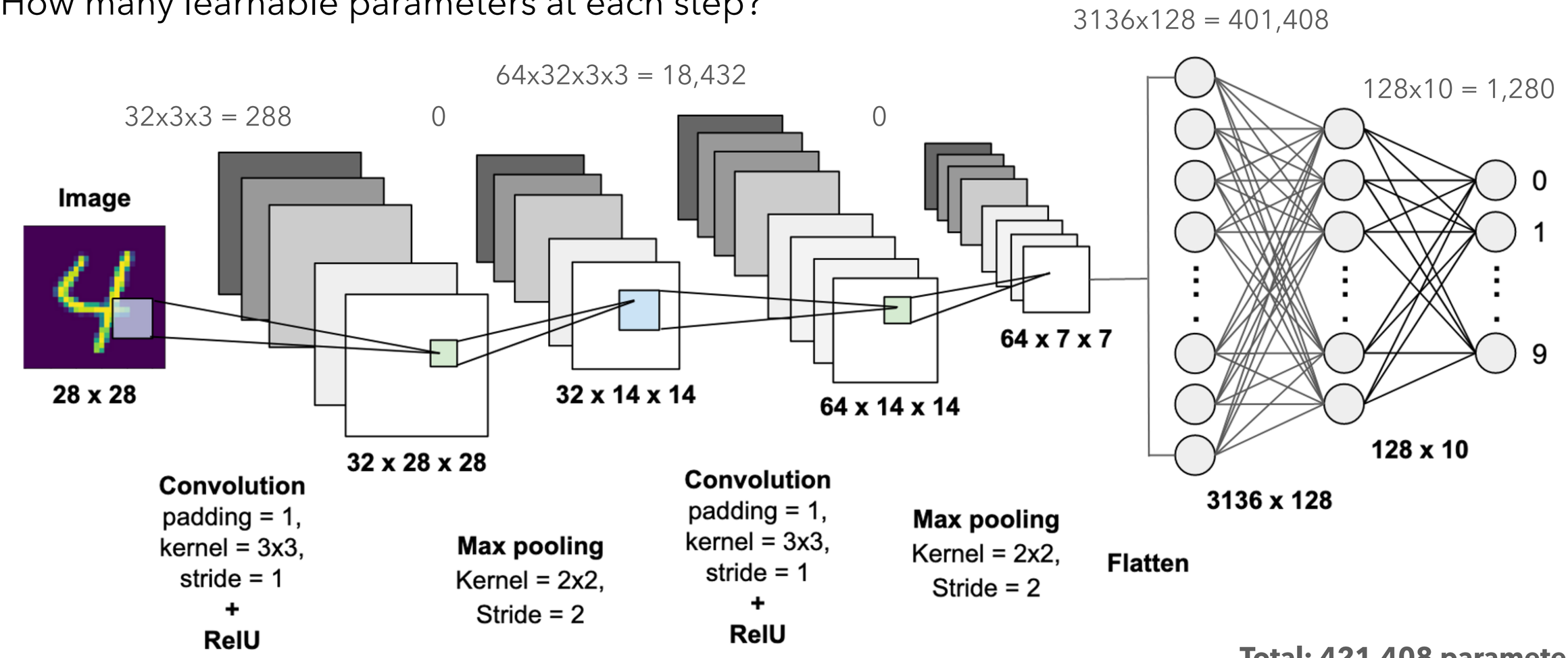
# Convolutional Neural Networks

How many learnable parameters at each step?



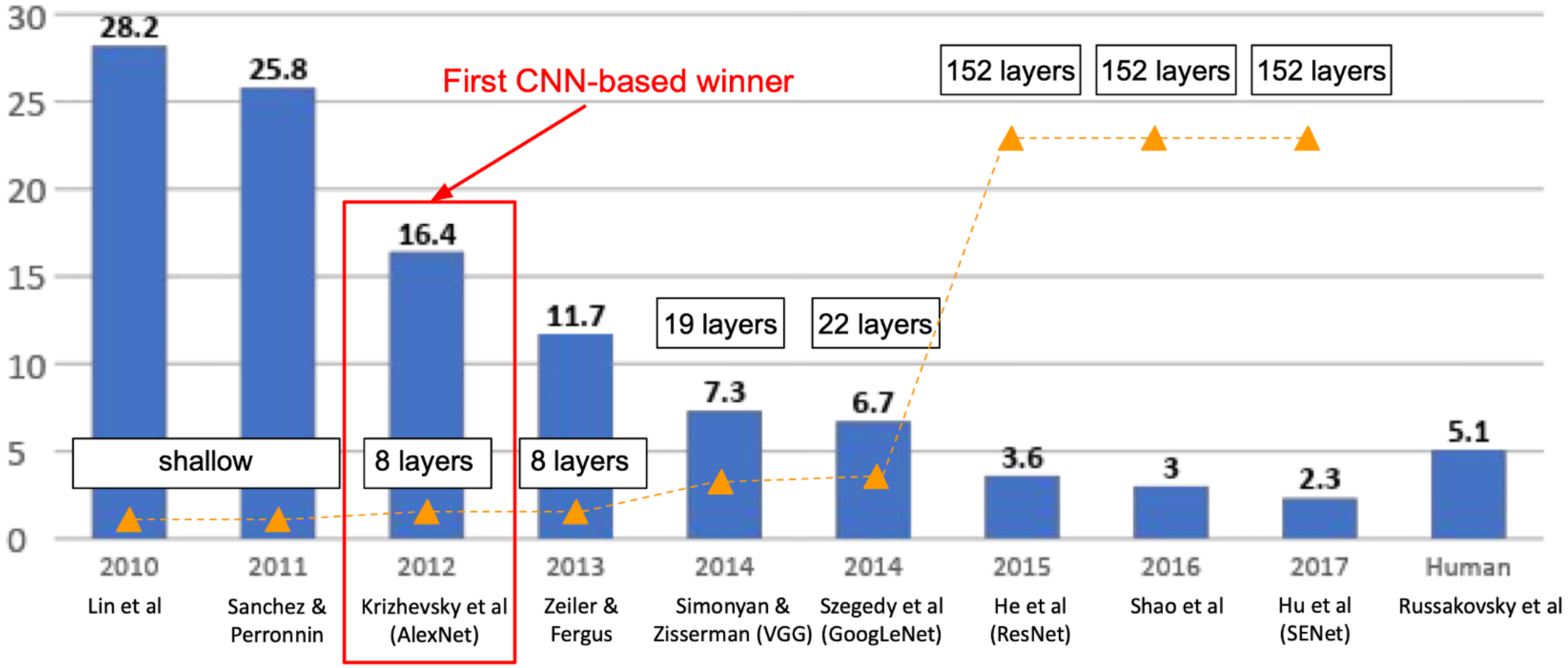
# Convolutional Neural Networks

How many learnable parameters at each step?




**Total: 421,408 parameters**

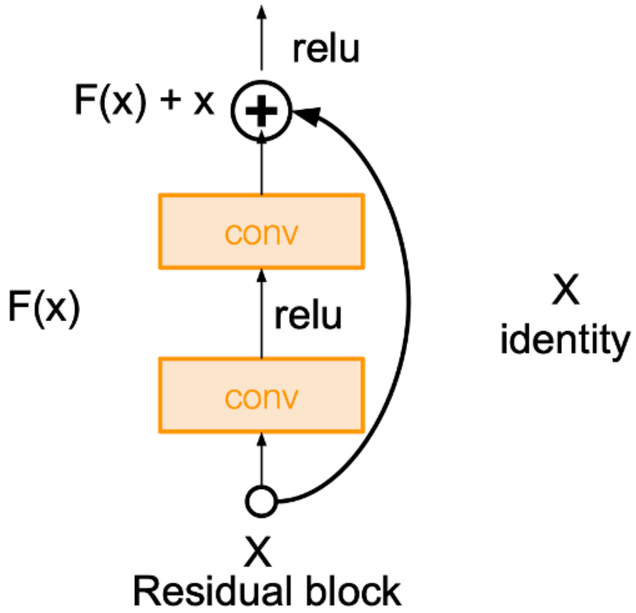
# Benchmarking on ImageNet

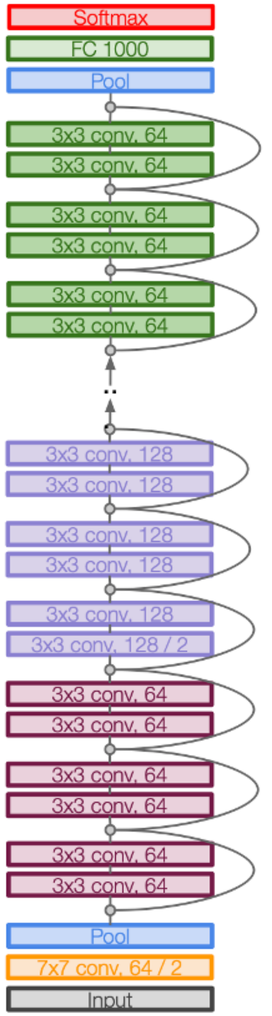


# ResNet (He, et al. 2015)

Key idea:

- Want deeper networks with more parameters, but training signal becomes weak
- Add “skip” connections between layers so that there are shorter paths between early parameters and the final loss function



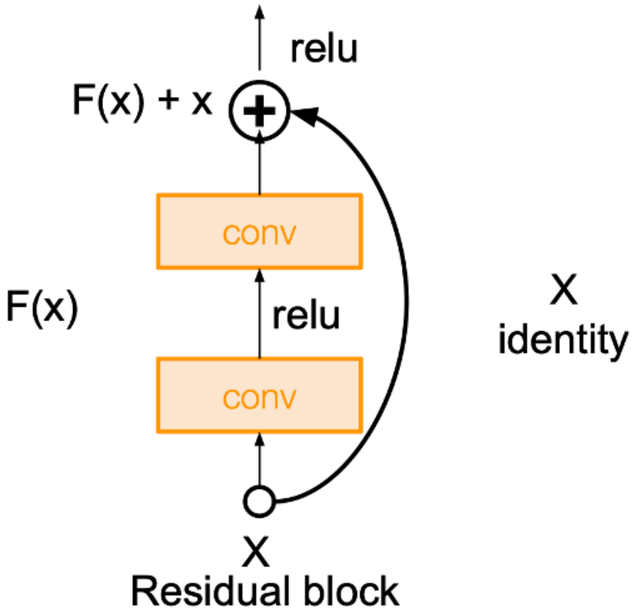
# ResNet (He, et al. 2015)

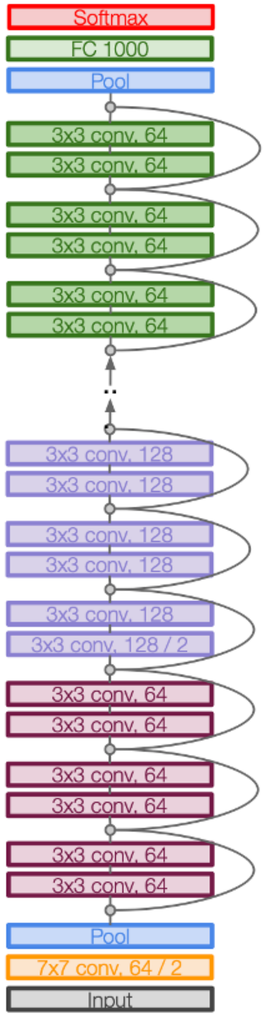
Key idea:

- Want deeper networks with more parameters, but training signal becomes weak
- Add “skip” connections between layers so that there are shorter paths between early parameters and the final loss function

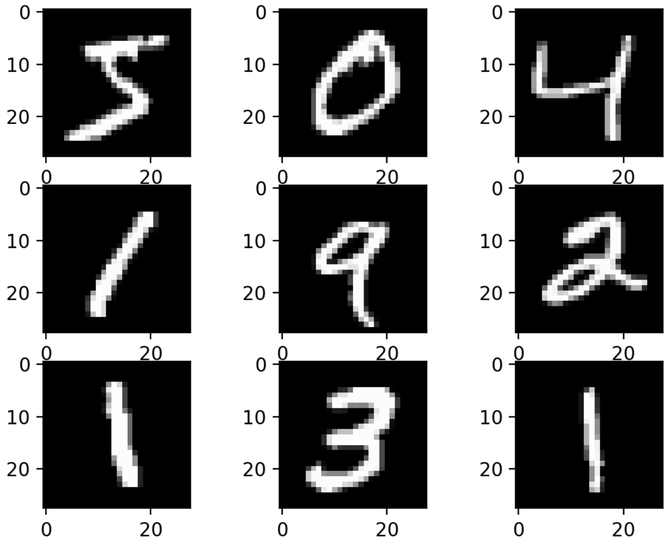
ResNet:

- 152-layer model for ImageNet
- Massive improvement over all previous CNN-based classification models circa 2015



