

# Week 01: Module Introduction

## CM50265 Machine Learning 2

# Teaching Team

- Dr. Vinay Namboodiri ([vpn22@bath.ac.uk](mailto:vpn22@bath.ac.uk))
- Dr. Hongping Cai ([hc551@bath.ac.uk](mailto:hc551@bath.ac.uk))
- Tutors (Joseph Goodier, Ferdie Krammer, Adam Hayes)
  - Lab sessions
  - Help with weekly exercises and coursework

*Main mode of contact: Moodle Discussion Forum*

# Timetable

- Lectures
  - 03.15pm Tuesday at 5W2.01
  - 03.15pm Wednesday at 5W2.01
- Lab sessions (from week 2)
  - Group A: 15.15 Monday at [10W 0.02 PC lab](#)
    - Tutor: Joseph Goodier, Adam Hayes
  - Group B: 13.15 Wednesday at [10W 0.02 PC lab](#)
    - Tutor: Adam Hayes, Ferdie Krammer

*Lecture slides and exercises (if any) will be uploaded on Moodle  
24 hours in advance*



## Approximate schedule

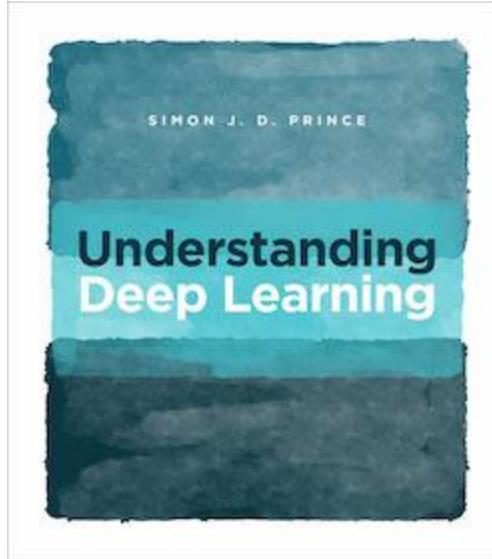
- Week 01: Boosting (Vinay)
- Week 02: Intro to deep learning, MLPs (Vinay)
- Week 03: CNNs (Hongping)
- Week 04: Training in practice (Hongping)
- Week 05: RNN, Attention-based Seq2Seq (Hongping)
- Week 06: Consolidation week, no lectures
- Week 07: GANs (Hongping)
- Week 08: Visualizing CNN (Hongping)
- Easter break
- Week 09: Transformer (Vinay)
- Week 10: NLP (Vinay)
- Week 11: Adaptive learning (Vinay)

# Assessment

- 60% exam
- 20% coursework 1 (Group of 2 people)
  - Coding & report for CNN
  - Date of Announcement: 19 February (Mon.), 2024
  - Due date for signing up a group in Moodle: **8pm, 23 February (Fri.), 2024**
  - Due date: **8pm, 22 March (Fri.), 2024**
- 20% coursework 2 (Individual)
  - Paper presentation & Peer reviewing
  - Date of Announcement: 25 March (Mon.), 2024
  - Due Date for video submission: **8pm, 24 April (Wed.) 2024 (no extension allowed)**
  - Due Date for peer assessment: **8pm, 1 May (Fri.), 2023 (1 week extension maximum)**

*Note: I WILL NOT reply emails about coursework questions. Instead, all questions about the coursework will be answered in the **Moodle Discussion Forum**, then everyone can benefit from the answers.*

# Main Reference Text



<https://udlbook.github.io/udlbook/>

Excellent book by Simon J.D. Prince  
Reference available in library and digital  
copy is freely available online

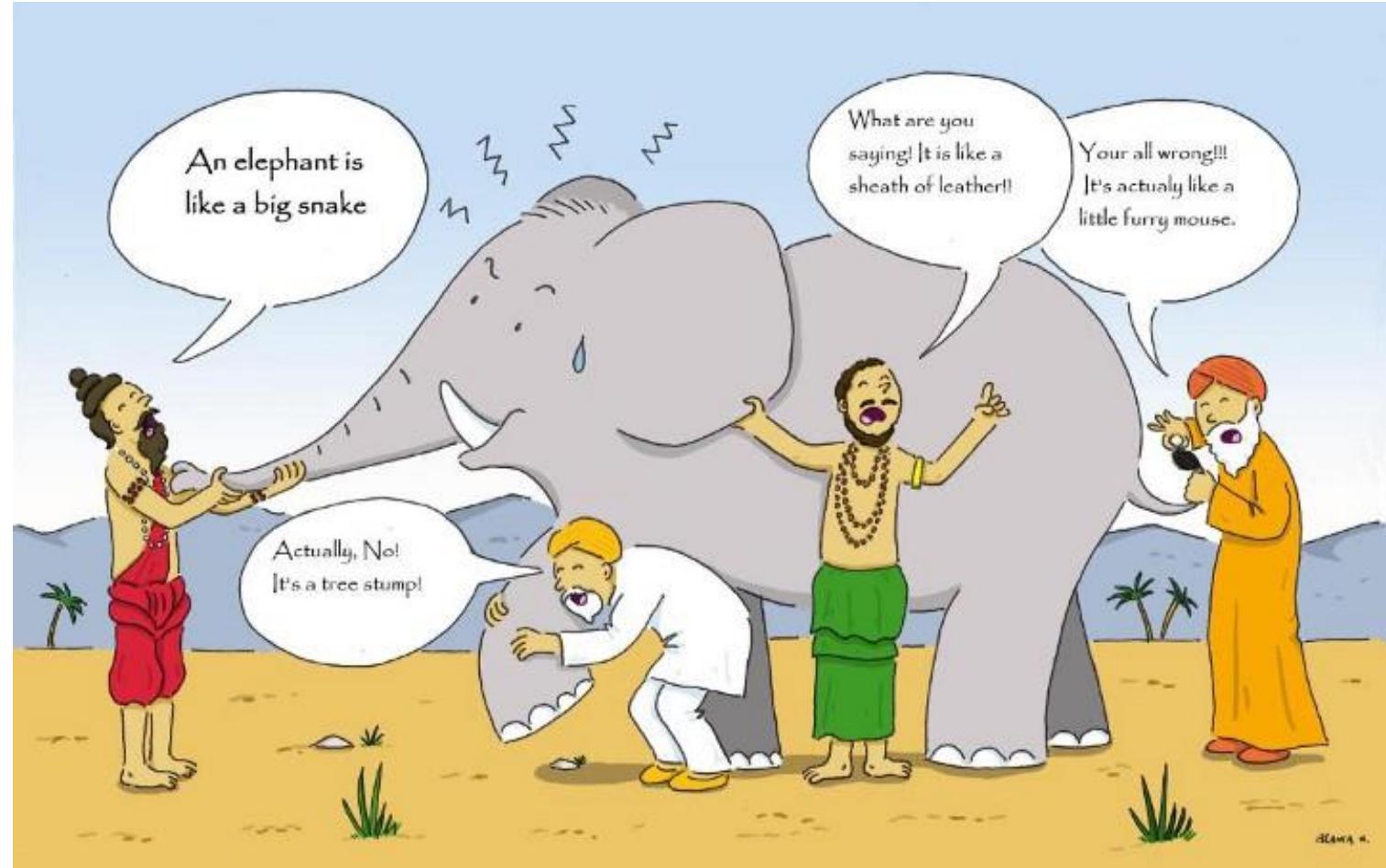
Any questions?

# Week 01: Boosting

CM50265 Machine Learning 2

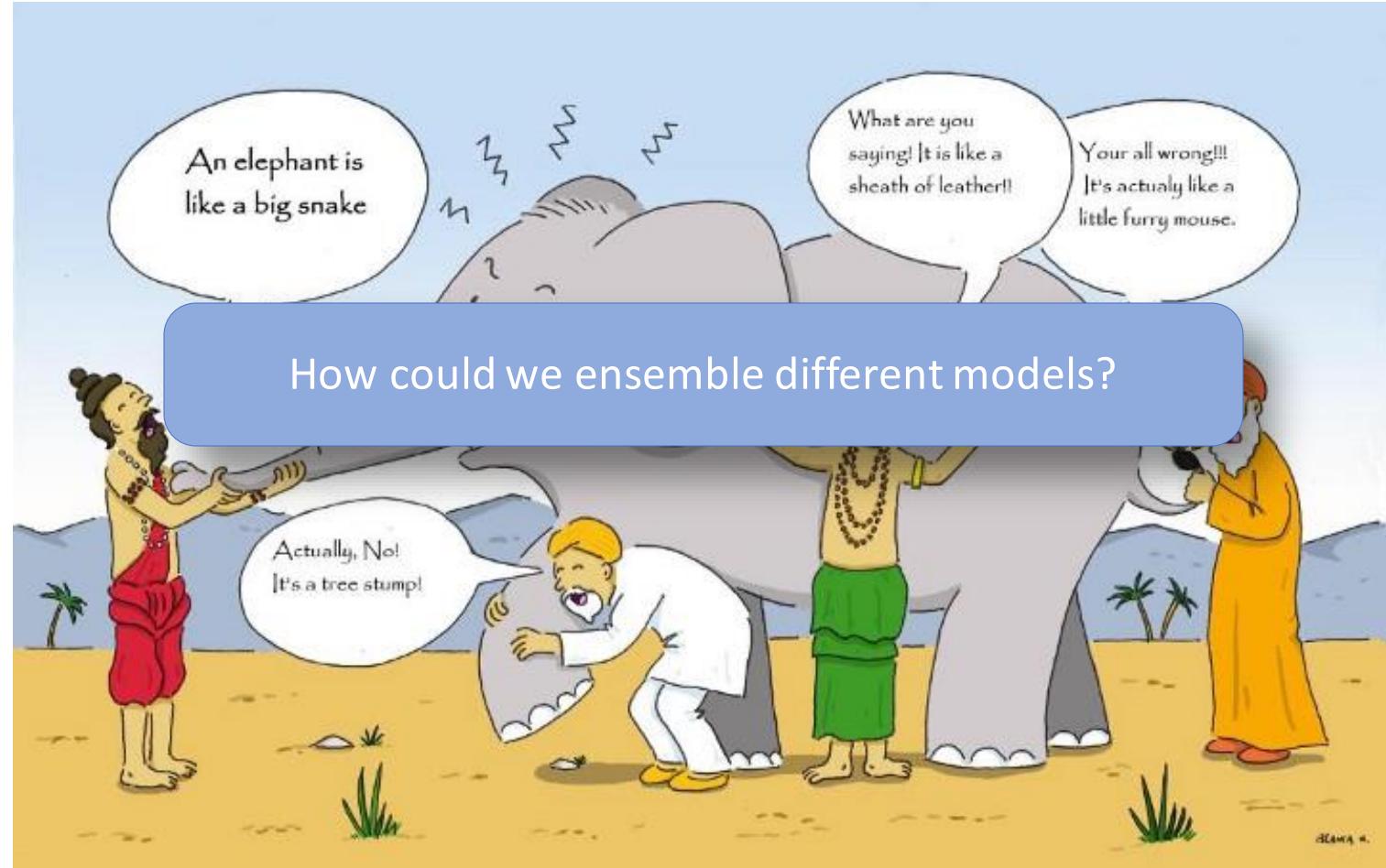
# Topic 1: Ensemble Learning

# Ensemble Learning



- Intuition: wisdom of the crowd

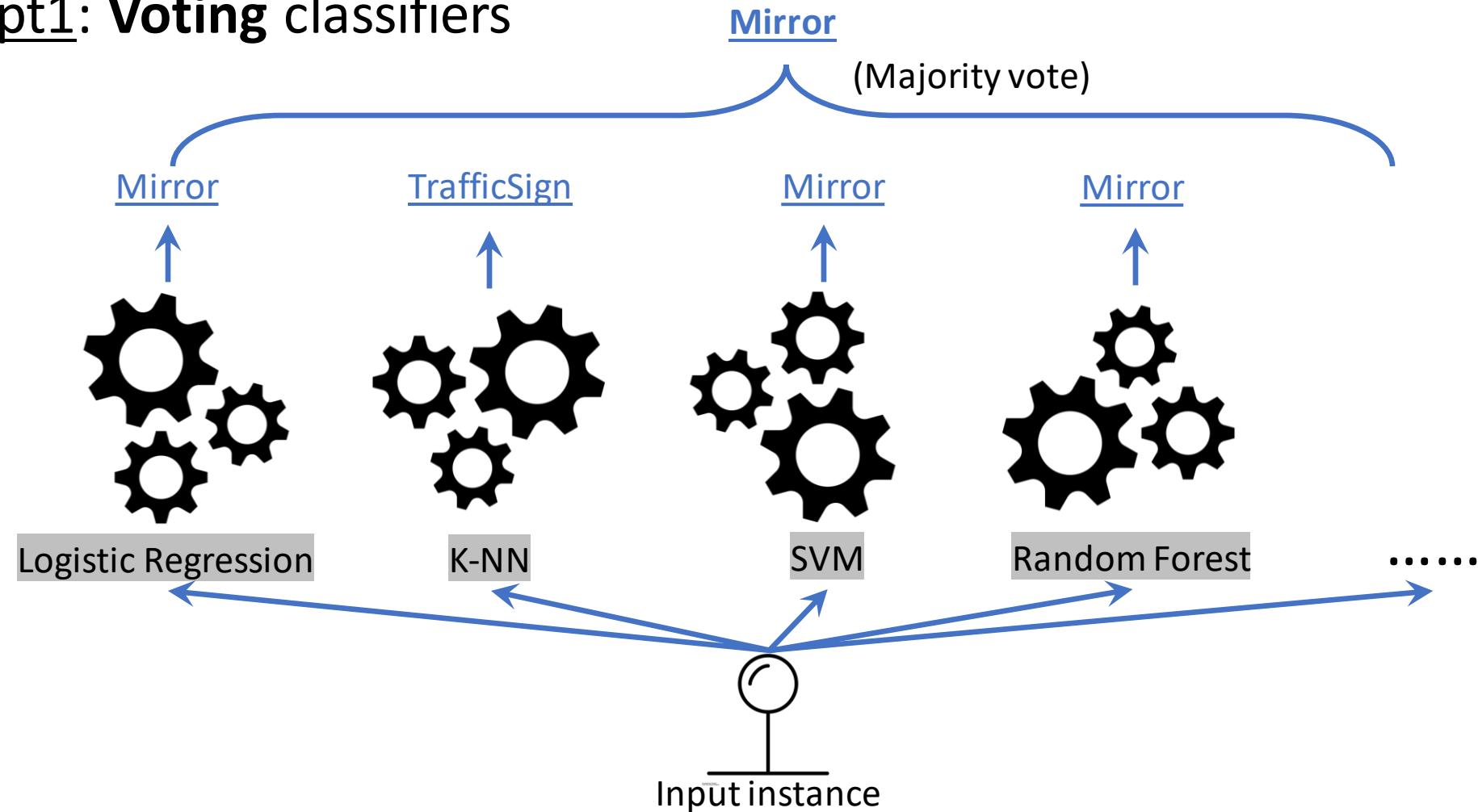
# Ensemble Learning



- Intuition: wisdom of the crowd

# Ensemble learning

- Opt1: Voting classifiers



# Ensemble learning

- Opt1: **Voting** classifiers

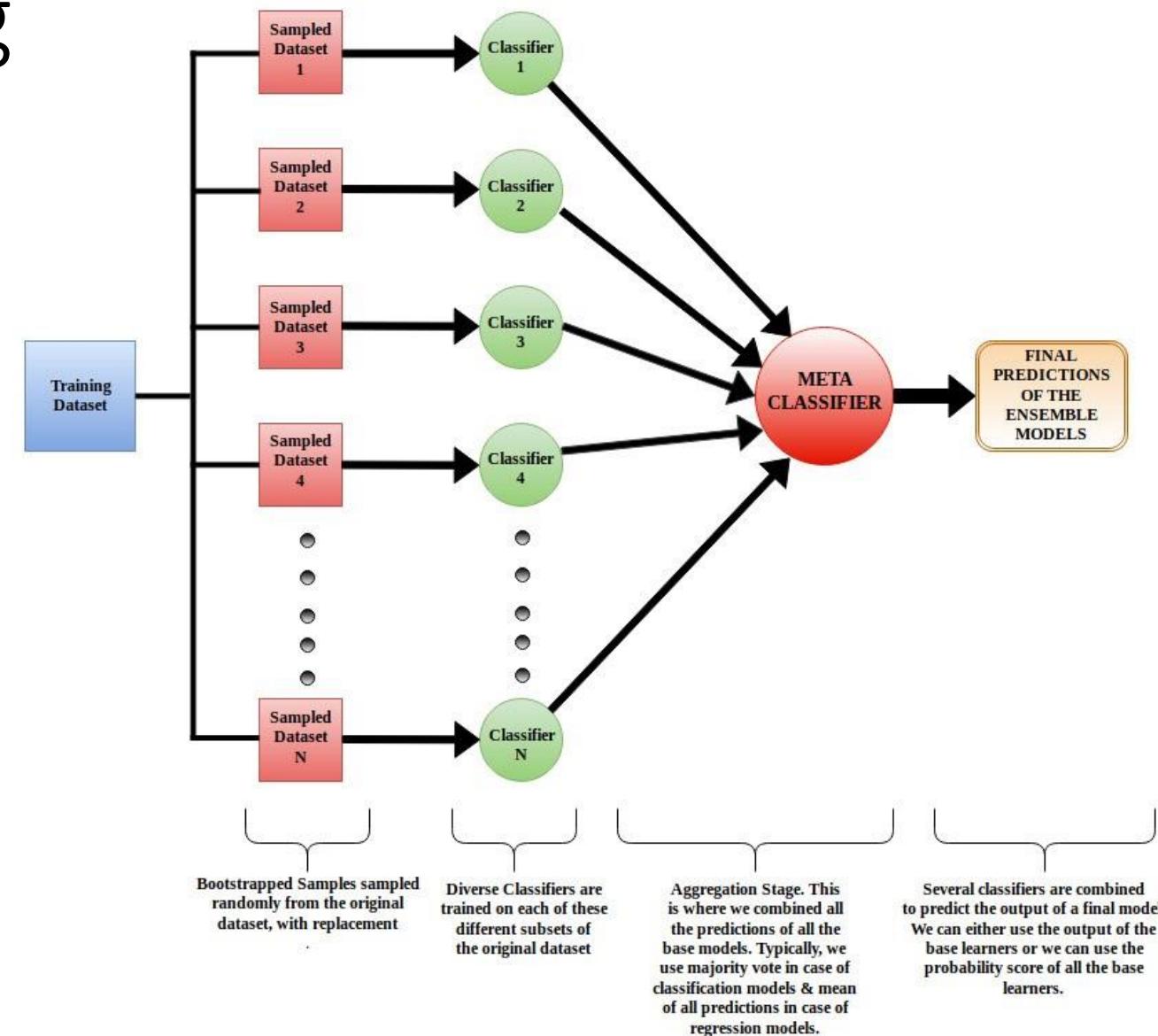
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier,
VotingClassifier

clf1 = LogisticRegression(multi_class='multinomial',
random_state=1)
clf2 = RandomForestClassifier(n_estimators=50,
random_state=1)
clf3 = GaussianNB()