# Many architectural advances have come from two domains



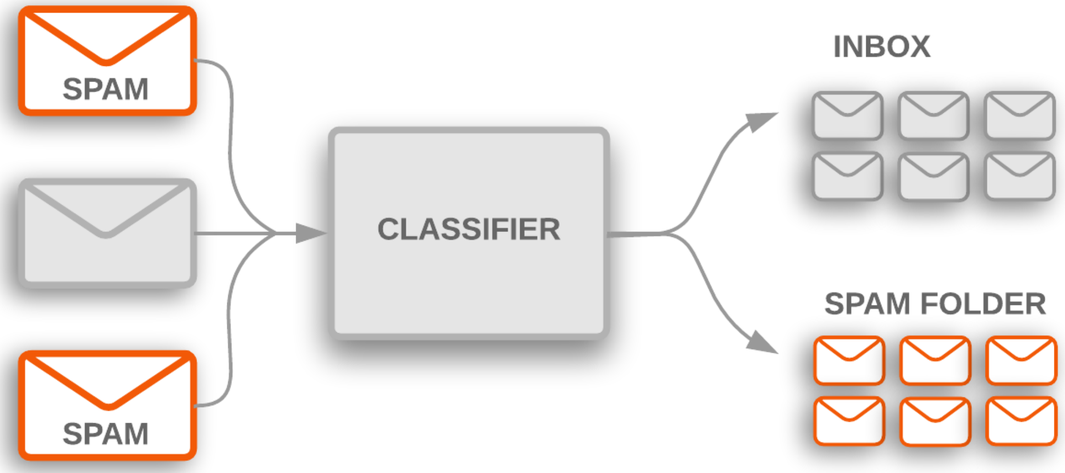
Computer Vision




Natural Language Processing

# Many architectural advances have come from two domains

Computer Vision

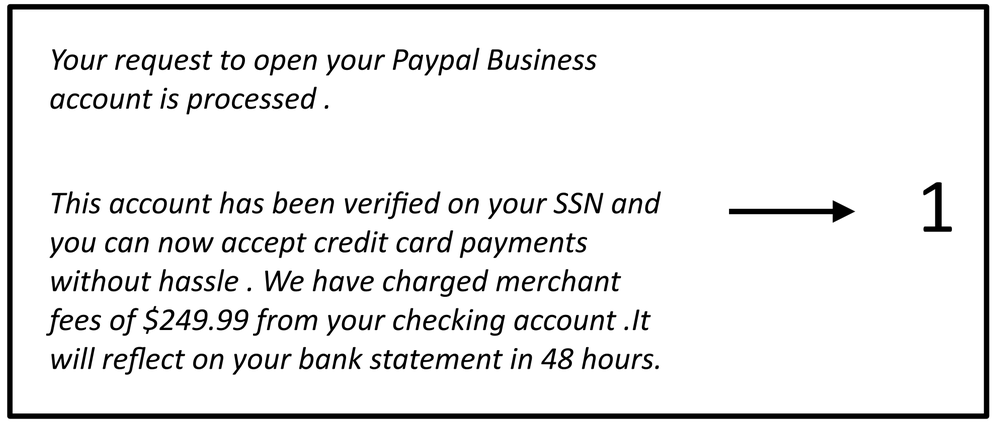



Natural Language Processing

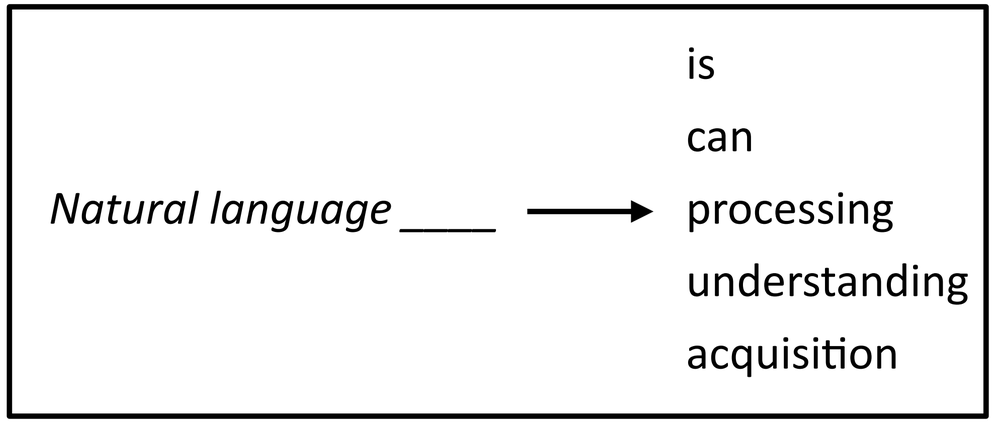
# What tasks does (classic) NLP care about?

- Text classification (e.g., email spam)
- Sequence tagging
- Syntactic and semantic parsing
- Machine translation
- Question answering
- Dialogue (social chat, task-oriented, etc.)
- …

# Many NLP tasks are classification tasks




Spam Classification




Next Word Prediction

# Cloze Task (The Shannon Game)

# Cloze Task (The Shannon Game)

Today,

# Cloze Task (The Shannon Game)

Today, I

# Cloze Task (The Shannon Game)

Today, I went

# Cloze Task (The Shannon Game)

Today, I went to

# Cloze Task (The Shannon Game)

Today, I went to the

# Cloze Task (The Shannon Game)

Today, I went to the store

# Cloze Task (The Shannon Game)

Today, I went to the store and

# Cloze Task (The Shannon Game)

Today, I went to the store and bought

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs.

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain,

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my umbrella,

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my umbrella, and

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my umbrella, and ended

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my umbrella, and ended up

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my umbrella, and ended up getting

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my umbrella, and ended up getting wet

# Cloze Task (The Shannon Game)

Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my umbrella, and ended up getting wet on

# Cloze Task (The Shannon Game)

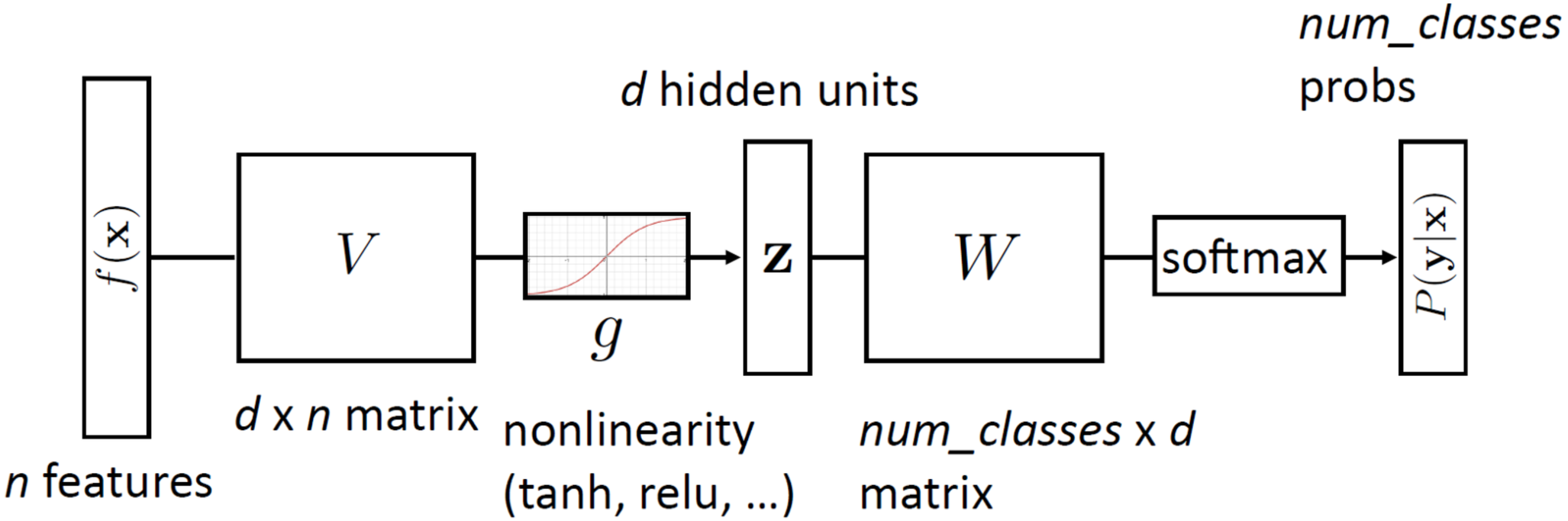
Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my umbrella, and ended up getting wet on the

# Cloze Task (The Shannon Game)

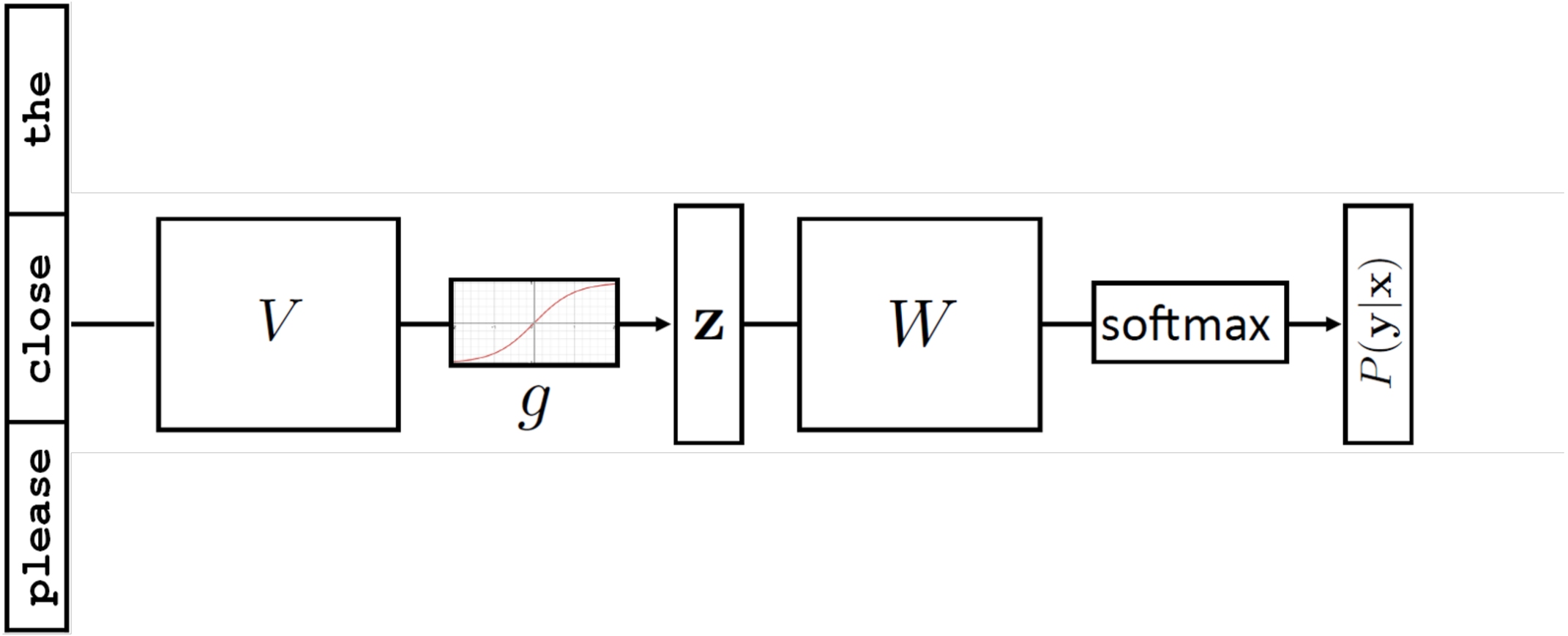
Today, I went to the store and bought some milk and eggs. I knew it was going to rain, but I forgot to take my umbrella, and ended up getting wet on the way.

# Recall: Feedforward Neural Networks

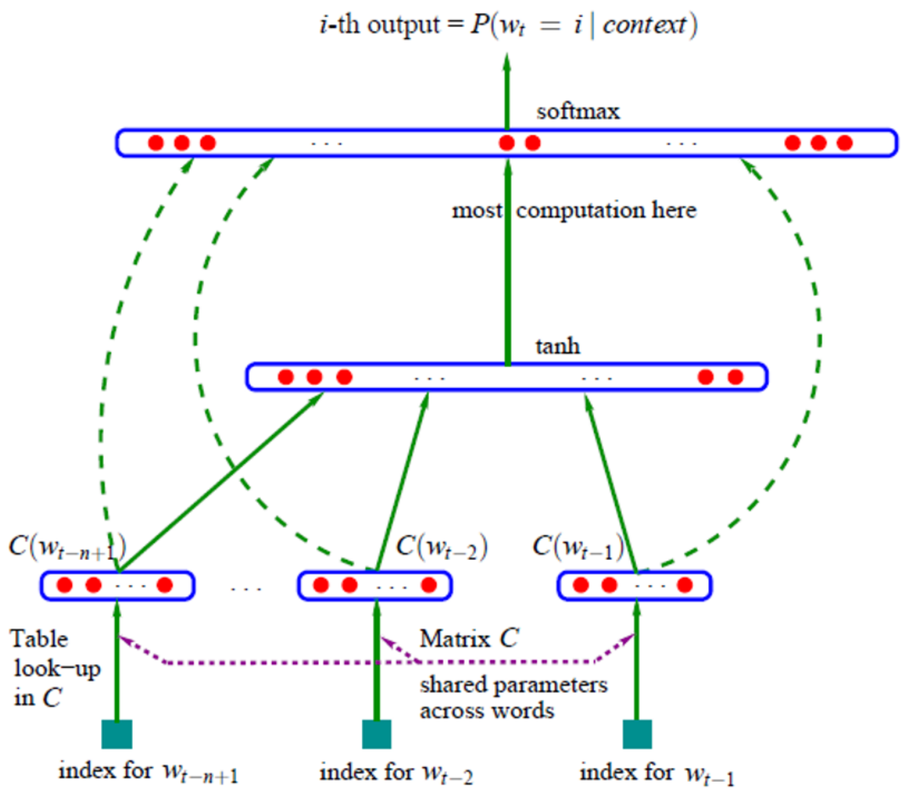
$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(W\,g(V\,f(\mathbf{x})))$$



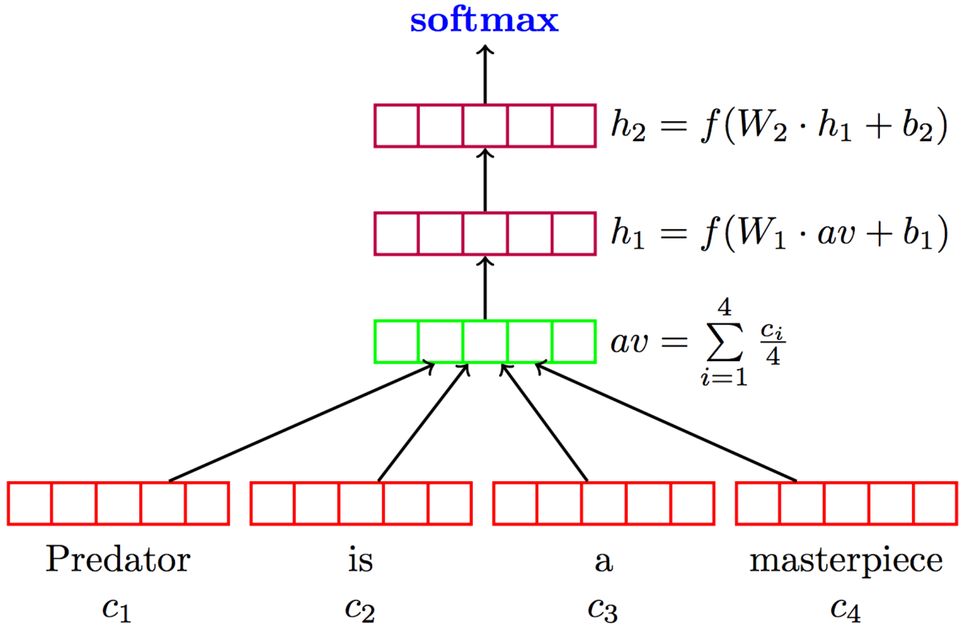
# A Feedforward Language Model



# A Feedforward Language Model

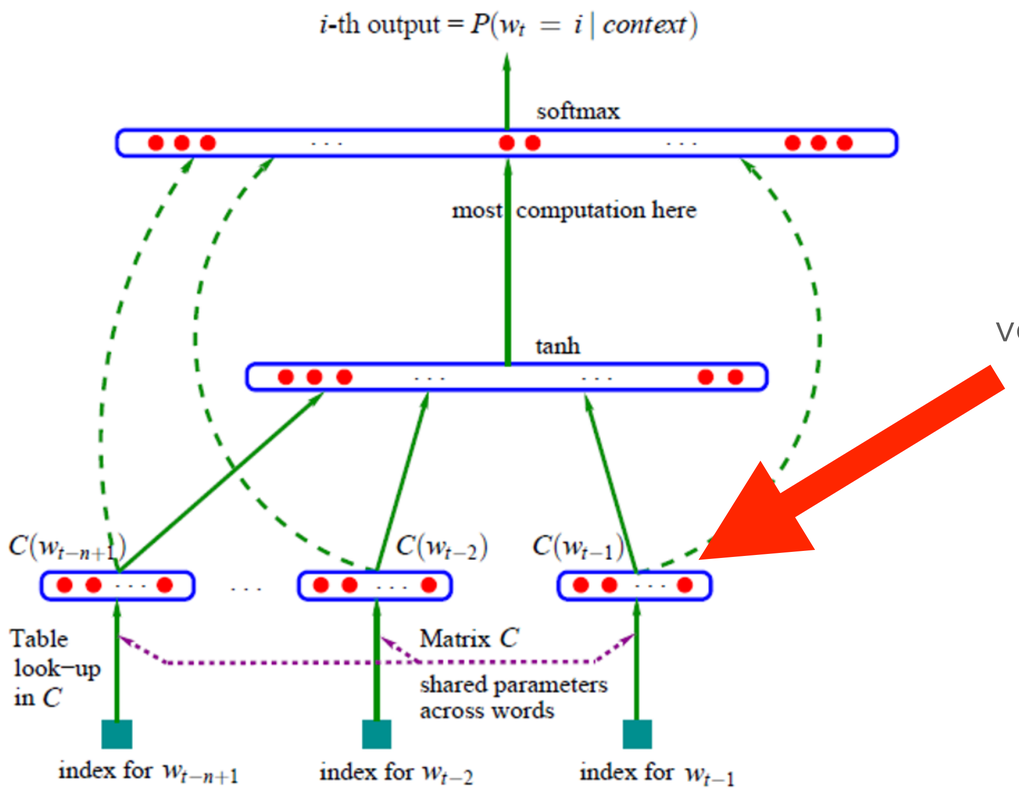



Bengio, et al. 2003

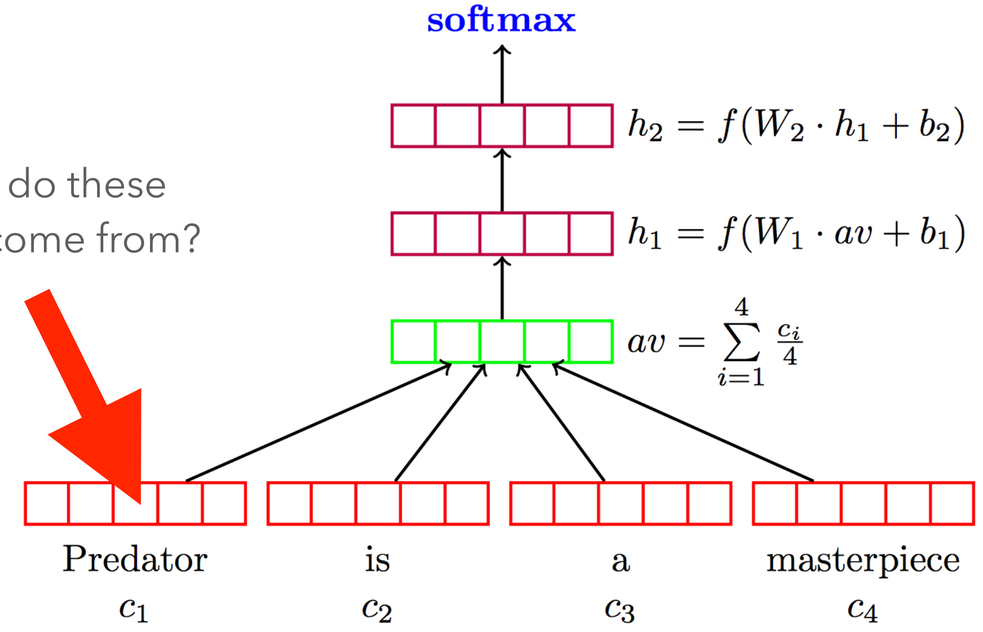



Iyyer, et al. 2015

# A Feedforward Language Model




Where do these vectors come from?




Bengio, et al. 2003

Iyyer, et al. 2015

# Challenge #1: Defining a Vocabulary

**“One-hot" vectors**

Gives us unique representations for each word, but requires a fixed-size vocabulary

$$\boldsymbol{\phi}(\text{a}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{the}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{dog}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$\boldsymbol{\phi}(\text{an}) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cat}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cats}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

# Challenge #1: Defining a Vocabulary

**“One-hot" vectors**

Idea #1: let’s just use the 30K most popular words, replace all of the rest with an <UNK> token

$$\boldsymbol{\phi}(\text{a}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{the}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{dog}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$\boldsymbol{\phi}(\text{an}) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cat}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cats}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

# Challenge #1: Defining a Vocabulary

**“One-hot" vectors**

Idea #1: let’s just use the 30K most popular words, replace all of the rest with an <UNK> token

$$\boldsymbol{\phi}(\text{a}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{the}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{dog}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

But: we don’t want systems to fail when they’re used on documents with out-of-vocabulary words

$$\boldsymbol{\phi}(\text{an}) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cat}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cats}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

# Challenge #1: Defining a Vocabulary

**“One-hot" vectors**

Idea #2: let’s just use characters instead of words

$$\boldsymbol{\phi}(\text{a}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{the}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{dog}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$\boldsymbol{\phi}(\text{an}) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cat}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cats}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

# Challenge #1: Defining a Vocabulary

**“One-hot" vectors**

Idea #2: let’s just use characters instead of words

$$\boldsymbol{\phi}(\text{a}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{the}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{dog}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

But: there are actually a lot of characters (¶¡™ºª•¶£), and we end up with very long sequences

$$\boldsymbol{\phi}(\text{an}) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cat}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cats}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

# Challenge #1: Defining a Vocabulary

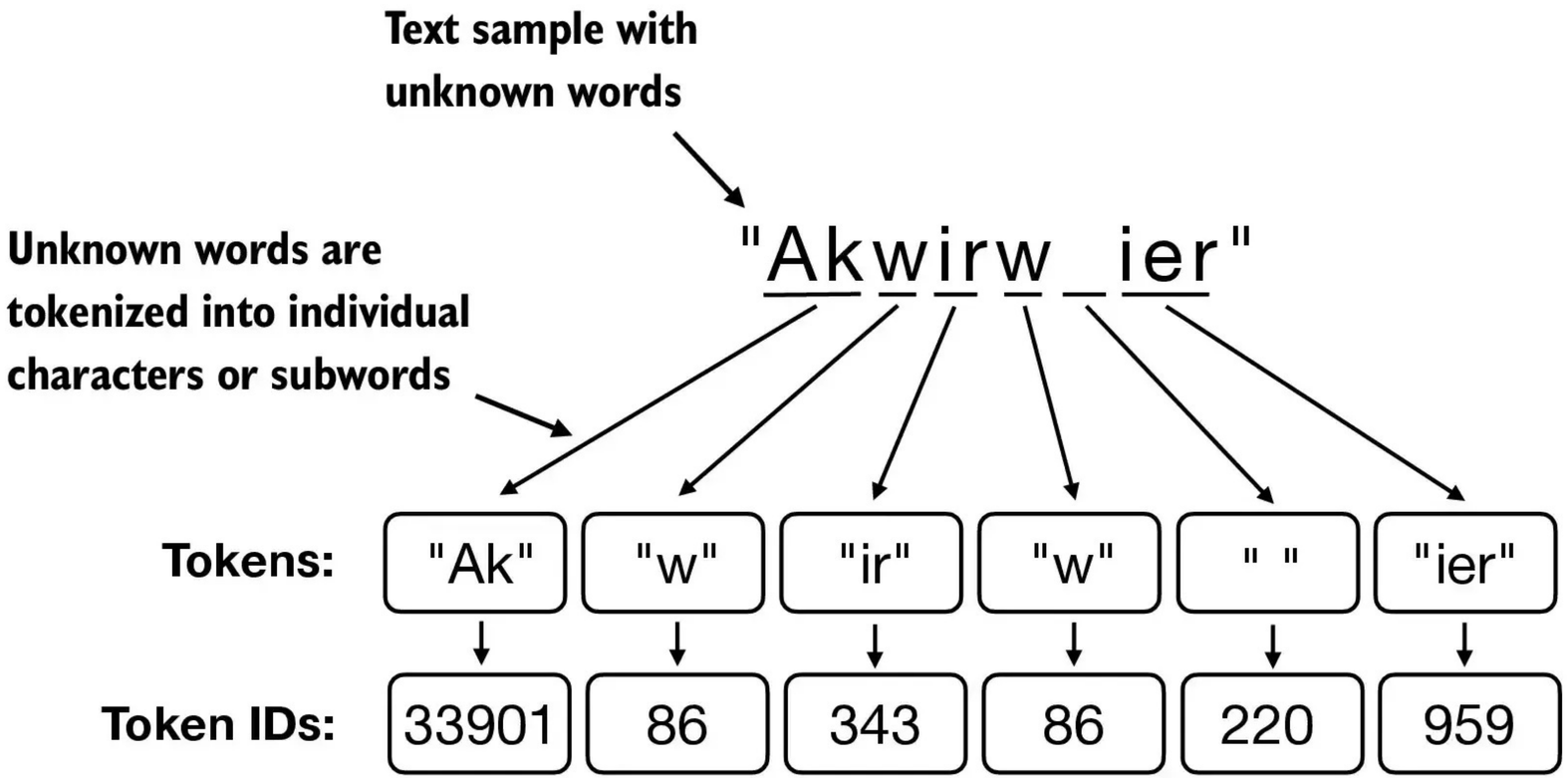
**“One-hot" vectors**

Idea #3: let’s use subword tokens as atomic units instead of words

$$\boldsymbol{\phi}(\text{a}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{the}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{dog}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$\boldsymbol{\phi}(\text{an}) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cat}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cats}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

# Subword Tokens



# Byte Pair Encoding

Algorithm:

- Choose a target vocabulary size (e.g., 30K)
- Start with a small fixed number of tokens in your vocabulary: [“a”, “b”, “c”…]
- Find the most popular sequences which aren’t in your vocabulary and gradually add them until the vocabulary reaches its target size

# Byte Pair Encoding

**Example:**

This is the cat that I think is cute.

**Vocabulary:**

[a,b,c,…z]

# Byte Pair Encoding

**Example:**

This is the cat that I think is cute.

This is **th**e cat **th**at I **th**ink is cute.

**Vocabulary:**

[a,b,c,…z]

# Byte Pair Encoding

**Example:**

This is the cat that I think is cute.

This is **th**e cat **th**at I **th**ink is cute.

**Vocabulary:**

[a,b,c,…z]

[a,b,c,…z, th]

# Byte Pair Encoding

**Example:**

This is the cat that I think is cute.

This is **th**e cat **th**at I **th**ink is cute.

Th**is is** the cat that I think **is** cute.

**Vocabulary:**

[a,b,c,…z]

[a,b,c,…z, th]

# Byte Pair Encoding

**Example:**

This is the cat that I think is cute.

This is **th**e cat **th**at I **th**ink is cute.

Th**is is** the cat that I think **is** cute.

**Vocabulary:**

[a,b,c,…z]

[a,b,c,…z, th]

[a,b,c,…z, th, is]

# Byte Pair Encoding

**Example:**

This is the cat that I think is cute.

This is **th**e cat **th**at I **th**ink is cute.

Th**is is** the cat that I think **is** cute.

T**hi**s is the cat that I t**hi**nk is cute.

**Vocabulary:**

[a,b,c,…z]

[a,b,c,…z, th]

[a,b,c,…z, th, is]

# Byte Pair Encoding

**Example:**

This is the cat that I think is cute.

This is **th**e cat **th**at I **th**ink is cute.

Th**is is** the cat that I think **is** cute.

T**hi**s is the cat that I t**hi**nk is cute.

**Vocabulary:**

[a,b,c,…z]

[a,b,c,…z, th]

[a,b,c,…z, th, is]

[a,b,c,…z, th, is, hi]

# Byte Pair Encoding

**Example:**

This is the cat that I think is cute.

This is **th**e cat **th**at I **th**ink is cute.

Th**is is** the cat that I think **is** cute.

T**hi**s is the cat that I t**hi**nk is cute.