eclf1 = VotingClassifier(estimators=[ ('lr', clf1),
('rf', clf2), ('gnb', clf3)], voting='hard')
```

# Ensemble learning

- Opt2: Bagging (short for Bootstrap aggregating)
- **Bagging** is a method of training multiple models using a random subset of the training data with replacement.



# Bagging

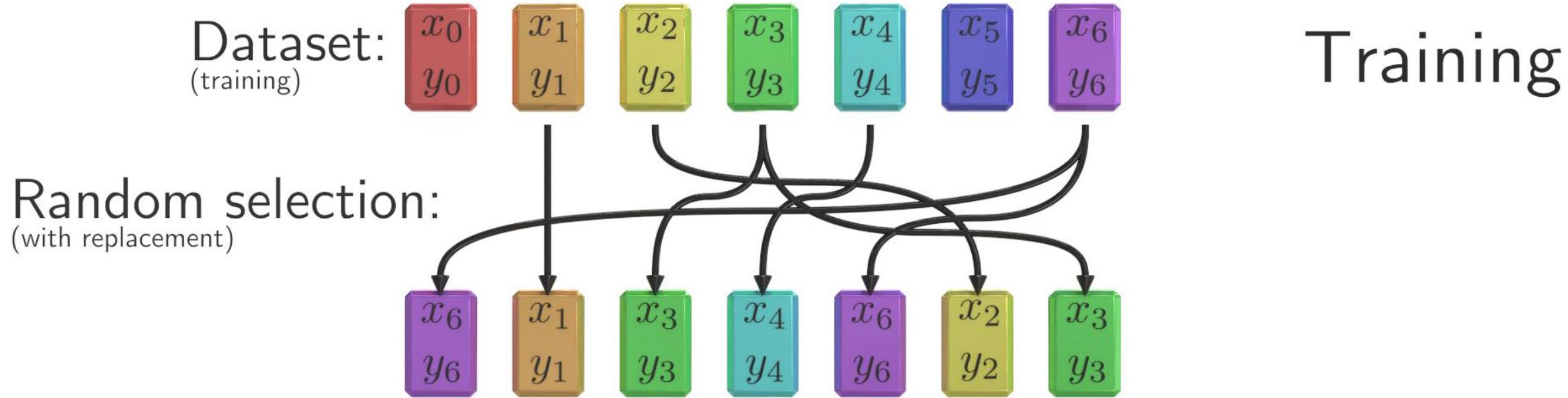
Dataset:  
(training)

$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$
$y_0$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$

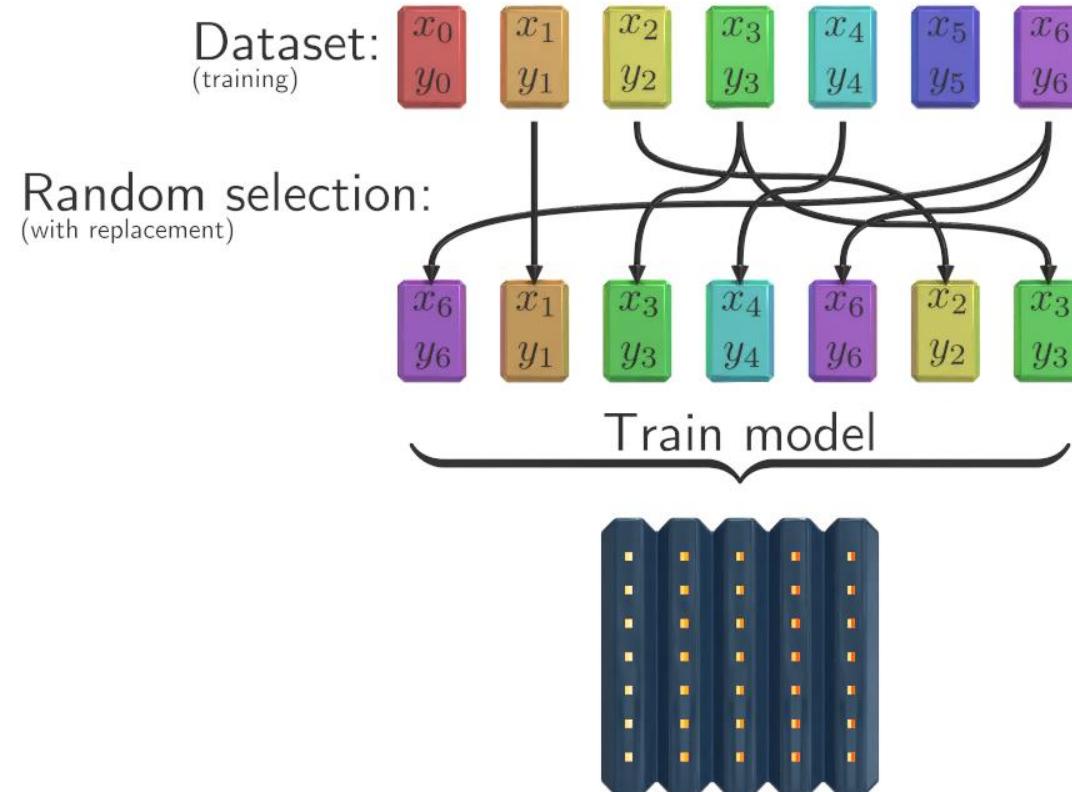
Training



# Bagging



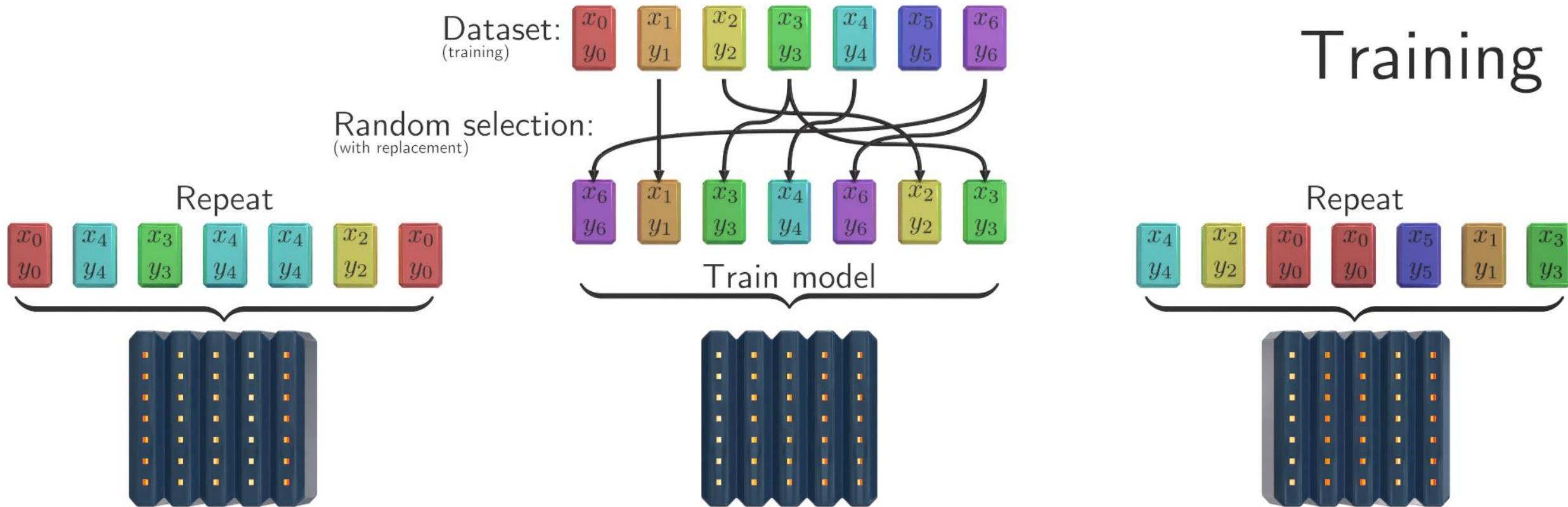
# Bagging



## Training

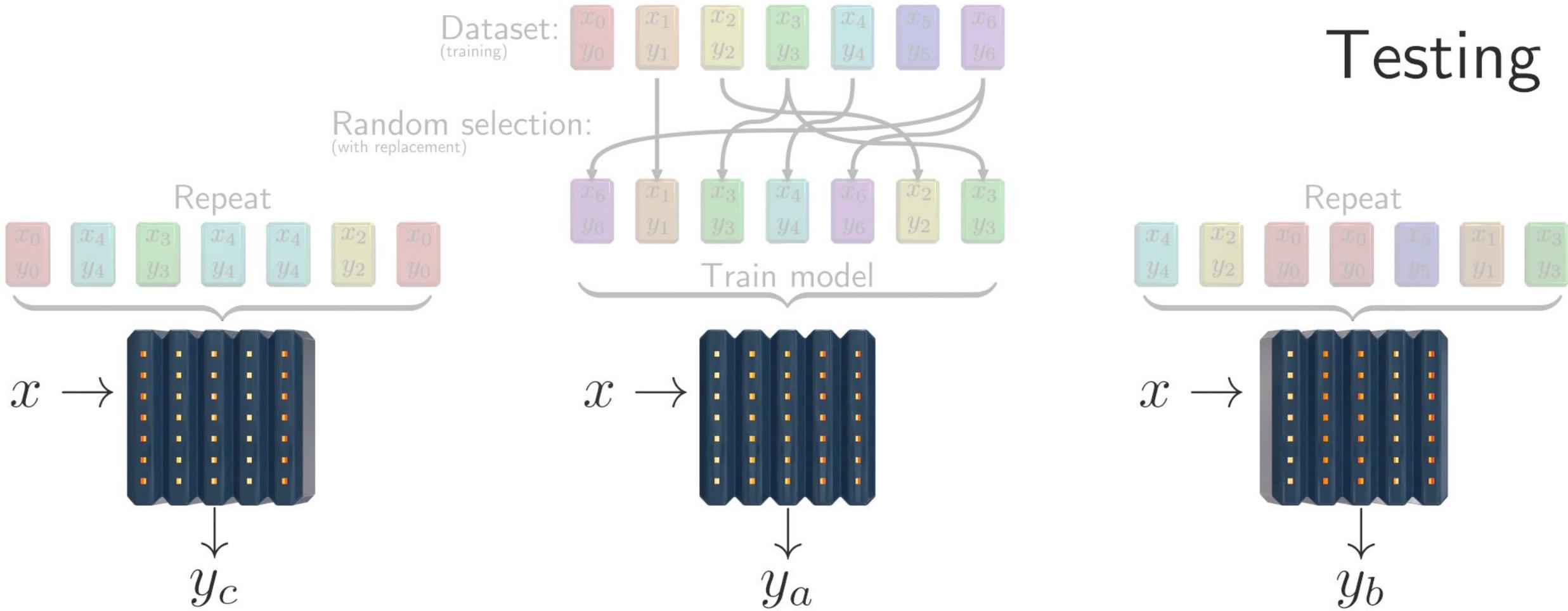


# Bagging



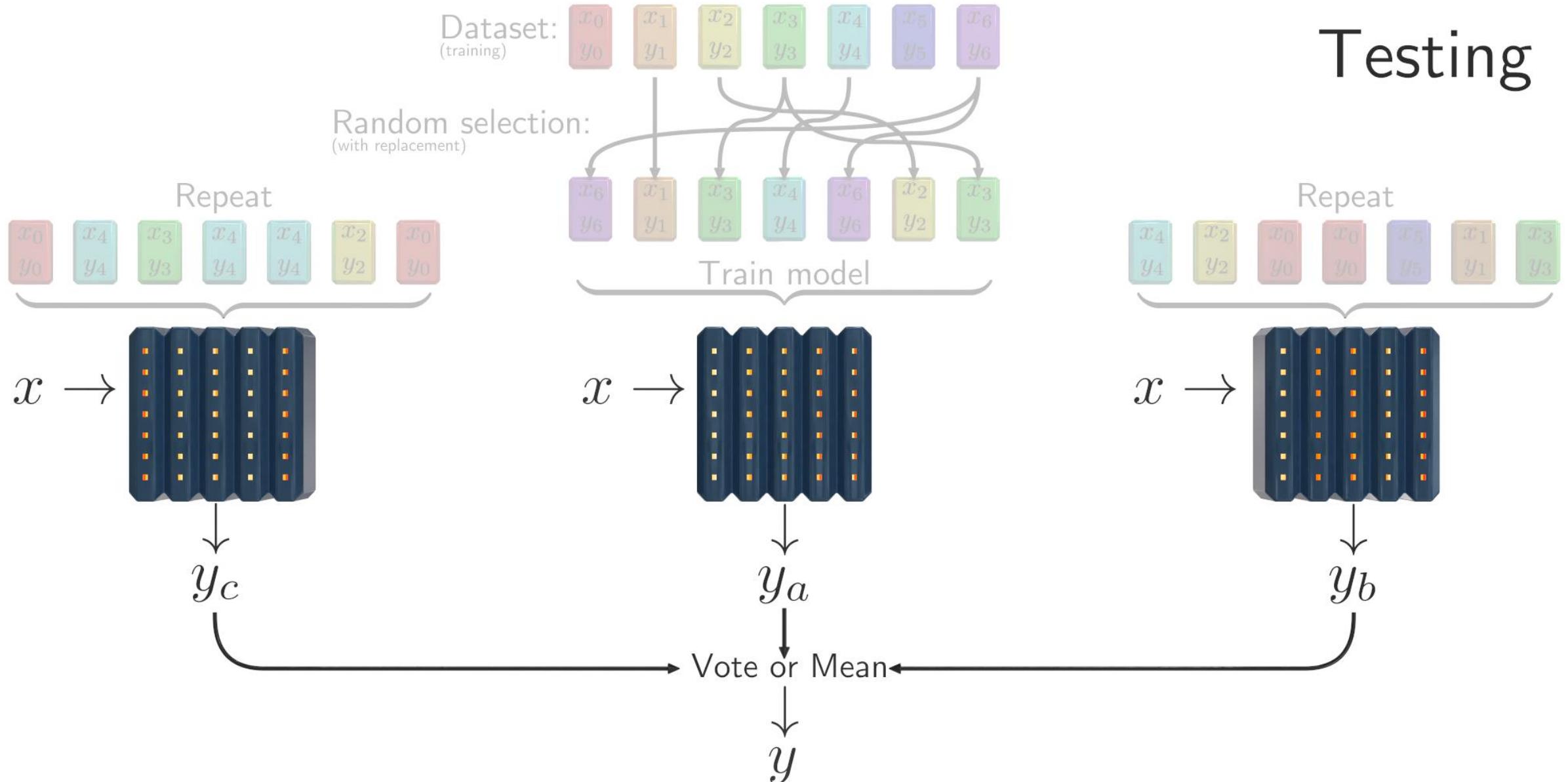


# Bagging



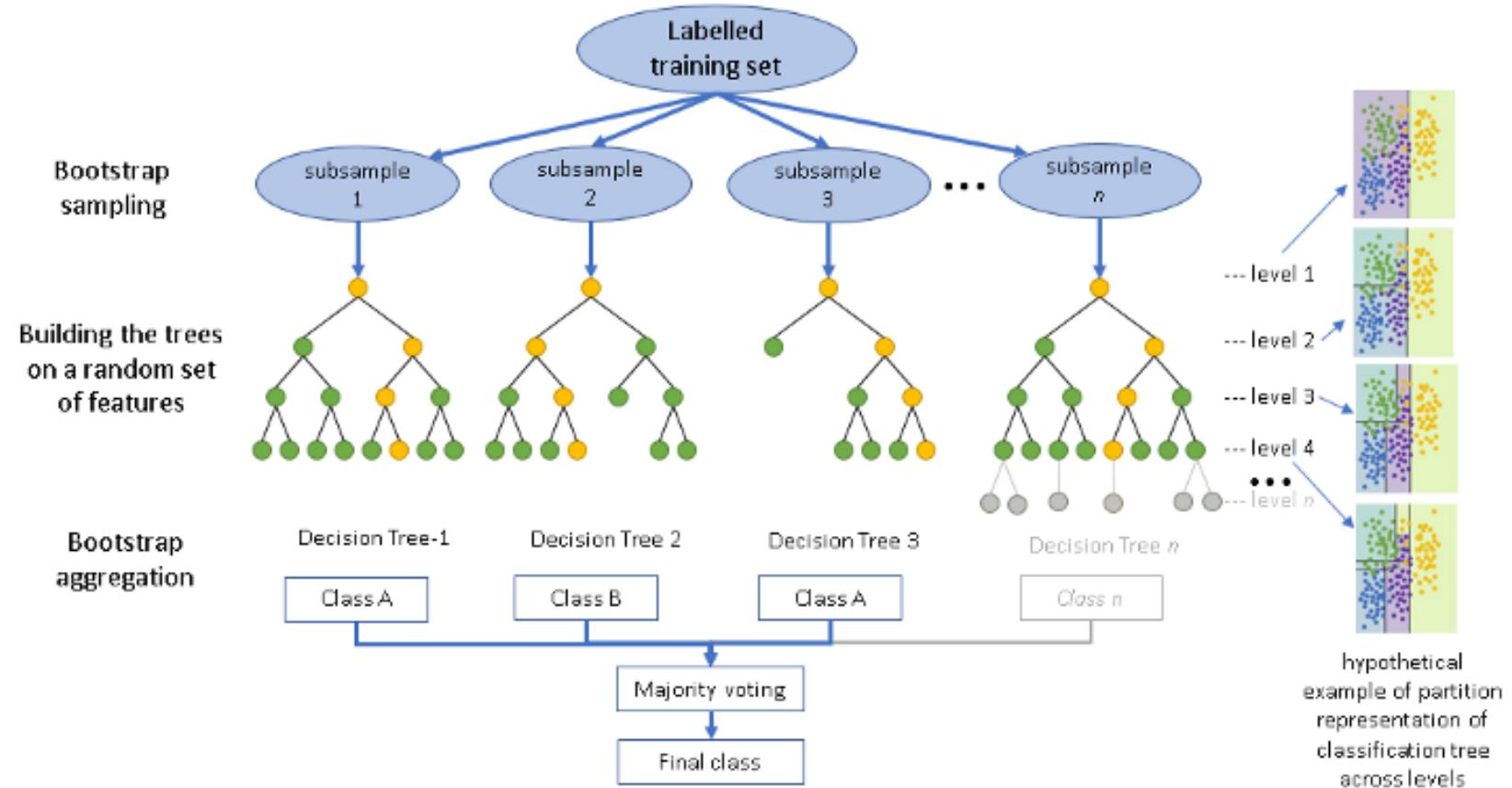


# Bagging



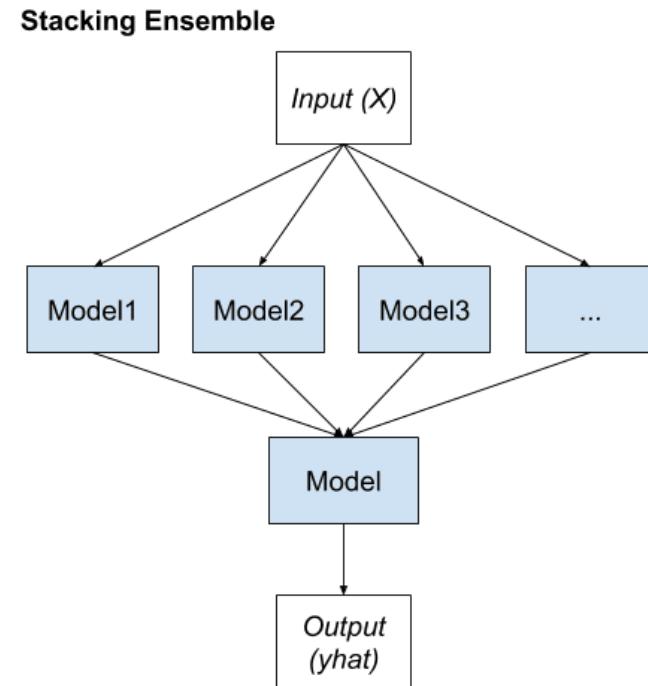
# Bagging

- **Random Forest** = Bagging + Decision Tree + Random set of features



# Ensemble learning

- **Opt 3: Stacking**
  - Unchanged training dataset.
  - Different machine learning algorithms for each ensemble member.
  - A **meta-learner** to learn how to best combine predictions.



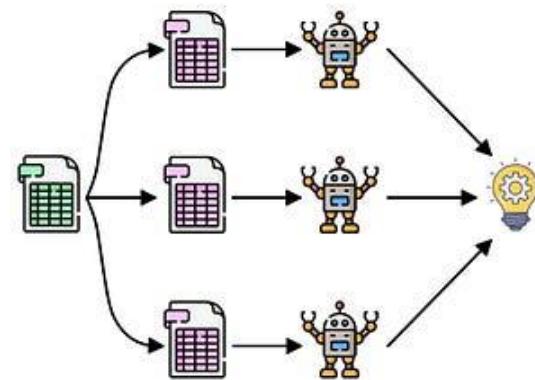
# Ensemble learning

- Opt 4: Boosting
  - “Learning from mistakes”

# Topic 2: Boosting

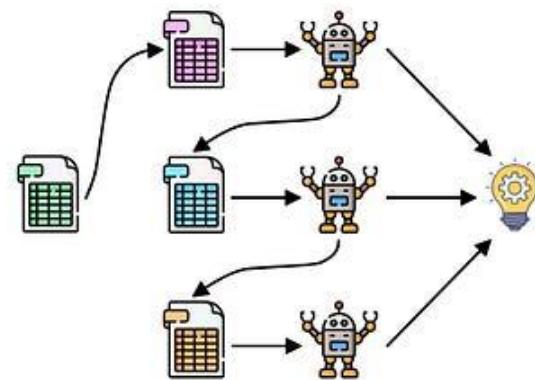
## Boosting = Learning from mistakes

### Bagging



Parallel

### Boosting



Sequential

**Principle:** Build a strong model from sequential models, where each model improves or corrects the mistake made in the previous ones.

# Most popular boosting algorithms

- AdaBoost (Adaptive Boosting)
  - Freund and Schapire, 1995
- Gradient Boosting
  - Jerome H. Friedman, 1999
- XGBoost (Extreme Gradient Boosting)

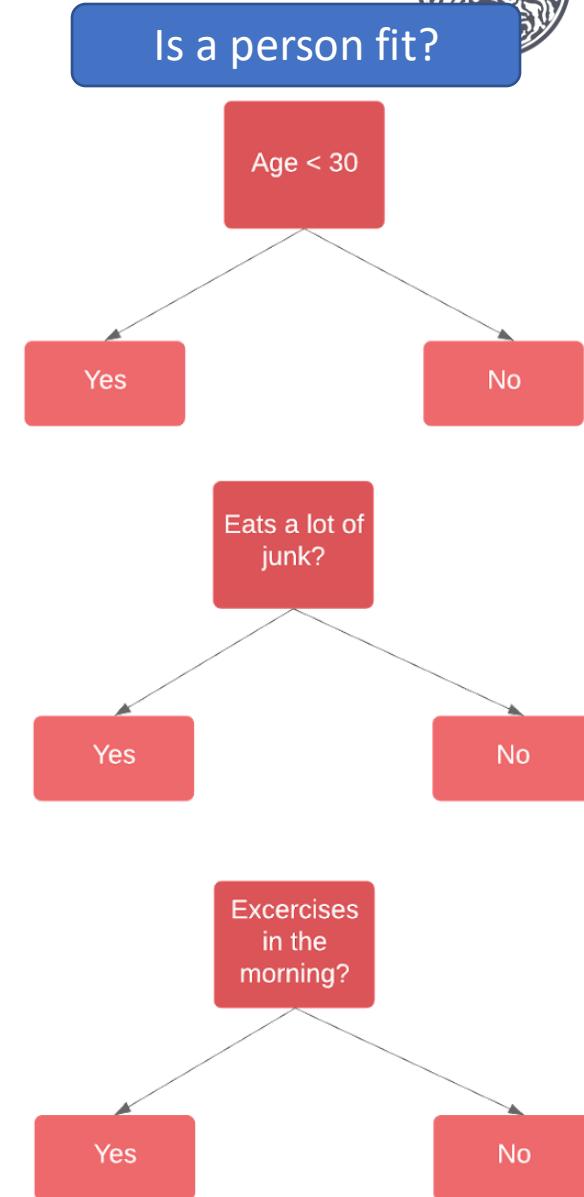
# Topic 3: AdaBoost



# Main ideas of AdaBoost

## 1. Combining “weak models”

- A **weak model/hypothesis** is one that performs better than random guessing, but still performs poorly.
- Usually is a **decision stump** (1-level decision tree)



Three decision stumps

# Main ideas of AdaBoost

## 2. Each weak model has a different significance value

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$
$$h_t : \mathcal{X} \rightarrow \{-1, +1\}$$

Significance for the t-th weak model

**Q:** How to define the significance value?

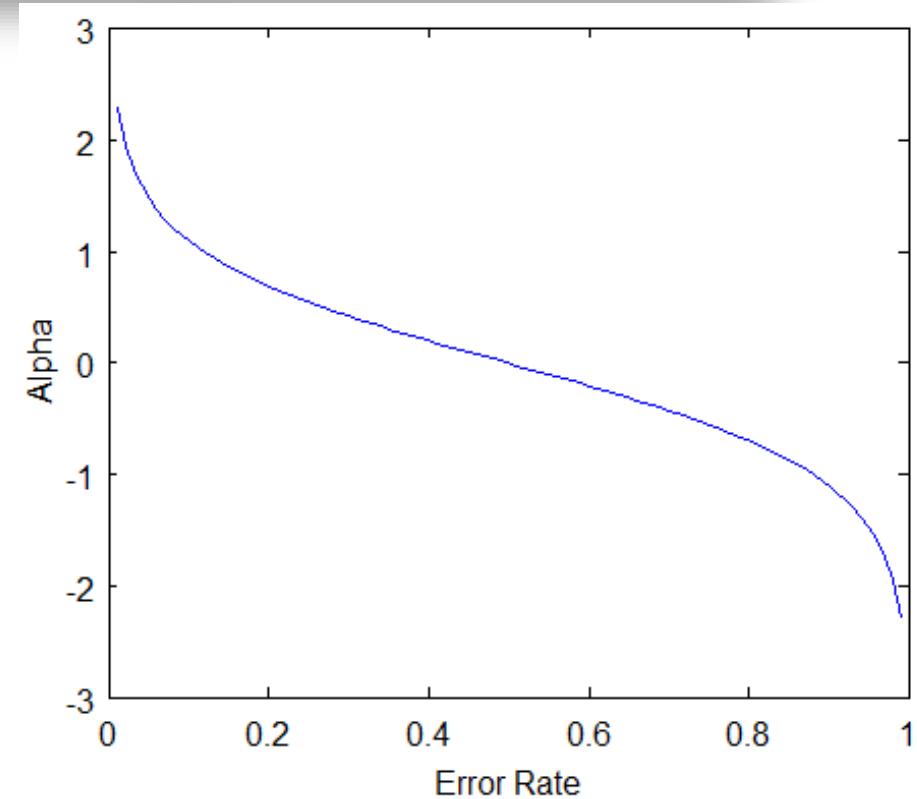
# Main ideas of AdaBoost

## 2. Each weak model has a different significance value

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Total error for the t-th weak model



**Total error** = sum of weights for incorrectly classified samples

# Main ideas of AdaBoost

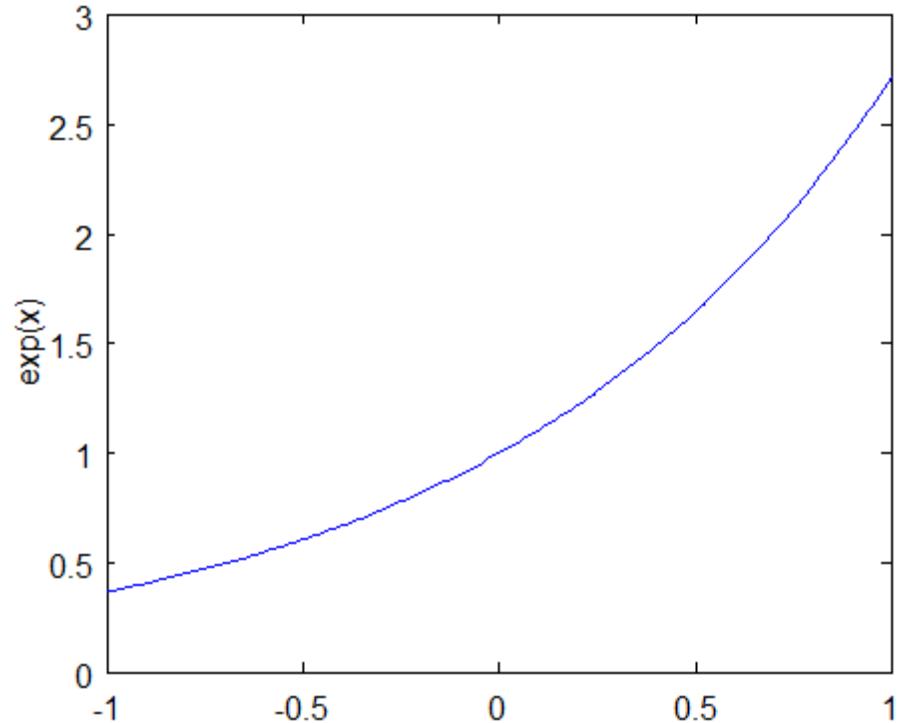
## 3. Higher sample weights for mis-classified samples

If the sample is mis-classified by the current model:

$$\text{New sample weight} = \text{sample weight} * e^{\alpha_t}$$

If the sample is correctly classified by the current model:

$$\text{New sample weight} = \text{sample weight} * e^{-\alpha_t}$$



# Main ideas of AdaBoost

## 3. Higher sample weights for mis-classified samples

If the sample is mis-classified by the current model:

$$\text{New sample weight} = \text{sample weight} * e^{\alpha_t}$$

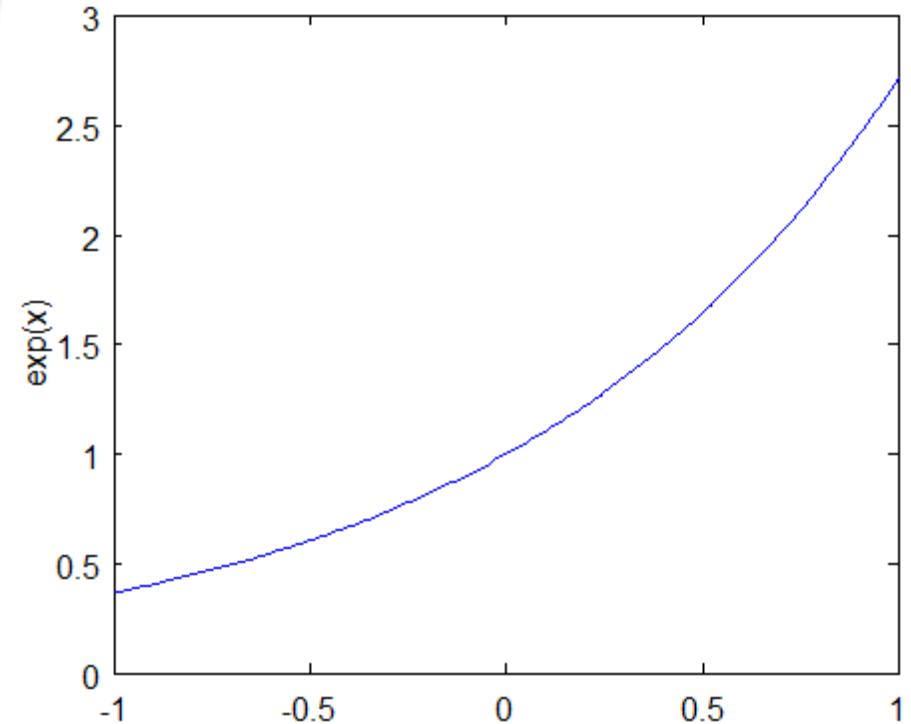
If the sample is correctly classified by the current model:

$$\text{New sample weight} = \text{sample weight} * e^{-\alpha_t}$$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

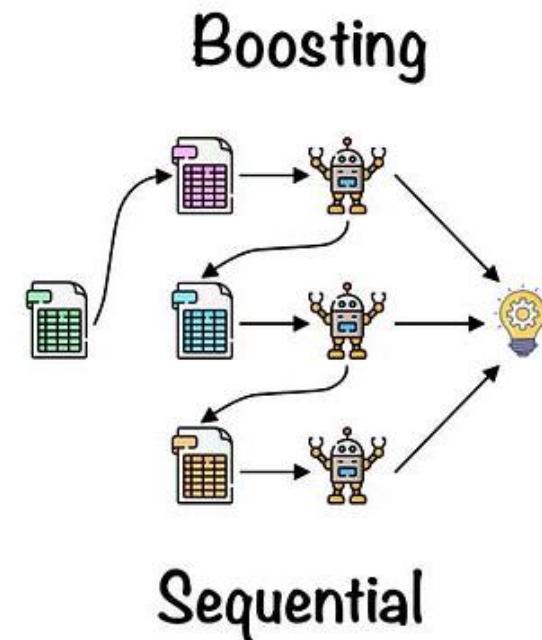
$h_t : \mathcal{X} \rightarrow \{-1, +1\}$   
 $y_i \in \{-1, +1\}$

New sample weight → D<sub>t+1</sub>(i)  
D<sub>t</sub>(i) → Z<sub>t</sub> ← The sum of all the weights



# Main ideas of AdaBoost

**4. Each weak model tries to correct the previous weak models' mistakes**



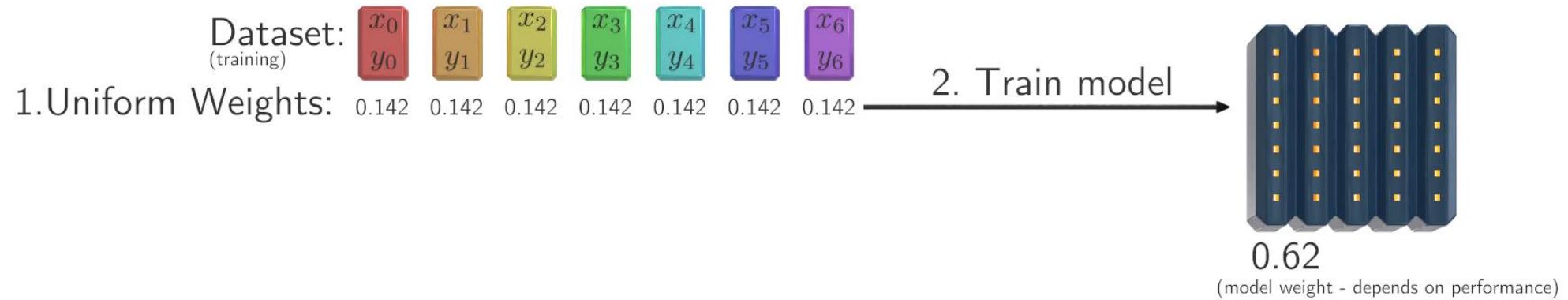
Dataset:  
(training)  $x_0$   $x_1$   $x_2$   $x_3$   $x_4$   $x_5$   $x_6$   
 $y_0$   $y_1$   $y_2$   $y_3$   $y_4$   $y_5$   $y_6$

# Training

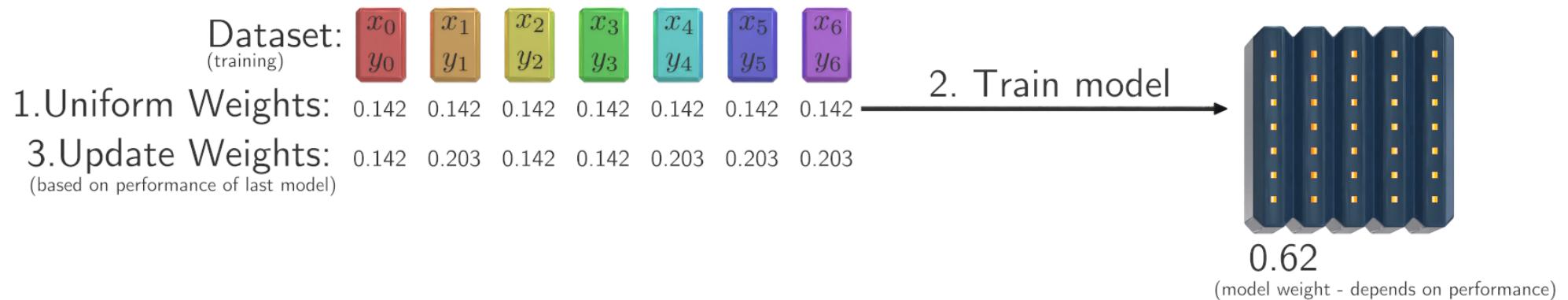
Dataset:  
(training)  $x_0$   $y_0$   $x_1$   $y_1$   $x_2$   $y_2$   $x_3$   $y_3$   $x_4$   $y_4$   $x_5$   $y_5$   $x_6$   $y_6$

1. Uniform Weights: 0.142 0.142 0.142 0.142 0.142 0.142 0.142 0.142

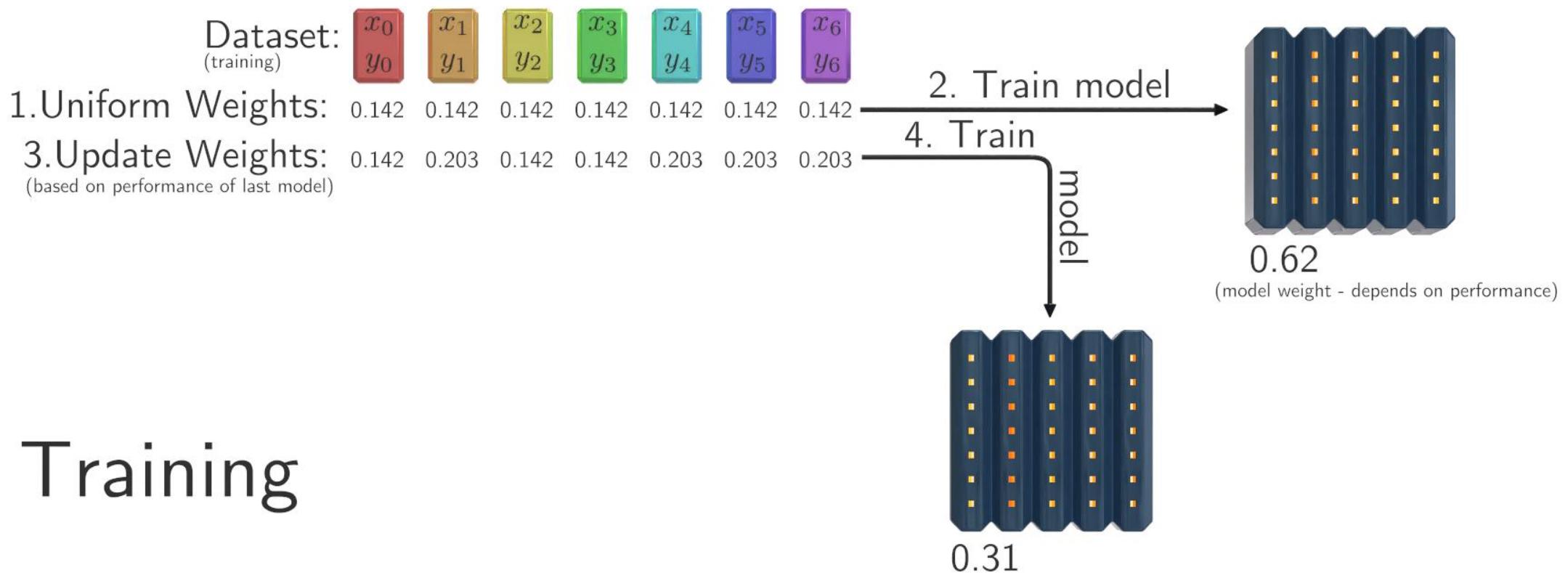
# Training



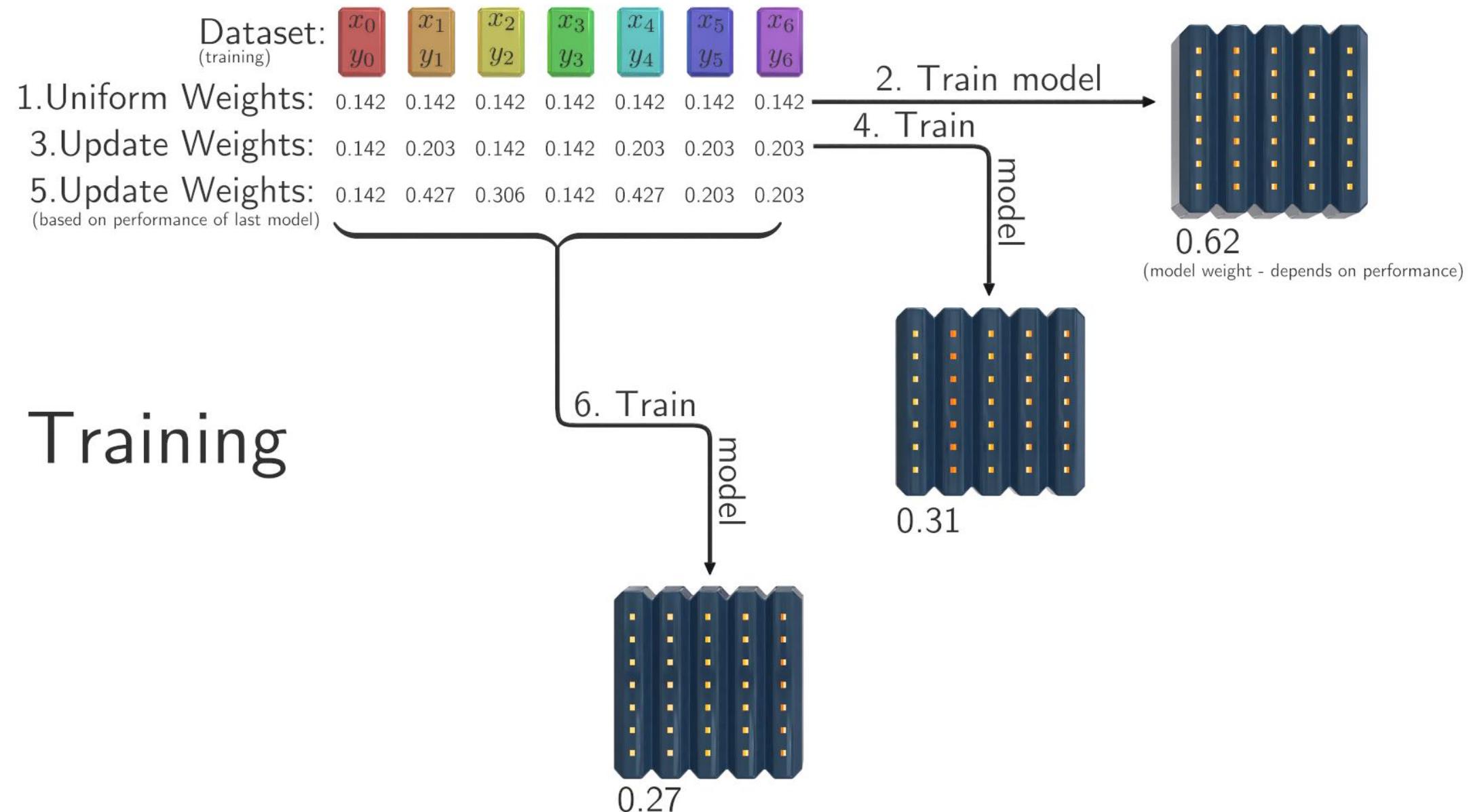
# Training

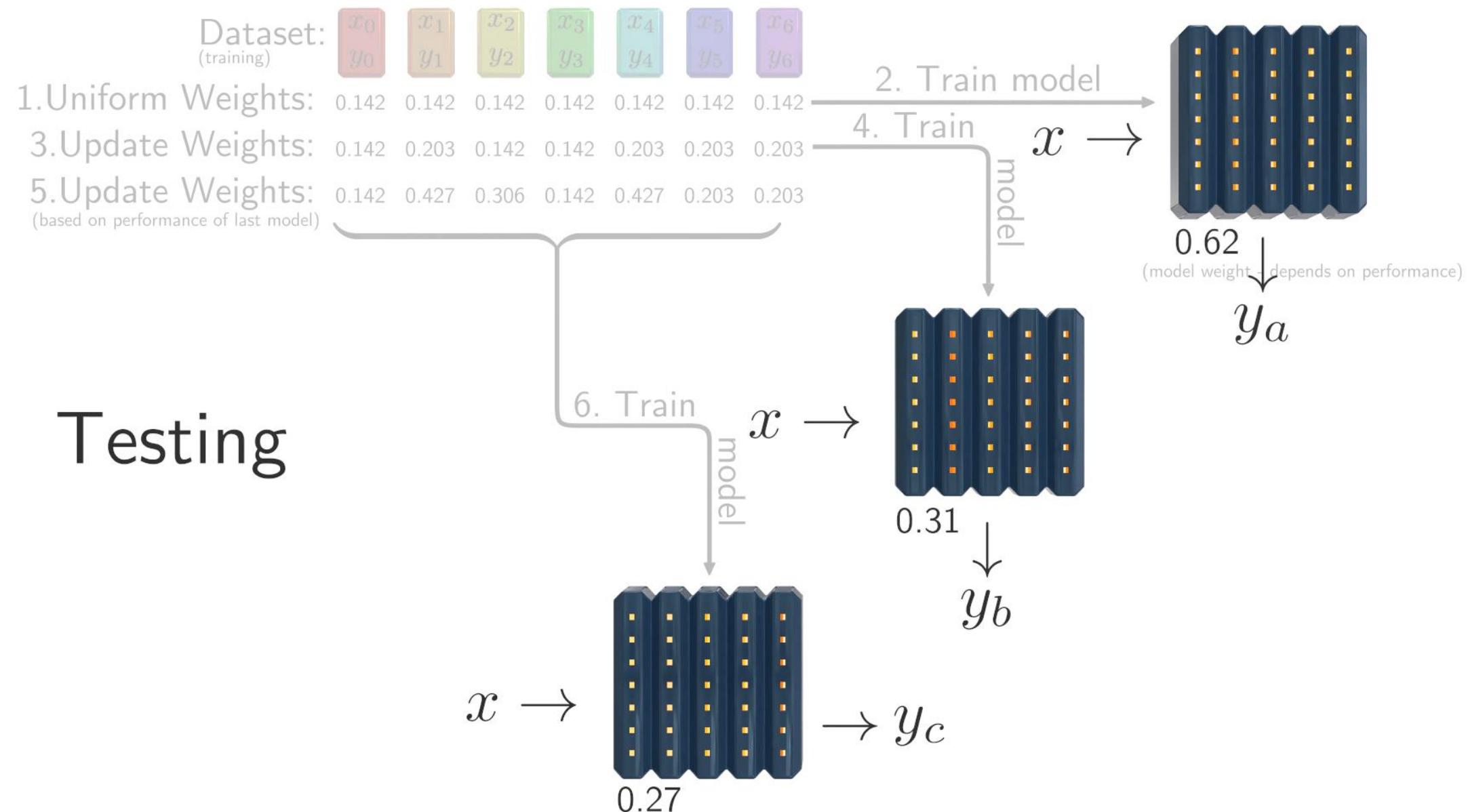


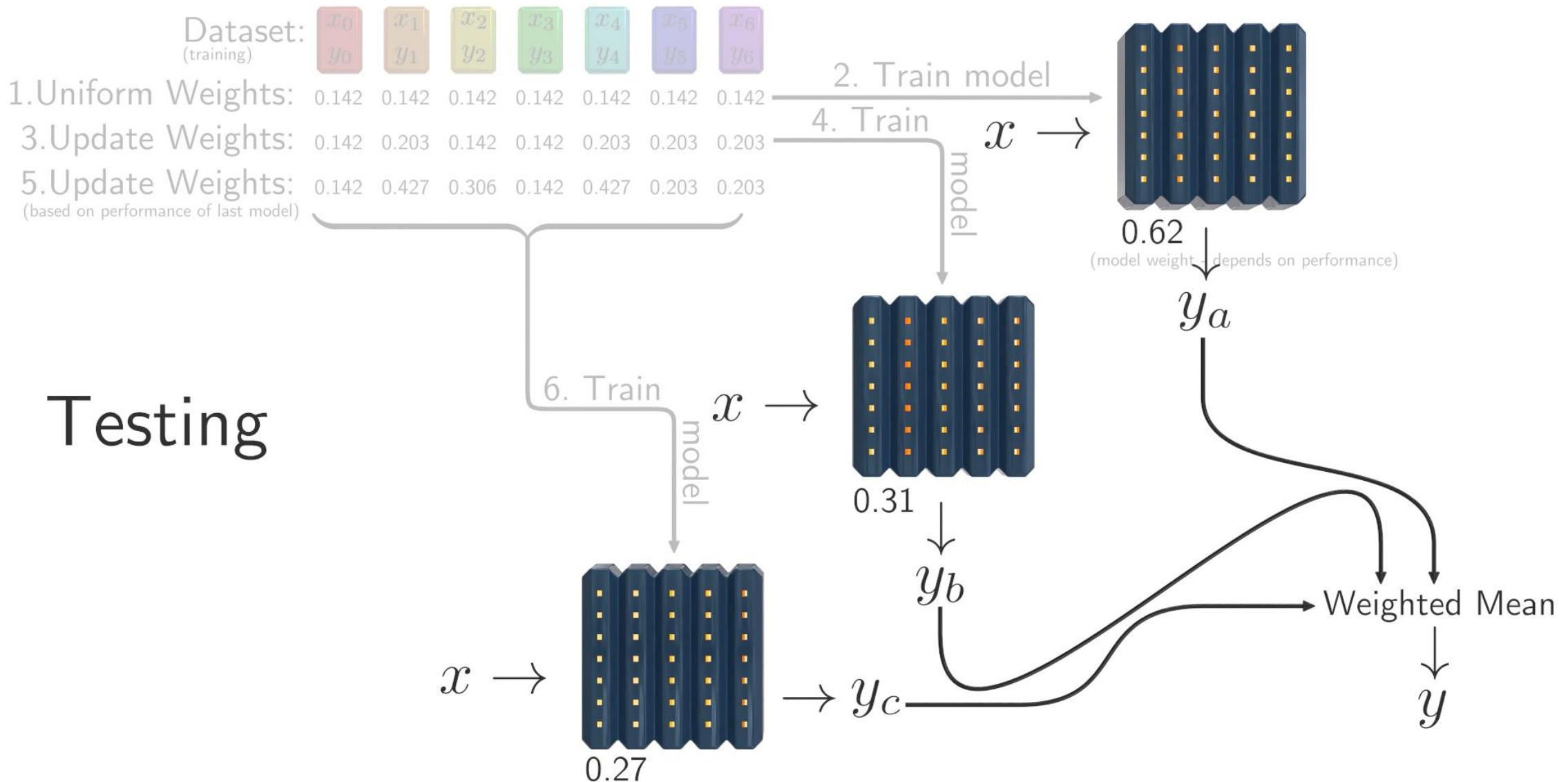
# Training



# Training







# Topic 4: AdaBoost algorithm

# AdaBoost algorithm

---

**Given:**  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}$ ,  $y_i \in \{-1, +1\}$ .

**Initialize:**  $D_1(i) = 1/m$  for  $i = 1, \dots, m$ .

For  $t = 1, \dots, T$ :

- Train weak learner using distribution  $D_t$ .
- Get weak hypothesis  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ .
- Aim: select  $h_t$  with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$ .
- Update, for  $i = 1, \dots, m$ :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where  $Z_t$  is a normalization factor (chosen so that  $D_{t+1}$  will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right).$$


---



# AdaBoost algorithm

Given:  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}, y_i \in \{-1, +1\}$ .

Initialize:  $D_1(i) = 1/m$  for  $i = 1, \dots, m$ .

For  $t = 1, \dots, T$ :

- Train weak learner using distribution  $D_t$ .
- Get weak hypothesis  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ .
- Aim: select  $h_t$  with low weighted error:

The error rate of  $h_t$   
calculated over  $D_t$

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$ .
- Update, for  $i = 1, \dots, m$ :

The weight of  $h_t$ . The bigger  $\varepsilon_t$   
outputs smaller  $\alpha_t$ .

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where  $Z_t$  is a normalization factor (chosen so that  $D_{t+1}$  will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right).$$

# AdaBoost algorithm

Given:  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}$ ,  $y_i \in \{-1, +1\}$ .

Initialize:  $D_1(i) = 1/m$  for  $i = 1, \dots, m$ .

For  $t = 1, \dots, T$ :

- Train weak learner using distribution  $D_t$ .
- Get weak hypothesis  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ .
- Aim: select  $h_t$  with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$ .
- Update, for  $i = 1, \dots, m$ :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Incorrectly predicted samples would be given larger weights.

where  $Z_t$  is a normalization factor (chosen so that  $D_{t+1}$  will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right).$$

# AdaBoost algorithm

Given:  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}$ ,  $y_i \in \{-1, +1\}$ .

Initialize:  $D_1(i) = 1/m$  for  $i = 1, \dots, m$ .

For  $t = 1, \dots, T$ :

- Train weak learner using distribution  $D_t$ .
- Get weak hypothesis  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ .
- Aim: select  $h_t$  with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$ .
- Update, for  $i = 1, \dots, m$ :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

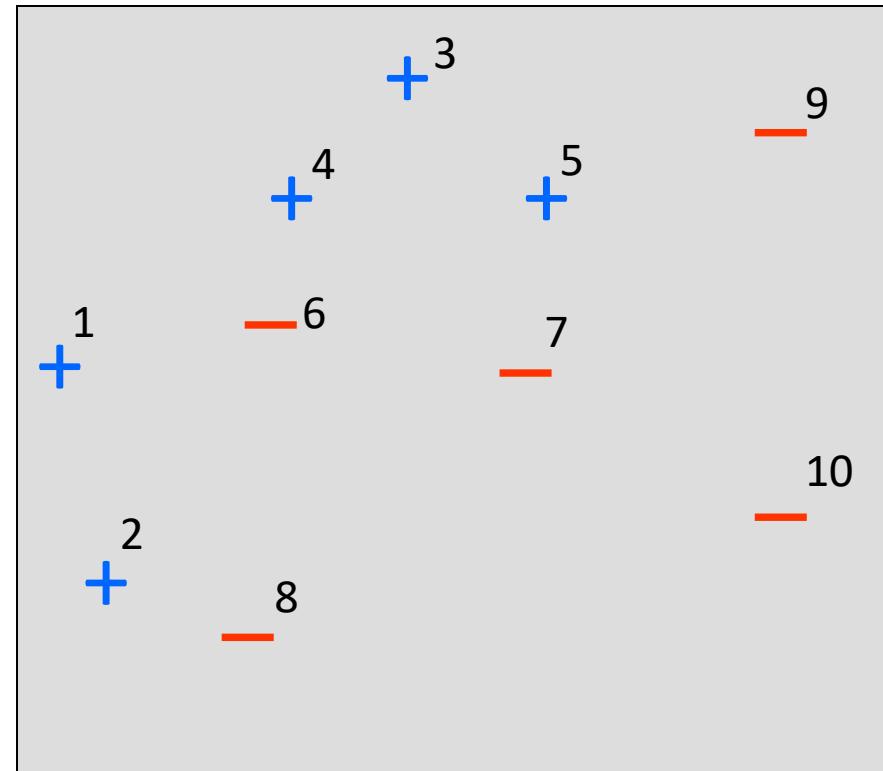
where  $Z_t$  is a normalization factor (chosen so that  $D_{t+1}$  will be a distribution).

Output the final hypothesis:

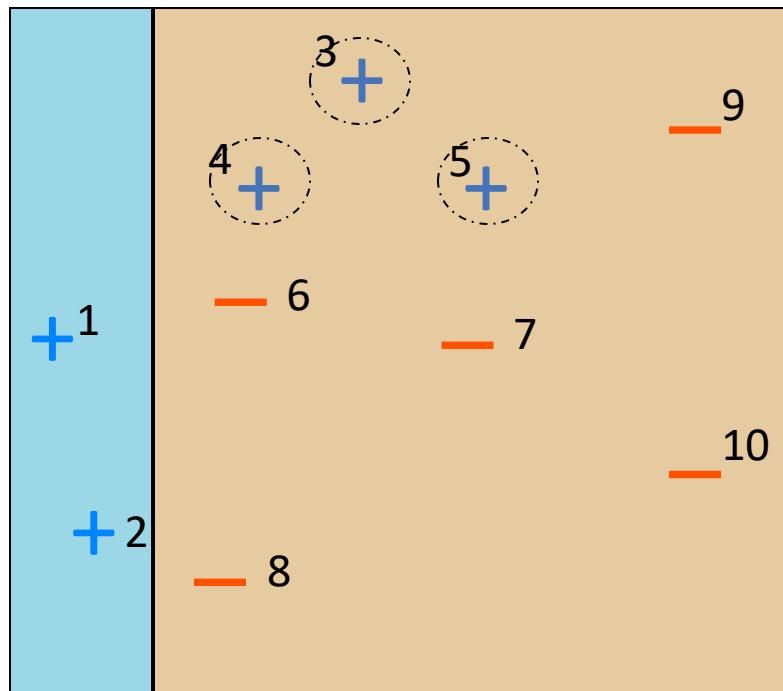
$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right).$$

Linear combination of all weak hypotheses.

# Example



# Round 1 of 3



$$h_1$$

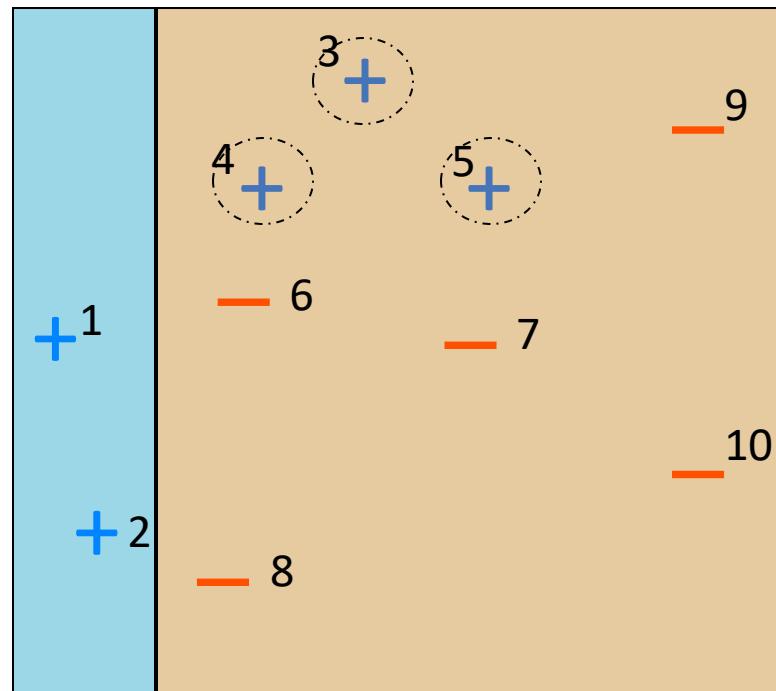
$$\varepsilon_1 = 0.300$$

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

$$\alpha_1 = 0.424$$

Sample ID	$D_1(i)$
1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
10	0.1
Sum weights $Z_t$	

# Round 1 of 3



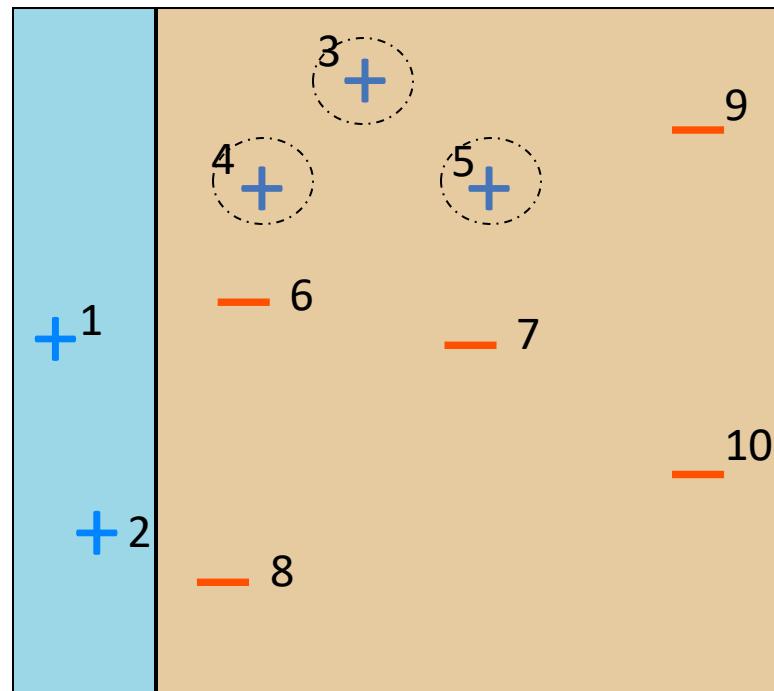
$$h_1 \quad \varepsilon_1 = 0.300$$

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \quad \alpha_1 = 0.424$$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Sample ID	$D_1(i)$	
1	0.1	0.0655
2	0.1	0.0655
3	0.1	0.153
4	0.1	0.153
5	0.1	0.153
6	0.1	0.0655
7	0.1	0.0655
8	0.1	0.0655
9	0.1	0.0655
10	0.1	0.0655
Sum weights $Z_t$		0.9165

# Round 1 of 3



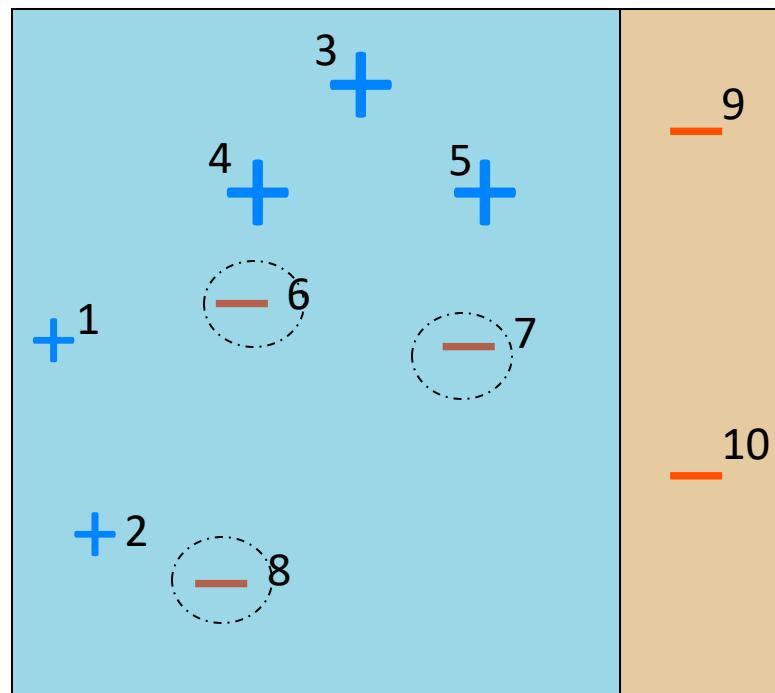
$$h_1 \quad \varepsilon_1 = 0.300$$

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \quad \alpha_1 = 0.424$$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Sample ID	$D_1(i)$		$D_2(i)$
1	0.1	0.0655	0.0714
2	0.1	0.0655	0.0714
3	0.1	0.153	0.1667
4	0.1	0.153	0.1667
5	0.1	0.153	0.1667
6	0.1	0.0655	0.0714
7	0.1	0.0655	0.0714
8	0.1	0.0655	0.0714
9	0.1	0.0655	0.0714
10	0.1	0.0655	0.0714
Sum weights $Z_t$		0.9165	

# Round 2 of 3



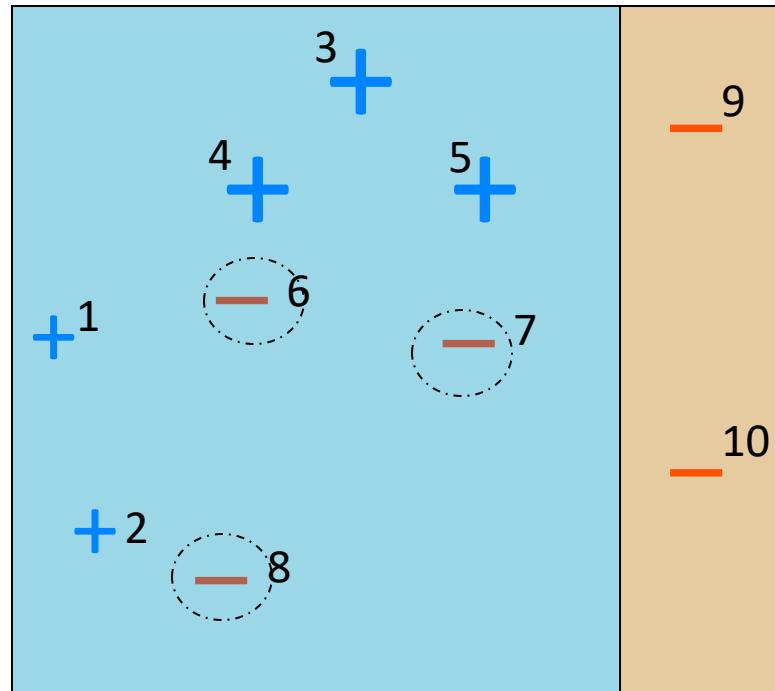
$$\varepsilon_2 = 0.2142 \quad h_2$$

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right) \quad \alpha_2 = 0.6496$$

Sample ID	$D_1(i)$		$D_2(i)$
1	0.1	0.0655	0.0714
2	0.1	0.0655	0.0714
3	0.1	0.153	0.1667
4	0.1	0.153	0.1667
5	0.1	0.153	0.1667
6	0.1	0.0655	0.0714
7	0.1	0.0655	0.0714
8	0.1	0.0655	0.0714
9	0.1	0.0655	0.0714
10	0.1	0.0655	0.0714
Sum weights $Z_t$		0.9165	

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

# Round 2 of 3



$$\varepsilon_2 = 0.2142 \quad h_2$$

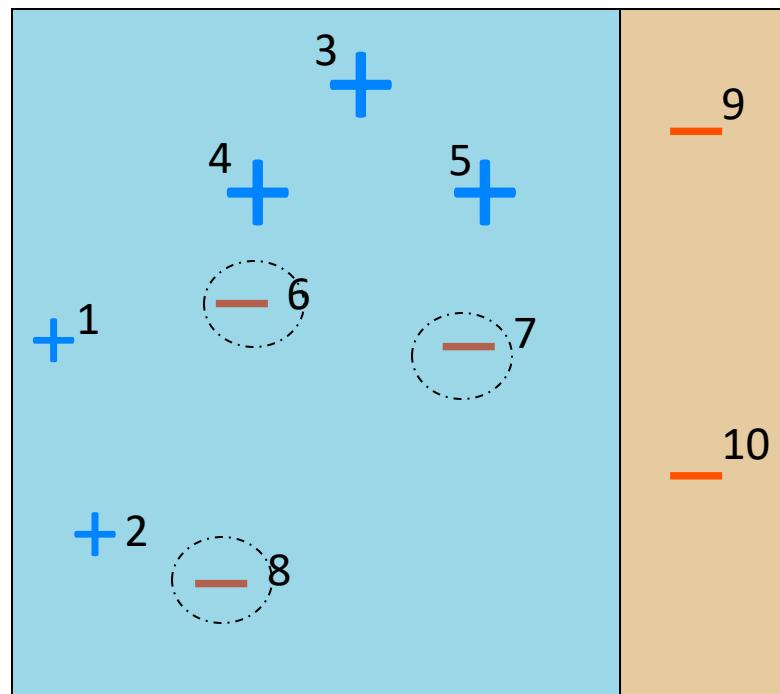
$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right) \quad \alpha_2 = 0.6496$$

Sample ID	$D_1(i)$		$D_2(i)$	
1	0.1	0.0655	0.0714	0.0373
2	0.1	0.0655	0.0714	0.0373
3	0.1	0.153	0.1667	0.0871
4	0.1	0.153	0.1667	0.0871
5	0.1	0.153	0.1667	0.0871
6	0.1	0.0655	0.0714	0.1367
7	0.1	0.0655	0.0714	0.1367
8	0.1	0.0655	0.0714	0.1367
9	0.1	0.0655	0.0714	0.0373
10	0.1	0.0655	0.0714	0.0373
Sum weights $Z_t$		0.9165		0.8206

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$



# Round 2 of 3

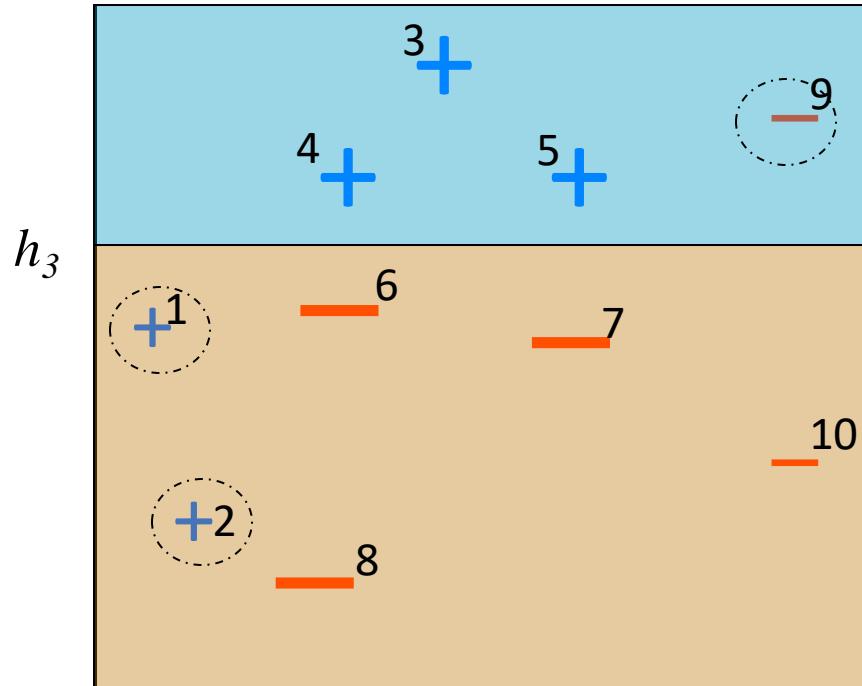


$$\varepsilon_2 = 0.2142 \quad h_2$$

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right) \quad \alpha_2 = 0.6496$$

Sample ID	$D_1(i)$		$D_2(i)$		$D_3(i)$
1	0.1	0.0655	0.0714	0.0373	0.0455
2	0.1	0.0655	0.0714	0.0373	0.0455
3	0.1	0.153	0.1667	0.0871	0.106
4	0.1	0.153	0.1667	0.0871	0.106
5	0.1	0.153	0.1667	0.0871	0.106
6	0.1	0.0655	0.0714	0.1367	0.1666
7	0.1	0.0655	0.0714	0.1367	0.1666
8	0.1	0.0655	0.0714	0.1367	0.1666
9	0.1	0.0655	0.0714	0.0373	0.0455
10	0.1	0.0655	0.0714	0.0373	0.0455
Sum weights $Z_t$		0.9165		0.8206	

# Round 3 of 3



$$\varepsilon_3 = 0.1365$$

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right) \quad \alpha_3 = 0.7378$$

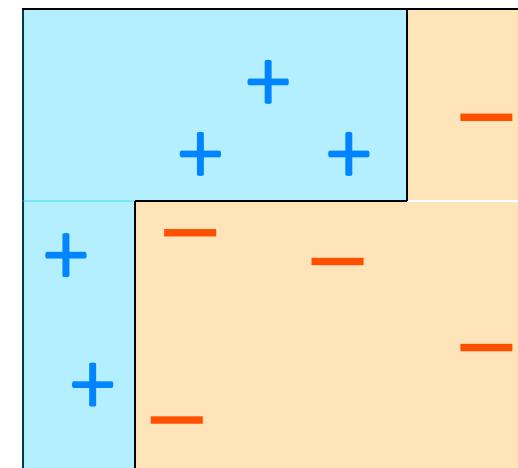
Sample ID	$D_1(i)$		$D_2(i)$		$D_3(i)$
1	0.1	0.0655	0.0714	0.0373	0.0455
2	0.1	0.0655	0.0714	0.0373	0.0455
3	0.1	0.153	0.1667	0.0871	0.106
4	0.1	0.153	0.1667	0.0871	0.106
5	0.1	0.153	0.1667	0.0871	0.106
6	0.1	0.0655	0.0714	0.1367	0.1666
7	0.1	0.0655	0.0714	0.1367	0.1666
8	0.1	0.0655	0.0714	0.1367	0.1666
9	0.1	0.0655	0.0714	0.0373	0.0455
10	0.1	0.0655	0.0714	0.0373	0.0455
Sum weights $Z_t$		0.9165		0.8206	



# Final Hypothesis

0.42      + 0.65      + 0.74

$$H_{\text{final}} = \text{sign}[ 0.42(h1? 1|-1) + 0.65(h2? 1|-1) + 0.74(h3? 1|-1) ]$$





Well Done!



# AdaBoost application: face detection

## Rapid object detection using a boosted cascade of simple features

[PDF] [ieee.org](http://ieee.org)

P Viola, M Jones - ... on computer vision and pattern recognition ..., 2001 - [ieeexplore.ieee.org](http://ieeexplore.ieee.org)

This paper describes a machine learning approach for visual **object detection** which is capable of processing images **extremely rapidly** and achieving high **detection** rates. This work is ...

☆ Save Cite Cited by 26342 Related articles All 103 versions

ACCEPTED CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION 2001

## Rapid Object Detection using a Boosted Cascade of Simple Features

Paul Viola  
[viola@merl.com](mailto:viola@merl.com)  
Mitsubishi Electric Research Labs  
201 Broadway, 8th FL  
Cambridge, MA 02139

Michael Jones  
[mjones@crl.dec.com](mailto:mjones@crl.dec.com)  
Compaq CRL  
One Cambridge Center  
Cambridge, MA 02142

### Abstract

*This paper describes a machine learning approach for visual object detection which is capable of processing images extremely rapidly and achieving high detection rates. This work is distinguished by three key contributions. The first is the introduction of a new image representation called the “Integral Image” which allows the features used by our detector to be computed very quickly. The second is a learning*

tected at 15 frames per second on a conventional 700 MHz Intel Pentium III. In other face detection systems, auxiliary information, such as image differences in video sequences, or pixel color in color images, have been used to achieve high frame rates. Our system achieves high frame rates working only with the information present in a single grey scale image. These alternative sources of information can also be integrated with our system to achieve even higher frame rates.

# Weak classifiers: “Haar-like features”

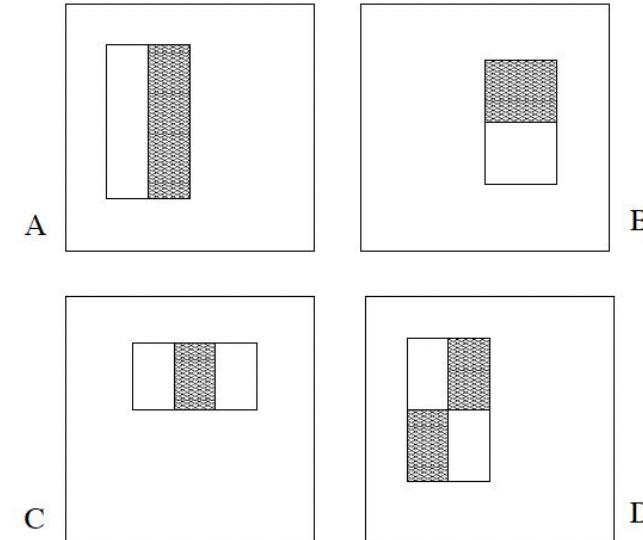
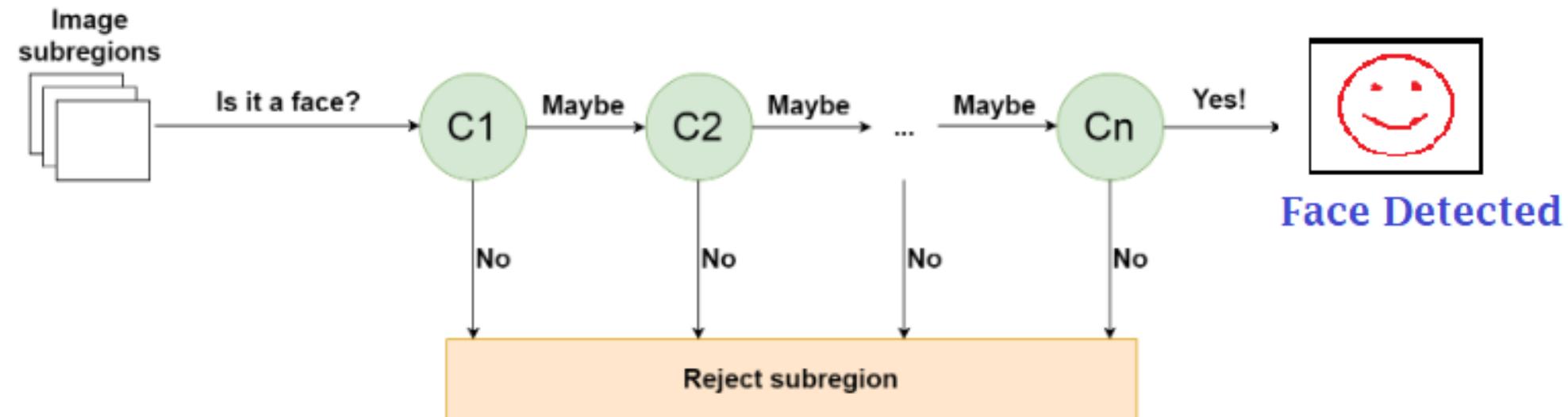


Figure 1: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.



# Cascade of classifiers



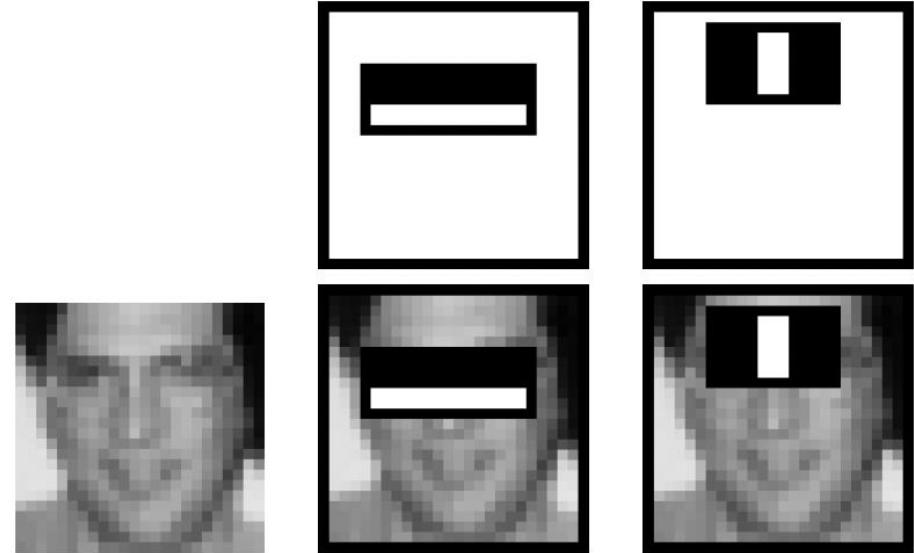
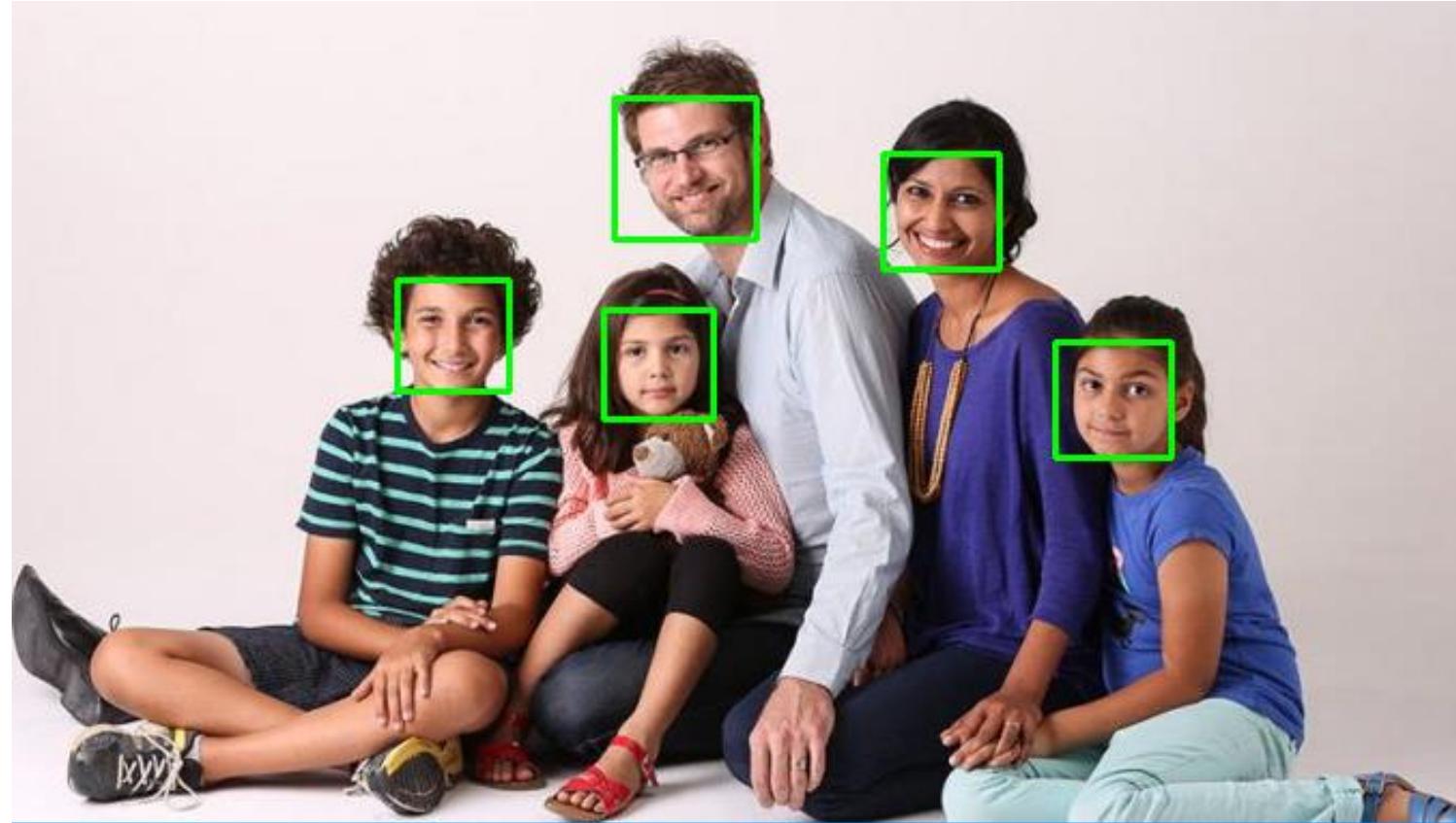


Figure 3: The first and second features selected by AdaBoost. The two features are shown in the top row and then overlayed on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.



# References

- C. McCormick's blog: AdaBoost Tutorial.  
<https://mccormickml.com/2013/12/13/adaboost-tutorial/>
- C. Maklin's blog: AdaBoost Classifier Example In Python  
<https://towardsdatascience.com/machine-learning-part-17-boosting-algorithms-adaboost-in-python-d00faac6c464>
- F. Lopez's blog: Ensemble Learning: Bagging & Boosting.  
<https://towardsdatascience.com/ensemble-learning-bagging-boosting-3098079e5422>
- P. Viola, M. Jones. Rapid object detection using a boosted cascade of simple features. CVPR 2001.

# Week 01: Boosting

CM50265 Machine Learning 2

# Topic 4:

# An Example: Predict your ML2 grade

# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark
1	16	10	Yes	85	90
2	4	4	Yes	60	55
3	15	6	Yes	85	84
4	7	8	No	45	64
5	17	10	No	90	78
6	8	7	Yes	68	67

**Q:** Suppose we have an initial model which outputs a constant value of ML2 marks, what value will it be?

$$\frac{90+55+84+64+78+67}{6} = 73$$

**A:** The average of ML2 marks (Observed values)



# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark	Residual 1
1	16	10	Yes	85	90	17
2	4	4	Yes	60	55	-18
3	15	6	Yes	85	84	11
4	7	8	No	45	64	-9
5	17	10	No	90	78	5
6	8	7	Yes	68	67	-6

Now calculate the residual for each sample.

$$\text{Residual} = \text{Observed} - \text{Predicted}$$

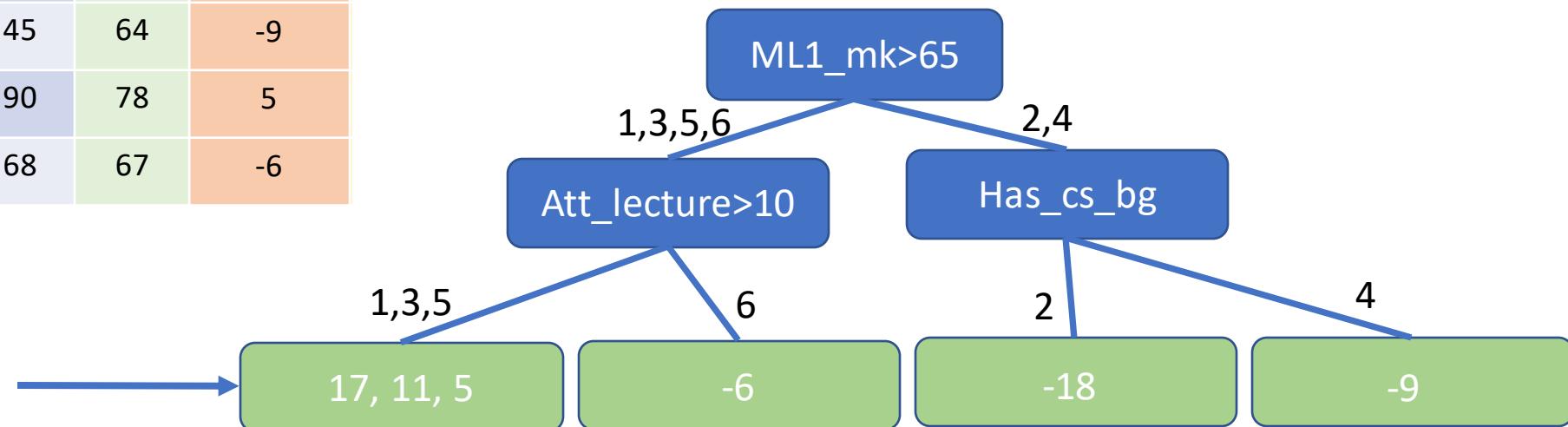
73

# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark	Residual 1
1	16	10	Yes	85	90	17
2	4	4	Yes	60	55	-18
3	15	6	Yes	85	84	11
4	7	8	No	45	64	-9
5	17	10	No	90	78	5
6	8	7	Yes	68	67	-6

Then build a decision tree to fit the residuals.

Take average of all  
the residuals in  
each leaf

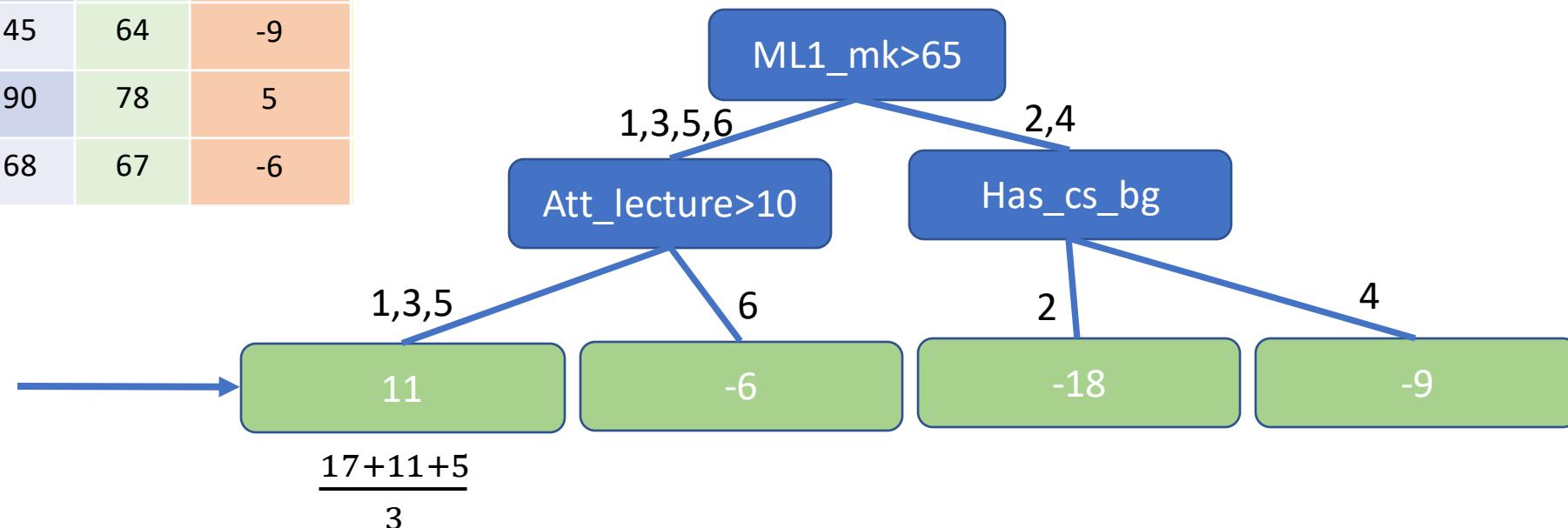


# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark	Residual 1
1	16	10	Yes	85	90	17
2	4	4	Yes	60	55	-18
3	15	6	Yes	85	84	11
4	7	8	No	45	64	-9
5	17	10	No	90	78	5
6	8	7	Yes	68	67	-6

Then build a decision tree to fit the residuals.

Take average of all  
the residuals in  
each leaf

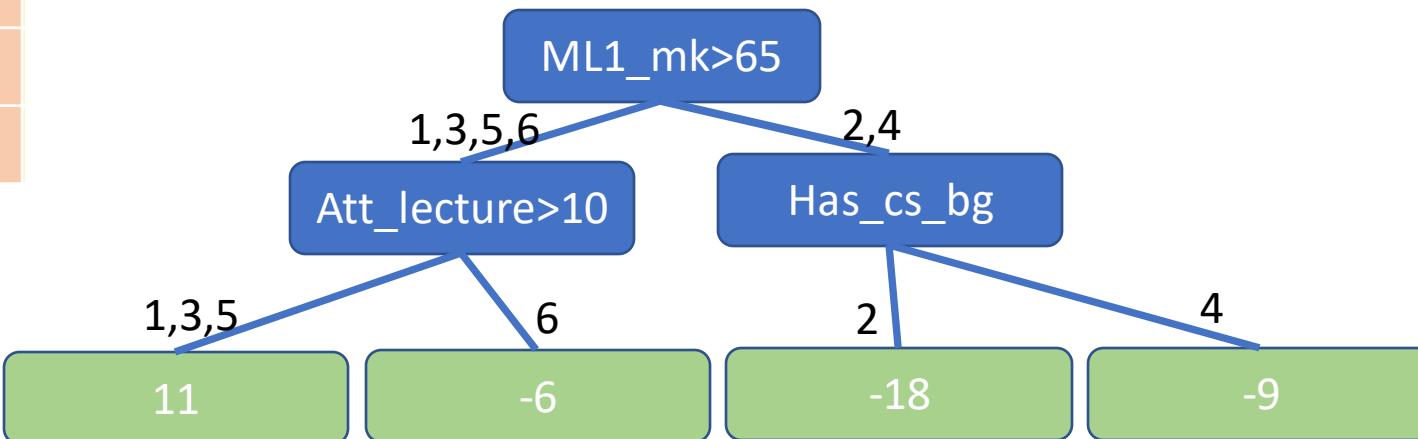


# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark	Residual 1
1	16	10	Yes	85	90	17
2	4	4	Yes	60	55	-18
3	15	6	Yes	85	84	11
4	7	8	No	45	64	-9
5	17	10	No	90	78	5
6	8	7	Yes	68	67	-6

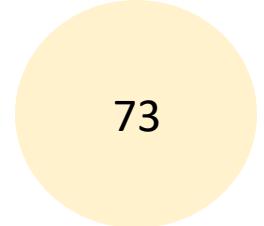
 + 0.1 ×   
 $F_0(x)$   
 Learning rate

The new prediction:  $F_1(x)$

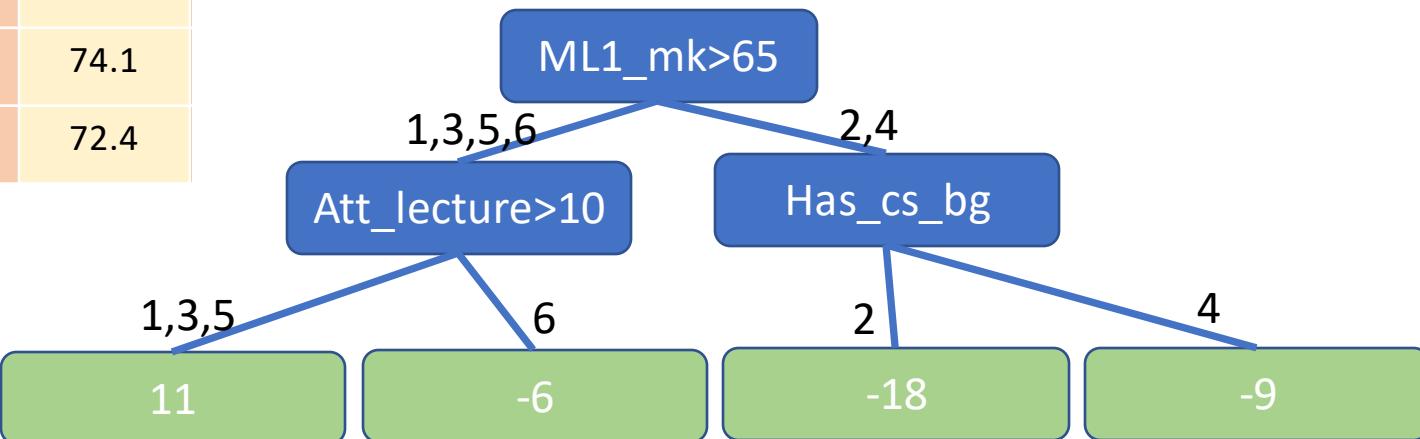


# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark	Residual 1	Predict 1
1	16	10	Yes	85	90	17	74.1
2	4	4	Yes	60	55	-18	71.2
3	15	6	Yes	85	84	11	74.1
4	7	8	No	45	64	-9	72.1
5	17	10	No	90	78	5	74.1
6	8	7	Yes	68	67	-6	72.4

  
 $73 + 0.1 \times$   
 $F_0(x)$   
 Learning rate

The new prediction:  $F_1(x)$



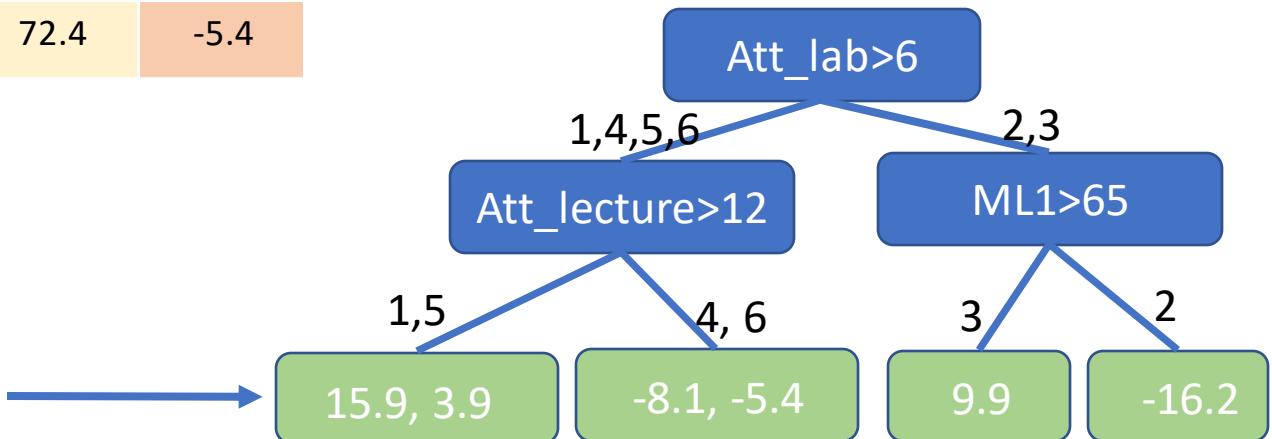
# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark	Residual 1	Predict 1	Residual 2
1	16	10	Yes	85	90	17	74.1	15.9
2	4	4	Yes	60	55	-18	71.2	-16.2
3	15	6	Yes	85	84	11	74.1	9.9
4	7	8	No	45	64	-9	72.1	-8.1
5	17	10	No	90	78	5	74.1	3.9
6	8	7	Yes	68	67	-6	72.4	-5.4

Take average of all the residuals in each leaf



Now calculate the new residual for each sample. Then build another decision tree to fit its residual:

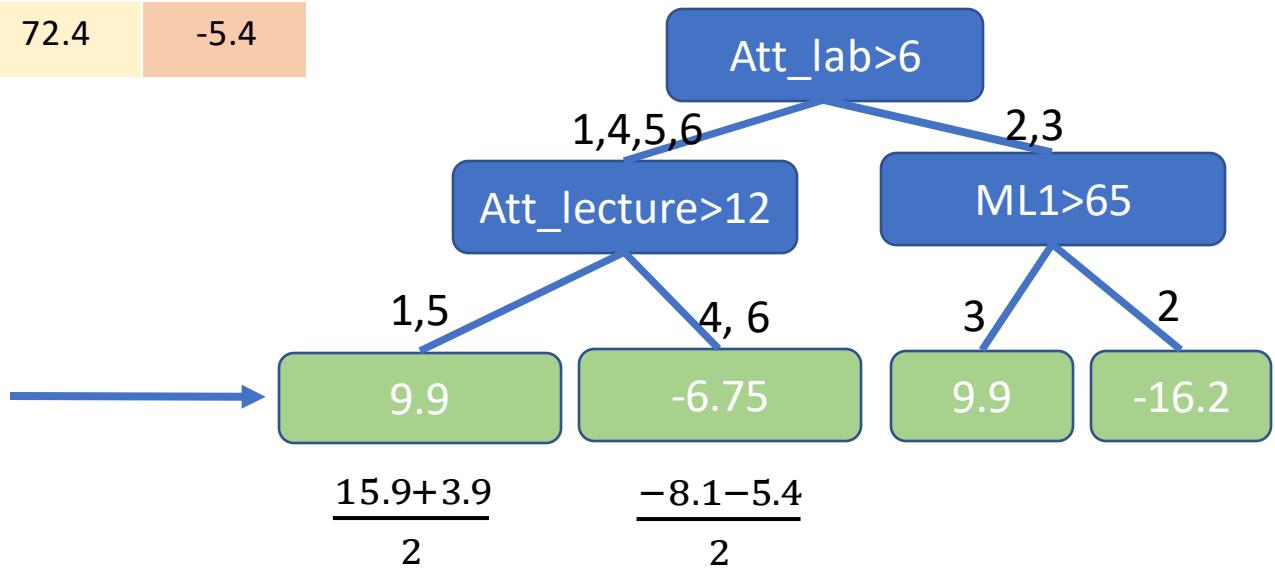


# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark	Residual 1	Predict 1	Residual 2
1	16	10	Yes	85	90	17	74.1	15.9
2	4	4	Yes	60	55	-18	71.2	-16.2
3	15	6	Yes	85	84	11	74.1	9.9
4	7	8	No	45	64	-9	72.1	-8.1
5	17	10	No	90	78	5	74.1	3.9
6	8	7	Yes	68	67	-6	72.4	-5.4

Take average of all the residuals in each leaf

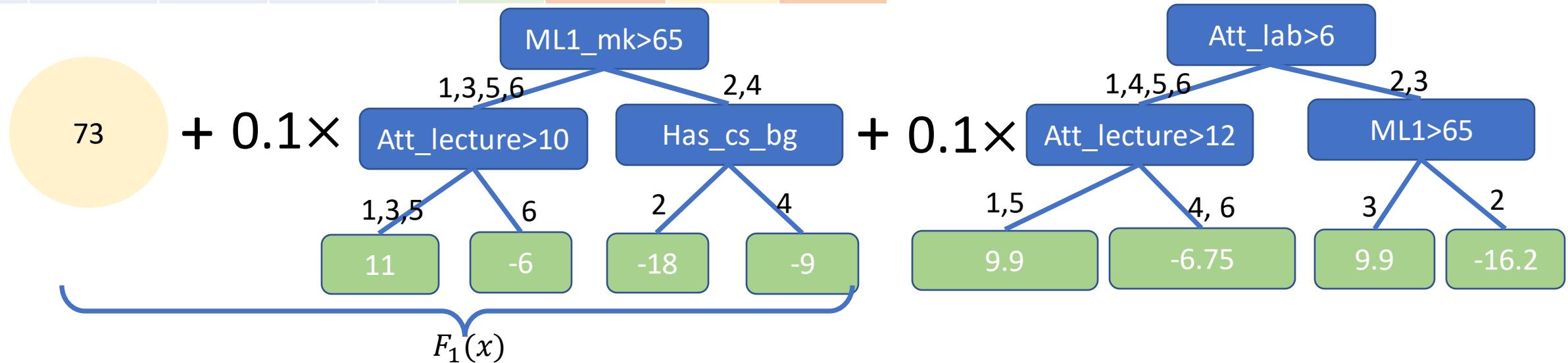
Now calculate the new residual for each sample.  
Then build another decision tree to fit its residual:



# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark	Residual 1	Predict 1	Residual 2
1	16	10	Yes	85	90	17	74.1	15.9
2	4	4	Yes	60	55	-18	71.2	-16.2
3	15	6	Yes	85	84	11	74.1	9.9
4	7	8	No	45	64	-9	72.1	-8.1
5	17	10	No	90	78	5	74.1	3.9
6	8	7	Yes	68	67	-6	72.4	-5.4

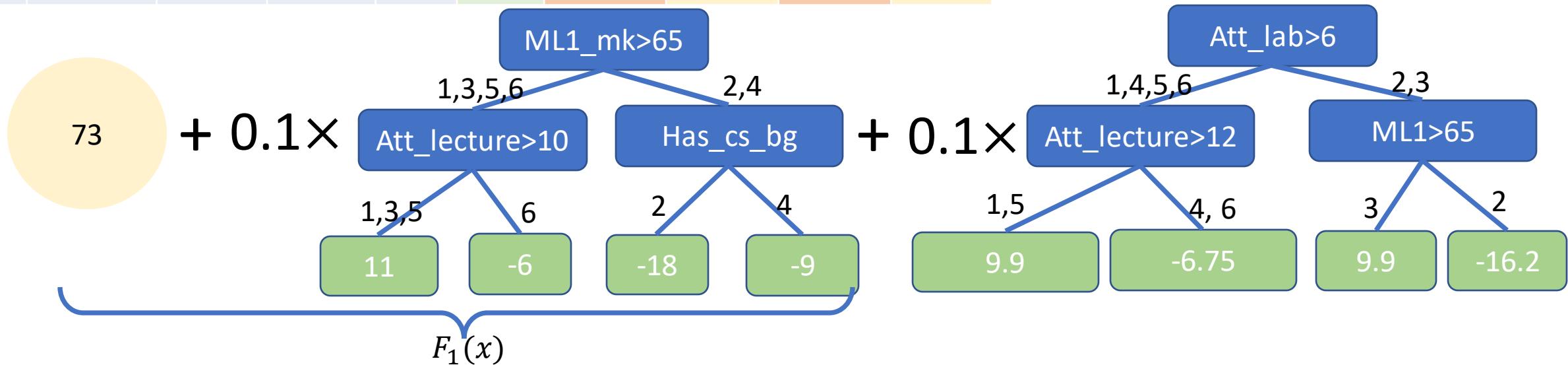
New prediction:  $F_2(x)$



# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	CS Bg	ML1 Mark	ML2 Mark	Residual 1	Predict 1	Residual 2	Predict 2
1	16	10	Yes	85	90	17	74.1	15.9	75.09
2	4	4	Yes	60	55	-18	71.2	-16.2	69.58
3	15	6	Yes	85	84	11	74.1	9.9	73.11
4	7	8	No	45	64	-9	72.1	-8.1	71.425
5	17	10	No	90	78	5	74.1	3.9	75.09
6	8	7	Yes	68	67	-6	72.4	-5.4	71.725

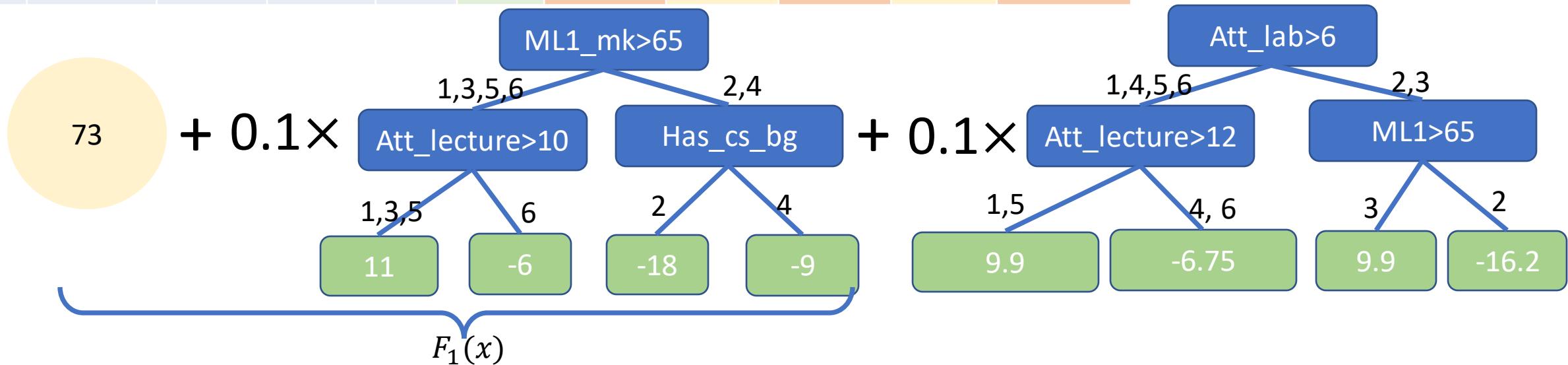
New prediction:  $F_2(x)$



# Training a Gradient Boosting model

	Attendance of lectures	Attendance of labs	Prior course	ML1 Mark	ML2 Mark	Residual 1	Predict 1	Residual 2	Predict 2	Residual 3
1	16	10	Yes	85	90	17	74.1	15.9	75.09	14.91
2	4	4	Yes	60	55	-18	71.2	-16.2	69.58	-14.58
3	15	6	Yes	85	84	11	74.1	9.9	73.11	10.89
4	7	8	No	45	64	-9	72.1	-8.1	71.425	-7.425
5	17	10	No	90	78	5	74.1	3.9	75.09	2.91
6	8	7	Yes	68	67	-6	72.4	-5.4	71.725	-4.725

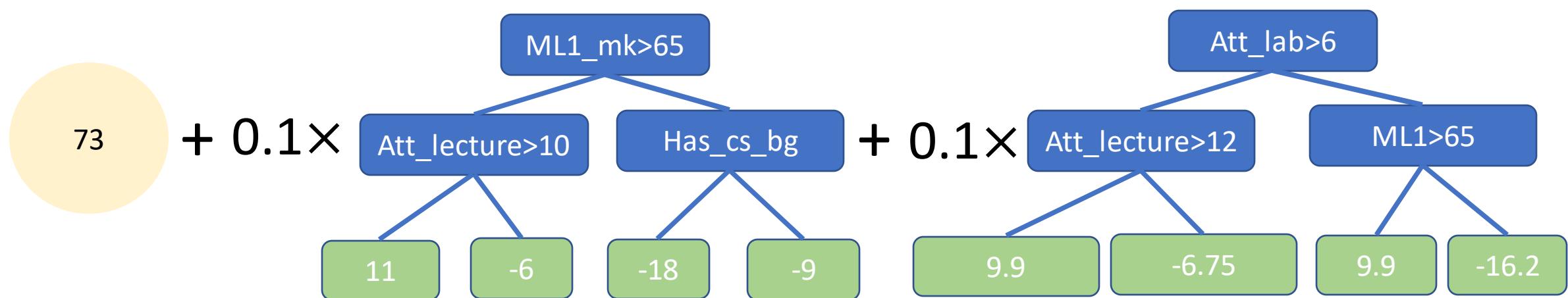
New prediction:  $F_2(x)$



# Now let's predict a new sample

	Attendance of lectures	Attendance of labs	Prior course	ML1 Mark	ML2 Mark
	14	5	Yes	80	?

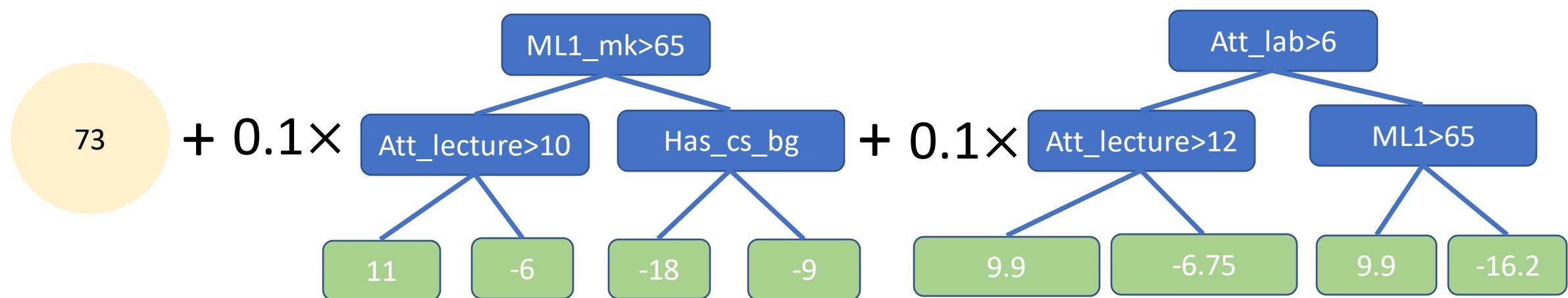
Final model:  $F_2(x)$



# Now let's predict a new sample

	Attendance of lectures	Attendance of labs	Prior course	ML1 Mark	ML2 Mark
	14	5	Yes	80	75.09

Final model:  $F_2(x)$



# Topic 5: Gradient Boosting

**Step 1** Initialize model  $F_0(x)$  with a constant value for prediction.

$F_0(x) =$  The average of the observed values

**WHY?**

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma).$$

MSE loss function for regression:  $J = \frac{1}{2} \sum_{i=1}^n (y_i - \gamma)^2$

$$\frac{\partial J}{\partial \gamma} = 0$$

$$\frac{\partial J}{\partial \gamma} = \frac{2}{2} \sum_{i=1}^n (y_i - \gamma) = 0$$



$$\gamma = \frac{1}{n} \sum_{i=1}^n y_i$$

**Step 2-1 Calculate the residual for each sample  
(Observed – Predicted)****WHY (Observed – Predicted)?**

Negative gradient

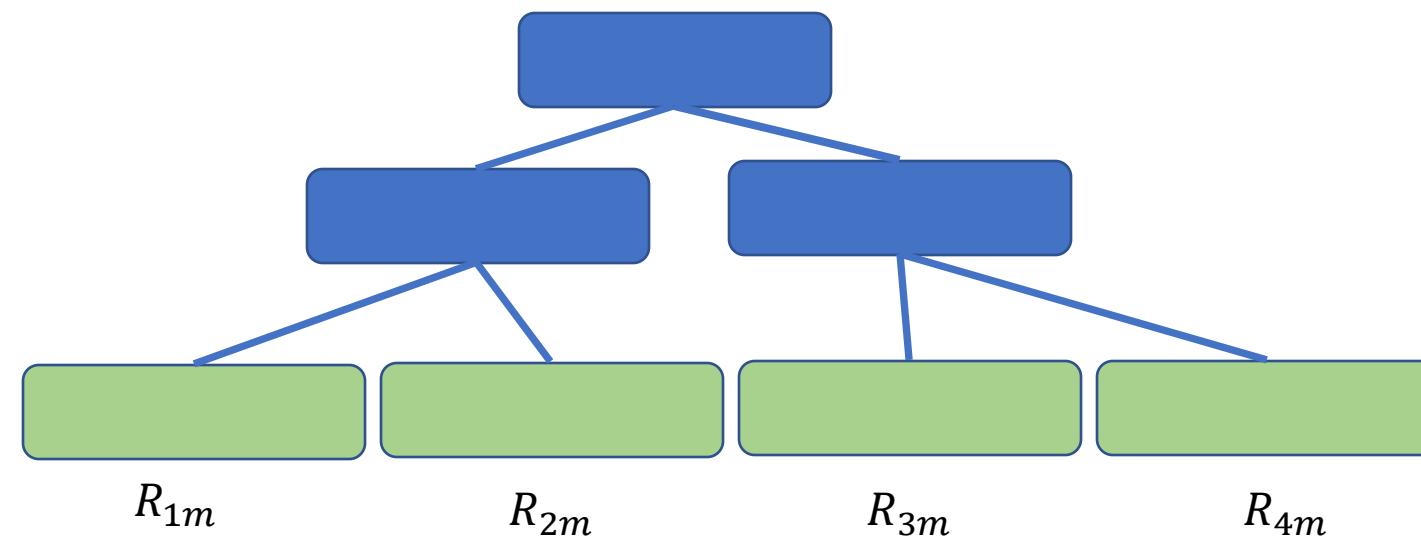
$$r_{im} = - \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \dots, n.$$

$$L(y_i - F(x_i)) = \frac{1}{2} (y_i - F(x_i))^2$$

$$r_{im} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$$

$$\rightarrow r_{im} = y_i - F_{m-1}(x_i)$$

**Step 2-2** Build a decision tree  $h_m(x)$  to fit the residuals and create terminal nodes  $R_{jm}, j = 1, \dots, J_m$



**Step 2-3** The output value  $\gamma_{jm}$  of  $j$ -th leaf is the average of all the residuals in that leaf.

**WHY average of all the residuals?**

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$$

$$= \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma)^2$$

$$\frac{\partial}{\partial \gamma} \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma)^2 = 0$$

$$-2 \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma) = 0$$

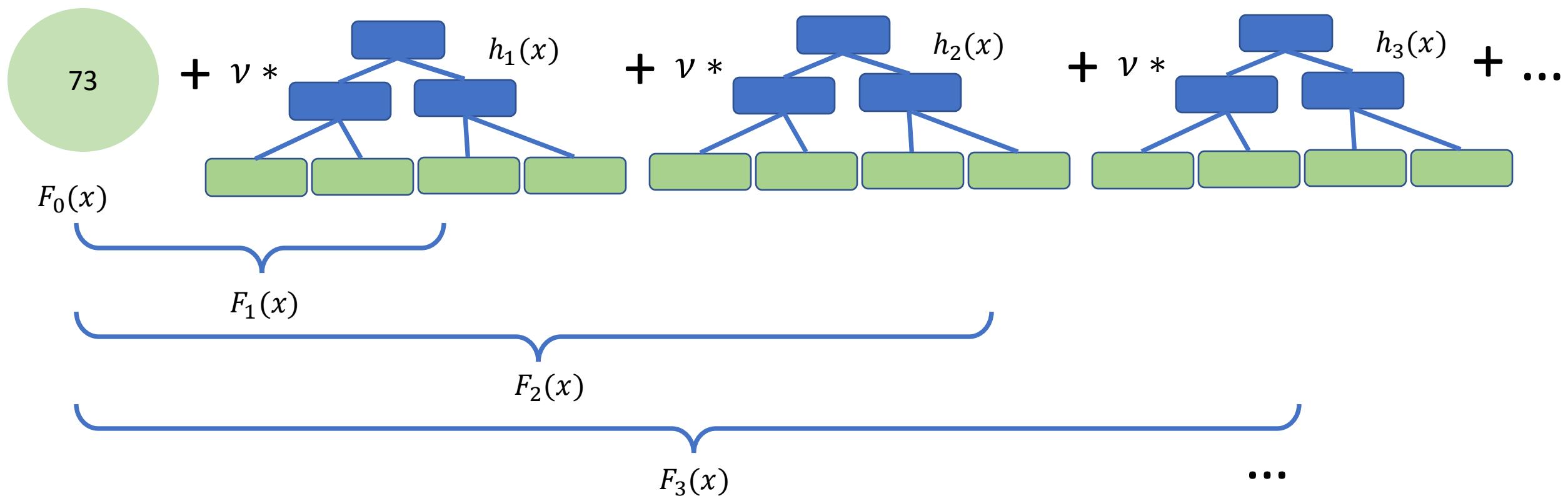
$$n_j \gamma = \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i))$$

$$\rightarrow \gamma = \frac{1}{n_j} \sum_{x_i \in R_{jm}} r_{im}$$

The number of samples in  $j$ -th leaf

Step 2-4 Update the model by adding the current tree with a factor.

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x \in R_{jm})$$





## Gradient Boosting Algorithm

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, \gamma)$$

2. for  $m = 1$  to  $M$ :

2-1. Compute residuals  $r_{im} = - \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}$  for  $i = 1, \dots, n$

Negative gradient

2-2. Train regression tree with features  $x$  against  $r$  and create terminal node regions  $R_{jm}$  for  $j = 1, \dots, J_m$

2-3. Compute  $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$  for  $j = 1, \dots, J_m$

2-4. Update the model:

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x \in R_{jm})$$

# Gradient boosting variants

- Through this approach, one could consider a variety of loss functions.
  - MSE loss, MAE loss ← Regression problem
  - Cross-entropy loss

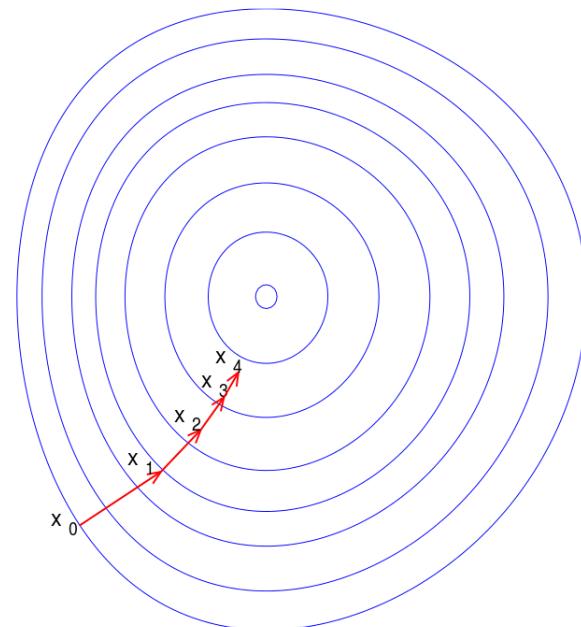
$$L = - (y_i \cdot \log(p) + (1 - y_i) \cdot \log(1 - p))$$

← Classification problem

# How is Gradient boosting related to Gradient Descent?

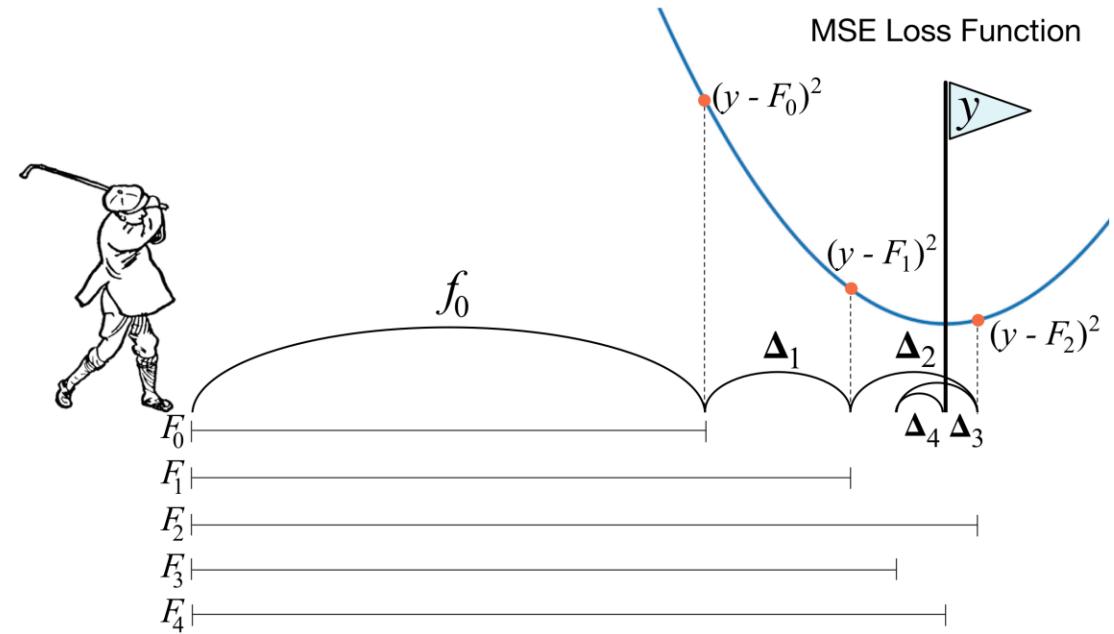
In Gradient Descent, if we want to minimise a function  $f(x)$ , the fastest way is to proceed in the direction of **negative of the gradient** of function

$$x_t = x_t - \eta \nabla f(x)$$



In Gradient boosting, we aim to minimize some loss function between the observed values and the prediction. We update the prediction by fitting the residual – **negative of gradient**.

$$F_m(x) = F_{m-1}(x) + v h_m(x)$$



Img from: [https://en.wikipedia.org/wiki/Gradient\\_descent](https://en.wikipedia.org/wiki/Gradient_descent)

Img from: <https://explained.ai/gradient-boosting/index.html>

**Q:** What is the similarity and different between Adaboost and Gradient boost?

# References

- Statquest videos that explain the Gradient boosting:  
<https://www.youtube.com/watch?v=3CC4N4z3GJc>
- Tomonori Masui's blog: All you need to know about gradient boosting algorithm. <https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502>
- Terence Parr and Jeremy Howard, How to explain gradient boosting.  
<https://explained.ai/gradient-boosting/index.html>

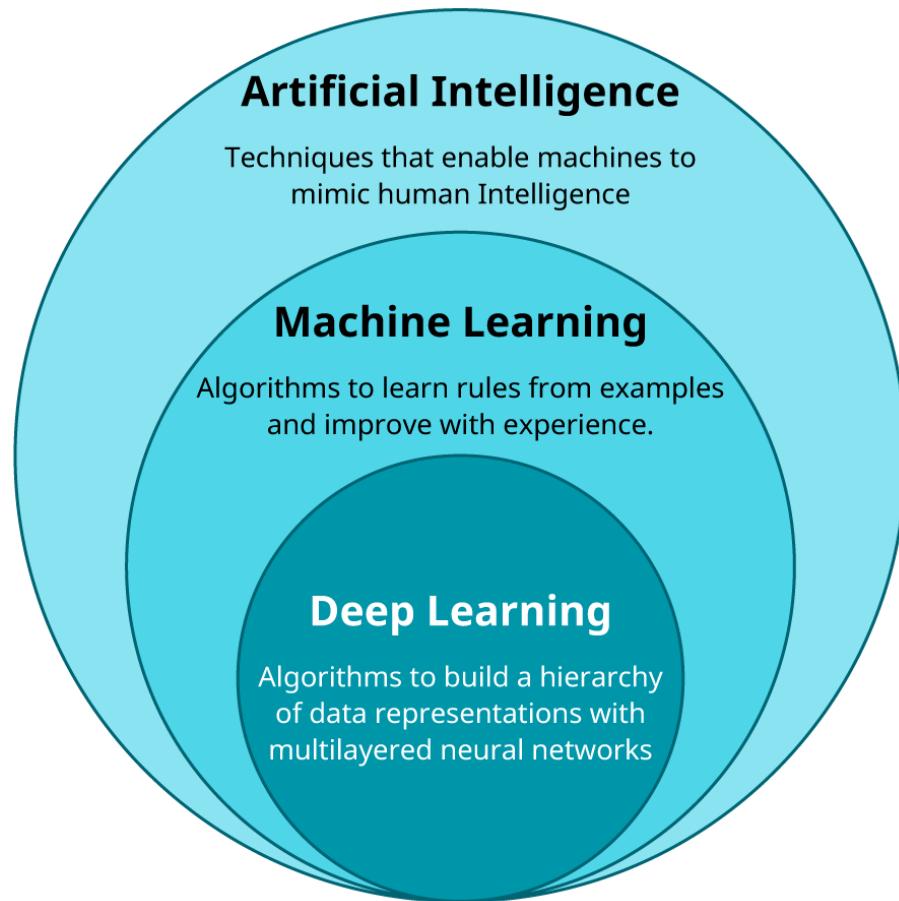
# Week 02:

# Multi Layer Perceptron (MLP)

## CM50265 Machine Learning 2

# Topic 1: Introduction to Deep Learning

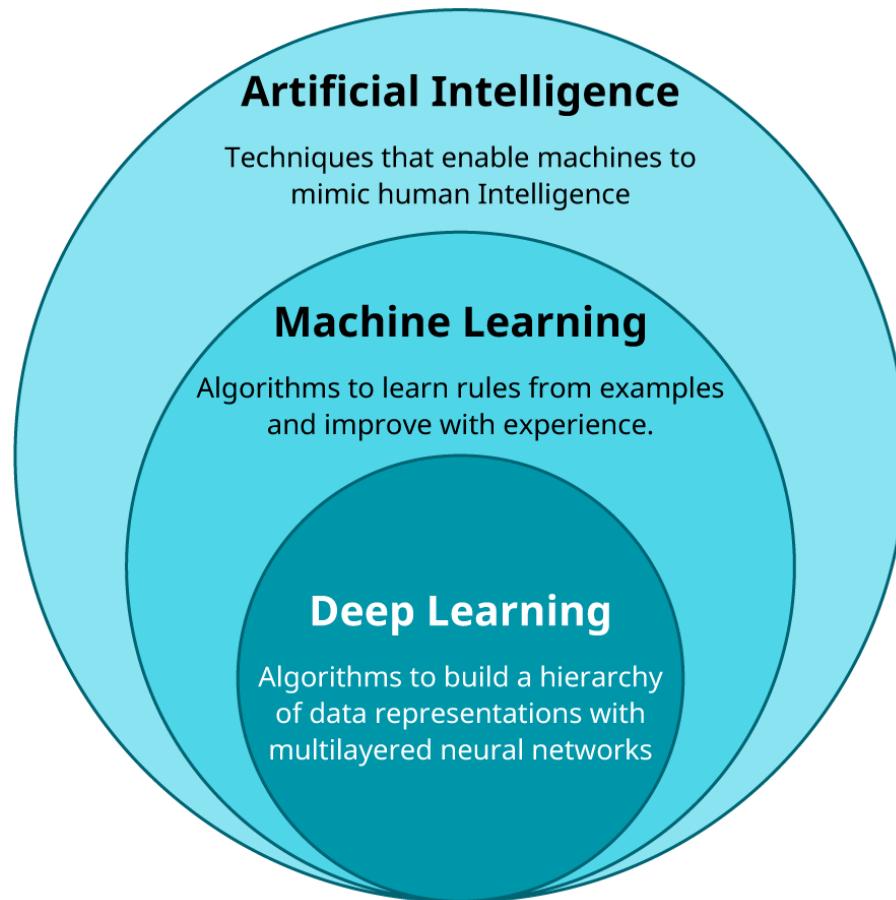
# AI vs Machine Learning vs Deep Learning



- From 1950s to late 1980s, the rule-based or expert system was the most popular solution to AI.
- However, it is intractable to figure out explicit rules for most real-life problems. E.g., how does a toddler start to know a giraffe is a giraffe?



# AI vs Machine Learning vs Deep Learning



- **Deep learning**, a subfield of machine learning, refers to algorithms to build a hierarchy of data representations with multi-layered neural networks.



# Shocking news



**2016, AlphaGo Vs Lee Sedol**

**4 : 1**



**2017, AlphaGo Vs Ke Jie**

**3 : 0**



# AI-generated Art

Won an Art Competition in 2022  
Jason Allison  
Théâtre D'opéra Spatial

\$750

Colorado State Fair





Img from: [https://www.reddit.com/r/ChatGPT/comments/10vac7y/how\\_i\\_feel\\_when\\_chatgpt\\_is\\_at\\_capacity\\_and\\_i\\_go/](https://www.reddit.com/r/ChatGPT/comments/10vac7y/how_i_feel_when_chatgpt_is_at_capacity_and_i_go/)



Google search interest in “ChatGPT”



Create a list of 3 startup ideas in enterprise B2B SaaS. The startup ideas should have a strong and compelling mission and also use AI in some way. Avoid cryptocurrency or blockchain. The startup ideas should have a cool and interesting name. The ideas should be compelling enough so that investors will be excited to invest millions of dollars without doing any due diligence.



1. "Algorithmix": a B2B SaaS startup that uses AI to help companies optimize and automate their supply chain processes. Algorithmix helps businesses make data-driven decisions, reduce costs, and improve efficiency in their supply chain operations.
2. "AI-Assist": a B2B SaaS startup that uses AI to help businesses improve their customer service operations. AI-Assist uses natural language processing and machine learning to provide real-time assistance to customer service agents, helping them resolve customer issues faster and more effectively.
3. "BrainBoost": a B2B SaaS startup that uses AI to help businesses improve the productivity and performance of their employees. BrainBoost uses machine learning to analyze employee data and provide personalized training and development recommendations, helping employees learn new skills and grow in their careers.



Say I go with the first idea, Algorithmix. Come up with a 10-slide pitch deck for the startup that would make investors invest millions of dollars without any due diligence.



1. Title slide: "Introducing Algorithmix: Optimizing and Automating the Supply Chain with AI"
2. Problem slide: The supply chain is a critical, but complex and often inefficient process for

# ChatGPT

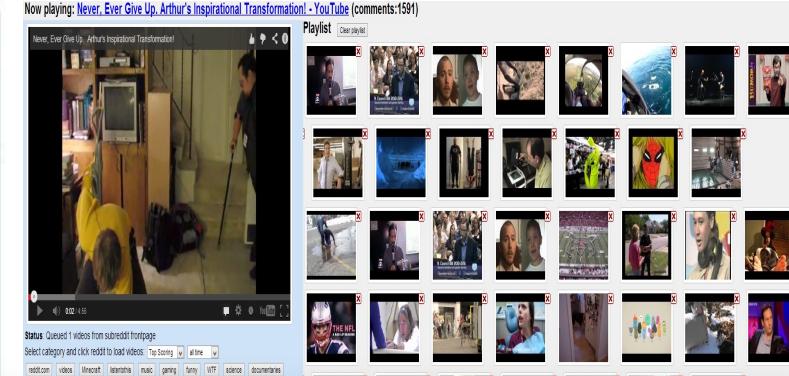
By OpenAI in Nov. 2022



# Deep learning applications



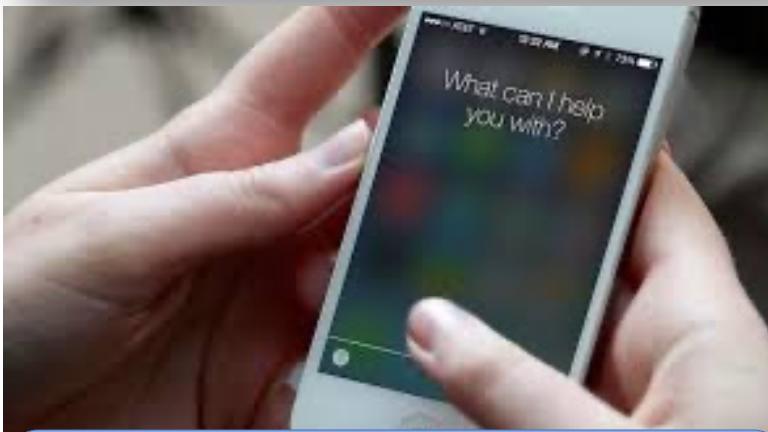
Recommendation



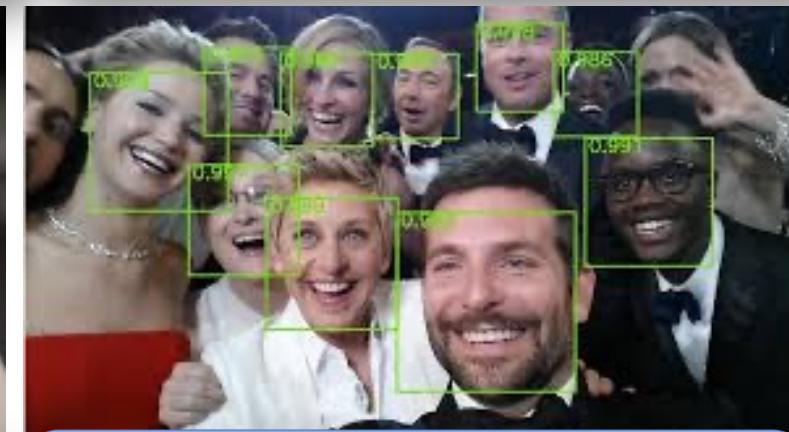
Video/image search



Self-driving car



NLP



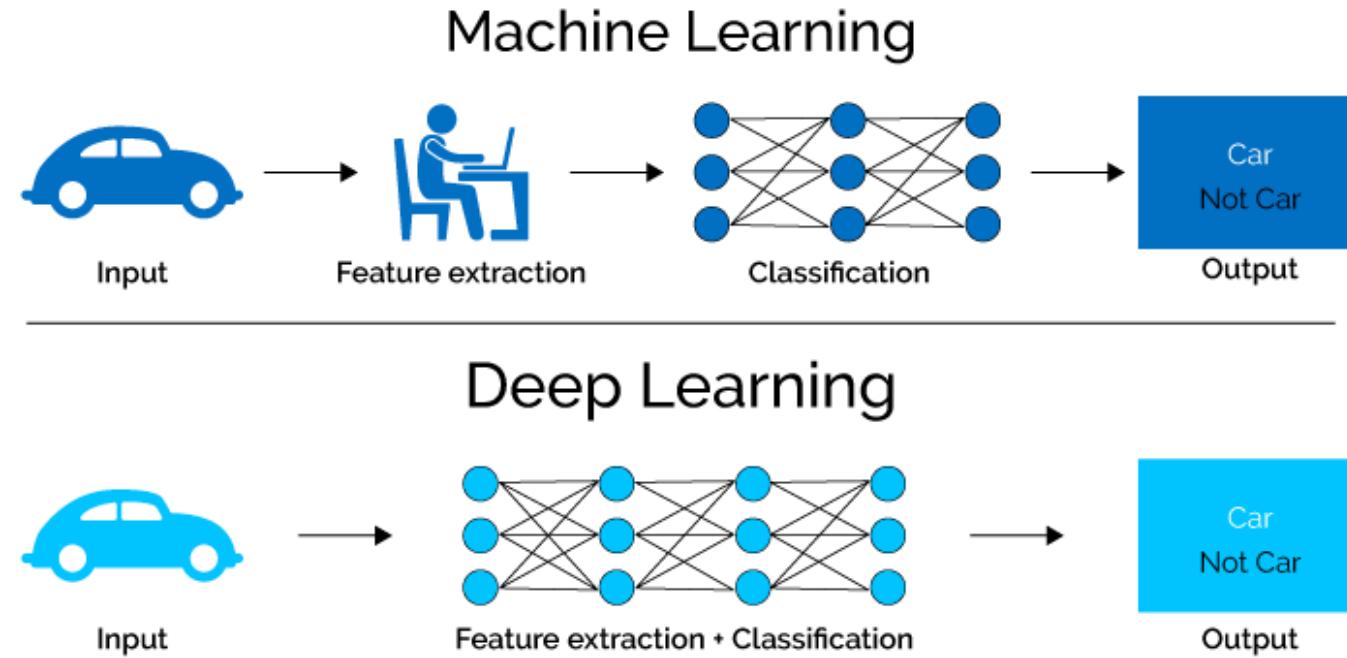
Object detection/recognition



Text to Image



# Why Deep Learning?

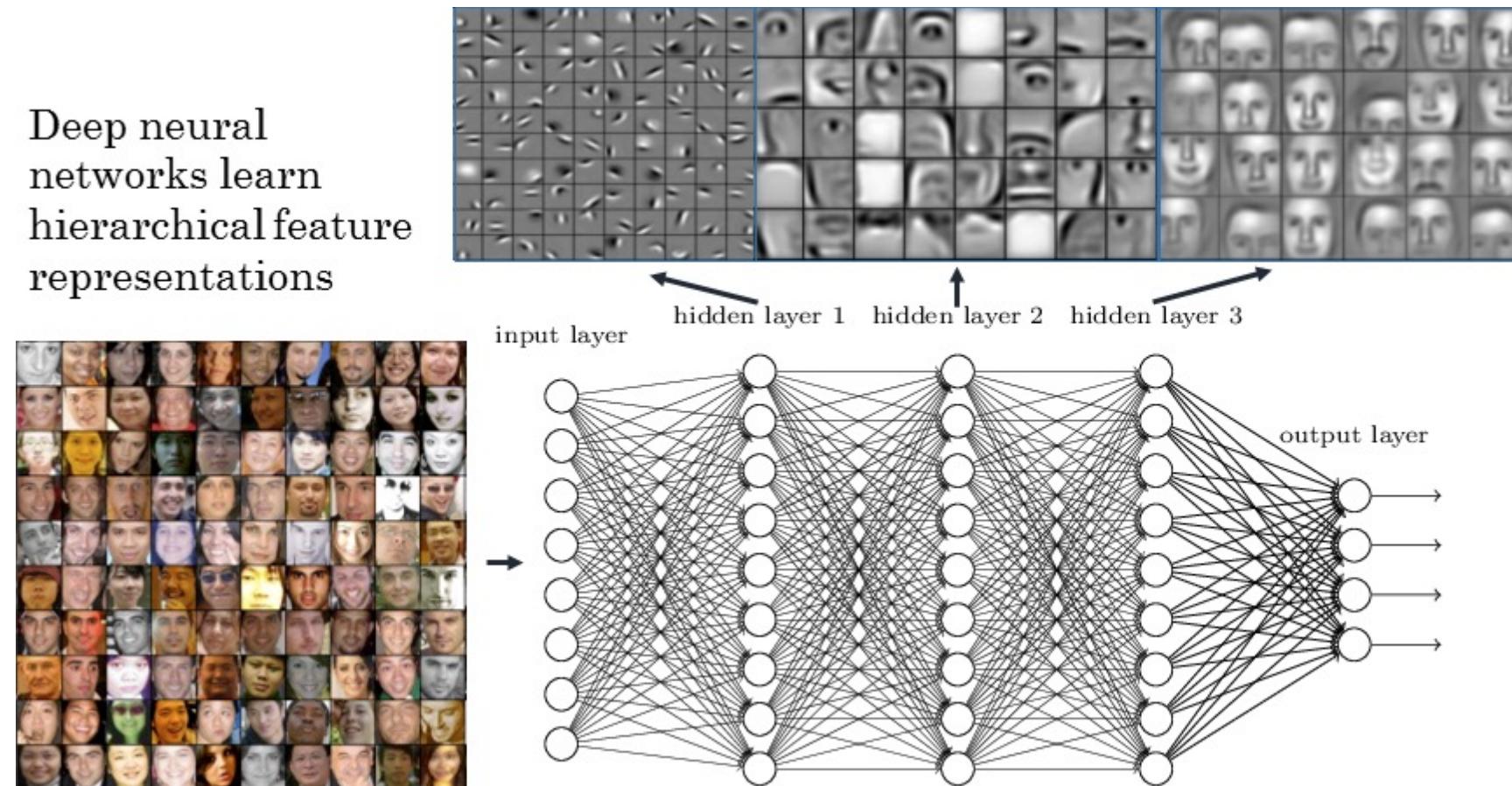


- Deep learning is an **end-to-end** process.
- It learns the **hierarchical features** from the raw data **automatically**.



# Why Deep Learning?

- The hierarchical structure represents **different levels** of features.

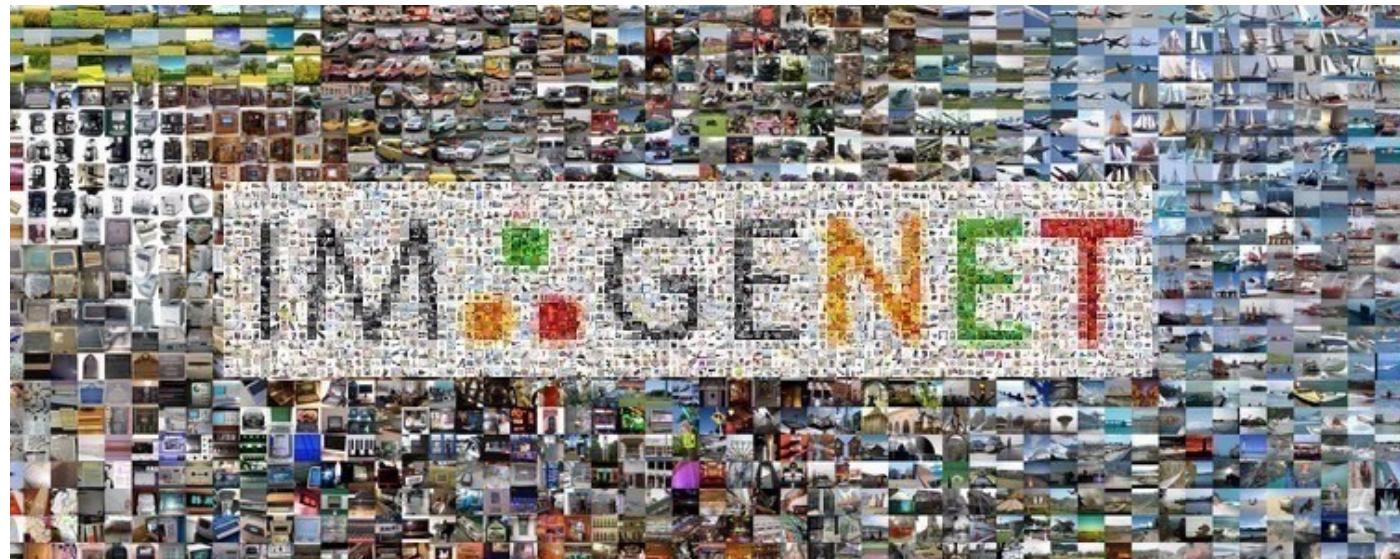


- Key ingredients of DL have been around for a while!
  - Neural networks, 1940s
  - Stochastic Gradient Descent (SGD) for optimization, 1950s
  - Convolutional Neural Networks (CNN), 1989
  - Long short-term memory (LSTM), 1997
  - ...

# Why Now?

# Why Now?

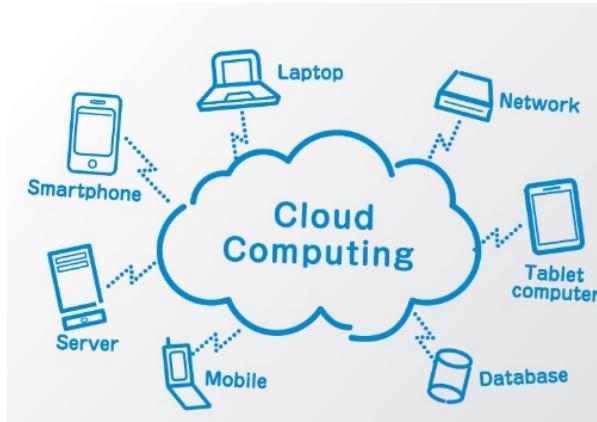
- Big Data
  - Thanks to the internet and the smartphone popularity, it is feasible to collect and store huge amount of data.



**Milestone:** ImageNet dataset (2010), 1.4 million images, 1000 categories

# Why Now?

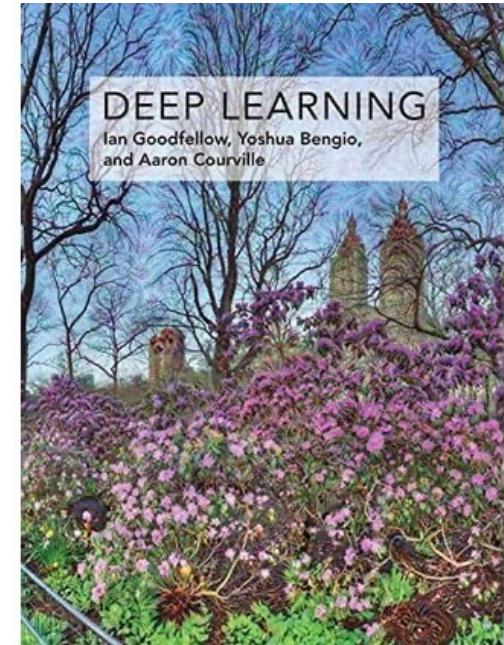
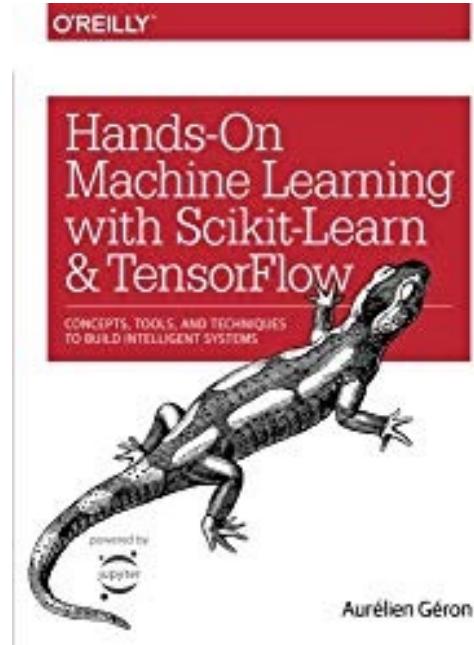
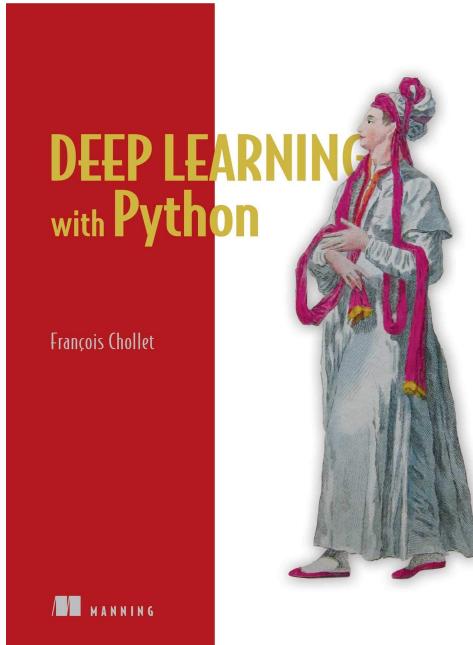
- Hardware
  - GPU, even TPU (2016)
  - Cloud services
- Softwares and algorithms
  - Caffe, Theano, Tensorflow, Keras, ...
  - Several simple but important algorithm improvements: activation functions, weight-initialization schemes, optimization schemes, etc.





# Reading List

- Books

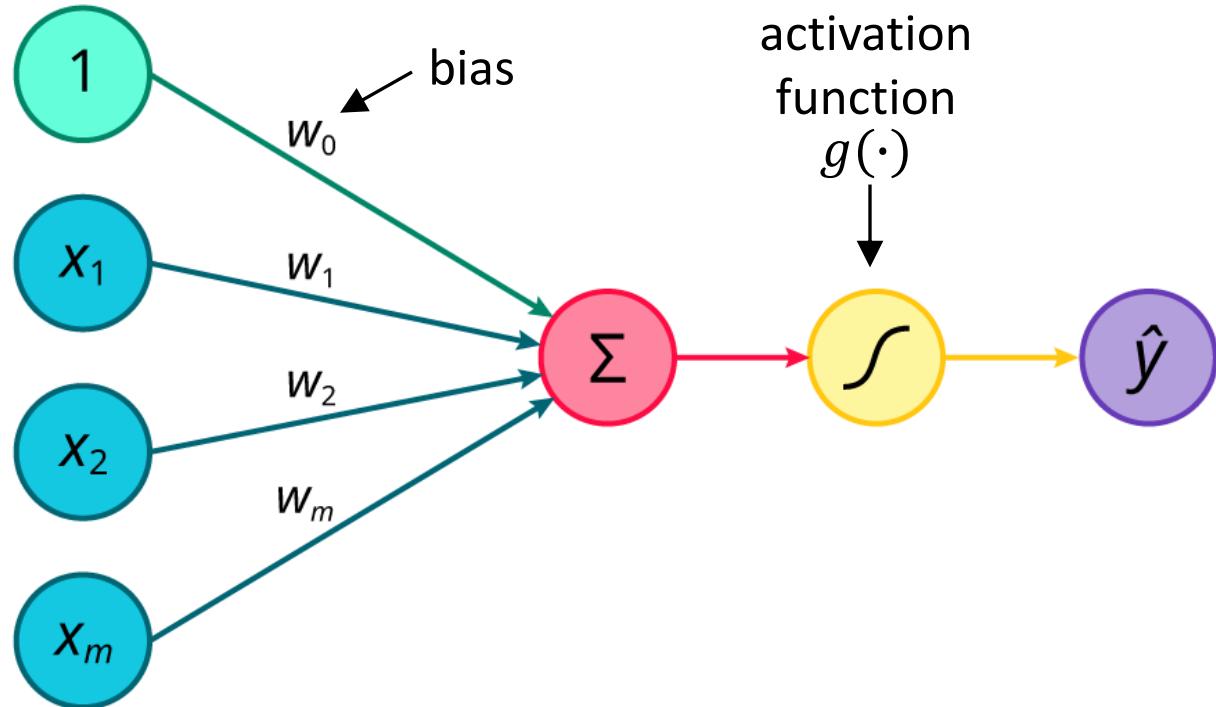


- Online tutorials

- <http://introtodeeplearning.com>
- <http://cs231n.stanford.edu>
- ...

# Topic 2: Perceptron

# Perceptron

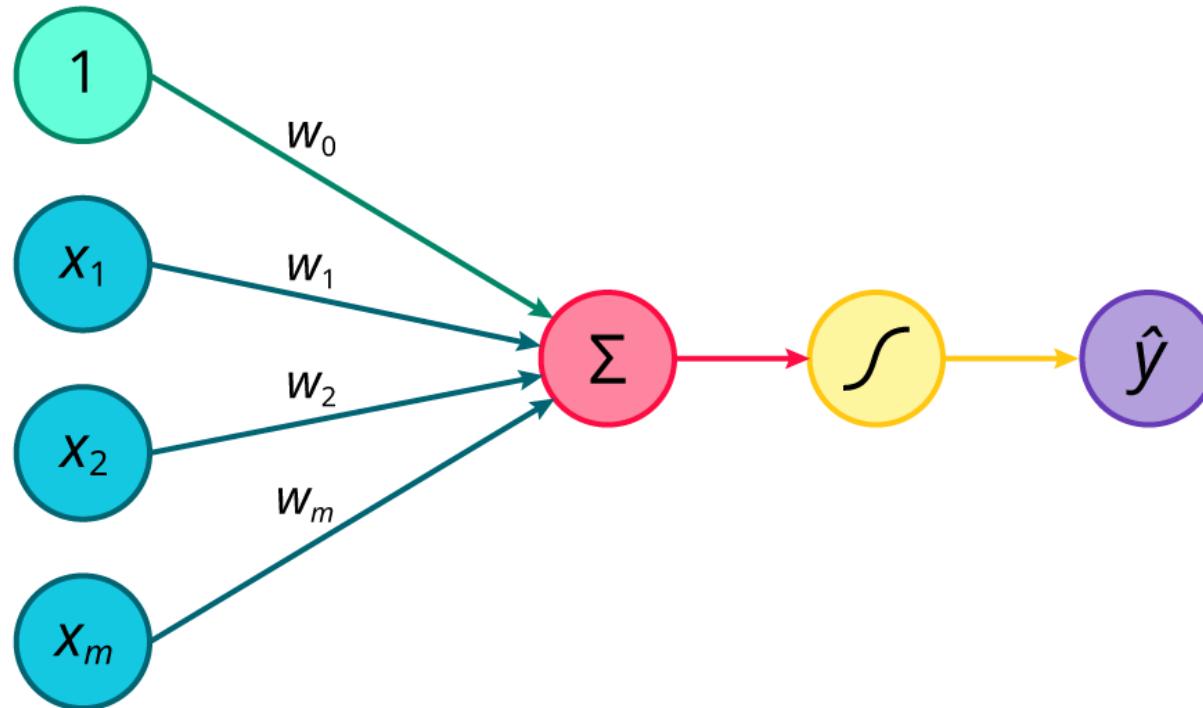


$$\hat{y} = g(w_0 + \sum_{i=1}^m x_i \cdot w_i)$$

Inputs      Weights      Sum      Non-Linearity      Output



# Perceptron

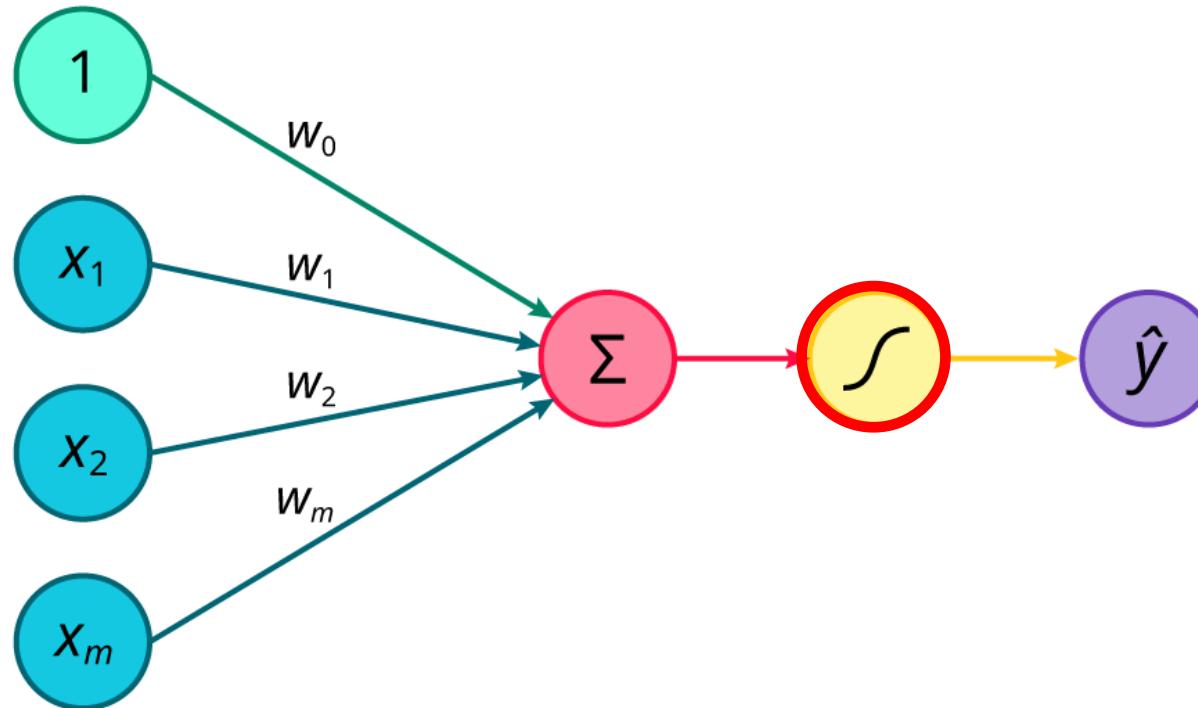


Re-write it with linear algebra:

$$\hat{y} = g(w_0 + X^T W)$$

where  $X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$   $W = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

# Perceptron



Inputs      Weights      Sum      Non-Linearity      Output

Re-write it with linear algebra:

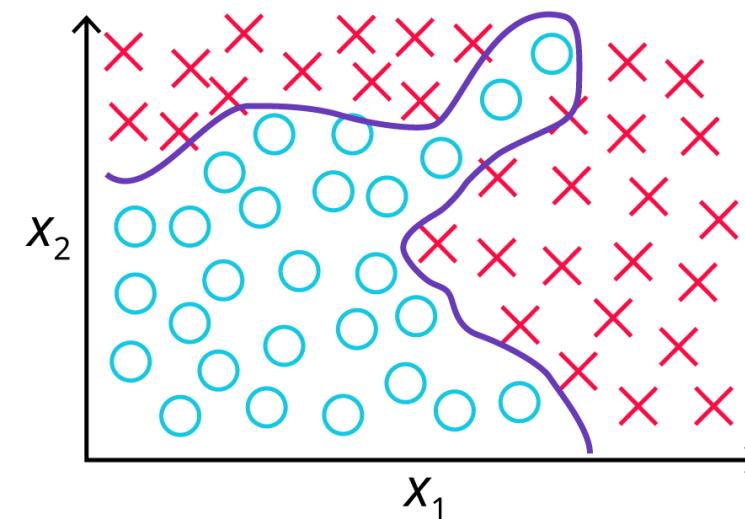
$$\hat{y} = g(w_0 + X^T W)$$

**Activation function**

where  $X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$   $W = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

# Activation function

- It is attached to each neuron, determines whether it should be activated (or fired) or not.
- Activation functions are better to be non-linear. Why?
  - Because in real-life applications, the data are usually not separable with linear boundaries (or hyper-planes).
  - Activation function's key function is to **introduce non-linearities** into the network.



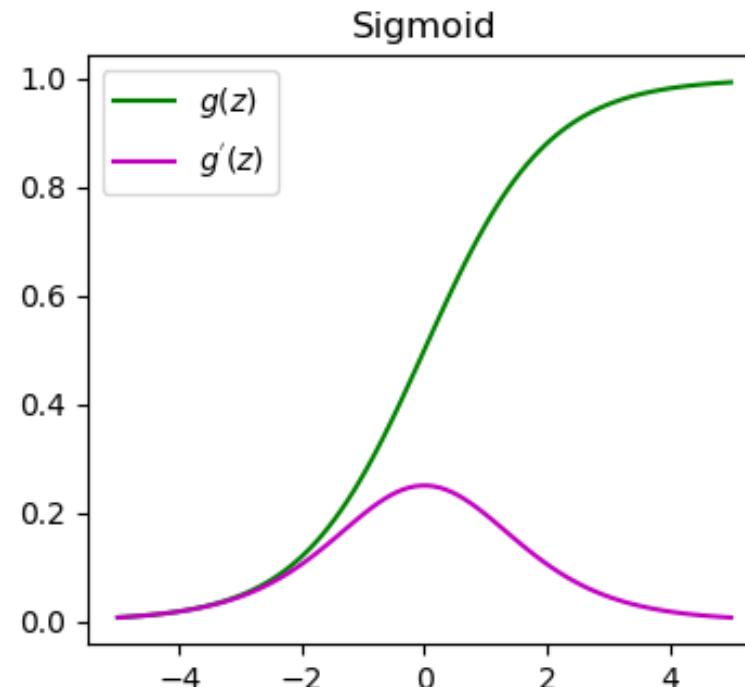
# Commonly Used Activation Functions

# Sigmoid/Logistic function

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

- + Smooth gradient
- + Output bound

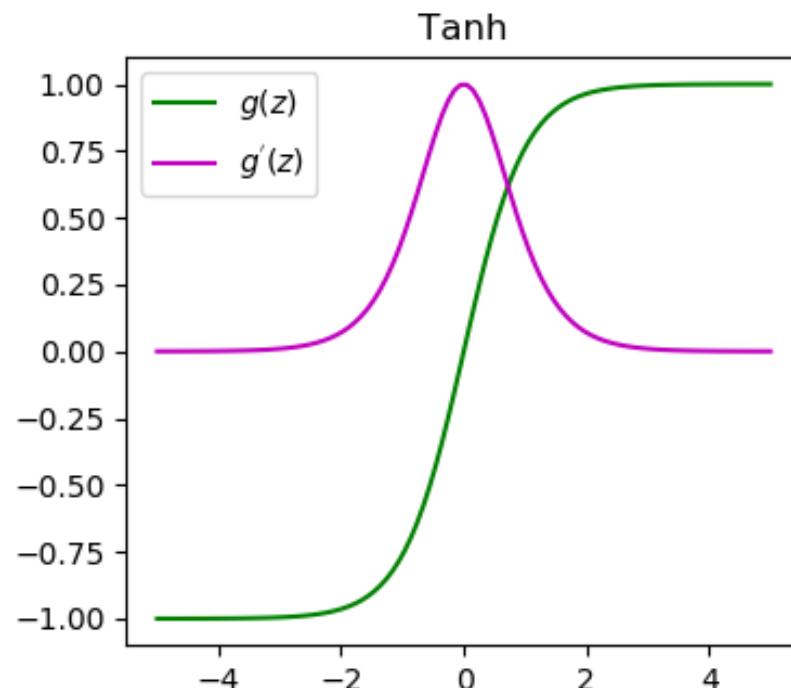


- Outputs not zero-centered
- Computationally expensive
- Vanishing gradient problem

# Hyperbolic Tangent (Tanh) function

$$g(z) = \frac{1 - e^{-z}}{1 + e^{-z}}$$

$$g'(z) = 1 - g^2(z)$$

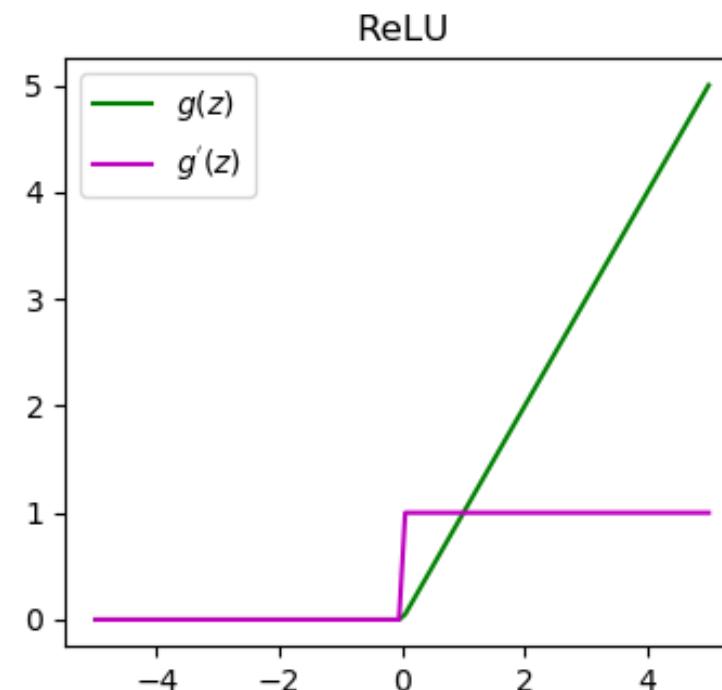


- + Smooth gradient
- + Output bound
- + Zero-centered
- Computationally expensive
- Vanishing gradient problem

# ReLU (Rectified Linear Unit) function

$$g(z) = \max(0, z)$$

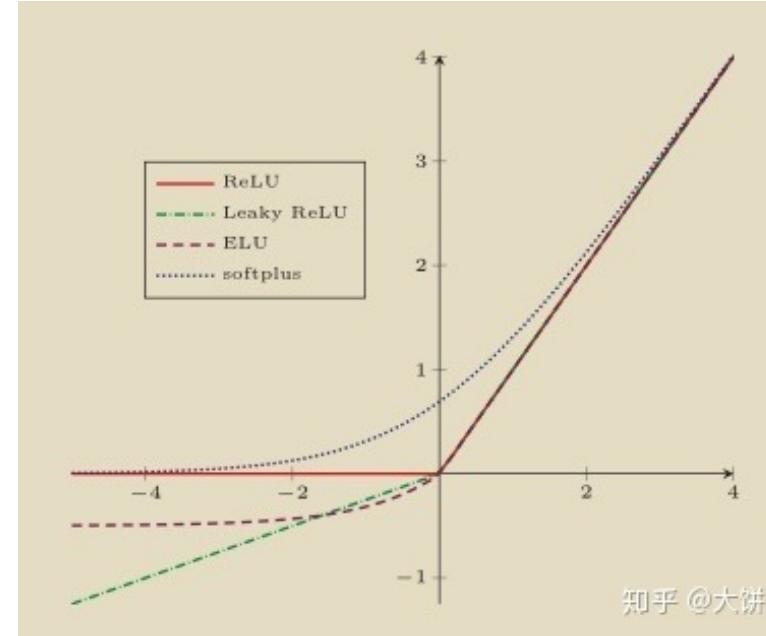
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$$



- + Allows neuron to express a *strong opinion*
- + Computationally efficient
- + Cause sparsity and efficiency
- Output not bound
- The dying problem
- Gradient discontinuous

# More ...

- ReLU variants:



- Softmax – for the output layer for multi-class classification

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

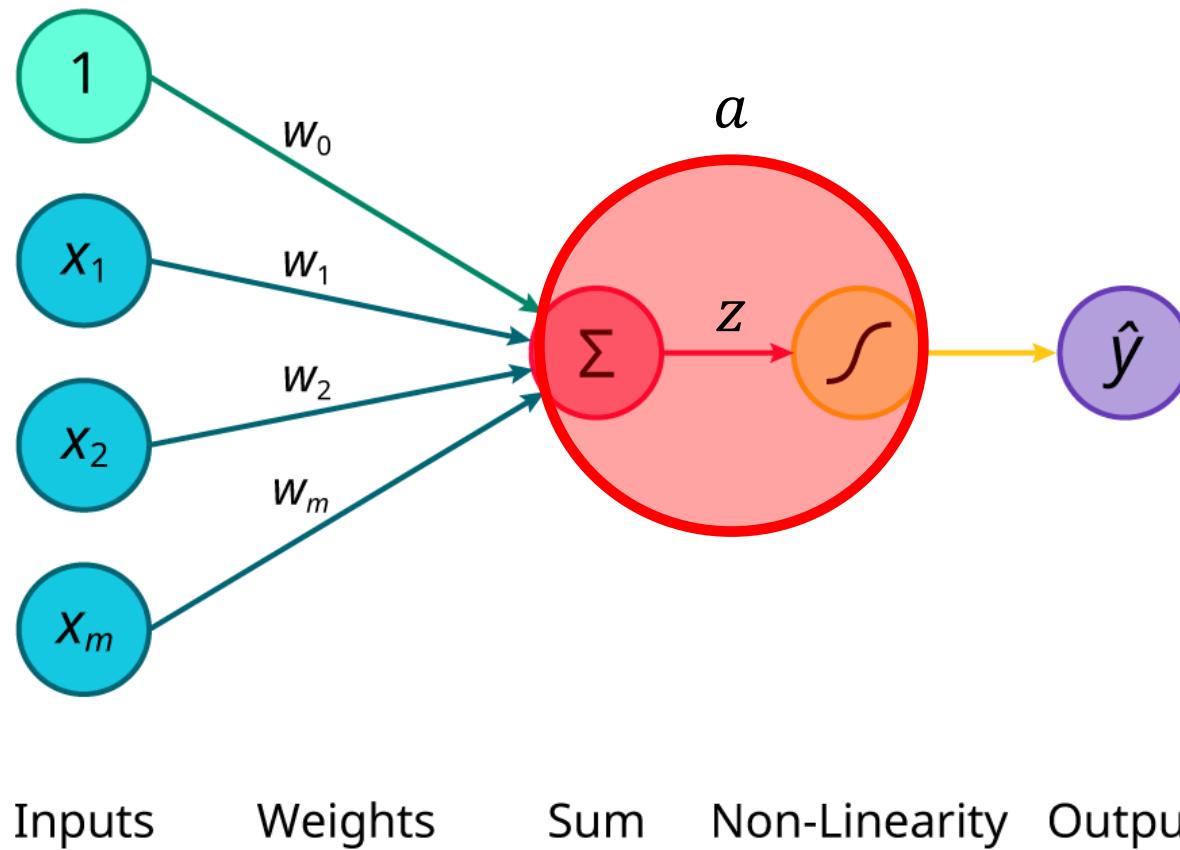
- ...

# Reference

- Video lecture by Alexander Amini: MIT course on deep learning,  
<https://www.youtube.com/watch?v=njKP3FqW3Sk>
- <https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/>
- <https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/>
- <https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253>
- <https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/>

# Topic 3: Multi Layer Perceptron (MLP)

# Perceptron



$$a = g(z)$$

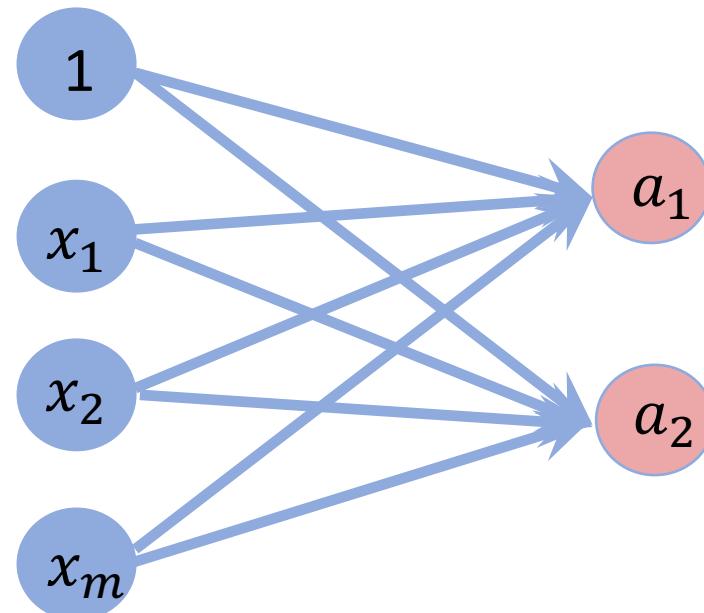
$$z = w_0 + \sum_{j=1}^m x_j w_j$$

**How about adding more perceptrons?**

$$a_1, a_2, \dots, a_i, \dots$$

$$z_1, z_2, \dots, z_i, \dots$$

# More Perceptrons



$$a_i = g(z_i)$$

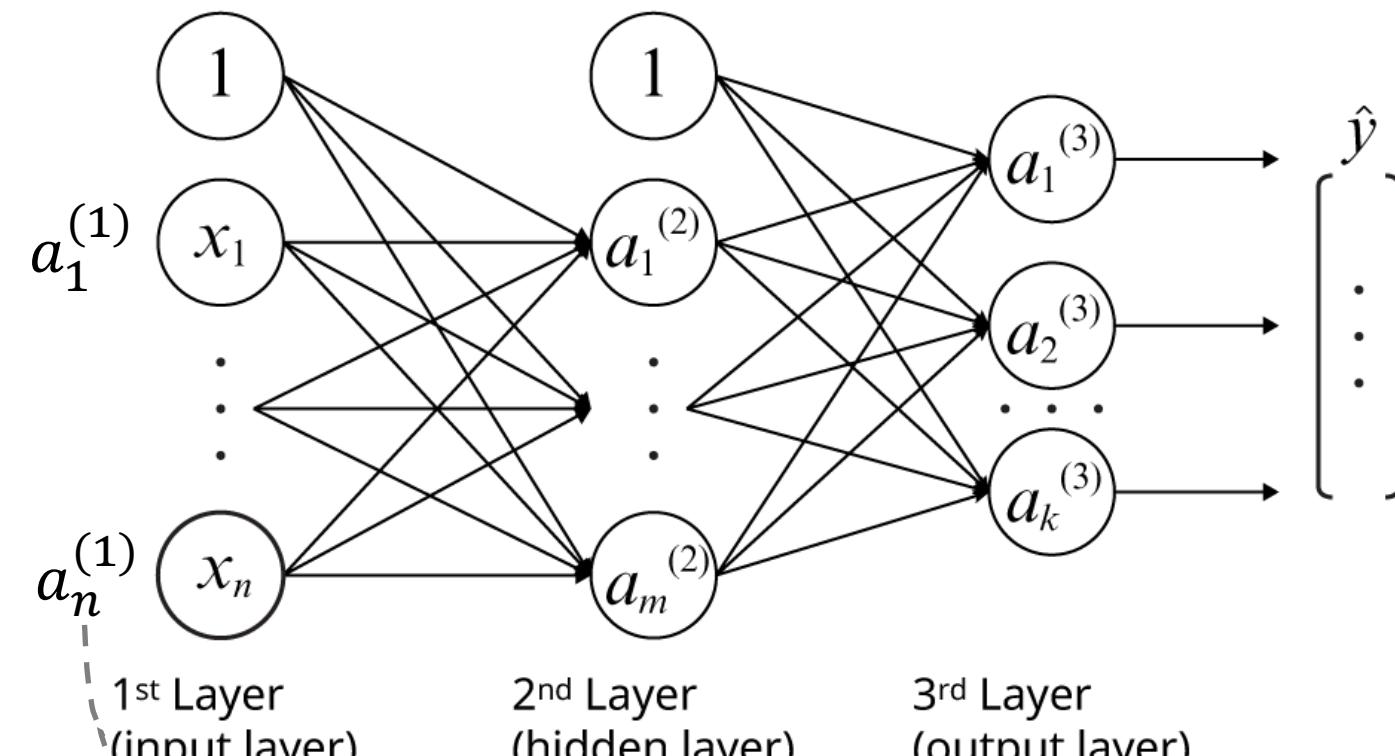
$$z_i = w_{0i} + \sum_{j=1}^m x_j w_{ji}$$

**How about adding more layers?**

$$a_1^{(l)}, a_2^{(l)}, \dots, a_i^{(l)} \dots$$

$$z_1^{(l)}, z_2^{(l)}, \dots, z_i^{(l)} \dots$$

# Multi Layer Perceptron (MLP)



$$a_i^{(l)} = g(z_i^{(l)})$$

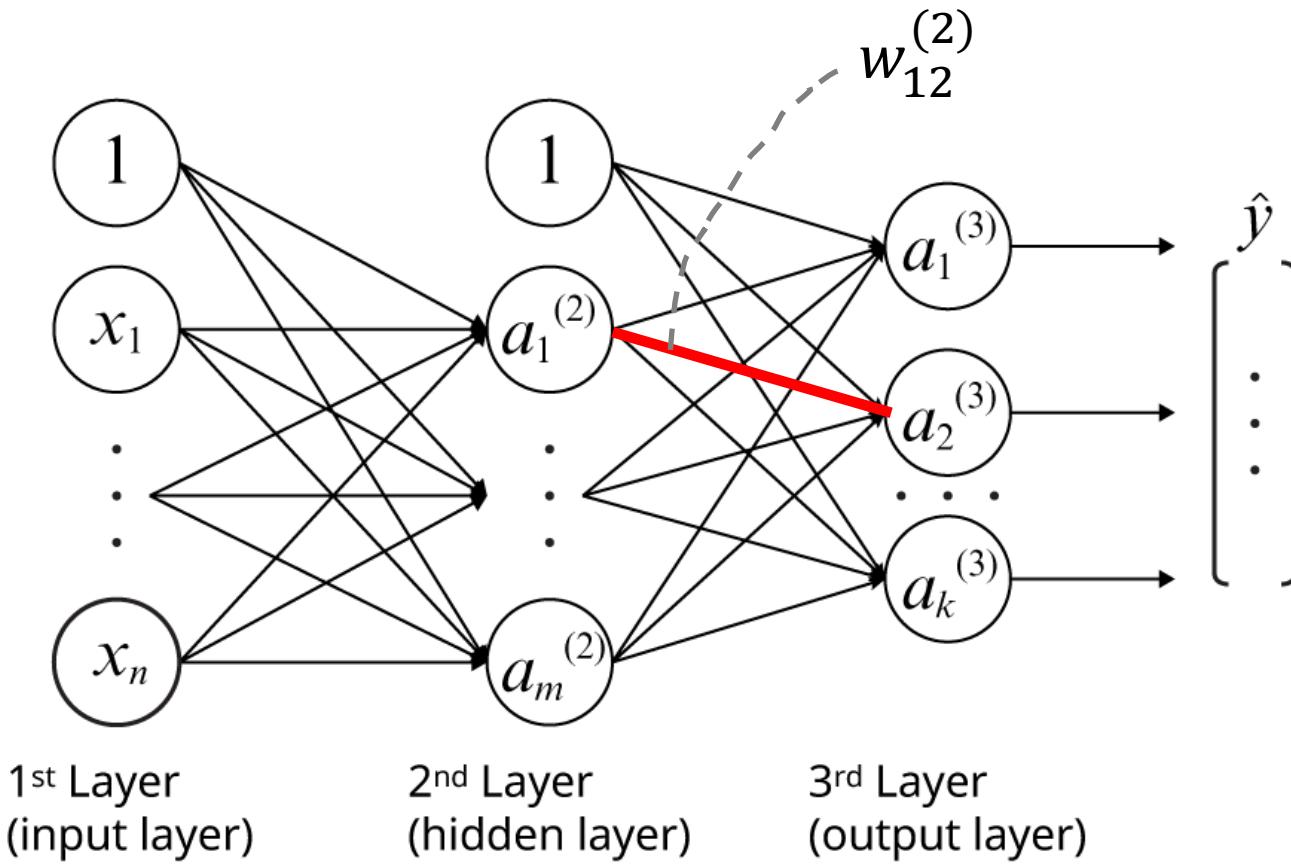
$$z_i^{(l)} = w_{0i}^{(l-1)} + \sum_{j=1}^m a_j^{(l-1)} w_{ji}^{(l-1)}$$

Where  $l = 2, 3, \dots$

The weight from the  $j$ -th neuron in the  $l-1$ -th layer to the  $i$ -th neuron in the  $l$ -th layer.

The input layer can be also written in the form of  $a_i^{(1)}$

# Multi Layer Perceptron (MLP)

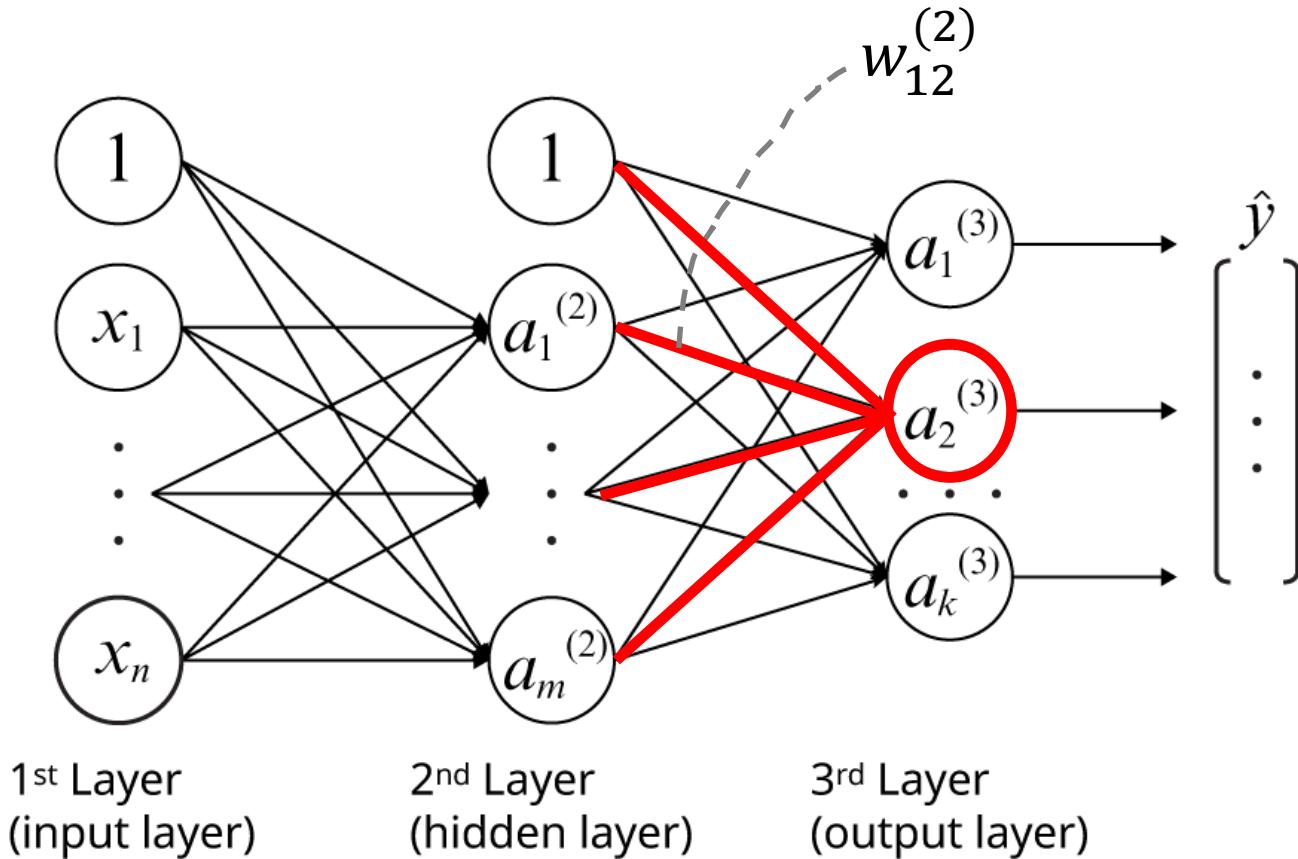


$$a_i^{(l)} = g(z_i^{(l)})$$

$$z_i^{(l)} = w_{0i}^{(l-1)} + \sum_{j=1}^m a_j^{(l-1)} w_{ji}^{(l-1)}$$

The weight from the  $j$ -th neuron in the  $l-1$ -th layer to the  $i$ -th neuron in the  $l$ -th layer.

# Multi Layer Perceptron (MLP)



$$a_i^{(l)} = g(z_i^{(l)})$$

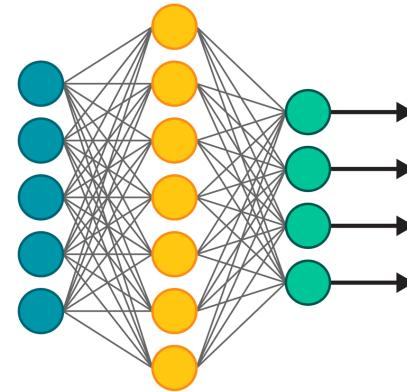
$$z_i^{(l)} = w_{0i}^{(l-1)} + \sum_{j=1}^m a_j^{(l-1)} w_{ji}^{(l-1)}$$

Let's see how to calculate  $a_2^{(3)}$

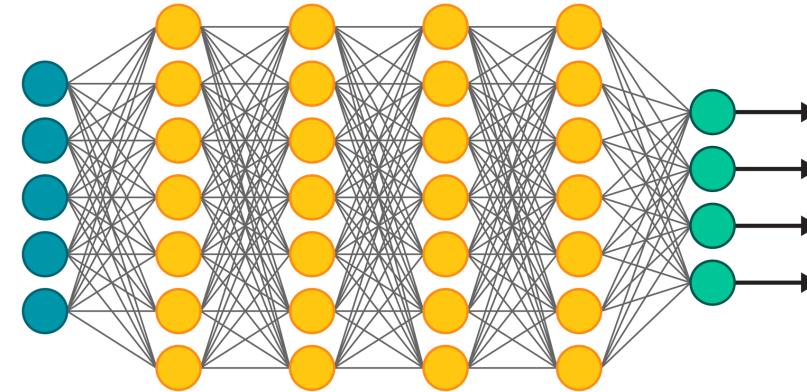
$$a_2^{(3)} = g(w_{02}^{(2)} + a_1^{(2)} w_{12}^{(2)} + a_2^{(2)} w_{22}^{(2)} + \dots + a_m^{(2)} w_{m2}^{(2)})$$

# Summary for MLP

Simple Neural Network



Deep Neural Network



● Input Layer

● Hidden Layer

● Output Layer

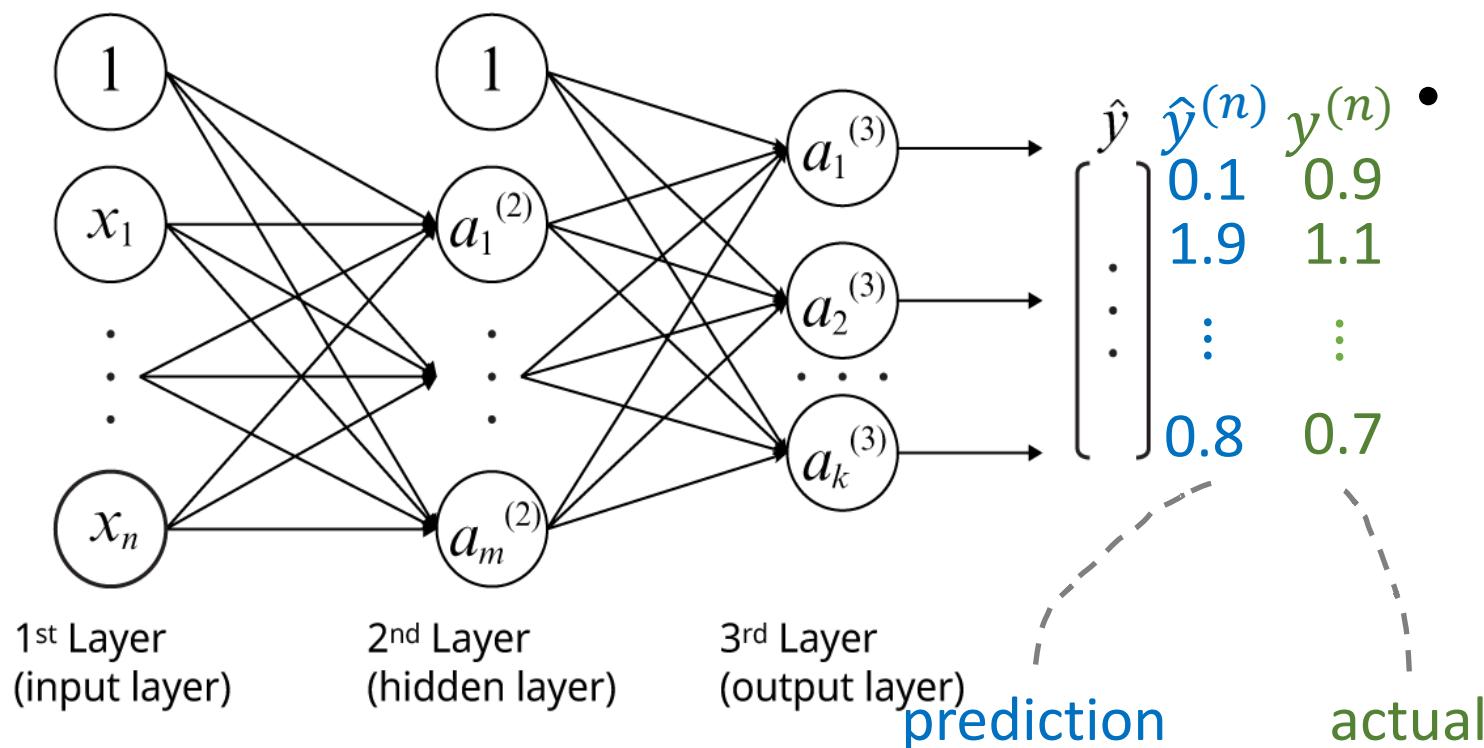
- Stacking even more layers to create a **deep neural network**.
- Multiple hidden layers are able to learn complex data with high levels of accuracy.
- MLP network consists of **fully connected layers** (or **dense layers**) - each neuron is connected to every other neuron in the adjacent layer

# Reference

- Video lecture by Alexander Amini: MIT course on deep learning,  
<https://www.youtube.com/watch?v=njKP3FqW3Sk>
- Book: Aurelien Geron. Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly. 2019.

# Topic 4: Loss Function

# Loss function

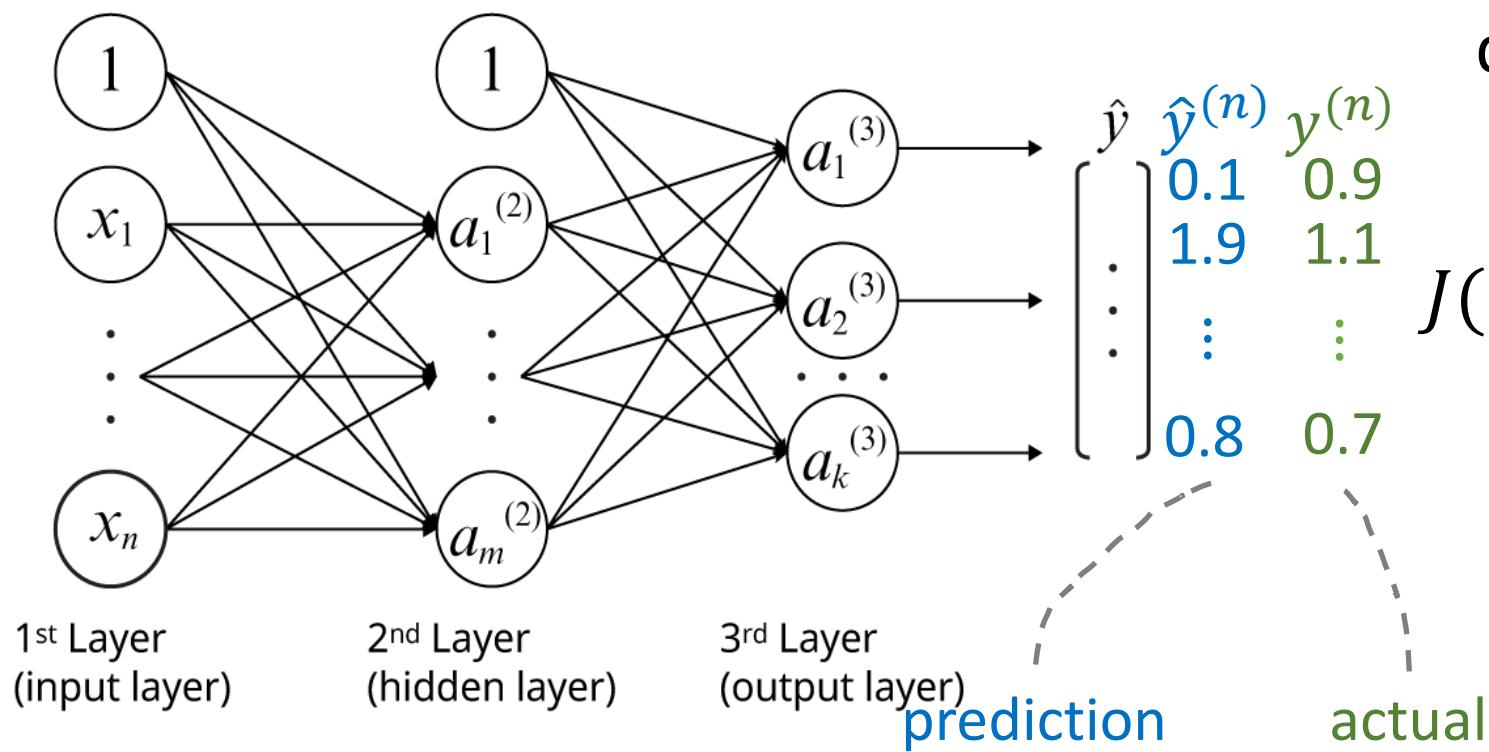


- We need to measure how good/bad the network prediction compared to the actual value(s).
- For a single sample  $(x^{(n)}, y^{(n)})$ , the prediction error is:

$$L(\hat{y}^{(n)}, y^{(n)})$$

$$= L(f(x^{(n)}; W), y^{(n)})$$

# Loss function



- The **empirical loss** measures the average loss over the entire dataset:

$$J(W) = \frac{1}{N} \sum_{n=1}^N L(f(x^{(n)}; W), y^{(n)})$$



# Binary Cross Entropy Loss

- **Binary cross-entropy loss** is used for **binary classification**, i.e., only two classes ( $y^{(n)}$  is either 0 or 1).

$$J(W) = -\frac{1}{N} \sum_{n=1}^N (y^{(n)} \log f(x^{(n)}; W) + (1 - y^{(n)}) \log(1 - f(x^{(n)}; W)))$$

actual                          prediction  
 $\{0, 1\}$                            $[0,1]$



# Multi-class Cross Entropy Loss

- **Multi-class cross-entropy loss** is used for **multi-class classification**, i.e., each example belong to ONE of  $C$  classes,  $C>2$

The network have  $C$  outputs, gathered into a vector

$$f(x^{(n)}; W) = \begin{bmatrix} 0.02 \\ \vdots \\ 0.81 \\ \vdots \\ 0.09 \end{bmatrix}$$

A **one-hot** vector: only the  $c$ -th element is 1, means this example belongs to class  $c$ .

$$J(W) = -\frac{1}{N} \sum_{n=1}^N \sum_{c=1}^C y_c^{(n)} \log f_c(x^{(n)}; W)$$

actual {0, 1}      prediction [0,1]

# Mean Squared Error (MSE) Loss

- **Mean Squared Error (MSE)** loss is the default loss to use for **regression problems** which output continuous real numbers.

$$J(W) = \frac{1}{N} \sum_{n=1}^N (y^{(n)} - f(x^{(n)}; W))^2$$

actual    prediction

# Reference

- Video lecture by Alexander Amini: MIT course on deep learning,  
<https://www.youtube.com/watch?v=njKP3FqW3Sk>
- <https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/>

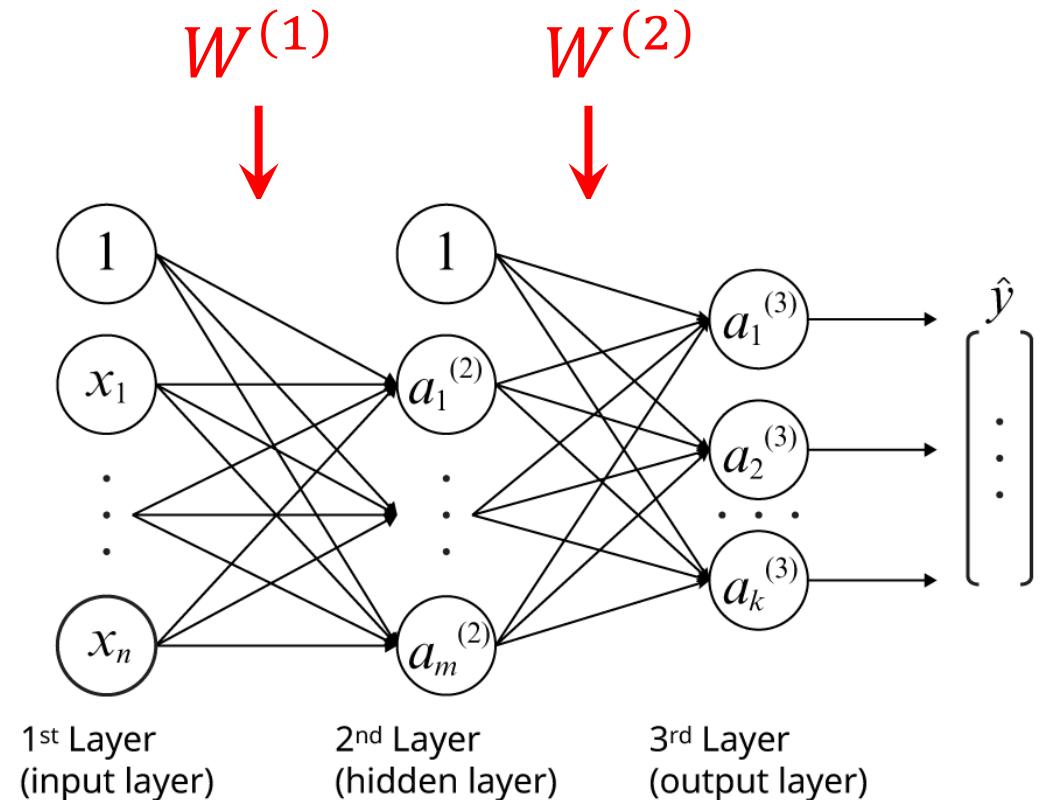
# Topic 5: Training Neural Networks

# Optimization

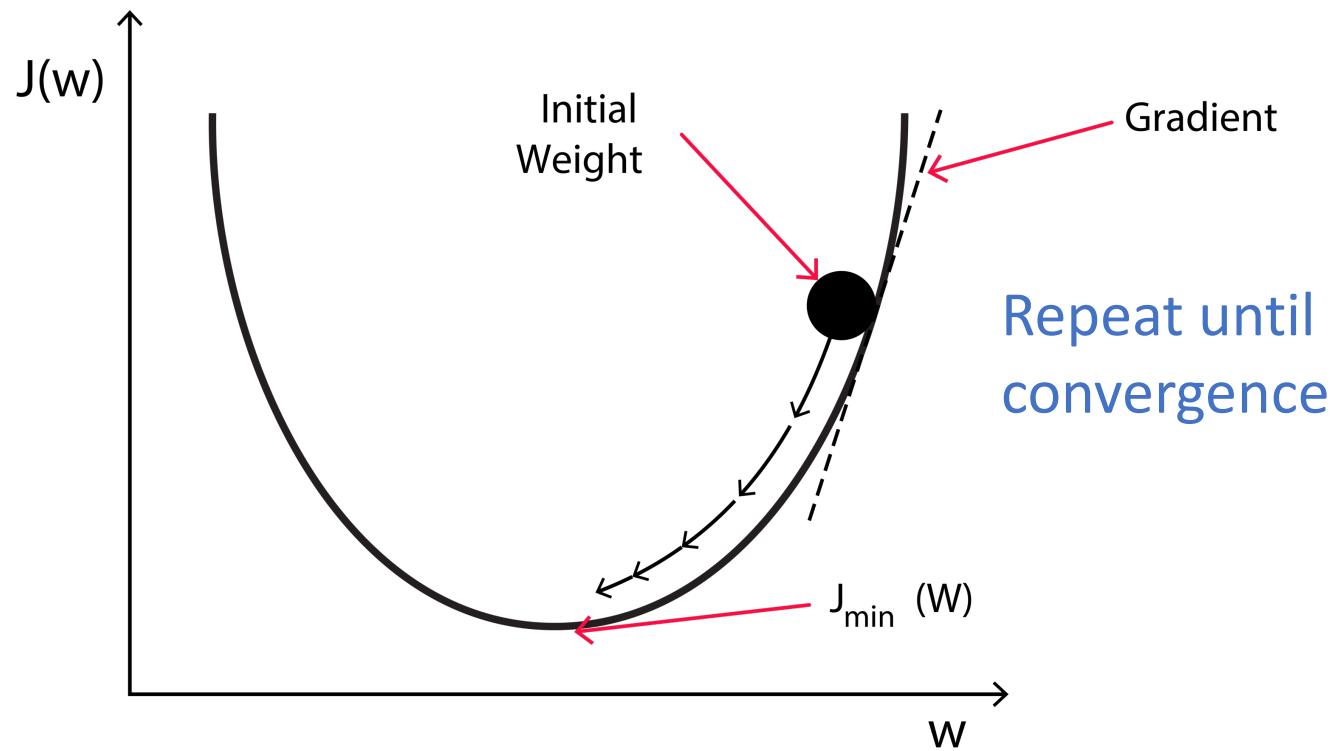
- *Objective:* find the weights  $W^*$  produce the lowest loss:

$$\begin{aligned} W^* &= \underset{W}{\operatorname{argmin}} J(W) \\ &= \underset{W}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^N L(f(x^{(n)}; W), y^{(n)}) \end{aligned}$$

Where  $W = \{W^{(1)}, W^{(2)}, \dots, W^{(L-1)}\}$



# Gradient Descent



Randomly pick one initial weight  $W$ .

Calculate its gradient  $\frac{\partial J(W)}{\partial W}$ .

Take small steps in its opposite direction  $-\eta \frac{\partial J(W)}{\partial W}$ .

Update the weight  

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$

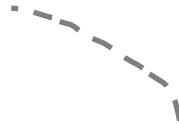
# Gradient Descent

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$

learning rate

For each weight:

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial J(W)}{\partial w_{ji}^{(l)}}$$



**Q:** How to compute the gradient with respect to each weight?

**A: Backpropagation**

# Chain Rule

$$[f(g(x))]' = f'(g(x)) \cdot g'(x)$$

OR:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

- Example:  $y = (5x + 1)^2$

$$y = u^2 \quad u = 5x + 1$$

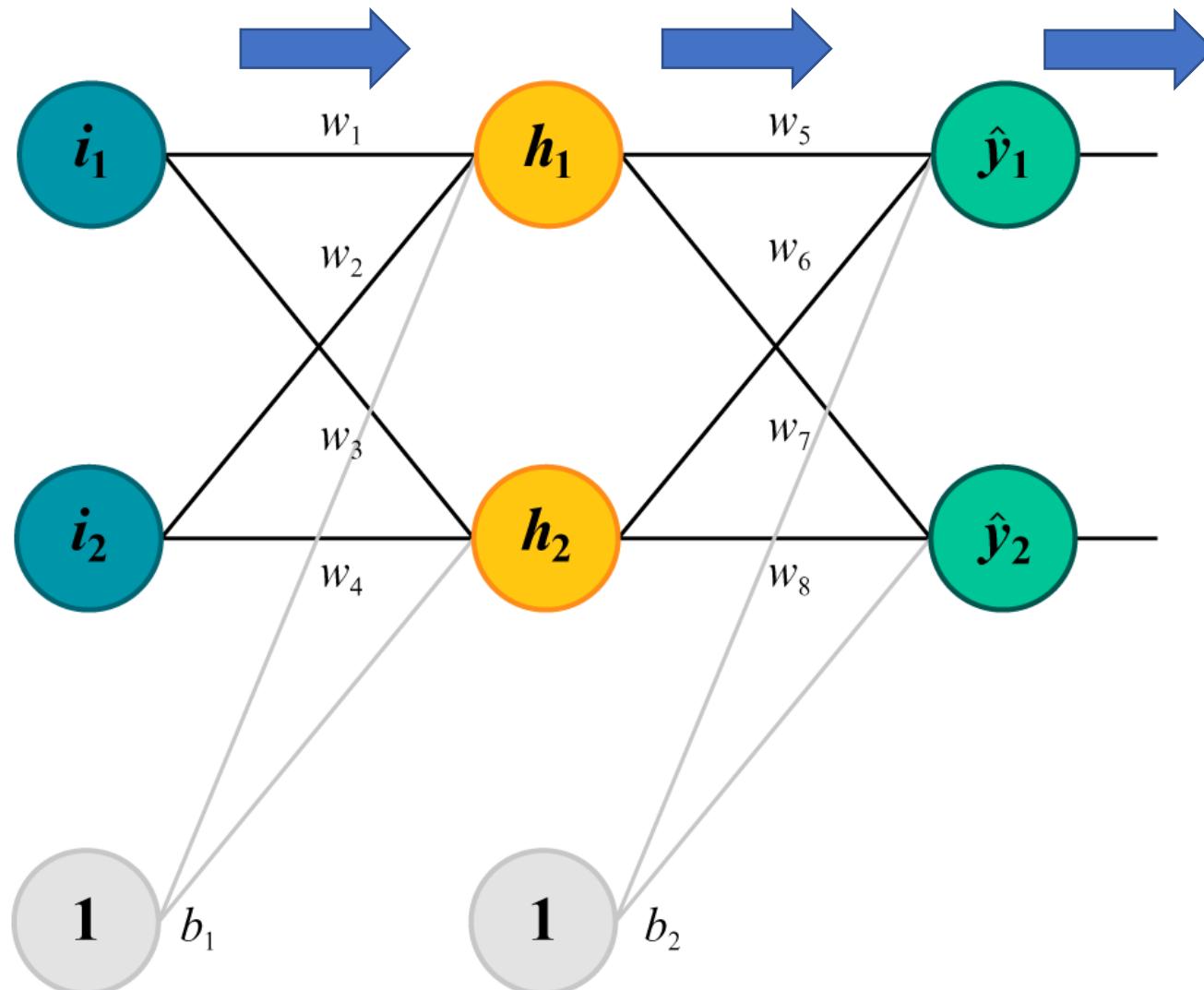
$$y' = \frac{dy}{du} \cdot \frac{du}{dx} = 2u \cdot 5 = 10(5x + 1)$$

# Reference

- Video lecture by Alexander Amini: MIT course on deep learning,  
<https://www.youtube.com/watch?v=njKP3FqW3Sk>
- Book: Aurelien Geron. Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly. 2019.

# Topic 6: Backpropagation

# Backpropagation



## Forward Pass

- The 2<sup>nd</sup>-layer:

$$h_1 = g(b_1 + i_1 \cdot w_1 + i_2 \cdot w_2)$$

$$h_2 = g(b_1 + i_1 \cdot w_3 + i_2 \cdot w_4)$$

- The 3<sup>rd</sup>-layer:

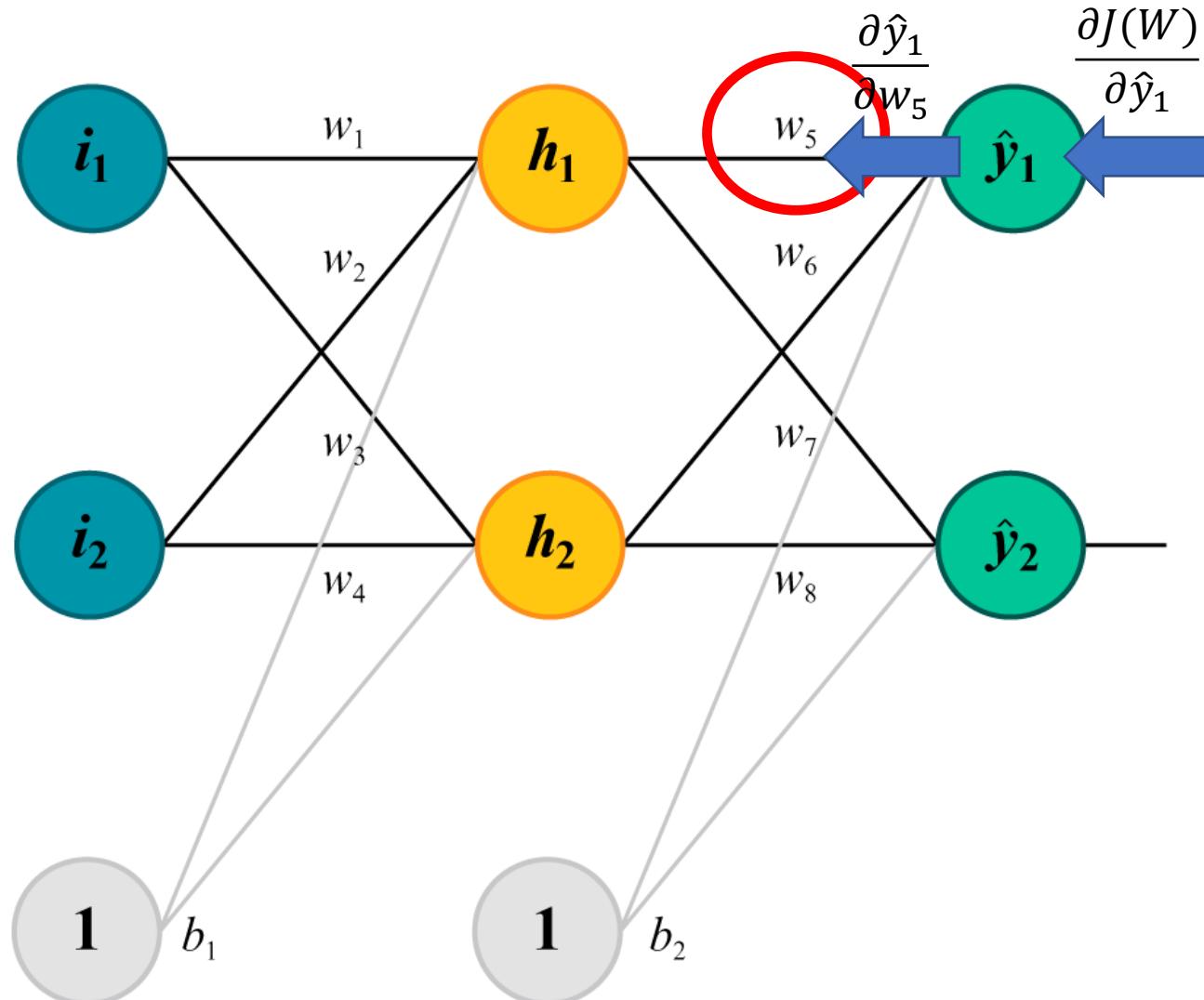
$$\hat{y}_1 = g(b_2 + h_1 \cdot w_5 + h_2 \cdot w_6)$$

$$\hat{y}_2 = g(b_2 + h_1 \cdot w_7 + h_2 \cdot w_8)$$

- The loss (MSE):

$$J(W) = \frac{1}{2} [(\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2]$$

# Backpropagation



## Backward Pass

$$\frac{\partial J(W)}{\partial w_5}$$

- Apply chain rule:

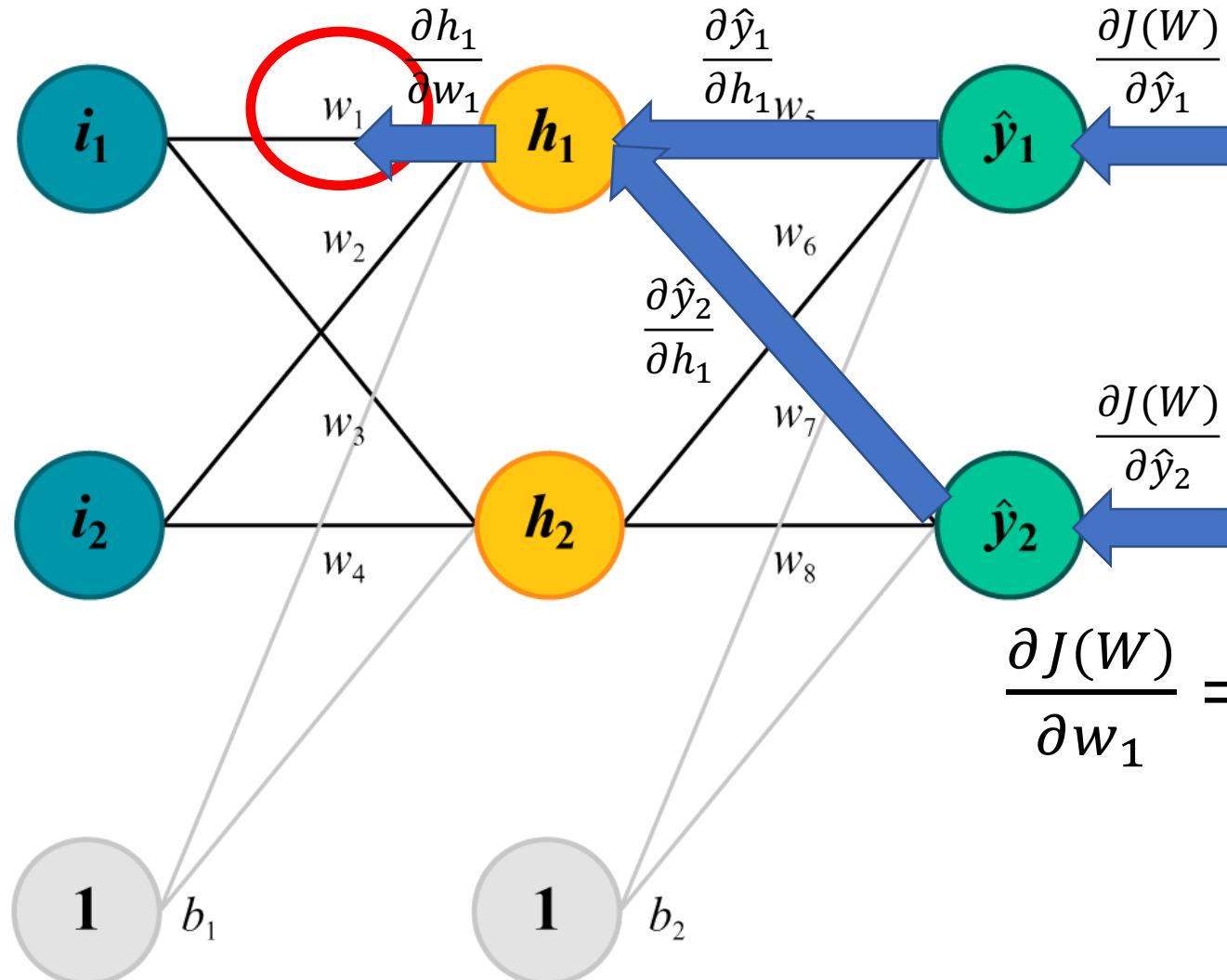
$$\frac{\partial J(W)}{\partial w_5} = \frac{\partial J(W)}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial w_5}$$

- Update the weight:

$$w_5 \leftarrow w_5 - \eta \frac{\partial J(W)}{\partial w_5}$$

- The same to  $w_6, w_7, w_8, b_2$

# Backpropagation

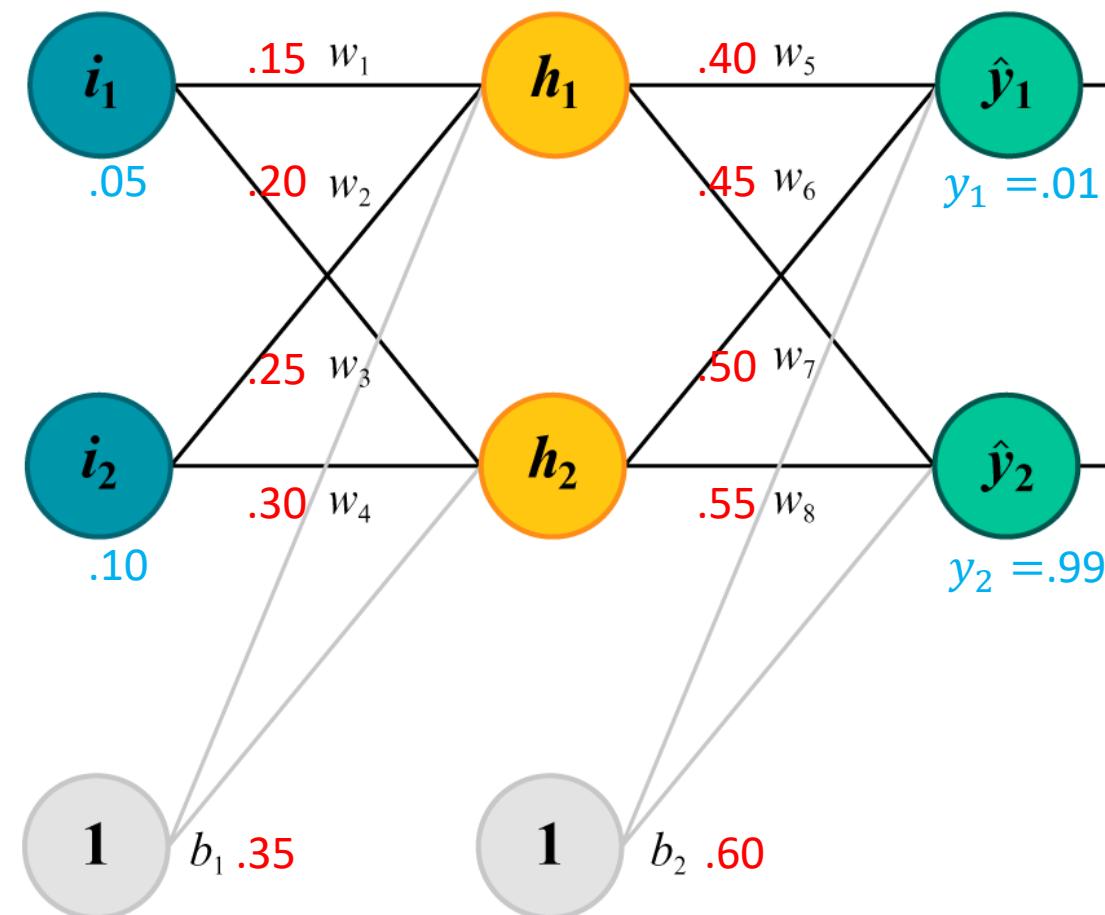


## Backward Pass

$$\frac{\partial J(W)}{\partial w_1}$$

- Apply chain rule:
- $$\frac{\partial J(W)}{\partial w_1} = \left( \frac{\partial J(W)}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial h_1} + \frac{\partial J(W)}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial h_1} \right) \cdot \frac{\partial h_1}{\partial w_1}$$
- Update the weight  $w_1, w_2, w_3, w_4, b_1$

**Exercise:** Write a simple Python code to do the backpropagation for the following example.



Given

- Input:  $x = \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} = \begin{pmatrix} 0.05 \\ 0.10 \end{pmatrix}$

- Actual output:  $y = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \end{pmatrix} = \begin{pmatrix} 0.01 \\ 0.99 \end{pmatrix}$

- Weights, biases:  $w_1, \dots, w_8, b_1, b_2 = \dots$

Suppose all activation functions are sigmoid function:

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

Calculate:

- All hidden neurons:  $h_1 = ?$ ,  $h_2 = ?$  Forward Pass
- All predictions:  $\hat{y}_1 = ?$ ,  $\hat{y}_2 = ?$
- The loss on this single example  $(x, y)$  using MSE loss:  $J(W) = ?$

- All partial derivatives:

$$\frac{\partial J(W)}{\partial w_1} = ?, \dots, \frac{\partial J(W)}{\partial w_8} = ?, \frac{\partial J(W)}{\partial b_1} = ?, \frac{\partial J(W)}{\partial b_2} = ?$$

- Update weights, biases (with  $\eta = 0.5$ ):

$$w_1^{\text{new}} = ?, \dots, w_8^{\text{new}} = ?, b_1^{\text{new}} = ?, b_2^{\text{new}} = ?$$

# Reference

- Blog by Matt Mazur: A Step by Step Backpropagation Example.  
<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>
- Blog by Michael Nielsen: Chapter 2 How the backpropagation algorithm works.  
<http://neuralnetworksanddeeplearning.com/chap2.html>

# Week 03: Convolutional Neural Networks (CNNs)

Machine Learning 2

Dr. Hongping Cai

# Topic 1: Challenges for Visual Recognition



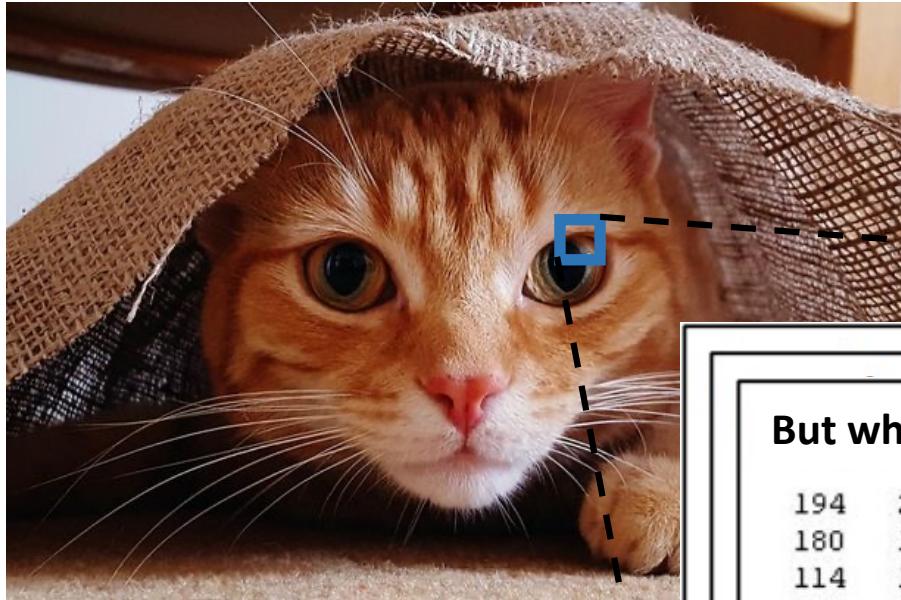
- Annual competition on:
  - Image Classification
  - Single-object localization
  - Object detection
- **ImageNet** dataset
  - 1.4 million annotated images
  - 1000 object classes

The best classification performance in 2011 (before deep learning):

**25.8% error**

<http://www.image-net.org/challenges/LSVRC/>

# Why is visual recognition so difficult?



An image is a matrix of numbers [0,255],  
i.e.,  $1024 \times 800 \times 3$  for a color image.

## But what the computer sees:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

# Why is visual recognition so difficult?

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



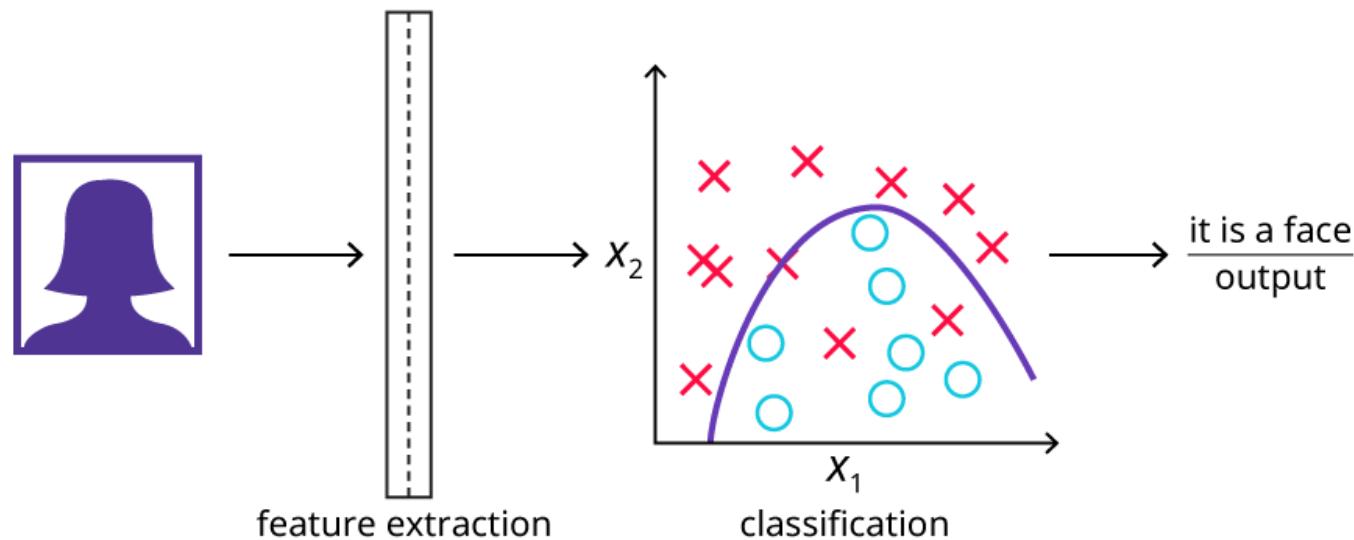
Background clutter



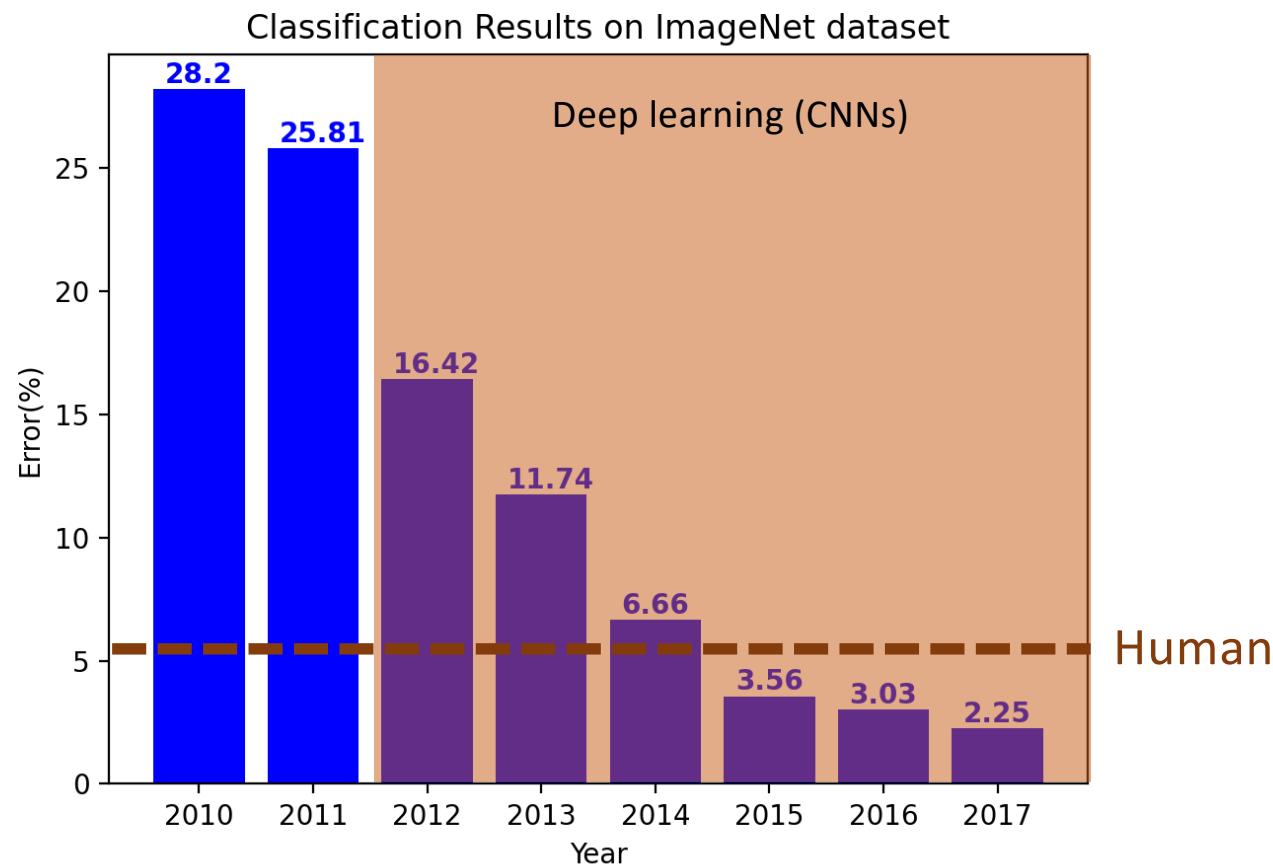
Intra-class variation



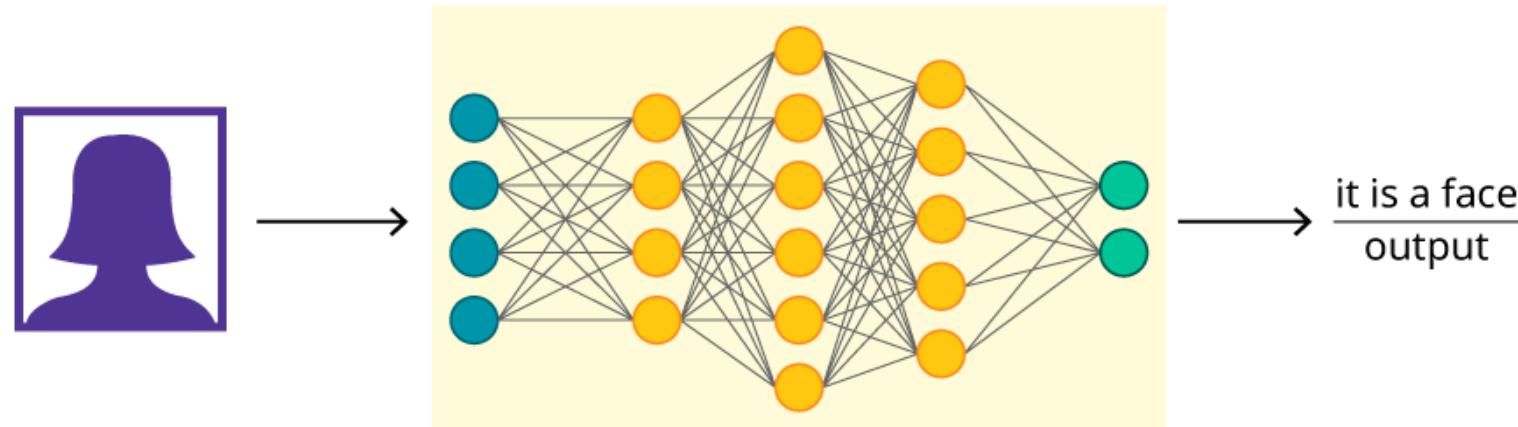
# Image classification before deep learning



# Things changed since 2012 ...



# Image classification with deep learning



# Reference for Topic 1

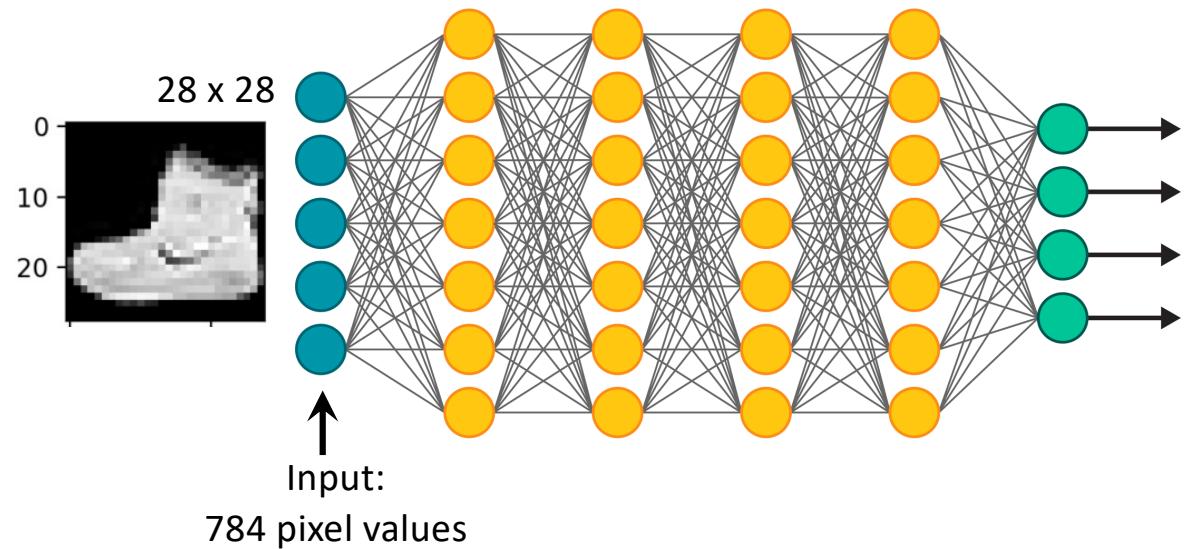
- Video lecture by Alexander Amini: MIT course on Convolutional Neural Networks: <https://www.youtube.com/watch?v=iaSUYvmCekI>
- <https://towardsdatascience.com/deep-learning-for-image-classification-why-its-challenging-where-we-ve-been-and-what-s-next-93b56948fce>

# Topic 2:

# Convolution Operation

# Can we use MLP to classify images?

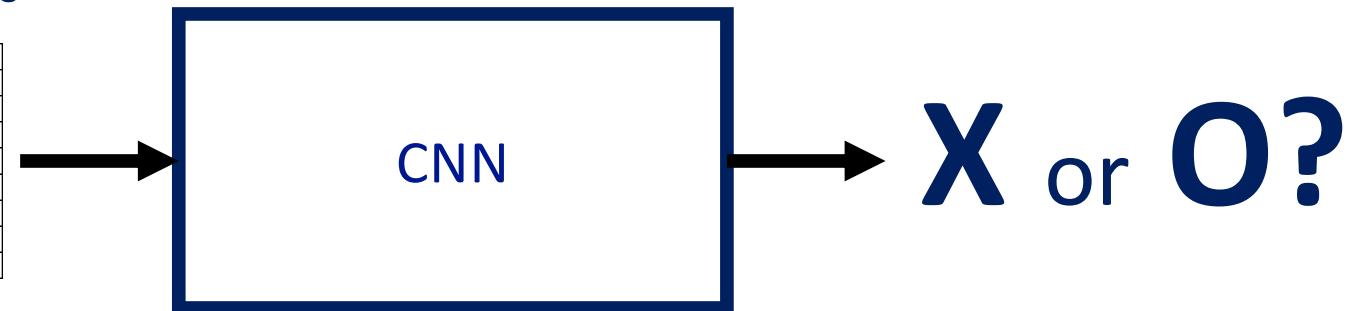
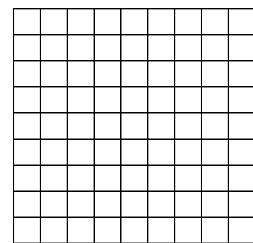
- Yes, we can.
- But,
  - Spatial information is removed.
  - Too many parameters for fully-connected layers



**Q:** If the input is a color image of size  $200 \times 200$ , the second layer is 300 neurons, how many weights needed for connecting the first two layers?  
**A:** 36,000,300 ( $=200 \times 200 \times 3 \times 300 + 300$ )!

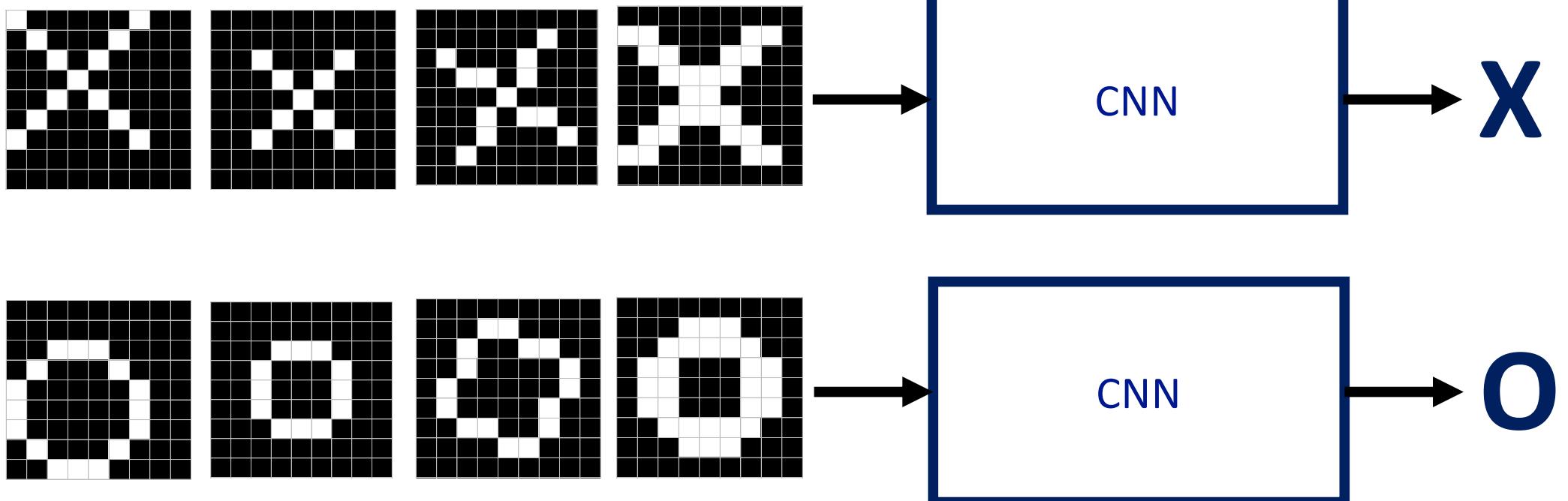
# A simple example: X's and O's

A two-dimensional  
array of pixels



Example from: <https://www.youtube.com/watch?v=FmpDlaiMleA>

A simple example: X's and O's

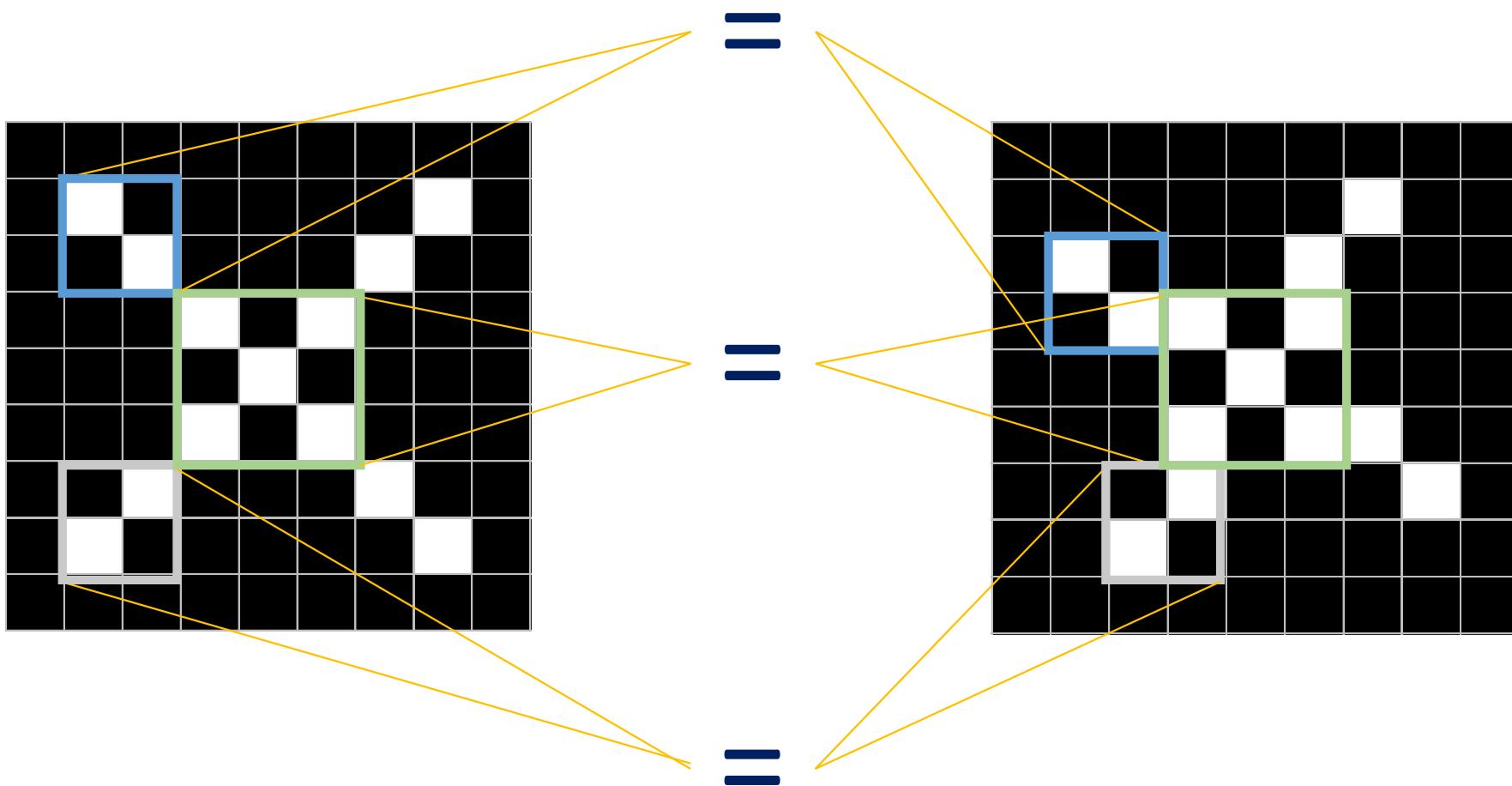




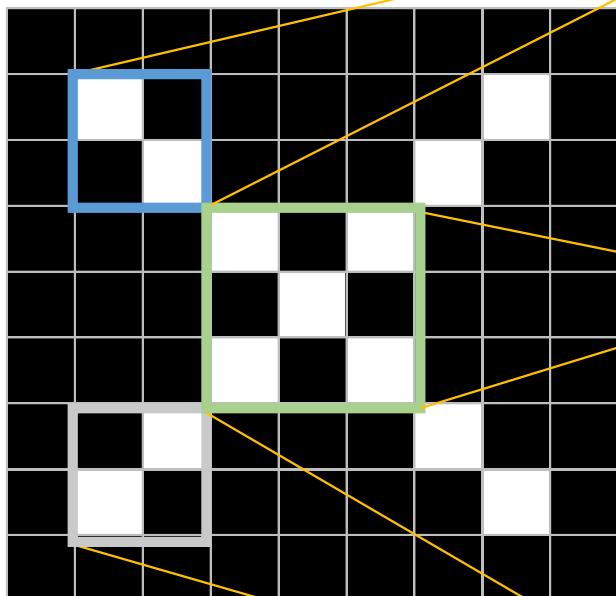
# A simple example: X's and O's



# Common local patterns



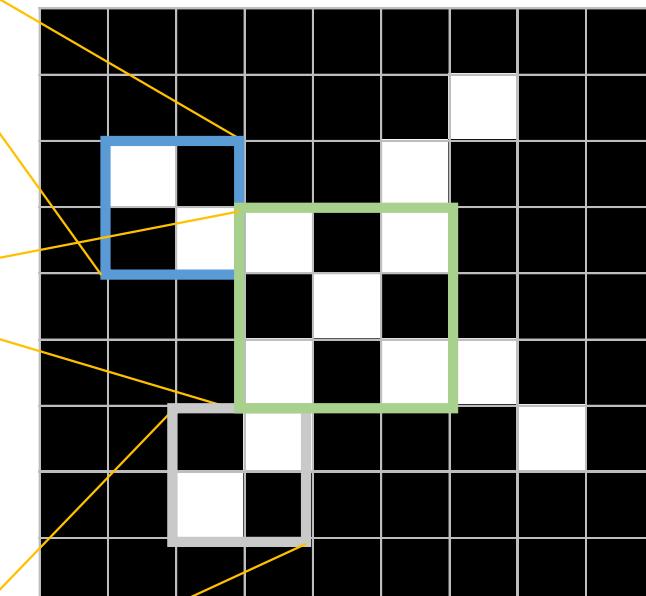
# Filters to detect X patterns



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

-1	-1	1
-1	1	-1
1	-1	-1



Filter/kernel

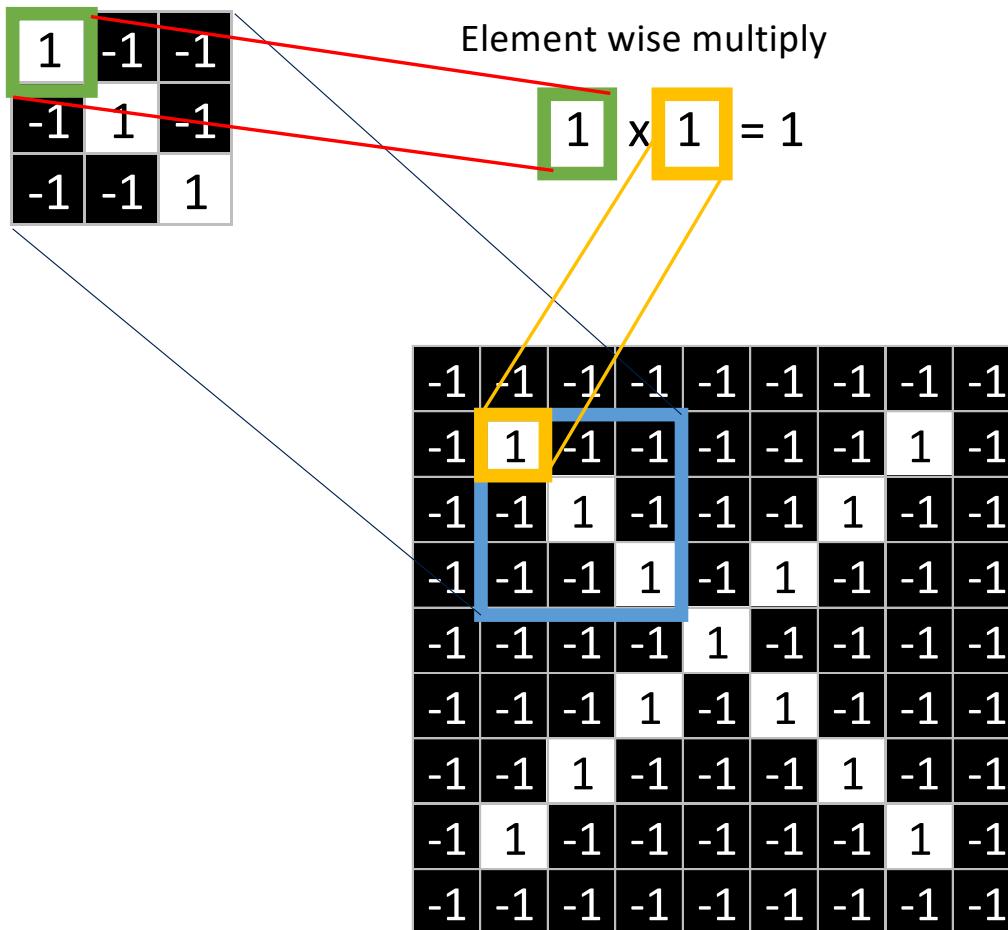
# Filtering: The math behind the match

h1	h2	h3
h4	h5	h6
h7	h8	h9

# ~~Filter~~/kernel

$$y = (x_1 * h_1 + x_2 * h_2 + \dots + x_9 * h_9) / 9$$

# Filtering: The math behind the match



# Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

Element wise multiply

$$-1 \times -1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	

# Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

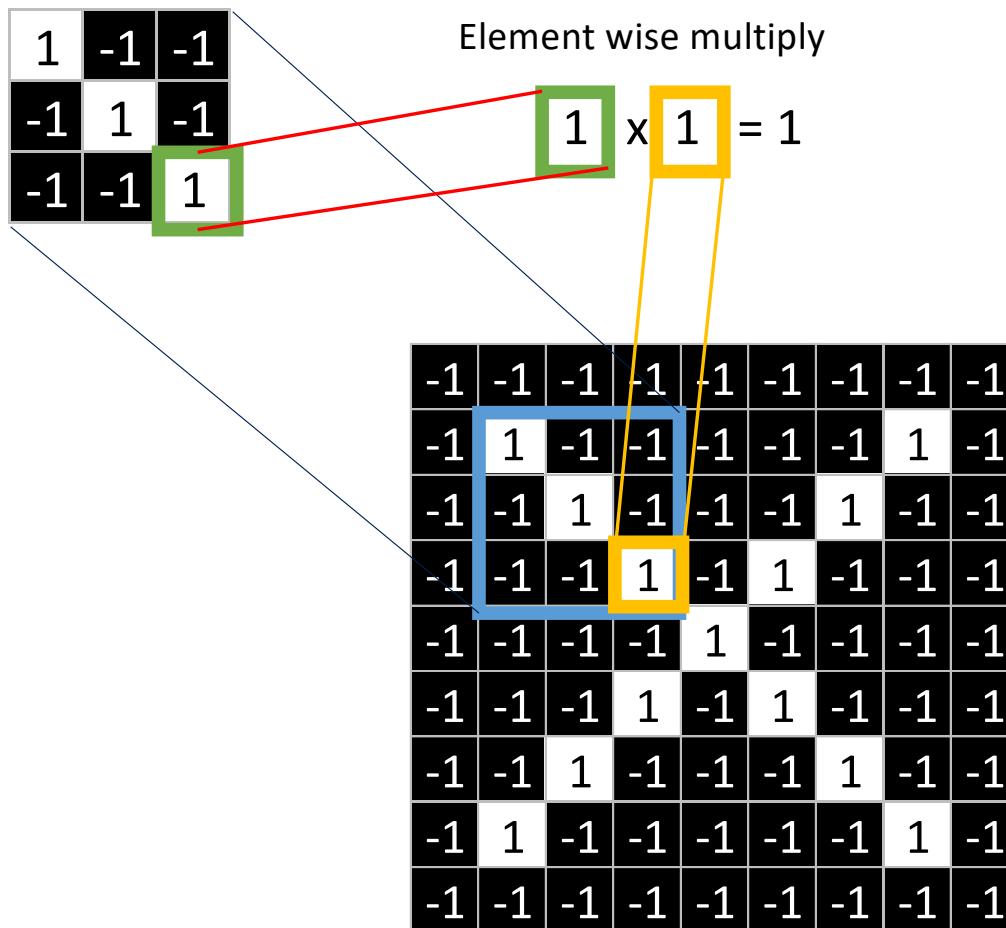
Element wise multiply

$$-1 \times -1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	1

# Filtering: The math behind the match



1	1	1
1	1	1
1	1	1

# Filtering: The math behind the match

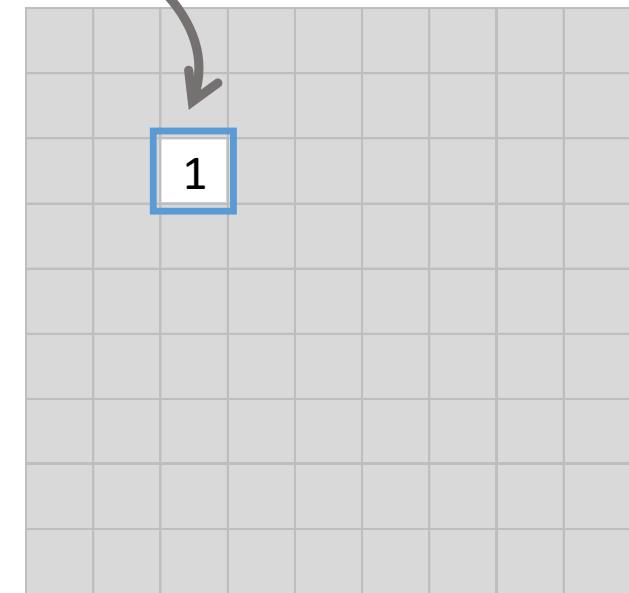
1	-1	-1
-1	1	-1
-1	-1	1

1	1	1
1	1	1
1	1	1

$$\frac{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1}{9} = 1$$

Add them up, divided by the number of filter pixels

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

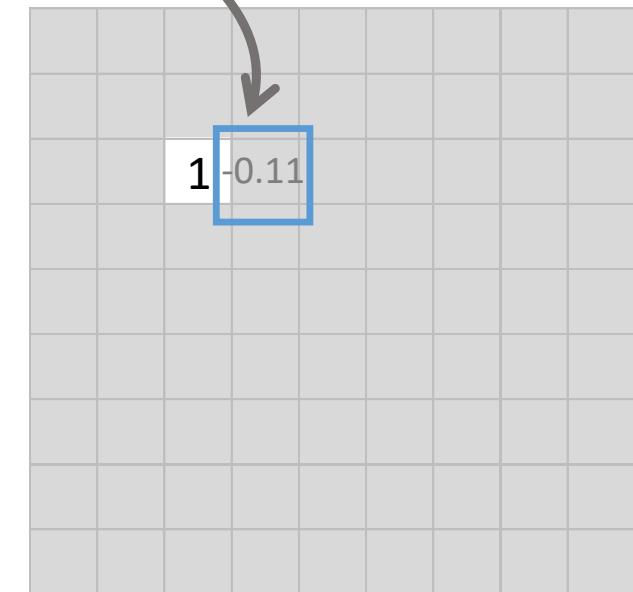


# Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

Slide the filter, repeat the process

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



# Convolution operation

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Input image



1	-1	-1
-1	1	-1
-1	-1	1

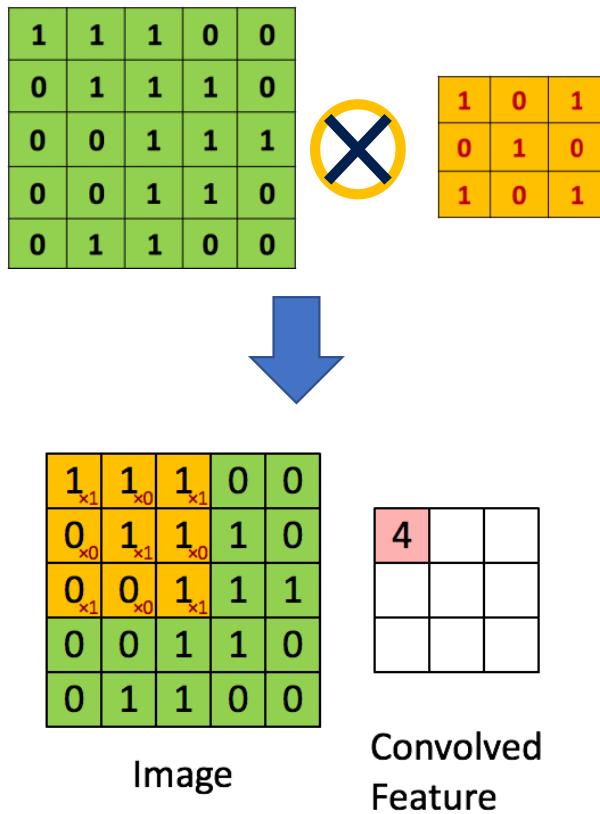
Filter/  
Kernel



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Feature map

# Convolution operation



From: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

# Reference for Topic 2

- Video lecture by Brandon Rohrer: How Convolutional Neural Networks work: <https://www.youtube.com/watch?v=FmpDlaiMleA>
- Video lecture by Alexander Amini: MIT course on Convolutional Neural Networks: <https://www.youtube.com/watch?v=iaSUYvmCekI>
- Blog by ujjwalkarn: An Intuitive Explanation of Convolutional Neural Networks: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

# Topic 3:

# CNN: Convolutional Layer

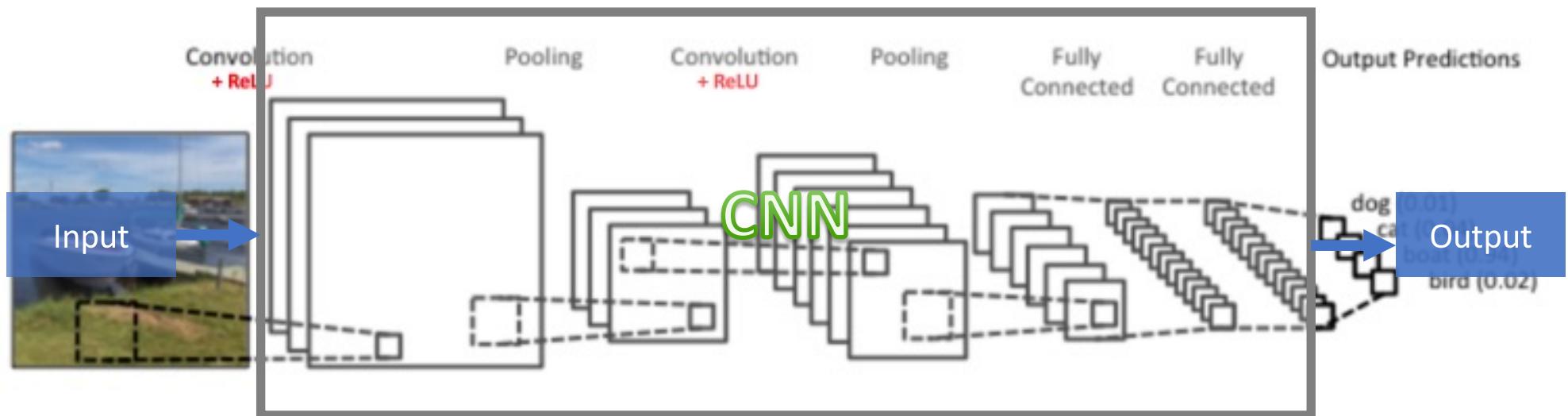
# Image classification with CNN



Convolutional Neural Network (**CNN**, or **ConvNet**)

Image from: <https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f>

# Image classification with CNN

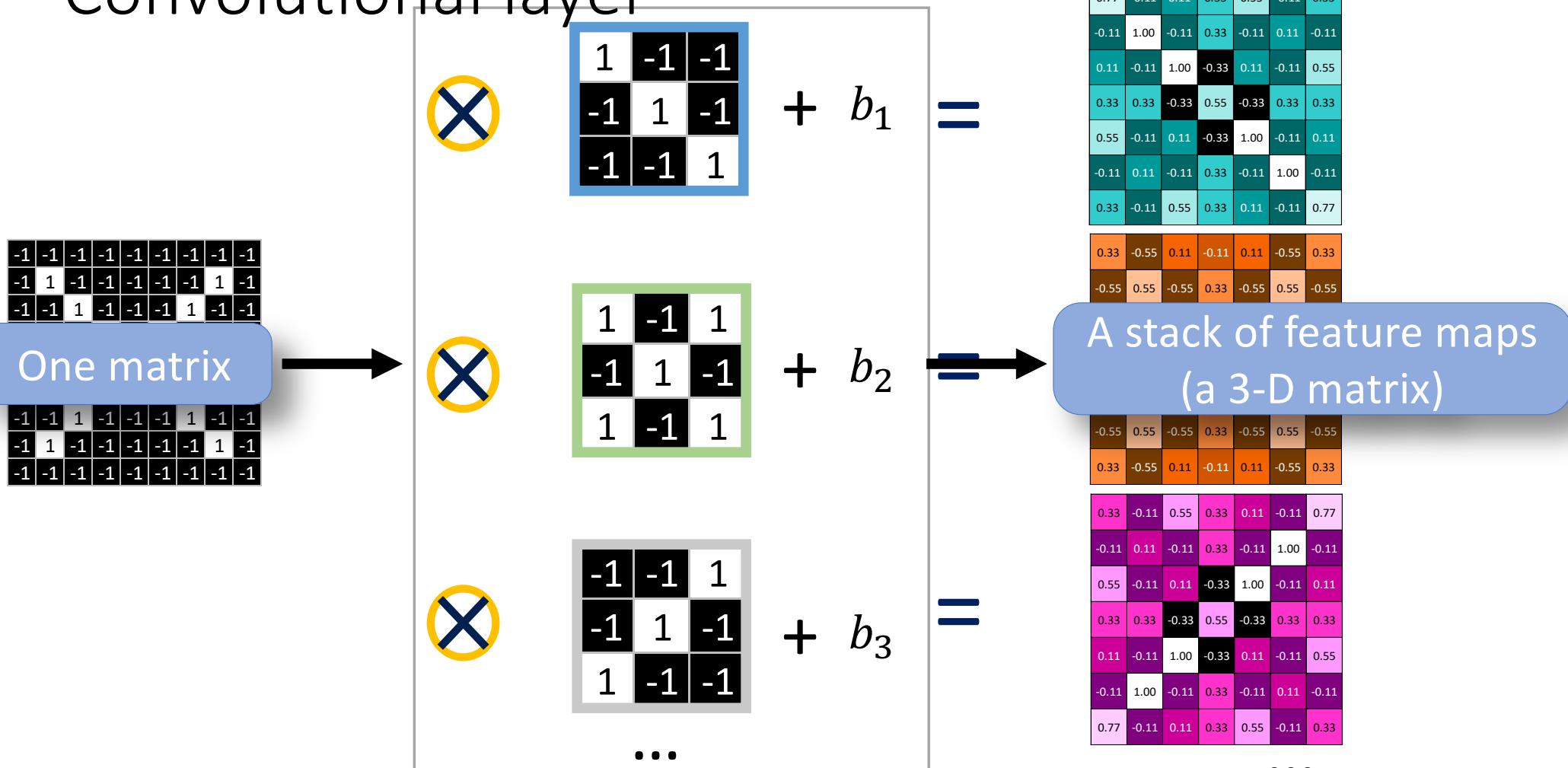


## → Convolutional Layer

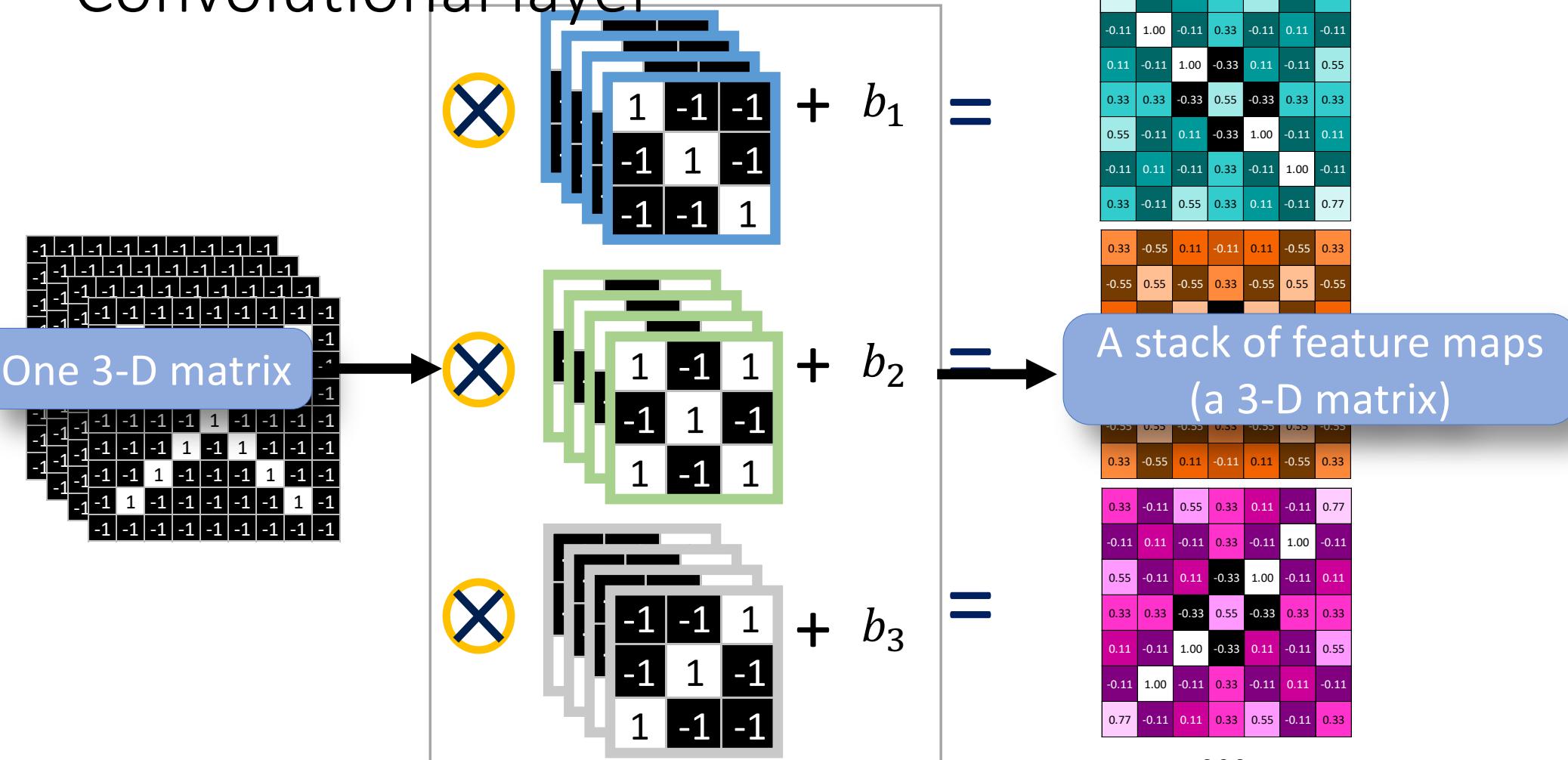
- ReLU Layer
- Pooling Layer
- Fully Connected Layer

Image from: <https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f>

# Convolutional layer

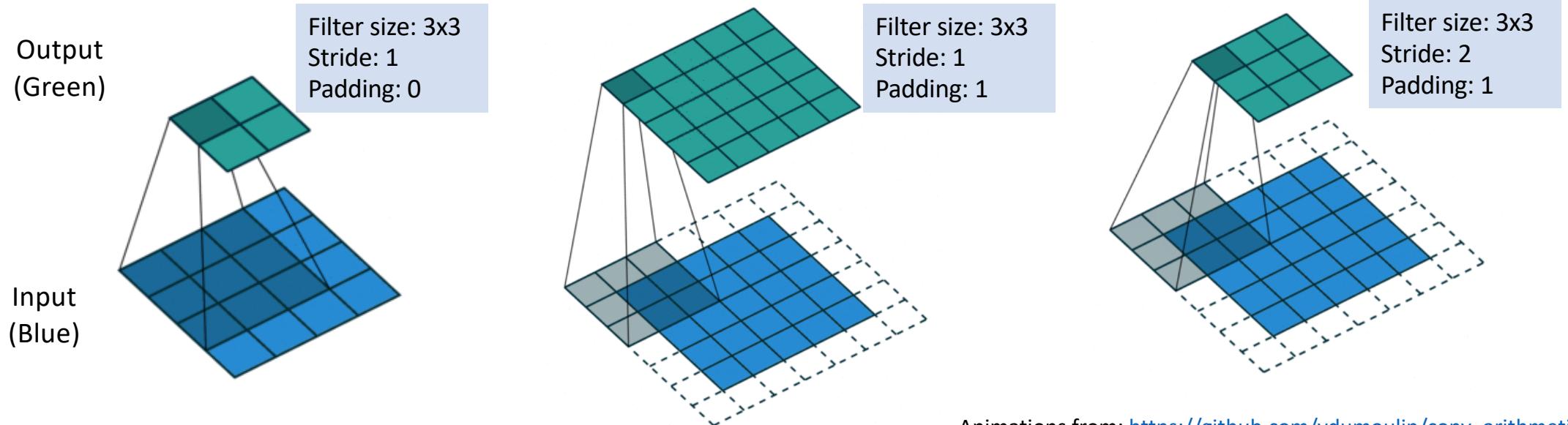


# Convolutional layer



# A few hyperparameters

- **Receptive field ( $F$ ):** the filter/kernel size
- **Stride ( $S$ ):** the number of pixels by which we slide the filters
- **Zero-padding ( $P$ ):** the number of zeros are padded at the border



Animations from: [https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)

# A few hyperparameters

- **Receptive field ( $F$ )**: the filter/kernel size
- **Stride ( $S$ )**: the number of pixels by which we slide the filters
- **Zero-padding ( $P$ )**: the number of zeros are padded at the border

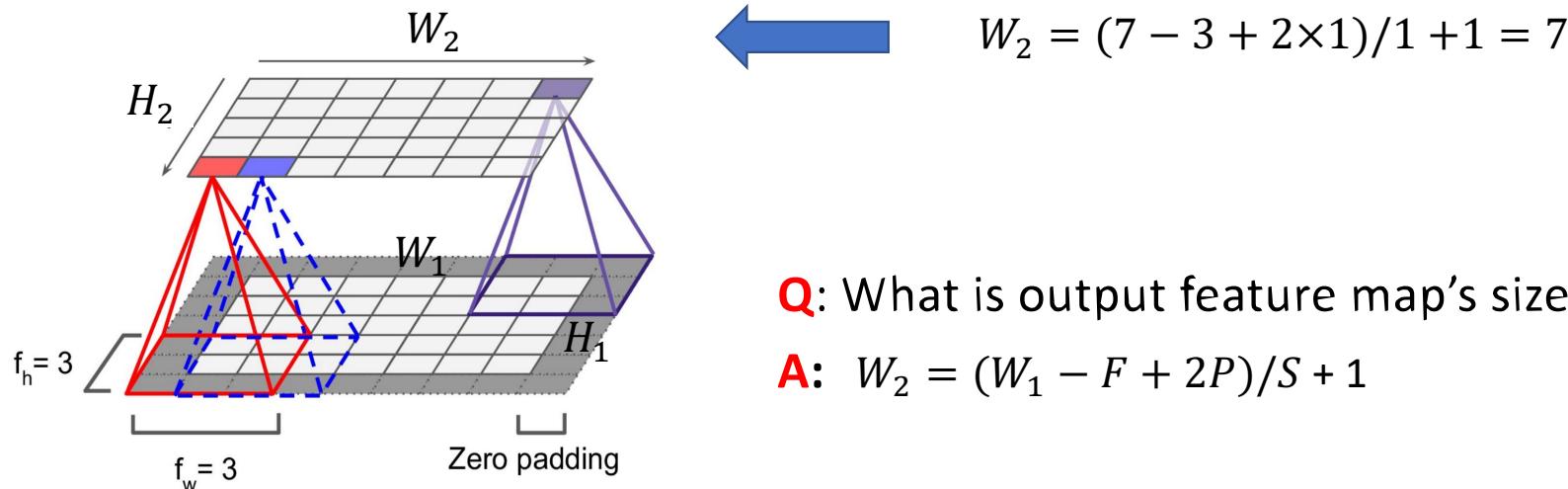


Image from: Aurelien Geron. Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly. 2019.

# The output volume

Given

- The volume of one layer:  $W_1 \times H_1 \times D_1$
- Filter size:  $F$
- Number of filters:  $K$
- Stride:  $S$
- Padding:  $P$

**Q:** What is the volume of the next layer?

**A:**  $W_2 = (W_1 - F + 2P)/S + 1$

$$H_2 = (H_1 - F + 2P)/S + 1$$

$$D_2 = K$$

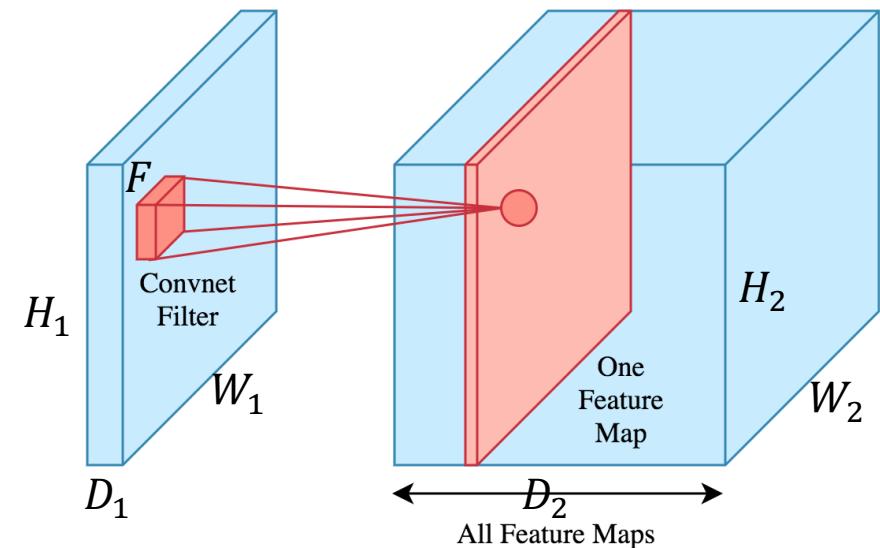
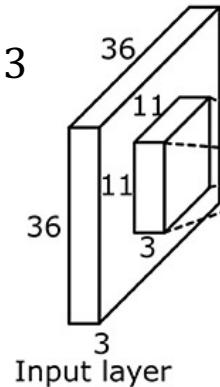


Image from: <https://brilliant.org/wiki/convolutional-neural-network/>

# The output volume - Example

## Example:

- The volume of input layer:  $36 \times 36 \times 3$
- Filter size:  $F = 11$
- Number of filters:  $K = 9$
- Stride:  $S = 1$
- Padding:  $P = 0$



**Q:** What is the volume of the next layer?

# The output volume - Example

## Example:

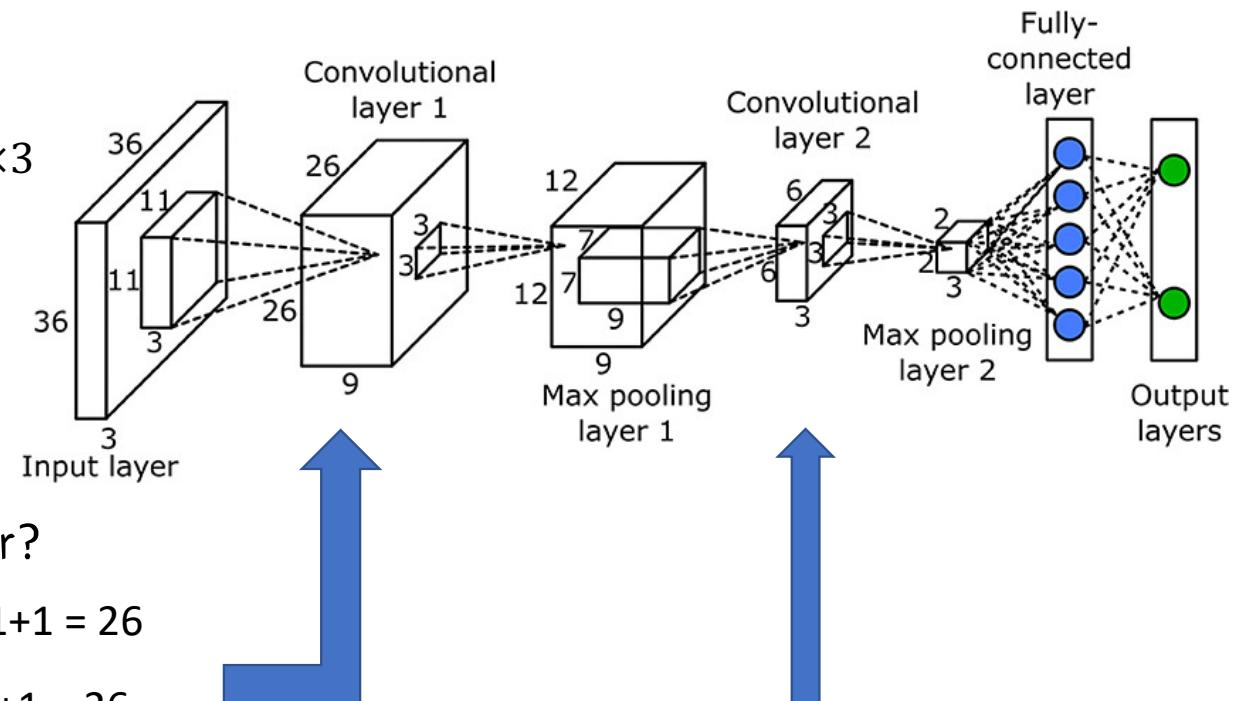
- The volume of input layer:  $36 \times 36 \times 3$
- Filter size:  $F = 11$
- Number of filters:  $K = 9$
- Stride:  $S = 1$
- Padding:  $P = 0$

**Q:** What is the volume of the next layer?

**A:**  $W_2 = (W_1 - F + 2P)/S + 1 = (36-11+0)/1+1 = 26$

$$H_2 = (H_1 - F + 2P)/S + 1 = (36-11+0)/1+1 = 26$$

$$D_2 = K = 9$$

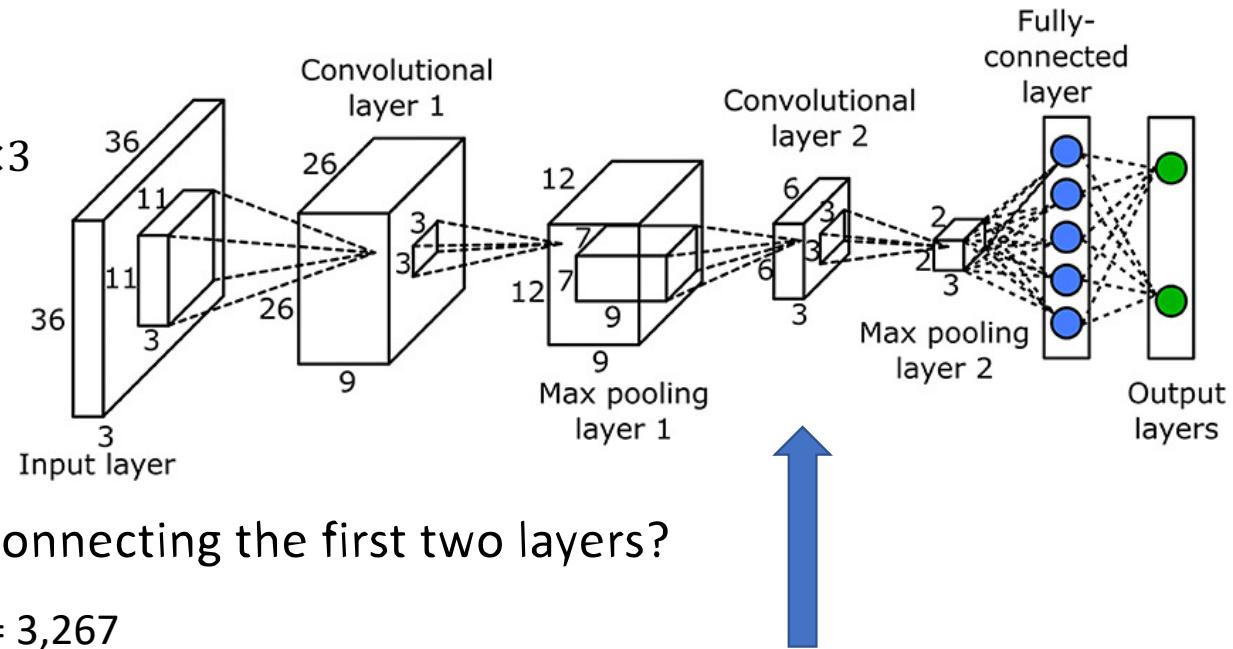


**Q:** Can you work out how this size  $6 \times 6 \times 3$  comes from?

# How many parameters?

## Example:

- The volume of input layer:  $36 \times 36 \times 3$
- Filter size:  $F = 11$
- Number of filters:  $K = 9$
- Stride:  $S = 1$
- Padding:  $P = 0$



**Q:** How many parameters needed for connecting the first two layers?

**A:**  $K \cdot F \cdot F \cdot D_1 + K = 9 \times 11 \times 11 \times 3 + 9 = 3,267$

One bias term for each filter

**Q:** Can you work out how many parameters needed for producing Conv-2?

# Summary of convolutional layer

- Use multiple filters to extract different local features
- Preserve the spatial relationship
- Local connectivity
- Parameter sharing → Use less number of parameters

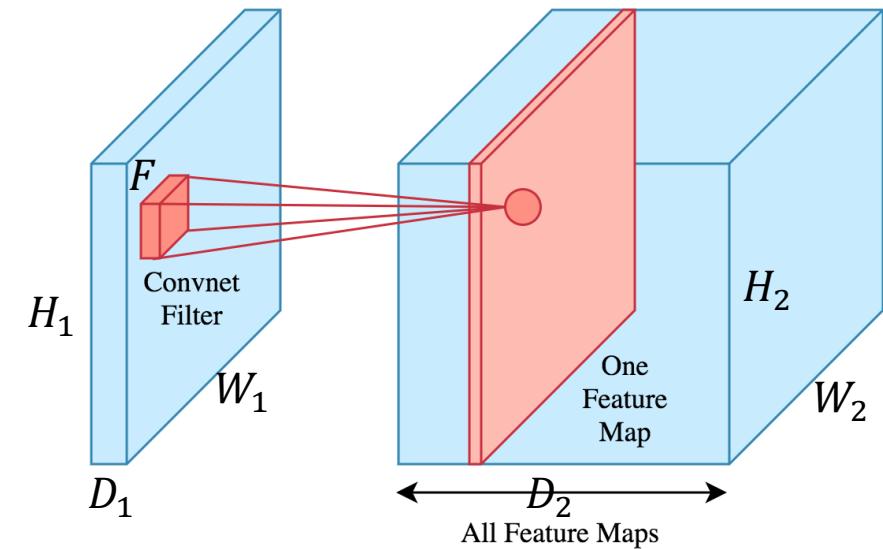


Image from: <https://brilliant.org/wiki/convolutional-neural-network/>

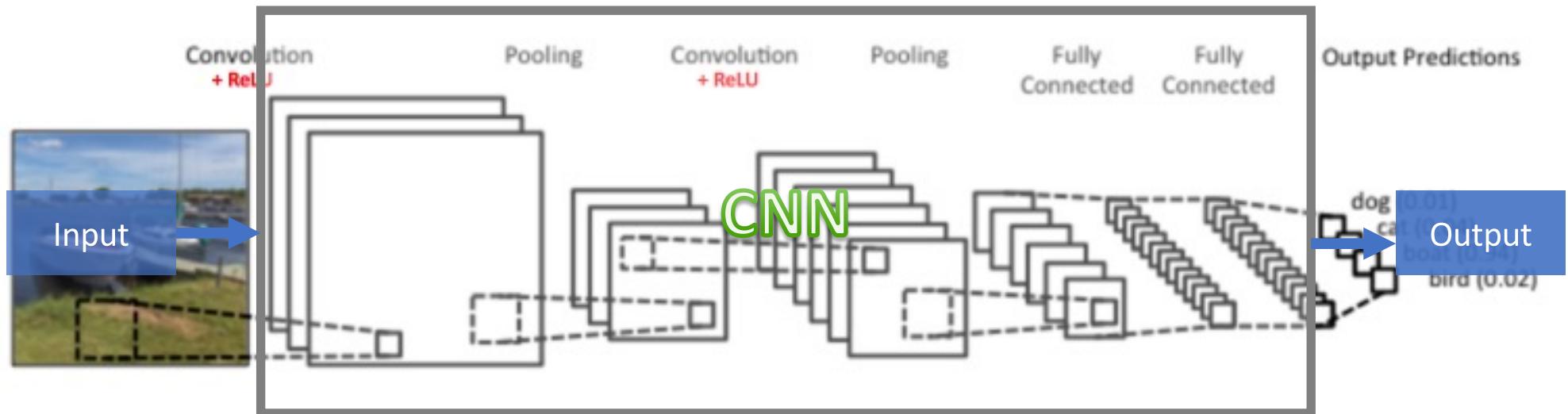
# Reference for Topic 3

- Video lecture by Brandon Rohrer: How Convolutional Neural Networks work: <https://www.youtube.com/watch?v=FmpDlaiMleA>
- Video lecture by Alexander Amini: MIT course on Convolutional Neural Networks: <https://www.youtube.com/watch?v=iaSUYvmCekI>
- [Arden Dertat's Blog: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2](https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2)
- Stanford course on CNNs: <https://cs231n.github.io/convolutional-networks/>

# Topic 4:

# CNN: ReLU Layer & Pooling Layer

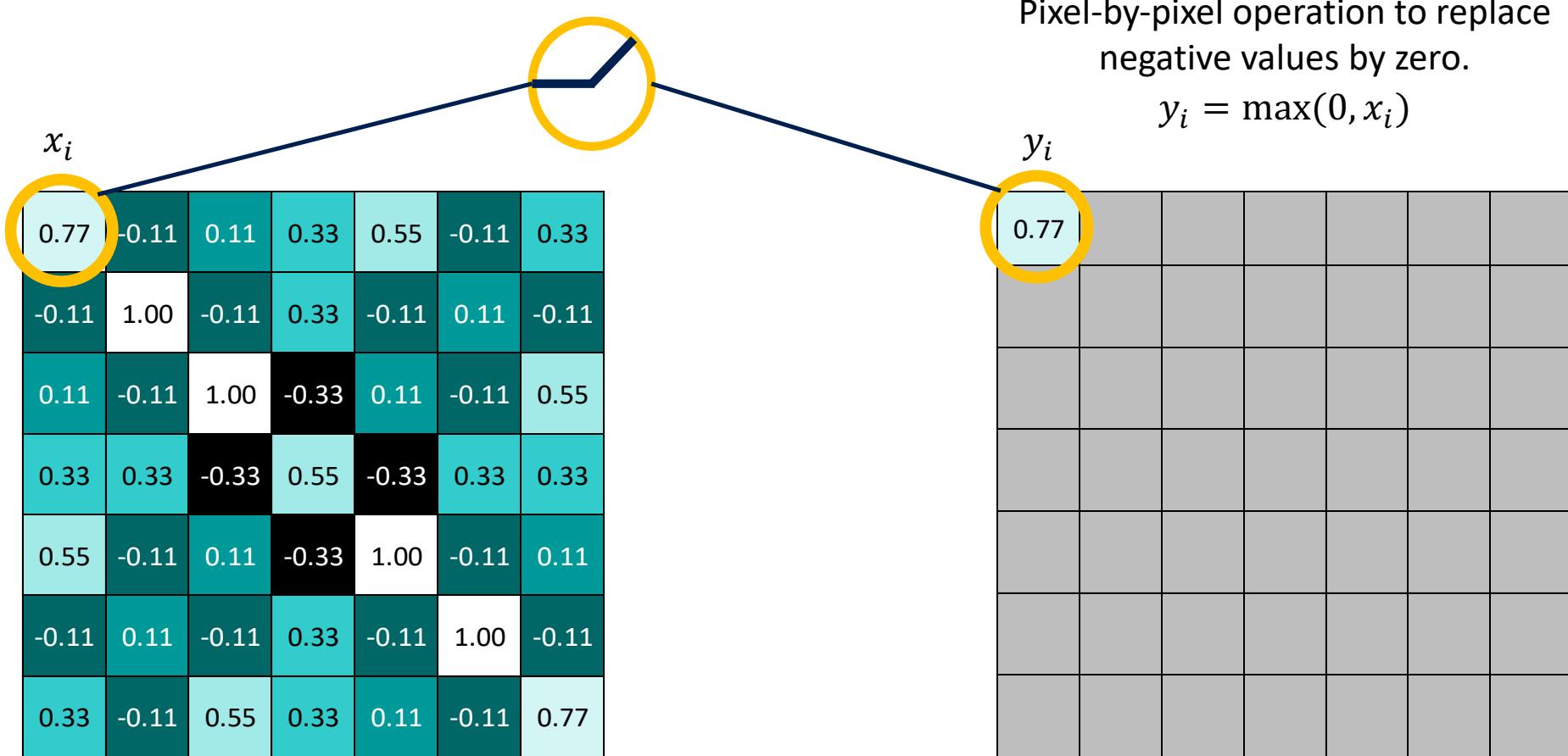
# Image recognition with CNN



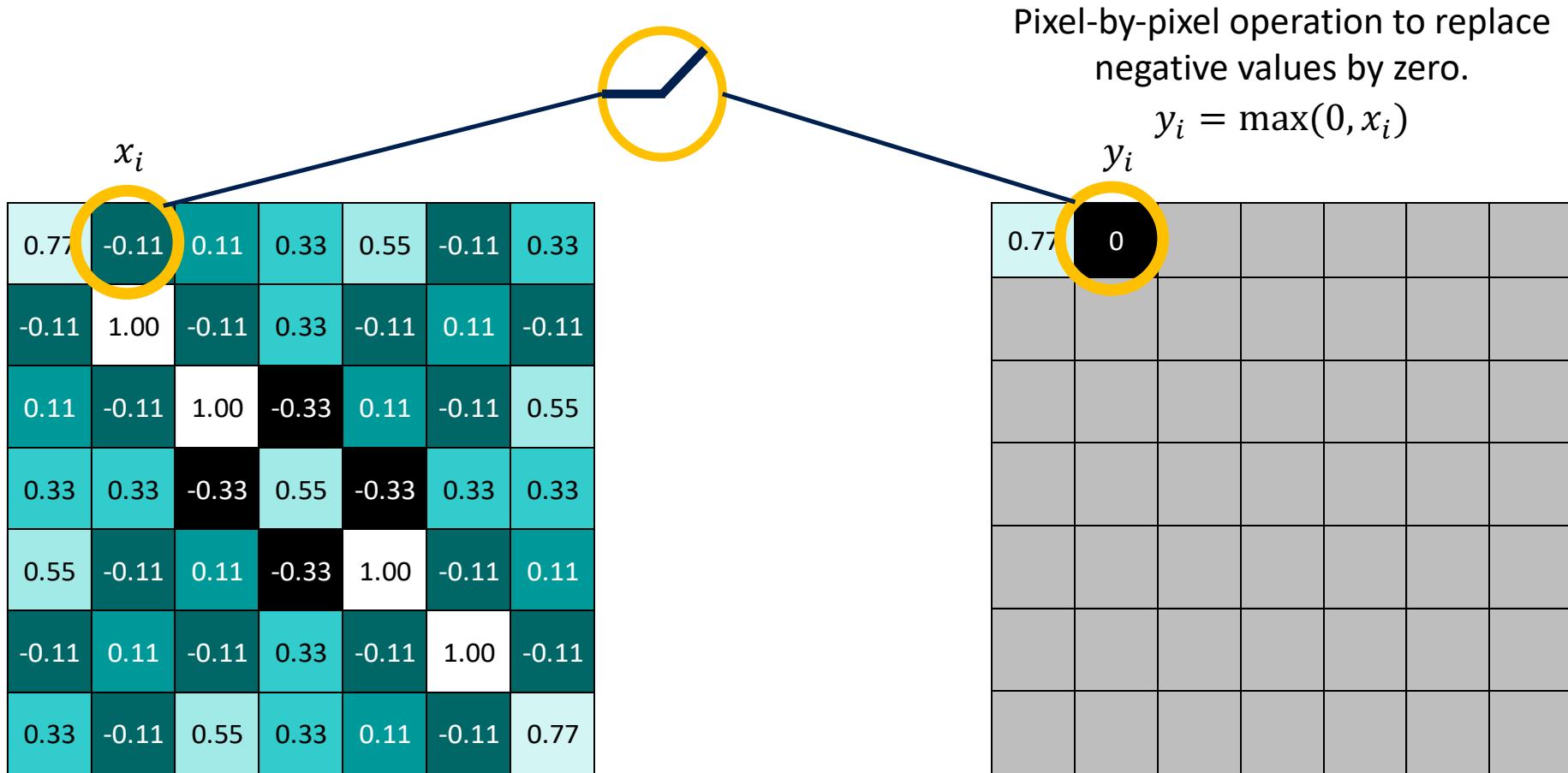
- Convolutional Layer
- **ReLU Layer**
- Pooling Layer
- Fully Connected Layer

Image from: <https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f>

# Rectified Linear Unit (ReLU)

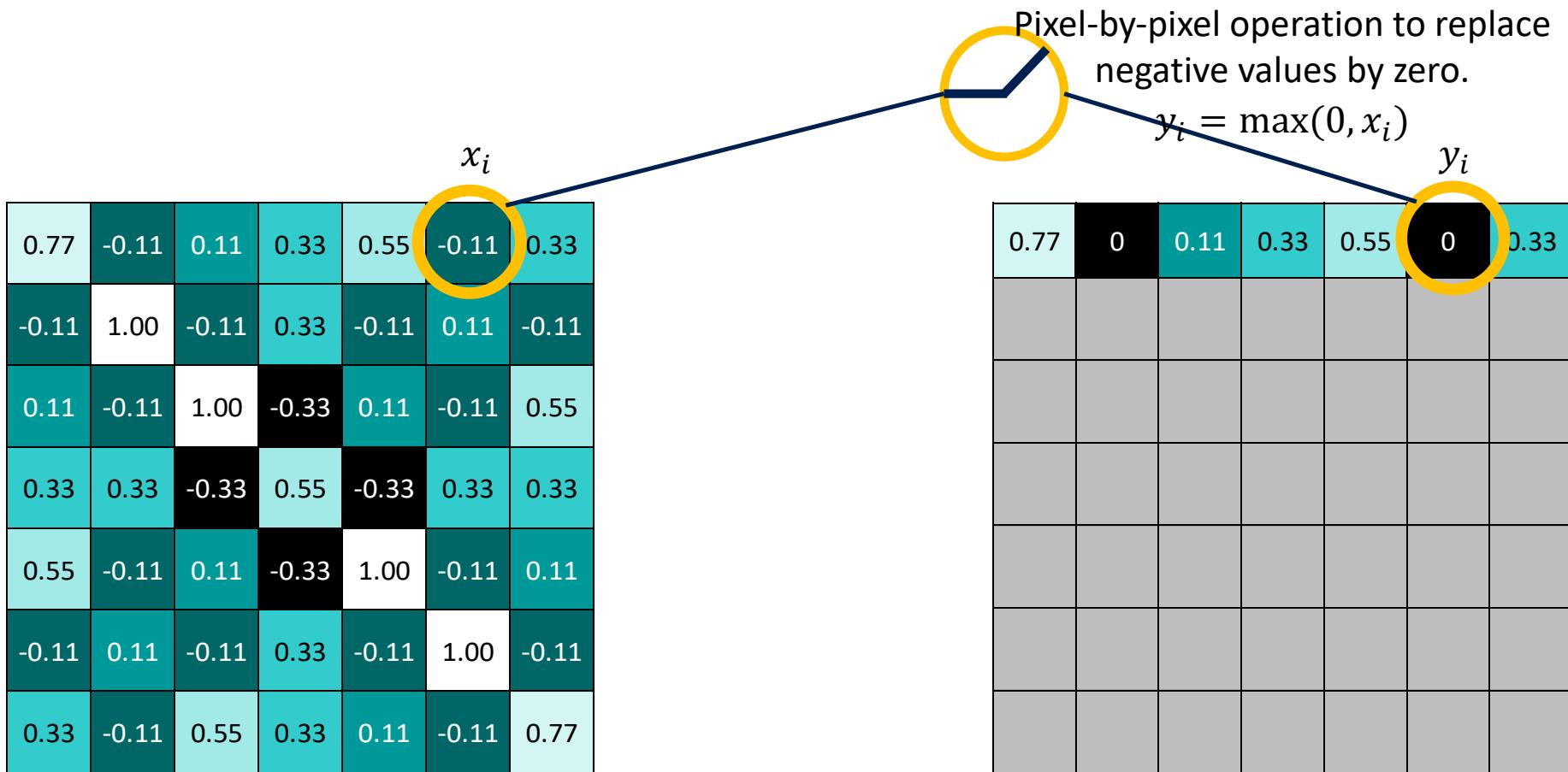


# Rectified Linear Unit (ReLU)





# Rectified Linear Unit (ReLU)



# Rectified Linear Unit (ReLU)

Pixel-by-pixel operation to replace negative values by zero.

$$y_i = \max(0, x_i)$$

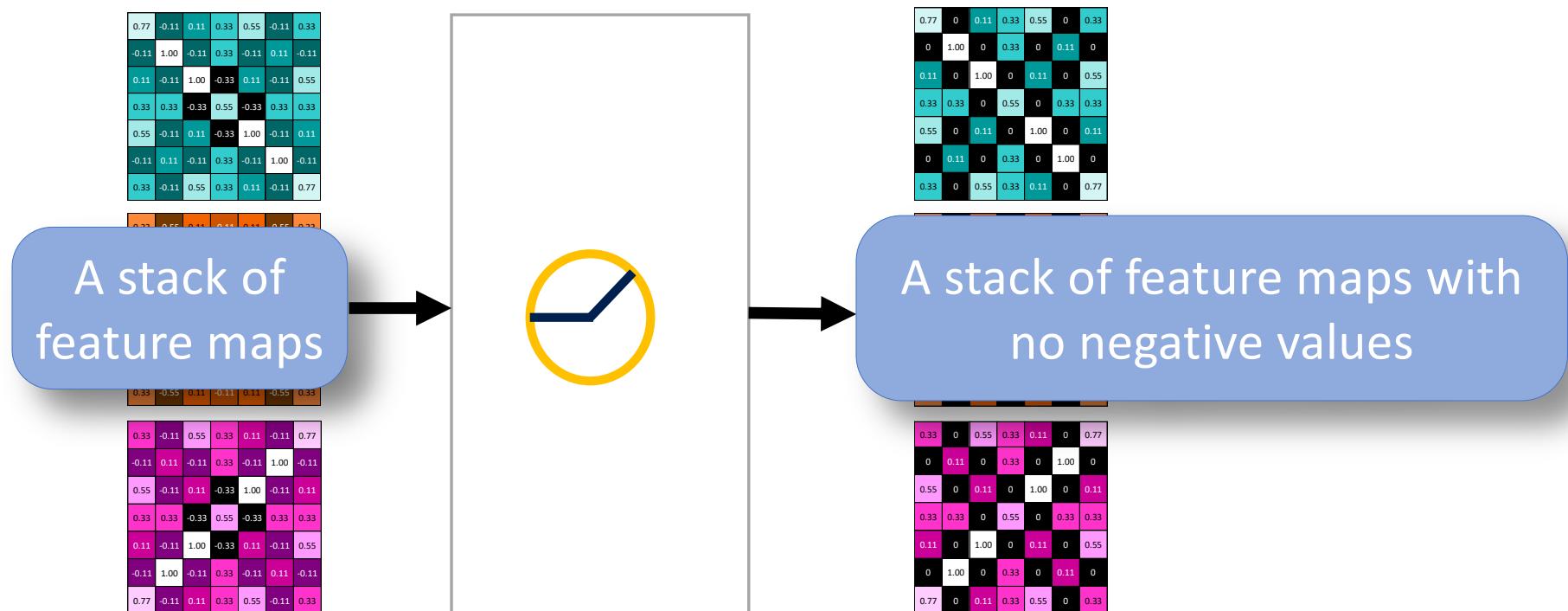
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



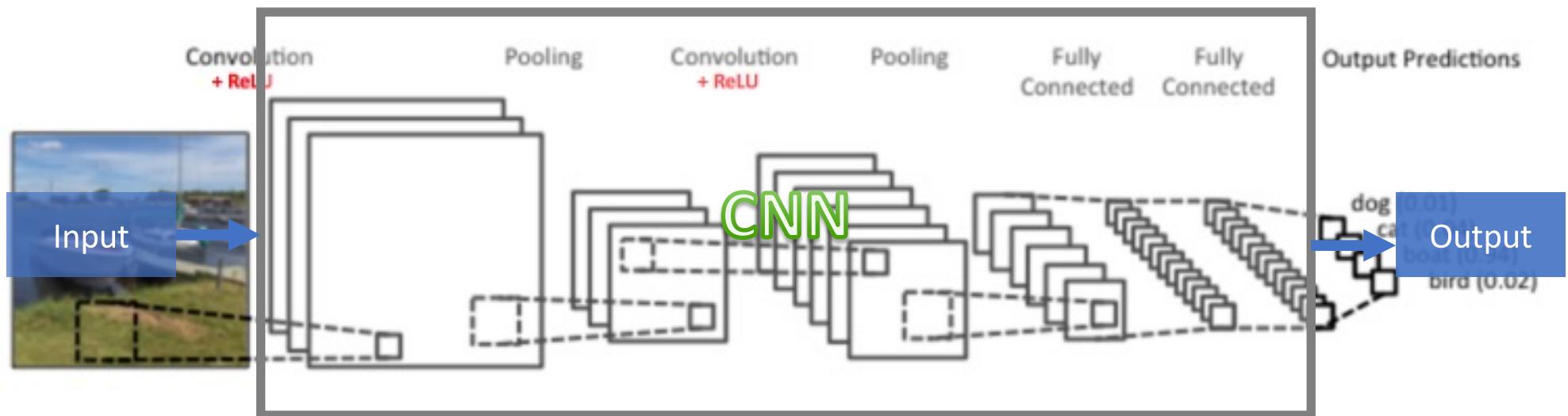
0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

# Summary of ReLU layer

- Introduce non-linearity in the CNN network
- Apply after every convolutional operation



# Image classification with CNN



- Convolutional Layer
  - ReLU Layer
- Pooling Layer**
- Fully Connected Layer

Image from: <https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f>

# Pooling

0.77	-0.11	0.11	0.33	0.55	-0.11
-0.11	1.00	-0.11	0.33	-0.11	0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11
0.33	0.33	-0.33	0.55	-0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00

maximum

Filter size:  $F = 2$   
Stride:  $S = 2$

1.00		

# Pooling

0.77	-0.11	0.11	0.33	0.55	0.11
-0.11	1.00	-0.11	0.33	-0.11	0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11
0.33	0.33	-0.33	0.55	-0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00

maximum

Filter size:  $F = 2$   
Stride:  $S = 2$

1.00	0.33	

# Pooling

0.77	-0.11	0.11	0.33	0.55	-0.11
-0.11	1.00	-0.11	0.33	-0.11	0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11
0.33	0.33	-0.33	0.55	-0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00

maximum

Filter size:  $F = 2$   
Stride:  $S = 2$

1.00	0.33	0.55
0.33		

# Pooling

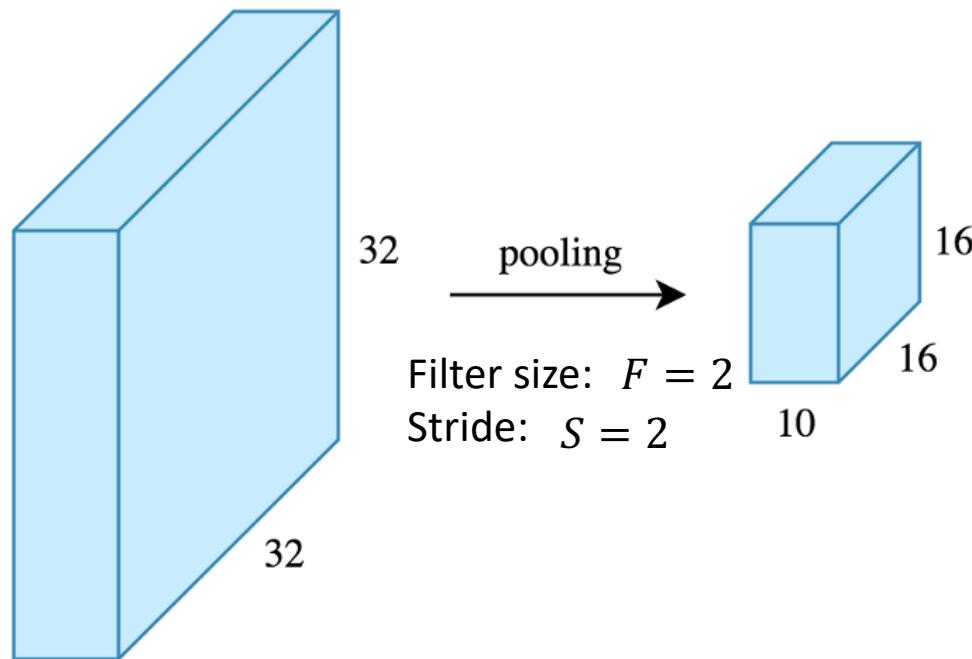
0.77	-0.11	0.11	0.33	0.55	-0.11
-0.11	1.00	-0.11	0.33	-0.11	0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11
0.33	0.33	-0.33	0.55	-0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00

max pooling



1.00	0.33	0.55
0.33	1.00	0.33
0.55	0.33	1.00

# The output volume

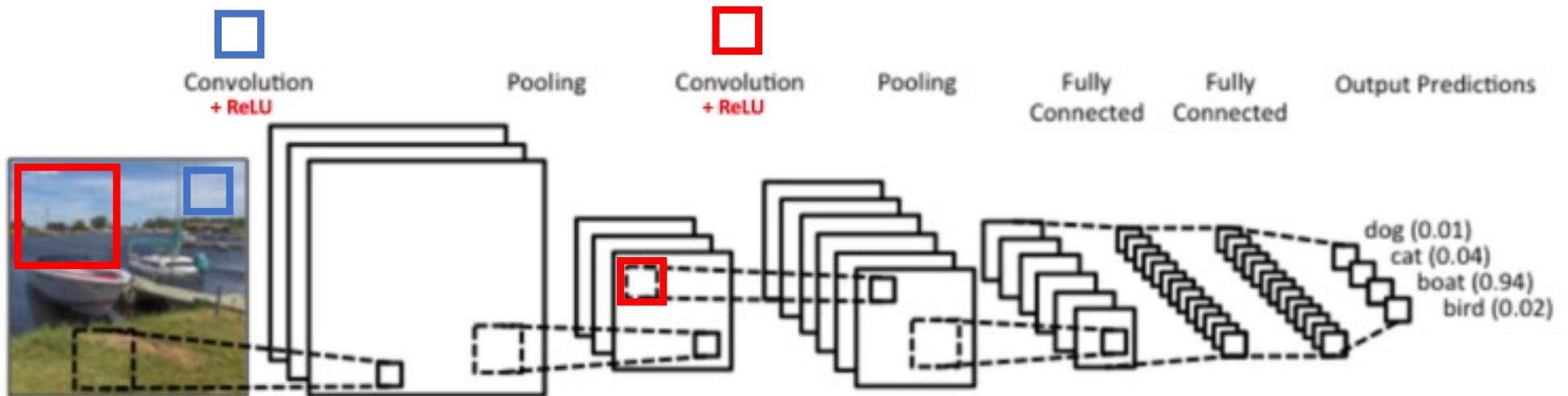


$$W_2 = (W_1 - F)/S + 1 = (32-2)/2+1 = 16$$

$$H_2 = (H_1 - F)/S + 1 = (32-2)/2+1 = 16$$

$$D_2 = D_1 = 10$$

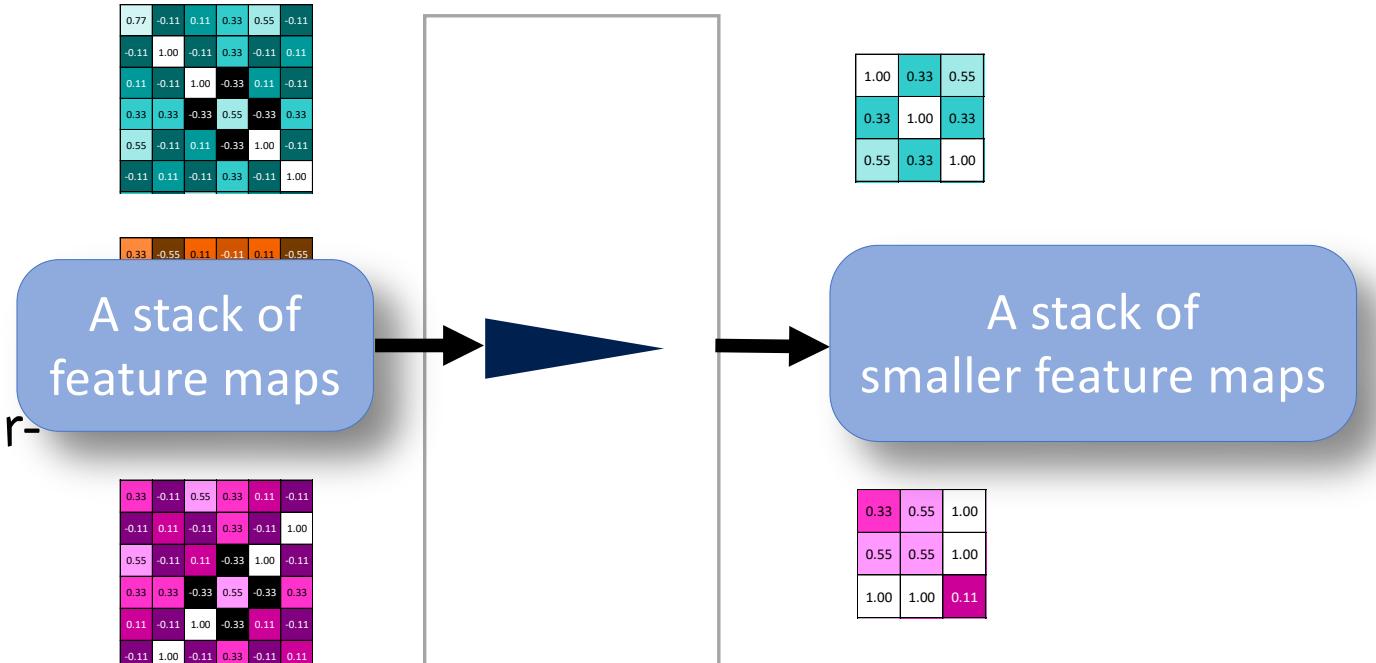
**Q:** If both the convolutional layers below use a same-sized filter, what is the difference of their receptive field size on the input image?



Because of Pooling, higher-level convolutional filters can cover a **larger region** of the image than equal-sized filters in the lower layers.

# Summary of pooling

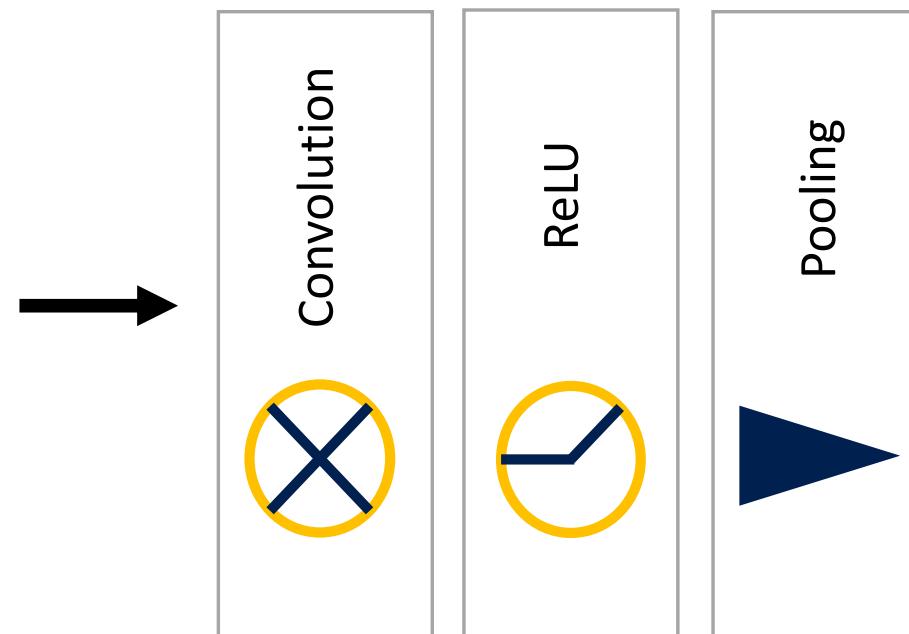
- Average pooling; L2-norm pooling; **Max pooling** (most common)
- Need no parameters
- Reduce dimensionality
- Some spatial invariance
- Larger coverage for higher-layer Conv layers, enable them represents higher-level features



# Layers get stacked

The output of one becomes the input of the next.

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

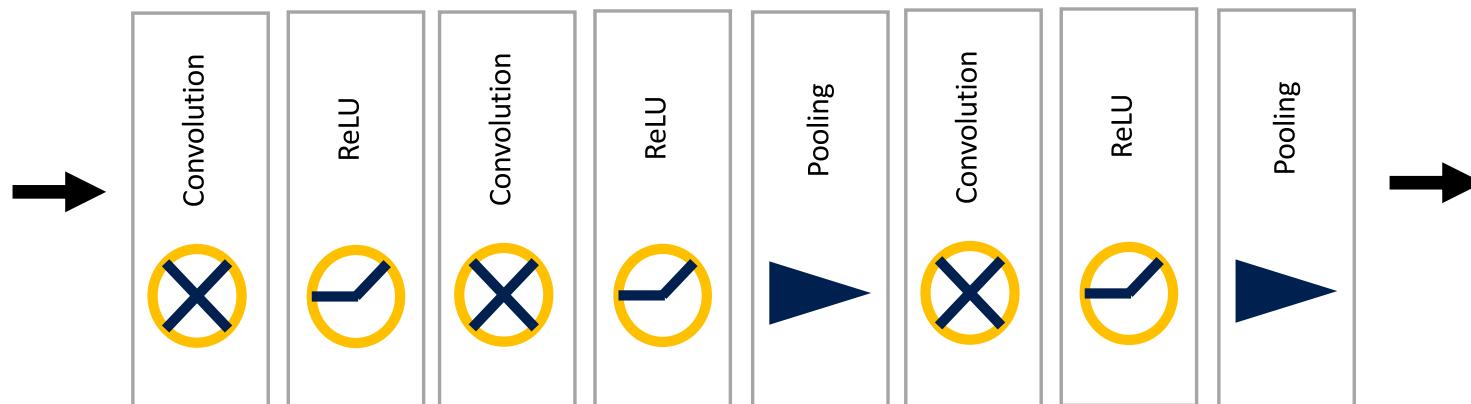
  

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

# Deep stacking

Layers can be repeated several (or many) times.

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



1.00	0.55
0.55	1.00
1.00	0.55
0.55	0.55
0.55	1.00
1.00	0.55

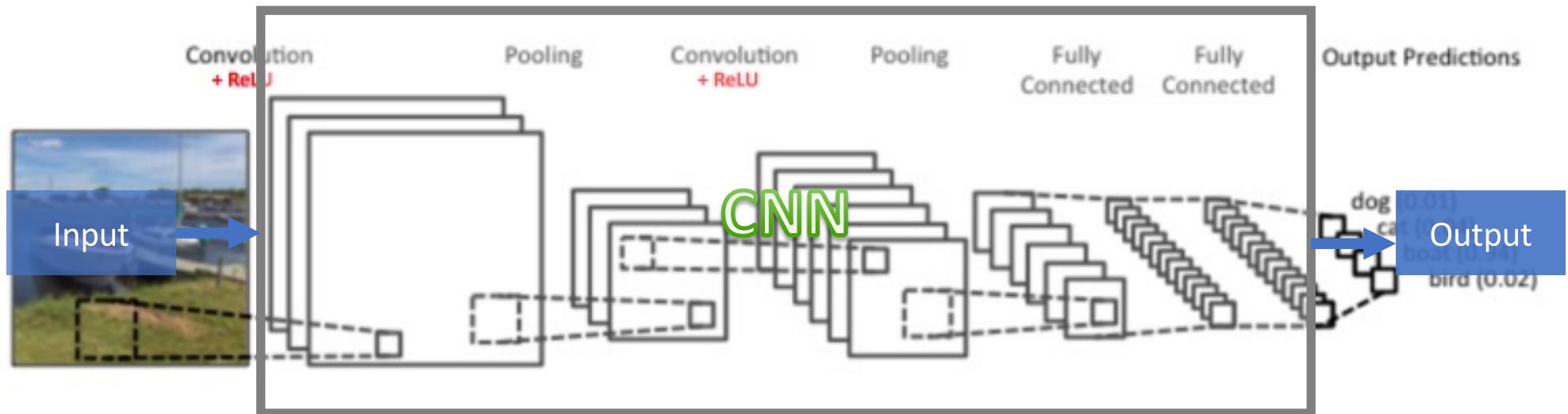
# Reference for Topic 4

- Video lecture by Brandon Rohrer: How Convolutional Neural Networks work: <https://www.youtube.com/watch?v=FmpDlaiMleA>
- Video lecture by Alexander Amini: MIT course on Convolutional Neural Networks: <https://www.youtube.com/watch?v=iaSUYvmCekI>
- [Arden Dertat's Blog: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2](https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2)
- Stanford course on CNNs: <https://cs231n.github.io/convolutional-networks/>

# Topic 5:

# CNN: Fully Connected (FC) Layer

# Image classification with CNN

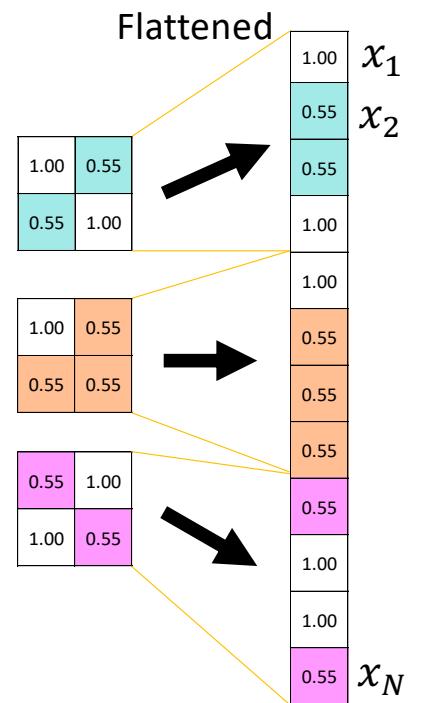


- Convolutional Layer
- ReLU Layer
- Pooling Layer

**→ Fully Connected Layer**

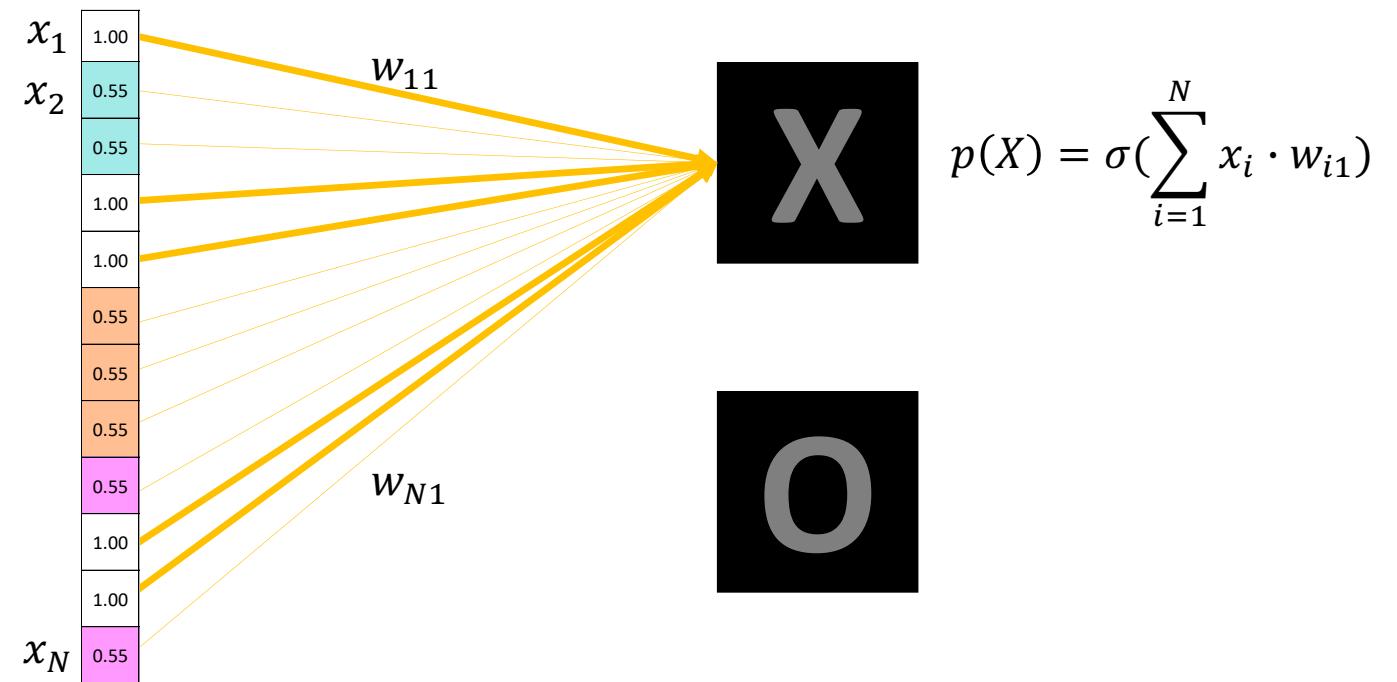
Image from: <https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f>

# Fully connected layer



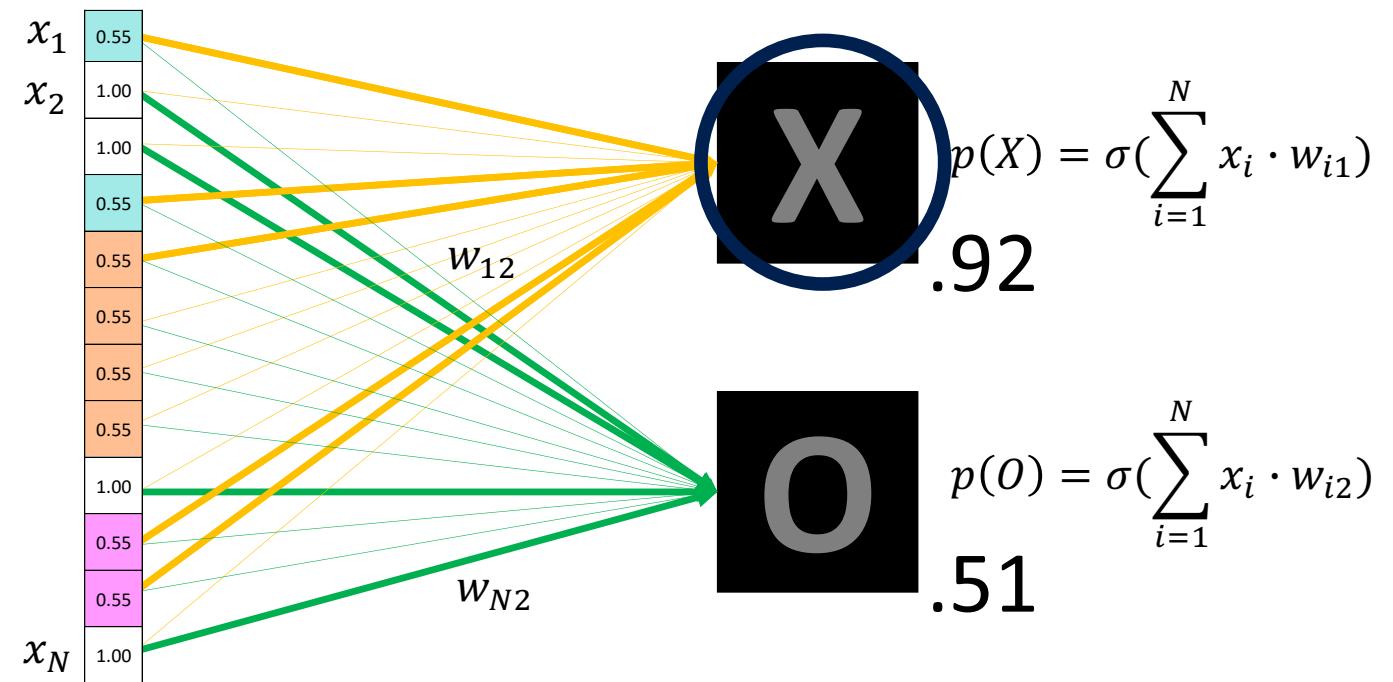
# Fully connected layer

Every feature ( $x$ ) gets a vote ( $w$ )



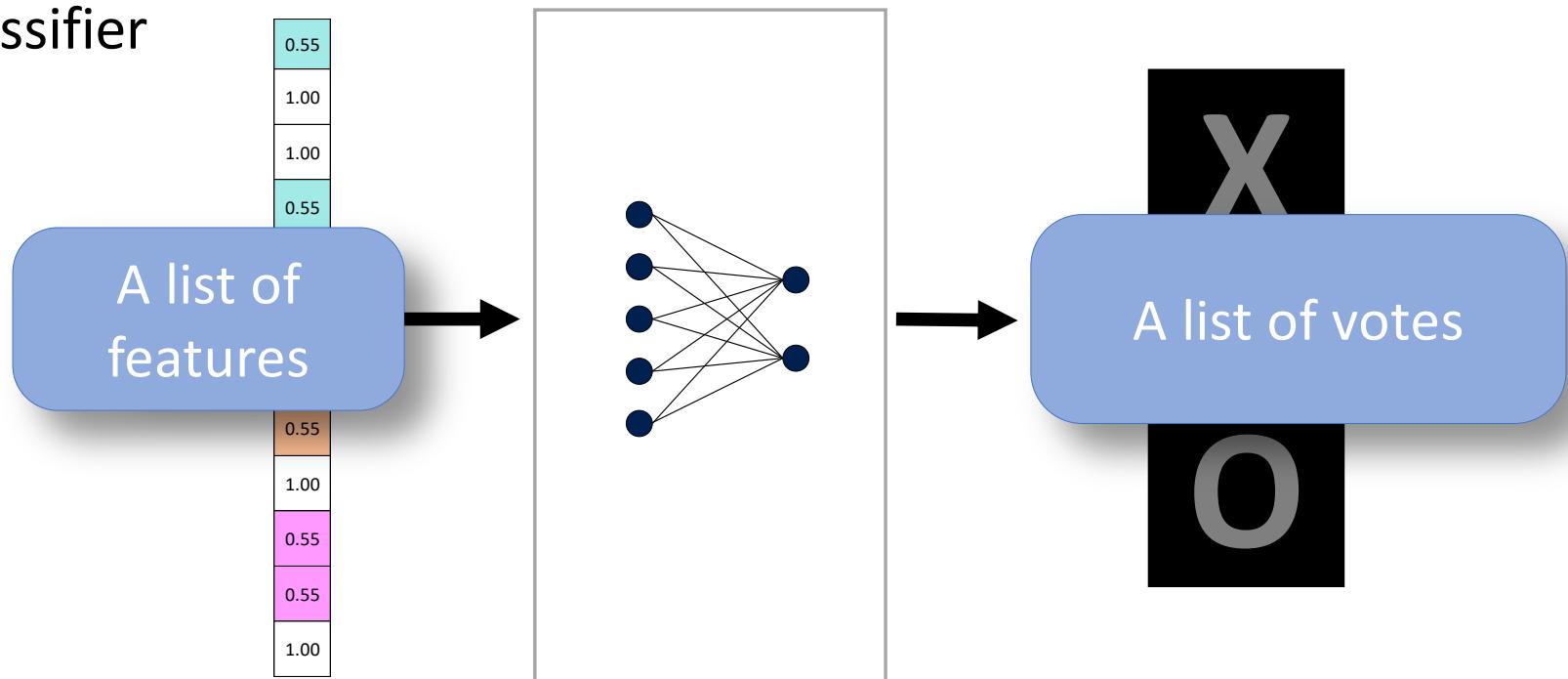
# Fully connected layer

Every feature ( $x$ ) gets a vote ( $w$ )



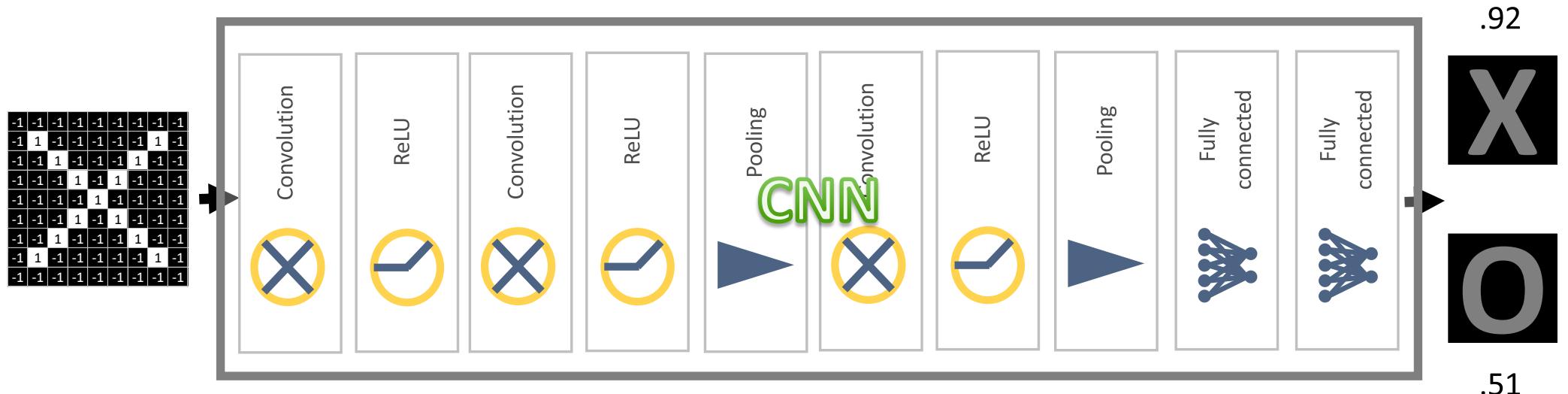
# Summary of fully connected (FC) layer

- Usually at the end of the CNN
- Can stack one or more FC layers
- Served as a classifier



# Putting it all together

A set of pixels becomes a set of votes.



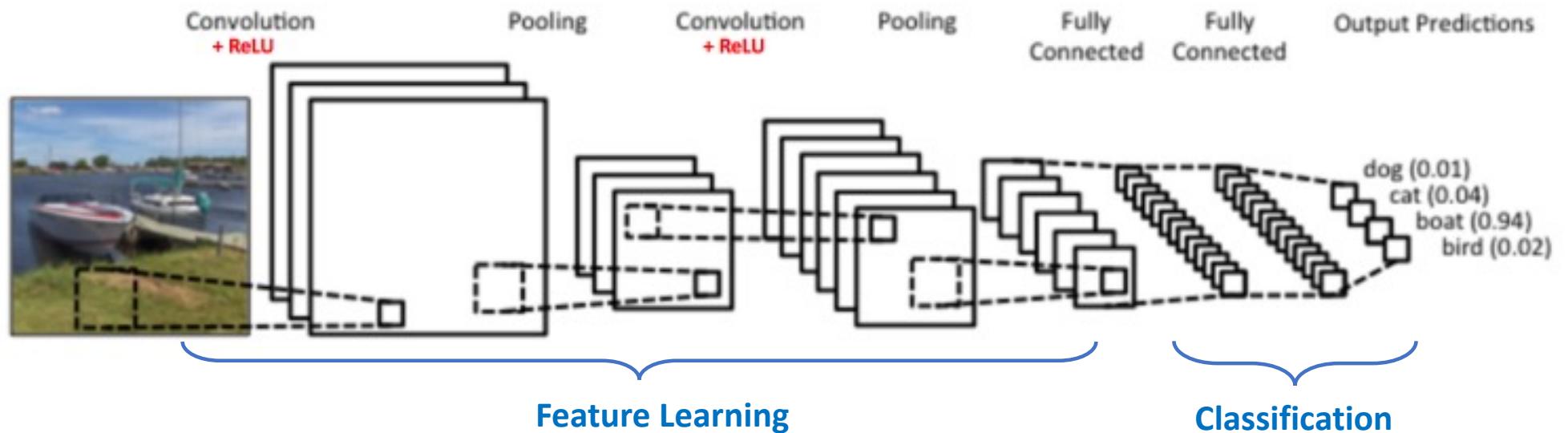
# Reference for Topic 5

- Video lecture by Brandon Rohrer: How Convolutional Neural Networks work: <https://www.youtube.com/watch?v=FmpDlaiMleA>
- Video lecture by Alexander Amini: MIT course on Convolutional Neural Networks: <https://www.youtube.com/watch?v=iaSUYvmCekI>
- [Arden Dertat's Blog: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2](https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2)
- Stanford course on CNNs: <https://cs231n.github.io/convolutional-networks/>

# Topic 6:

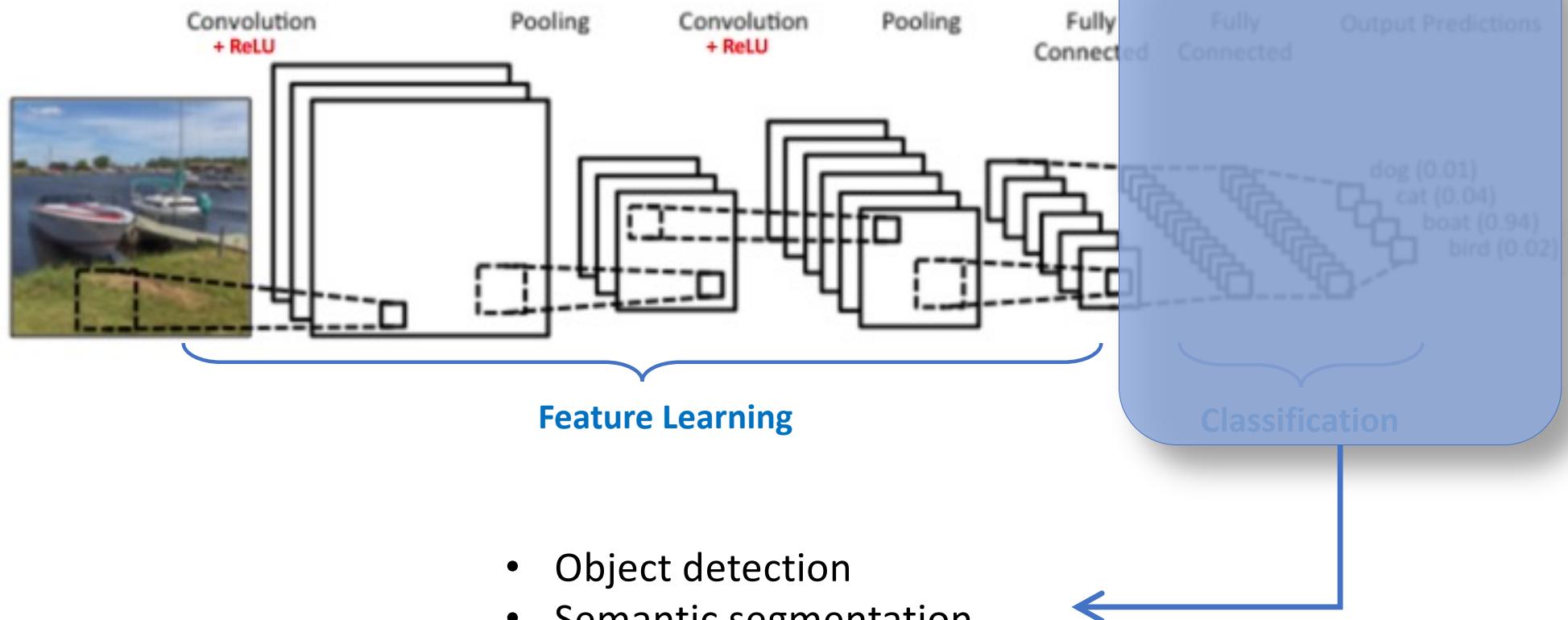
# Summary of CNNs

# Structure of CNN



- **Convolutional Layer** ← Feature extraction (from lower level to higher level)
- **ReLU Layer** ← Introduce non-linearity
- **Pooling Layer** ← Reduce dimensionality and preserve spatial invariance
- **Fully Connected Layer** ← Classification with softmax activation

# CNN not only for image classification



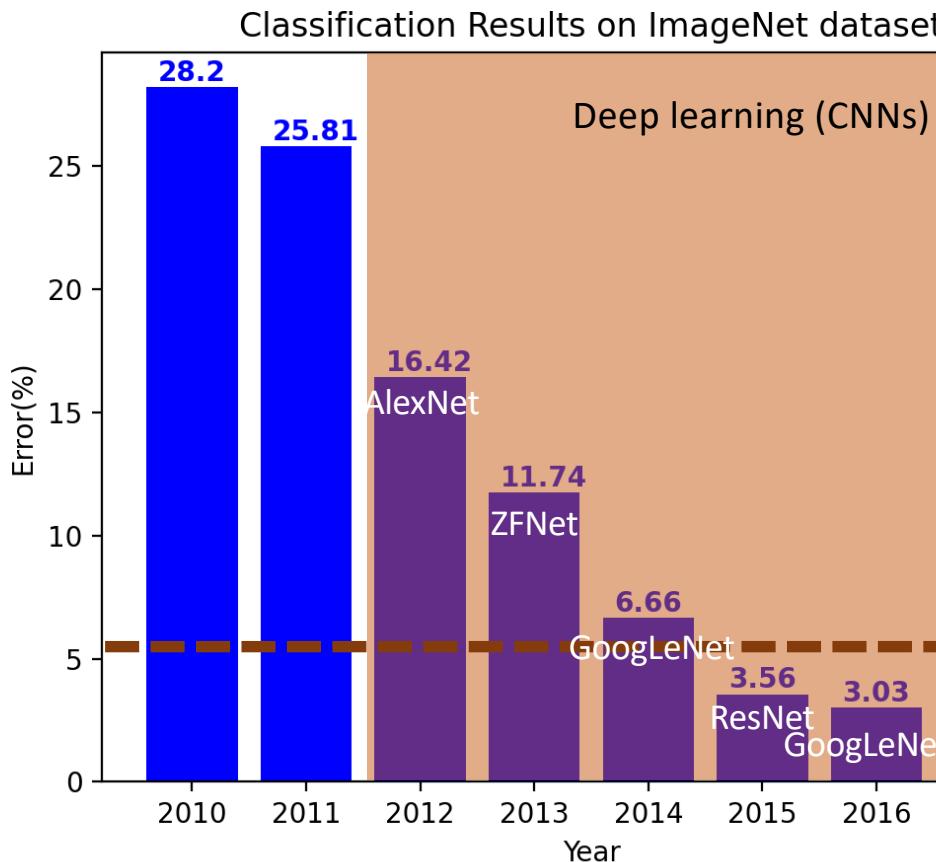
- Object detection
- Semantic segmentation
- Image captioning
- ...

# Implementation

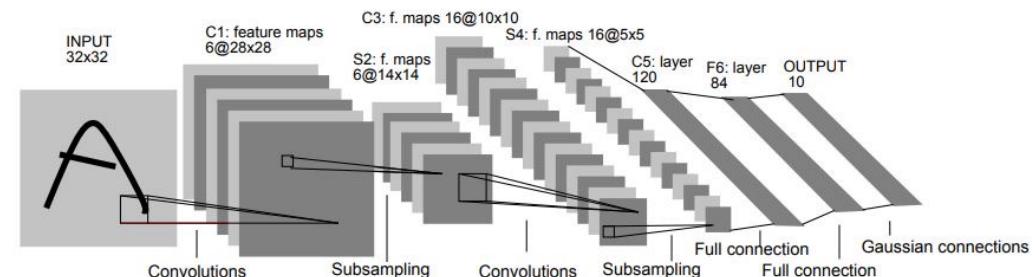
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

# Common CNN architectures



- LeNet-5 (1998): one of the very first CNNs.



LeCun, Y. et al, "Gradient-based learning applied to document recognition". 1998.

Human

More details of common CNN architectures:  
<https://machinelearningknowledge.ai/popular-image-classification-models-in-imagenet-challenge-ilsvrc-competition-history/>

# Case study: VGGNet

- Proposed by Oxford's Visual Geometry Group (VGG) in 2014
- 7.3% error rate in ILSVRC2014
- 13 conv layers + 3 fully-connected layers – also called **VGG16**
- **Q:** All stacked 3x3 conv layers, *why?*

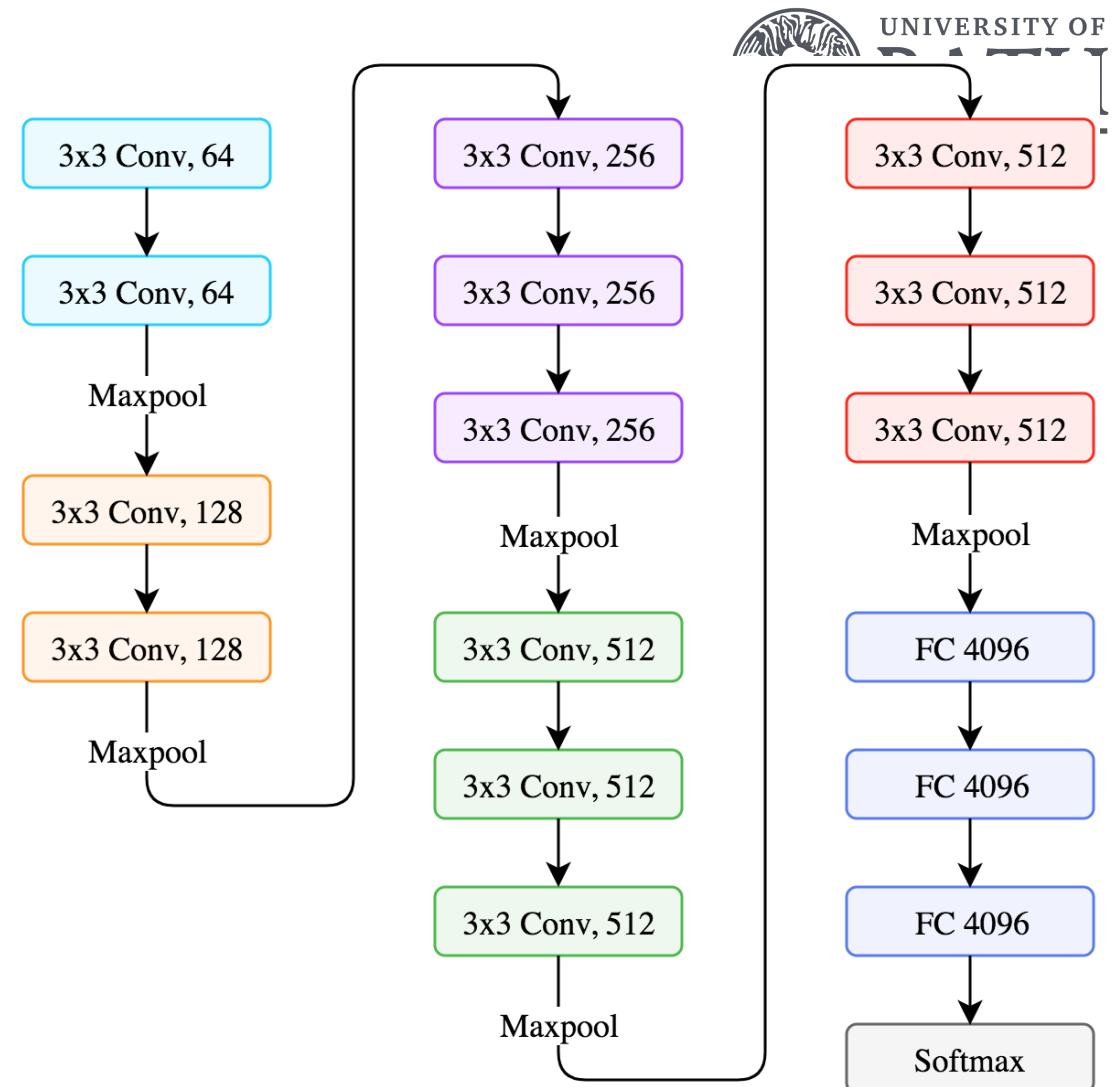
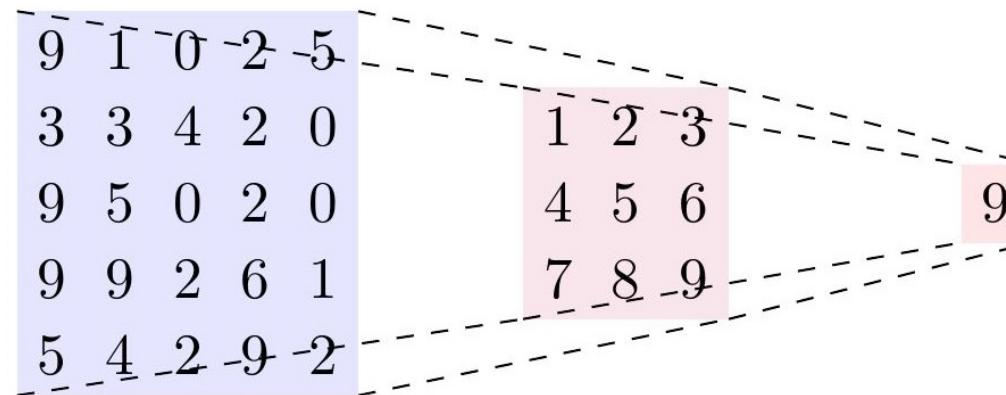


Image from: <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>

# Case study: VGGNet

$3 \times 3$  CONV  $\rightarrow$   $3 \times 3$  CONV



- Two  $3 \times 3$  conv layers has an effective receptive field of  $5 \times 5$ .
- Benefit for using two  $3 \times 3$  conv layers:
  - a decrease in the number of parameters
  - two ReLU layers, and more non-linearity gives more power to the model

# Case study: VGGNet

- Three 3x3 conv layers has an effective receptive field of 7x7, but less parameters, more non-linearity power.
- Prefer a **stack of small filter CONV** to one large receptive field CONV layer.

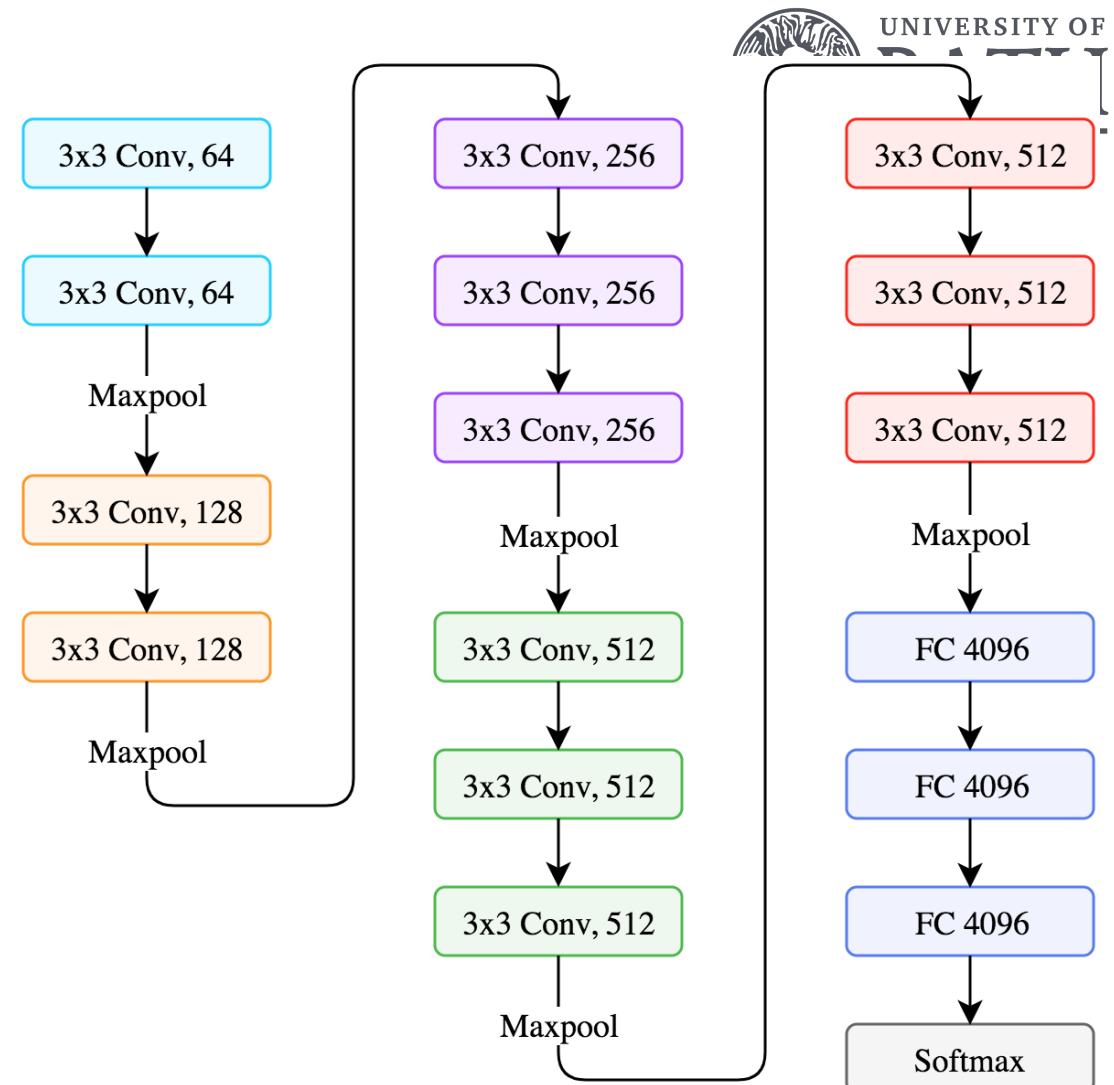


Image from: <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>

# Exercise

Search online the most popularly used CNN architectures. Choose at least 5 CNN models. For each model, give a brief introduction to it. What is the architecture? What are the main characteristics?

# Reference for Topic 6

- Video lecture by Alexander Amini: MIT course on Convolutional Neural Networks: <https://www.youtube.com/watch?v=iaSUYvmCekI>
- Arden Dertat's Blog: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2
- Stanford course on CNNs: <https://cs231n.github.io/convolutional-networks/>
- <https://machinelearningknowledge.ai/popular-image-classification-models-in-imagenet-challenge-ilsvrc-competition-history/>

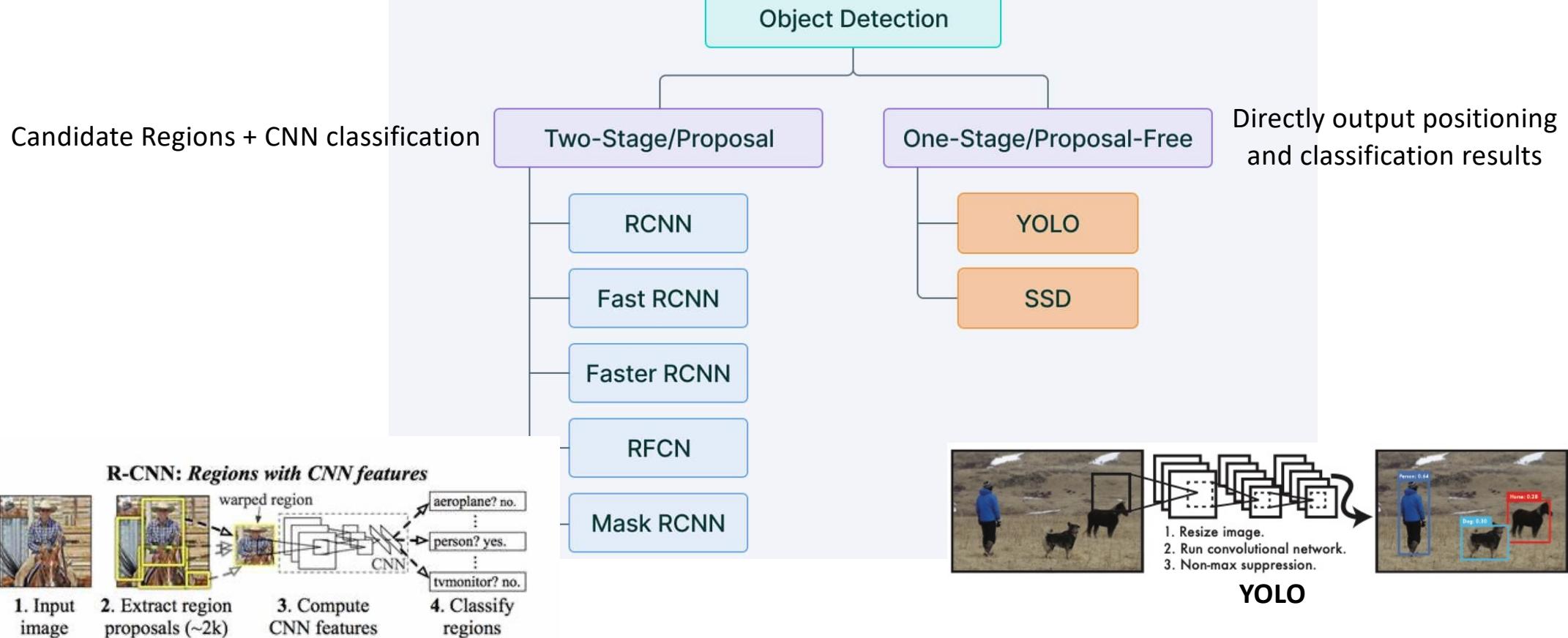
# Week 03: Convolutional Neural Networks (CNNs)

Machine Learning 2

Dr. Hongping Cai

# Topic 7: Object Detection - YOLO

# Two types of methods



## You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon

University of Washington

[pjreddie@cs.washington.edu](mailto:pjreddie@cs.washington.edu)

Ross Girshick

Facebook AI Research

[rbg@fb.com](mailto:rbg@fb.com)

Santosh Divvala

Allen Institute for Artificial Intelligence

[santoshd@allenai.org](mailto:santoshd@allenai.org)

Ali Farhadi

University of Washington

[ali@cs.washington.edu](mailto:ali@cs.washington.edu)

### Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network pre-

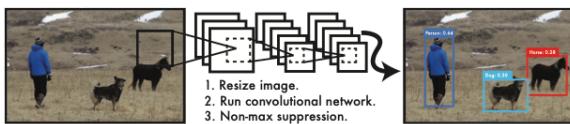


Figure 1: The YOLO Detection System. Processing images

### You only look once: Unified, real-time object detection

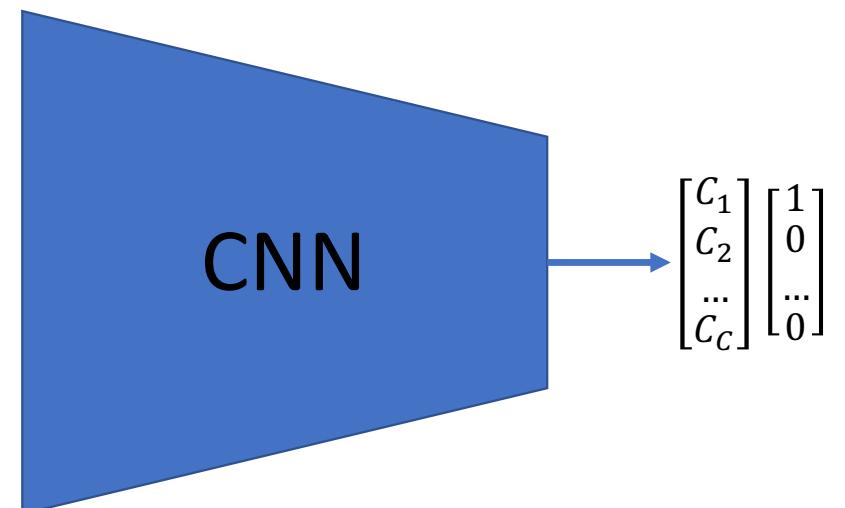
[J Redmon](#), [S Divvala](#), [R Girshick](#)... - Proceedings of the IEEE ..., 2016 - cv-foundation.org

... Using our system, **you only look once** (YOLO) at an image to ... of-the-art detectors, we **look** at a detailed breakdown of results ... [19] For each category at test time we **look** at the top N predic...

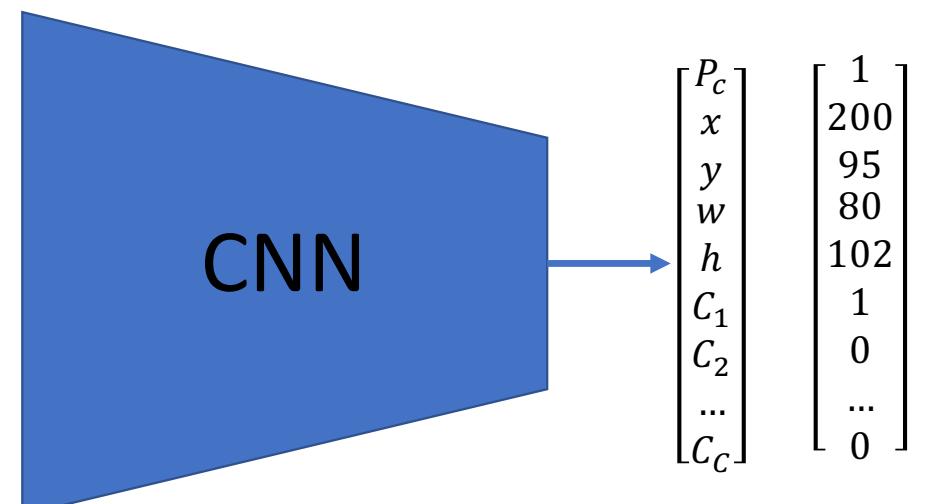
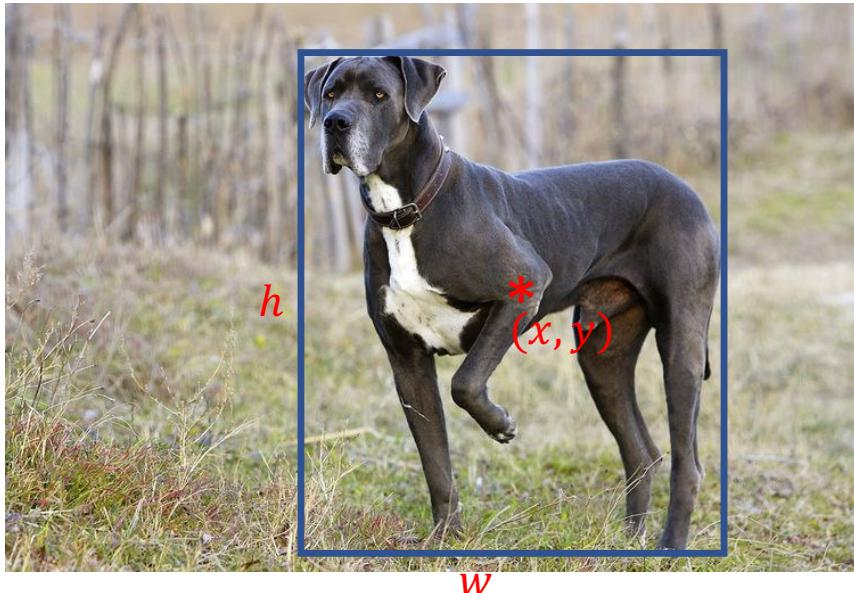
☆ Save 99 Cite Cited by 44281 Related articles All 48 versions ☰



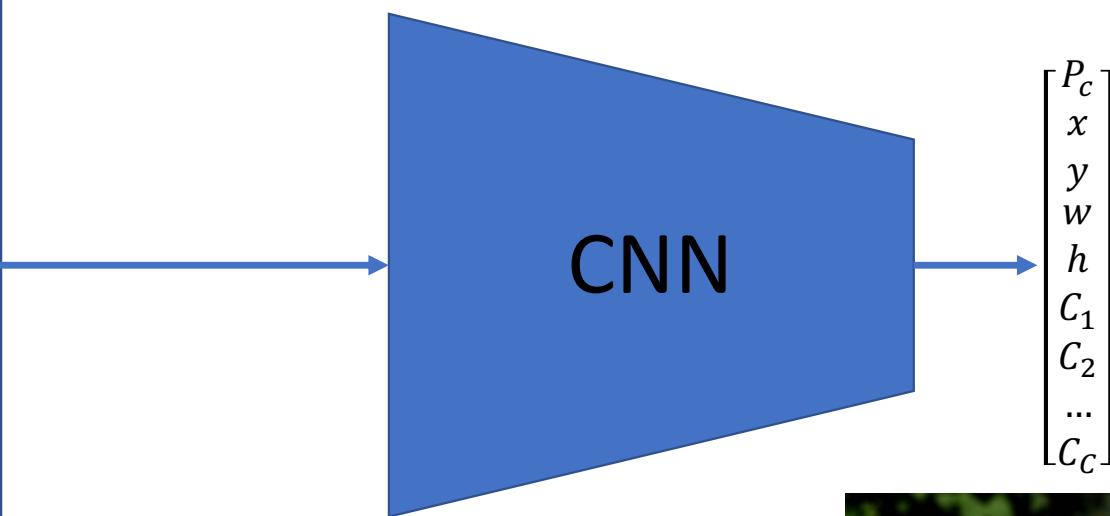
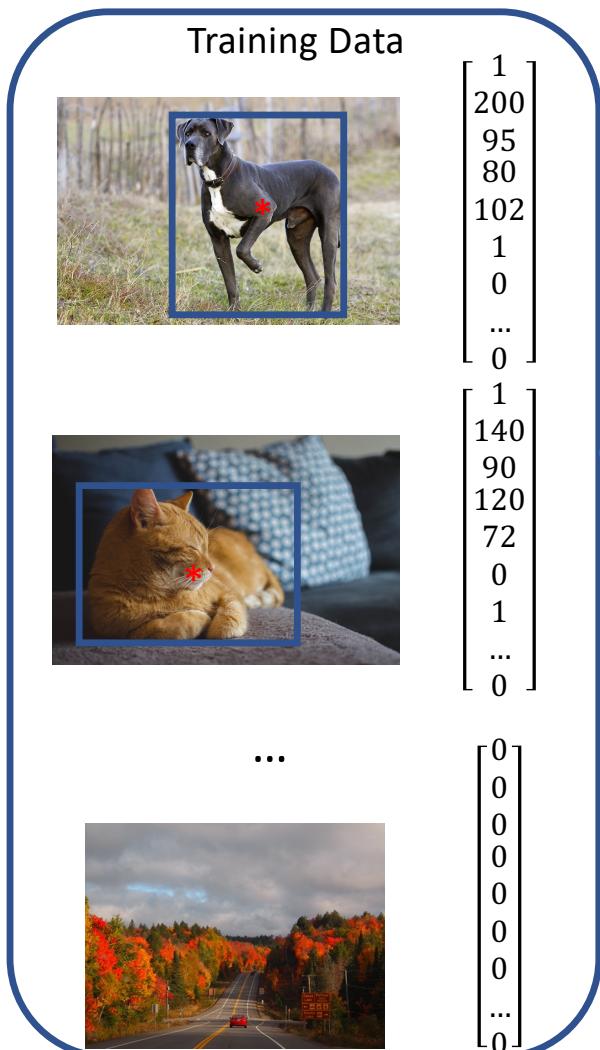
# Image classification



# Single object localization



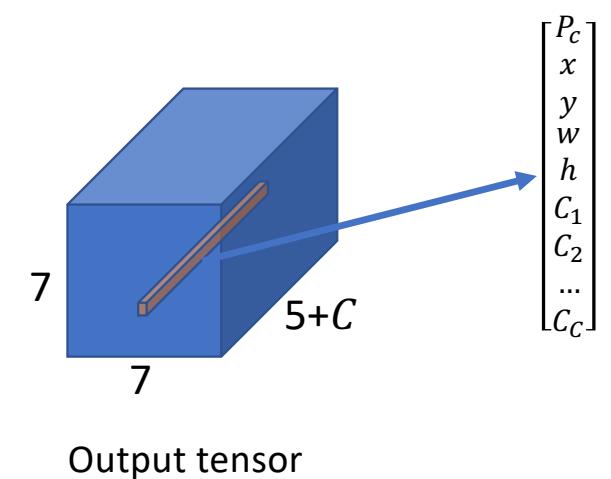
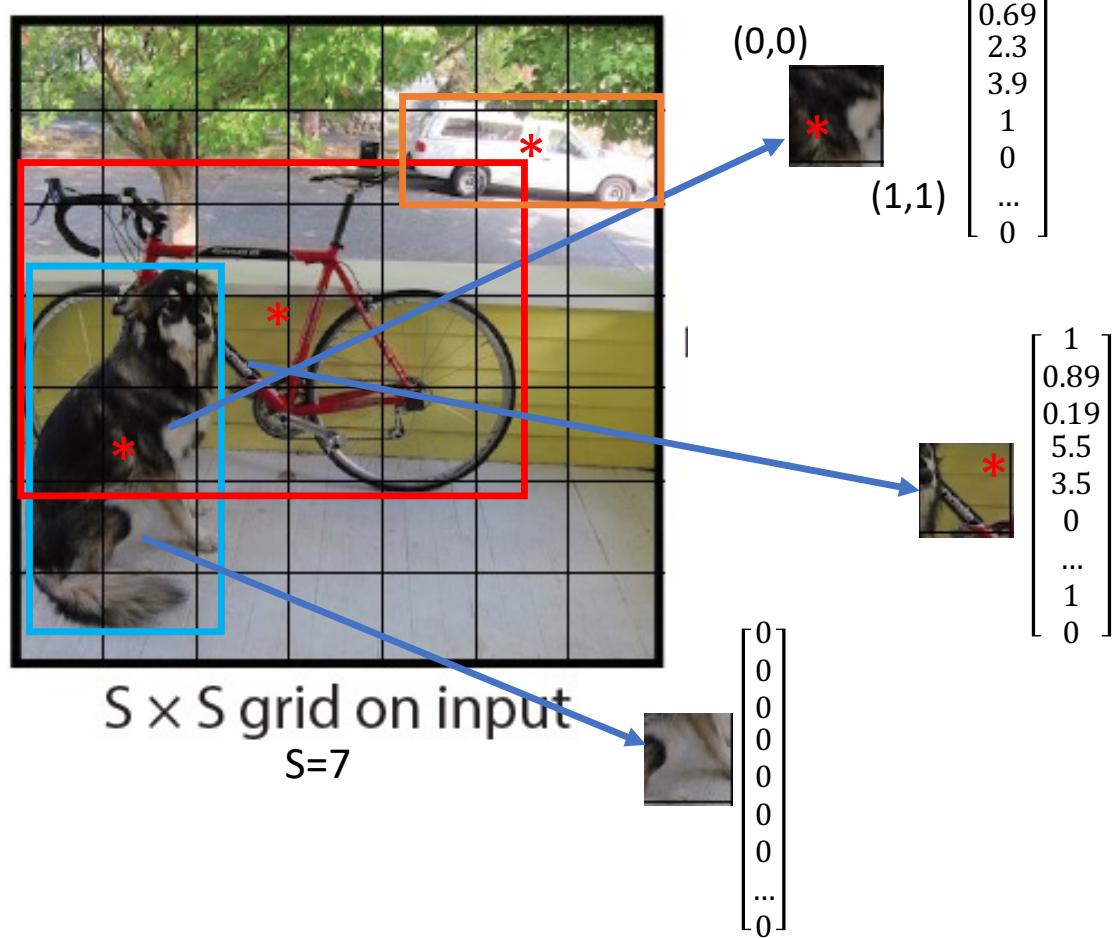
# Single object localization



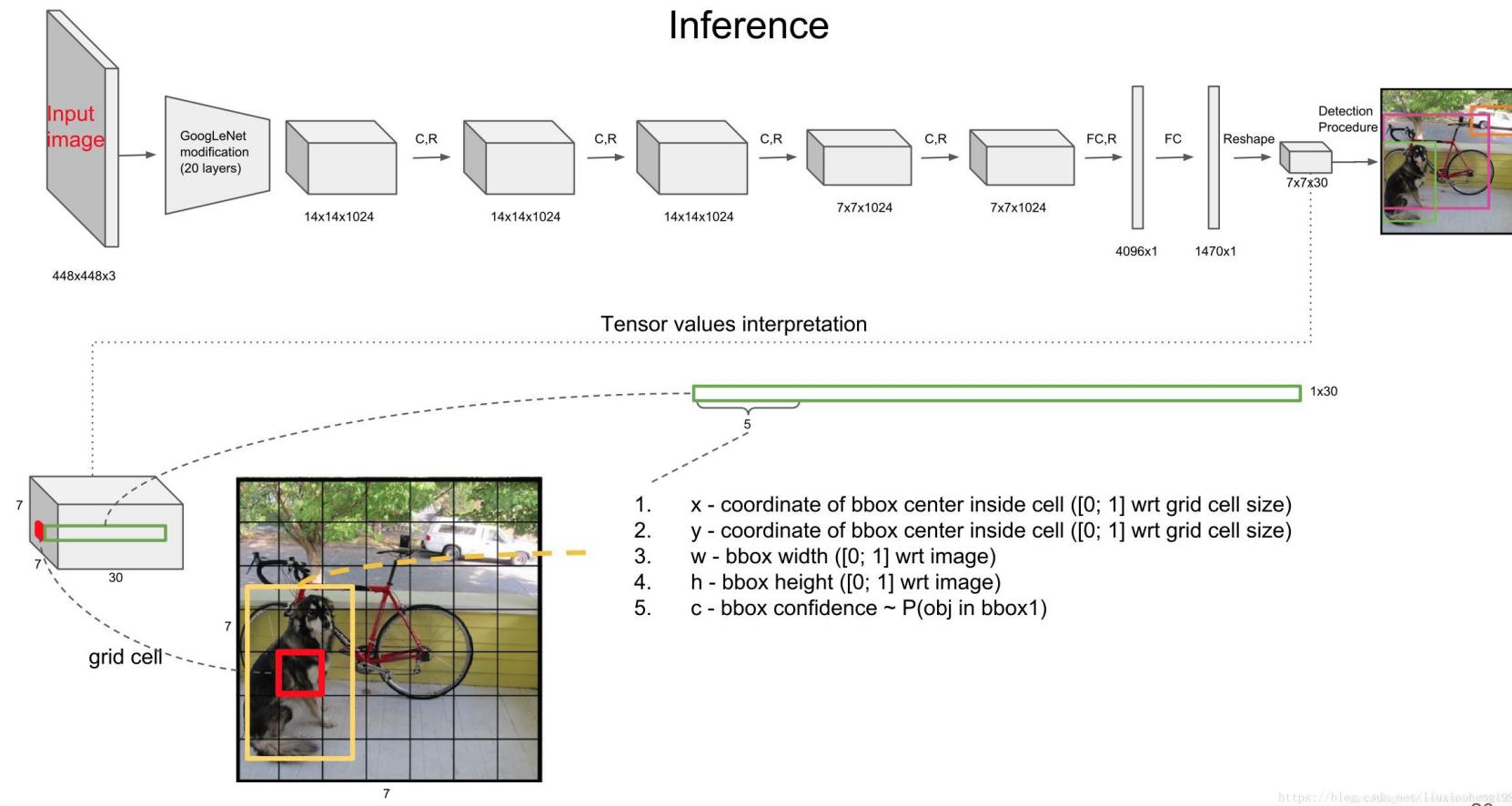
**Q:** But what about multiple objects in an image?



# YOLO basic idea

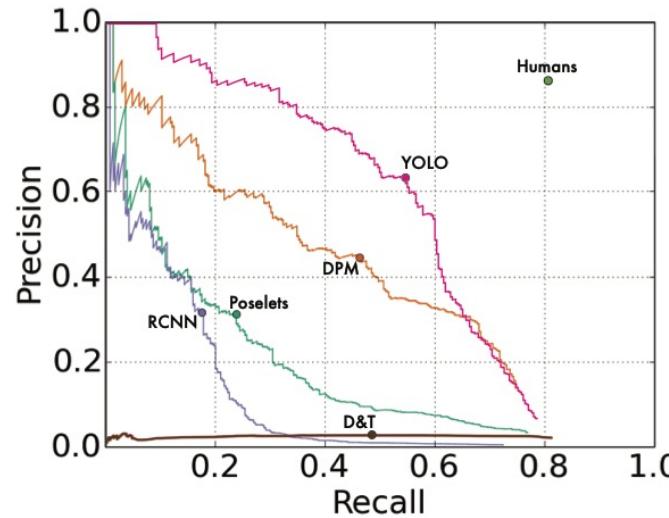


# YOLO basic idea

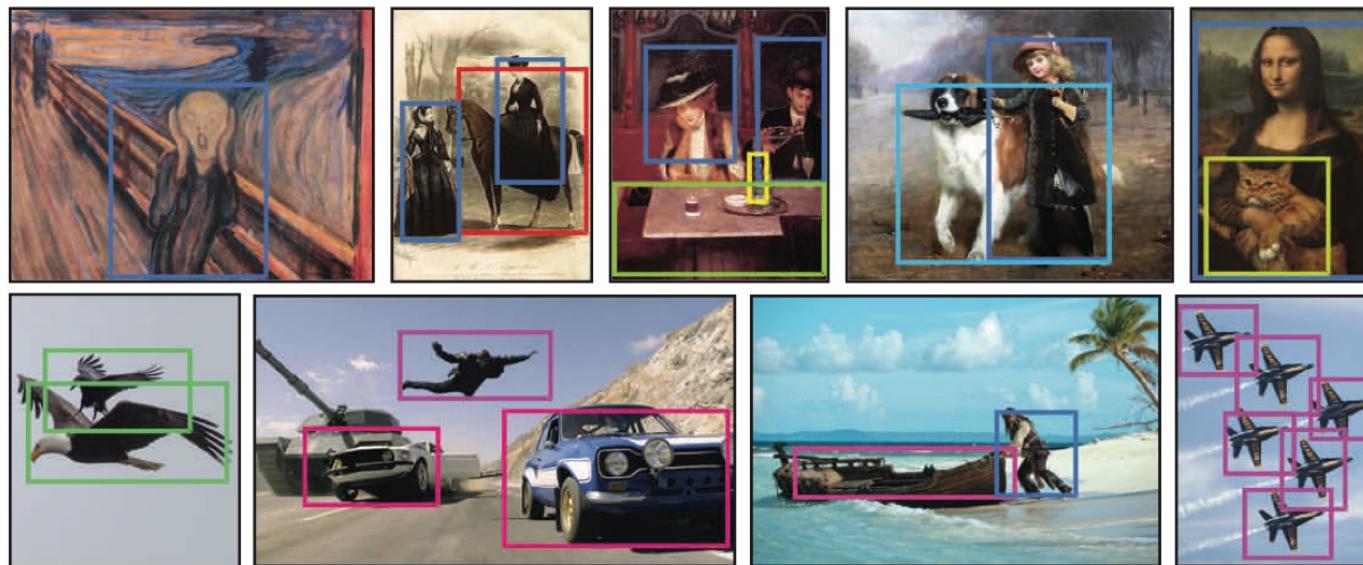


<https://blog.csdn.net/fuxuemingzi992>

Img from: <https://www.programmersought.com/article/7111753076/>



(a) Picasso Dataset precision-recall curves.



# Reference

- Video lecture by Codebasics: What is YOLO algorithm?:  
<https://www.youtube.com/watch?v=ag3DLKsl2vk>
- Blog: Introduction of YOLO network model in target detection:  
<https://www.programmersought.com/article/7111753076/>

# Week 04: Training in Practice

Machine Learning 2

Dr. Hongping Cai

# Topic 1: Optimization with Gradient Descent

# Batch Gradient Descent

Update the weight

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial J(W)}{\partial w_{ji}^{(l)}}$$

$$J(W) = \frac{1}{N} \sum_{n=1}^N L(f(x^{(n)}; W), y^{(n)})$$

Compute the gradients on the entire training set.  
This is called **Batch Gradient Descent**. It is very  
computationally extensive.

# Mini-Batch Gradient Descent

Update the weight

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial J(W)}{\partial w_{ji}^{(l)}}$$

$$J(W) = \frac{1}{N} \sum_{n=1}^N L(f(x^{(n)}; W), y^{(n)})$$



Randomly pick a batch of  $B$  training samples, compute the gradients over the batches. This is called **Mini-Batch Gradient Descent**.

$$J_B(W) = \frac{1}{B} \sum_{n=1}^B L(f(x^{(n)}; W), y^{(n)})$$

# Mini-Batch Gradient Descent

$$\frac{\partial J_B(W)}{\partial w_{ji}^{(l)}} \approx \frac{\partial J(W)}{\partial w_{ji}^{(l)}}$$

- It is a good approximation.
- Much faster convergence
- Can parallelize computation, achieve significant speed increases on GPU.

# Stochastic Gradient Descent (SGD)

Update the weight

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial J(W)}{\partial w_{ji}^{(l)}}$$

$$J(W) = \frac{1}{N} \sum_{n=1}^N L(f(x^{(n)}; W), y^{(n)})$$



Randomly pick only one training sample. This is called **Stochastic Gradient Descent (SGD)**.  
Easy to compute but very noisy (stochastic).

$$J_n(W) = L(f(x^{(n)}; W), y^{(n)})$$

# Three Gradient Descent Variants

- Batch Gradient Descent
- Mini-Batch Gradient Descent
- Stochastic Gradient Descent (SGD)

Is typically the choice.

**Note** that people often use term “SGD” refers to the Mini-batch Gradient Descent.