…

**Vocabulary:**

[a,b,c,…z]

[a,b,c,…z, th]

[a,b,c,…z, th, is]

[a,b,c,…z, th, is, hi]

…

# Challenge #1: Defining a Vocabulary

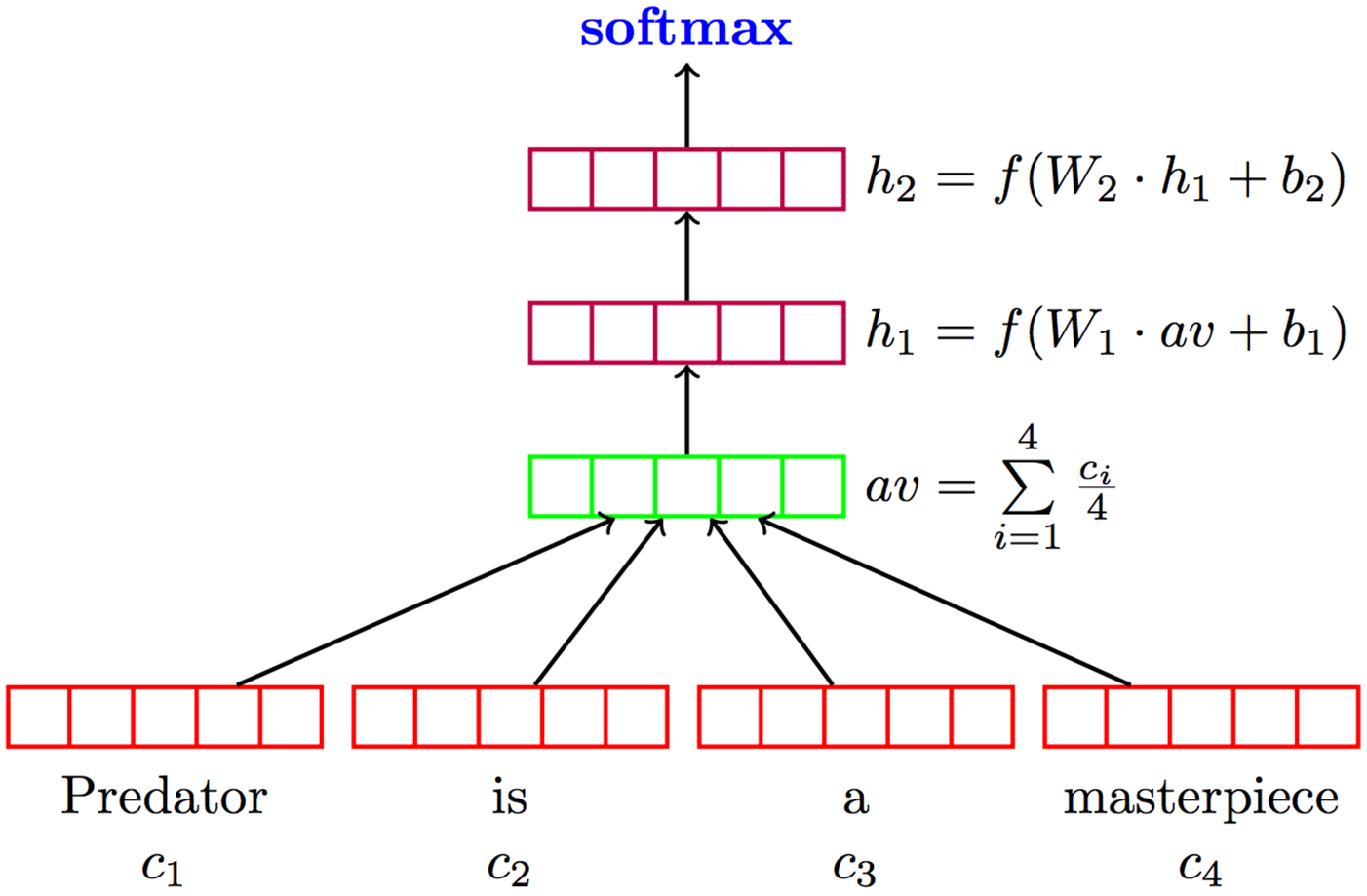
**“One-hot" vectors**

Idea #3: let’s use subword tokens as atomic units instead of words

$$\boldsymbol{\phi}(\text{a}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{th}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cat}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

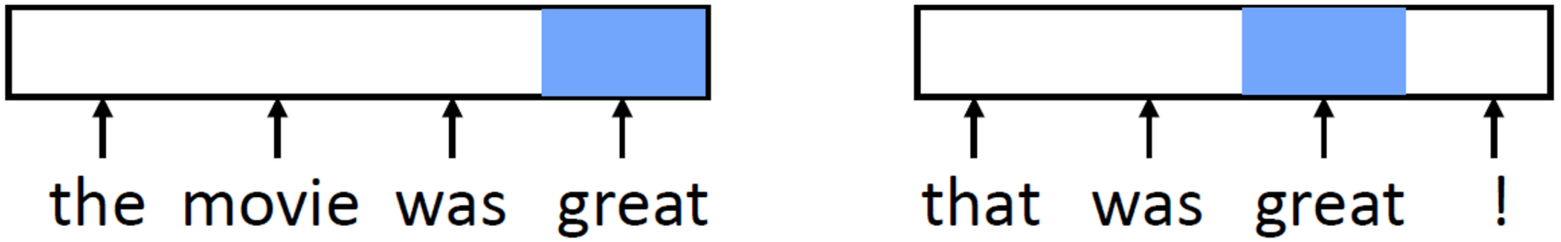
$$\boldsymbol{\phi}(\text{b}) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{is}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \boldsymbol{\phi}(\text{cats}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

# Challenge #2: Variable Sequence Lengths



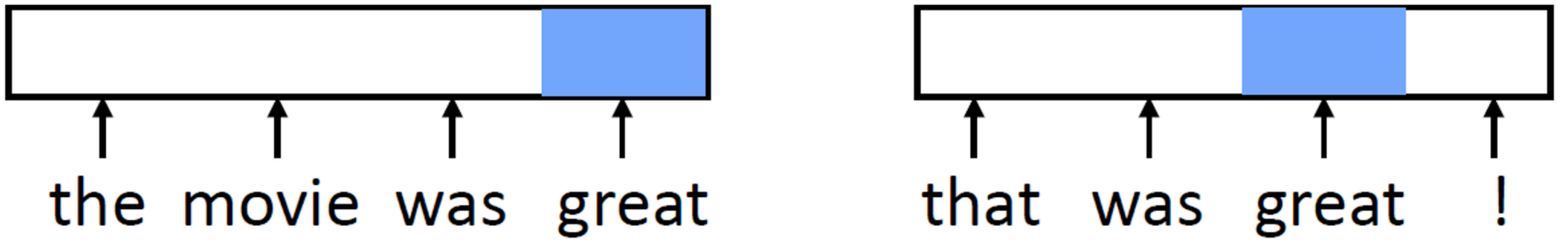
# Challenge #2: Variable Sequence Lengths

- Feedforward NNs can’t handle variable length input
- Each position in the feature vector has fixed semantics



# Challenge #2: Variable Sequence Lengths

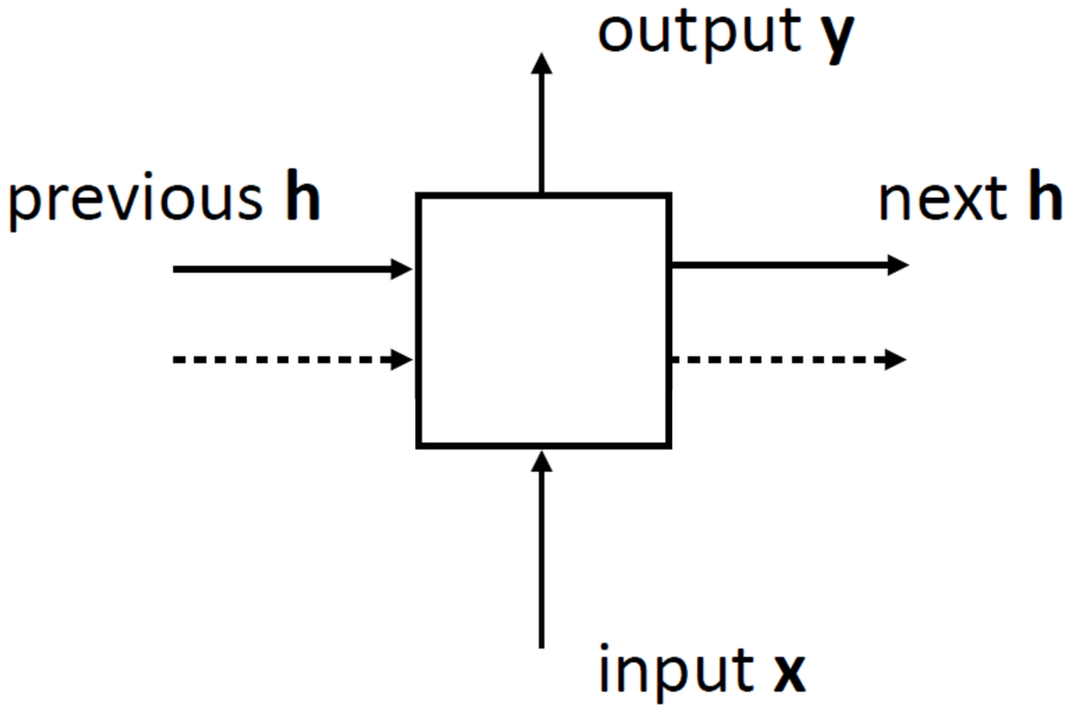
- Feedforward NNs can't handle variable length input
- Each position in the feature vector has fixed semantics




- These don't look related (*great* is in two different orthogonal subspaces)
- Instead, we need to process each word in a uniform way, while still leveraging the context

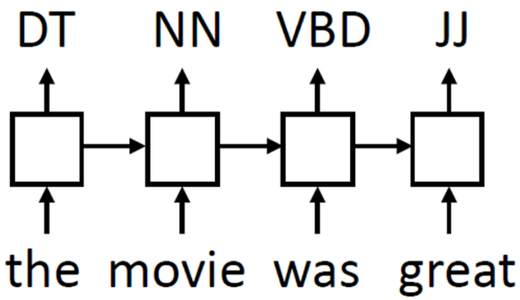
# General RNN Approach

Cell that takes some input **x**, has some hidden state **h**, updates that hidden state, and produces an output **y** (all vector-valued)



# RNN Uses

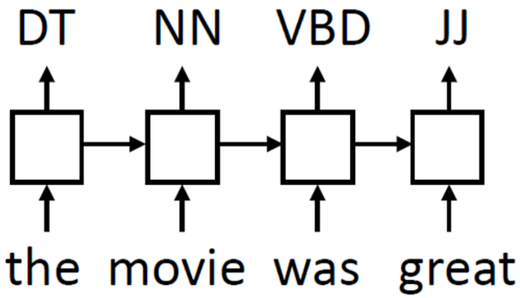
Transducer: make some prediction for each element in a sequence




output **y** = score for each tag, then softmax

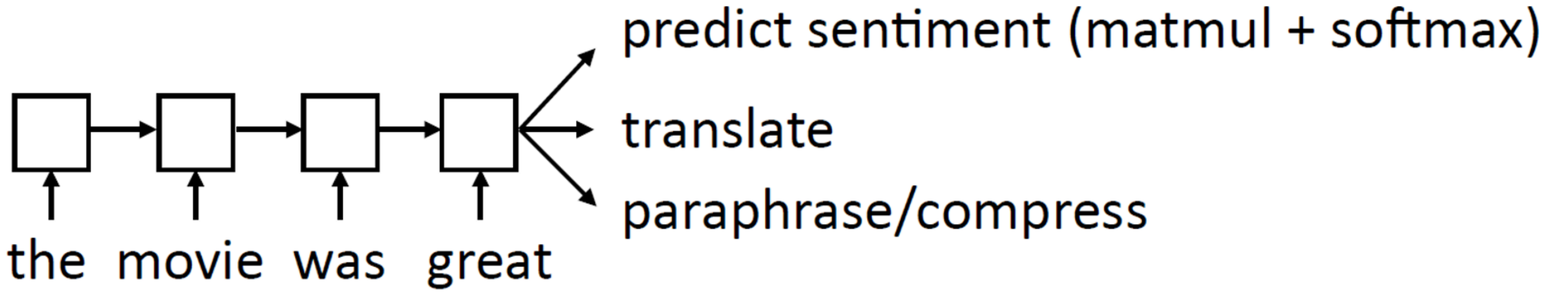
# RNN Uses

Transducer: make some prediction for each element in a sequence




output **y** = score for each tag, then softmax

Acceptor/encoder: encode a sequence into a fixed-size vector and use that for some purpose



# Basic RNNs

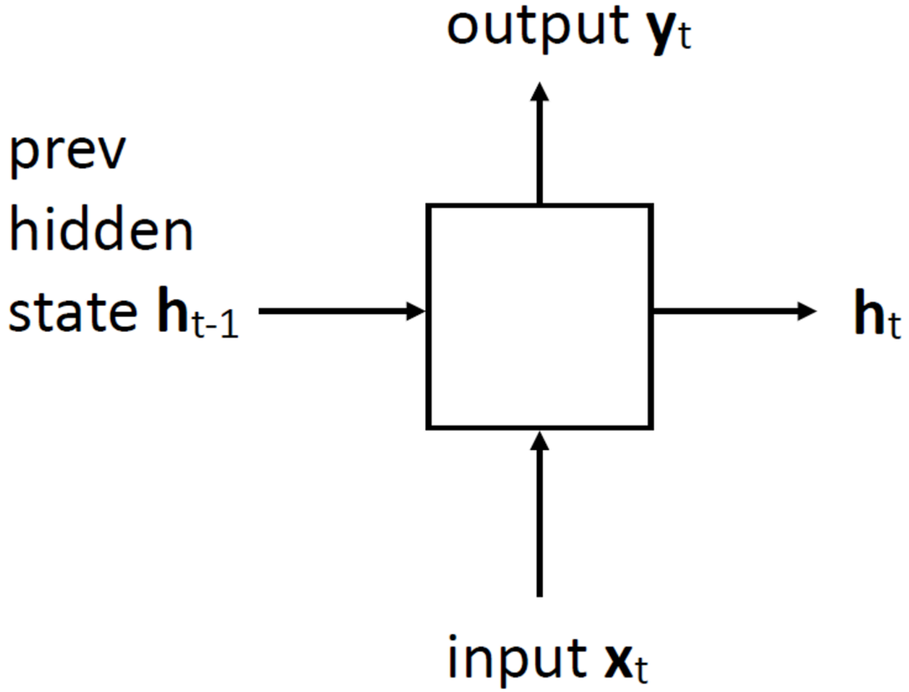
Hidden state update:

$$h_t = \tanh(Wx_t + Vh_{t-1} + b_h)$$

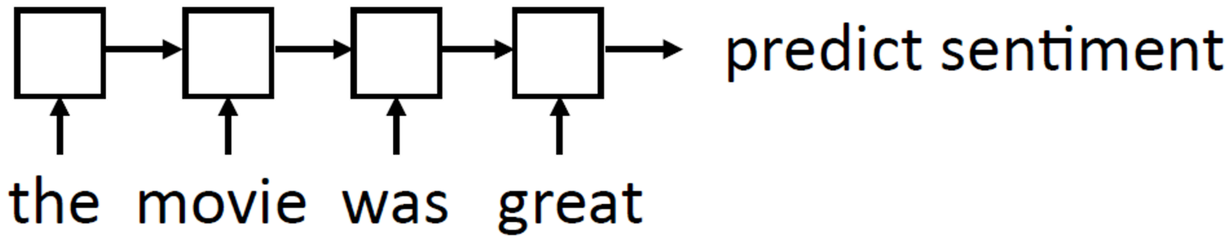
Prediction head:

$$y_t = \tanh(Uh_t + b_y)$$

*Parameters (W, V, U) are shared across timesteps!*

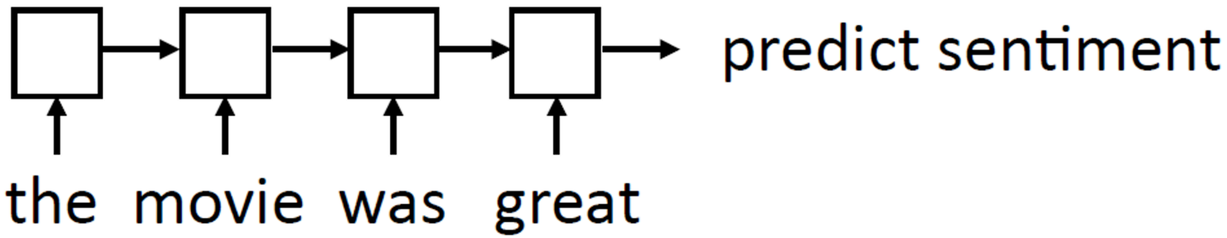


# Training RNNs




- “Backpropagation through time”: build the network as one big computation graph

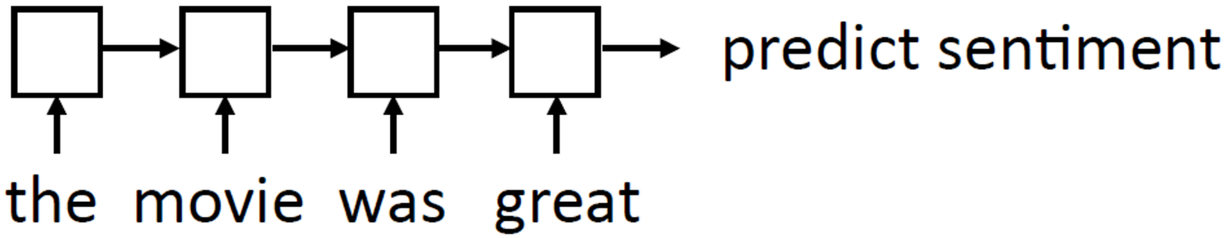
# Training RNNs




- "Backpropagation through time": build the network as one big computation graph
- RNNs potentially need to remember information for a long time:

  *"It was my favorite movie of 2015, even though it wasn't without problems."* → **Positive**

# Training RNNs



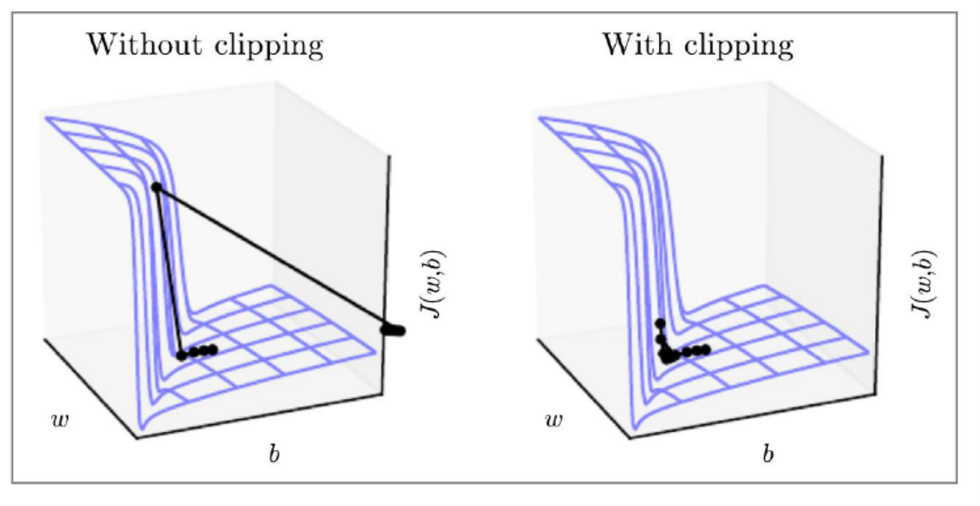

- "Backpropagation through time": build the network as one big computation graph
- RNNs potentially need to remember information for a long time:

  *"It was my favorite movie of 2015, even though it wasn't without problems."* → **Positive**

- The "correct" parameter update is to do a better job of remembering the sentiment of favorite

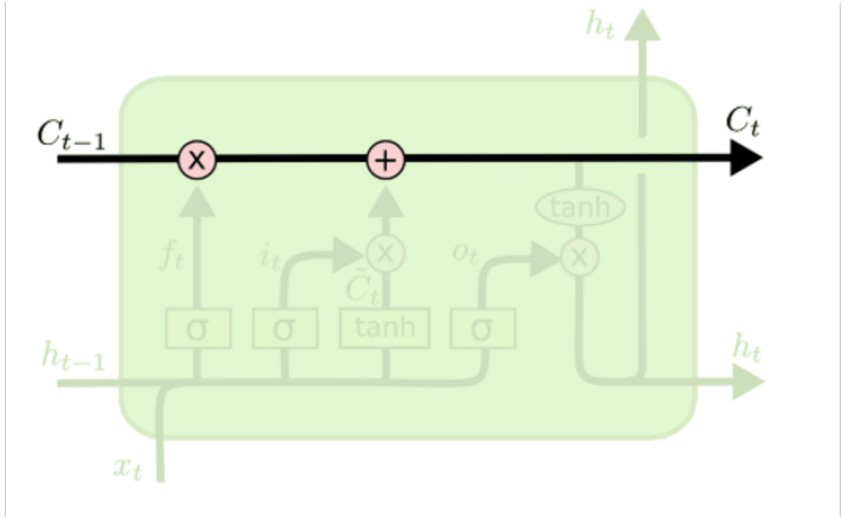
# Core Issue: Information Decay

- Vanishing gradients: contribution of earlier inputs decays over time, leading models to primarily capture local information
- Exploding gradients: gradients can also become too large, leading to overshooting/jumping around the parameter space (common fix: gradient clipping)
- The core issue is that it's difficult for an RNN to learn to preserve information over many timesteps

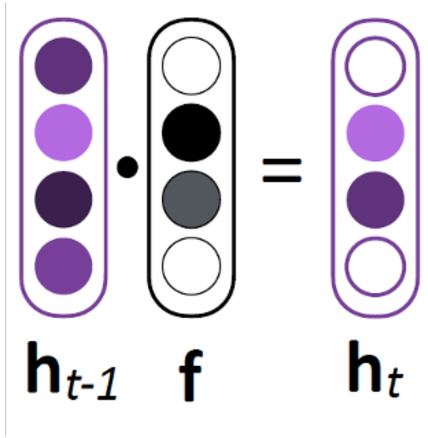


# Key Idea: Propagated State

- Information decays in RNNs because it gets multiplied each time step
- Fix: add a channel (the “cell state”) that by default just gets propagated (the “conveyer belt”)
- Gates make explicit decisions about what to add / forget from this channel