```
model.compile(loss='categorical_crossentropy',
              optimizer= 'SGD',
              metrics=[ 'accuracy'])
history = model.fit(trainX, trainy, epochs=150, batch_size=64, validation_split=0.2)
```

One **epoch**: one learning cycle through the entire training data.

# Reference for Topic 1

- Video lecture by Alexander Amini: MIT course on deep learning,  
<https://www.youtube.com/watch?v=njKP3FqW3Sk>
- <https://cs231n.github.io/optimization-1/>
- <https://ruder.io/optimizing-gradient-descent/>

# Topic 2:

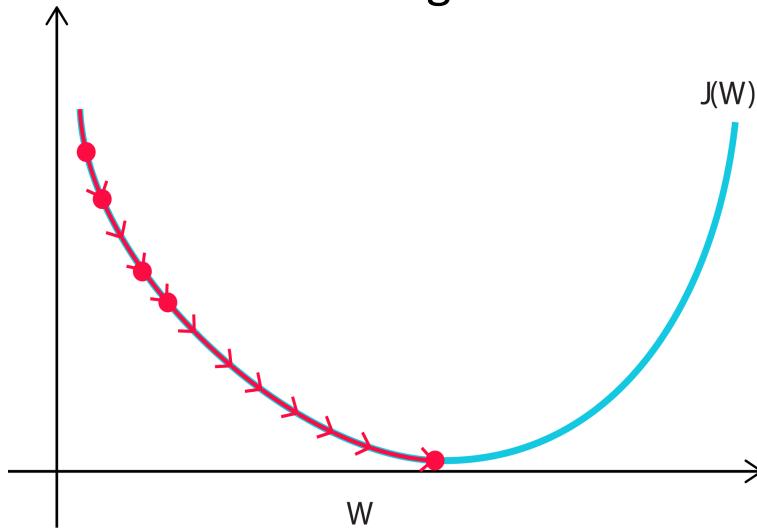
# Learning Rate

# Learning Rate

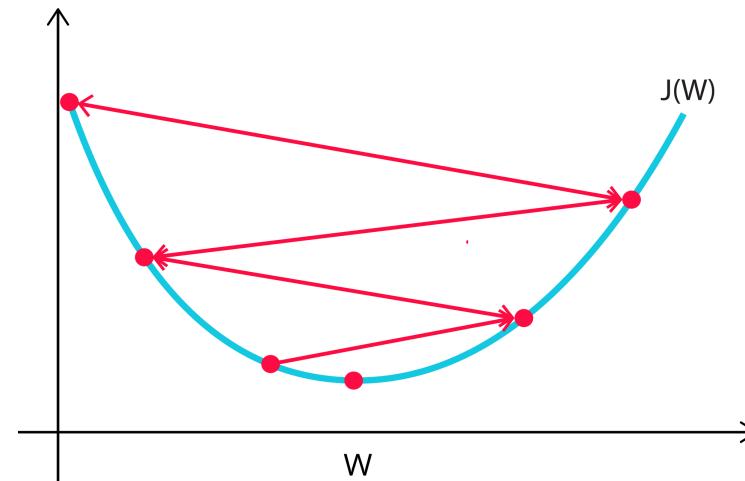
Update the weight

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial J(W)}{\partial w_{ji}^{(l)}}$$

**Too small:** requires many updates before reaching the minimum.



**Too large:** overshoot and even diverge.

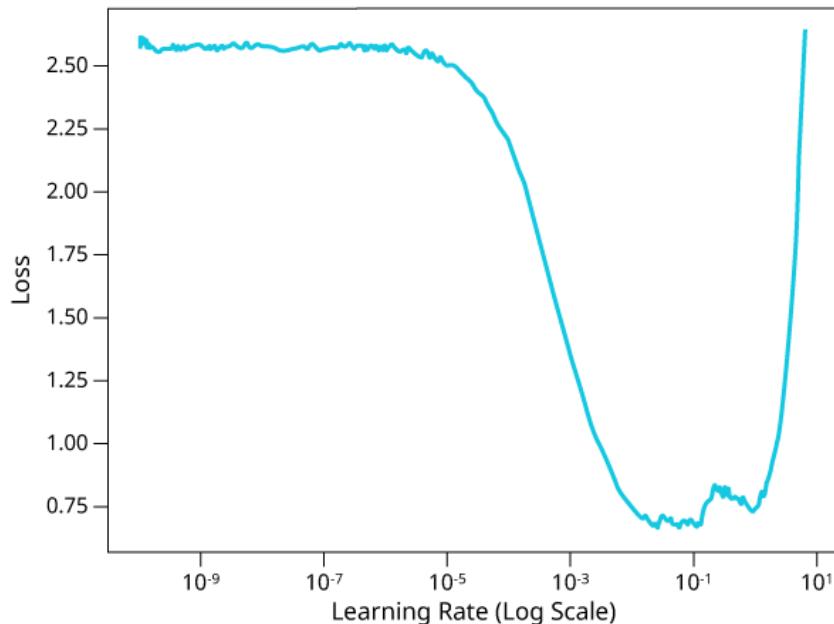


# Learning Rate

Update the weight

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial J(W)}{\partial w_{ji}^{(l)}}$$

**Q:** How to find the proper learning rate?



Option 1: Try different learning rate

$$\eta = 10, 1, 10^{-1}, 10^{-2}, 10^{-3}, \dots$$

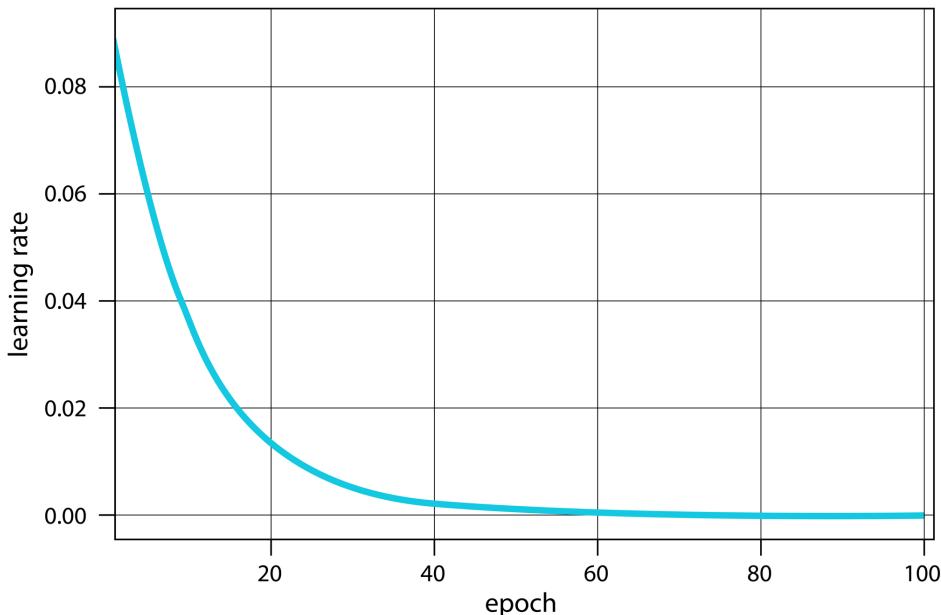
- Train the model for a few hundred iterations with each learning rate. Then plot the loss varied with learning rate.

# Learning Rate

Update the weight

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial J(W)}{\partial w_{ji}^{(l)}}$$

Learning rate



**Q:** How to find the proper learning rate?

## Option 2: Learning rate schedule

- Decrease the learning rate during training according to a pre-defined schedule.
  - Time-based decay
  - Step decay
  - Exponential decay

$$\eta = \eta_0 \cdot e^{-kt}$$

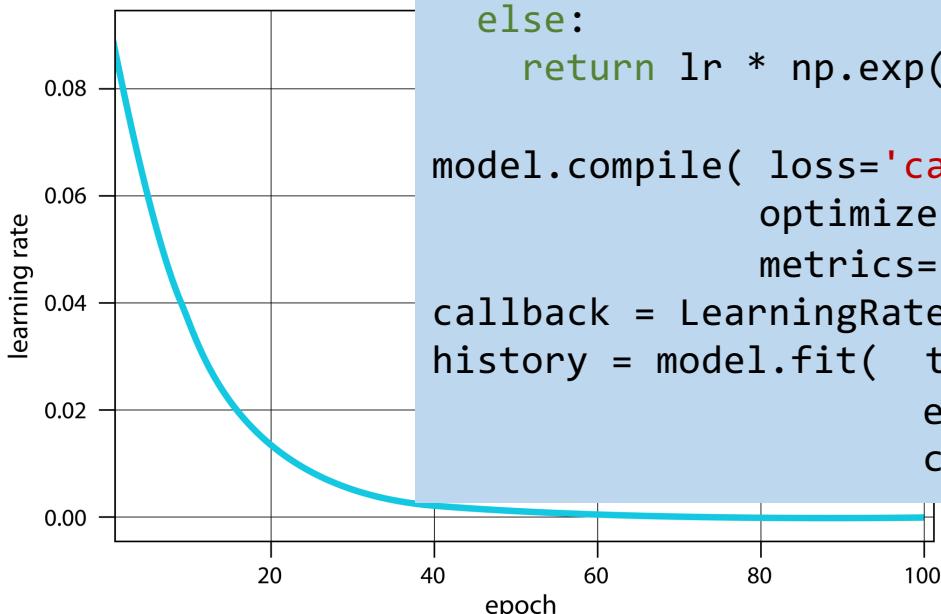
$k$  is usually set to 0.1  
 $t$  is the iteration number

# Learning Rate

```
from keras.callbacks import LearningRateScheduler
import numpy as np

def scheduler(epoch, lr): # define a scheduler
    if epoch < 10:
        return lr
    else:
        return lr * np.exp(-0.1*epoch)

model.compile( loss='categorical_crossentropy',
                optimizer= 'SGD',
                metrics=['accuracy'])
callback = LearningRateScheduler(scheduler)
history = model.fit( trainX, trainy,
                    epochs=150, batch_size=32,
                    callbacks=[callback])
```



to find the proper  
rate?

schedule  
rate during training  
ned schedule.

$k$  is usually set to 0.1  
 $t$  is the iteration number

# Learning Rate

Update the weight

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial J(W)}{\partial w_{ji}^{(l)}}$$

**Q:** How to find the proper learning rate?

## Option 3: Adaptive learning rate

```
model.compile( loss='categorical_crossentropy',
                optimizer= 'RMSprop',
                metrics=[ 'accuracy'])
```

OR:

```
opt = keras.optimizers.RMSprop(lr=0.001, rho=0.9)
model.compile( loss='categorical_crossentropy',
                optimizer=opt,
                metrics=[ 'accuracy'])
```

- AdaGrad optimizer
- Adadelta optimizer
- RMSProp optimizer
- Adam optimizer
- ...

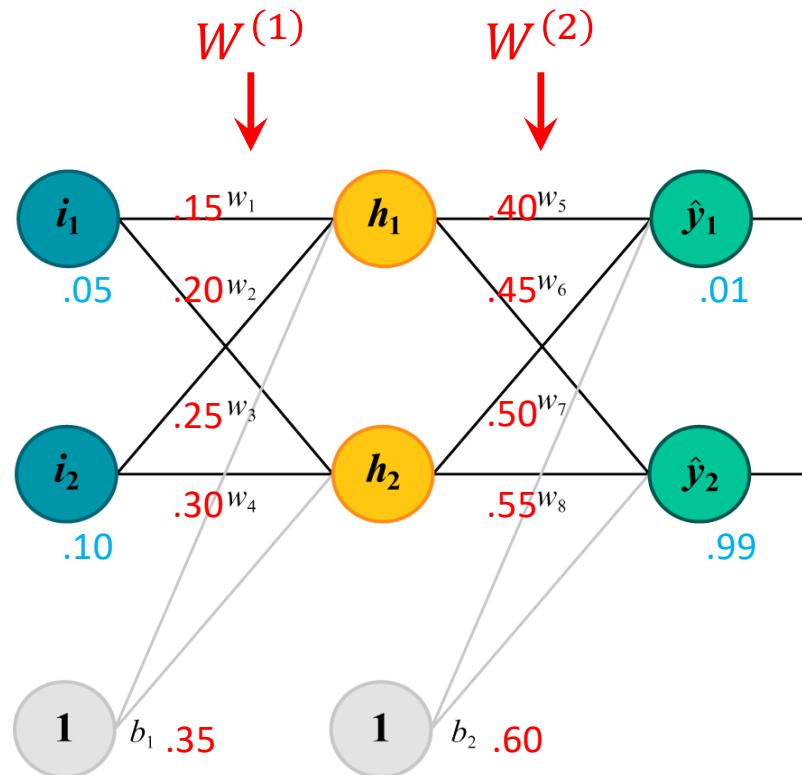
Usually a good choice

# Reference for Topic 2

- Book: Aurelien Geron. Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly. 2019.
- <https://www.allaboutcircuits.com/technical-articles/understanding-learning-rate-in-neural-networks/>
- <https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1>

# Topic 3: Vanishing Gradient Problem

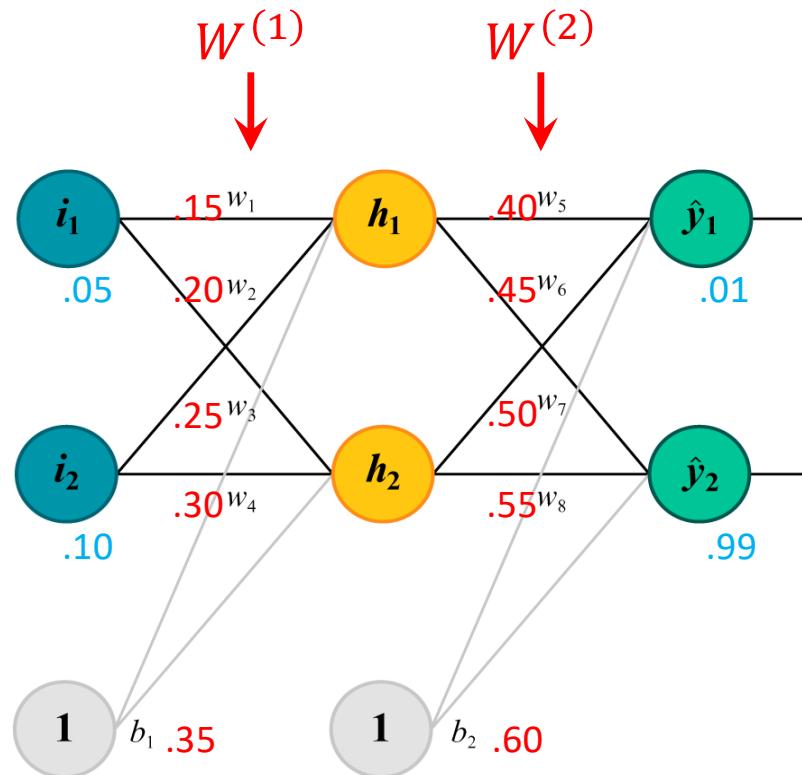
# Vanishing gradient problem



	Iteration 0	Iteration 1	Iteration 2
$\frac{\partial J(W)}{\partial W^{(2)}}$	$dJ_{dw5} = +0.082167$ , $dJ_{dw6} = +0.083706$ , $dJ_{dw7} = +0.084819$ , $dJ_{dw8} = +0.082668$ , $dJ_{dw9} = +0.084219$ , $dJ_{dw10} = +0.085340$ , $dJ_{dw11} = -0.022603$ , $dJ_{dw12} = -0.023732$ , $dJ_{dw13} = -0.024896$ , $dJ_{dw14} = -0.022740$ , $dJ_{dw15} = -0.023877$ , $dJ_{dw16} = -0.025049$ ,		
$\frac{\partial J(W)}{\partial W^{(1)}}$	$dJ_{dw1} = +0.000439$ , $dJ_{dw2} = +0.000366$ , $dJ_{dw3} = +0.000285$ , $dJ_{dw4} = +0.000877$ , $dJ_{dw5} = +0.000733$ , $dJ_{dw6} = +0.000570$ , $dJ_{dw7} = +0.000498$ , $dJ_{dw8} = +0.000426$ , $dJ_{dw9} = +0.000345$ , $dJ_{dw10} = +0.000995$ , $dJ_{dw11} = +0.000852$ , $dJ_{dw12} = +0.000689$ ,		

**Q:** The gradients on the lower layer are much smaller than those on the higher layer. What effect?

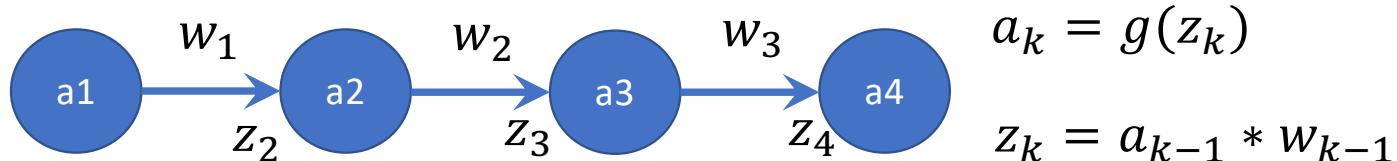
# Vanishing gradient problem



- This is called **vanishing gradient** problem: gradients get smaller and smaller as the backpropagation progresses down to the lower layers.
  - The lower layers' weights are updated very little. Therefore, the lower layers contribute very little to reduce the total loss.
- $$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial J(W)}{\partial w_{ji}^{(l)}} \approx 0$$

# Vanishing gradient problem

- Why?



$$\frac{\partial J(W)}{\partial w_3} = \frac{\partial J(W)}{\partial a_4} \cdot \frac{\partial a_4}{\partial z_4} \cdot \frac{\partial z_4}{\partial w_3} = \frac{\partial J(W)}{\partial a_4} \cdot g'(z_4) \cdot a_3$$

$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial a_4} \cdot \frac{\partial a_4}{\partial z_4} \cdot \frac{\partial z_4}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_2} = \frac{\partial J(W)}{\partial a_4} \cdot g'(z_4) \cdot w_3 \cdot g'(z_3) \cdot a_2$$

$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial a_4} \cdot \frac{\partial a_4}{\partial z_4} \cdot \frac{\partial z_4}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_1} = \frac{\partial J(W)}{\partial a_4} \cdot g'(z_4) \cdot w_3 \cdot g'(z_3) \cdot w_2 \cdot g'(z_2) \cdot a_1$$

# Vanishing gradient problem

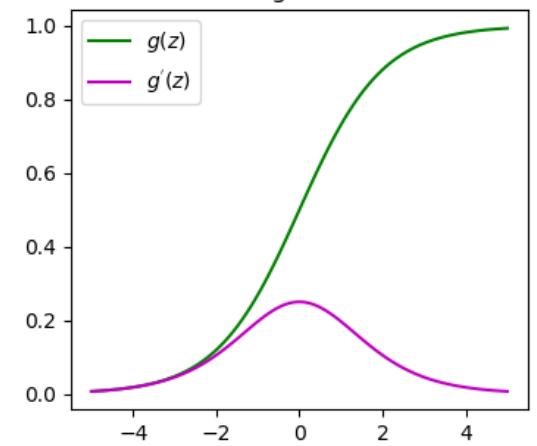
- Why?

Standard weight initialization approach is using Gaussian distribution  $\mu = 0, \sigma = 1$ .  
 Therefore,  $|w_k| \leq 1$  (mostly)

$$\frac{\partial J(W)}{\partial w_3} = \frac{\partial J(W)}{\partial a_4} \cdot \frac{\partial a_4}{\partial z_4} \cdot \frac{\partial z_4}{\partial w_3} = \frac{\partial J(W)}{\partial a_4} \cdot g'(z_4) \cdot a_3 \xrightarrow{\text{dashed arrow}} \leq 0.25$$

$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial a_4} \cdot \frac{\partial a_4}{\partial z_4} \cdot \frac{\partial z_4}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_2} = \frac{\partial J(W)}{\partial a_4} \cdot g'(z_4) \cdot w_3 \cdot g'(z_3) \cdot a_2 \xrightarrow{\text{dashed arrow}} \leq 0.25$$

$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial a_4} \cdot \frac{\partial a_4}{\partial z_4} \cdot \frac{\partial z_4}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_1} = \frac{\partial J(W)}{\partial a_4} \cdot \underbrace{g'(z_4)}_{\leq 0.25} \cdot \underbrace{w_3 \cdot g'(z_3)}_{\leq 0.25} \cdot \underbrace{w_2 \cdot g'(z_2)}_{\leq 0.25} \cdot a_1$$



$$g'(z_k) \leq 0.25$$

# Vanishing gradient problem

- How to avoid it?

## Activation function

ReLU, instead of sigmoid or tanh

## Weight initialization

Glorot initialization (or Xavier initialization)

## Layer restriction

Batch normalization

## Network structure

Residual networks  
(ResNet, DenseNet, etc)

# Deep Residual Learning (ResNet)

- **Q:** Will stacking more layers always give better performance?
- **A:** Not always.

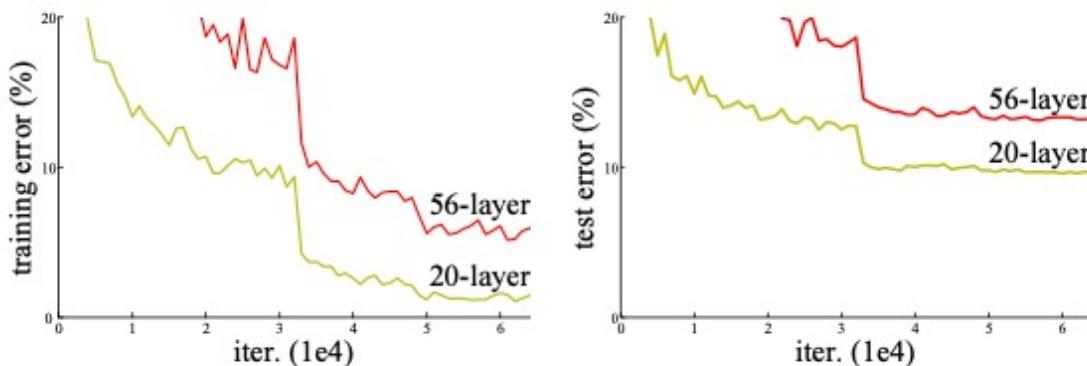
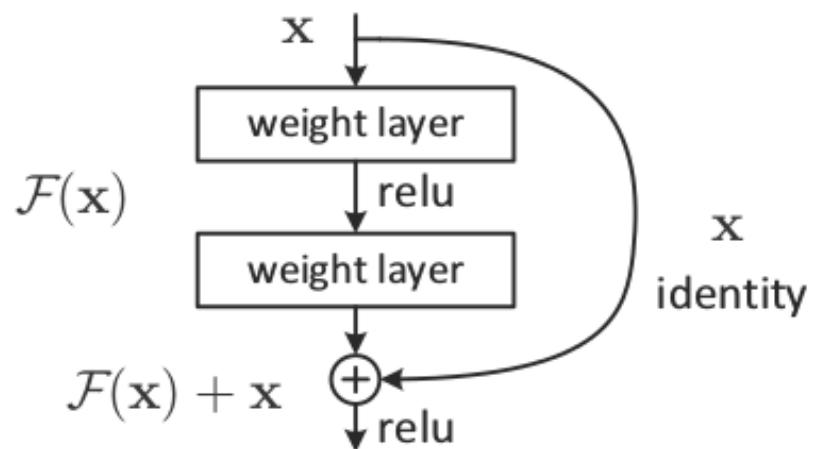


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

# ResNet<sup>[1]</sup>

- Winner in the ILSVRC 2015 classification competition (3.57% top-5 error) with 152 layers
- Main technique: **skip connection**

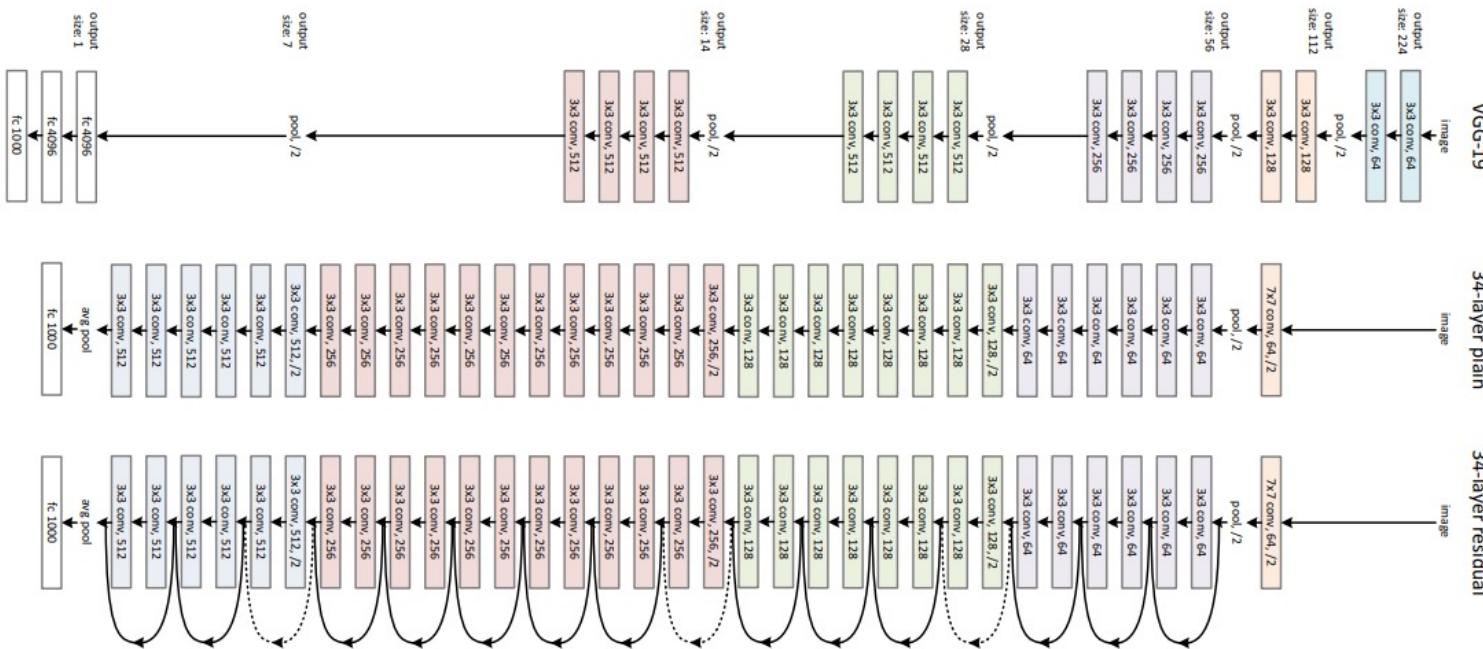


Desired underlying mapping:  $\mathcal{H}(x)$

We fit the **residual mapping**:  $\mathcal{F}(x) := \mathcal{H}(x) - x$

That's why it was called **Deep Residual Learning**

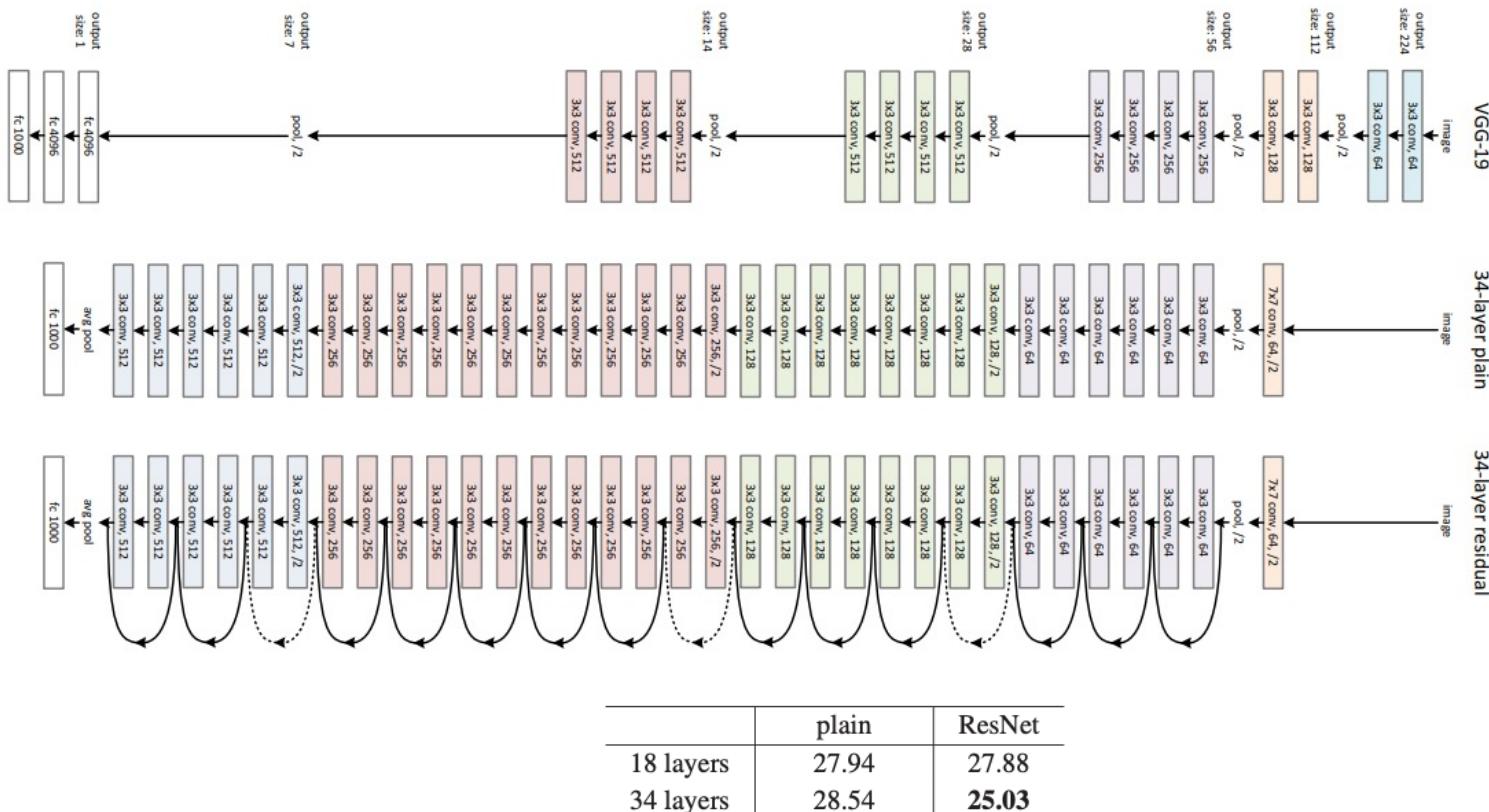
# ResNet



- If any layer hurts the performance of architecture then it will be skipped. This results in training a very deep neural network.
- No extra parameter nor computational complexity

Img from: <https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/>

# ResNet



	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	<b>25.03</b>

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

Img from: <https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/>

# Reference for Topic 3

- Book: Aurelien Geron. Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly. 2019.
- <http://neuralnetworksanddeeplearning.com/chap5.html>
- <https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484>

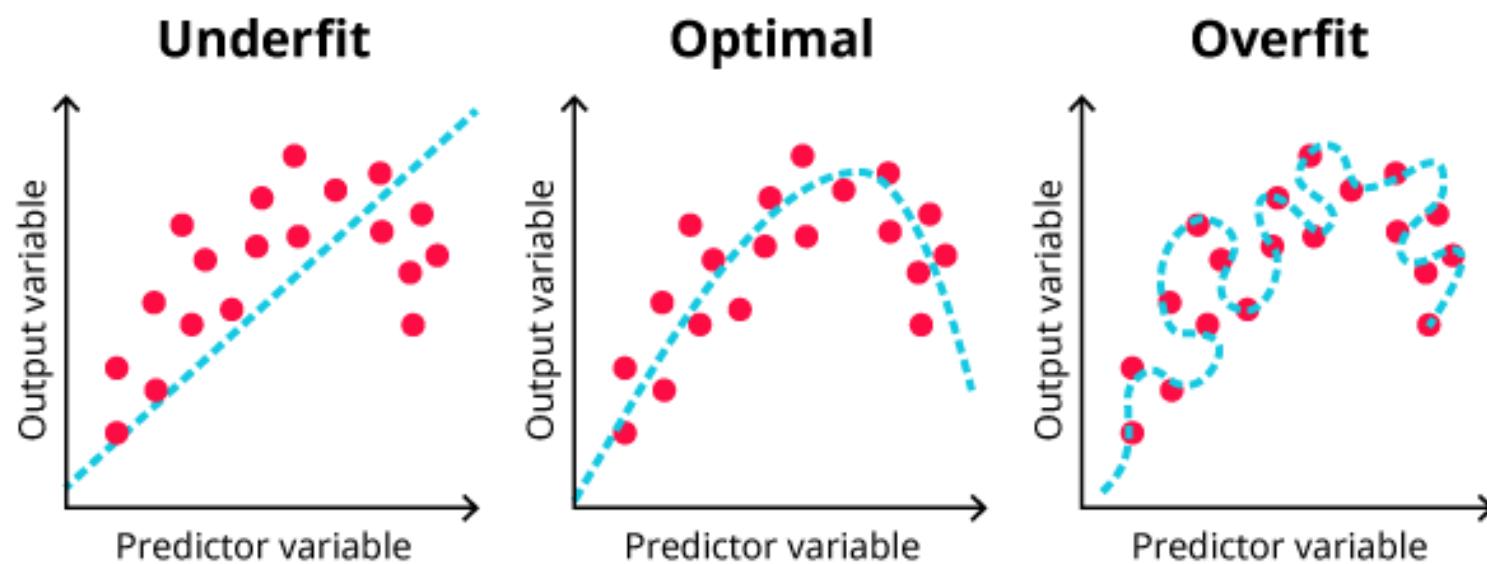
# Topic 4:

# Overfitting Problem

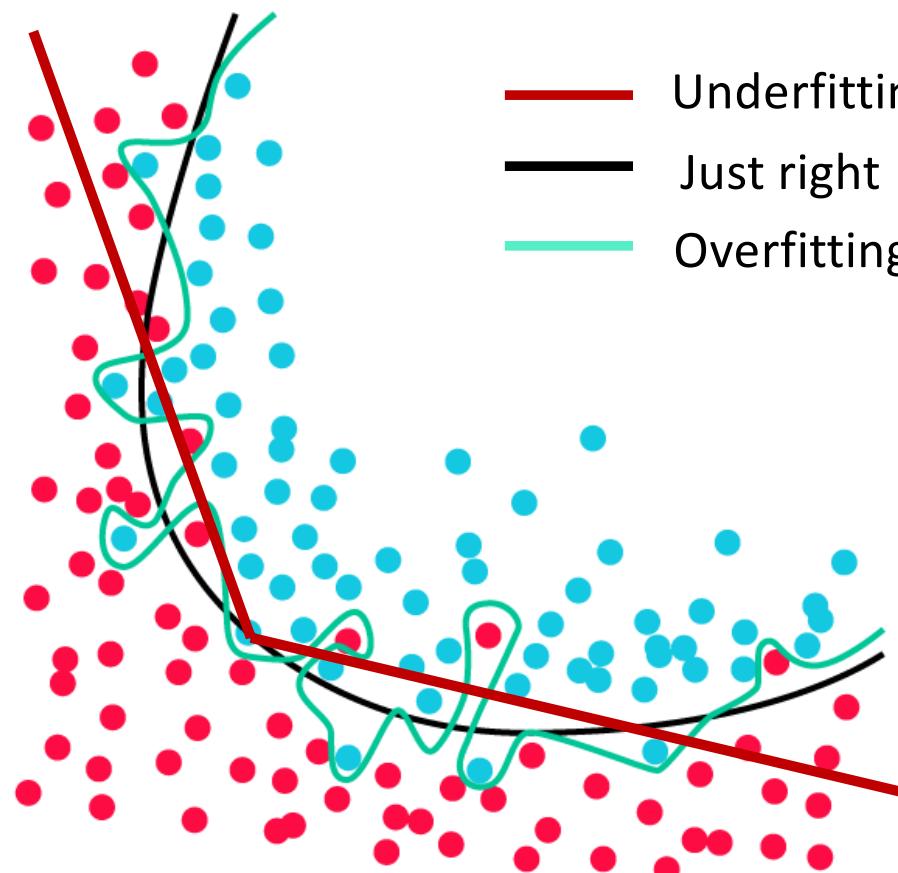
# Underfitting and Overfitting

- Too simple
- Does not fully learn the data

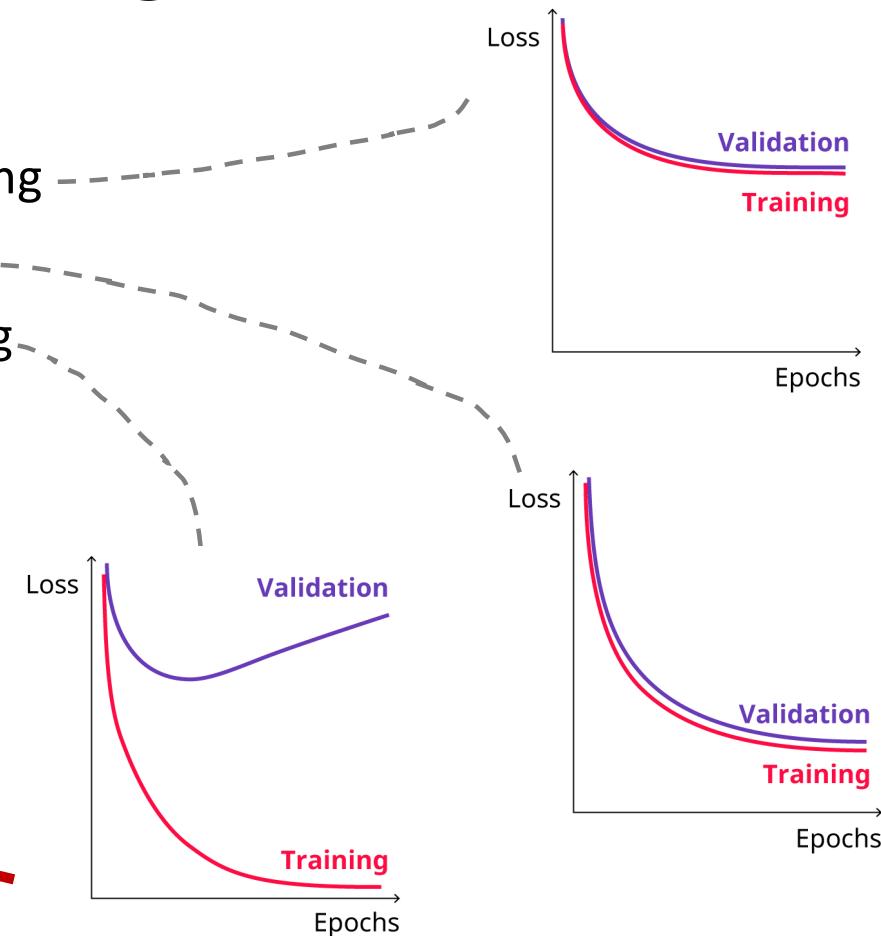
- Too complex
- Fits the noise in the training data
- Does not generalize well



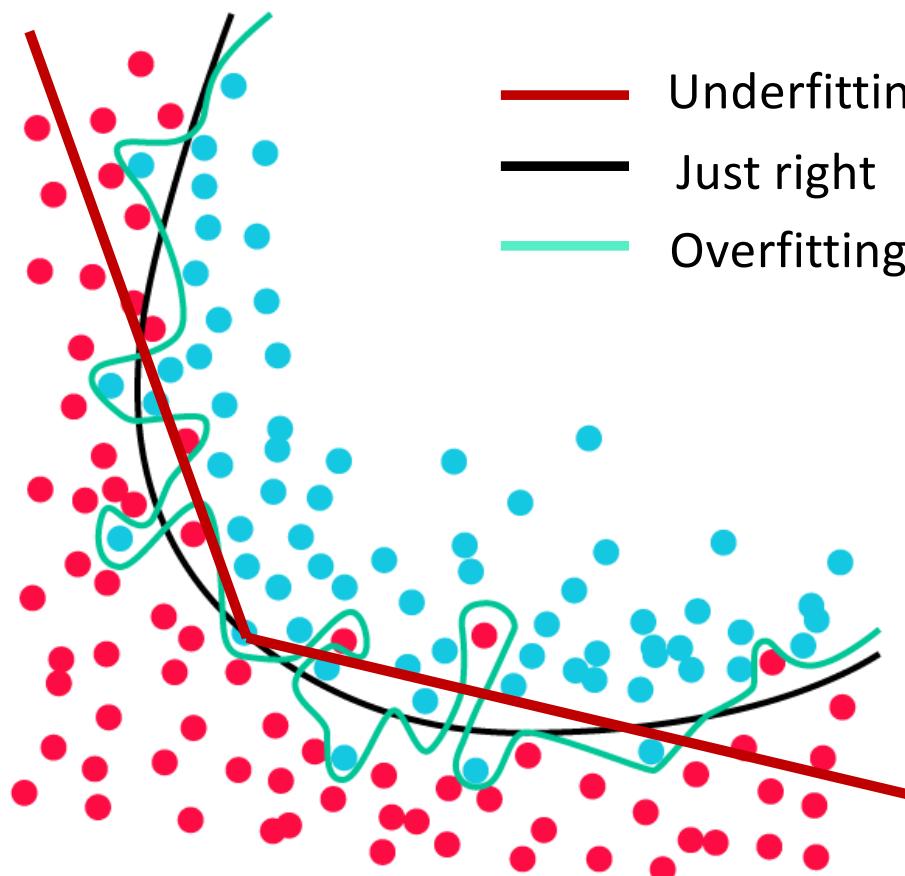
# How to Identify Underfitting and Overfitting



— Underfitting  
 — Just right  
 — Overfitting



# How to Prevent Underfitting and Overfitting

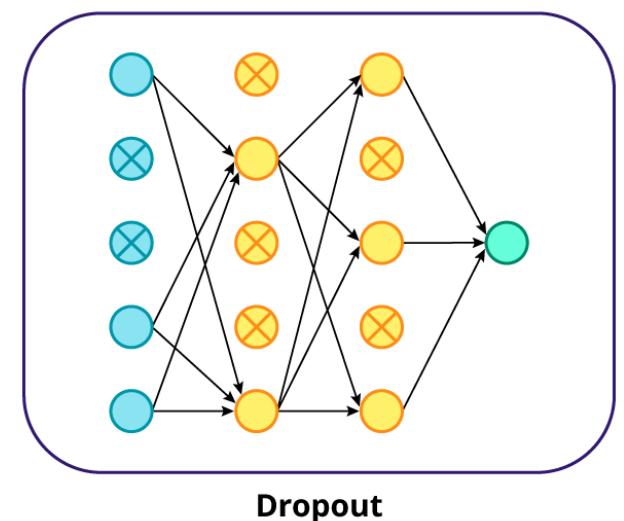
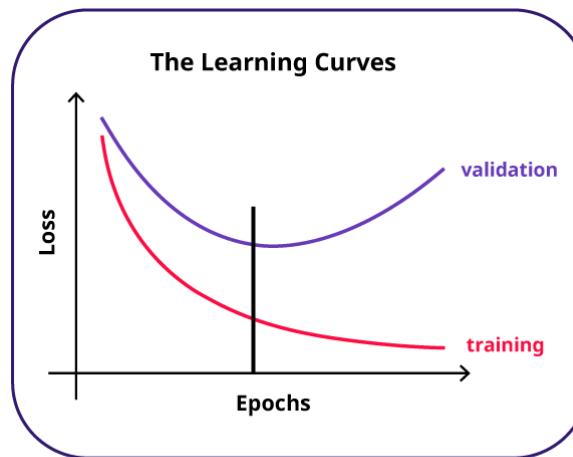
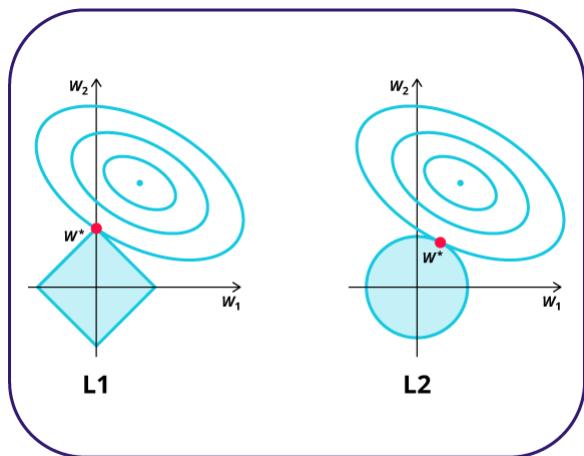


— Underfitting  
— Just right  
— Overfitting

- Complexify model
  - Add more nodes/layers
  - Train longer
- 
- More training data
    - Data augmentation
  - Simplify model
  - Regularization

# Regularization

- **Regularization** is a technique which constraints the optimization problem to discourage complex model.
- Regularization helps the model generalize better on unseen data.



# Weight Regularization

$$J(W) = \frac{1}{N} \sum_{n=1}^N L(f(x^{(n)}; W), y^{(n)})$$



**L2 Regularization:**  $J(W) = \frac{1}{N} \sum_{n=1}^N L(f(x^{(n)}; W), y^{(n)}) + \lambda \sum_k w_k^2$

**L1 Regularization:**  $J(W) = \frac{1}{N} \sum_{n=1}^N L(f(x^{(n)}; W), y^{(n)}) + \lambda \sum_k |w_k|$

# Weight Regularization

```
from keras.regularizers import l2

model = Sequential()
model.add(Dense(128, kernel_regularizer=l2(0.01), input_dim=8, activation='relu'))
model.add(Dense(64, kernel_regularizer=l2(0.01), activation='relu'))
model.add(Dense(8, kernel_regularizer=l2(0.01), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

$$\lambda = 0.01$$

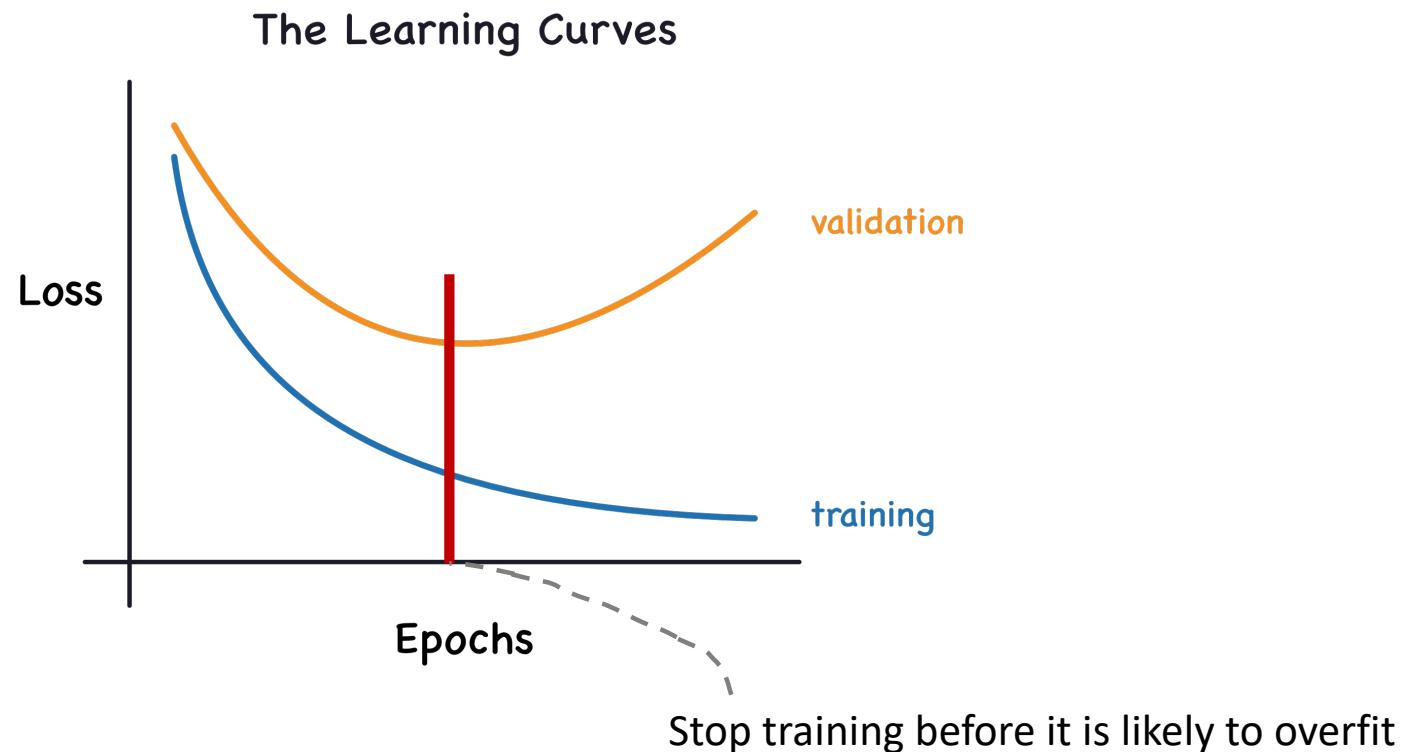
## L2 Regularization:

- Is able to learn complex data patterns
- Is more commonly used
- Is not robust to outliers

## L1 Regularization:

- Generates sparse models
- Is robust to outliers

# Early Stopping



# Early Stopping

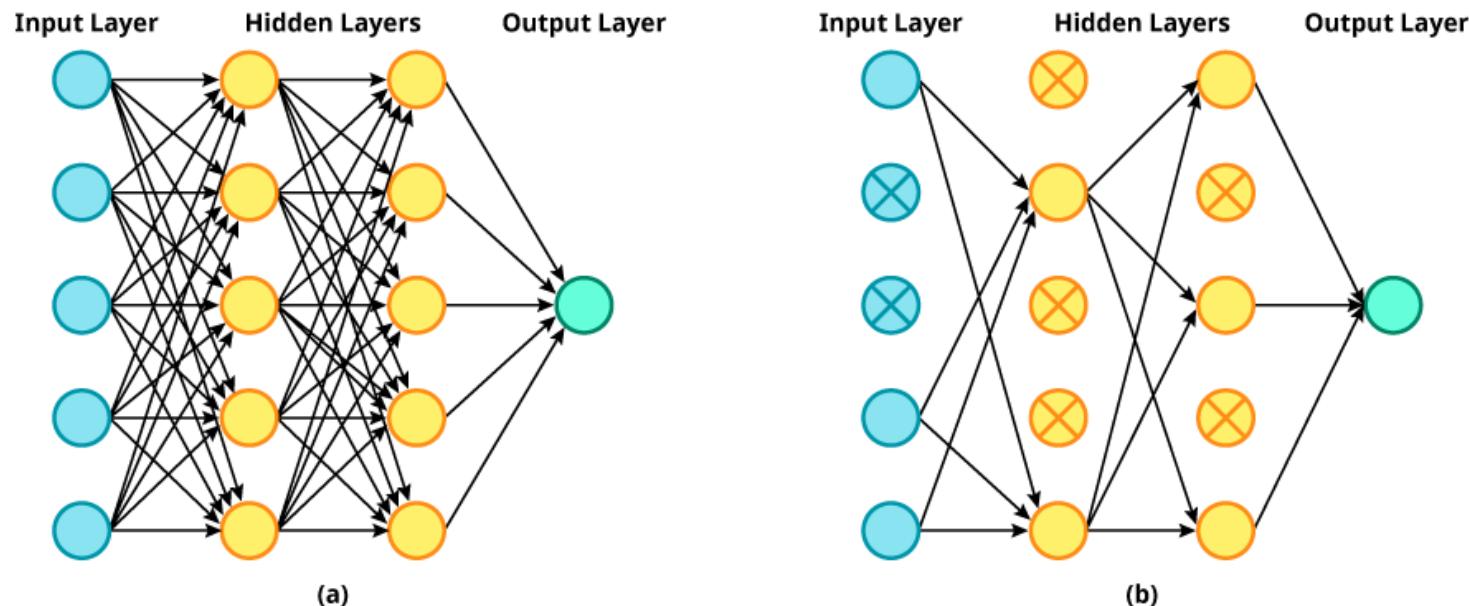
If after 5 epochs there is no reduce of validation loss (with a tolerance of 0.001), the training will be stopped, the best weight for the lowest loss is kept.

```
from keras.callbacks import EarlyStopping

es_callback = EarlyStopping( monitor='val_loss',
                             min_delta = 0.001,
                             patience=5,
                             restore_best_weights=True)
model.fit(trainX, trainy, callbacks=[es_callback], epochs=1000, validation_split=0.3)
```

# Dropout

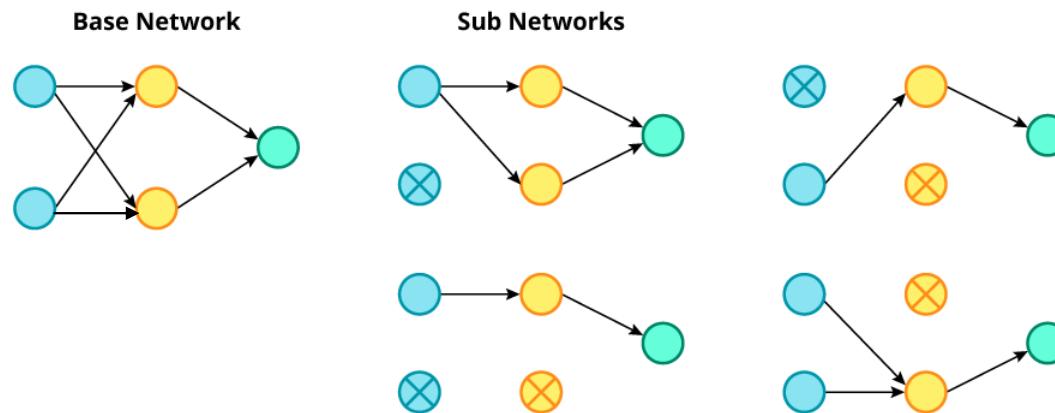
- **Dropout** is to randomly remove some hidden neurons along with their connections during training.



**Q:** Is it equivalent to using less nodes in each layer?

# Dropout

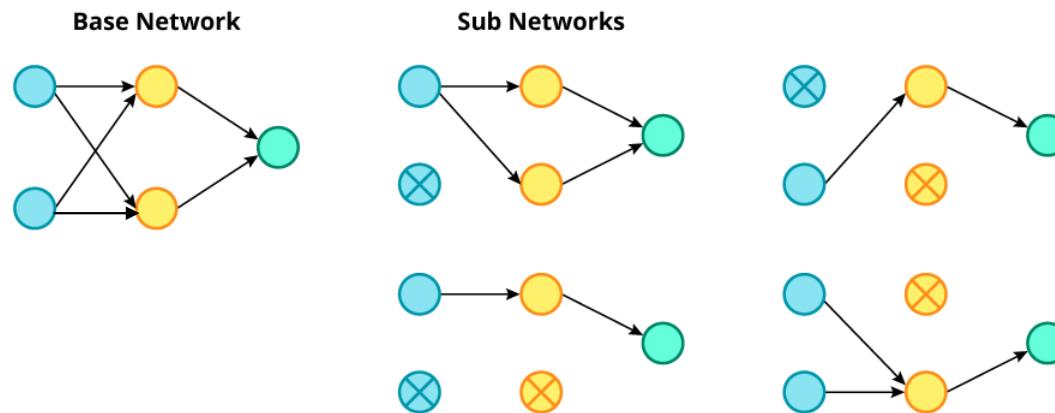
**A:** NO. Because the removed nodes are different in each training iteration.



- Dropout is a kind of **ensemble of sub-networks** with shared parameters.
- Force the network **not to rely on any particular connections** of neurons.

# Dropout

**A:** NO. Because the removed nodes are different in each training iteration.



**Q:** Should dropout process be also applied in the testing stage?

**A:** No.

# Dropout

Randomly drop out 40% of the 128 nodes

```
from keras.layers import Dropout

model = Sequential()
model.add(Dense(128, input_dim=8, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(8, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(1, activation='sigmoid'))
```

In practice, you can usually apply dropout after all the  
**dense layers** excluding the output layer.

# Reference for Topic 4

- Video lecture by Alexander Amini: MIT course on deep learning,  
<https://www.youtube.com/watch?v=njKP3FqW3Sk>
- <https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html>
- <https://medium.com/@jennifer.arty/regularization-methods-to-prevent-overfitting-in-neural-networks-1a79b5e3081f>
- <https://www.kaggle.com/ryanholbrook/dropout-and-batch-normalization>

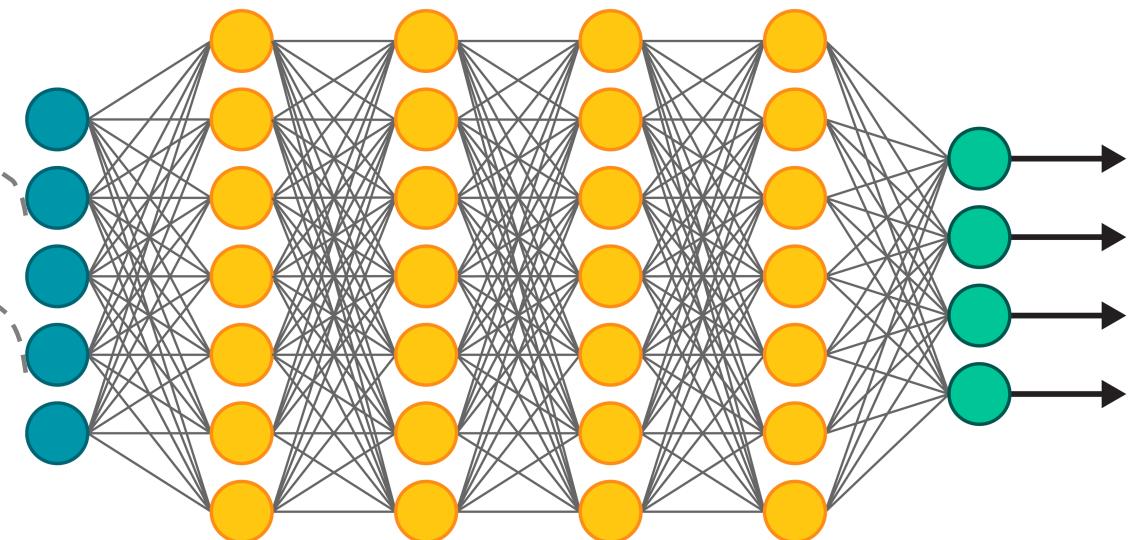
# Topic 5:

# Batch Normalization

# Batch Normalization

x2 (house area): 78.5, 200.2, 12, 380.4, 60, ...

x9 (No. of bedroom): 1, 3, 2, 7, 5, 2, ...



**Q:** What should we do to train the network?

**A:** Standardize the input data: mean 0 and 1 std.

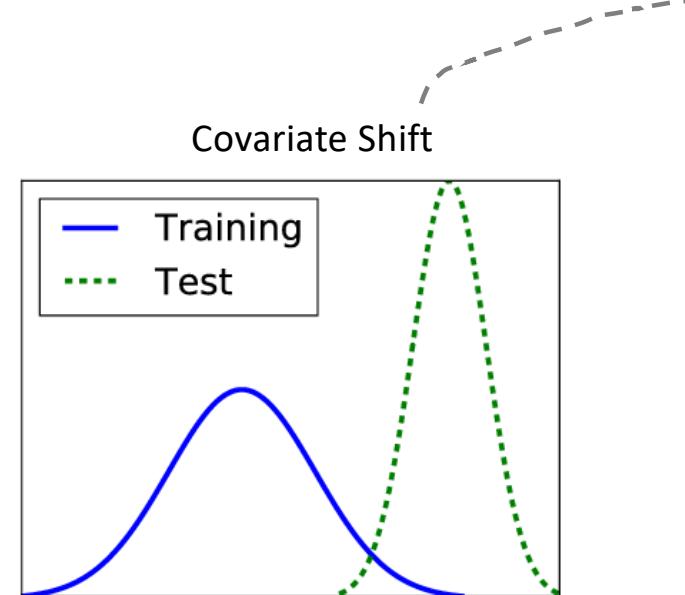
**Q:** Actually, all hidden layers have the same problem. How to solve it?

**A: Batch Normalization.**

# Intuition for Batch Normalization

- Limit the **internal covariate shift**, allow more stable distribution of input for the internal layers.

The change in the input distribution to a learning algorithm.



# Batch Normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

- **Batch normalization** is a technique that standardizes the inputs to a layer for each mini-batch, then rescale and offsets them.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

# Before or After Activation Function?

*“The goal of Batch Normalization is to achieve a stable distribution of activation values throughout training, and in our experiments we apply it **before** the nonlinearity .” -- [S. Loffe, 2015]*

```
from keras.layers import BatchNormalization, Activation

model = Sequential()
model.add(Dense(128, input_dim=8))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(64))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(8))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(1, activation='sigmoid'))
```

However, some others observed better performance with batch normalization **after** the activations.

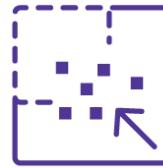
# Benefits



The networks are much less sensitive to the **weight initialization**.



**Larger learning rates** could be used, significantly speeding up the learning process



The **vanishing gradients** problem is strongly reduced.



Act like a **regularizer**, reducing the need for other regularizations(such as dropout)

It can be used with most network types, such as Multilayer Perceptrons (MLPs), Convolutional Neural Networks(CNNs) and Recurrent Neural Networks(RNNs).

# Reference for Topic 5

- Book: Aurelien Geron. Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly. 2019.
- <https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c>
- <https://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/>
- <https://paperswithcode.com/method/batch-normalization>

# Topic 6:

# Data Augmentation

# Training CNNs

- CNN is trained in the same way as MLP: gradient descent with backpropagation.
- In practice, it is relatively rare to have sufficient training data:
  - Data collection and annotation is expensive.
  - Sometimes near-impossible (e.g., medical images of a rare disease)

# Training CNNs

“Deep learning is powerful only when you have a huge amount of data.”

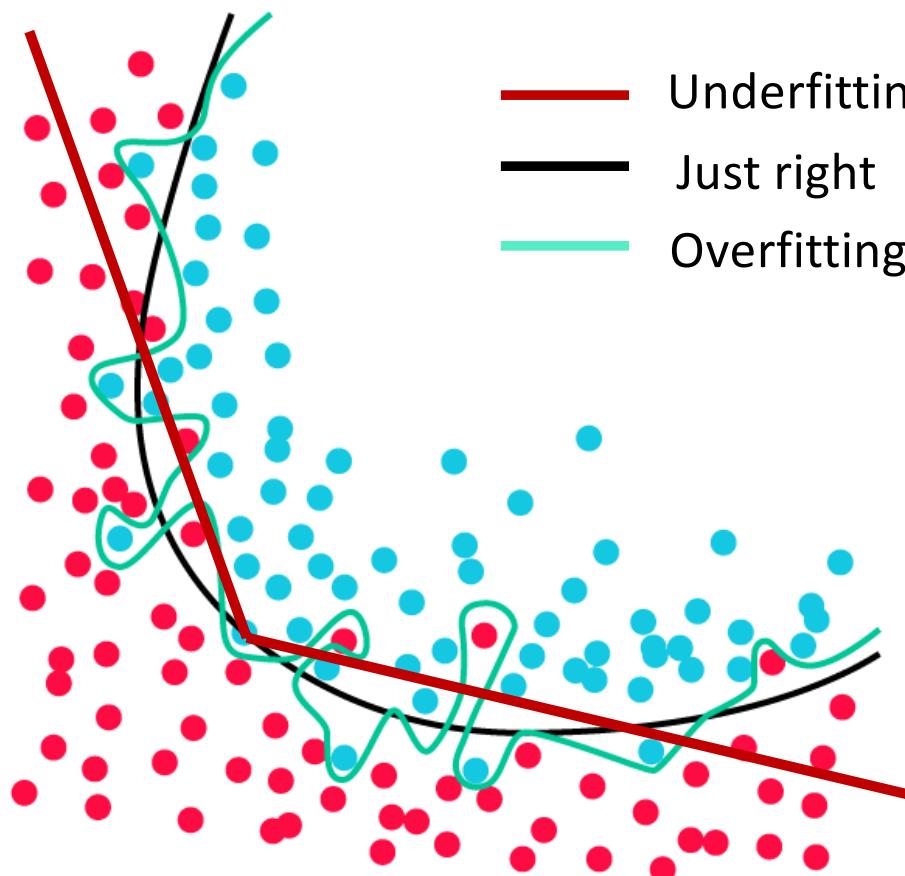
**BUSTED**

“You can only use small CNNs if your dataset is small.”

**BUSTED**

- ✓ **Data augmentation:** Expanding the training set
- ✓ **Transfer learning:** Making Less Data Cool Again

# How to Prevent Underfitting and Overfitting



— Underfitting  
— Just right  
— Overfitting

- Complexify model
- Add more nodes/layers
- Train longer
- More training data
  - Data augmentation
- Simplify model
- Regularization

# Data Augmentation

- **Data augmentation** is to generate synthetic data from existing training examples, by *augmenting* the samples via a number of random transformations.

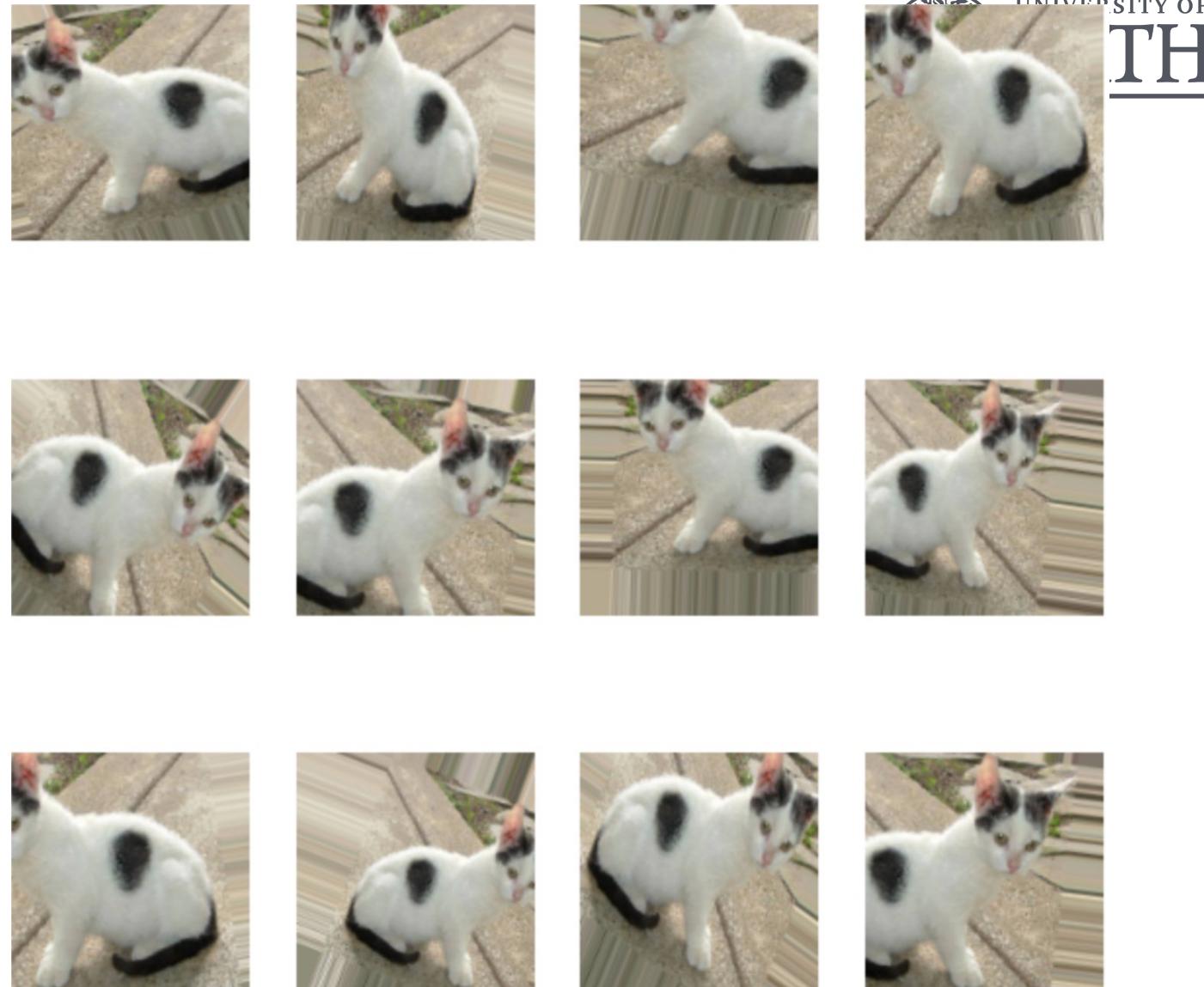
```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = 'nearest')

test_datagen = ImageDataGenerator(rescale=1./255)
```

The validation and test data shouldn't be augmented.

# Augmented examples



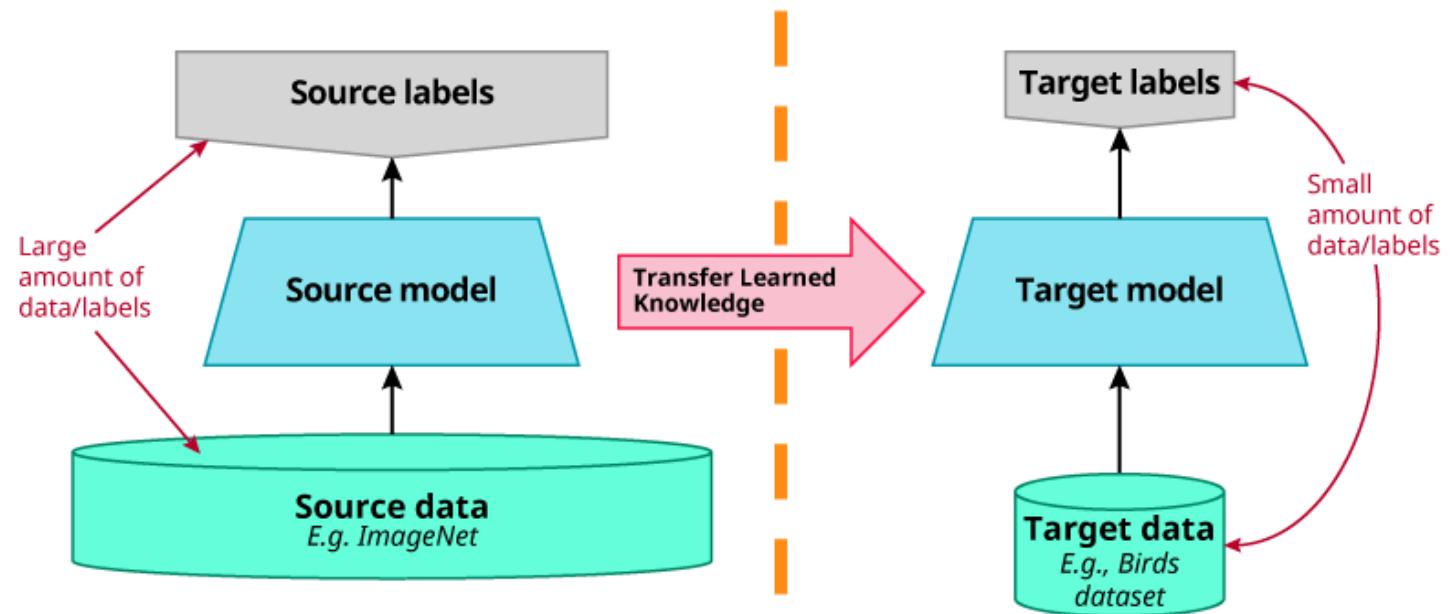
# Reference for Topic 6

- Book: Francois Chollet. Deep Learning with Python. Manning. 2018.
- Blog by Francois Chollet: <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

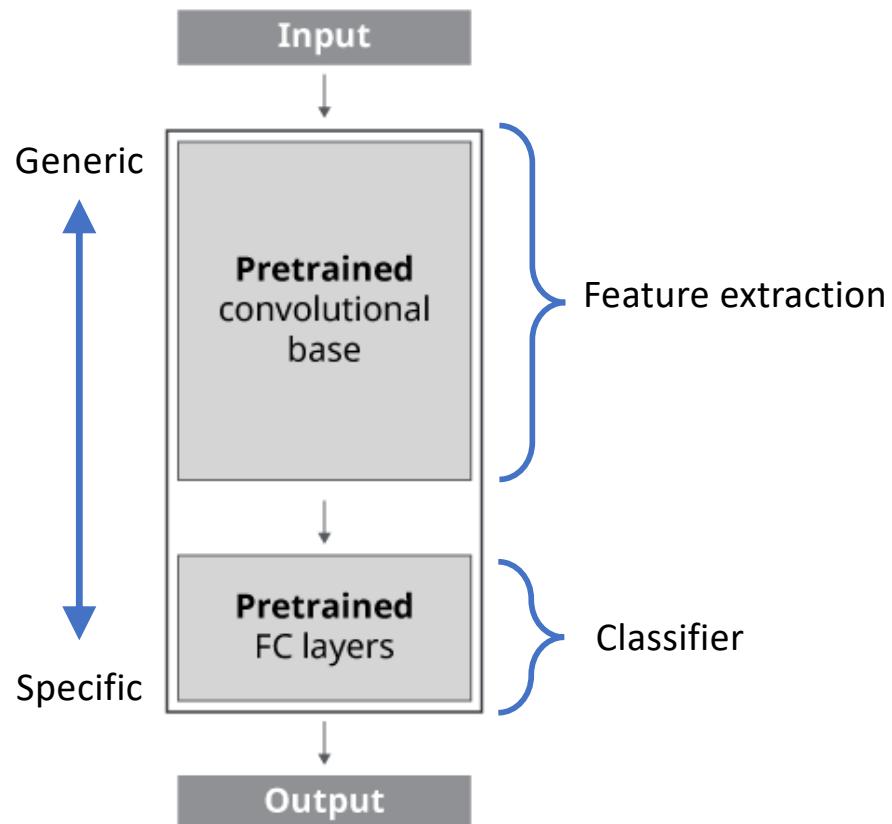
# Topic 7: Transfer Learning

# Transfer learning

- **Transfer learning** is a machine learning technique where a model trained on one task is re-purposed on another related task.



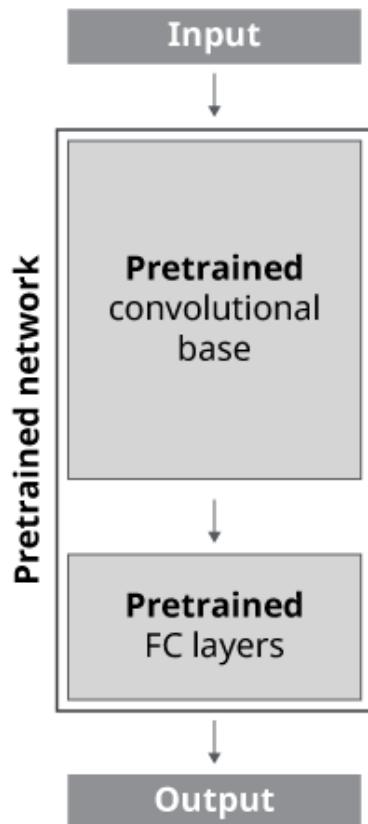
# The underlying assumption



- The **underlying assumption** of transfer learning is that generic features learned on a large enough dataset can be shared among seemingly disparate datasets.

# Two ways for transfer learning with CNNs

## 1. Feature extraction



## 2. Fine-tuning

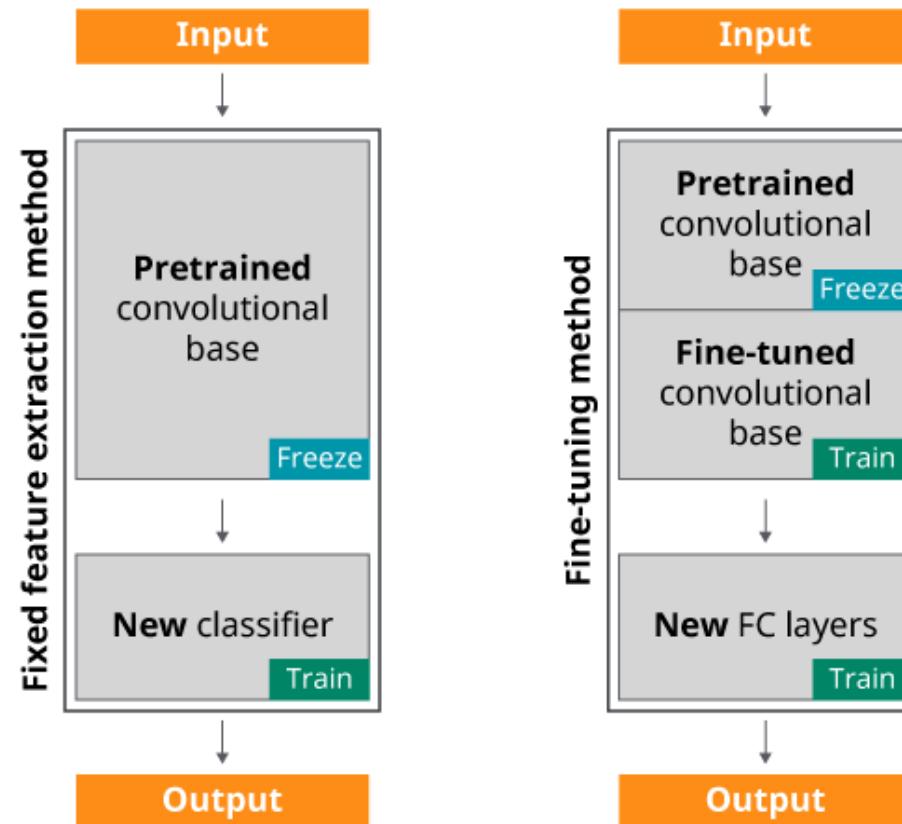
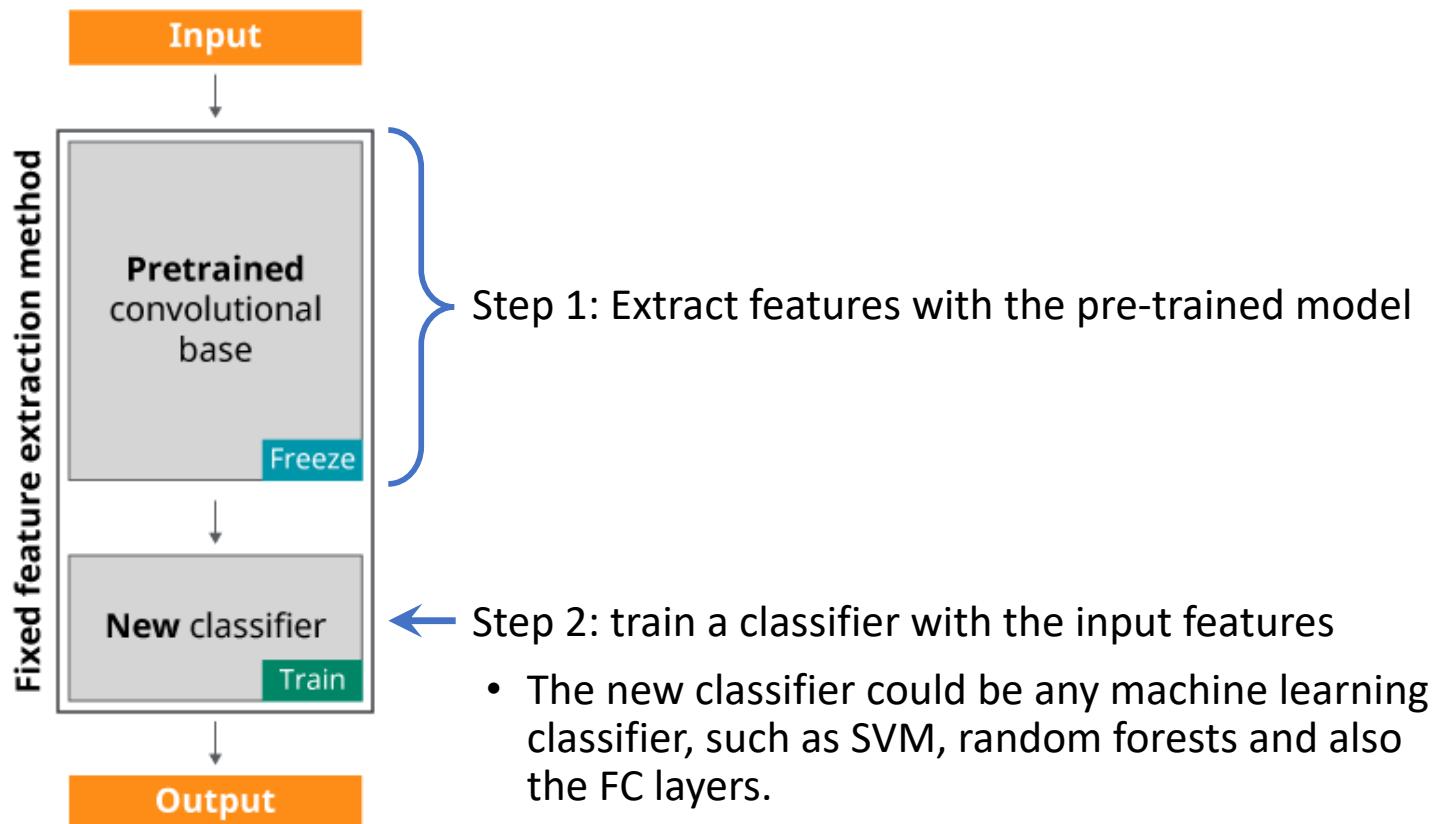
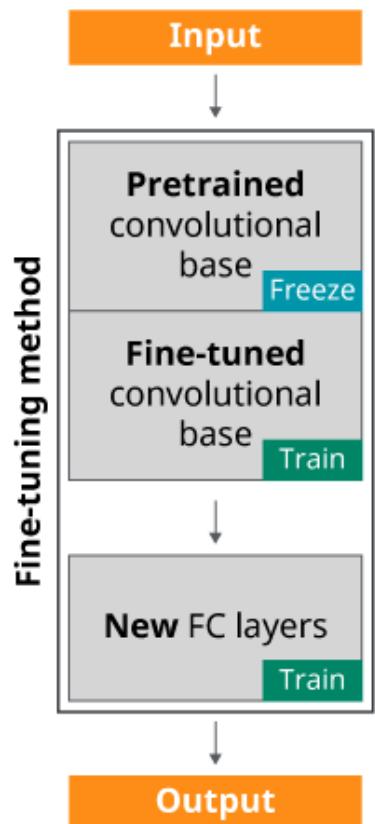


Image from: <https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9>

## 1. Feature extraction



## 2. Fine-tuning



- **Fine-tuning:** the learnt filters are not randomly initialized, but start from the pre-trained model. These filters will be slightly adjusted.
- It is common to fine-tune only the higher layers in the convolutional base, but still freeze the early layers, since early-features appear more generic.
- Use lower learning rate while fine-tuning, 1/10 of the original learning rate would be a good start point.

# Pre-trained models

- **Q:** Do I have to pre-train a model myself on a large dataset, then use it for feature extraction or fine-tuning on a small dataset?
- **A:** Not necessary. Many models pretrained on ImageNet are open to public:
  - AlexNet,
  - VGG
  - ResNet
  - Inception
  - Xception
  - DenseNet
  - MobileNet
  - ....

# Pre-trained models

- In Keras, all pre-trained models are available in `keras.application`.

```
from keras.application import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150,150,3))
```

`False` means not including the fully connected layers .

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201

...