Cell State




Gating

# RNNs

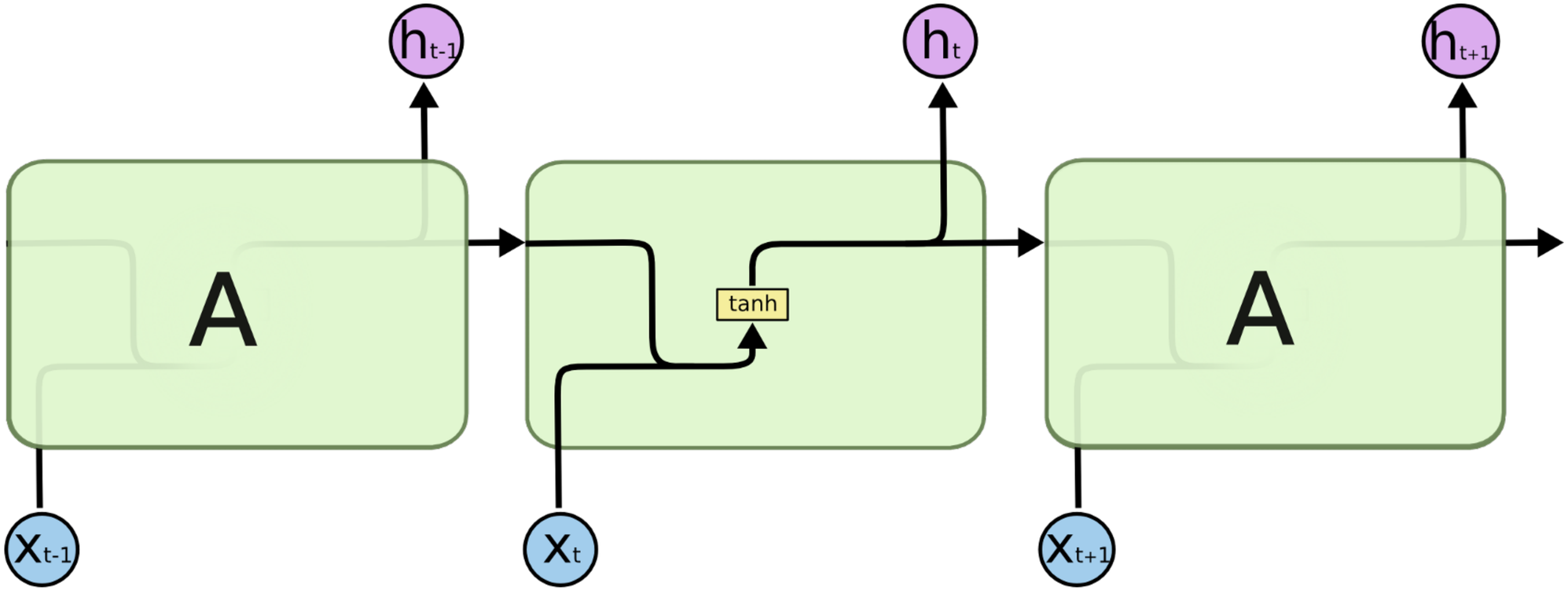



Image from Chris Olah

# LSTMs

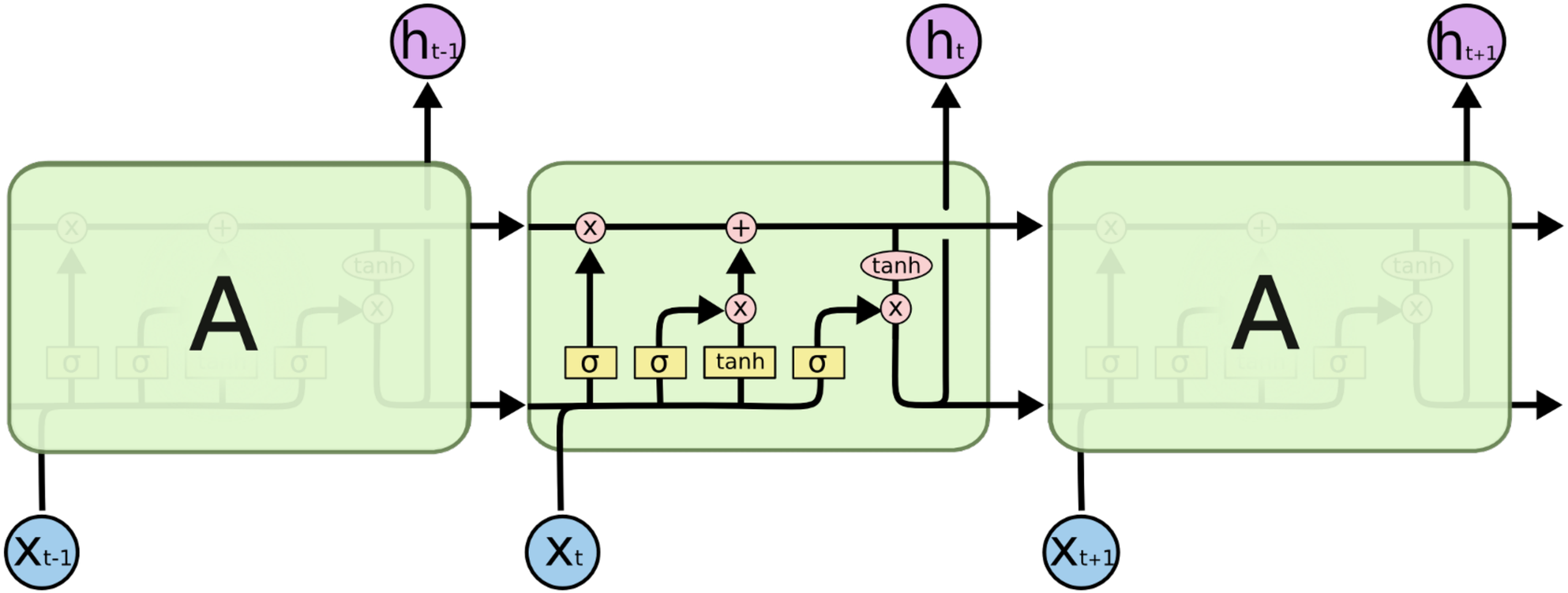



Image from Chris Olah

# LSTMs

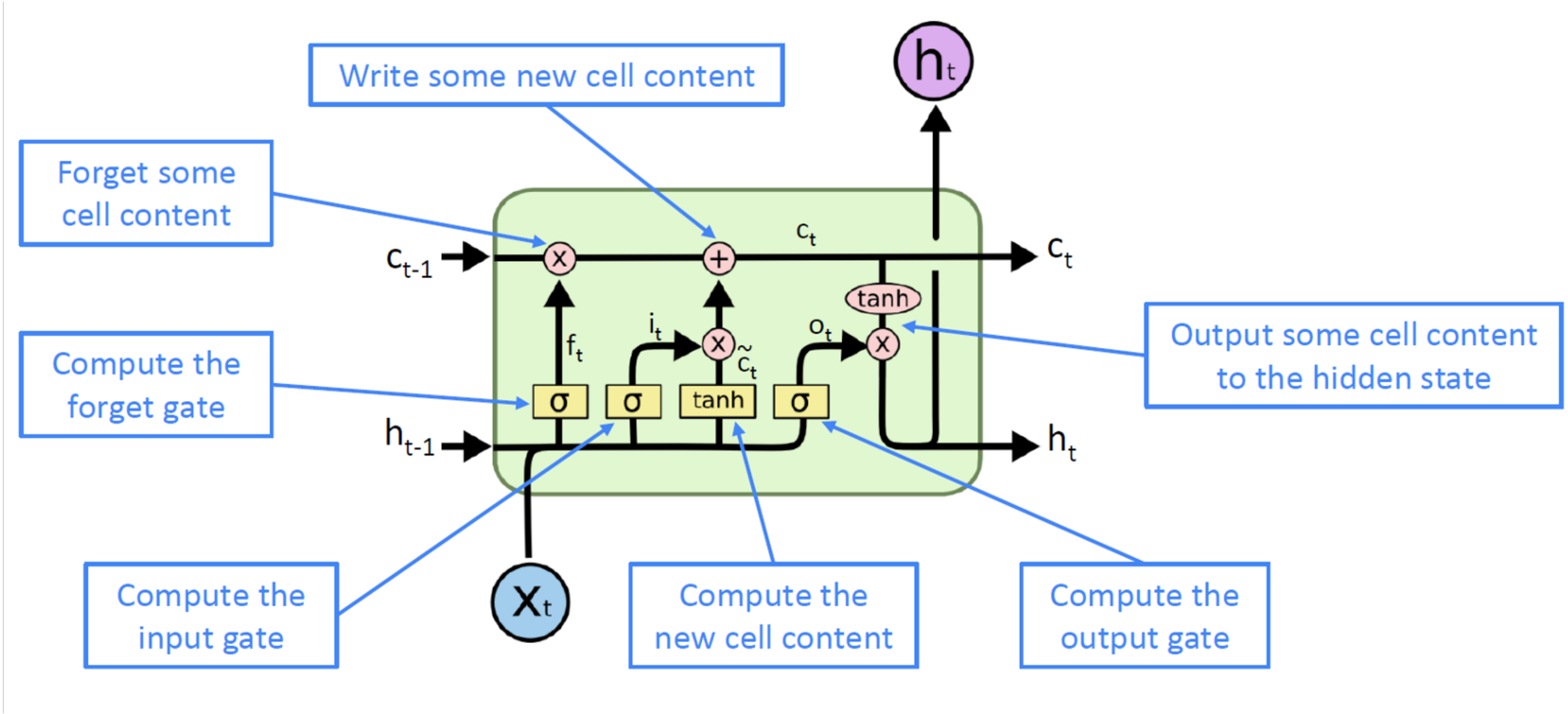



Image from Chris Olah

# How do LSTMs propagate information?

- Ignoring the hidden state entirely:
  - Gives us a feedforward layer over each token

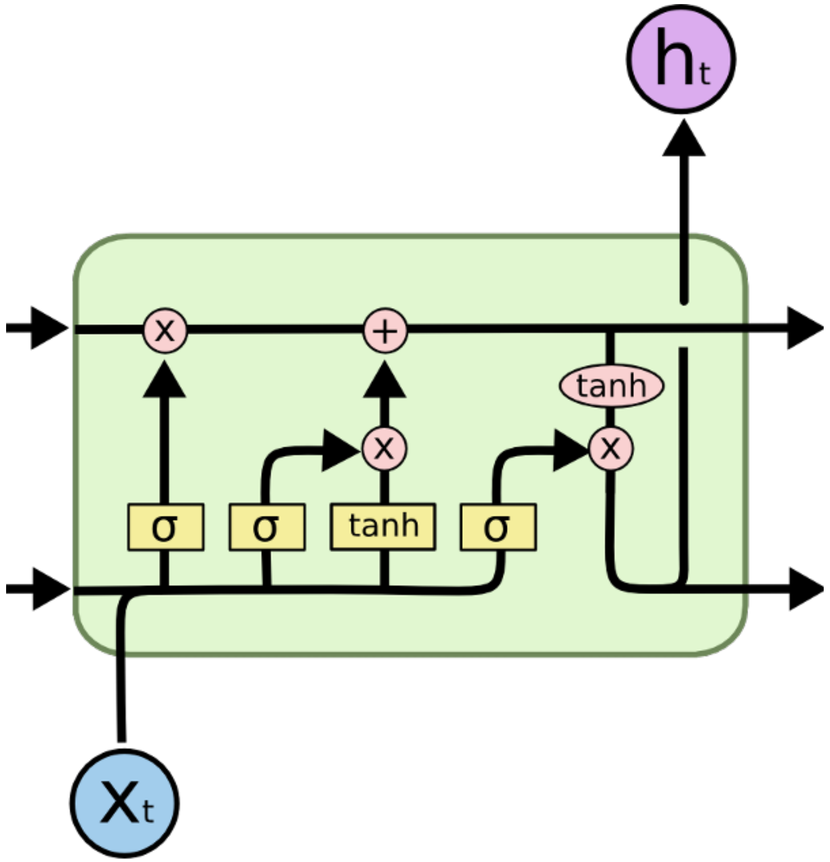



Image from Chris Olah

# How do LSTMs propagate information?

- Ignoring the hidden state entirely:
  - Gives us a feedforward layer over each token
- Ignoring input:
  - Lets us discard stopwords

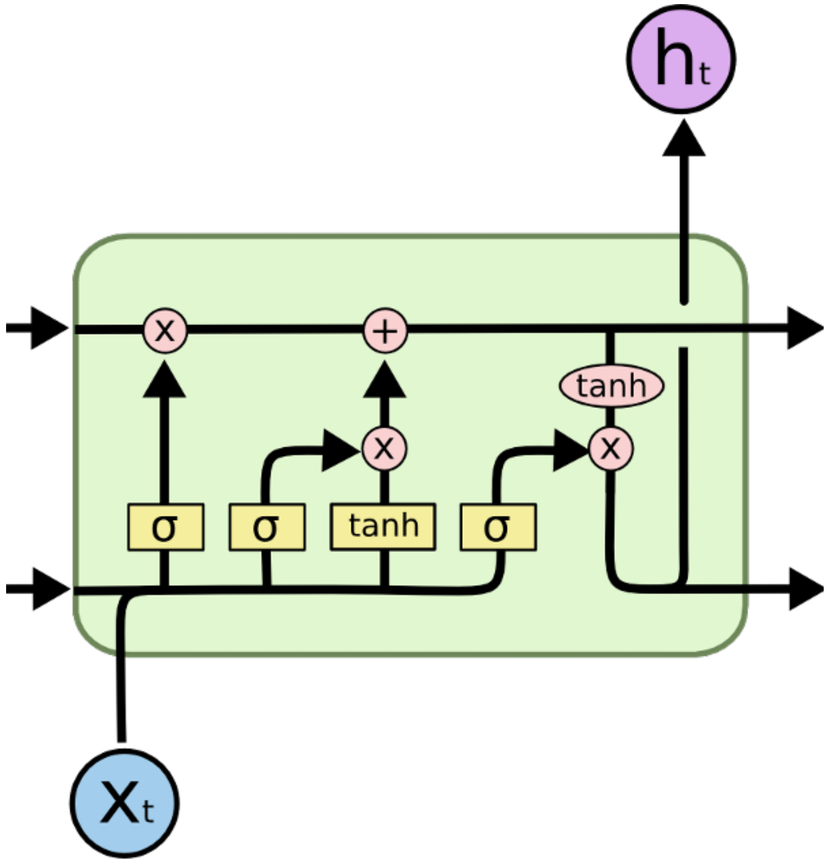



Image from Chris Olah

# How do LSTMs propagate information?

- Ignoring the hidden state entirely:
  - Gives us a feedforward layer over each token
- Ignoring input:
  - Lets us discard stopwords
- Summing input:
  - Lets us compute a bag-of-words representation

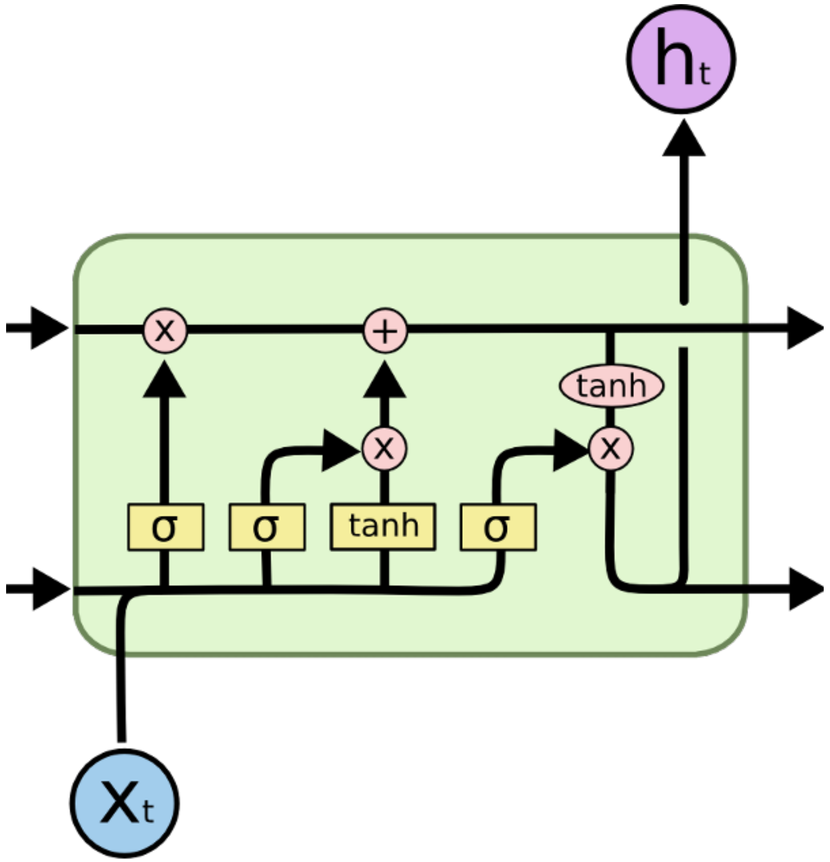



Image from Chris Olah

# How do LSTMs propagate information?

- Ignoring the hidden state entirely:
  - Gives us a feedforward layer over each token
- Ignoring input:
  - Lets us discard stopwords
- Summing input:
  - Lets us compute a bag-of-words representation

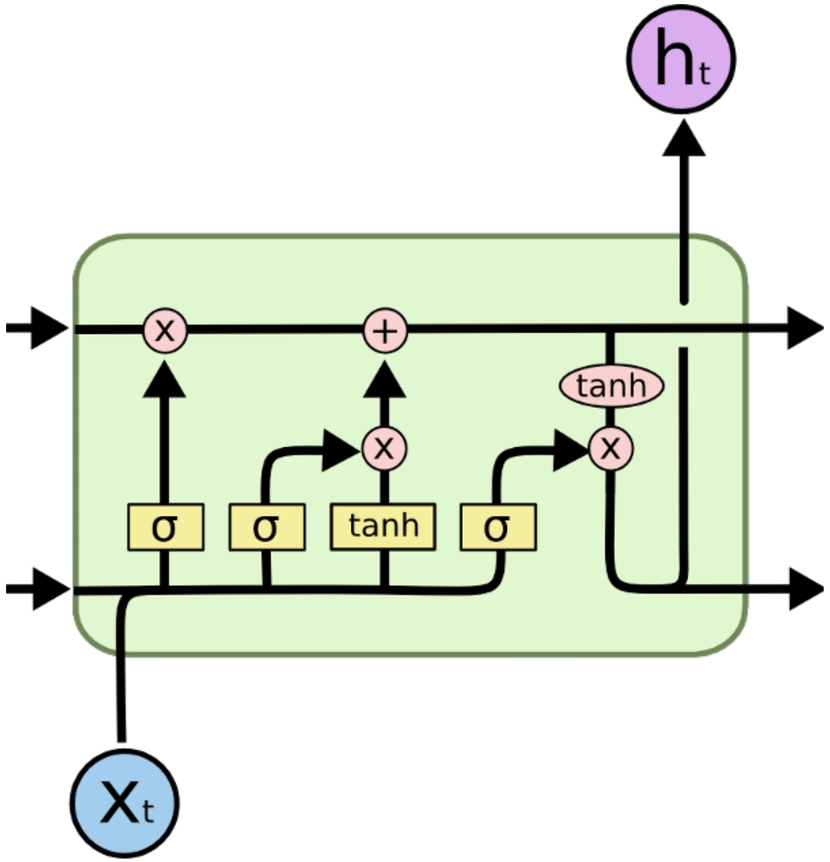
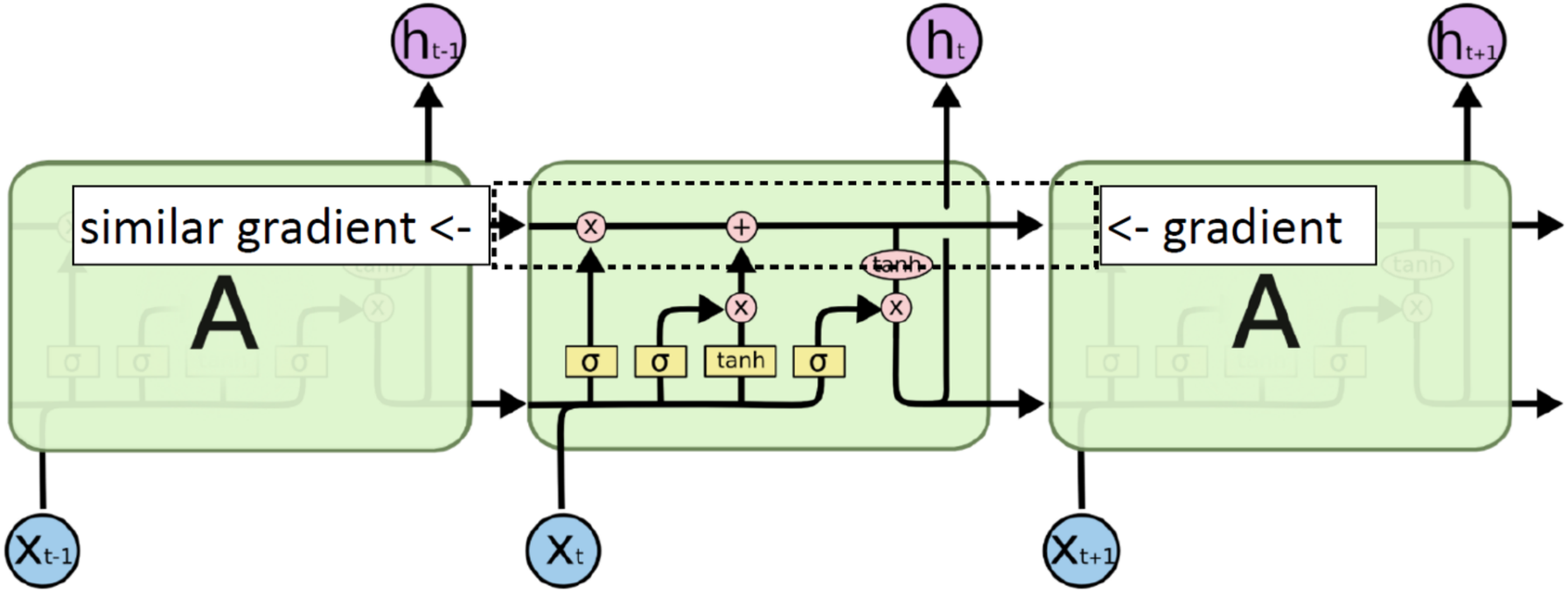