All available models in Keras: <https://keras.io/api/applications/>

# Reference for Topic 7

- Book: Francois Chollet. Deep Learning with Python. Manning. 2018.
- Stanford University course: CNNs for Visual Recognition  
<https://cs231n.github.io/transfer-learning/>
- Blog by Francois Chollet: <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>
- R. Yamashita, et al. Convolutioanl neural networks: an overview and applicatin in radiology. Insights into imaging. P611-629, 2018.  
<https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9>

# Week 05: Recurrent Neural Networks (RNNs)

Machine Learning 2

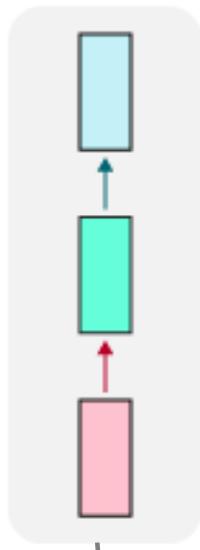
Dr. Hongping Cai

# Topic 1:

# Types of Sequence Problems

So far: Standard  
“Feedforward”  
Neural Networks

one to one



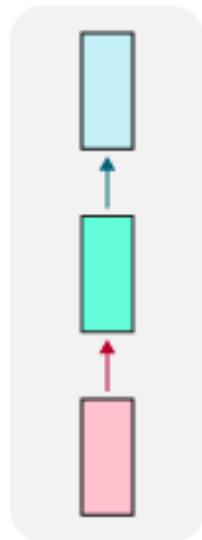
e.g. image classification, house price prediction

Image from: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

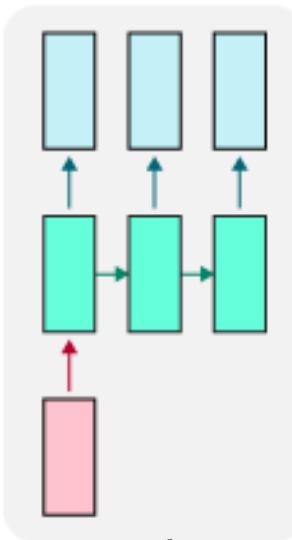
So far: Standard  
“Feedforward”  
Neural Networks

# Types of Sequence Problems

one to one



one to many



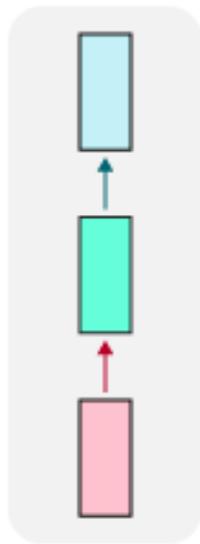
e.g. image captioning

Image from: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

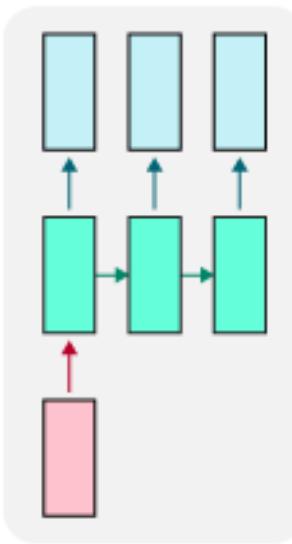
So far: Standard  
“Feedforward”  
Neural Networks

# Types of Sequence Problems

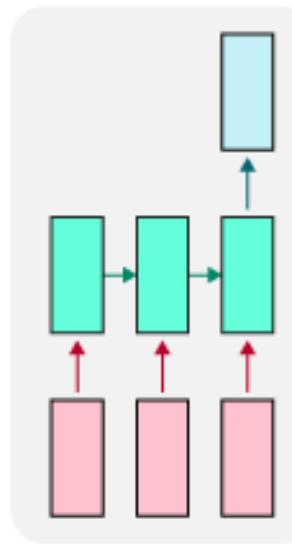
one to one



one to many



many to one

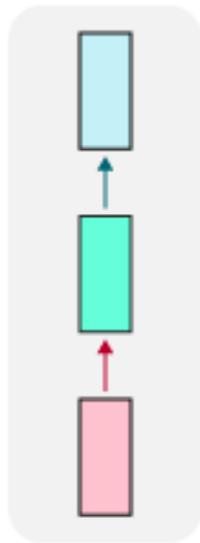


e.g. video classification, sentiment classification

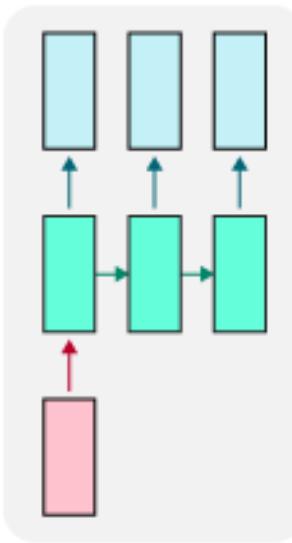
So far: Standard  
“Feedforward”  
Neural Networks

# Types of Sequence Problems

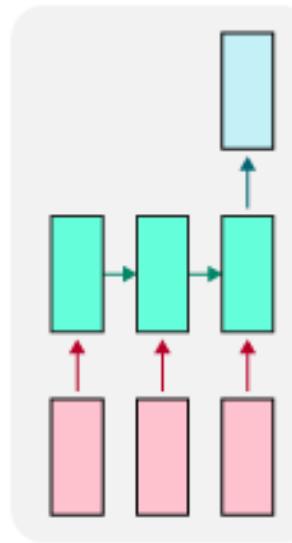
one to one



one to many

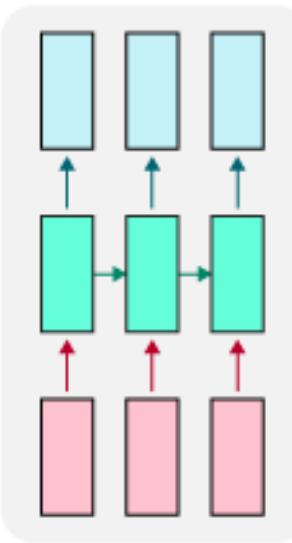


many to one



(Input length = output length)

many to many

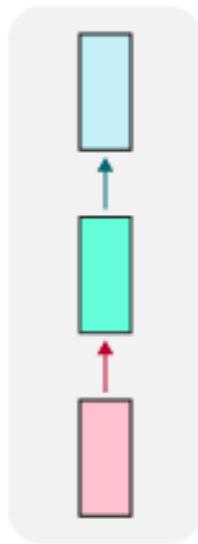


e.g. Per-frame video classification

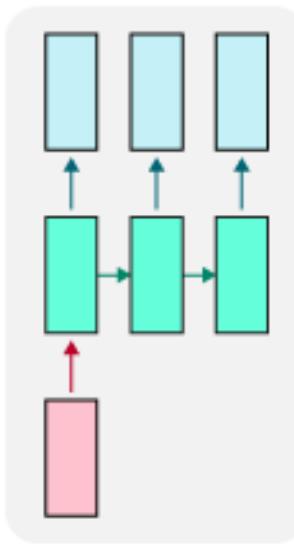
So far: Standard  
“Feedforward”  
Neural Networks

# Types of Sequence Problems

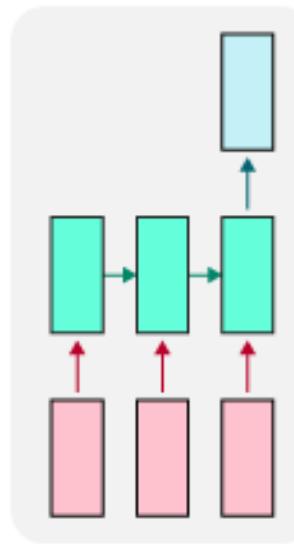
one to one



one to many

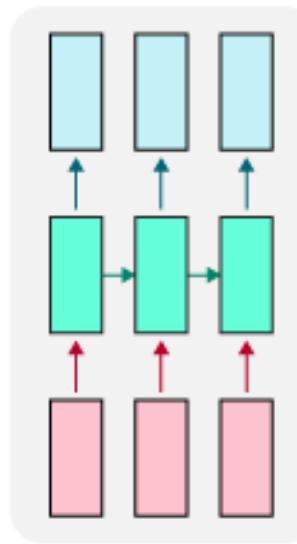


many to one



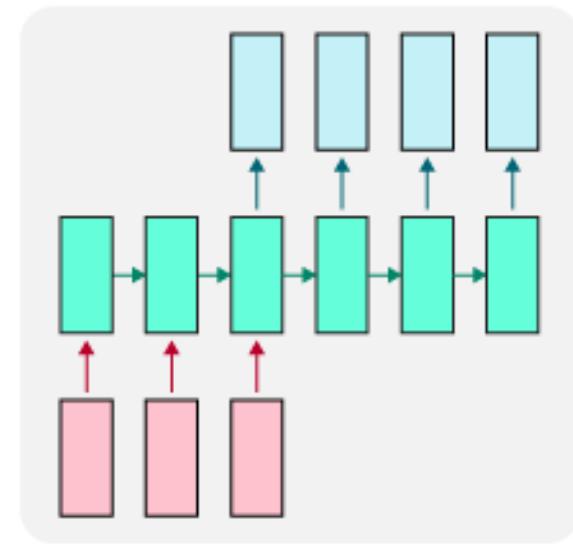
(Input length = output length)

many to many



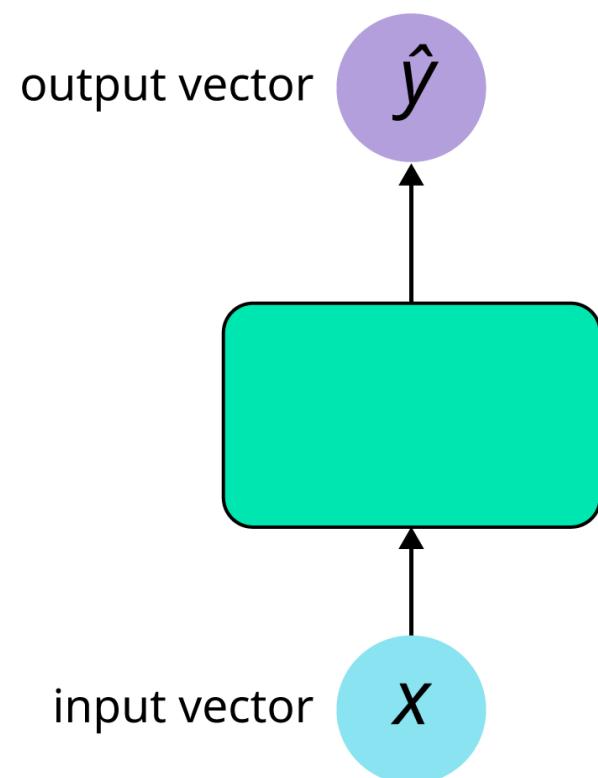
(Input length  $\neq$  output length)

many to many



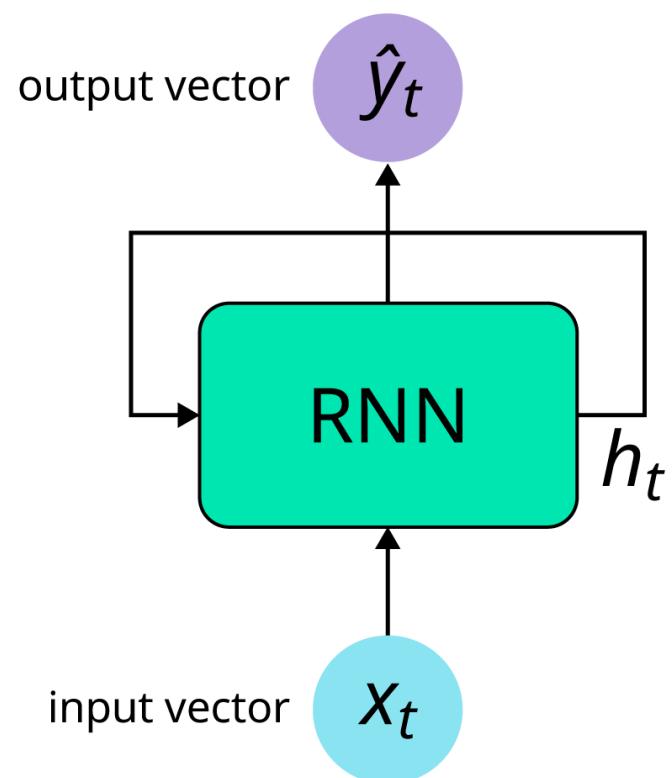
e.g. machine translation, chatbots

So far: Standard “Feedforward”  
Neural Networks



Recurrent Neural Networks (RNNs)  
for sequential modelling

Have an **internal loop**



# Reference for Topic 1

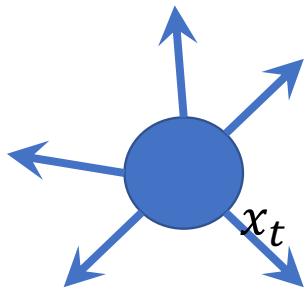
- Blog by Andrej Karpathy: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- Lectures from University of Michigan:  
<https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html>
- Video lecture by Alexander Amini: MIT course on Recurrent Neural Networks, YouTube.

# Topic 2:

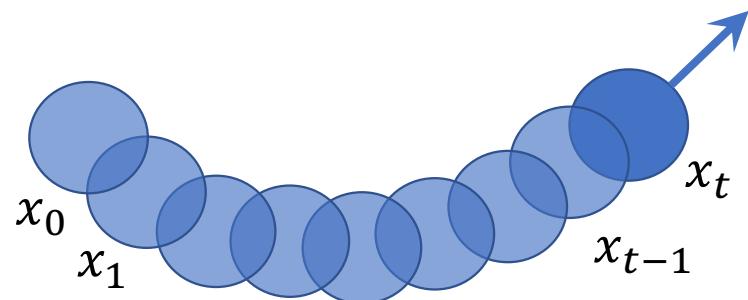
# Recurrent Neural Networks

## (RNNs)

Given a location of a ball at present, can you predict where it will go in the next second?



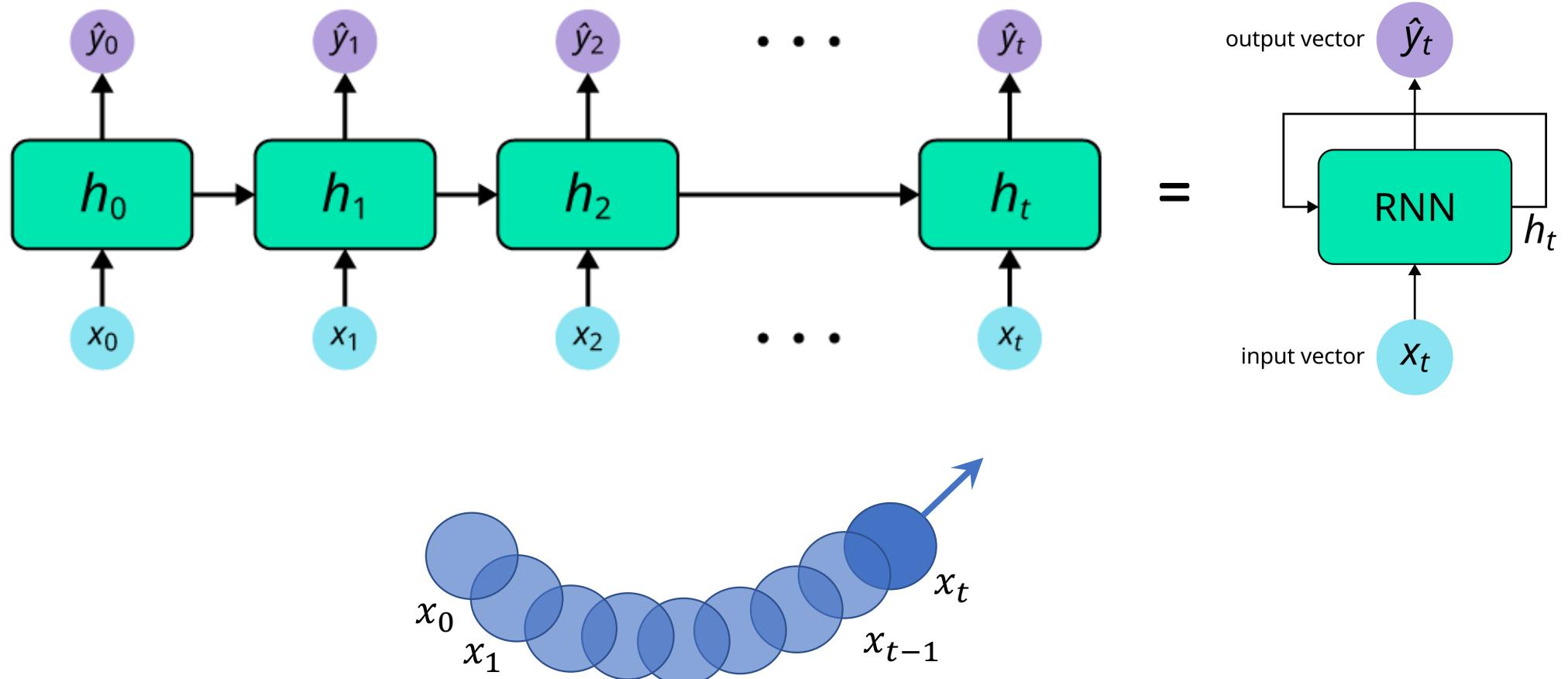
Given a location of a ball at present, can you predict where it will go in the next second?



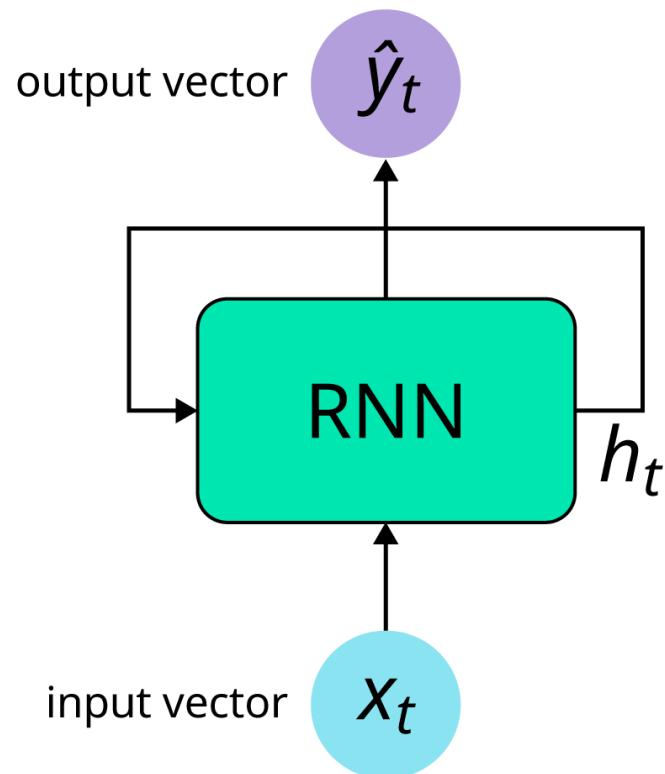
**Q:** How to model such a dependency of an input sequence?

**A:** Using a hidden state

# Recurrent Neural Networks (RNNs)



# Recurrent Neural Networks (RNNs)

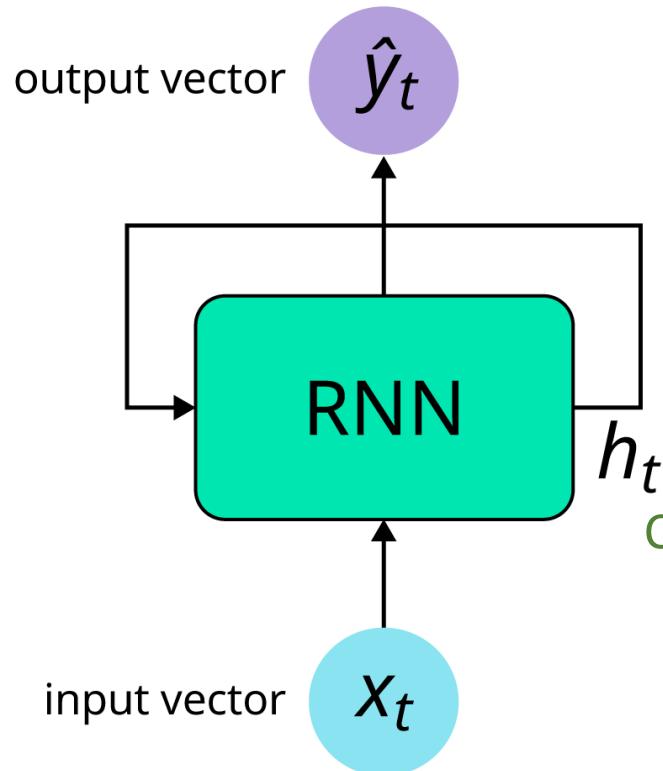


Also called “Vanilla RNNs”

Key idea: RNNs have an **internal/hidden state**  $h_t$  that can represent context information.

The	cat	sat	on	the	mat
$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$

# “Recurrent” Neural Networks (RNNs)



Apply a **recurrence formula** at every time step to update the internal/hidden state:

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

Current hidden state

Function  
parameterized by  $W$

Current input vector

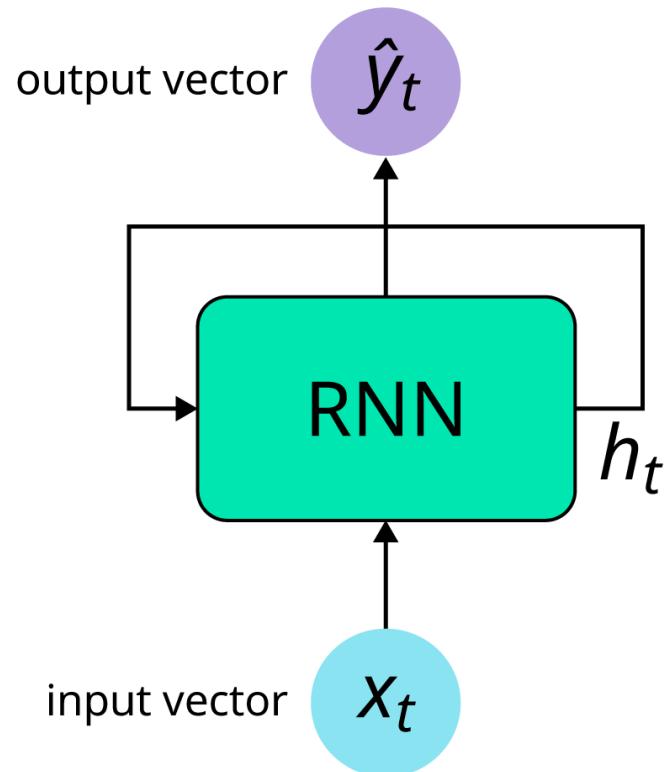
old state

$$h_{t-1} = f_W(h_{t-2}, x_{t-1})$$

...

$$h_1 = f_W(h_0, x_1)$$

# State update and output



**Output vector**

$$\hat{y}_t = g(W_{hy}h_t + b_y)$$



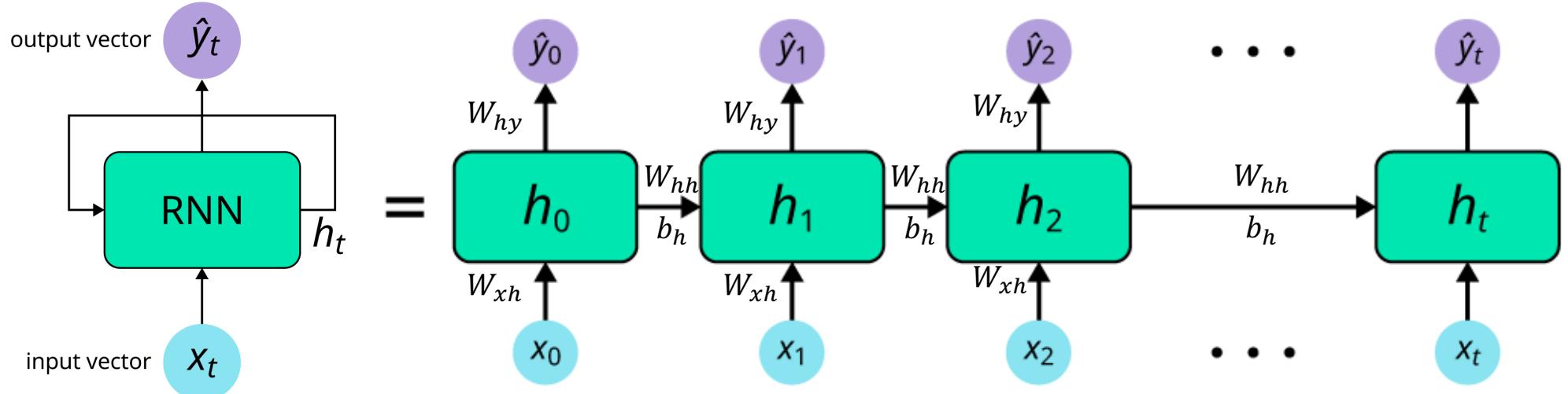
**Update the hidden state**

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

# Unfolding an RNN

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

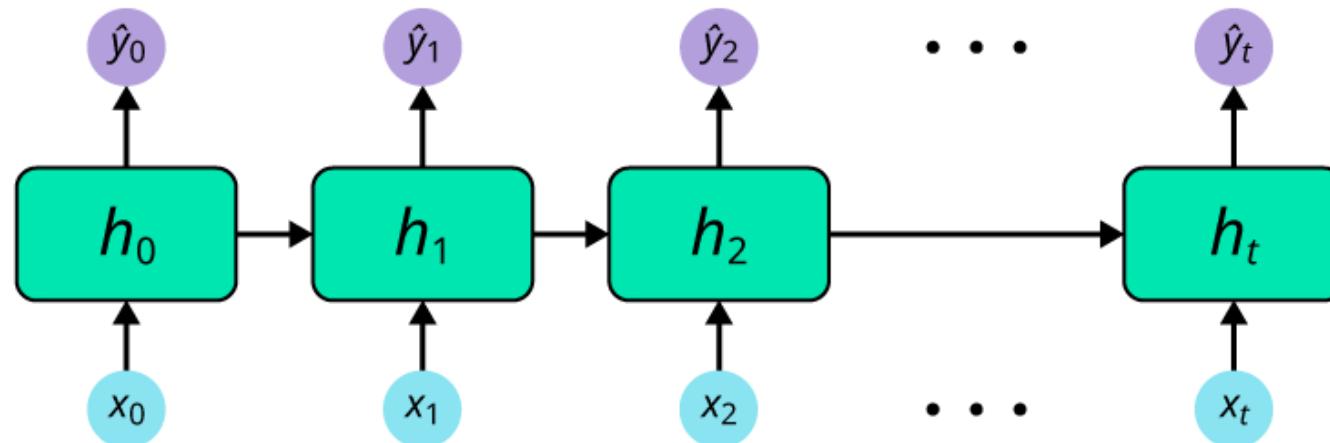
$$\hat{y}_t = g(W_{hy}h_t + b_y)$$



**The same weight matrices** are used at every time-step

# Summary of RNNs

- Main feature of RNNs is its **hidden state**, considered as the **memory** of the network.
- **Sharing parameters** across all time steps.
- We may not need inputs or output at every time step, depending on the task.

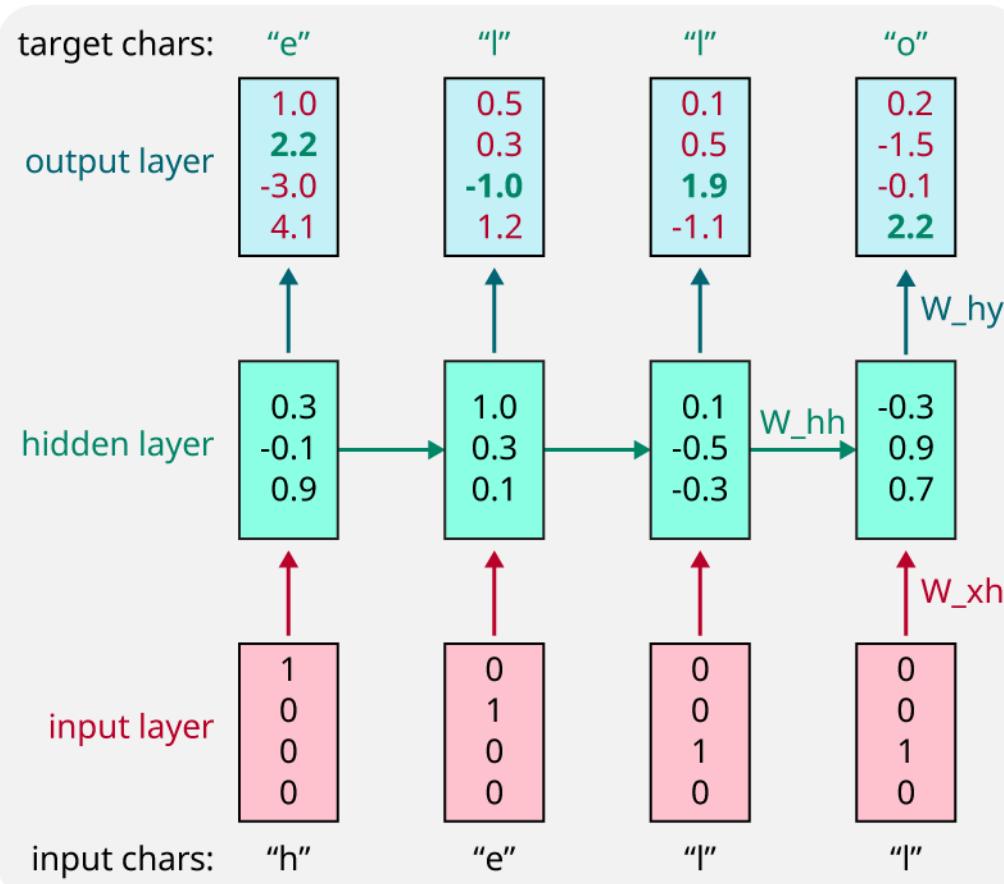


# Reference for Topic 2

- Video lecture by Alexander Amini: MIT course on Recurrent Neural Networks, YouTube.
- Blog by Andrej Karpathy: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- Lectures from University of Michigan:  
<https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html>

# Topic 3: A Simple Language- Modelling Example

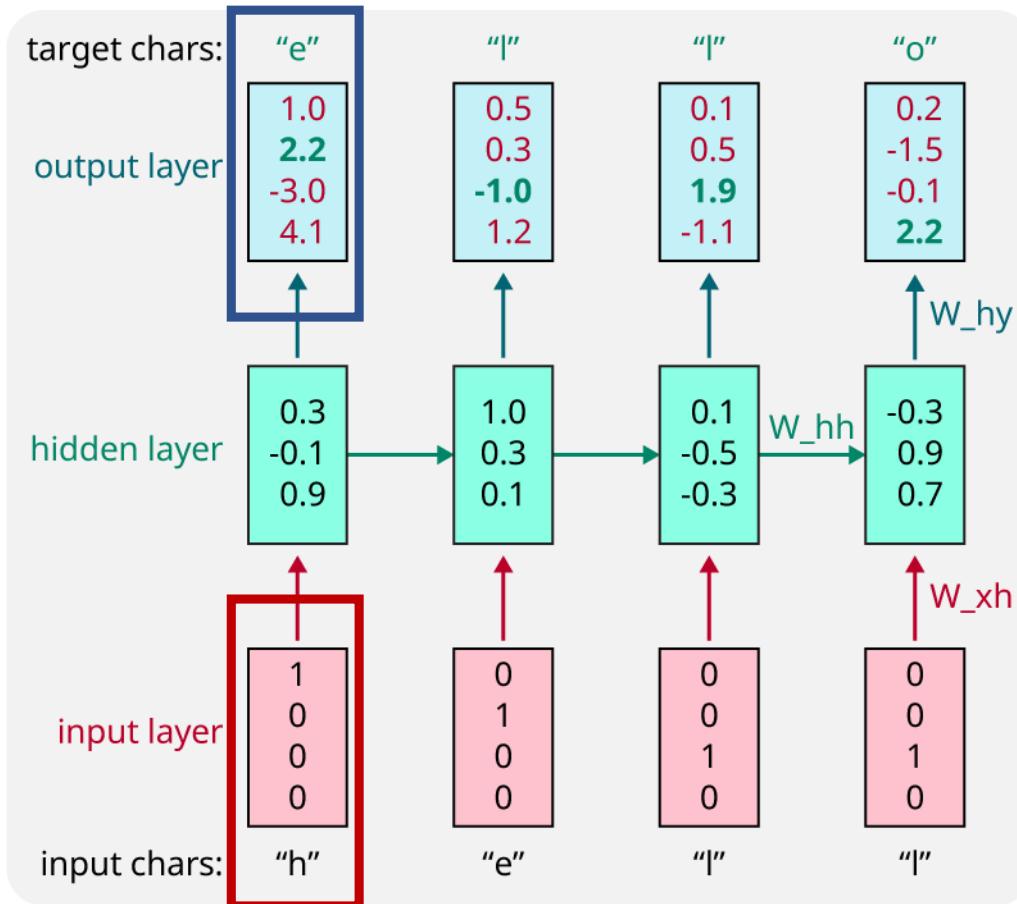
# Example: Character-level Language Modelling



**Task:** Given characters at time 1, 2, ..., t, predicts the next character (at time t+1)

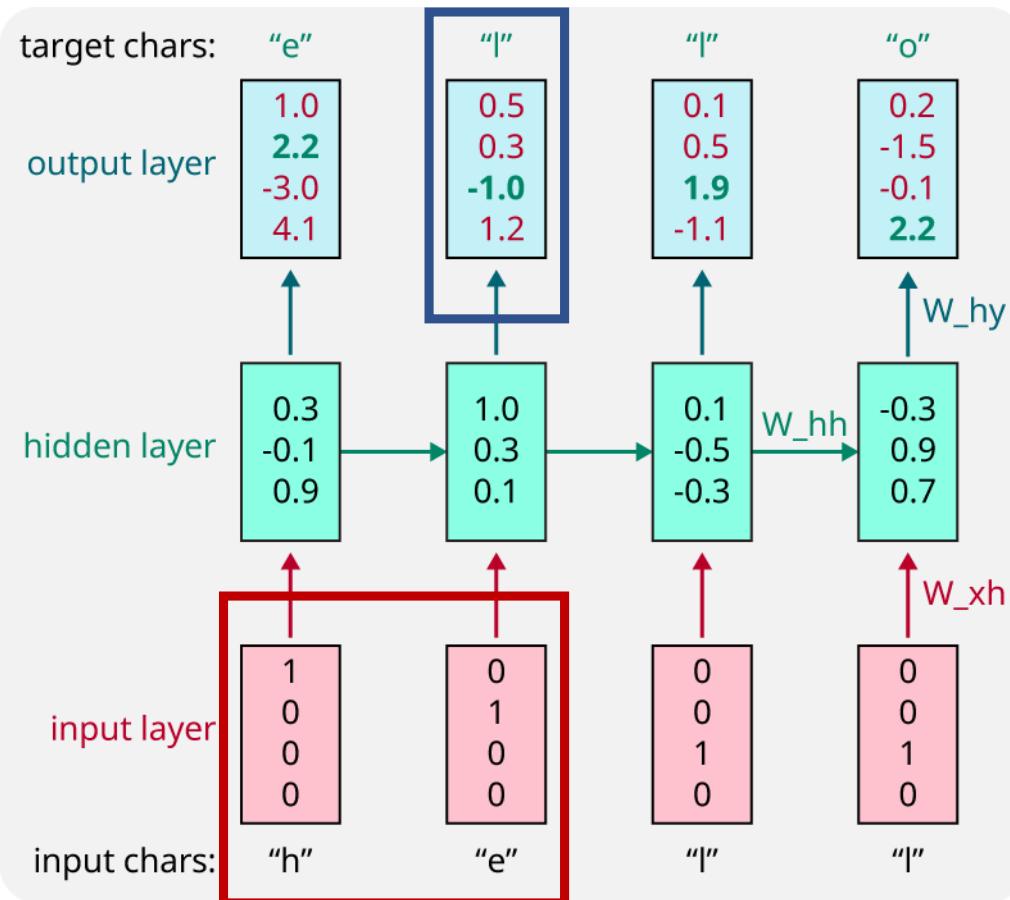
Example from: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Given “h”, target output: “e”



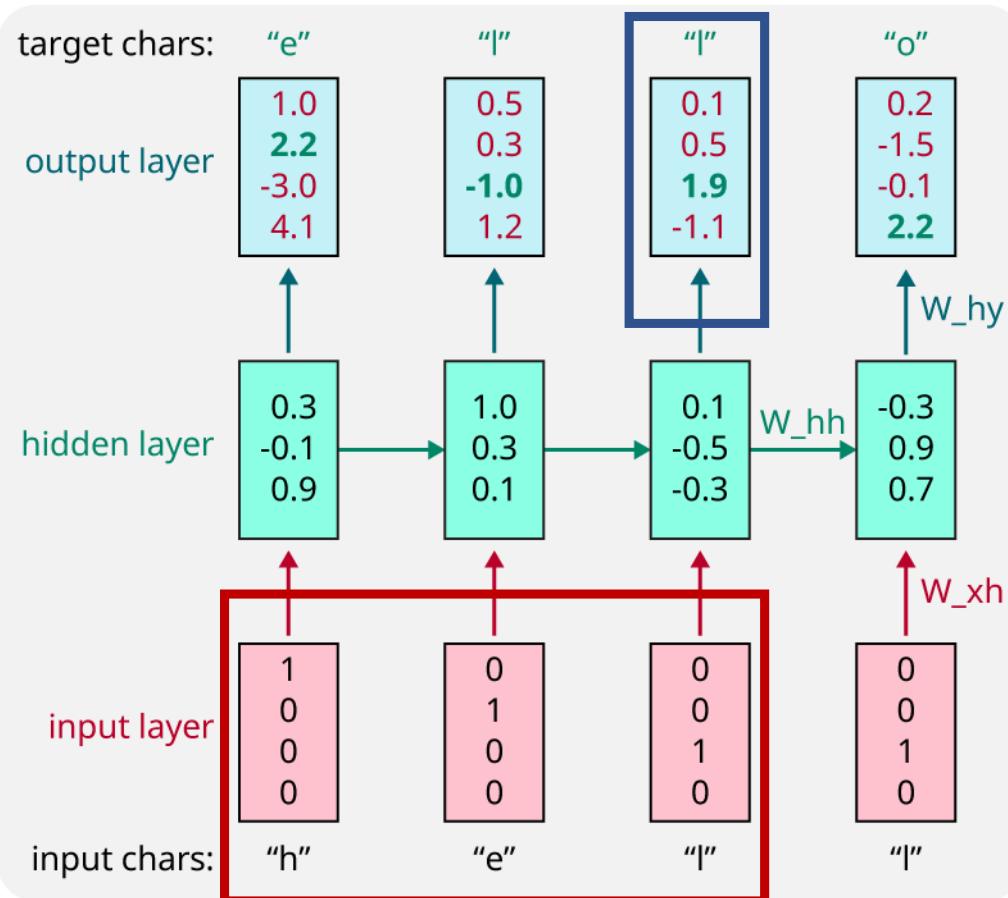
**Task:** Given characters at time 1, 2, ..., t, predicts the next character (at time t+1)

Given “he”, target output: “l”



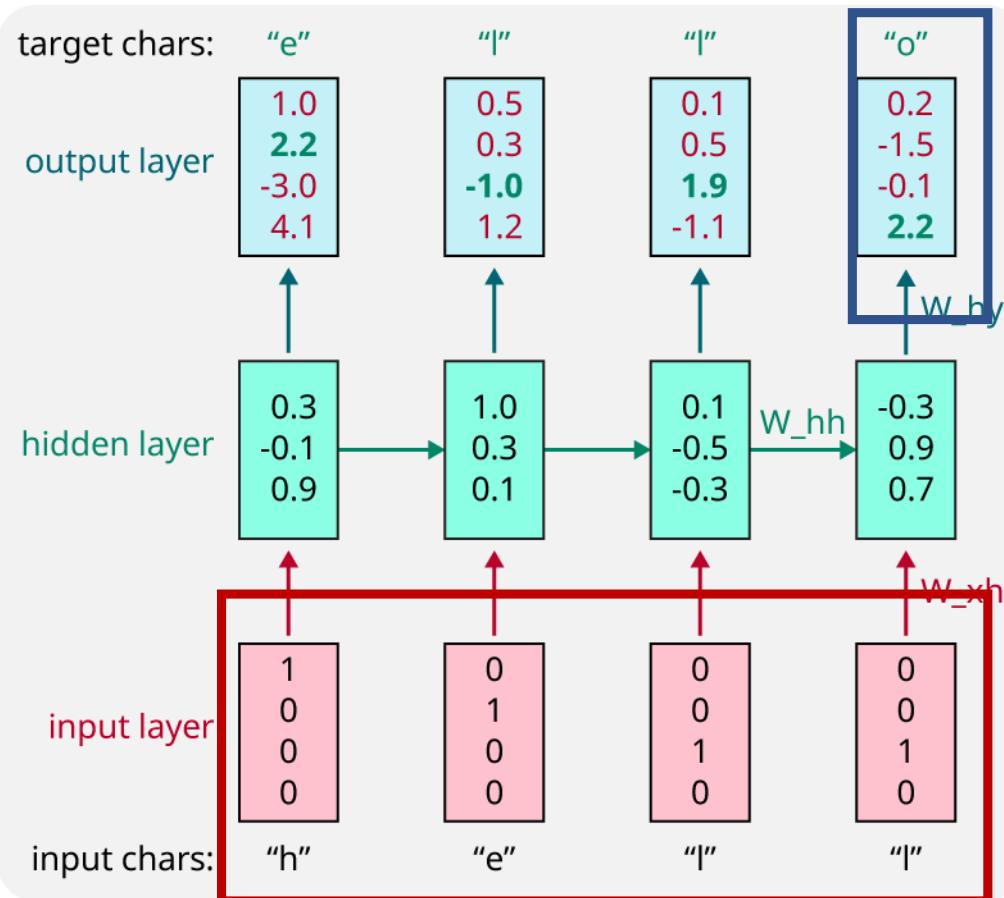
**Task:** Given characters at time 1, 2, ..., t, predicts the next character (at time t+1)

Given “hel”, target output: “l”



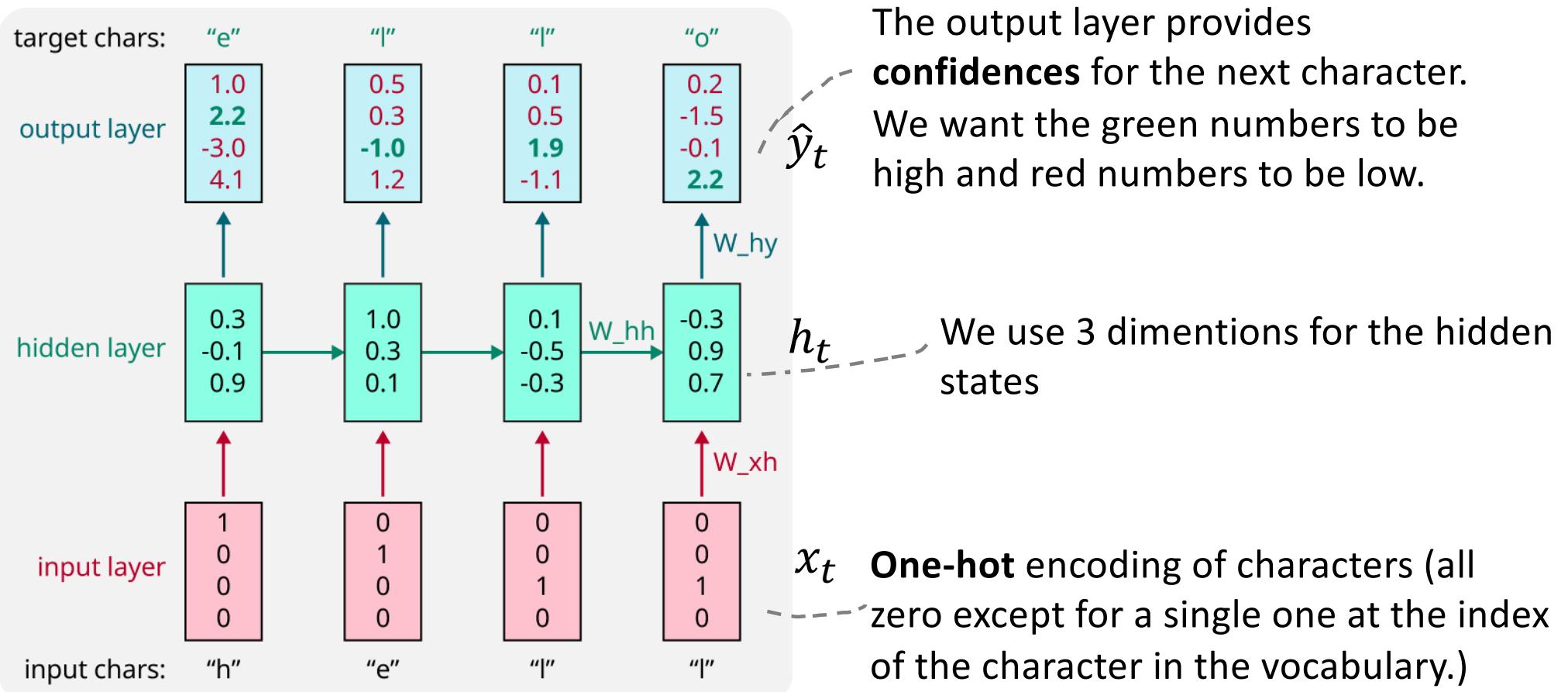
**Task:** Given characters at time 1, 2, ..., t, predicts the next character (at time t+1)

Given “hell”, target output: “o”

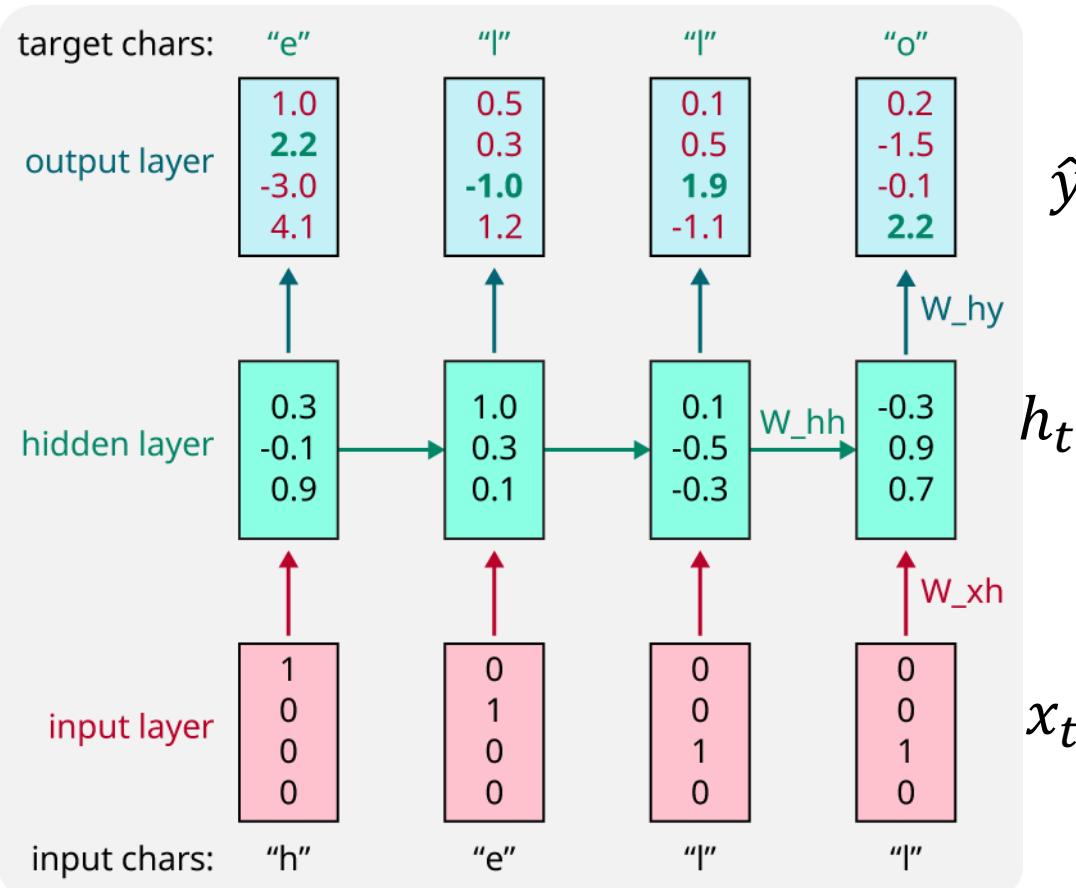


**Task:** Given characters at time 1, 2, ..., t, predicts the next character (at time t+1)

# Example: Character-level Language Modelling



# Example: Character-level Language Modelling



$$\hat{y}_t = g(W_{hy}h_t + b_y)$$

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

Simplified RNN notation:

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

$$h_t, b_h \in \mathbb{R}^H, x_t \in \mathbb{R}^M$$

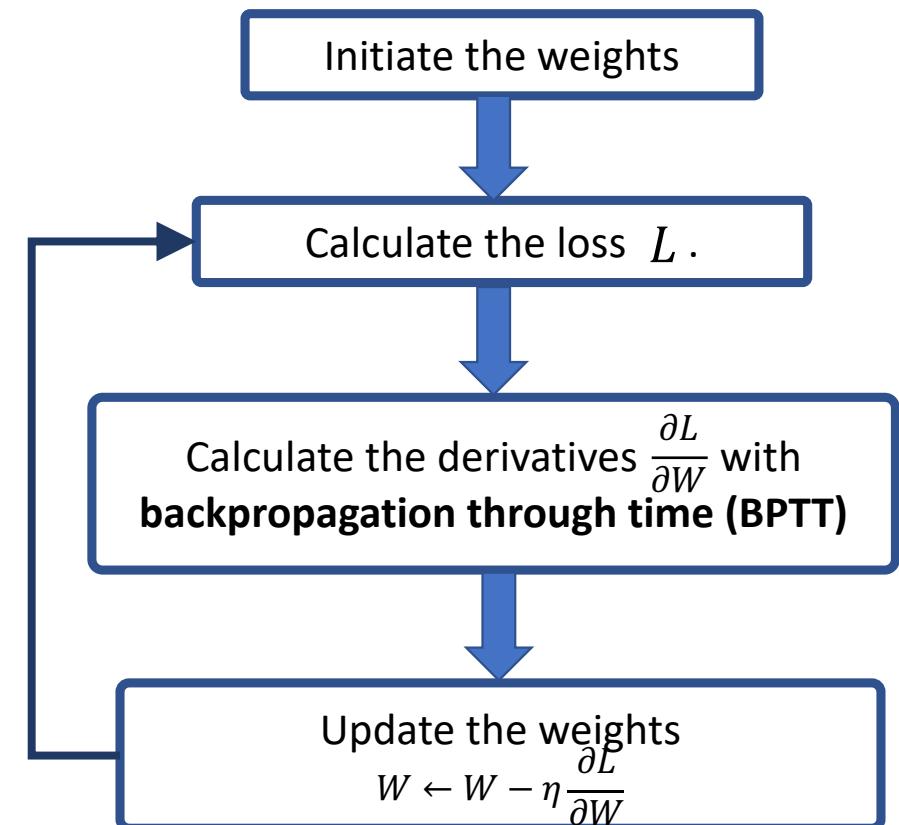
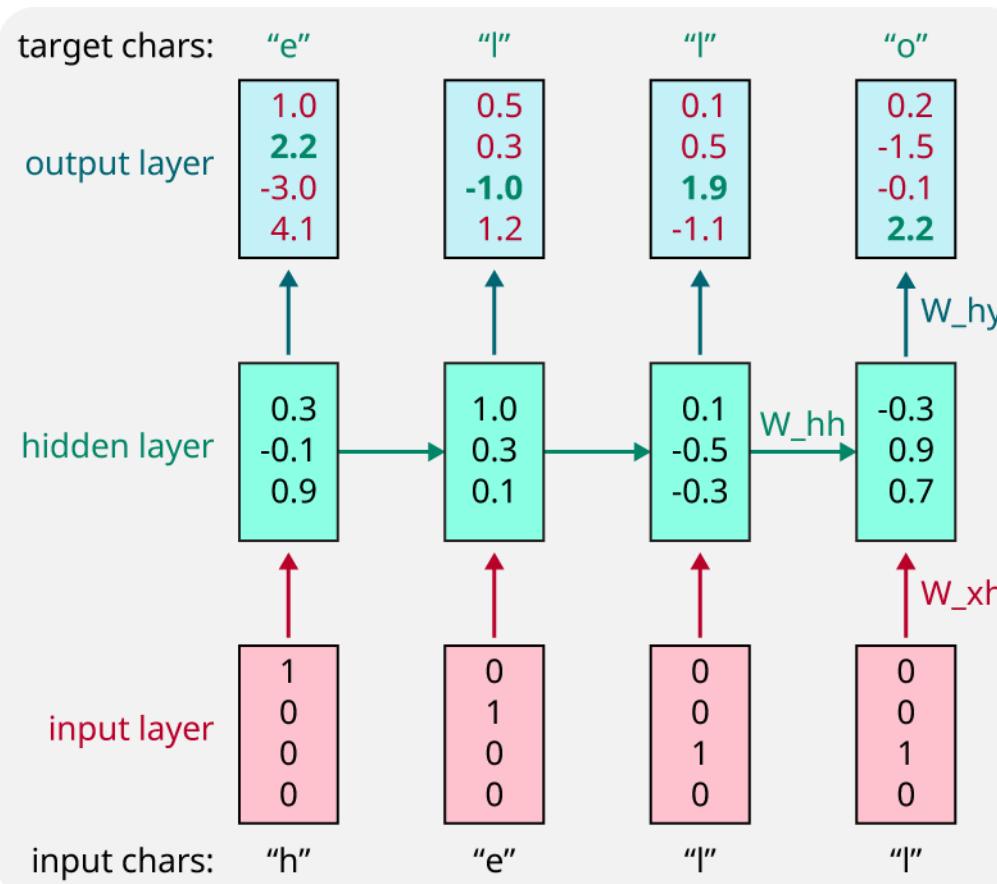
Q:  $W \in ? \quad W \in \mathbb{R}^{H \times (H+M)}$

# Reference for Topic 3

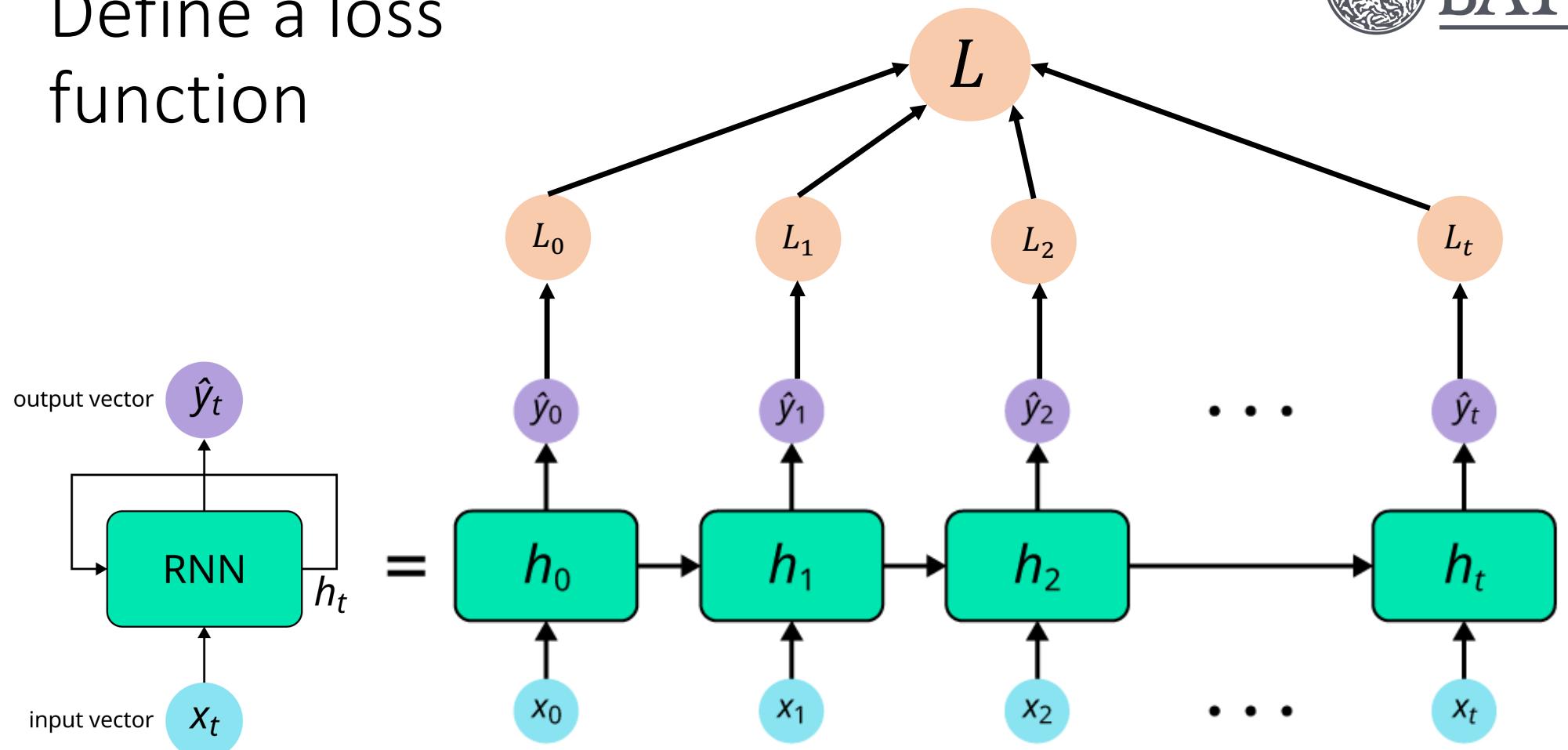
- Blog by Andrej Karpathy: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- Lectures from University of Michigan:  
<https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html>

# Topic 4: Backpropagation Through Time (BPTT)

# Training process



# Define a loss function

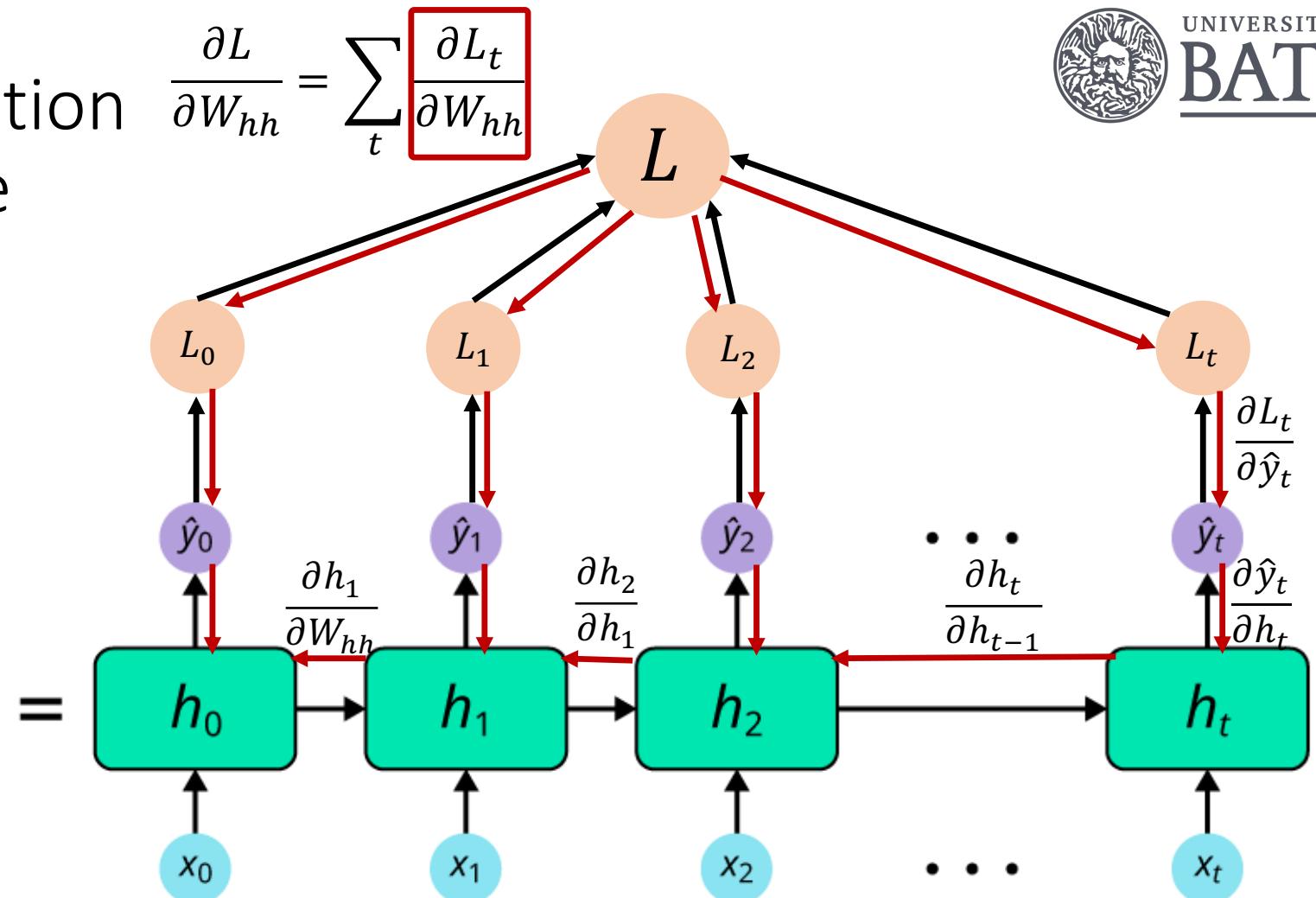
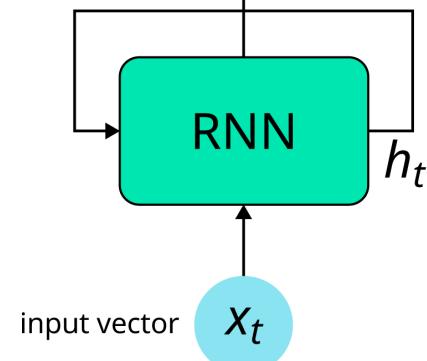


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

# Backpropagation through time (BPTT)

→ Forward pass  
← Backward pass

output vector  $\hat{y}_t$



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

See Weberna's blog for detailed equations: <https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html>

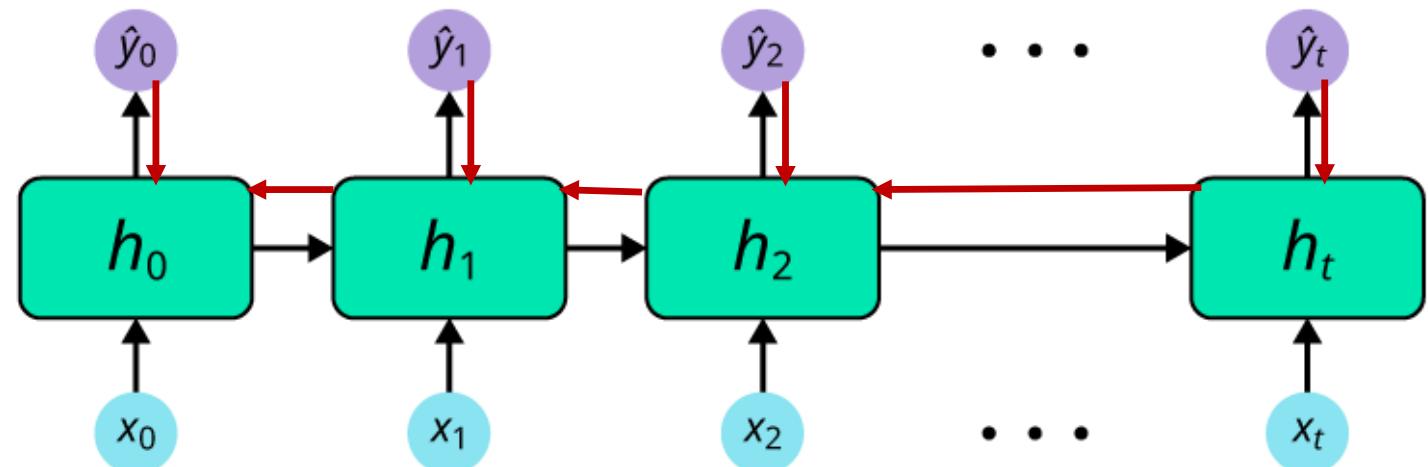
# RNN gradient flow

Many values  $> 1$ :  
**exploding gradients**

**Gradient clipping:** scale big gradients

Computing the gradient  
wrt  $W$  involves  
multiplying many factors

Many values  $< 1$ :  
**vanishing gradients**

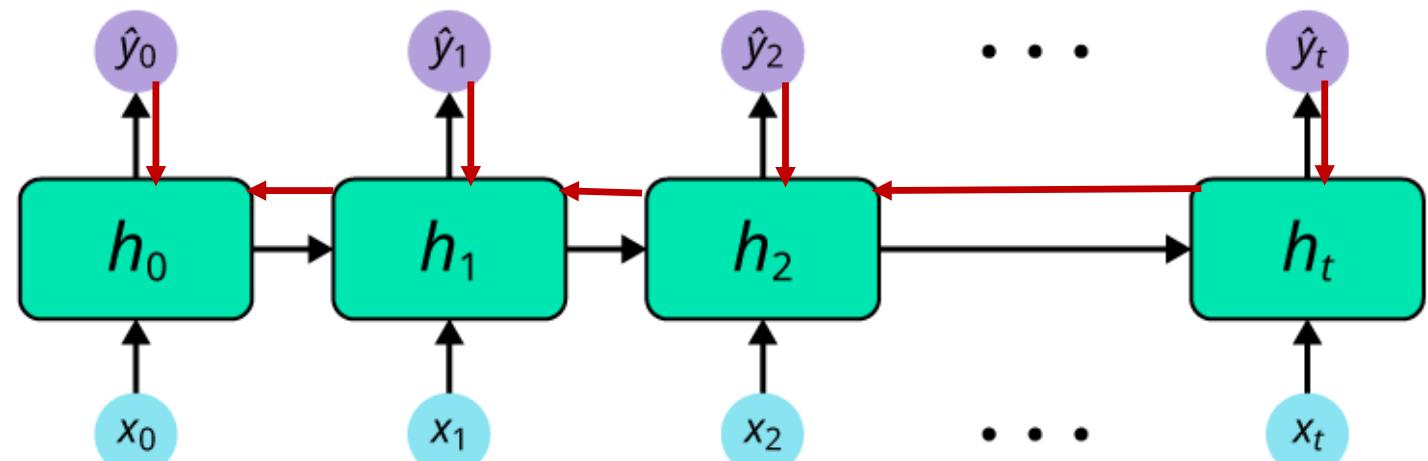


# RNN gradient flow

Many values  $> 1$ :  
**exploding gradients**

Computing the gradient  
wrt  $W$  involves  
multiplying many factors

Many values  $< 1$ :  
**vanishing gradients**

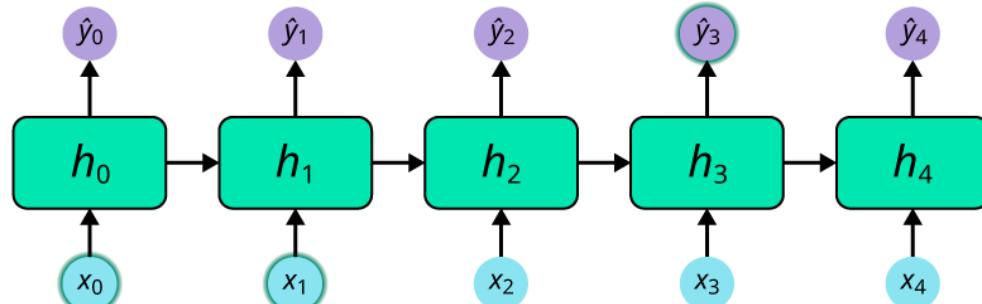


**Change RNN architecture**

# The vanishing gradient problem

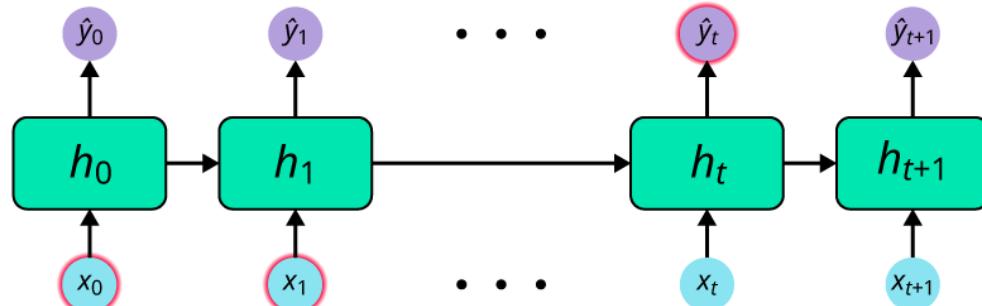
- Short term dependencies:

May I have some water to drink



- Long term dependencies:

It started raining. Mia still played in the garden, with her cloth all wet



Standard RNNs have **difficulties in modelling long-term dependencies** because of vanishing gradient problem.

# Solutions

- Key idea: use a **more complex recurrent unit** with **gates** to control the flow of information.
  - Long Short Term Memory (LSTM) ← **Next topic**
    - Sepp Hochreiter et al., “Long short-term memory”, 1997.
  - Gated Recurrent Units (GRU)
    - Cho et al “Learning phrase representations using RNN encoder-decoder for statistical machine translation”, 2014

# Reference for Topic 4

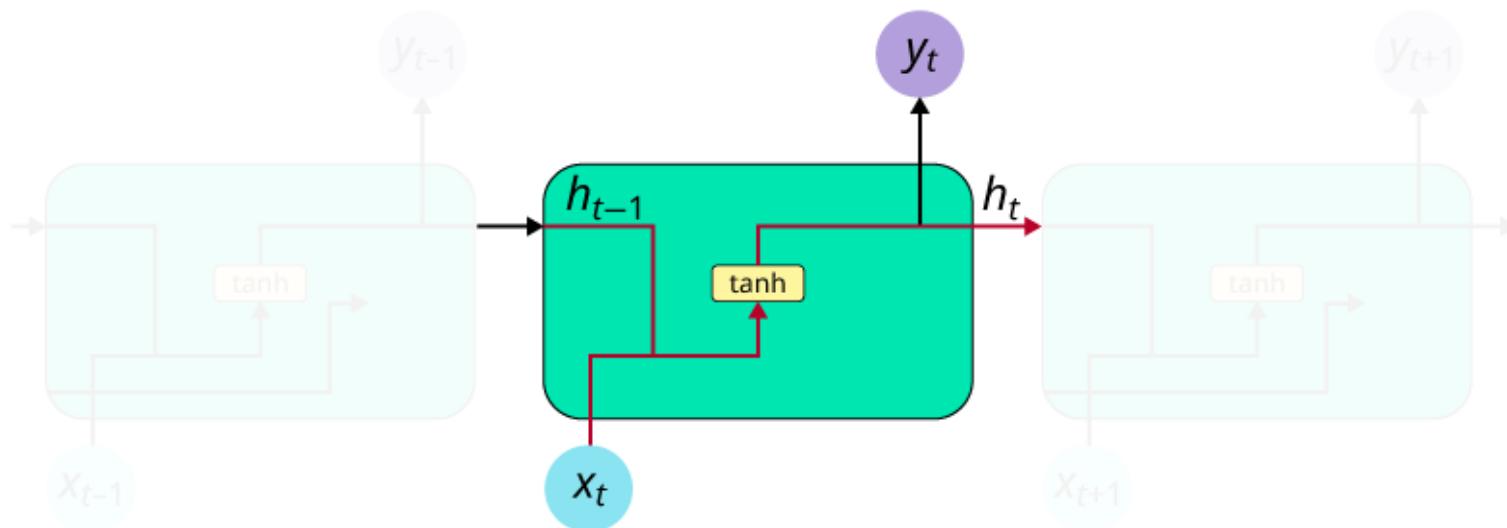
- Video lecture by Alexander Amini: MIT course on Recurrent Neural Networks, YouTube.
- Blog by Denny Brits: <http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/>
- Lectures from University of Michigan:  
<https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html>
- Blog by Weberna: <https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html>

# Topic 5: LSTM-1

# Standard/Vanilla RNNs

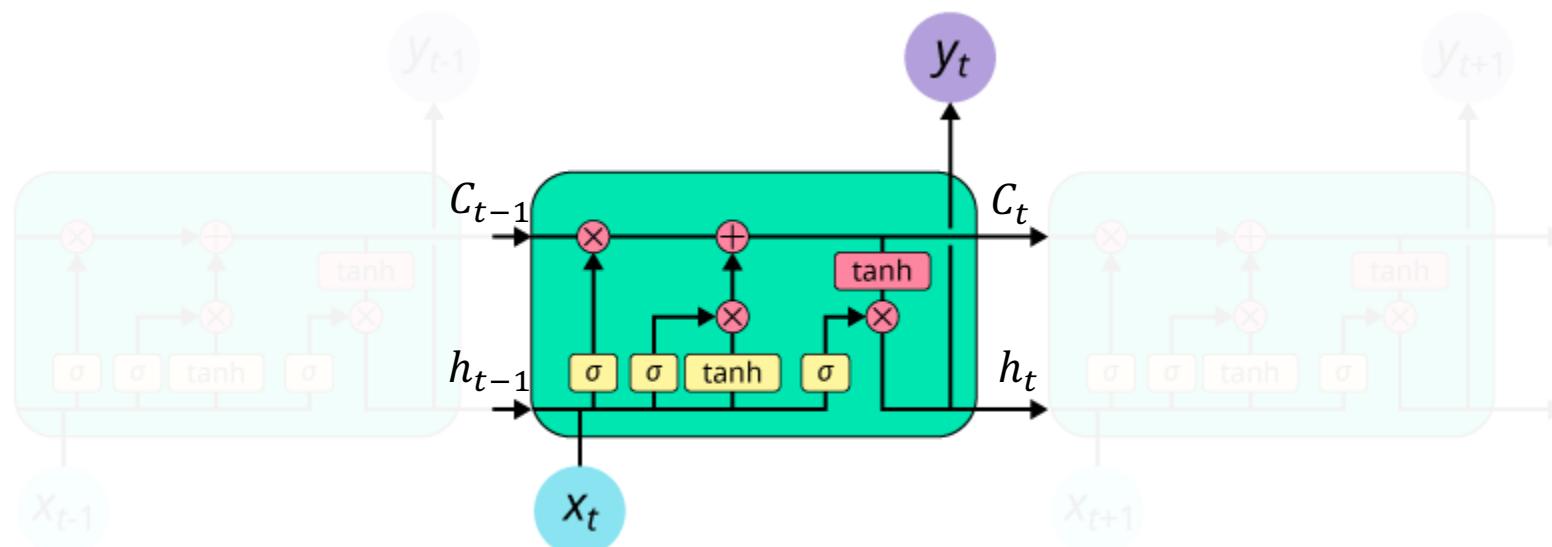
- In a standard RNN, repeating modules contain a **simple computation** node.

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



# Long-Short Term Memory (LSTM)

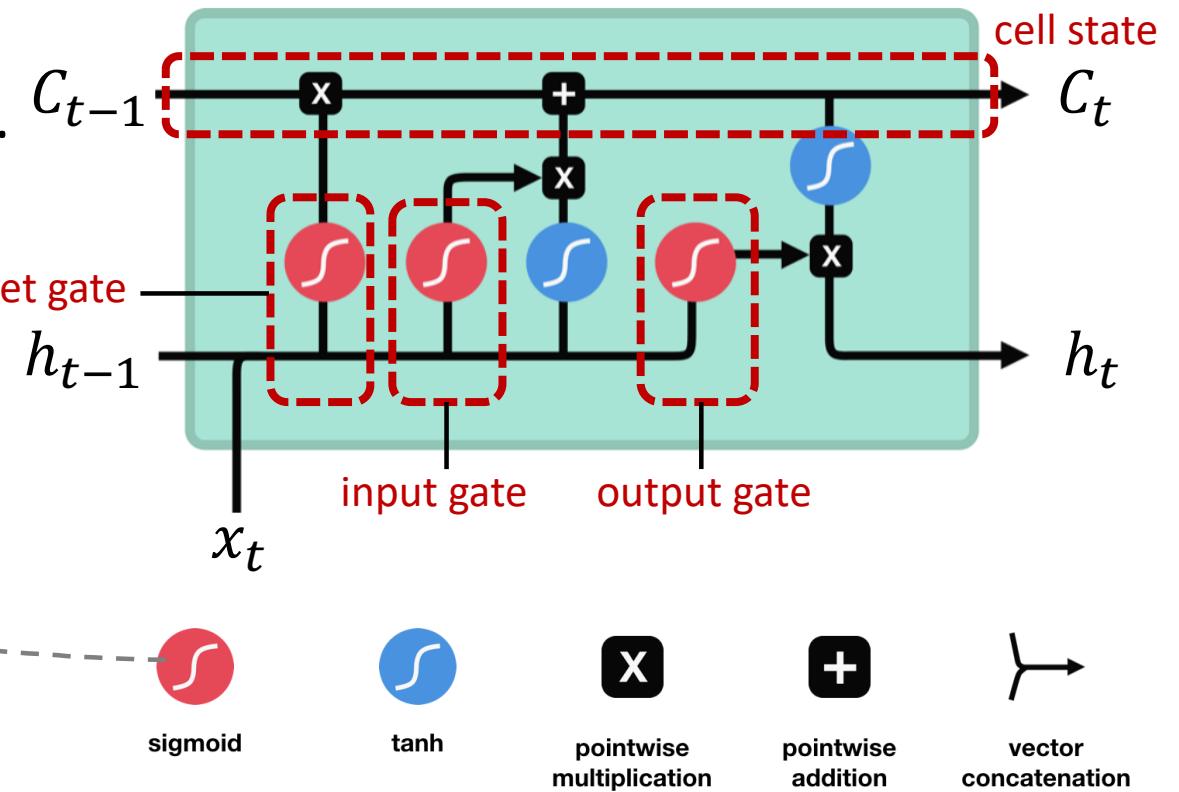
- LSTMs are explicitly designed to deal with the long-term dependency problem.
- LSTM modules contain computational blocks that control information flow.



# Two core concepts of LSTMs

- **Gates:** to control what information is to keep and forget.
- **Cell state:** act as a transport highway that transfers relative information all way down the sequence chain, thus store long-term information.

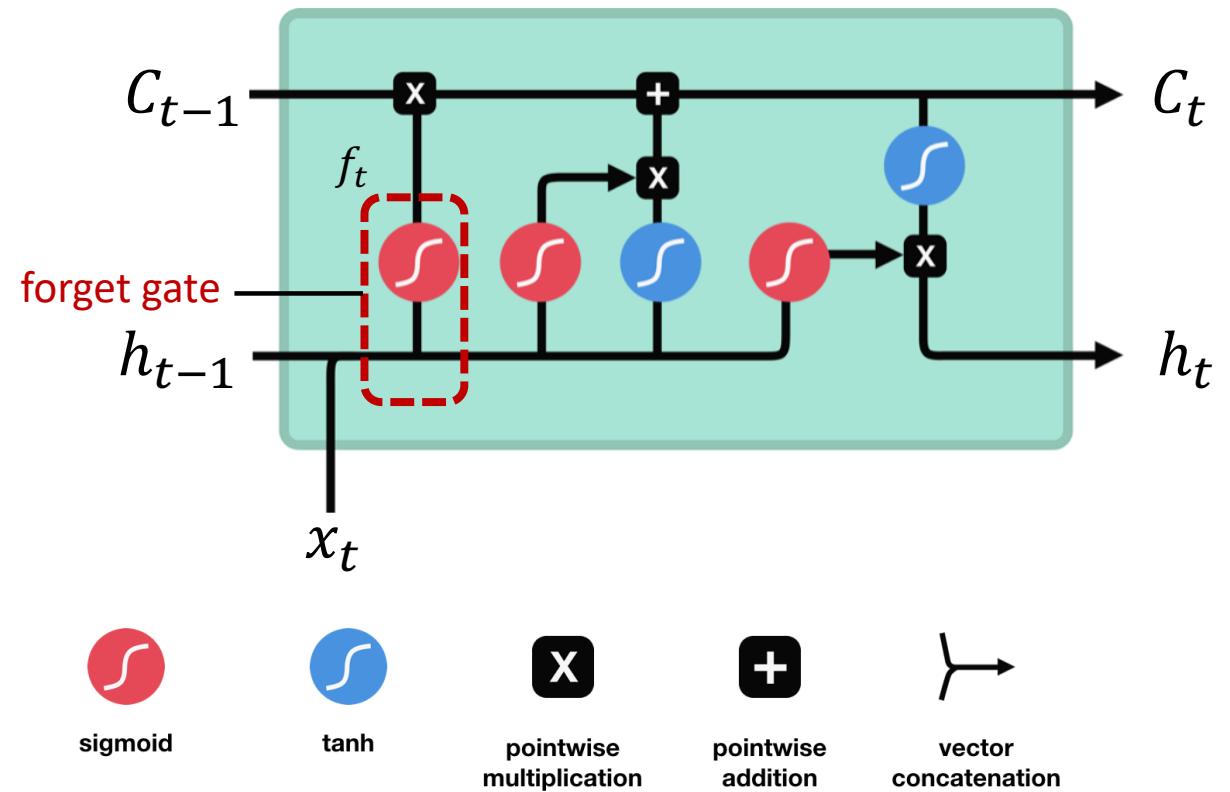
All gate values are between 0 (**discard**) and 1 (**keep**).



# Forget gate

- **Forget gate:** forget irrelevant parts of the previous state.

$$f_t = \sigma(W_f \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_f)$$



Animation from: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

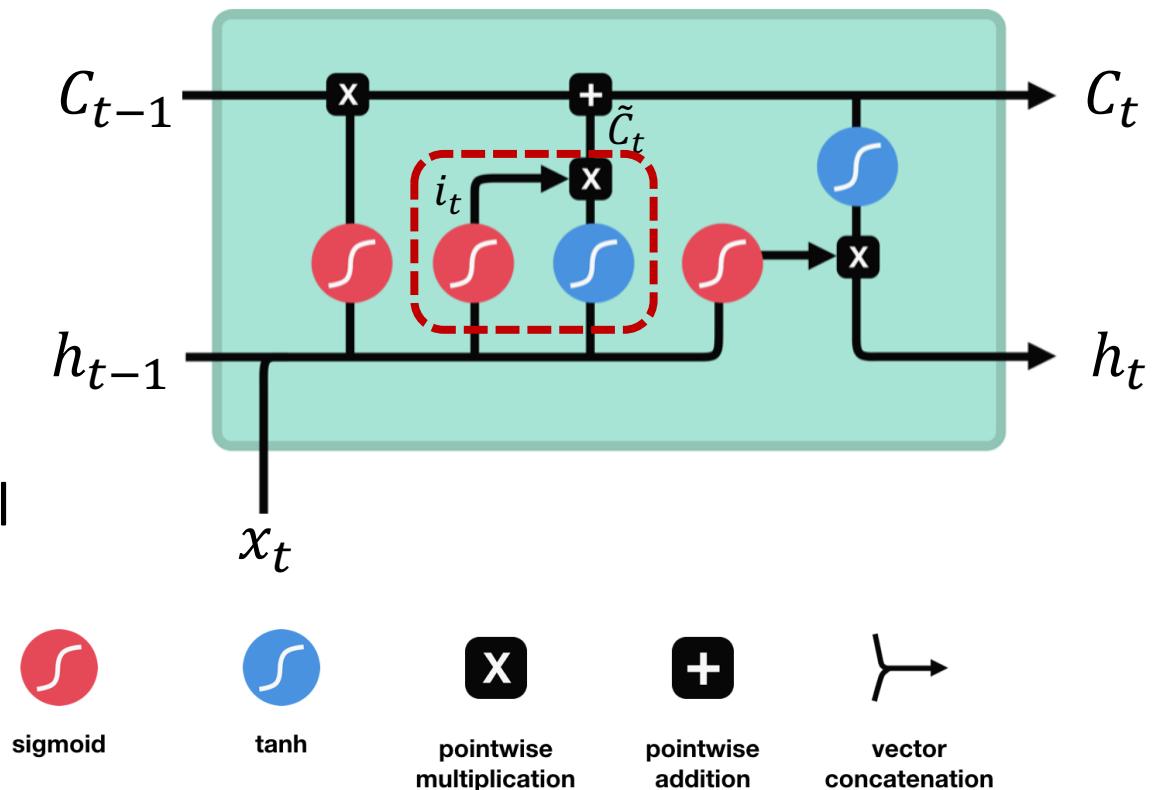
# Input gate

- **Input gate:** decides what information is relevant to add from the current step.

$$i_t = \sigma(W_i \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_i)$$

- **New cell content:** the new content to be written to the cell

$$\tilde{C}_t = \tanh(W_C \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_C)$$



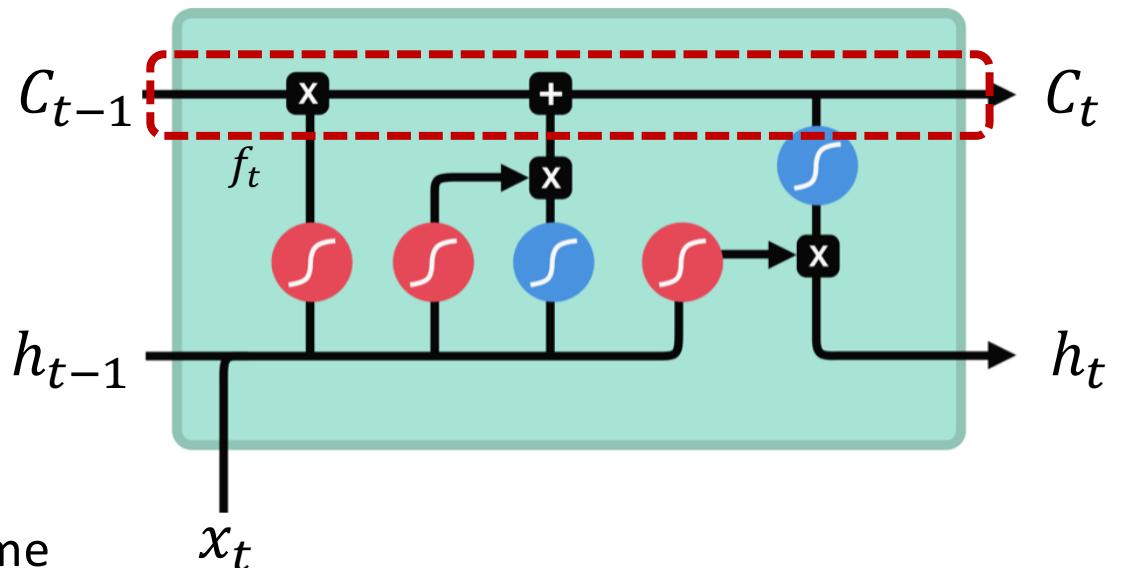
# Cell state

- **Cell state:** update the cell state to new values that the network finds relevant.

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


 Erase ("forget") some content from the previous state


 Keep ("input") some new cell content



sigmoid



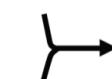
tanh



pointwise  
multiplication



pointwise  
addition



vector  
concatenation

Animation from: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

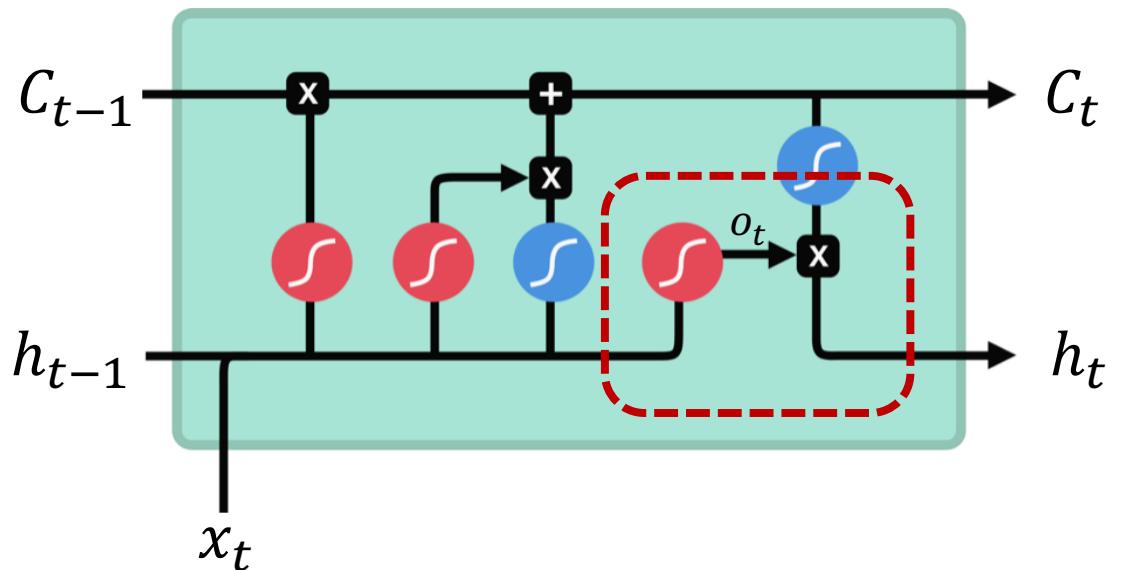
# Output gate

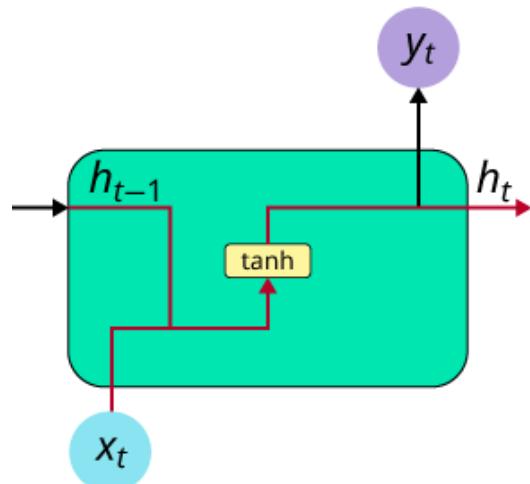
- **Output gate:** determines what parts of the cell are output to the hidden state.

$$o_t = \sigma(W_o \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_o)$$

- **Hidden state:** read (“output”) some content from the cell.

$$h_t = o_t \circ \tanh(C_t)$$

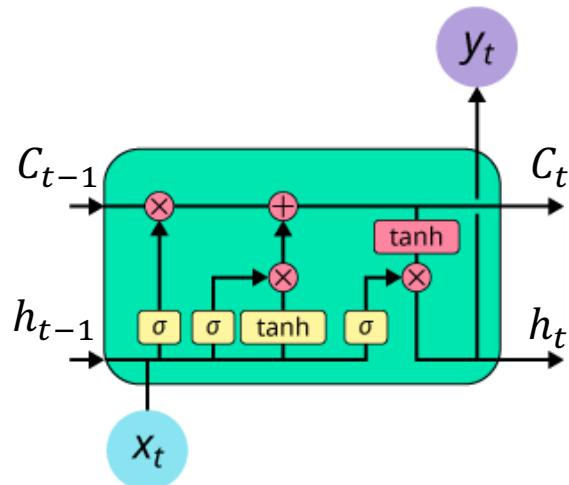




## Vanilla RNN:

$$\begin{aligned} h_t, b_h &\in \mathbb{R}^H \\ x_t &\in \mathbb{R}^M \\ W &\in \mathbb{R}^{H \times (H+M)} \end{aligned}$$

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



## LSTM:

$$\begin{aligned} f_t &\in \mathbb{R}^H \\ i_t &\in \mathbb{R}^H \\ o_t &\in \mathbb{R}^H \\ \tilde{C}_t &\in \mathbb{R}^H \\ C_t &\in \mathbb{R}^H \\ h_t &\in \mathbb{R}^H \\ x_t &\in \mathbb{R}^M \end{aligned}$$

$$W_f, W_i, W_o, W_C \in \mathbb{R}^{H \times (H+M)}$$

$$f_t = \sigma(W_f \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_f)$$

$$i_t = \sigma(W_i \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_i)$$

$$o_t = \sigma(W_o \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_o)$$

$$\tilde{C}_t = \tanh(W_C \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} + b_C)$$

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

$$h_t = o_t \circ \tanh(C_t)$$

# Reference for Topic 5

- Blog by Colah: Understanding LSTM Networks.  
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- Blog by Michael Phi: Illustrated Guide to LSTM's and GRU's: A step to step explanation. <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>
- Video lecture by Alexander Amini: MIT course on Recurrent Neural Networks, YouTube.
- Lectures from University of Michigan:  
<https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html>

# Topic 6: LSTM-2

# LSTM Architecture

**Forget** some previous cell content

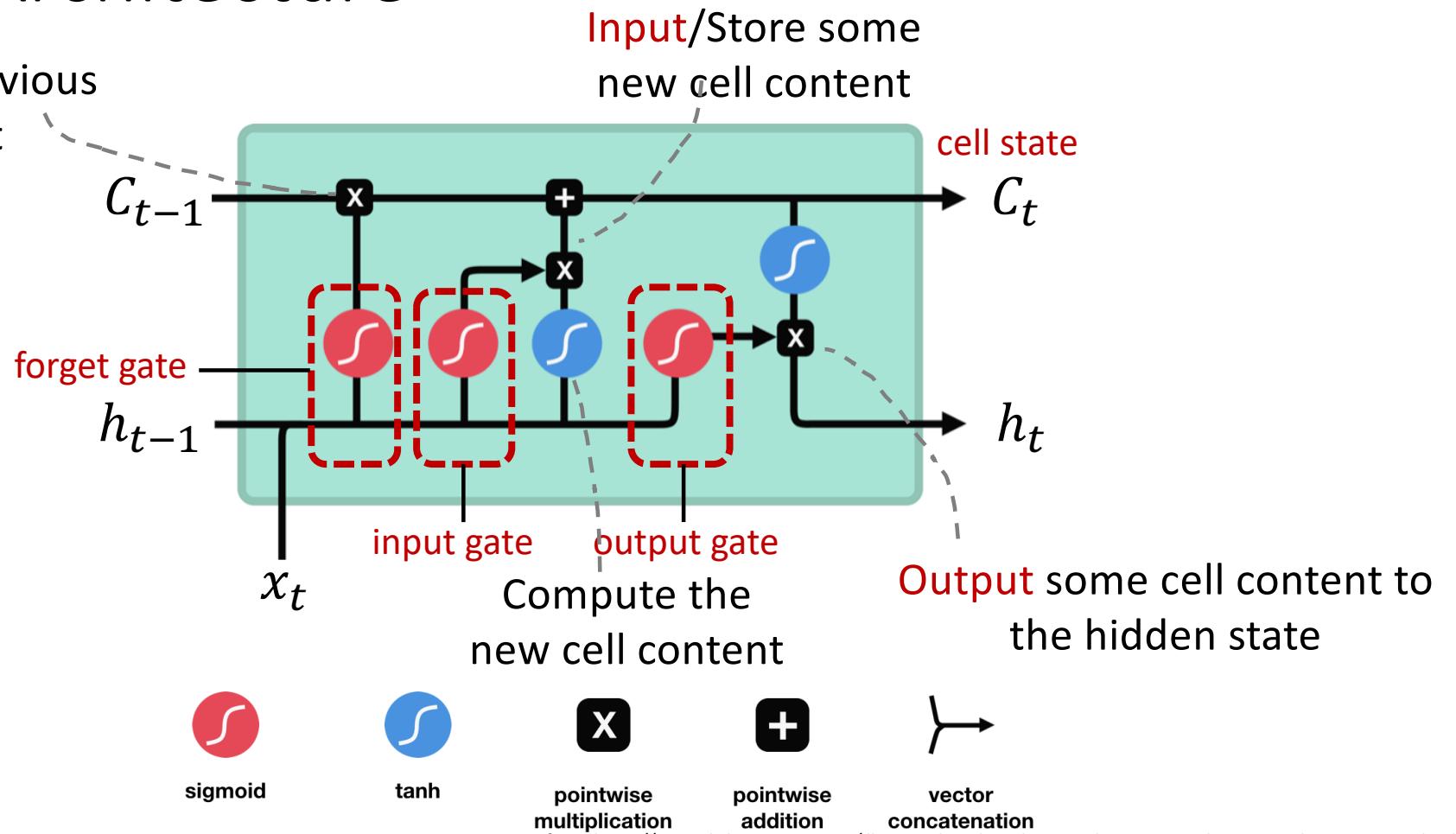
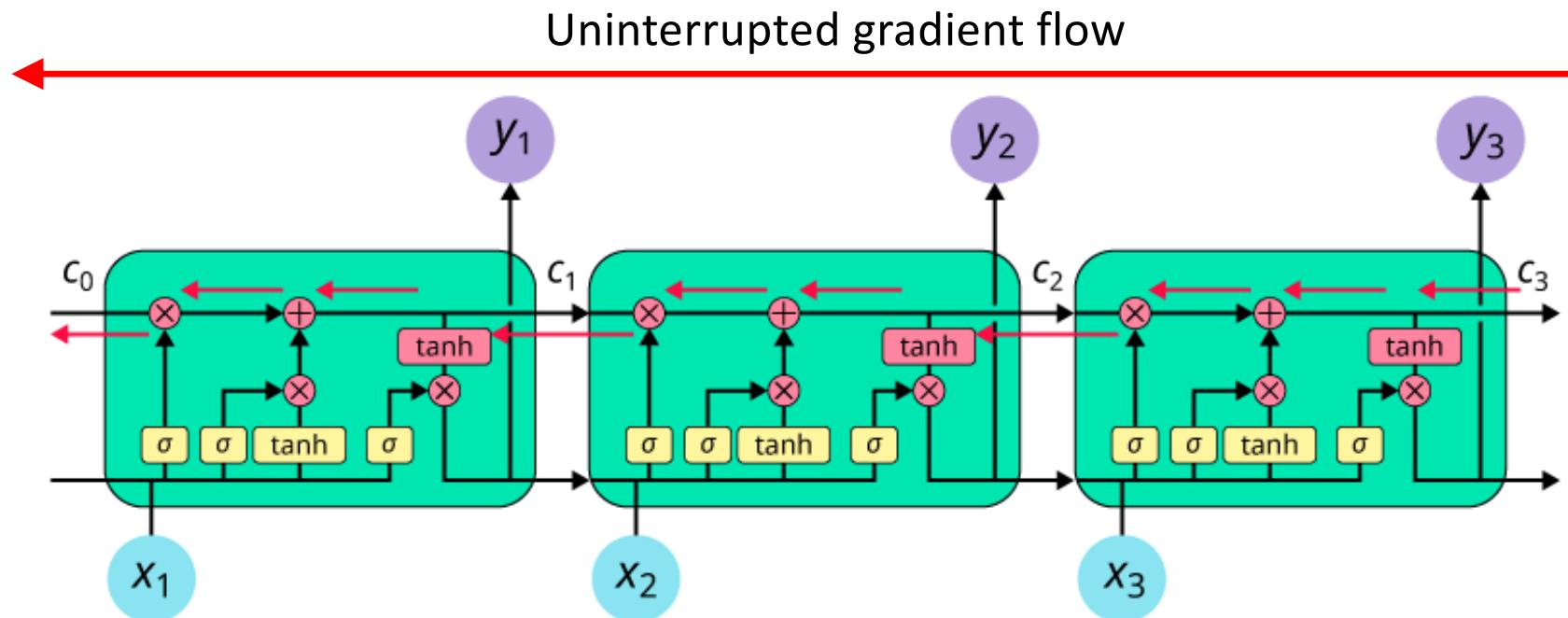


Image from: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

# LSTM Gradient Flow



**LSTMs solve the vanishing/exploding gradient problem  
using an additive gradient structure.**

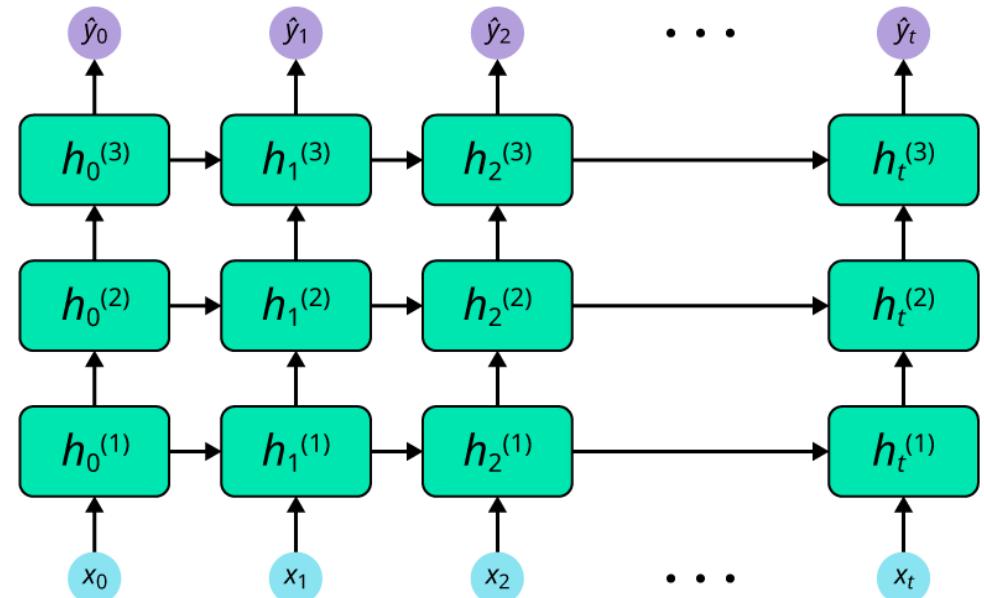
# Advanced use of RNNs/LSTMs

- **Multi-layer RNNs (Deep RNNs):**

stack more than one RNN. It increases the representation power of the network, at the cost of higher computational loads.

```
from keras.layers import LSTM
...
model.add(LSTM(32), return_sequences=True)
model.add(LSTM(32), return_sequences=True)
model.add(LSTM(32))
...

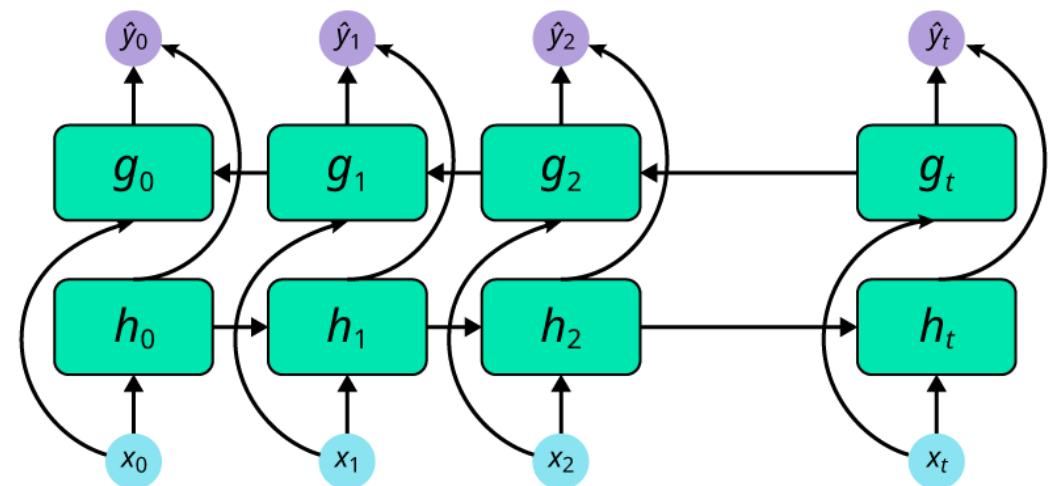
```



**True** means: output all the hidden states

# Advanced use of RNNs/LSTMs

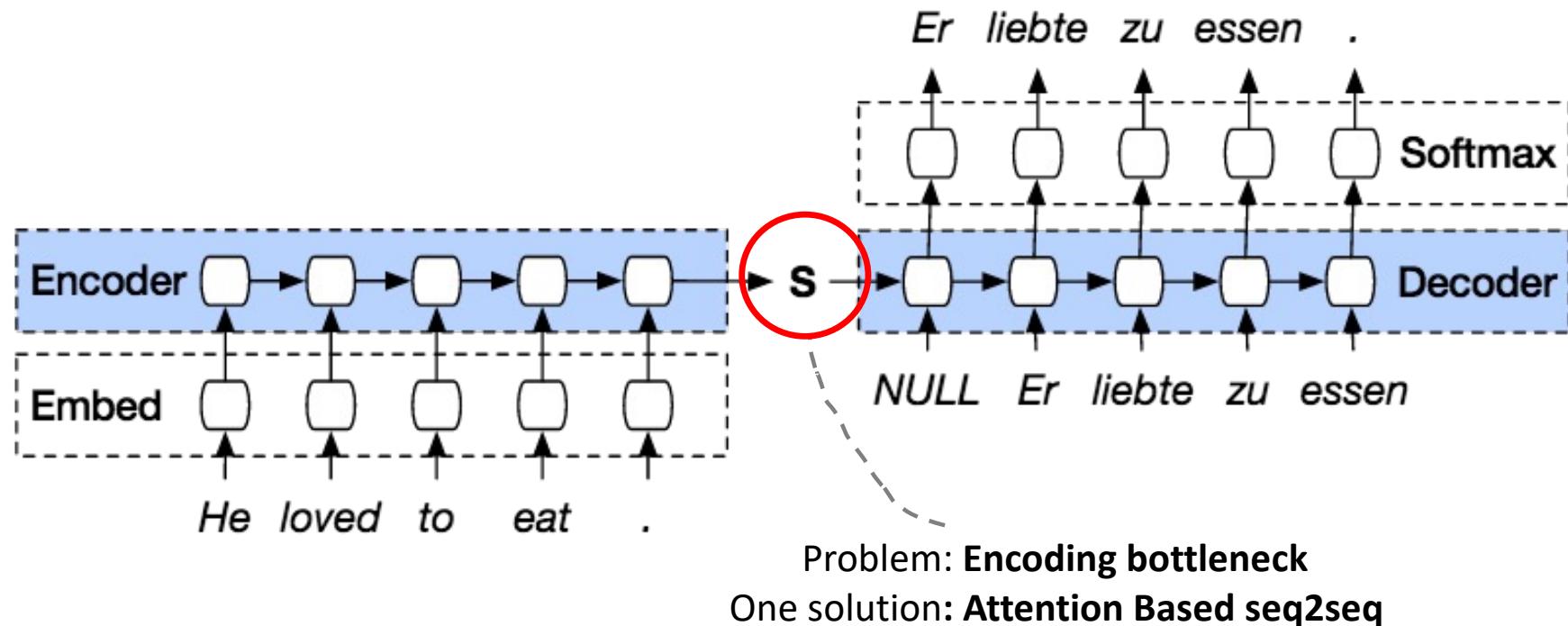
- **Bidirectional RNNs:** process a sequence in both directions, capturing patterns that may be missed by the chronological-order version alone.



```
from keras.layers import Bidirectional, LSTM
...
model.add(Bidirectional(LSTM(32)))
...
```

# Example Tasks: Neural Machine Translation (NMT)

- Seq2seq [Sutskever et al, 2014][Cho et al, 2014]



# Example Tasks: Image Caption Generation

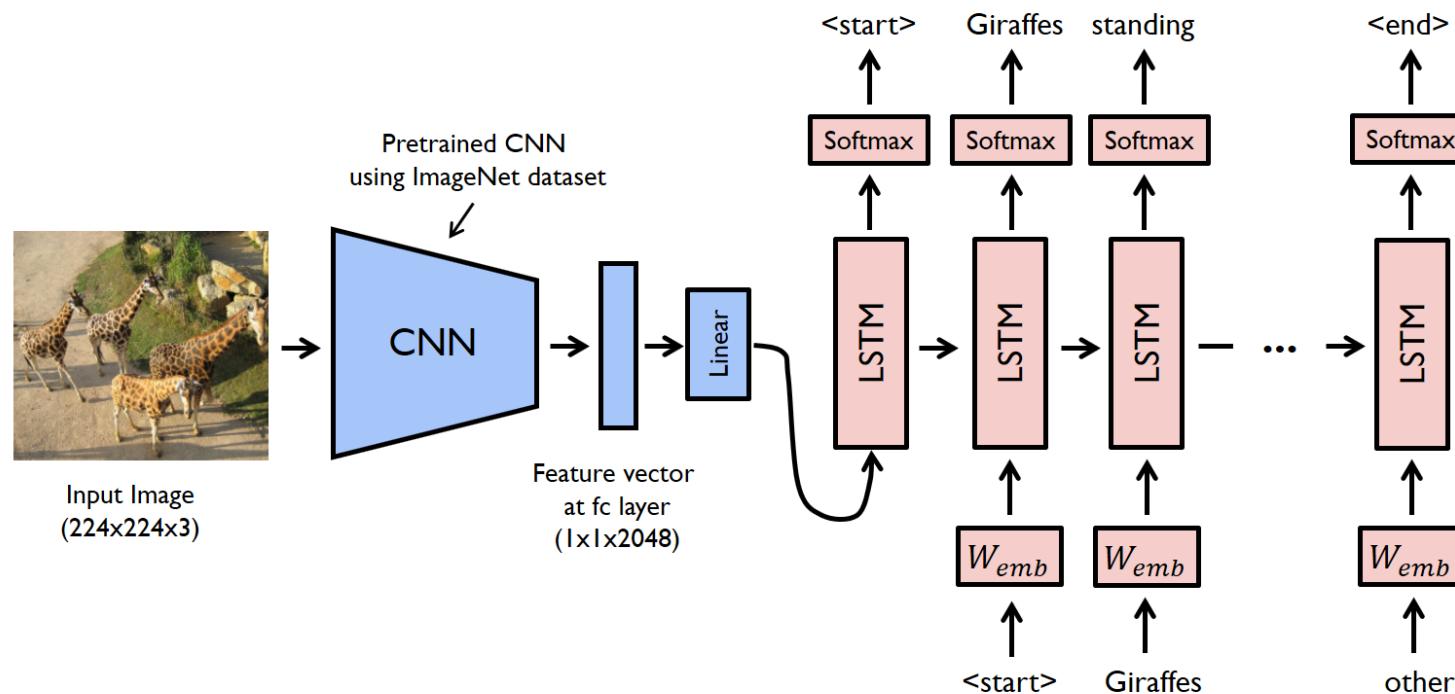


Image from: <https://www.analyticsvidhya.com/blog/2018/04/solving-an-image-captioning-task-using-deep-learning/>

# Summary of LSTMs

- Beside the hidden state, also maintain a **cell state** to store **long-term information**.
- Use **gates** to control the flow of information
  - **Forget** gate: gets rid of irrelevant **old** information
  - **Input** gate: stores relevant information from **current** input
  - **Output** gate: **output** a filtered version of the cell state
- Backpropagation through time with **uninterrupted gradient flow**, to avoid the vanishing/exploding gradient problem.