Image from Chris Olah

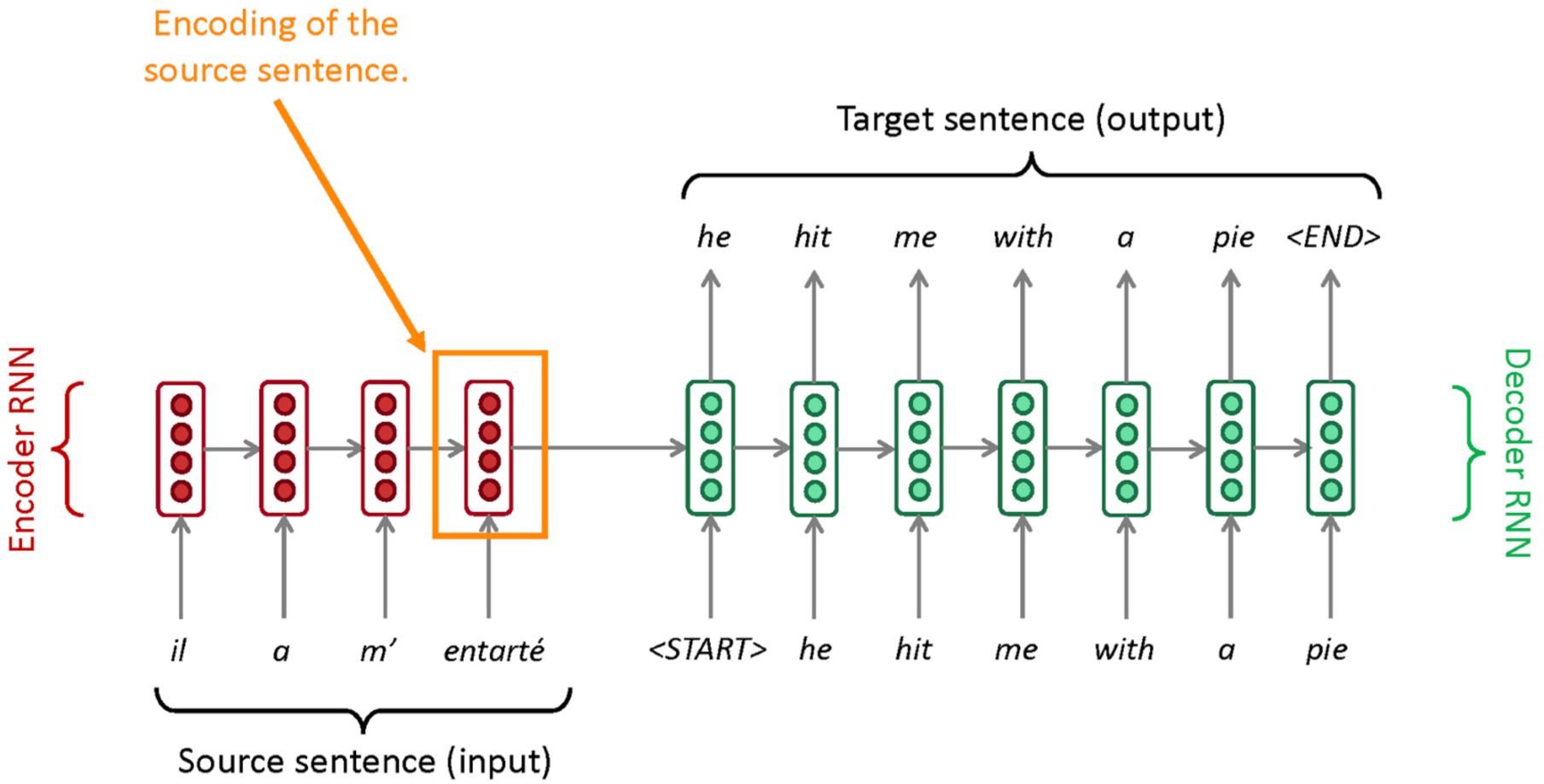
# What about the gradients?



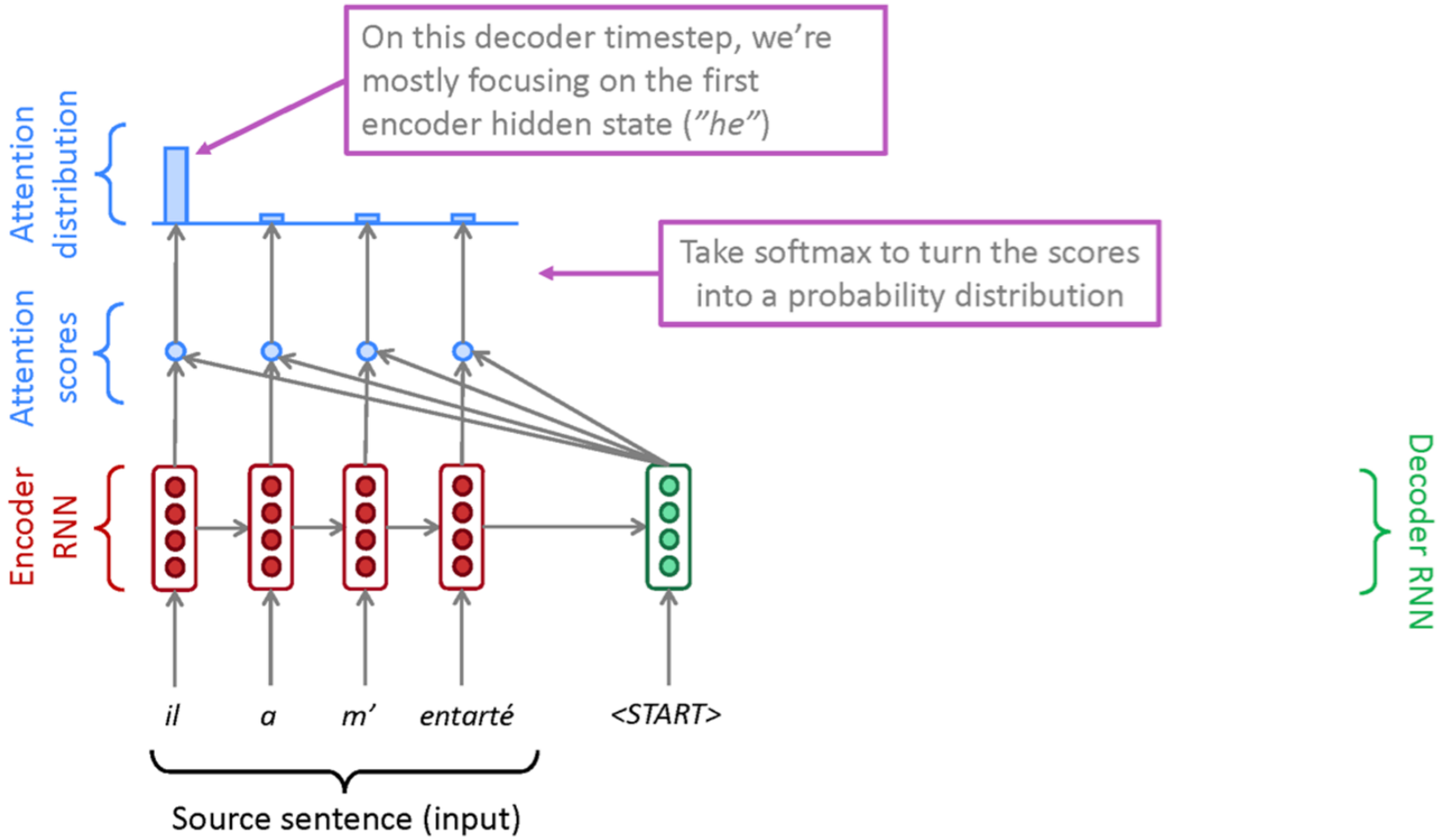

Gradient still diminishes, but in a more controlled way.

Image from Chris Olah

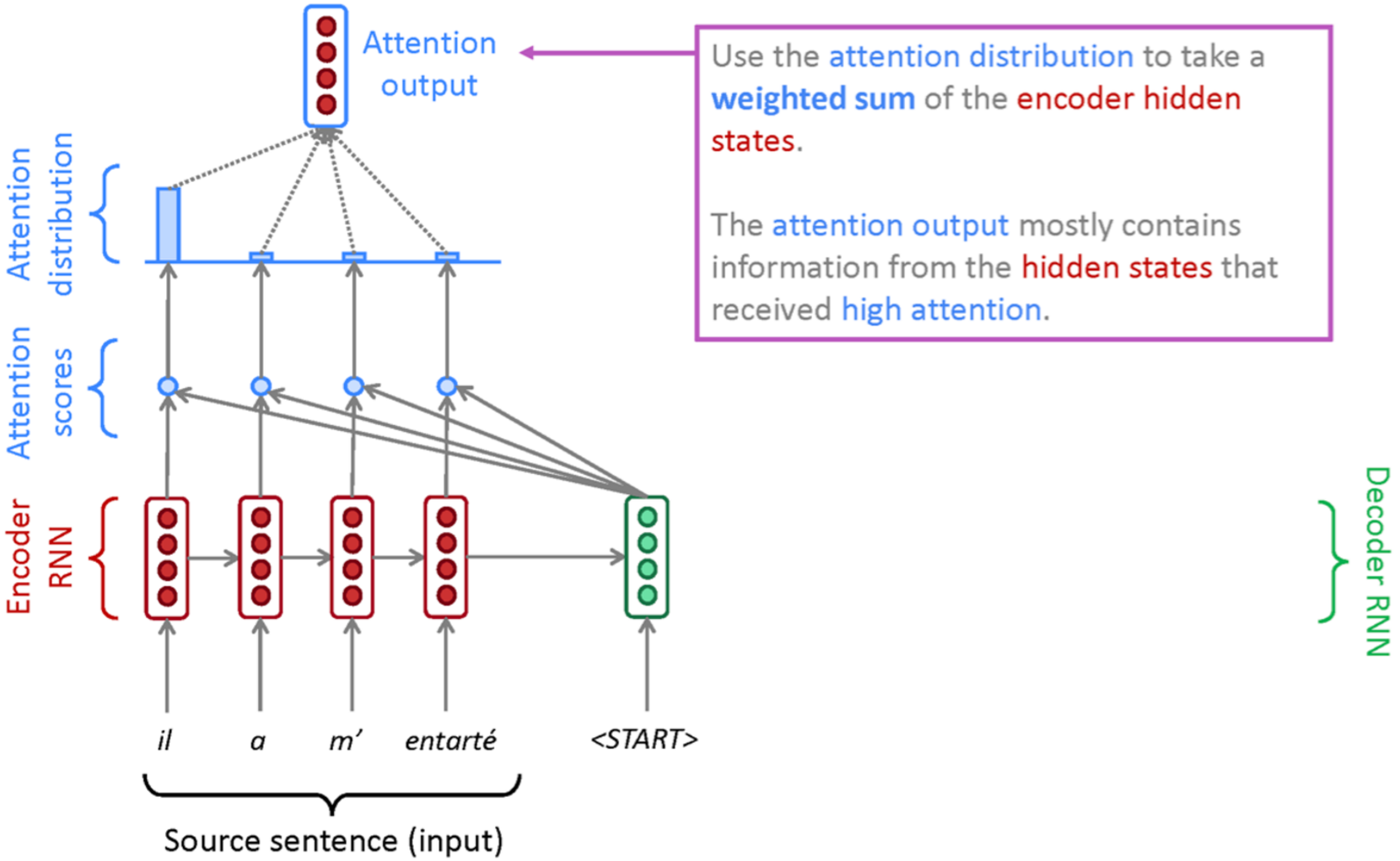
# The Bottleneck Problem



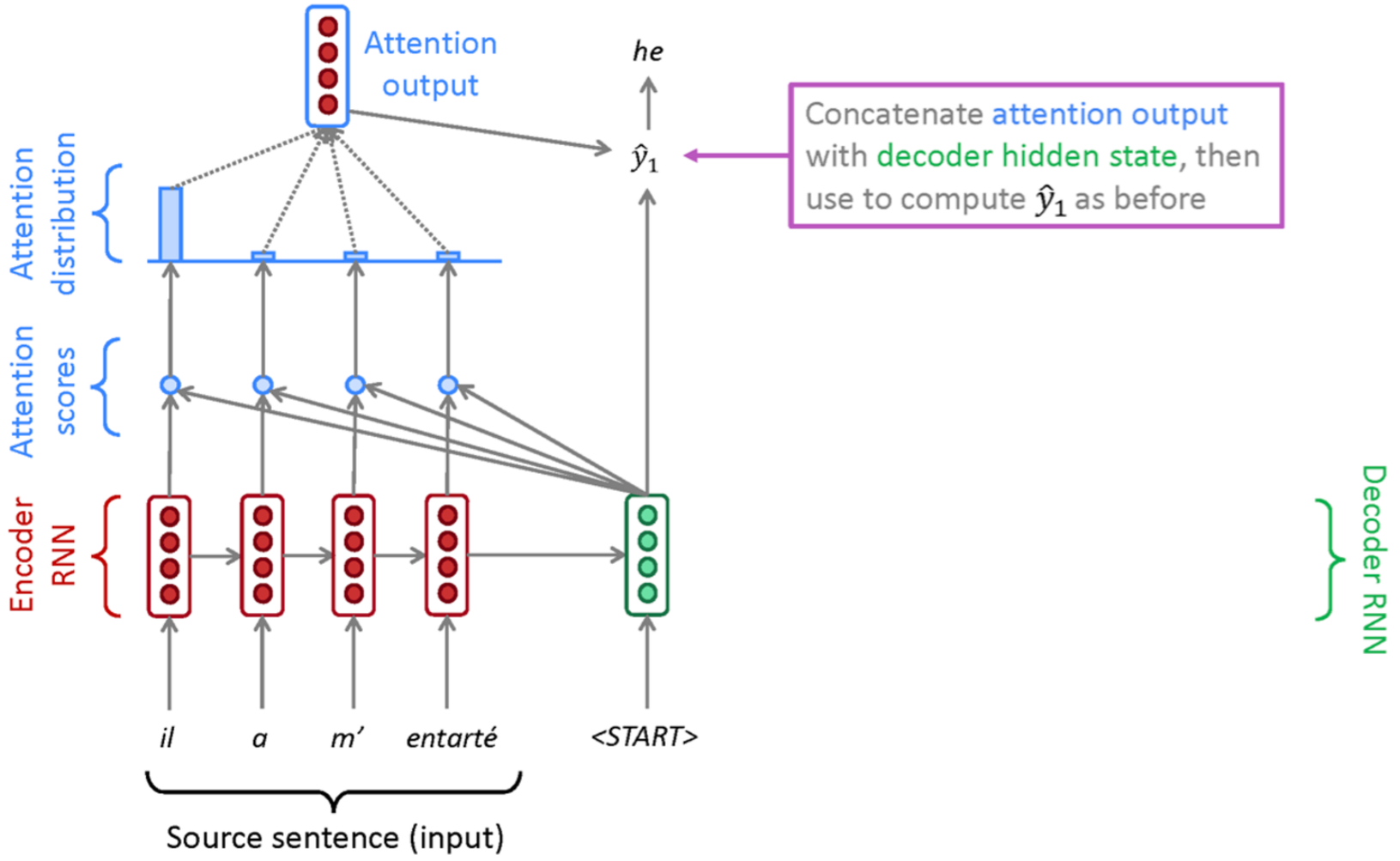
# LSTMs with Attention



# LSTMs with Attention



# LSTMs with Attention



# LSTMs with Attention: In Equations

- We have encoder hidden states: $h_1, \ldots, h_N \in \mathbb{R}^h$

# LSTMs with Attention: In Equations

- We have encoder hidden states: $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep $t$, we have decoder hidden state $s_t$

# LSTMs with Attention: In Equations

- We have encoder hidden states: $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep $t$, we have decoder hidden state $s_t$
- We get the attention scores $e_t$ for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

# LSTMs with Attention: In Equations

- We have encoder hidden states: $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep $t$, we have decoder hidden state $s_t$
- We get the attention scores $e_t$ for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution $\alpha^t$ for this step (which is a prob dist):

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

# LSTMs with Attention: In Equations

- We have encoder hidden states: $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep $t$, we have decoder hidden state $s_t$
- We get the attention scores $e_t$ for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution $\alpha^t$ for this step (which is a prob dist):

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use $\alpha^t$ to take a weighted sum of the encoder states to get the attention output $a_t$:

$$a^t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

# LSTMs with Attention: In Equations

- We have encoder hidden states: $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep $t$, we have decoder hidden state $s_t$
- We get the attention scores $e_t$ for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution $\alpha^t$ for this step (which is a prob dist):

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use $\alpha^t$ to take a weighted sum of the encoder states to get the attention output $a_t$:

$$a^t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

- Finally, we concatenate the attention output $a_t$ with the decoder hidden state $s_t$ to obtain: $[a^t; s_t] \in \mathbb{R}^{2h}$

# The Transformer Architecture

- Key idea: instead of an RNN, just use attention
- High throughput & expressivity: compute queries, keys and values as (different) linear transformations of the input
- Attention weights are queries • keys; outputs are sums of weighted values

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

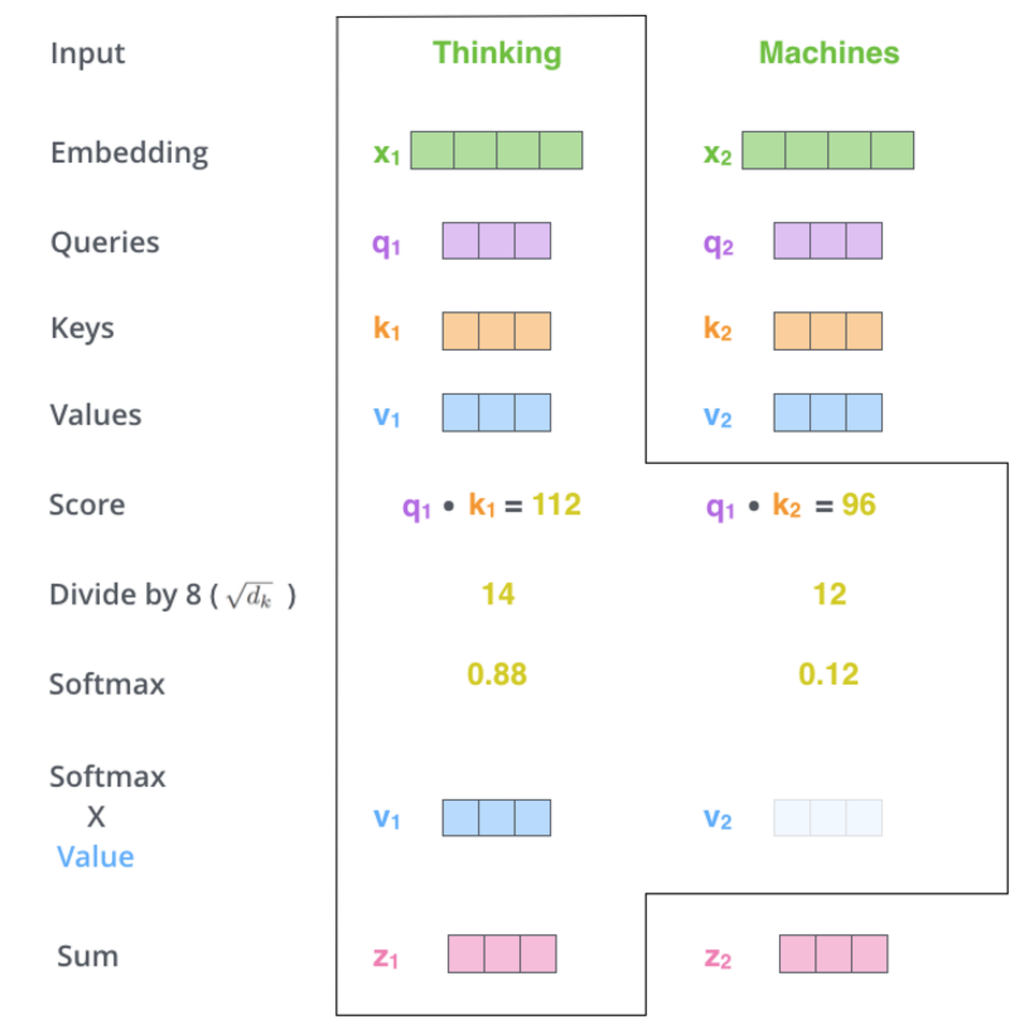
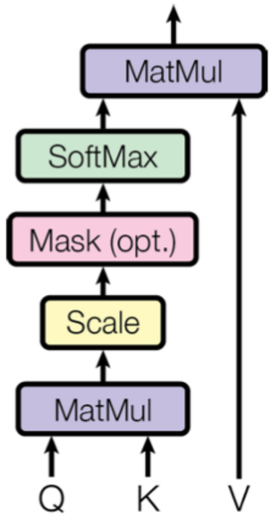



Image from Jay Alammar

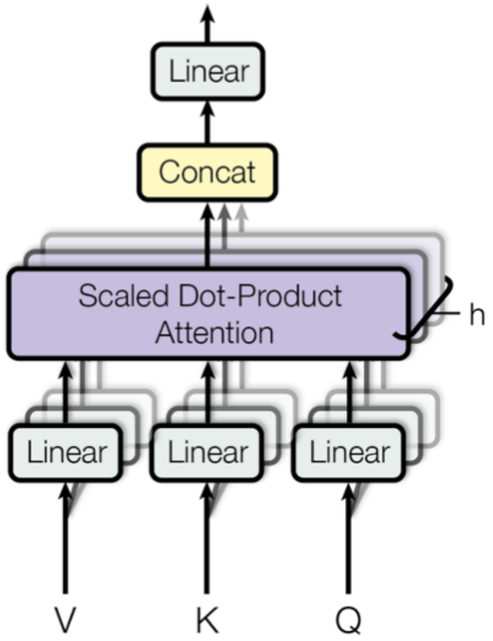
# The Transformer Architecture

**Scaled Dot-Product Attention**

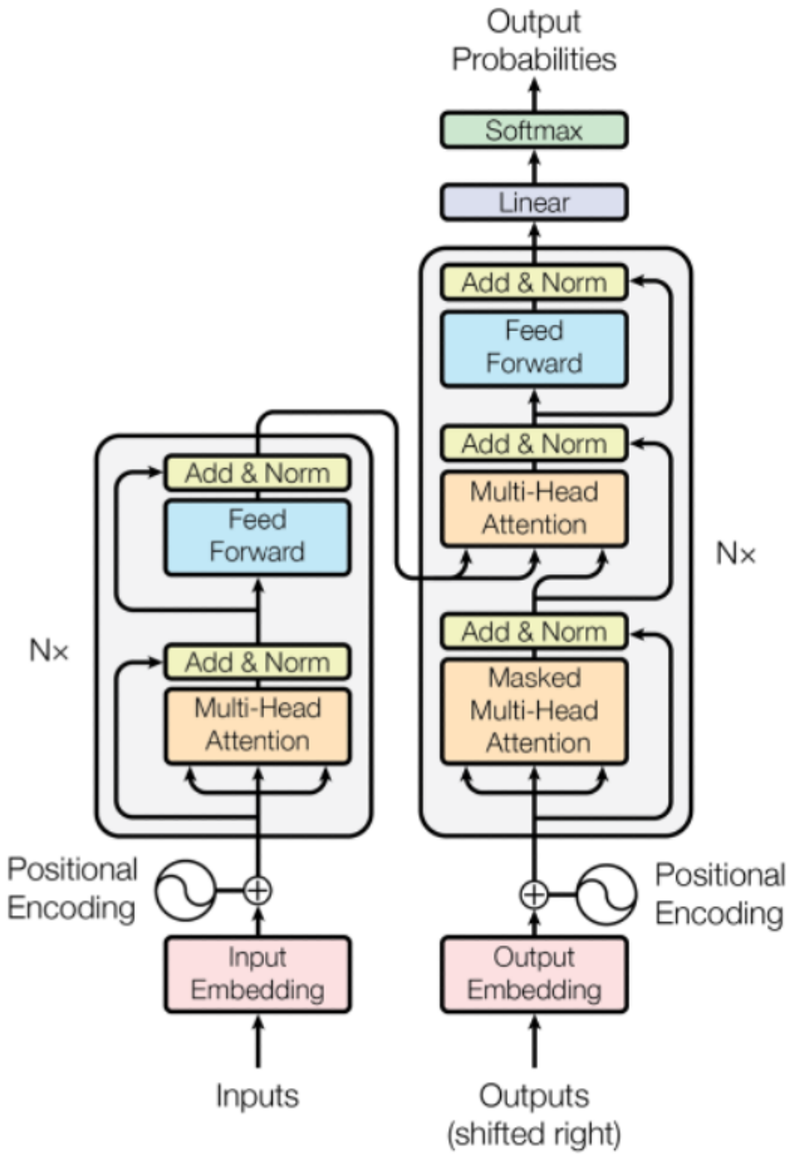



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

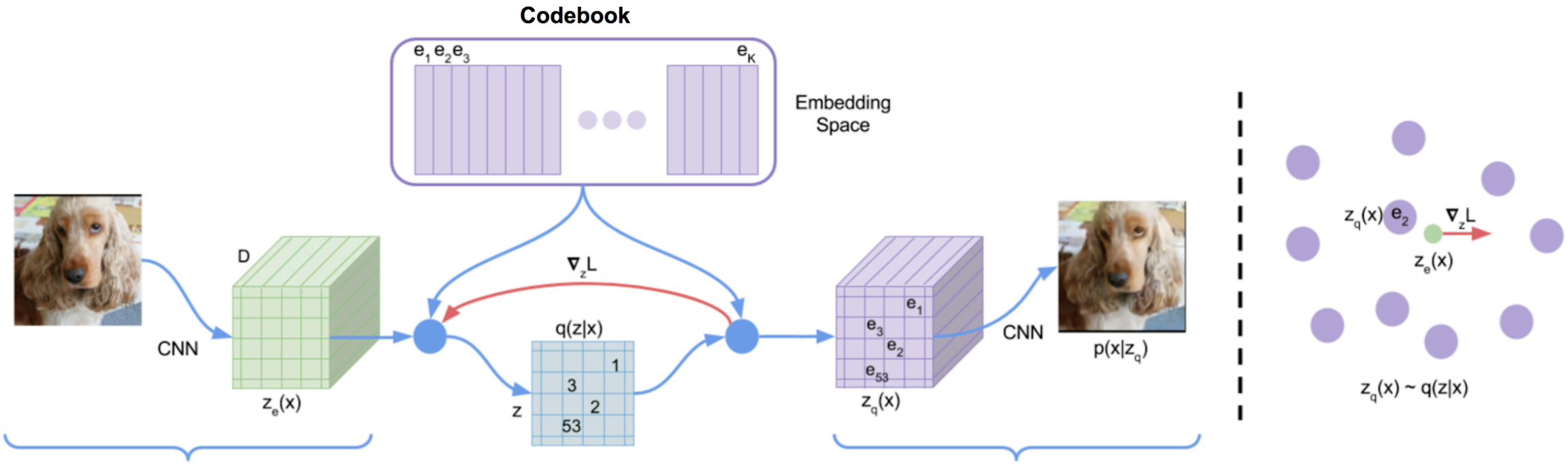
**Multi-Head Attention**




$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



# Next Time: VQ-VAEs




Posterior categorical distribution:

$$q(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg\min_i \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_i\|_2 \\ 0 & \text{otherwise.} \end{cases}$$