# Reference for Topic 6

- Blog by Colah: Understanding LSTM Networks.  
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- Blog by Michael Phi: Illustrated Guide to LSTM's and GRU's: A step to step explanation. <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>
- Video lecture by Alexander Amini: MIT course on Recurrent Neural Networks, YouTube.
- Lectures from University of Michigan:  
<https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html>
- Blogs by Nir Arbel: How do LSTM networks solve the problem of vanishing gradients: <https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577>

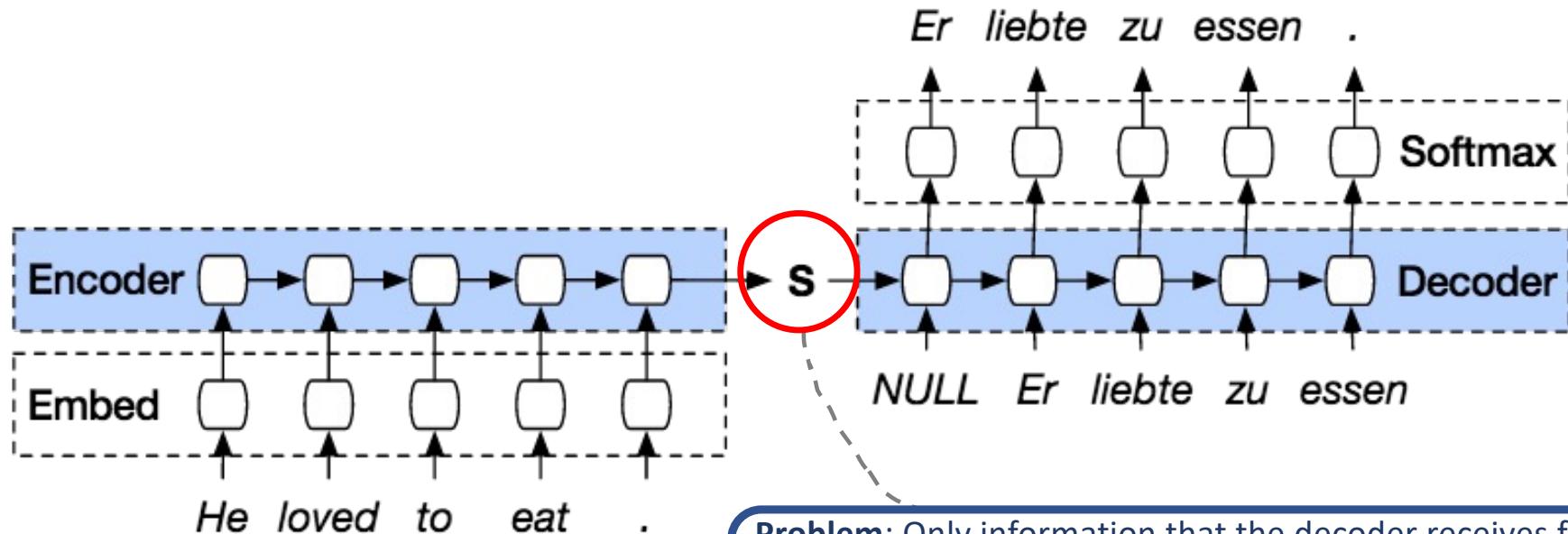
# Week 05: Attention Mechanism

Dr. Hongping Cai  
Department of Computer Science  
University of Bath

# Topic 1: Attention Mechanism

# Neural Machine Translation (NMT)

- Seq2seq [Sutskever et al, 2014][Cho et al, 2014]



**Problem:** Only information that the decoder receives from the encoder is the last encoder hidden state – **encoding bottleneck**. It doesn't work well when the sequence is long.  
**Solution:** seq2seq + **attention**

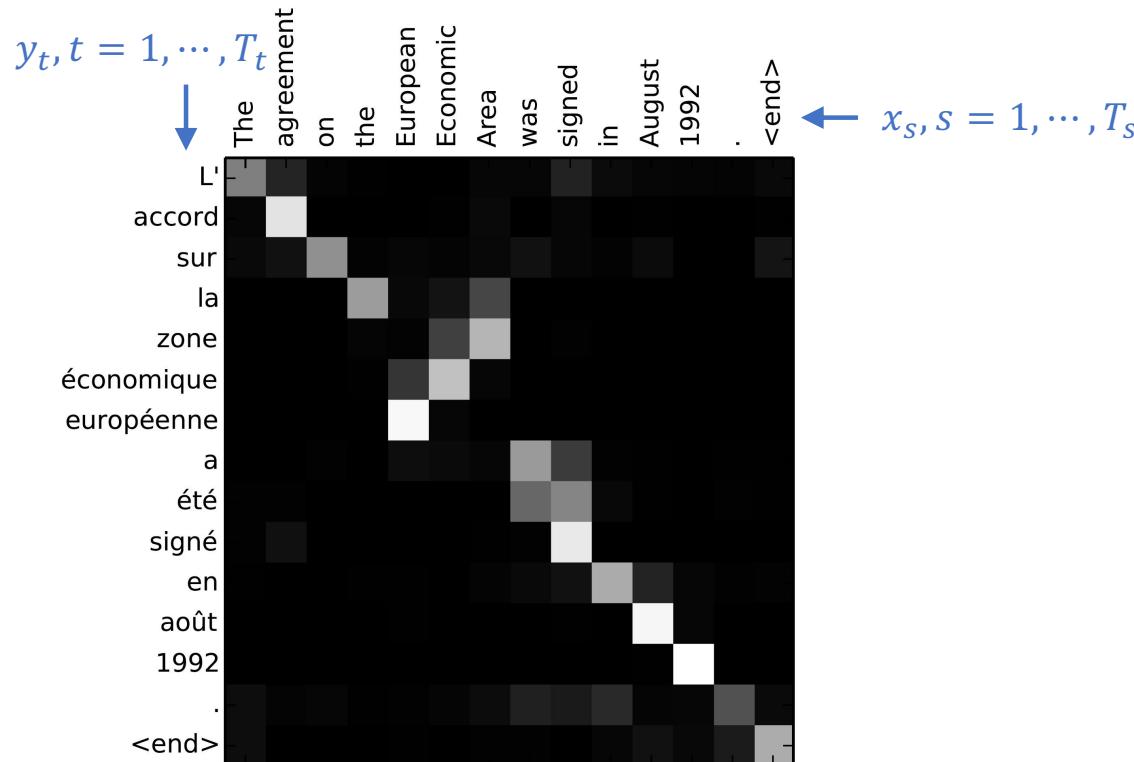
# Intuition of Attention Mechanism

Convolutional networks (LeCun, 1998), also known as convolutional neural networks, or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which can be thought of as a 2-D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name “convolutional neural network” indicates that the network employs a mathematical operation called **convolution**. Convolution is a specialized kind of linear operation. *Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.*

In this chapter, we first describe what convolution is. Next, we explain the motivation behind using convolution in a neural network. We then describe an operation called **pooling**, which almost all convolutional networks employ. Usually, the operation used in a convolutional neural network does not correspond precisely to the definition of convolution as used in other fields, such as engineering or pure mathematics. We describe several variants on the convolution function that are widely used in practice for neural networks. We also show how convolution

**Attention in translation: modelling the contextual relations by selectively focusing on parts of the source sentence.**

# Intuition of Attention Mechanism



$\alpha_{ts}$ : The amount of “attention” the target word  $y_t$  should pay to the source word  $x_s$ .

# Seq2seq

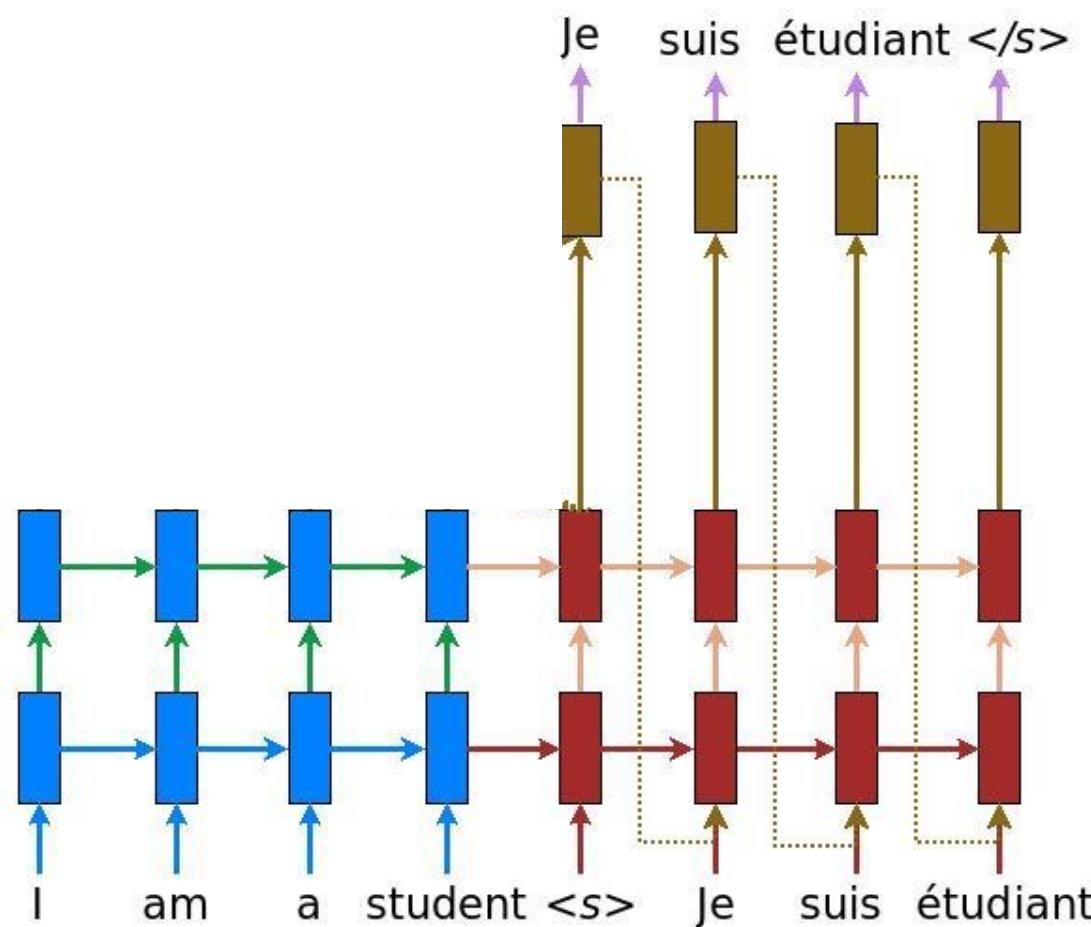


Image from: <https://github.com/tensorflow/nmt>

# Seq2seq + Attention [Bahdanau et al, 2014]

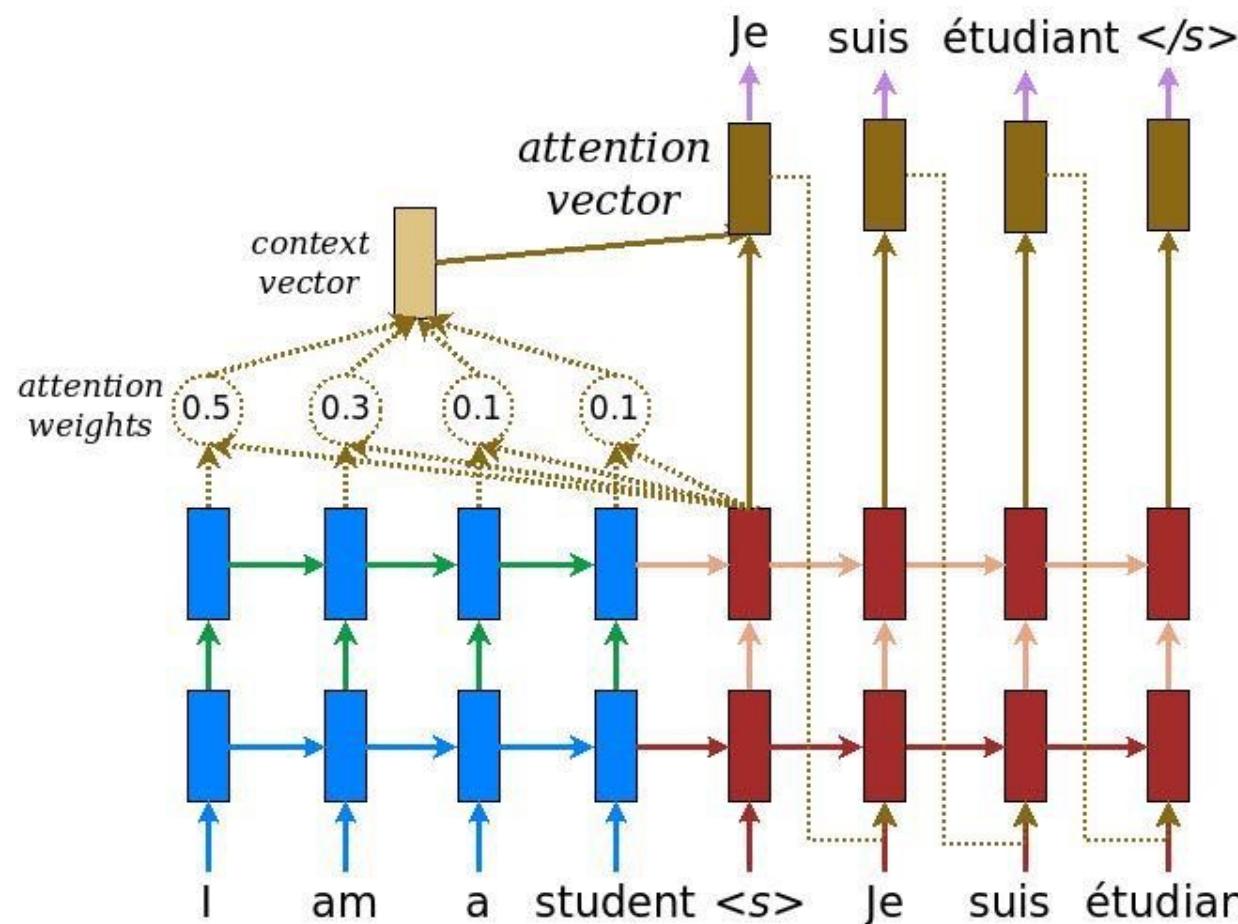
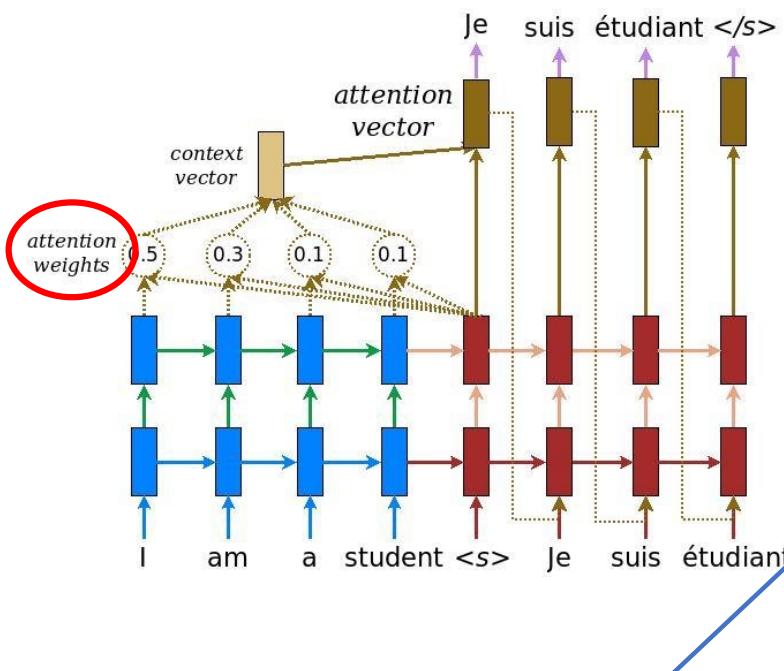


Image from: <https://github.com/tensorflow/nmt>

[Bahdanau et al, 2014] Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

# STEP1: Attention Weight

- **Attention weight** ( $\alpha_{ts}$ ) measures how much relevance between each source state ( $\bar{h}_s$ ) with the target state ( $h_t$  ).



$$\text{score}(h_t, \bar{h}_s) = h_t^T \cdot \bar{h}_s$$

$$\alpha_{ts} = \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'} \exp(\text{score}(h_t, \bar{h}_{s'}))}$$

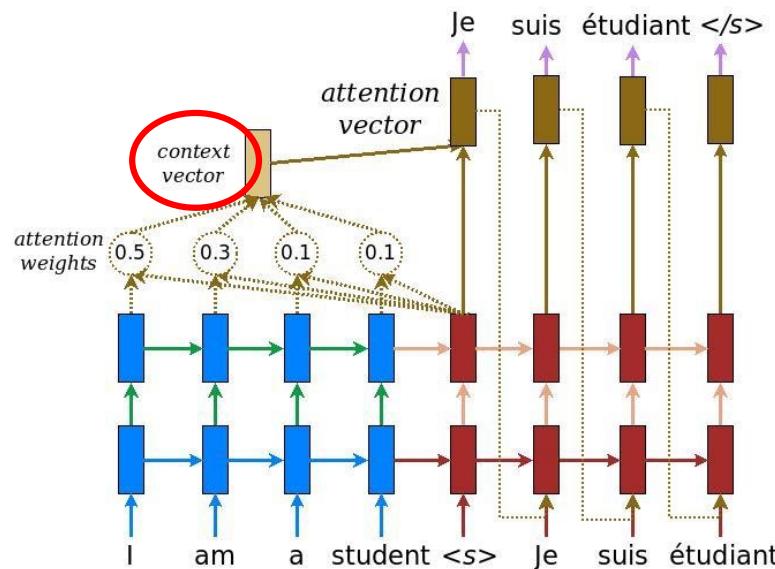
There are various choices of scoring function,  
e.g., dot-product, additive, general/multiplicative, etc.

$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^T W \bar{h}_s & [\text{Luong's multiplicative style}] \\ v_a^T \tanh(W_1 h_t + W_2 \bar{h}_s) & [\text{Bahdanau's additive style}] \end{cases}$$

Image from: <https://github.com/tensorflow/nmt>

# STEP 2: Context vector

- **Context vector ( $c_t$ )** is the weighted average of the source states



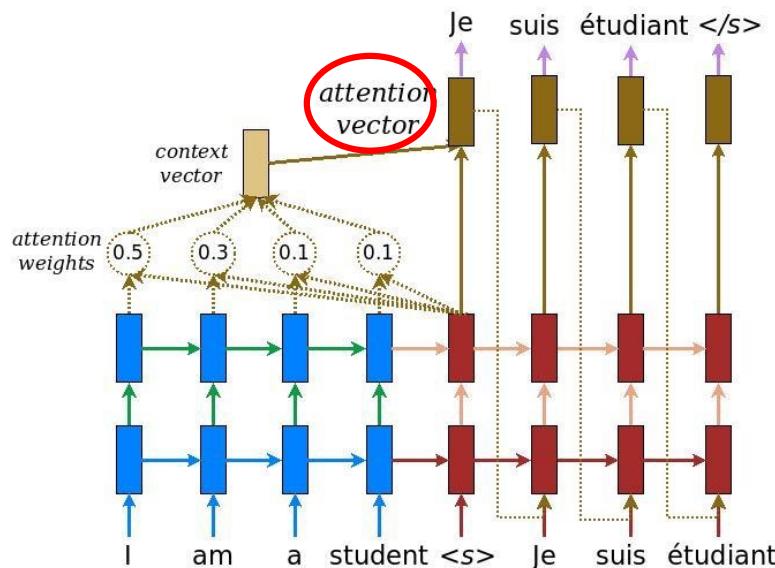
$$c_t = \sum_s \alpha_{ts} \bar{h}_s$$

# STEP 3: Attention vector

- **Attention vector ( $a_t$ )** is yielded by combining the context vector with the current target hidden state.

$$a_t = f(c_t, h_t) = \tanh(W_c[c_t; h_t])$$

The output vector (with attention):  $\hat{y}_t = g(W_a a_t)$



The output vector (without attention):  $\hat{y}_t = g(W_{hy} h_t)$

### German-English translations

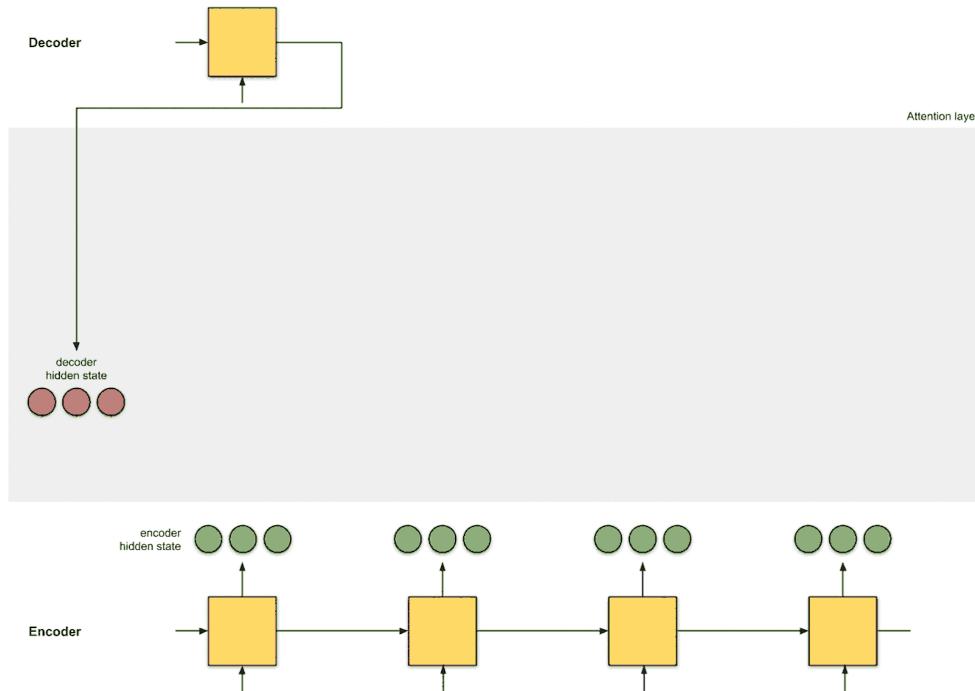
	src	In einem Interview sagte Bloom jedoch , dass er und Kerr sich noch immer lieben .
Seq2seq	ref	However , in an interview , Bloom has said that he and <i>Kerr</i> still love each other .
attention → best	best	In an interview , however , Bloom said that he and <i>Kerr</i> still love .
Seq2seq → base	base	However , in an interview , Bloom said that he and <i>Tina</i> were still <unk> .
	src	Wegen der von Berlin und der Europäischen Zentralbank verhängten strengen Sparpolitik in Verbindung mit der Zwangsjacke , in die die jeweilige nationale Wirtschaft durch das Festhalten an der gemeinsamen Währung genötigt wird , sind viele Menschen der Ansicht , das Projekt Europa sei zu weit gegangen
Seq2seq	ref	The <i>austerity imposed by Berlin and the European Central Bank , coupled with the straitjacket</i> imposed on national economies through adherence to the common currency , has led many people to think Project Europe has gone too far .
attention → best	best	Because of the strict <i>austerity measures imposed by Berlin and the European Central Bank in connection with the straitjacket</i> in which the respective national economy is forced to adhere to the common currency , many people believe that the European project has gone too far .
Seq2seq → base	base	Because of the pressure <b>imposed by the European Central Bank and the Federal Central Bank with the strict austerity</b> imposed on the national economy in the face of the single currency , many people believe that the European project has gone too far .

# References

- Thang Luong. Neural Machine Translation (seq2seq) Tutorial.  
<https://github.com/tensorflow/nmt>
- Blog by Raimi Karim. Attn: Illustrated Attention.2022.  
<https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3>

# Topic 2: An Example for Attention-based Seq2Seq

# Example

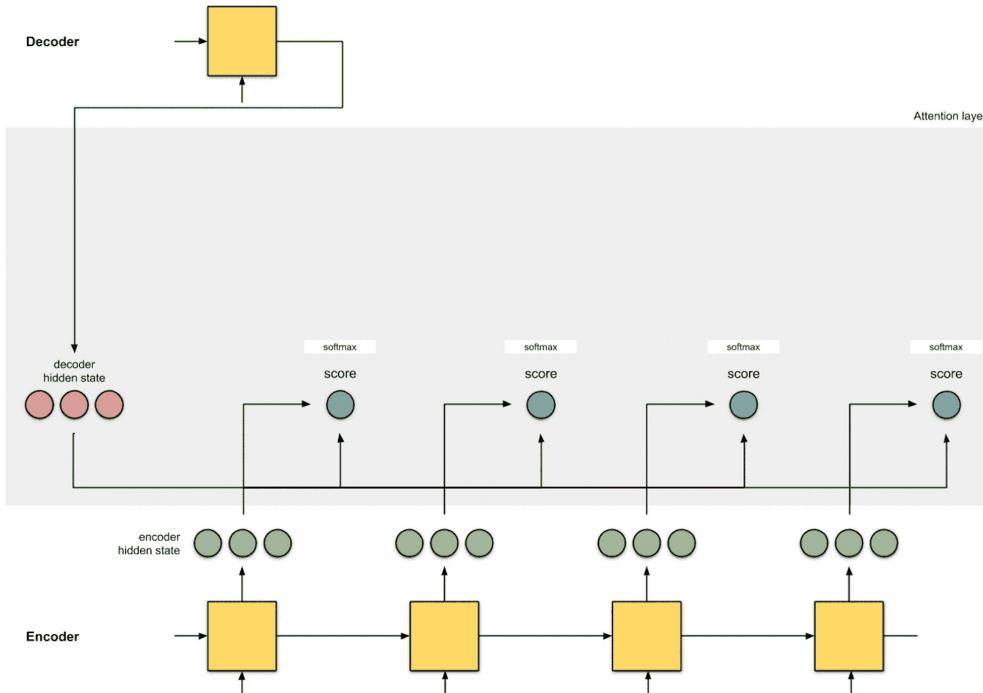


**dec\_h** = [10 5 10]

**enc\_h**

-----  
[0 1 1]  
[5 0 1]  
[1 1 0]  
[0 5 1]

**STEP 1:** The current target hidden state is compared with all source states to derive attention weights.



### Attention weight

$$\text{score}(h_t, \bar{h}_s) = h_t^T \cdot \bar{h}_s$$

$$\alpha_{ts} = \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'} \exp(\text{score}(h_t, \bar{h}_{s'}))}$$

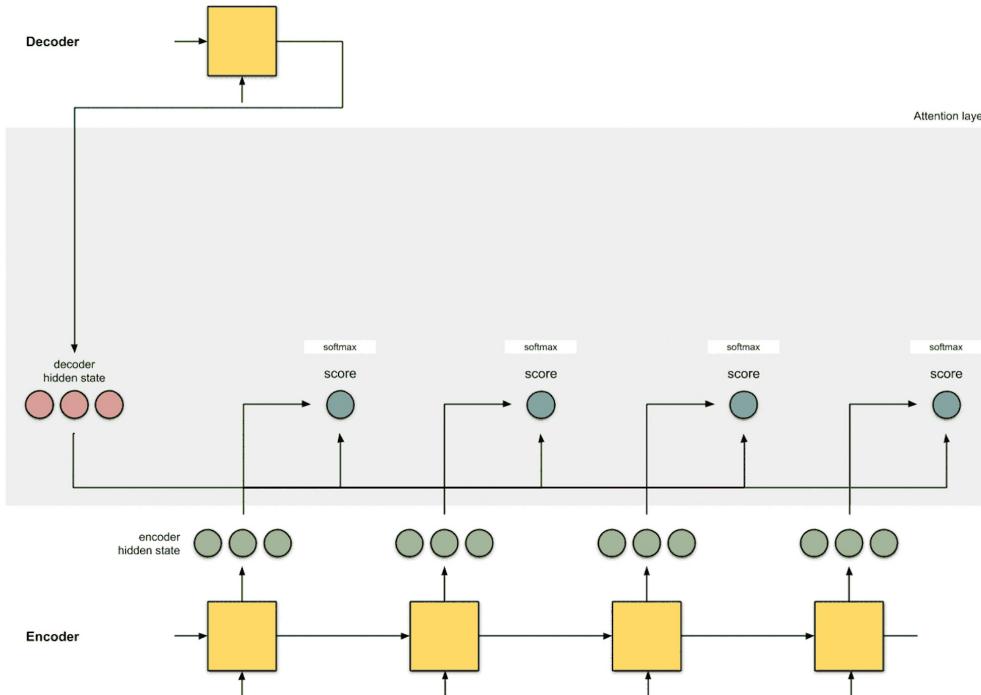
**dec\_h** = [10 5 10]

**enc\_h score attn\_w**

---

[0	1	1]
[5	0	1]
[1	1	0]
[0	5	1]

**STEP 1:** The current target hidden state is compared with all source states to derive attention weights.



### Attention weight

$$\text{score}(h_t, \bar{h}_s) = h_t^T \cdot \bar{h}_s$$

$$\alpha_{ts} = \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'} \exp(\text{score}(h_t, \bar{h}_{s'}))}$$

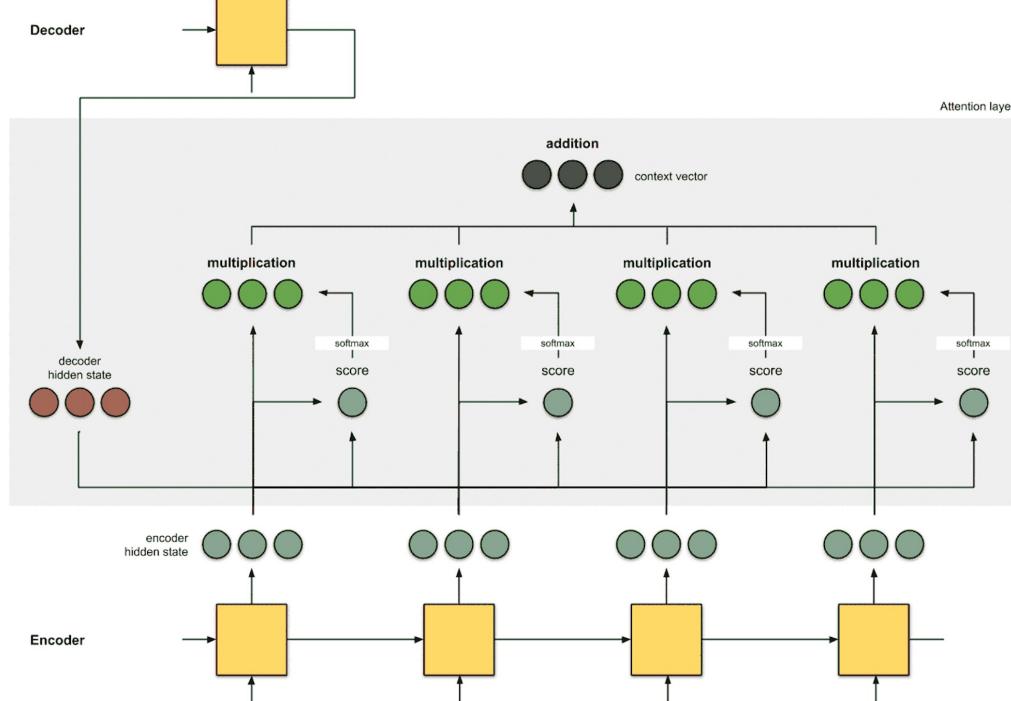
**dec\_h** = [10 5 10]

**enc\_h score attn\_w**

---

[0 1 1]	15	0
[5 0 1]	60	1
[1 1 0]	15	0
[0 5 1]	35	0

**STEP 2:** Multiply each encoder hidden state by its attention weight, then sum these vectors up.



**Context vector**

$$c_t = \sum_s \alpha_{ts} \bar{h}_s$$

$$\text{dec\_h} = [10 \ 5 \ 10]$$

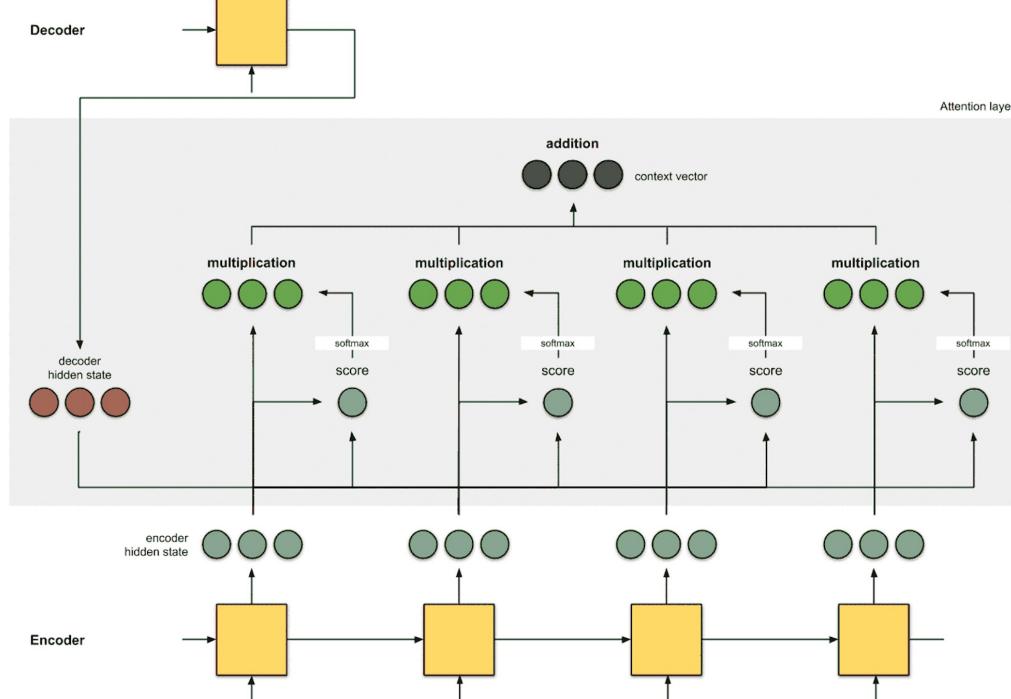
**enc\_h score attn\_w alignment**

---

[0 1 1]	15	0	[0 0 0]
[5 0 1]	60	1	[5 0 1]
[1 1 0]	15	0	[0 0 0]
[0 5 1]	35	0	[0 0 0]

**Context** =

**STEP 2:** Multiply each encoder hidden state by its attention weight, then sum these vectors up.



**Context vector**

$$c_t = \sum_s \alpha_{ts} \bar{h}_s$$

$$\text{dec\_h} = [10 \ 5 \ 10]$$

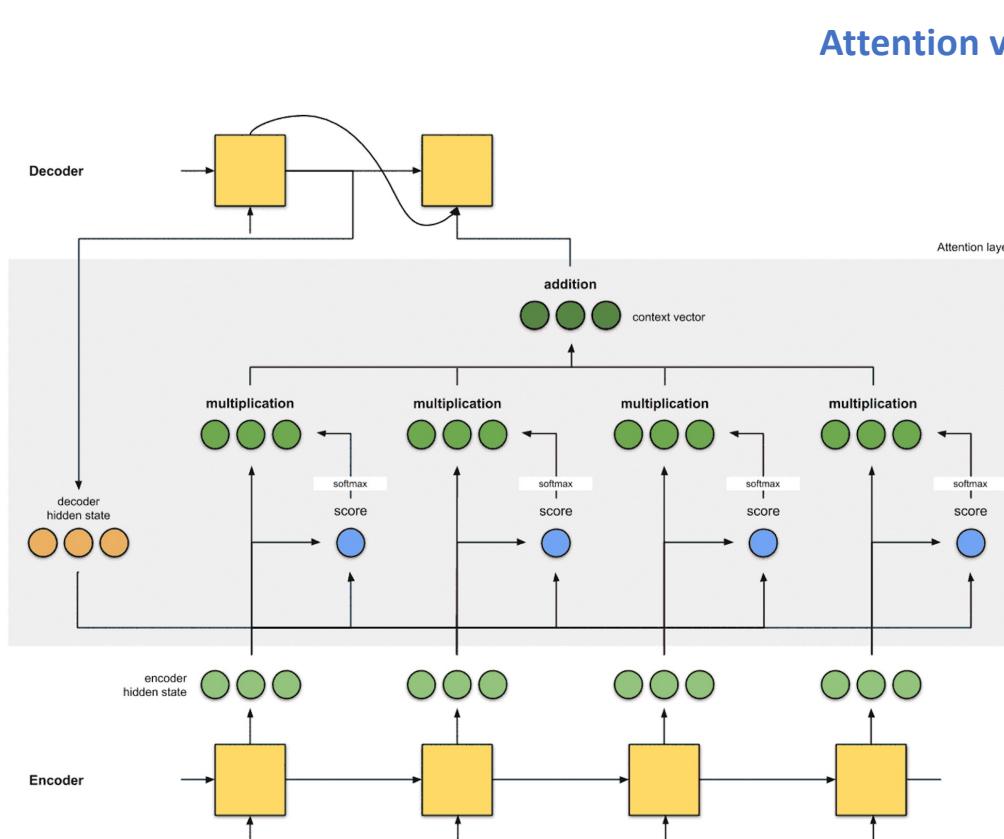
**enc\_h score attn\_w alignment**

---

[0 1 1]	15	0	[0 0 0]
[5 0 1]	60	1	[5 0 1]
[1 1 0]	15	0	[0 0 0]
[0 5 1]	35	0	[0 0 0]

$$\text{Context} = [5 \ 0 \ 1]$$

## STEP 3: Feed the context vector into the decoder.



**dec\_h** = [10 5 10]

**enc\_h score attn\_w alignment**

---

[0 1 1]	15	0	[0 0 0]
[5 0 1]	60	1	[5 0 1]
[1 1 0]	15	0	[0 0 0]
[0 5 1]	35	0	[0 0 0]

**Context** = [5 0 1]

**dec\_h\_context** = [5 0 1 10 5 10]

# References

- Thang Luong. Neural Machine Translation (seq2seq) Tutorial.  
<https://github.com/tensorflow/nmt>
- Blog by Raimi Karim. Attn: Illustrated Attention.2022.  
<https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3>

# Topic 3: Three Attention-Based Seq2Seq Models

Bahdanau et al. (2014)

- Encoder:
- Decoder:
- Score function:
- Context vector integrated to decoder:

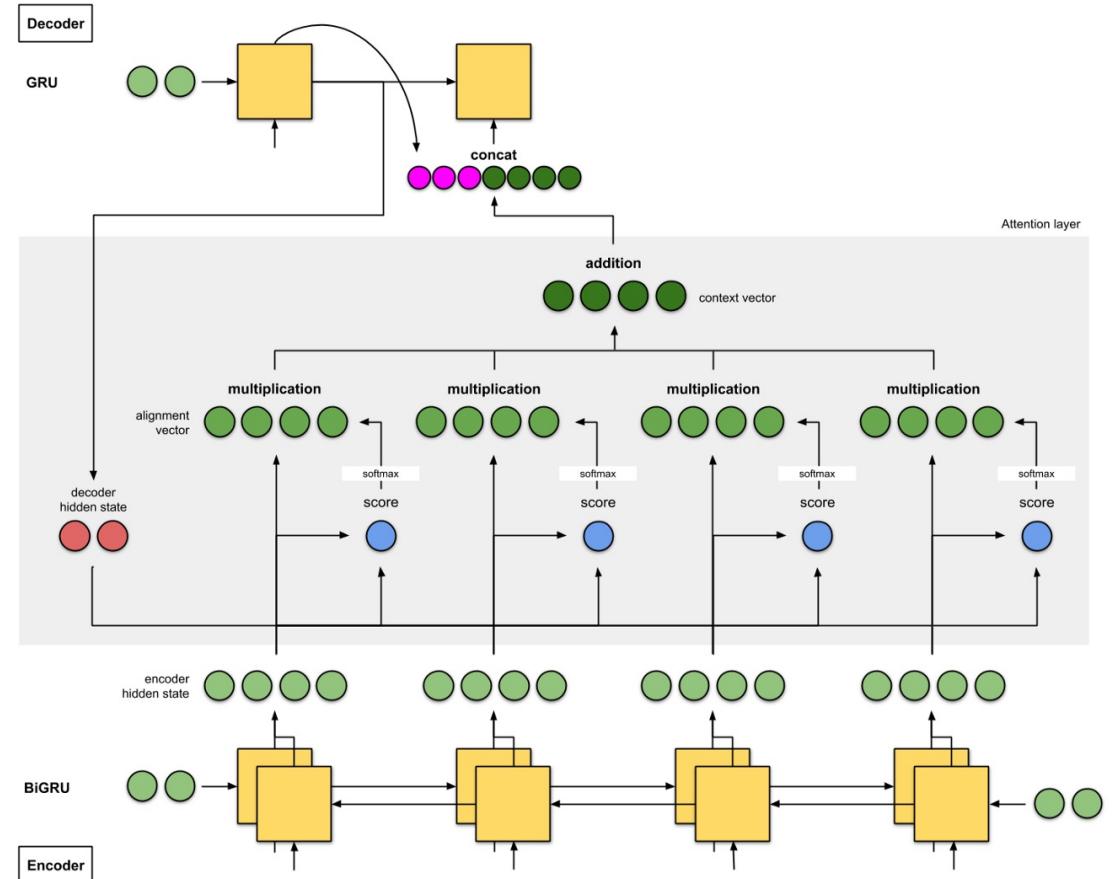
Luong et al. (2015)

Google's NMT (2016)

# Bahdanau et al. (2014)

- **Encoder:** Bidirectional GRU
- **Decoder:** GRU
- **Score function:** concat
- **Context vector integrated to decoder:** concat

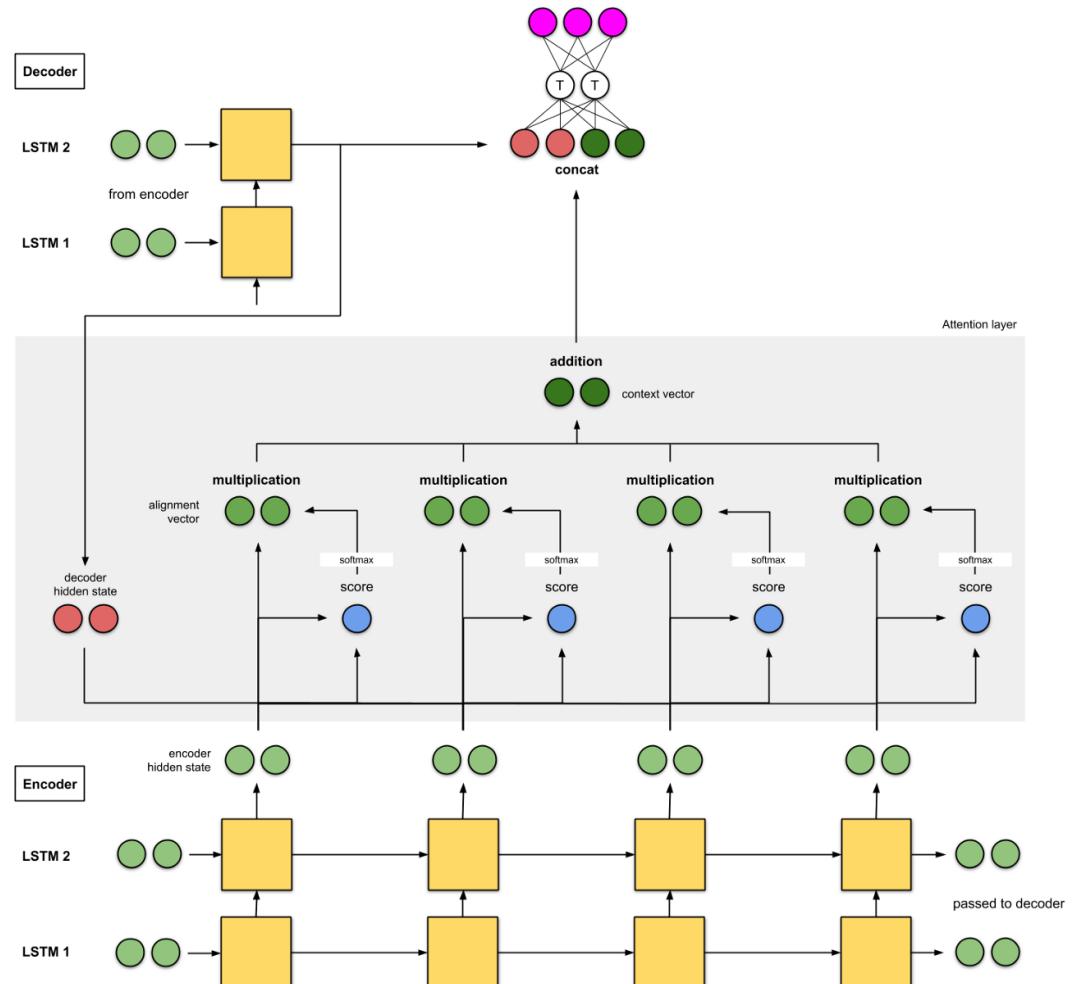
$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & \text{dot} \\ h_t^\top W_a \bar{h}_s & \text{general} \\ v_a^\top \tanh(W_a[h_t; \bar{h}_s]) & \text{concat} \end{cases}$$



# Luong et al. (2015)

- **Encoder:** two-stacked LSTM
- **Decoder:** two-stacked LSTM
- **Score function:** concat; dot; location based; general
- **Context vector integrated to decoder:** concat + dense layer

$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & \text{dot} \\ h_t^\top W_a \bar{h}_s & \text{general} \\ v_a^\top \tanh(W_a[h_t; \bar{h}_s]) & \text{concat} \end{cases}$$

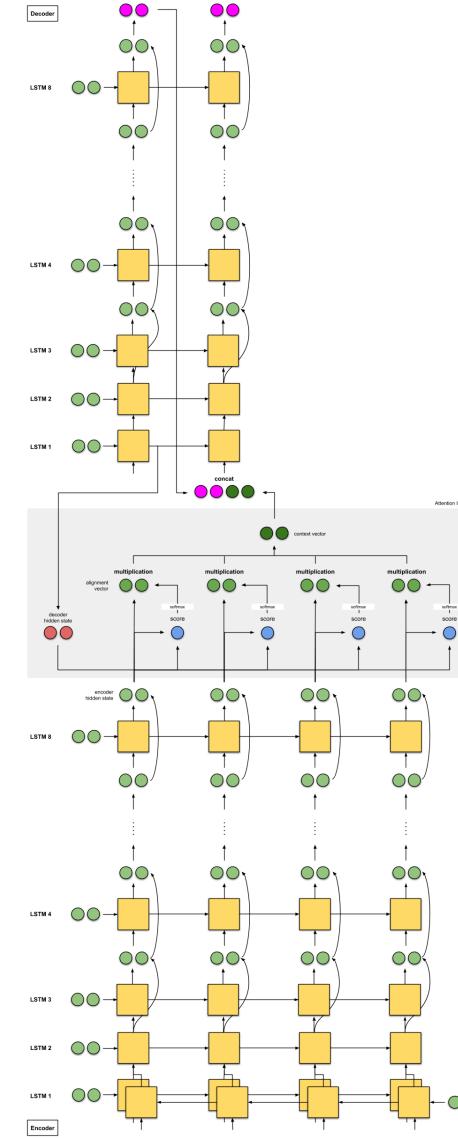


Luong et al. Effective approaches to Attention-based Neural Machine Translation, EMNLP 2015.  
 Image from: <https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3>

# Google's NMT (2016)

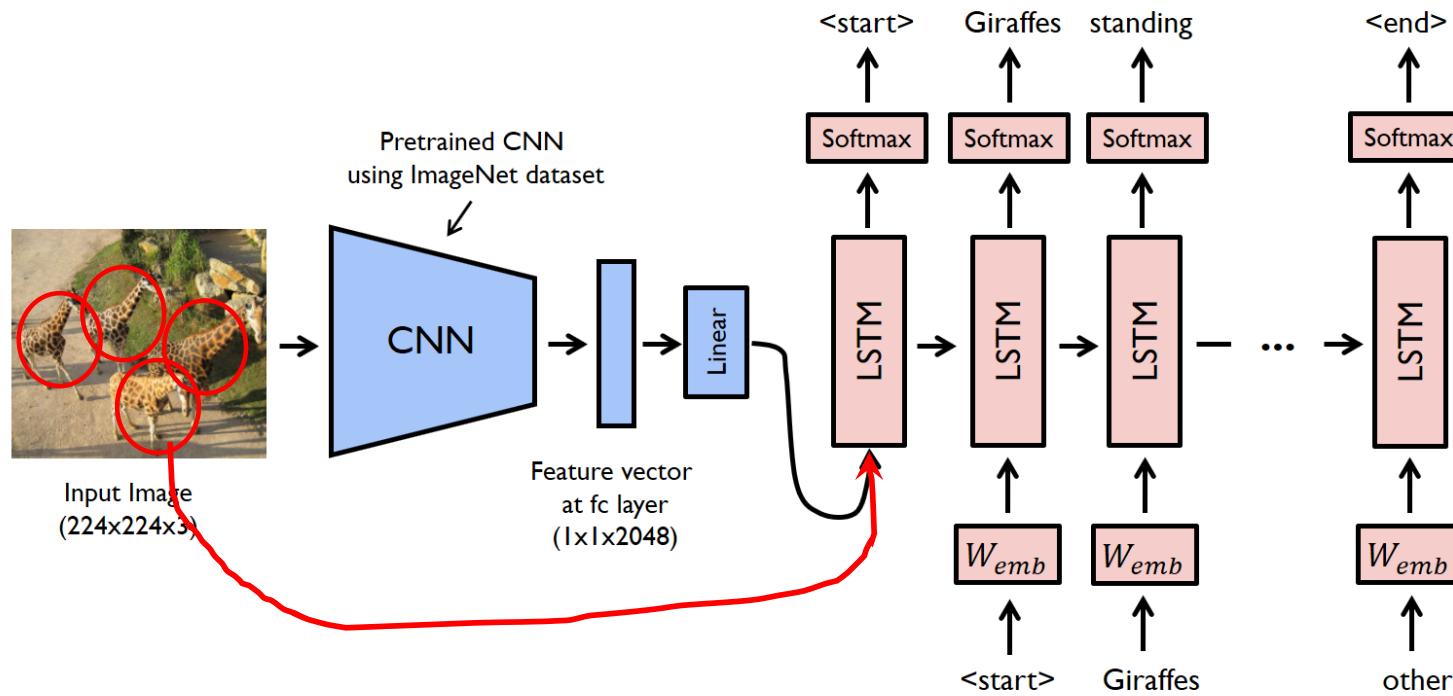
- **Encoder:** 8-stacked LSTM, bidirectional (first layer), residual connections
- **Decoder:** 8-stacked LSTM
- **Score function:** concat
- **Context vector integrated to decoder:** concat

$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & \text{dot} \\ h_t^\top W_a \bar{h}_s & \text{general} \\ v_a^\top \tanh(W_a[h_t; \bar{h}_s]) & \text{concat} \end{cases}$$



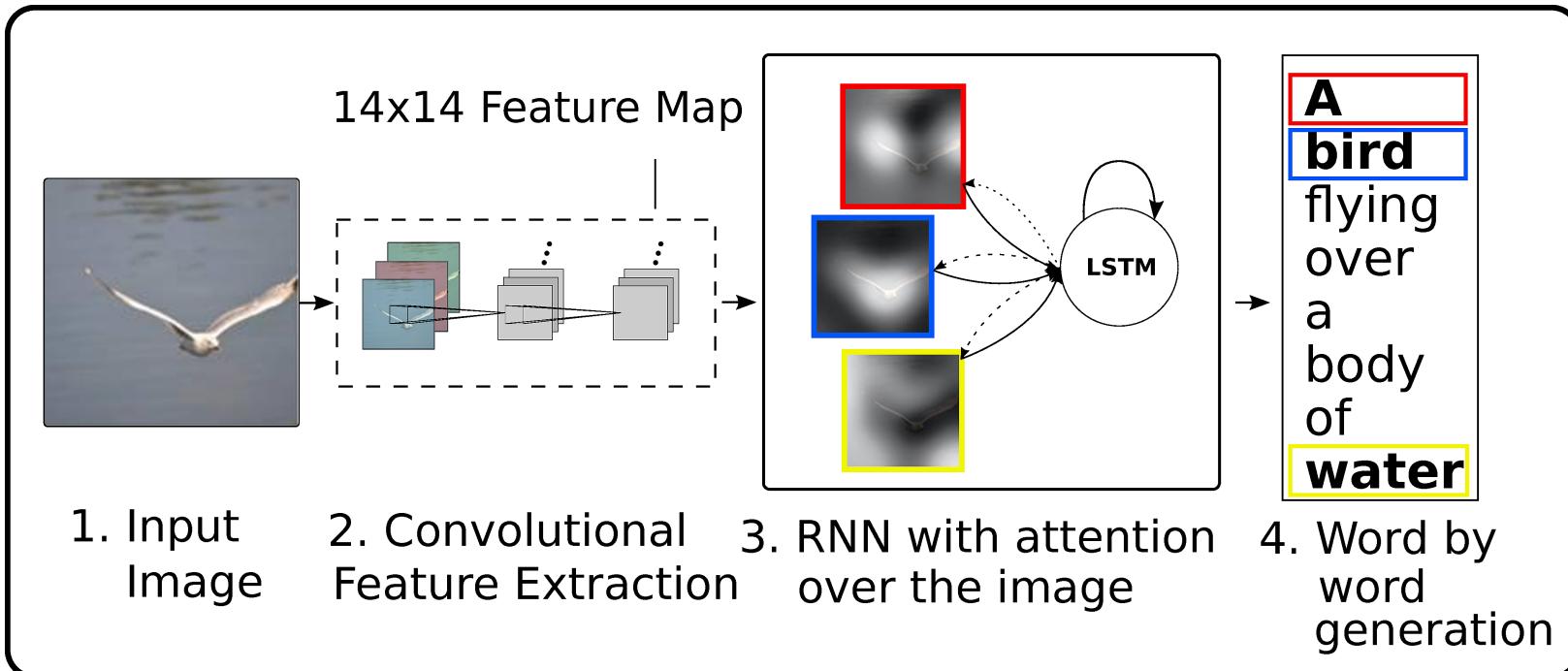
Google's neural machine translation system: Bridge the gap between human and machine translation, 2016.  
Image from: <https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3>

# Attention is not limited for NMT ...



The decoder focuses its **attention** at certain spatial location when generating each word.

# Attention-based Image Captioning [Xu-ICML15]



# Attention-based Image Captioning [Xu-ICML15]

Figure 4. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



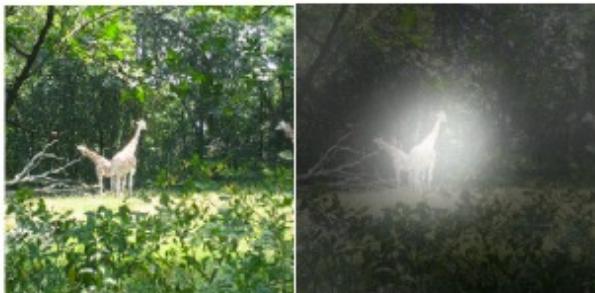
A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

# Attention-based Image Captioning [Xu-ICML15]

*Figure 5.* Examples of mistakes where we can use attention to gain intuition into what the model saw.



A large white bird standing in a forest.



A woman holding a clock in her hand.



A man wearing a hat and a hat on a skateboard.



A person is standing on a beach with a surfboard.



A woman is sitting at a table with a large pizza.



A man is talking on his cell phone while another man watches.

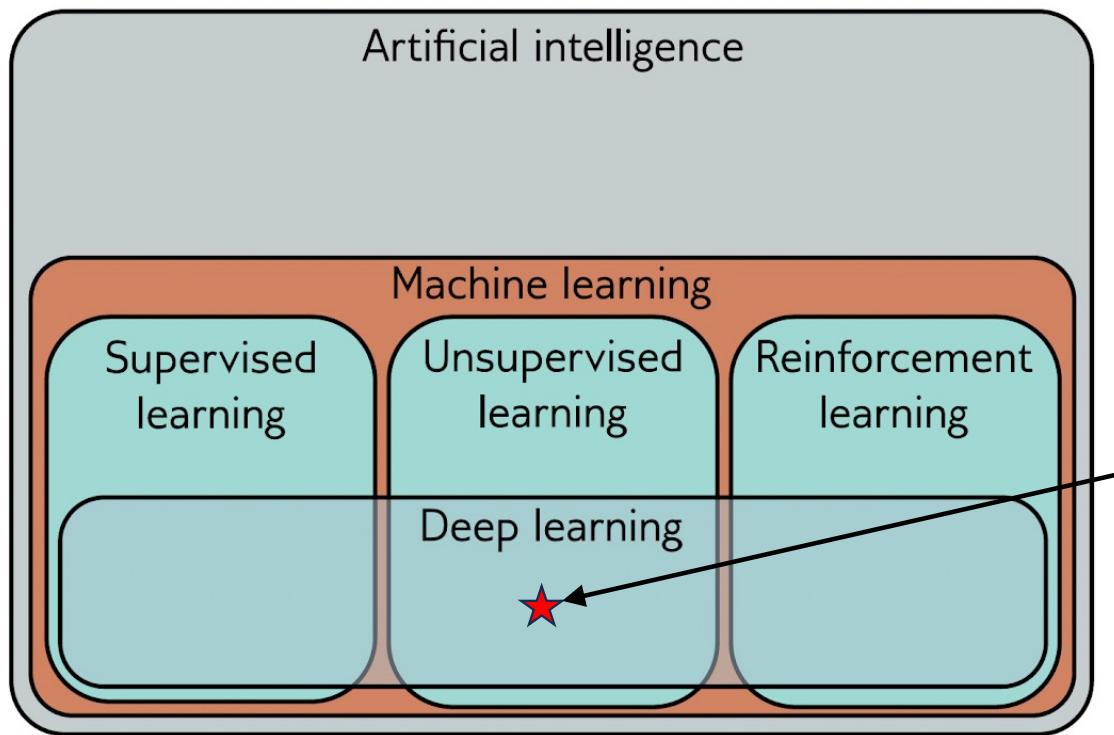
# References

- Thang Luong. Neural Machine Translation (seq2seq) Tutorial.  
<https://github.com/tensorflow/nmt>
- Blog by Raimi Karim. Attn: Illustrated Attention.2022.  
<https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3>

# Week 07: Generative Adversarial Networks (GANs)

Dr. Hongping Cai  
Department of Computer Science  
University of Bath

# Topic 1: Introduction to Deep Generative Models



- **Data:**  $X$  – data,  $Y$  – label
- **Goal:** Learn a mapping:  

$$Y = f(X)$$

**Supervised**

- **Data:**  $X$  – data
- **Goal:** Learn underlying structure of the data.

**Unsupervised**

# Discriminative models vs Generative models

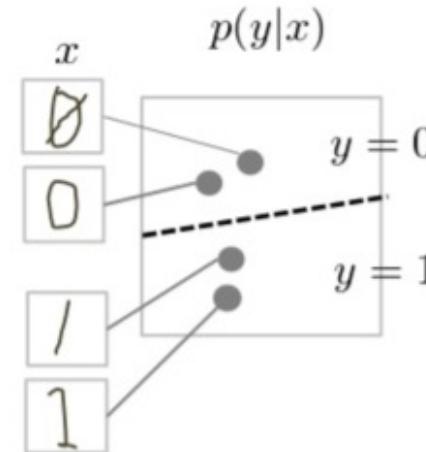
- Discriminative models

- Discriminate between different kind of data instance
- Care about boundary between classes only
- E.g., Linear regression, SVM, neural networks
- Model  $p(Y|X)$

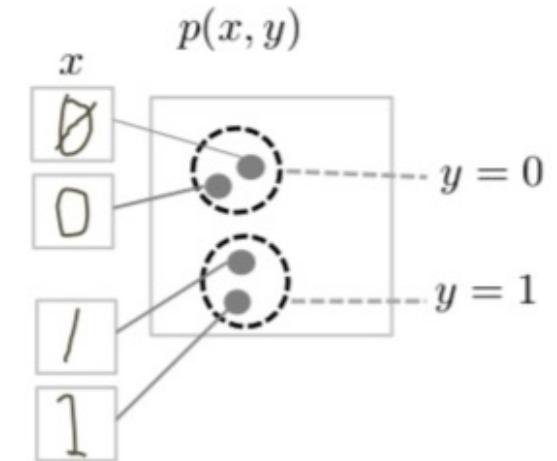
- Generative models

- Discover the distribution of training samples
- Care about entire distribution – harder, but more powerful
- E.g., Gaussian process, density estimation, HMM
- Model  $p(X, Y)$ , or just  $p(X)$  if no labels

- Discriminative Model



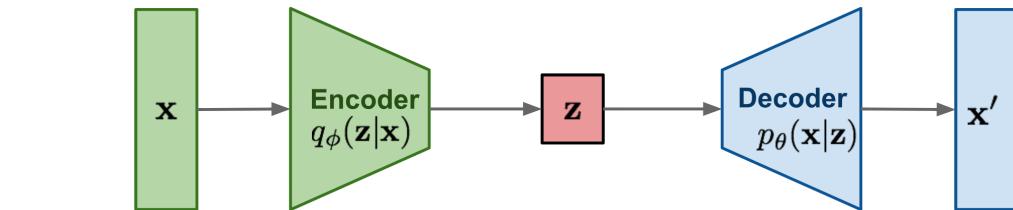
- Generative Model



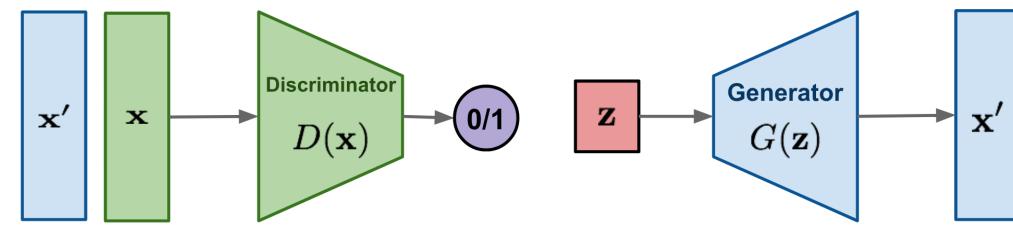
# Two Typical Deep Generative Models

Both are unsupervised methods. Neither of them explicitly learns  $p(x)$

## Variational Autoencoder (VAE)



## Generative Adversarial Network (GAN)



## Other deep generative models:

- Augoregressive models, e.g., PixelRNN, PixelCNN
- Normalizing flows
- Diffusion models

# Reference

- Lecture on Generative Adversarial Networks:  
<https://developers.google.com/machine-learning/gan/generative>  
Video lecture by Ava Soleimany: MIT course on Deep Generative Models, YouTube.
- Lecture by Rowel Atienza:  
[https://docs.google.com/presentation/d/13fiFibqjl9ps\\_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.g389666c0c0\\_0\\_5](https://docs.google.com/presentation/d/13fiFibqjl9ps_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.g389666c0c0_0_5)

# Topic 2: GANs

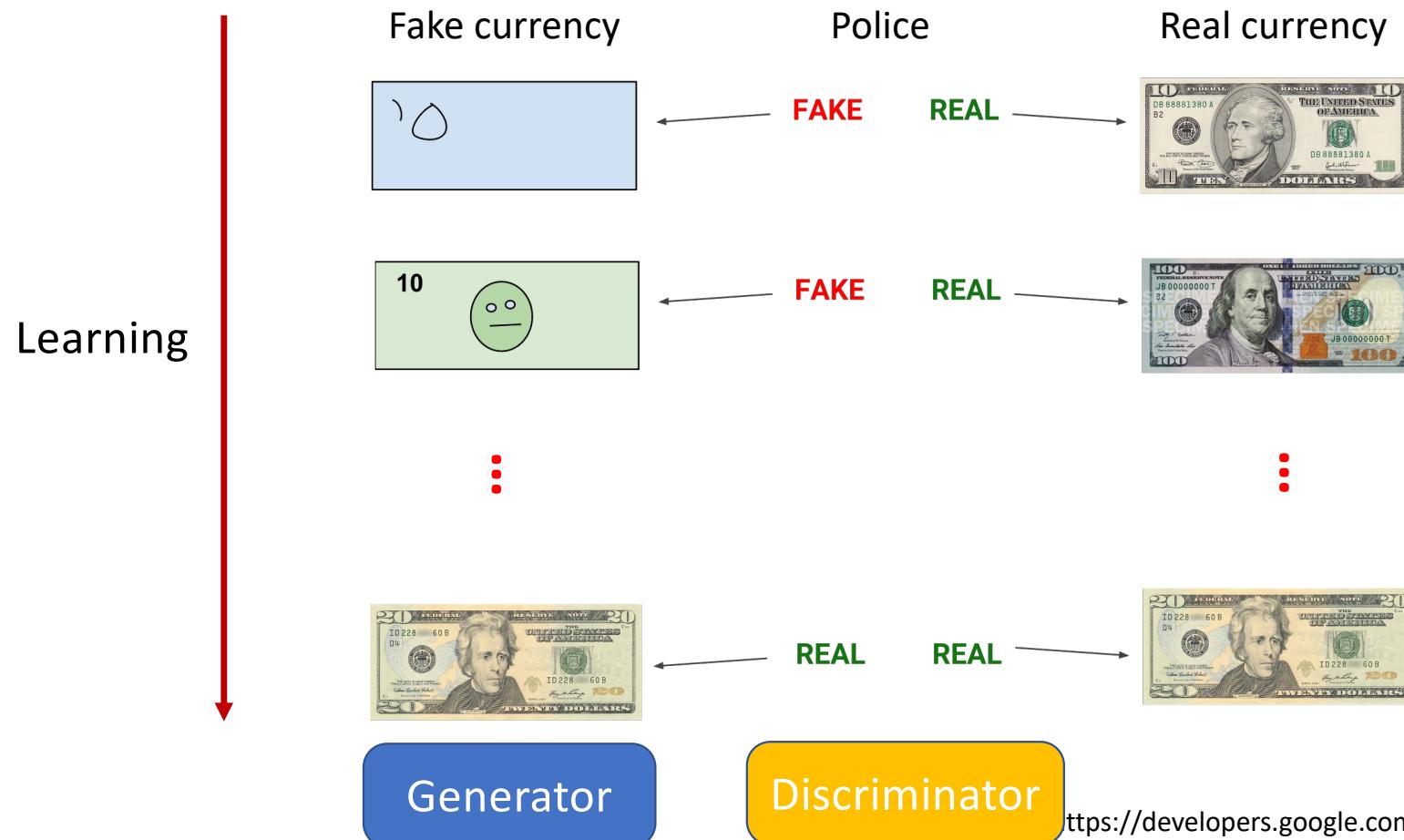
# Generative Adversarial Networks (GANs)

- GAN [Goodfellow-NIPS2014] is an unsupervised machine learning technique.
- GAN is a generative model to generate data with similar characteristics as the real data.



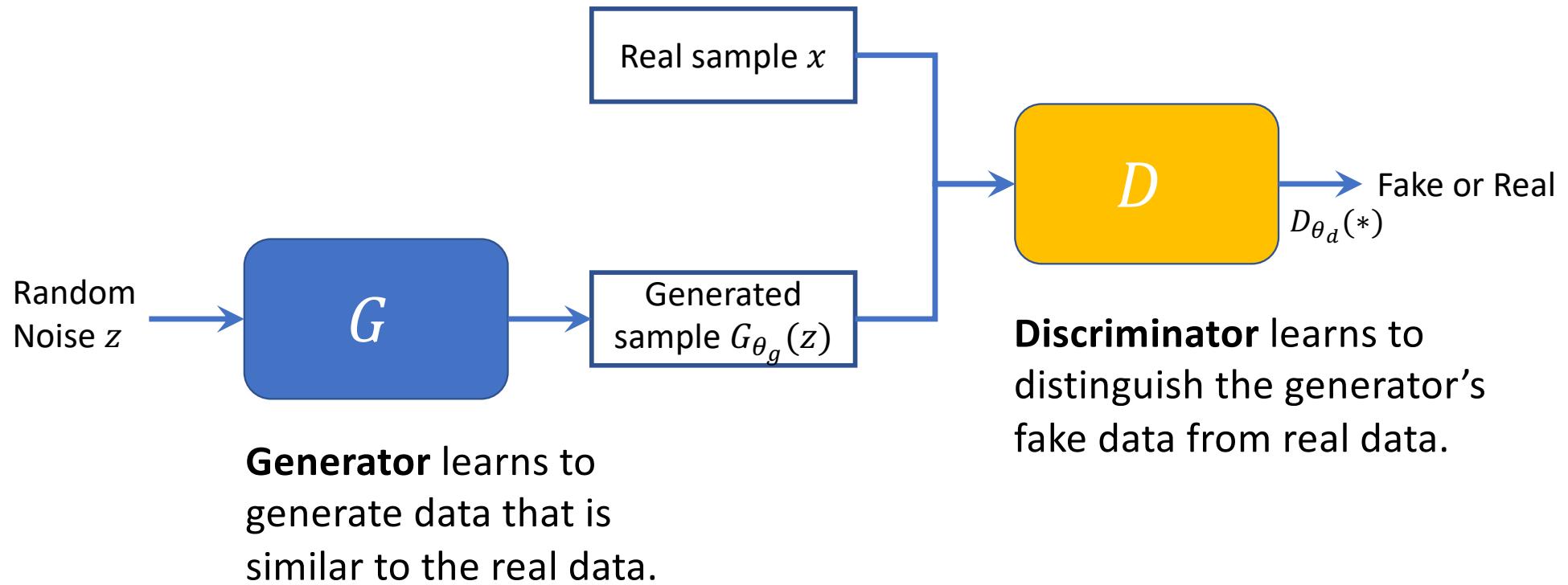
GAN generated faces by training on the CELEBA-HQ dataset [Karras-ICLR2018]

# Intuition behind GANs

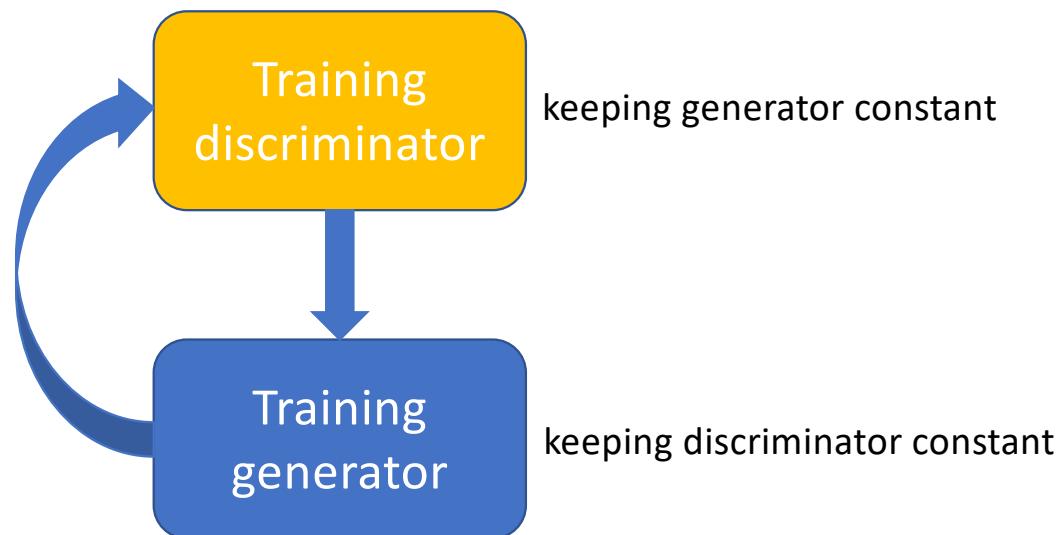


# GAN Architecture

- A GAN has two neural networks competing with each other.



# Training GANs – Alternating training



# Training Discriminator

- **Binary cross-entropy loss** for binary classification:

$$J(W) = -\frac{1}{N} \sum_{n=1}^N (y^{(n)} \log f_W(x^{(n)}) + (1 - y^{(n)}) \log(1 - f_W(x^{(n)})))$$

actual                              prediction  
 $\{0, 1\}$                                $[0, 1]$

# Training Discriminator

- **Binary cross-entropy loss** for binary classification:

$$J(W) = -\frac{1}{N} \sum_{n=1}^N (y^{(n)} \log f_W(x^{(n)}) + (1 - y^{(n)}) \log(1 - f_W(x^{(n)})))$$

actual  
 $\{0, 1\}$

prediction  
 $[0, 1]$



$$J(W) =$$

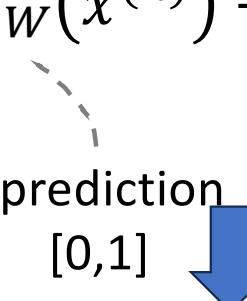
For all positive samples ( $y^{(n)} = 1$ )

For all negative samples ( $y^{(n)} = 0$ )

# Training Discriminator

- **Binary cross-entropy loss** for binary classification:

$$J(W) = -\frac{1}{N} \sum_{n=1}^N (y^{(n)} \log f_W(x^{(n)}) + (1 - y^{(n)}) \log(1 - f_W(x^{(n)})))$$

actual  
 $\{0, 1\}$       prediction  
 $[0, 1]$ 


$$J(W) = -\frac{1}{N_1} \sum_{y^{(n)}=1} \log f_W(x^{(n)}) - \frac{1}{N_0} \sum_{y^{(n)}=0} \log(1 - f_W(x^{(n)}))$$

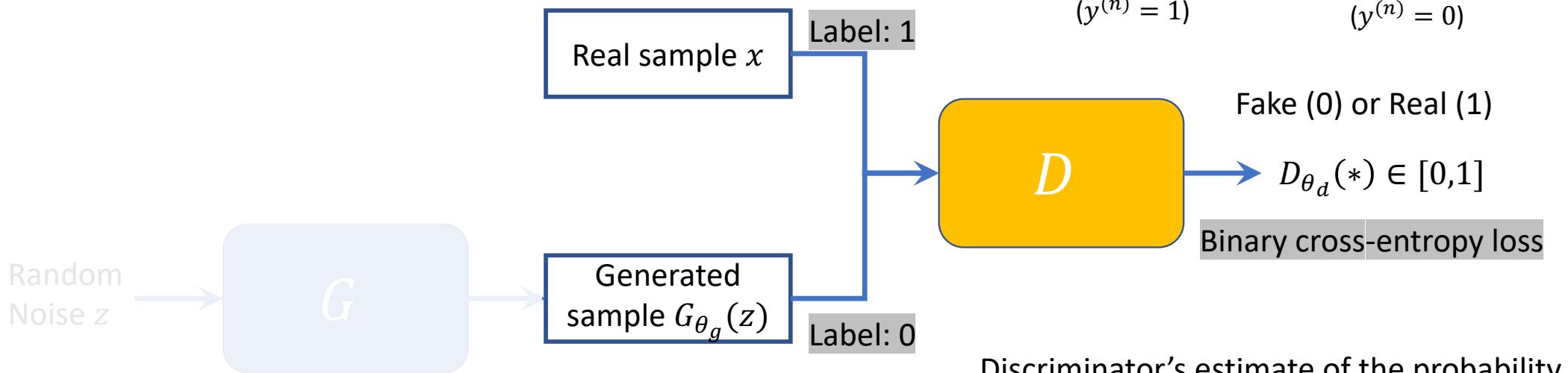



For all positive samples ( $y^{(n)} = 1$ )      For all negative samples ( $y^{(n)} = 0$ )

# Training Discriminator

$$\text{Binary cross-entropy loss: } -\frac{1}{N_1} \sum_{y^{(n)}=1} \log f_W(x^{(n)}) - \frac{1}{N_0} \sum_{y^{(n)}=0} \log(1 - f_W(x^{(n)}))$$

For all positive samples ( $y^{(n)} = 1$ )      For all negative samples ( $y^{(n)} = 0$ )

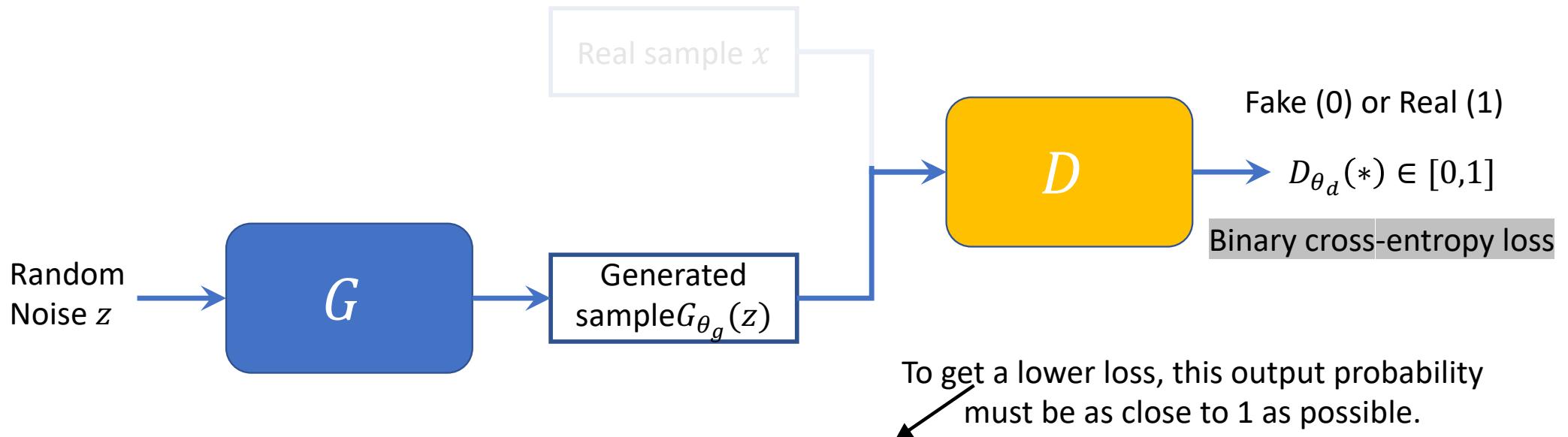


- The discriminator loss:  $\mathcal{L}_D = -E_x[\log D_{\theta_d}(x)] - E_z[\log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$
- It penalizes the misclassification.

Discriminator's estimate of the probability that the real sample is real

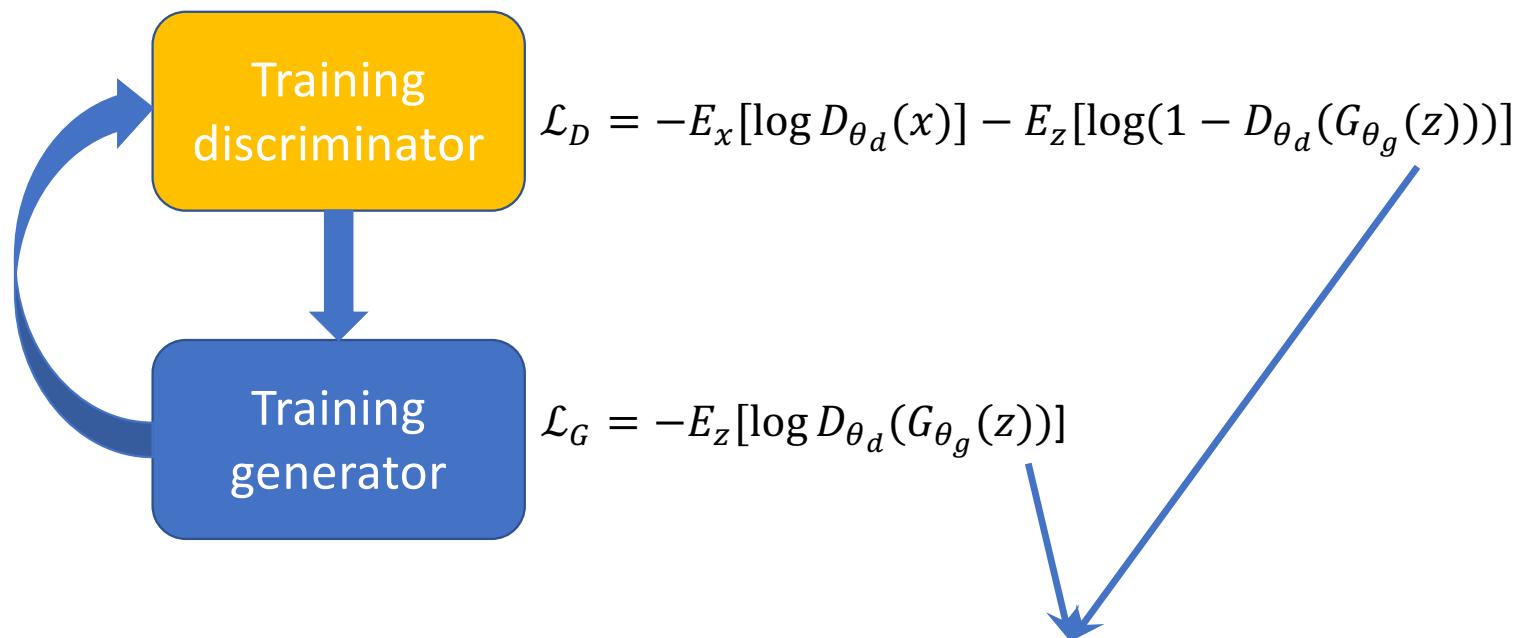


# Training Generator



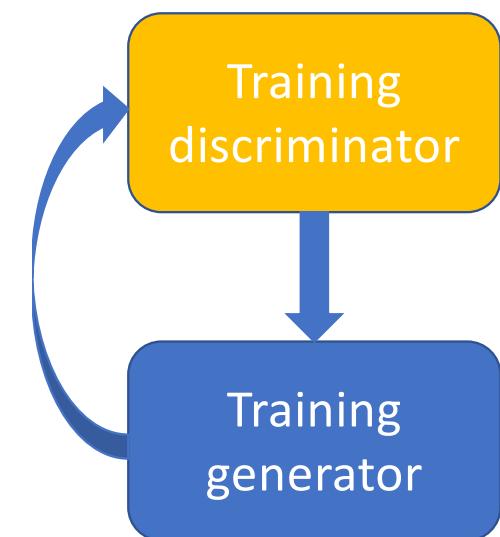
- The generator loss:  $\mathcal{L}_G = -E_z[\log D_{\theta_d}(G_{\theta_g}(z))]$
- It penalizes the generator for failing to fool the discriminator.
- During the back propagation, the discriminator's gradients are passed down to the Generator, however, its weight are frozen.

# Training GANs – Alternating training



**Objective:**  $\min_{\theta_g} \max_{\theta_d} E_x[\log D_{\theta_d}(x)] + E_z[\log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$

# Training GANs – Alternating training



- Step 1: Sample minibatch of  $B$  training samples  $x^{(1)}, x^{(2)}, \dots, x^{(B)}$
- Step 2: Sample minibatch of  $B$  random vectors  $z^{(1)}, z^{(2)}, \dots, z^{(B)}$
- Step 3: Update the generator parameters  $\theta_g$  by mini-batch gradient descent for one or more epochs.

$$\nabla_{\theta_g} \mathcal{L}_G = -\frac{1}{B} \nabla_{\theta_g} \sum_{i=1}^B \log D_{\theta_d}(G_{\theta_g}(z^{(i)}))$$

- Step 4: Update the discriminator parameters  $\theta_d$  by mini-batch gradient descent for one or more epochs.

$$\nabla_{\theta_d} \mathcal{L}_D = -\frac{1}{B} \nabla_{\theta_d} \sum_{i=1}^B [\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)})))]$$

- Repeat Step 3 and Step 4 for fixed number of epochs

# Challenges of training GANs

- **Mode Collapse:** the generator learns to produce a limited variety of outputs, or even the same output, regardless of the input  $z$ .

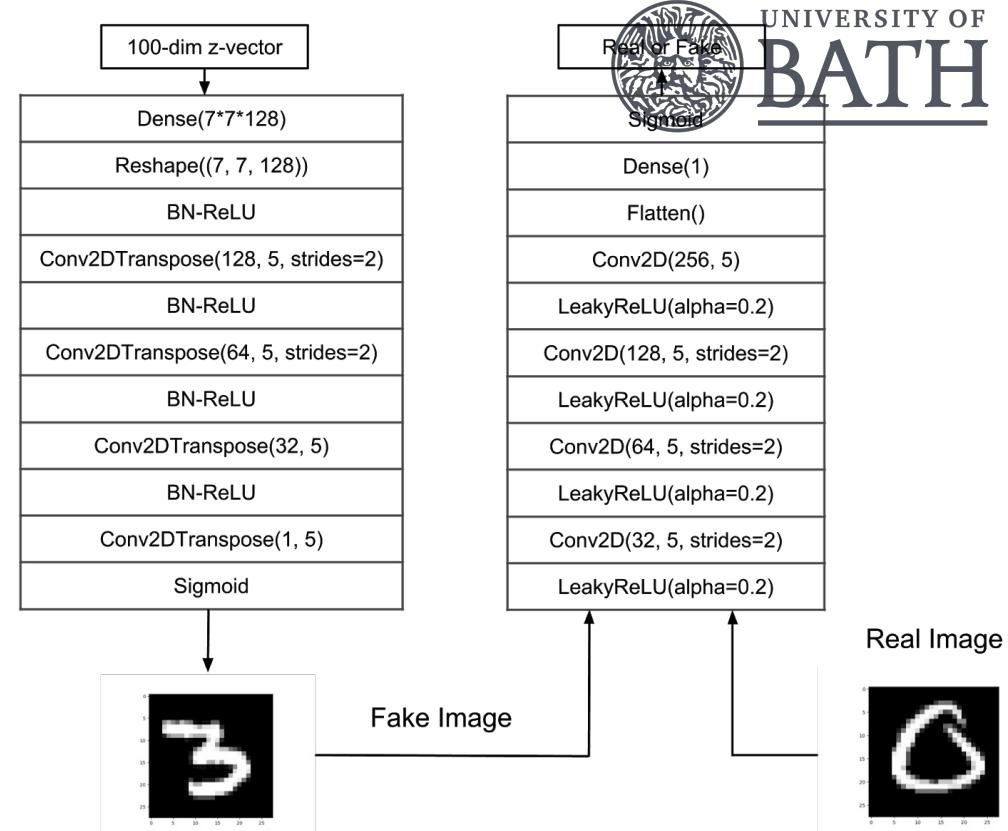


Arjovsky et al., 2017

- **Non-convergence:** the generator and discriminator continue to adapt in response to each other without reaching a stable equilibrium

# DCGAN

- Deep Convolutional GAN [Radford-2016] is a GAN implemented using Deep CNN.



## Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

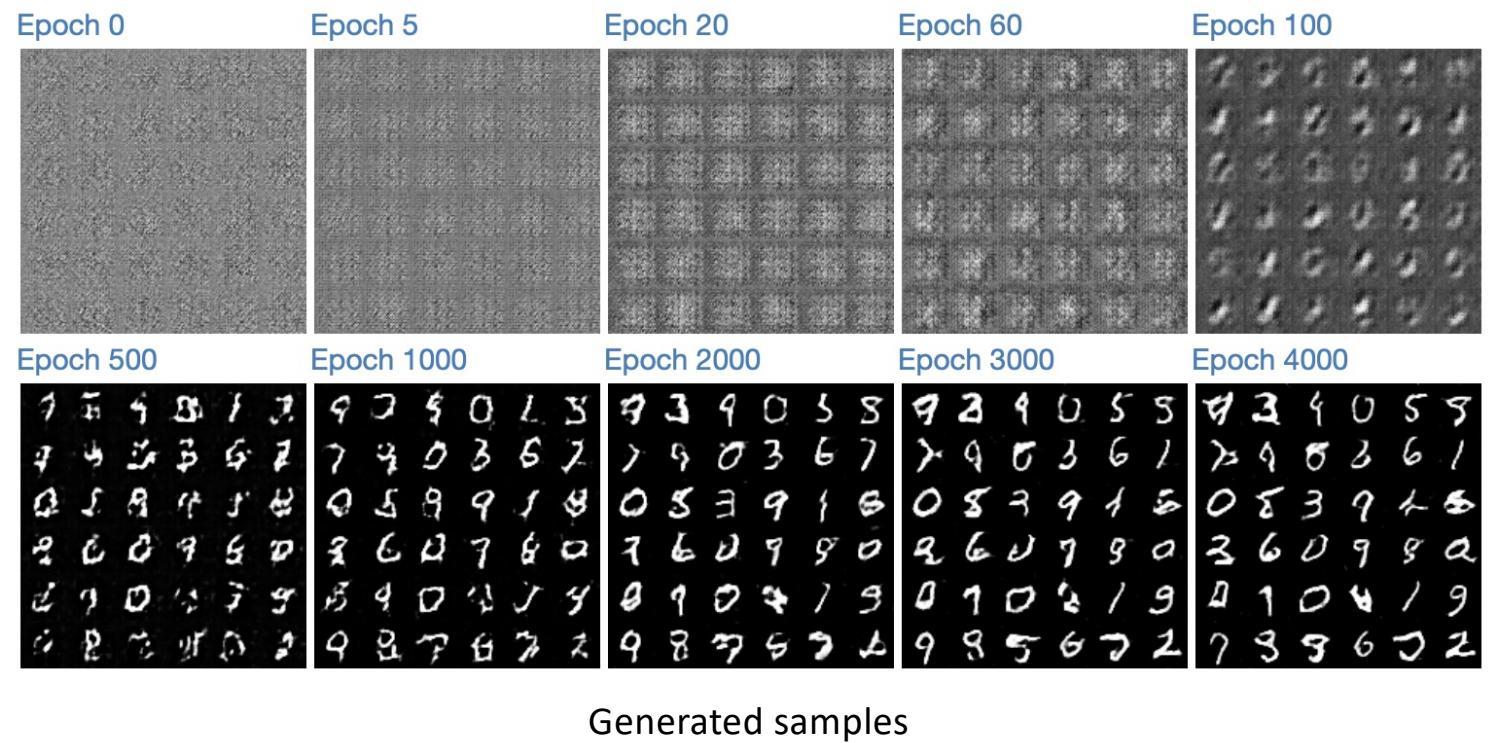
Img from: [https://docs.google.com/presentation/d/13fiFibqjI9ps\\_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.p](https://docs.google.com/presentation/d/13fiFibqjI9ps_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.p)

[Radford-2016] Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

# DCGAN



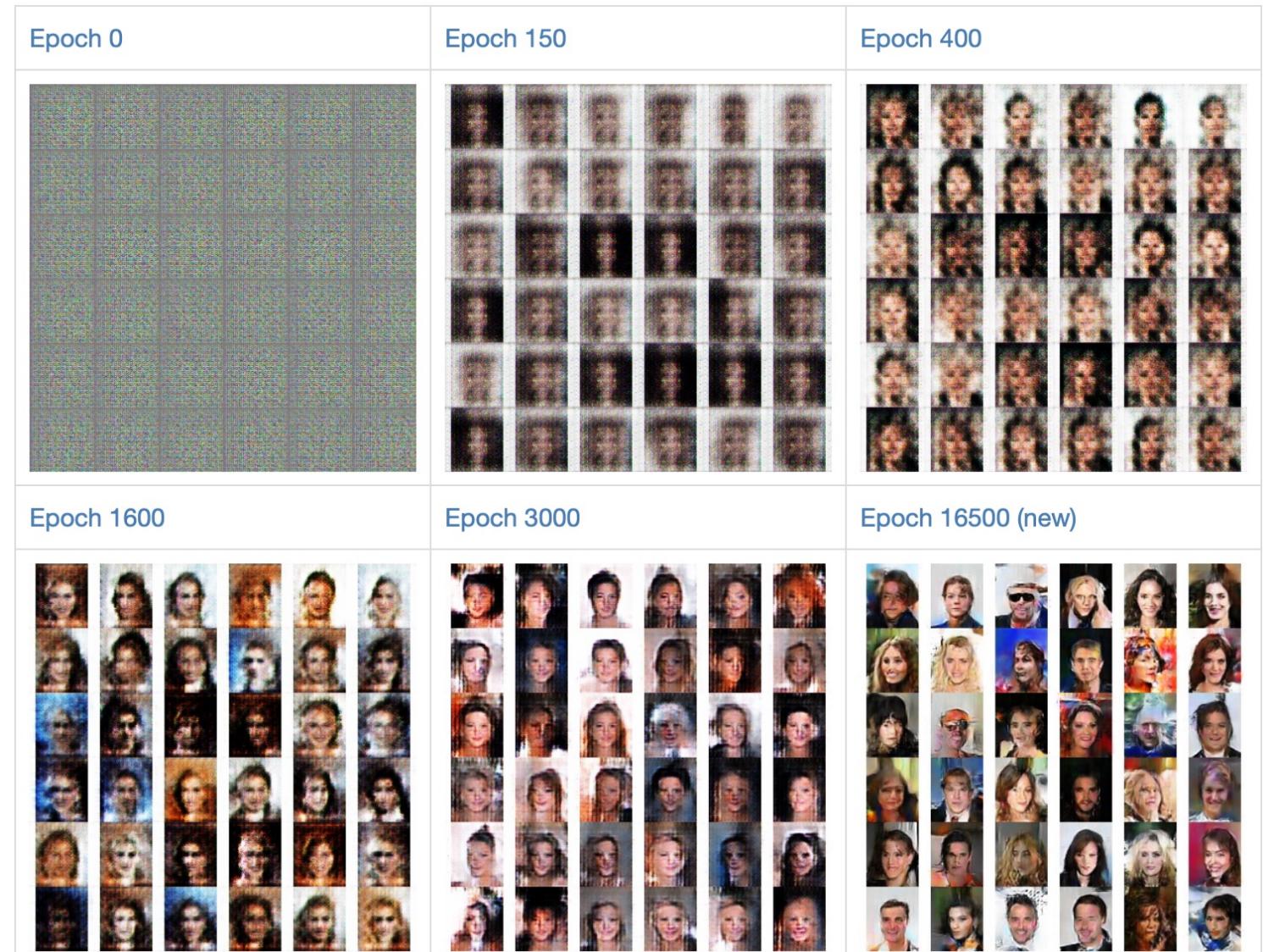
MNIST dataset



# DCGAN



CelebA dataset

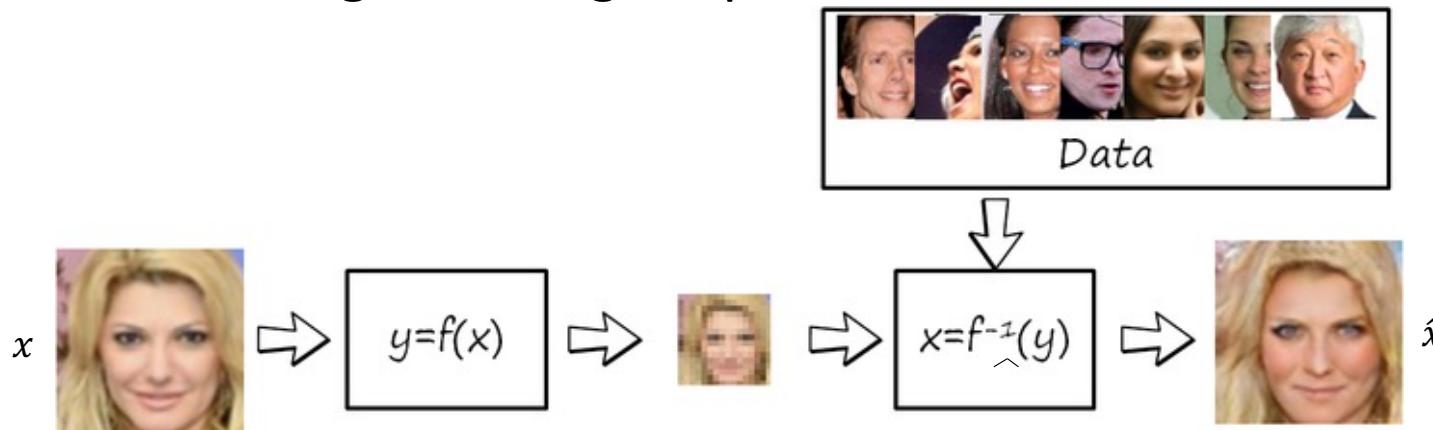


Generated samples

Image from: [http://www.timzhangyuxuan.com/project\\_dcgan/](http://www.timzhangyuxuan.com/project_dcgan/)

# Super Resolution

- Data-driven image-to-image super resolution



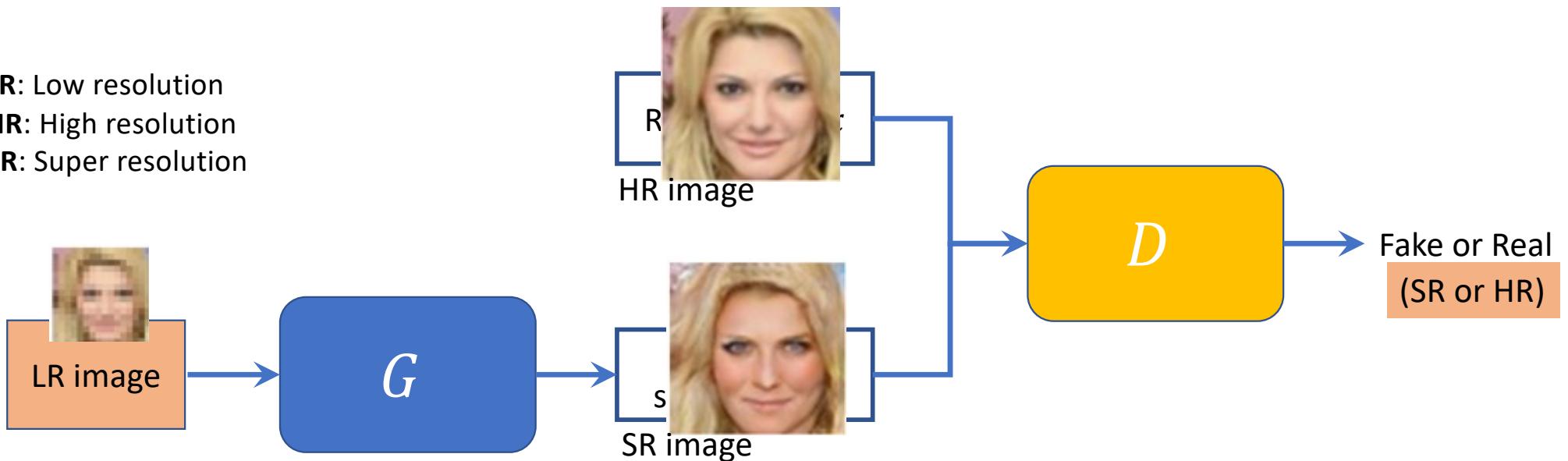
Standard Objective: Minimize the MSE between  $x$  and  $\hat{x}$  (**pixel-level**)

- + Benefit: Maximizing Peak Signal-to-Noise Ratio (PSNR)
- Drawback: Limit for capturing perceptually relevant features, e.g., losing high textural details

Minimize the MSE between  $\phi_{ij}(x)$  and  $\phi_{ij}(\hat{x})$  (**feature-level**)

# SRGAN [Ledig-2017]

LR: Low resolution  
 HR: High resolution  
 SR: Super resolution



**Objective:**

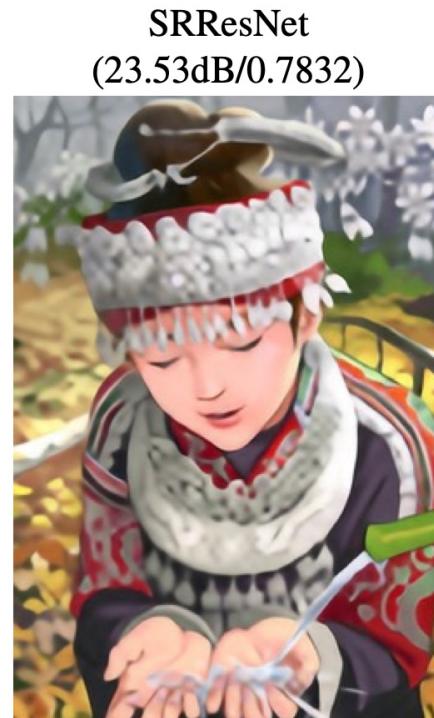
$$l^{SR} = \underbrace{l_X^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}}$$

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

$$l_{Gen}^{SR} = \sum_{n=1}^N -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

Ledig et al. "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", CVPR 2017.

# SRGAN [Ledig-2017]



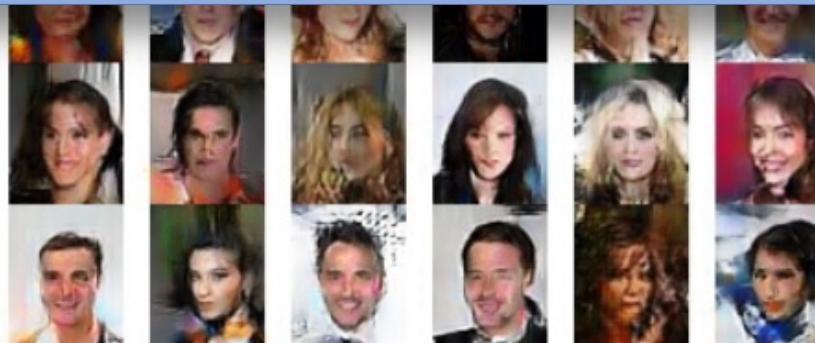
# Reference

- Lecture on Generative Adversarial Networks:  
<https://developers.google.com/machine-learning/gan>
- Video lecture by Ava Soleimany: MIT course on Deep Generative Models, YouTube.
- Lecture by Rowel Atienza:  
[https://docs.google.com/presentation/d/13fiFibqjl9ps\\_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.g389666c0c0\\_0\\_5](https://docs.google.com/presentation/d/13fiFibqjl9ps_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.g389666c0c0_0_5)

# Topic 3: GANs Variations (Conditional GAN)



**Drawback 1 of standard GAN:** lack of control of the types of images generated.

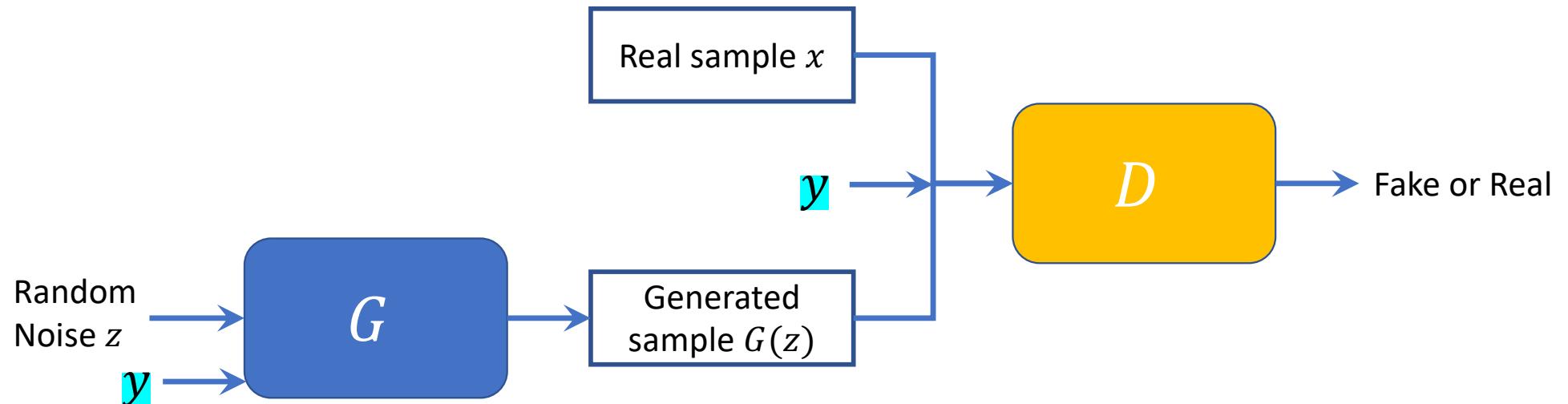


Generated samples of GAN

Image from: [http://www.timzhangyuxuan.com/project\\_dcgan/](http://www.timzhangyuxuan.com/project_dcgan/)

# Conditional GAN (CGAN)

- CGAN [Mirza-2014] imposes a constraint (e.g., a label/category) to control the attribute of the generator output.
- CGANs model the conditional probability  $P(X | Y)$ .

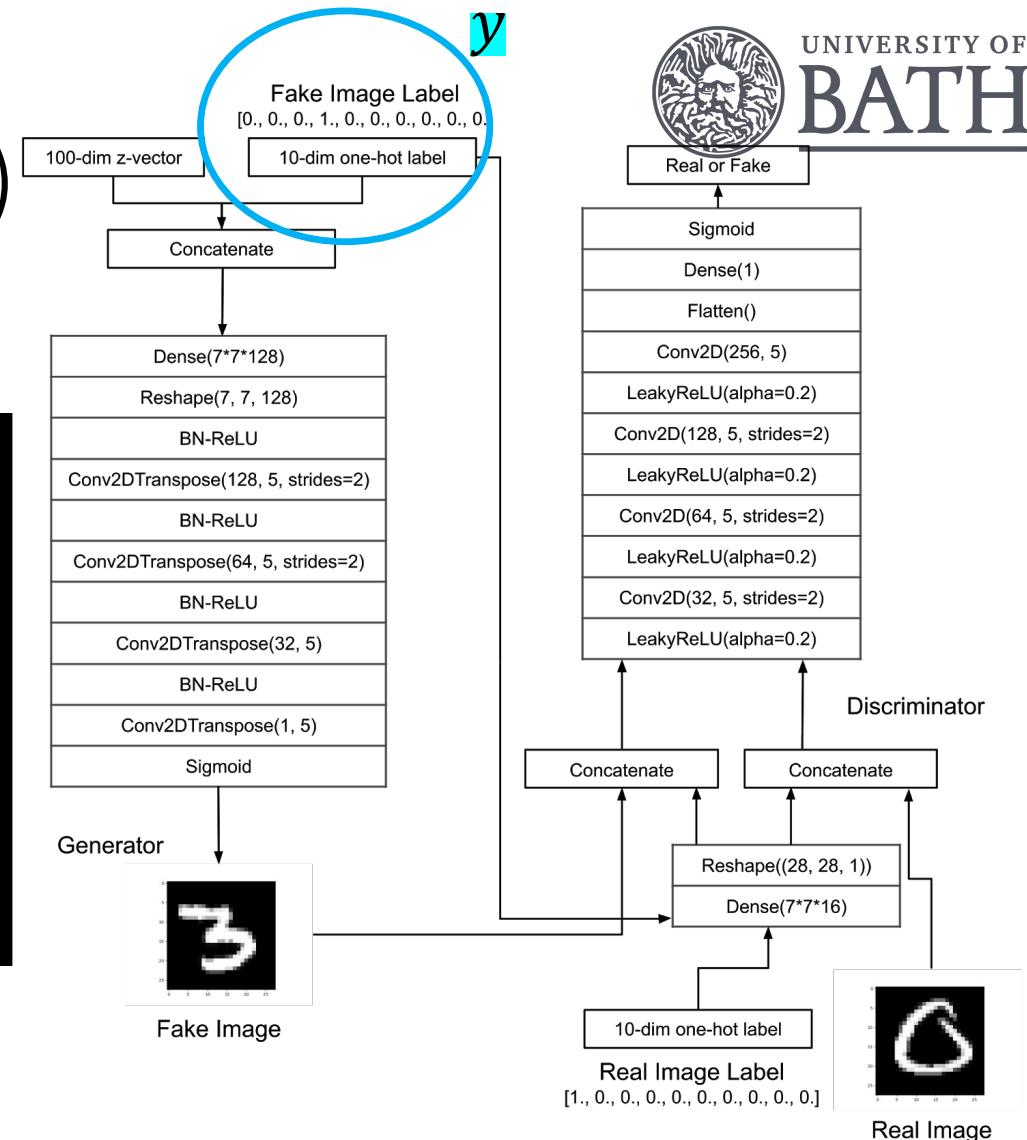


**Objective:**  $\min_{\theta_g} \max_{\theta_d} E_x[\log D_{\theta_d}(x|y)] + E_z[\log(1 - D_{\theta_d}(G_{\theta_g}(z|y)))]$

# Conditional GAN (CGAN)



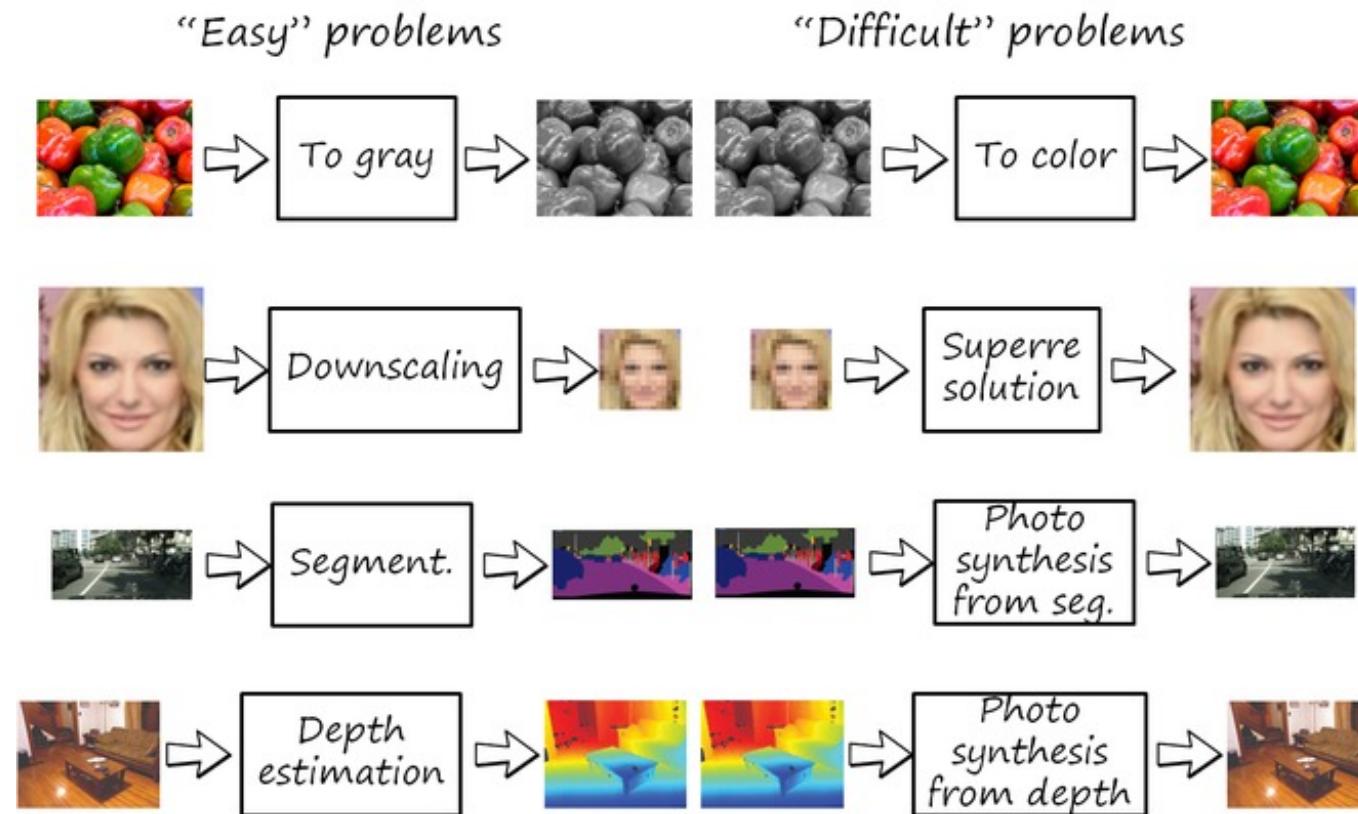
Generated MNIST digits, each row conditioned  
on one label [Mirza-2014]



# Conditional GAN (CGAN)

- Category is only one type of conditions. Actually, the condition could be:
  - Image embeddings → image-to-image translation
  - Text embeddings → text-to-image synthesis
    - E.g., StackGAN [Zhang-2017]
  - Attribute embeddings → attribute-to-image synthesis
    - E.g., InfoGAN [Chen-2016]
  - ...

# Image-to-Image Translation



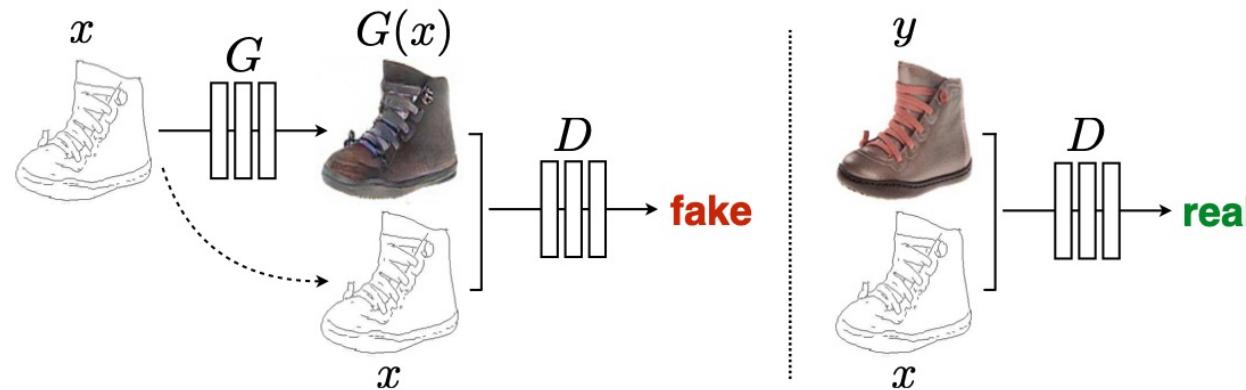
# Pix2Pix

Paired image-to-image translation



# Pix2Pix

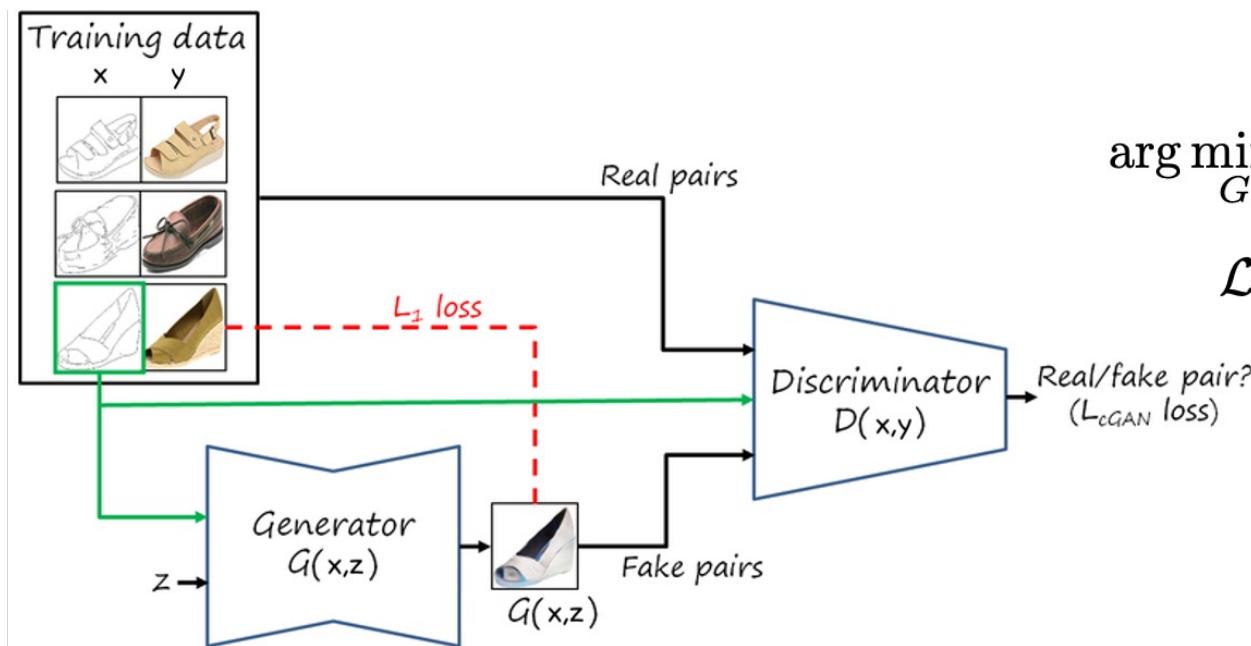
- Pix2Pix [Isola-2017] is a type of CGAN, where the condition is the paired image.



- Unlike the unconditional GAN, both the generator and discriminator observe the input edge map  $x$ .

# Pix2Pix

- In addition to the the CGAN loss, a L1 loss that forces the generated image to remain as similar as possible to the ground-truth image.

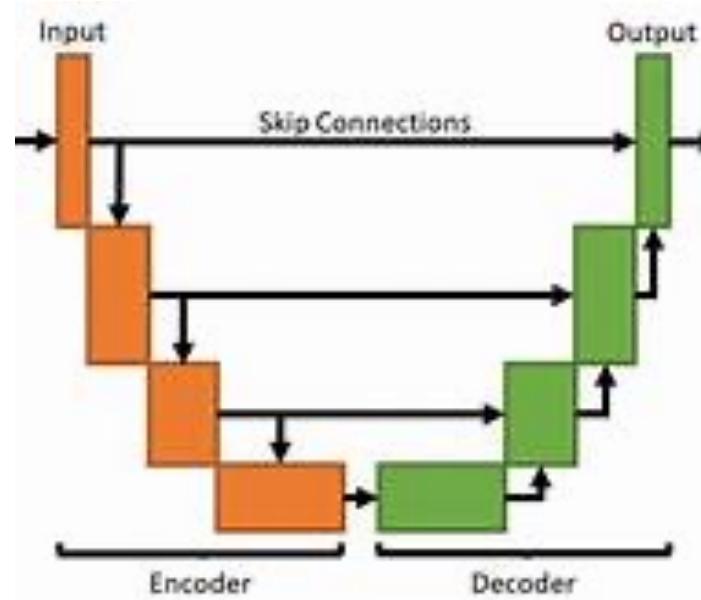
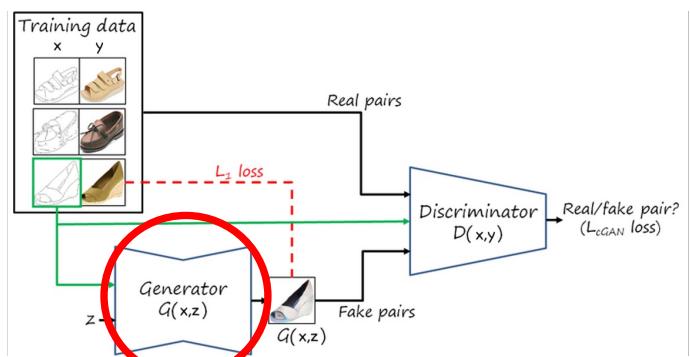


$$\arg \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1]$$

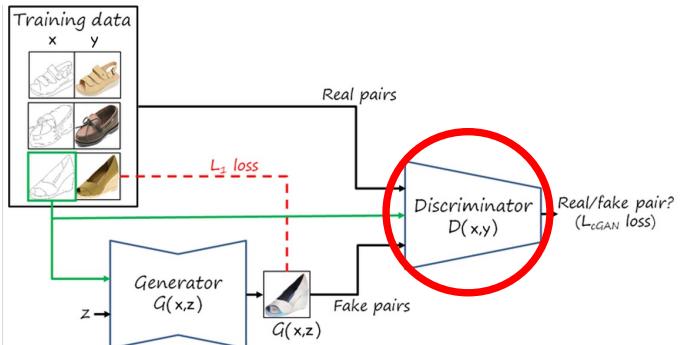
# Pix2Pix

- Generator: U-Net + skips



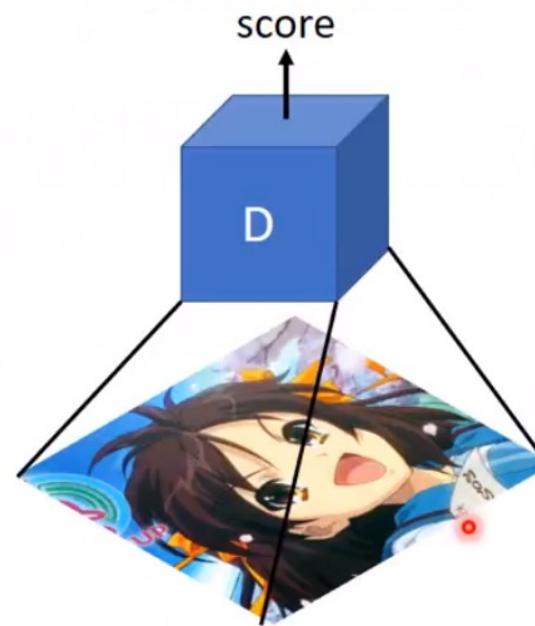
# Pix2Pix

- Discriminator: **PatchGAN**



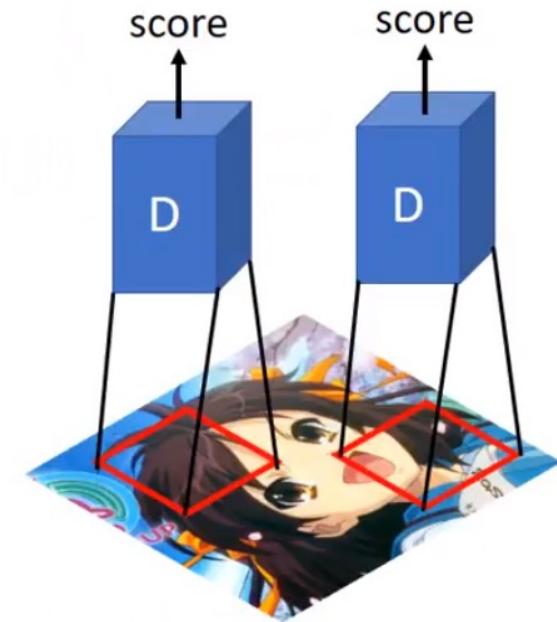
## Standard discriminator

Classify an entire image as fake or real



## PatchGAN

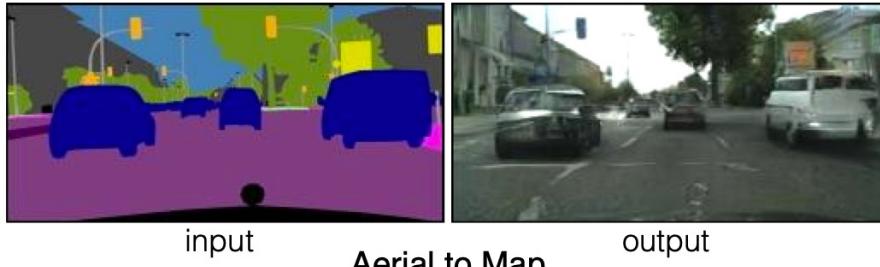
Classify if each of N patches is real or fake



# Pix2Pix

Pix2Pix is suitable for any tasks where paired image dataset can be created.

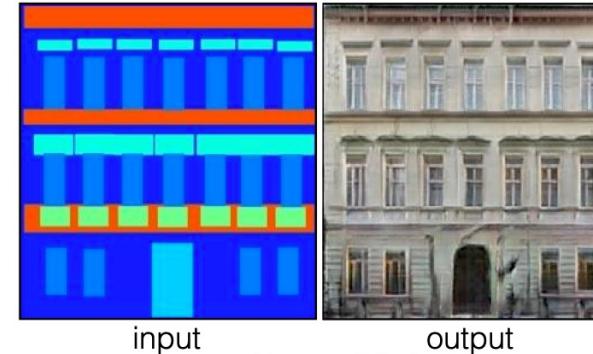
Labels to Street Scene



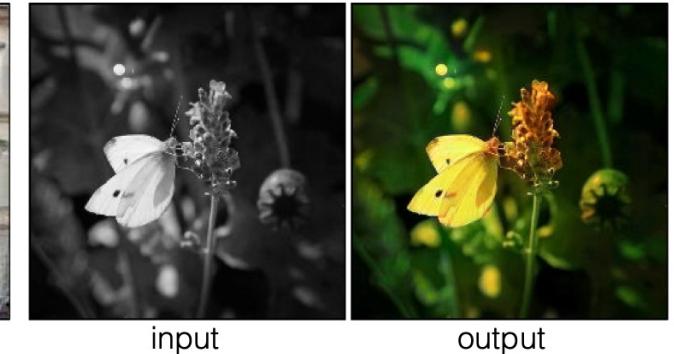
Aerial to Map



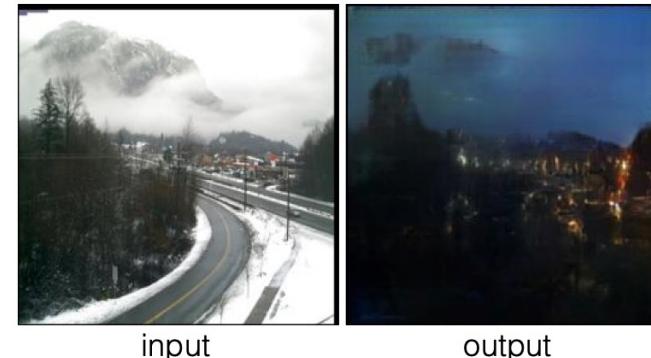
Labels to Facade



BW to Color



Day to Night



Edges to Photo



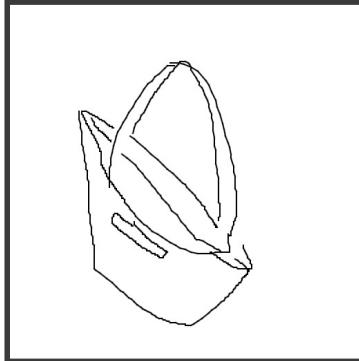
# Interactive demo of pix2pix

- Demo by Christopher Hesse:  
<https://affinelayer.com/pixsrv/>
- Tensorflow Code:  
<https://github.com/affinelayer/pix2pix-tensorflow>

edges2handbags

TOOL  
line   
eraser

INPUT



OUTPUT



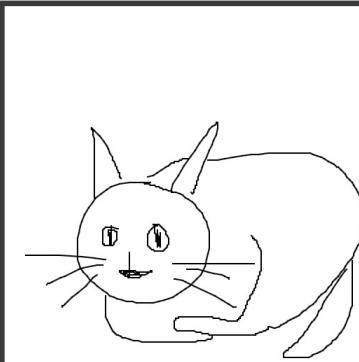
undo clear random

save

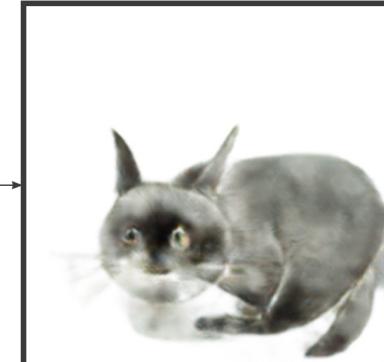
edges2cats

TOOL  
line   
eraser

INPUT



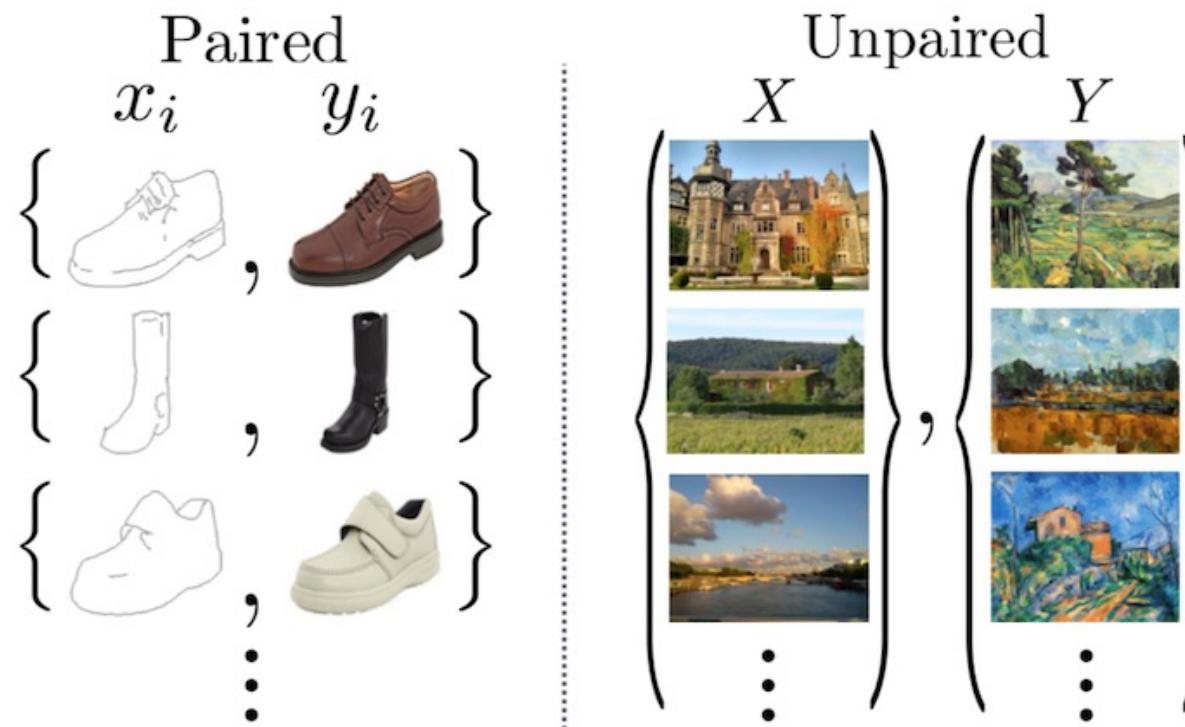
OUTPUT



undo clear random

save

But aligned pairs are not always available ...

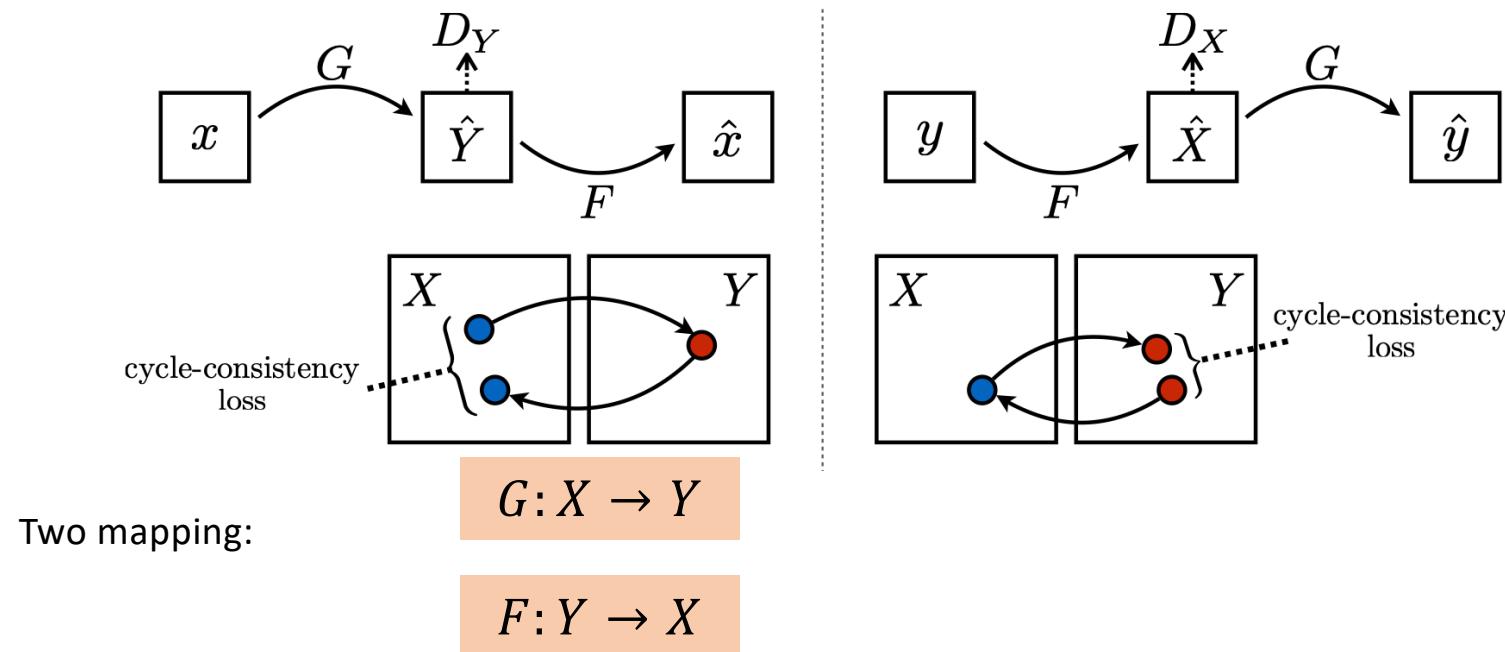


# CycleGAN

A model for domain transfer

# CycleGAN: Unpaired image-to-image translation

- CycleGAN[Zhu-ICCV2017] learns transformations across domains with unpaired data.



# Unpaired image-to-image translation: CycleGAN

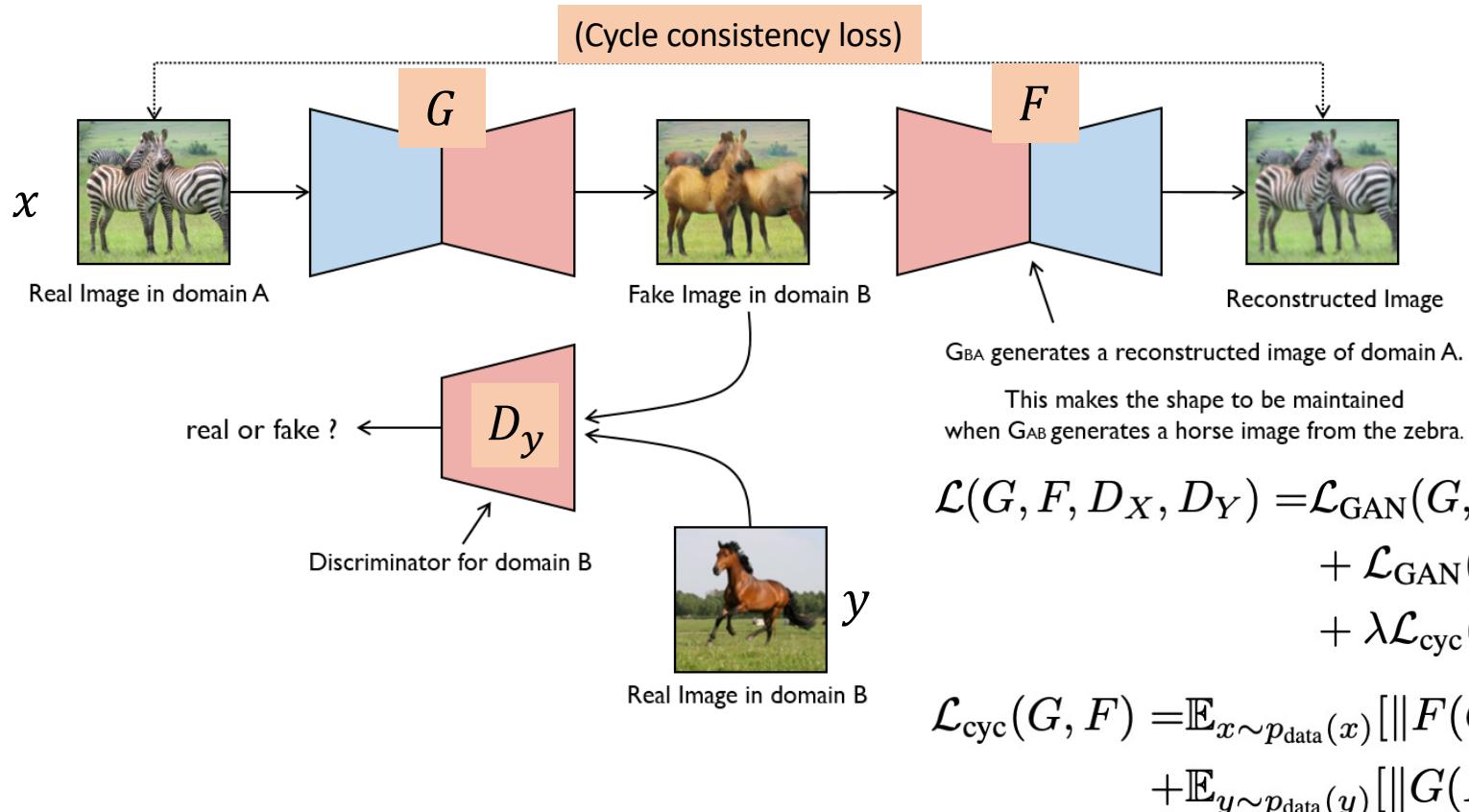


Image from <https://modelzoo.co/model/mnist-svhn-transfer>

Zhu et al. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV 2017.

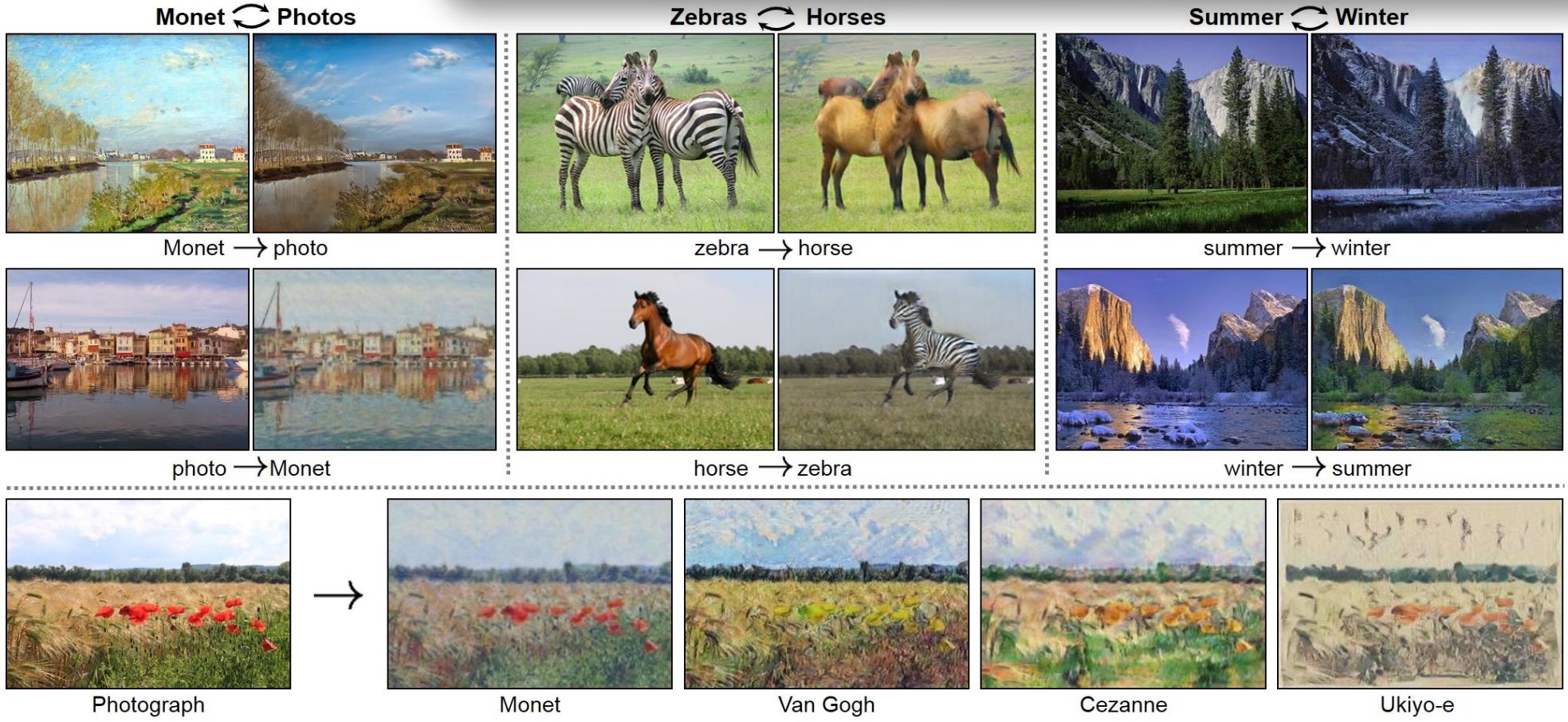
# CycleGAN



Video from: <https://youtu.be/9reHvktowLY>

# CycleGAN

CycleGAN is suitable for stylistic transformations where paired image dataset is not feasible.



More results: <https://junyanz.github.io/CycleGAN/>  
Code: <https://github.com/junyanz/CycleGAN>

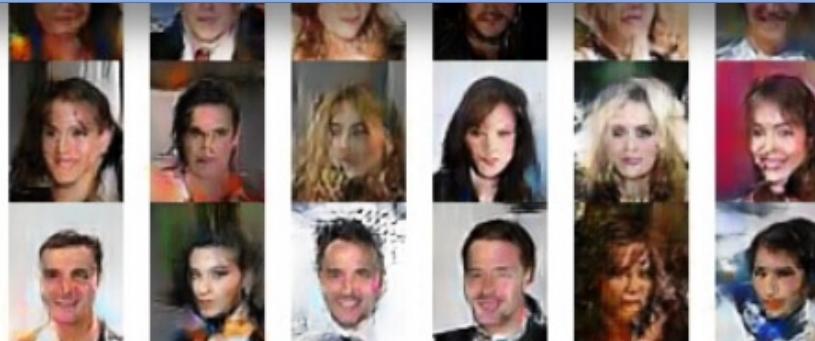
# Reference

- Lecture on Generative Adversarial Networks:  
<https://developers.google.com/machine-learning/gan/applications>
- YouTube Video by Ava Soleimany: MIT course on Deep Generative Models.
- Lecture by Rowel Atienza:  
[https://docs.google.com/presentation/d/13fiFibql9ps\\_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.g389666c0c0\\_0\\_5](https://docs.google.com/presentation/d/13fiFibql9ps_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.g389666c0c0_0_5)
- <https://junyanz.github.io/CycleGAN/>
- Blog by Luis Herranz:  
<https://www.lherranz.org/2018/08/07/imagetranslation/>

# Topic 4: GANs Variations (Progressive GAN and StyleGAN)



**Drawback 2 of standard GAN:** low-quality outputs and instability during training



Generated samples of GAN

Image from: [http://www.timzhangyuxuan.com/project\\_dcgan/](http://www.timzhangyuxuan.com/project_dcgan/)

# Progression of GAN's capability



2014



2015



2016



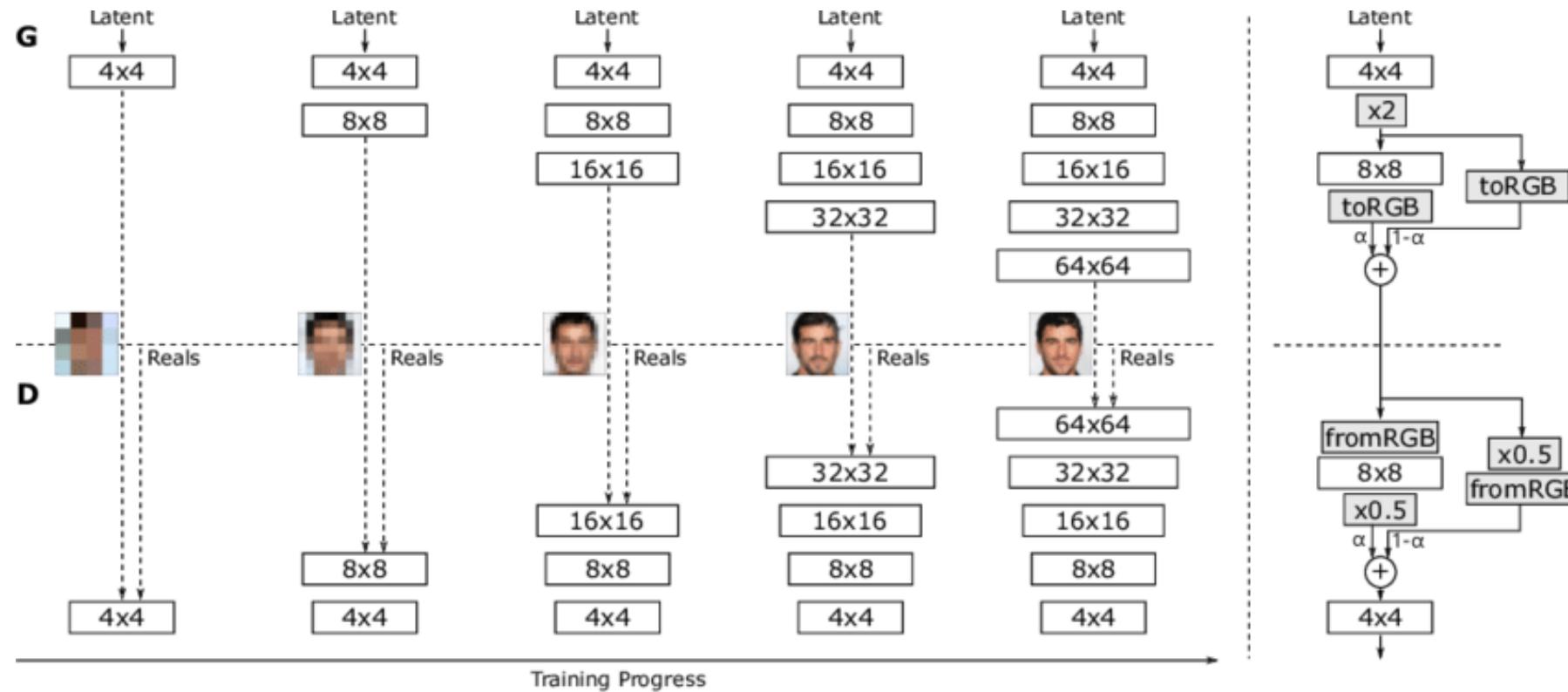
2018

Generating high-resolution images has been considered difficult by GAN due to the unstable training.  
But things changed since 2018 ...

# Progressive GAN

First discover large-scale structure of the image distribution,  
then shift attention to increasingly finer scale details

# Progressive GAN [Karras-2018]



It starts with low-resolution images and then progressively increase the resolution by adding layers to the networks.

# Progressive GAN [Karras-2018]



- Produce high resolution images
- Stabilize the training
- Speed up training

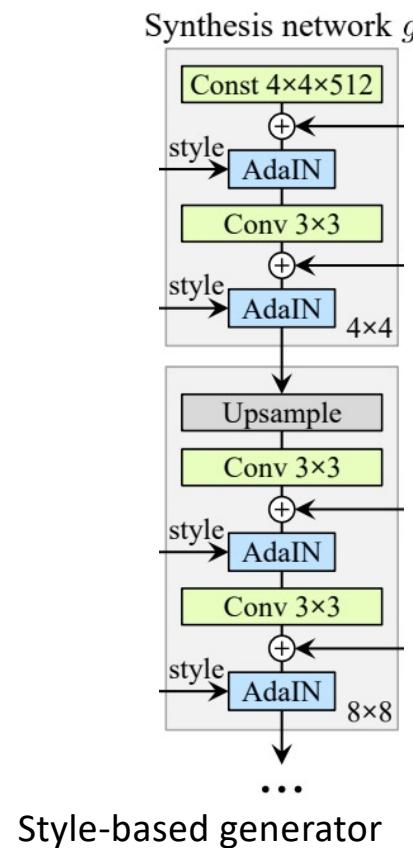
Karras et al. "Progressive growing of gans for improved quality, stability, and variation", ICLR 2018.

# StyleGAN

Extension of progressive GAN, adding control of the generated image style.

# StyleGAN [Karras-2019]

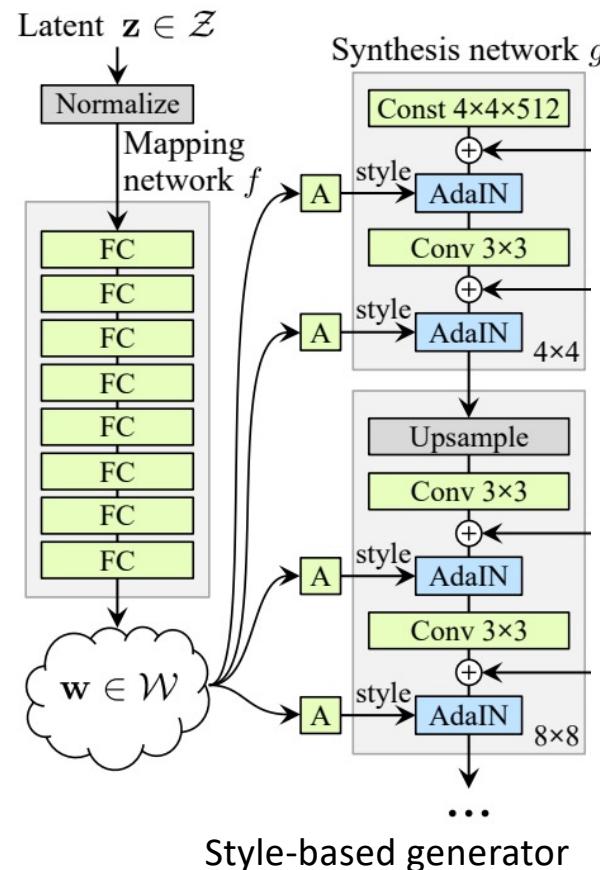
Baseline Progressive GAN



Karras et al. "A style-based generator architecture for generative adversarial networks", 2019.

# StyleGAN [Karras-2019]

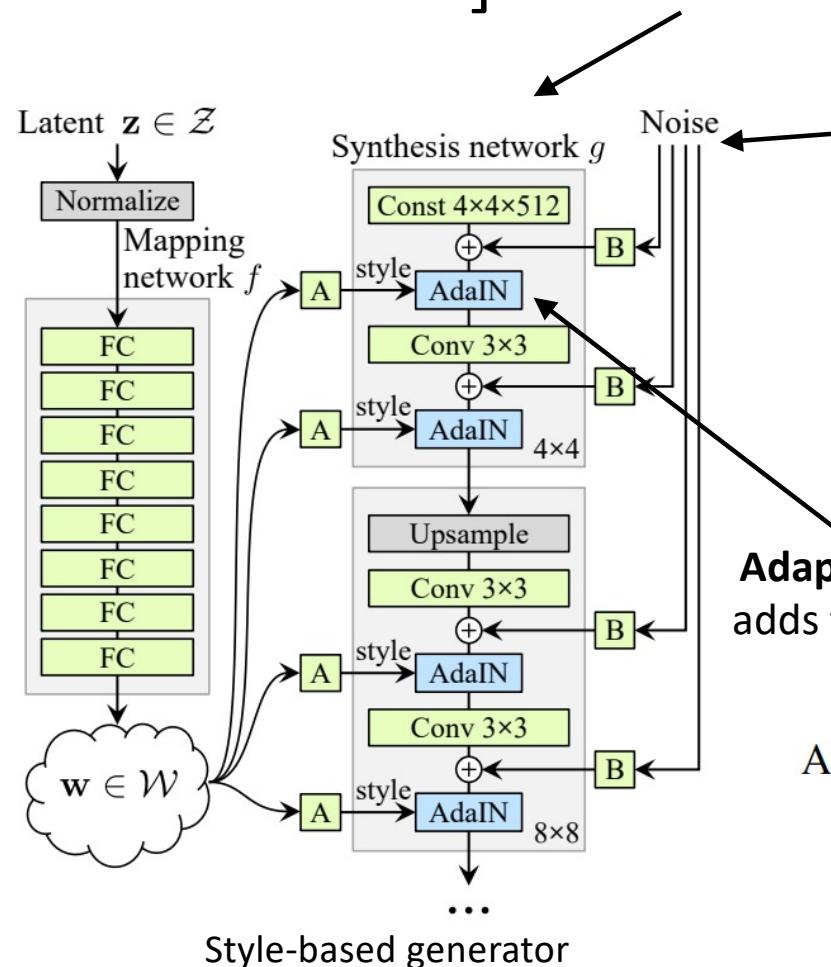
**Mapping network**  
outputs style vectors



Baseline Progressive GAN

# StyleGAN [Karras-2019]

**Mapping network**  
outputs style vectors



Baseline Progressive GAN

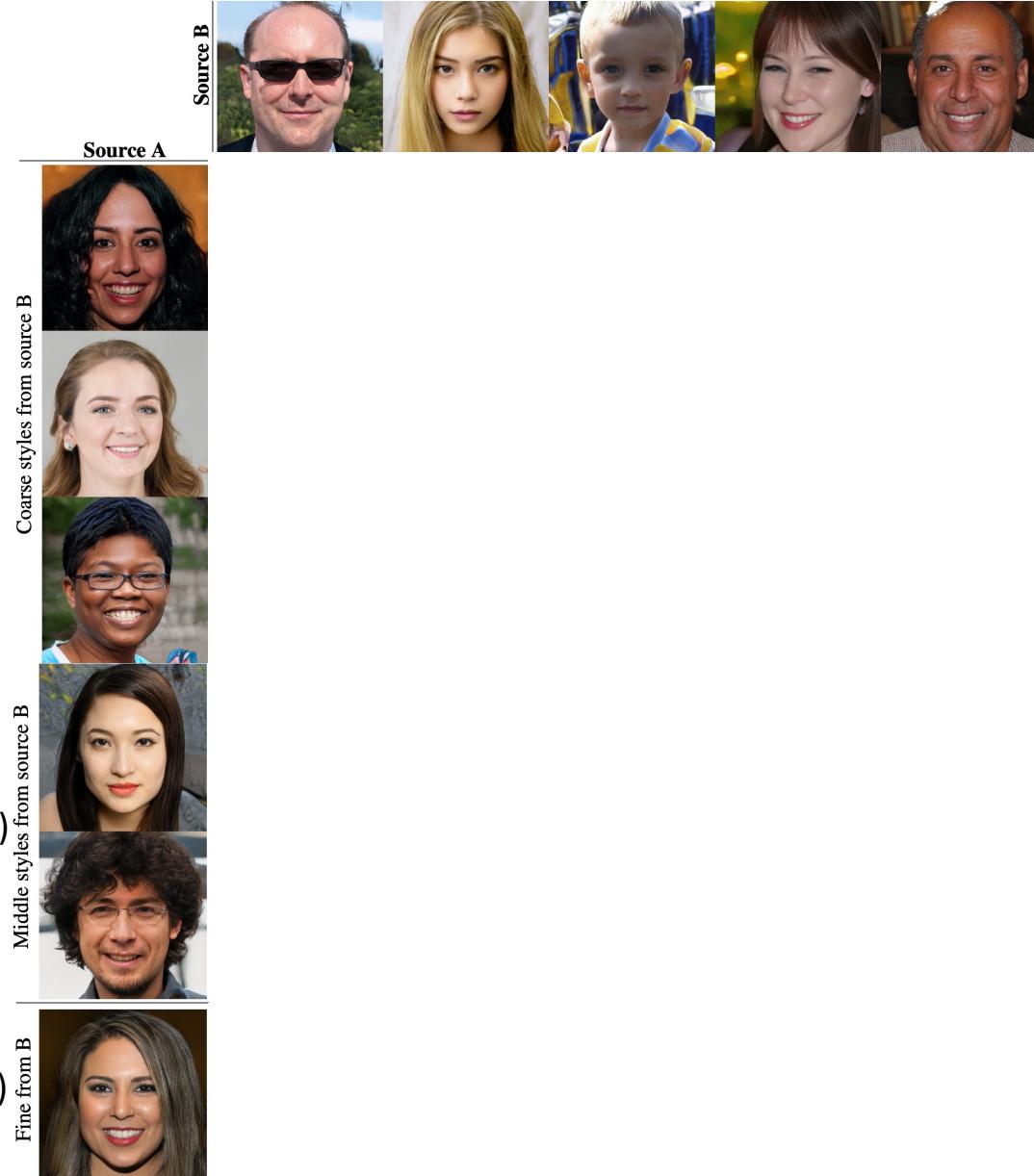
**Adding noise:** generate more stochastic details in the images.

**Adaptive Instance Normalization (AdaIN)** adds the style vector ( $y$ ) to the normalized each feature map  $X_i$ .

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$

# StyleGAN

Coarse resolutions ( $4^2 - 8^2$ )

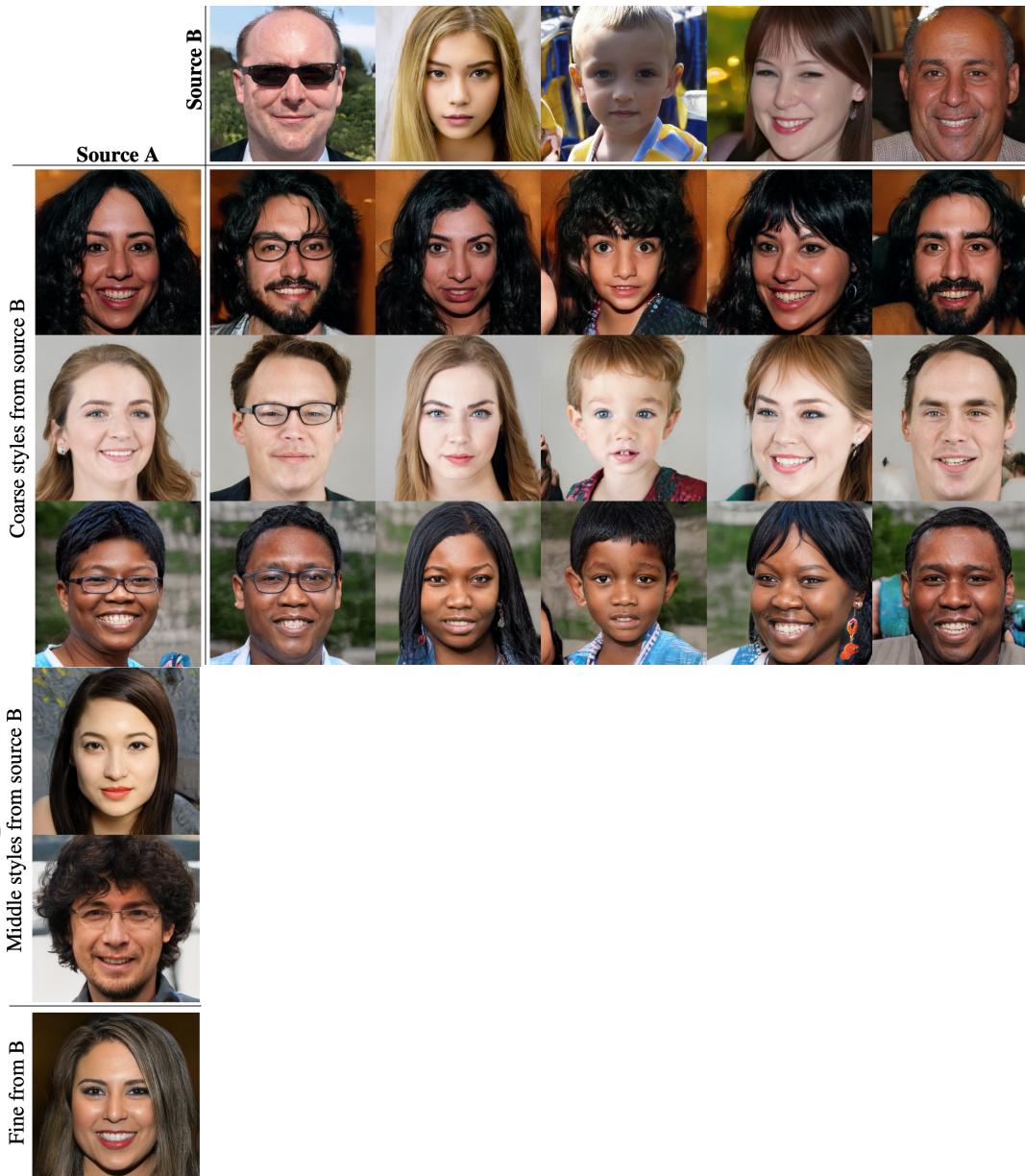


Middle resolutions ( $16^2 - 32^2$ )

Fine resolutions ( $64^2 - 1024^2$ )

# StyleGAN

Coarse resolutions ( $4^2 - 8^2$ )



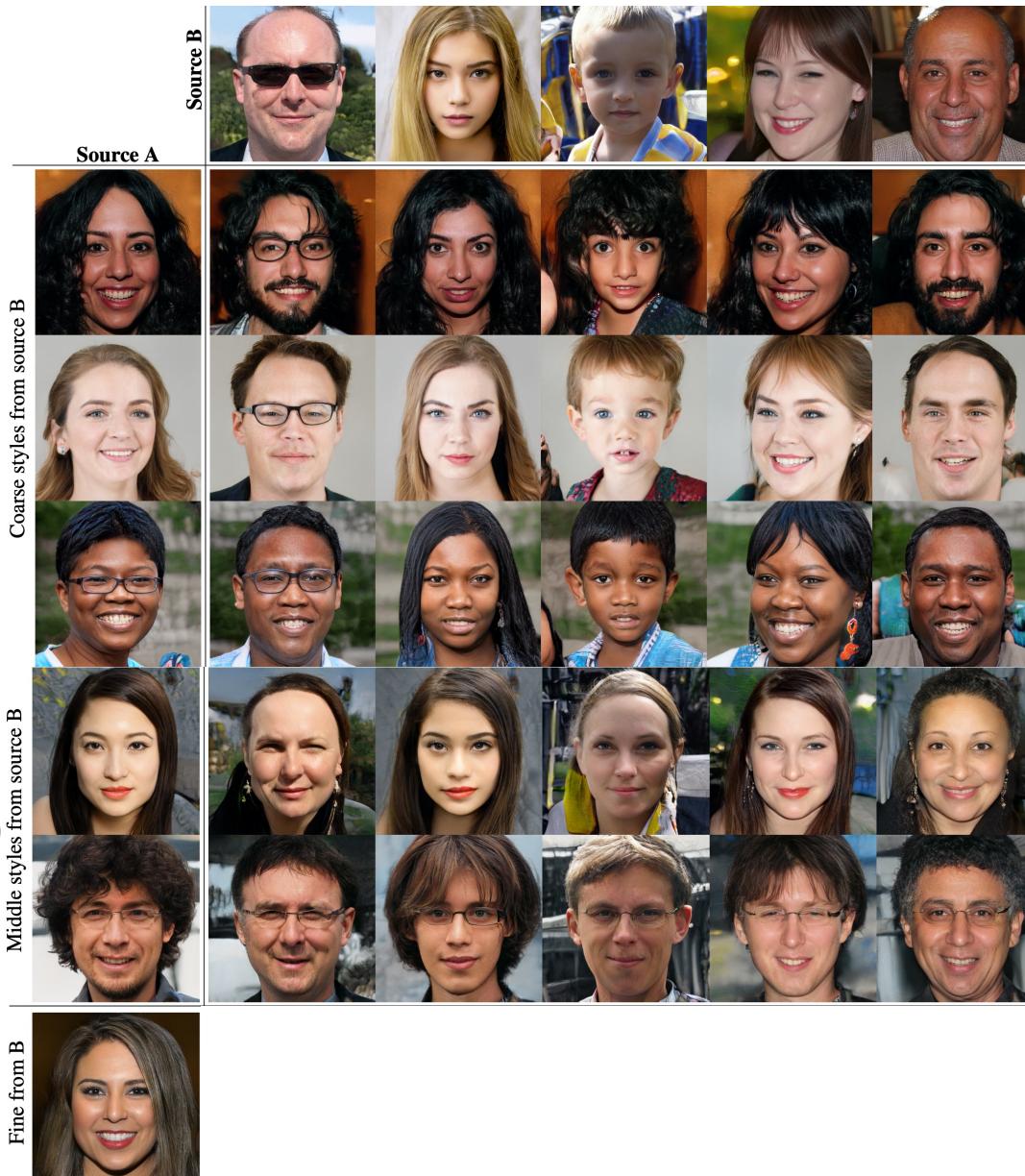
Middle resolutions ( $16^2 - 32^2$ )

Fine resolutions ( $64^2 - 1024^2$ )

Pose, hair style,  
face shape,  
gender, age,  
glasses

# StyleGAN

Coarse resolutions ( $4^2 - 8^2$ )



Pose, hair style,  
face shape,  
gender, age,  
glasses

Facial features,  
eyes

Middle resolutions ( $16^2 - 32^2$ )

Fine resolutions ( $64^2 - 1024^2$ )

# StyleGAN

Coarse resolutions ( $4^2 - 8^2$ )



Pose, hair style,  
face shape,  
gender, age,  
glasses

In StyleGAN, the generator thinks of an image as a collection of “style”,  
where each style controls the effects at a particular scale.

Middle resolutions ( $16^2 - 32^2$ )



Facial features,  
eyes

Fine resolutions ( $64^2 - 1024^2$ )



Color scheme

# Reference

- Lecture on Generative Adversarial Networks:  
<https://developers.google.com/machine-learning/gan/applications>
- YouTube Video by Ava Soleimany: MIT course on Deep Generative Models.
- Lecture by Rowel Atienza:  
[https://docs.google.com/presentation/d/13fiFibql9ps\\_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.g389666c0c0\\_0\\_5](https://docs.google.com/presentation/d/13fiFibql9ps_CktJzMNAvoZXOlzHQDu8eRSb3a227g/edit#slide=id.g389666c0c0_0_5)
- Blog by Joson Brownlee:  
<https://machinelearningmastery.com/introduction-to-style-generative-adversarial-network-stylegan/>

# Topic 5: More GANs Applications

# Text-to-Image Synthesis

Input Text: A stained glass window with an image of a blue strawberry



Stage-I images



The small bird has a red head with feathers that fade from red to gray from head to tail

Stage-II images



Stage-I images



This bird is black with green and has a very short beak

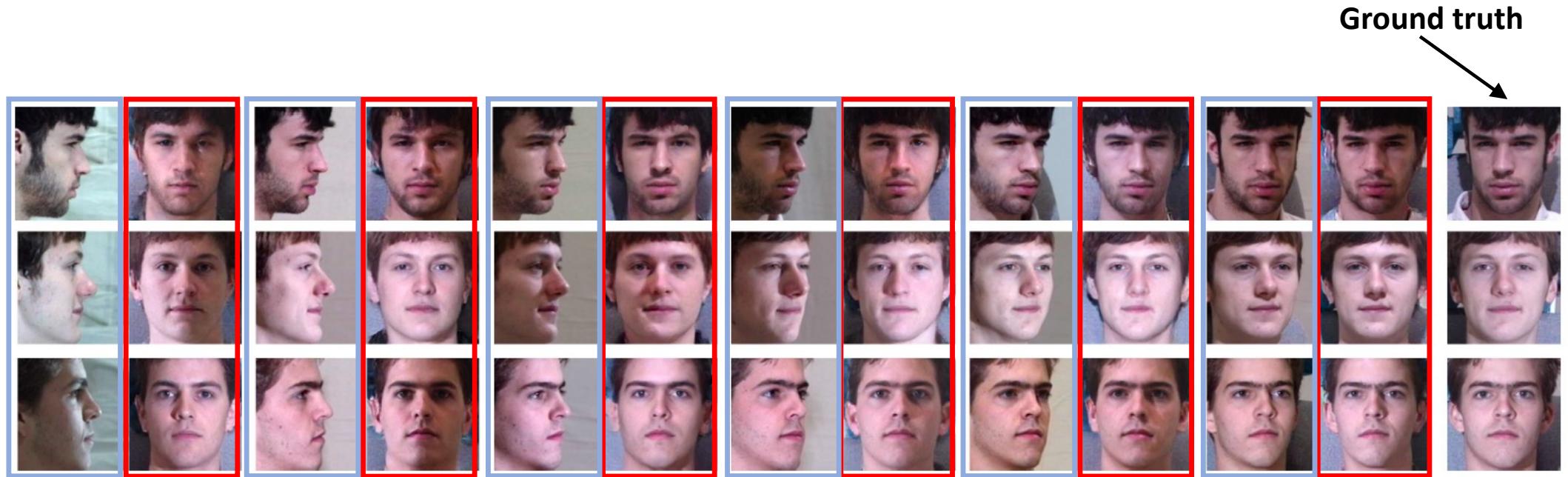
Stage-II images



Reed et al. "Generative adversarial text to image synthesis". ICML 2016

Han Zhang et al. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks , ICCV 2017.

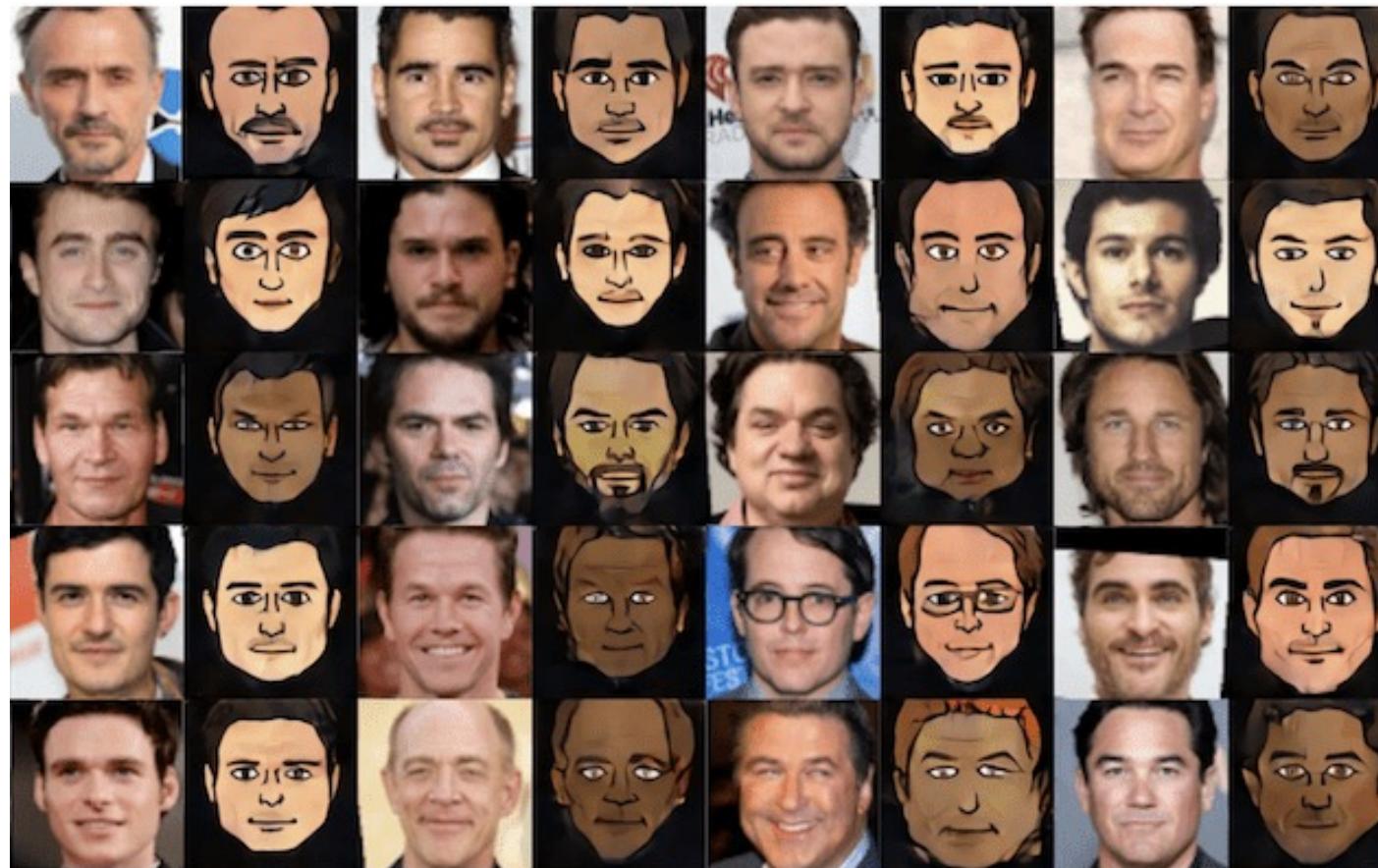
# Face Frontal View Generation



**Input face taken at an angle**

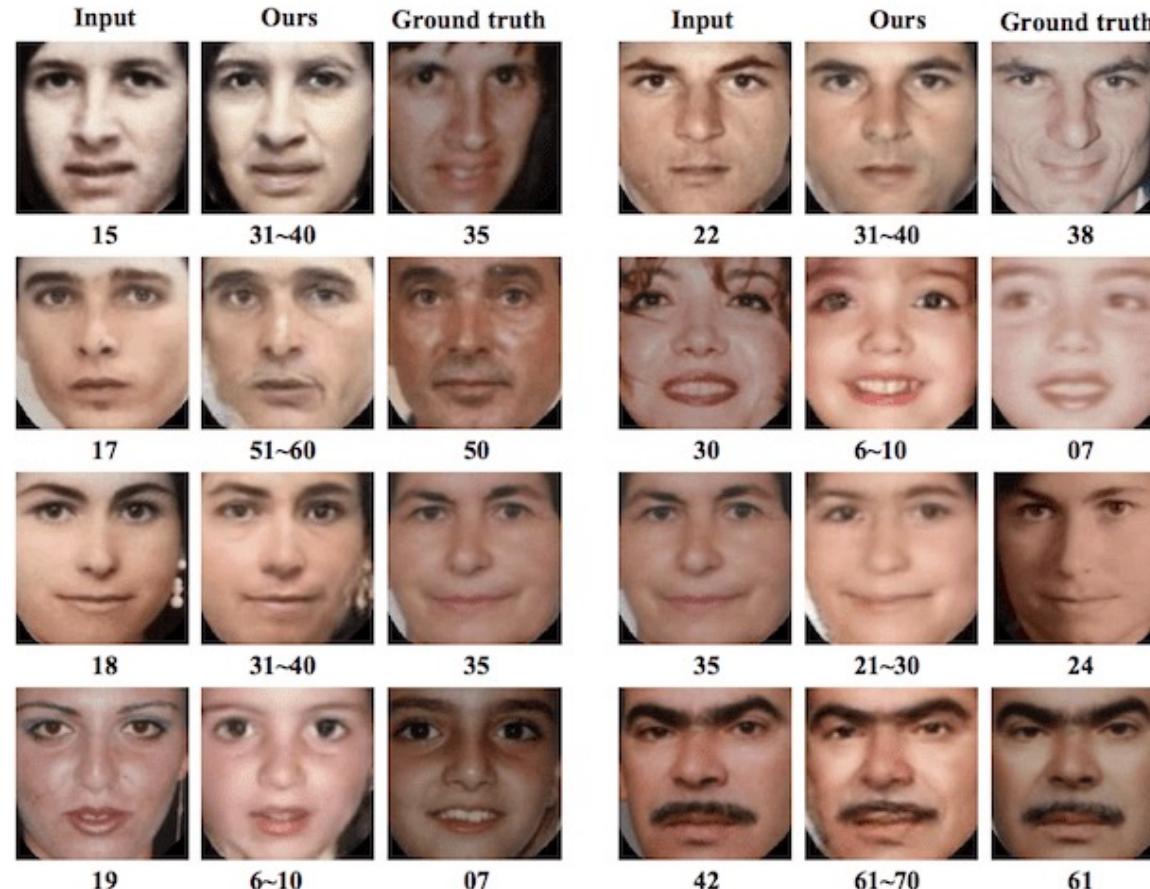
**Generated frontal faces**

# Photos to Emojis



Taigman, et al. Unsupervised cross-domain image generation, 2016.

# Face Aging



ANtipov et. al. Face Aging With Conditional Generative Adversarial Networks, 2017

Zhang et. al. Age Progression/Regression by Conditional Adversarial Autoencoder, 2017.

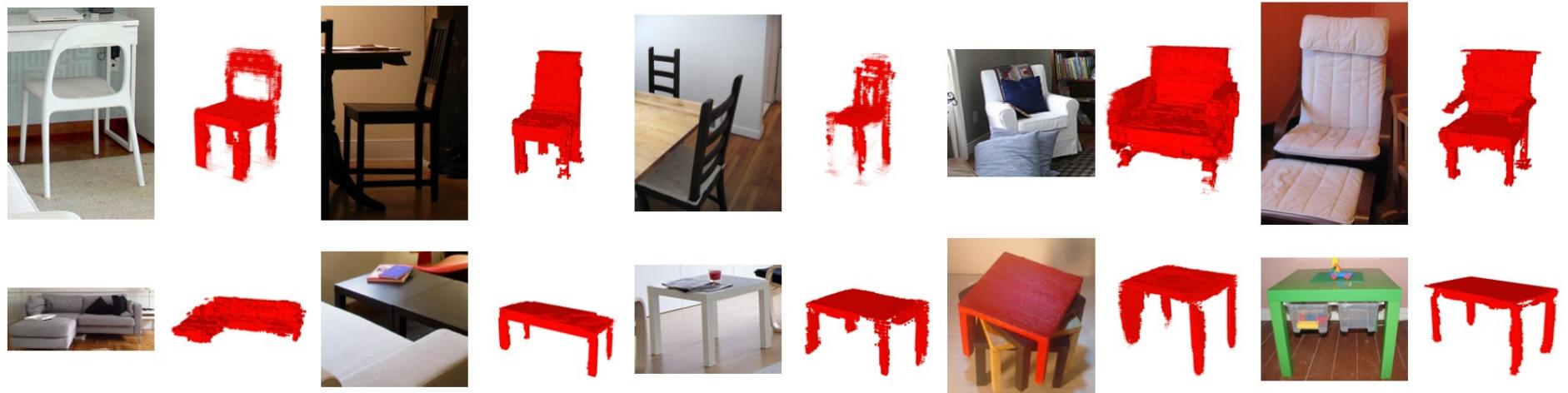
# Photo Inpainting

Real      Input      Ours      NN



Yeh et. al. "Semantic Image Inpainting with Deep Generative Models", CVPR 2017

# 3D Object Generation



# Video prediction

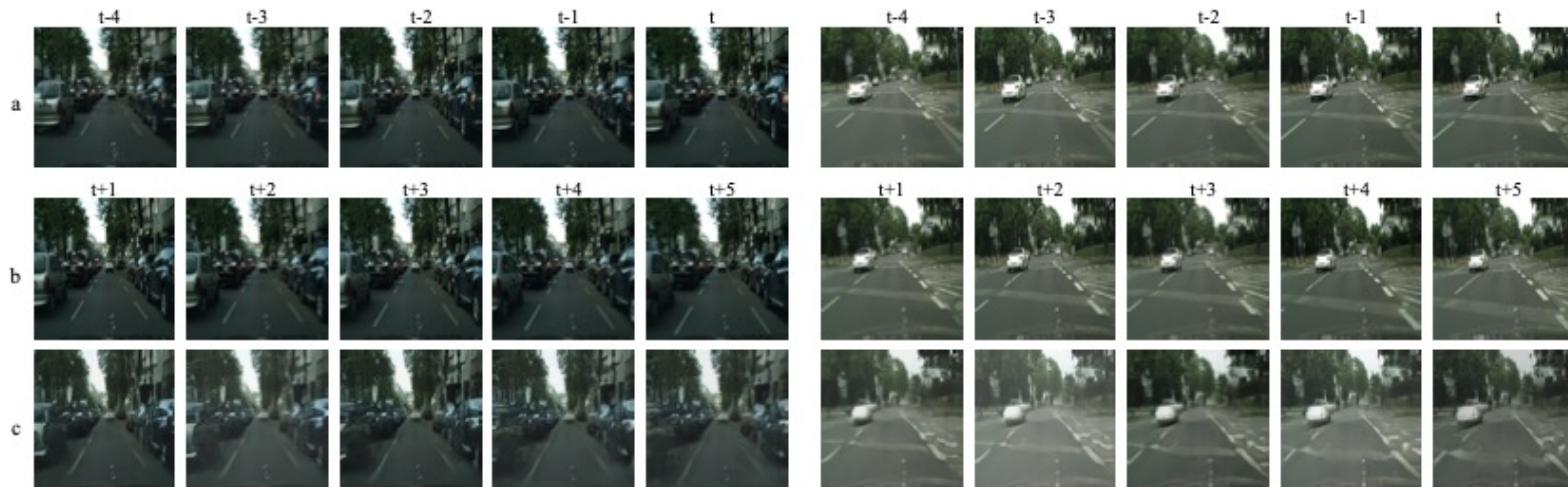


Figure 6: Prediction results for the Cityscapes test sequences. a: Input, b: Ground Truth, c: FutureGAN (ours)

GAN is a new discovered land.  
Looking for treasures has never been stopped ...

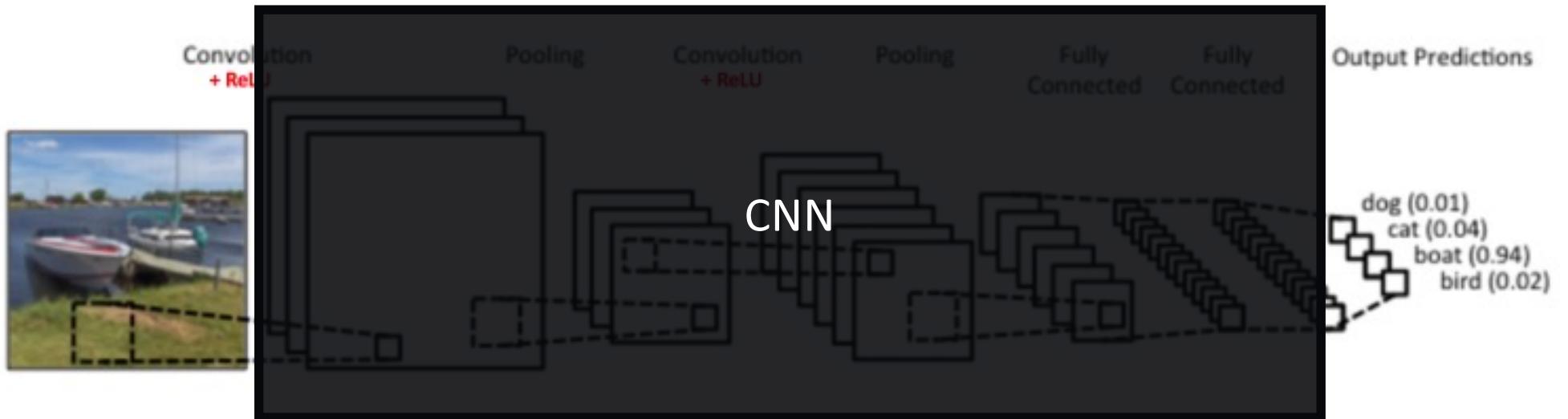
# Reference

- Blog by Jason Brownlee: 18 Impressive Applications of GANs.  
<https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/>
- Blog by Martin Isaksson: Five GANs for Better Image Processing.  
<https://towardsdatascience.com/five-gans-for-better-image-processing-fabab88b370b>
- Noshory: <https://github.com/nashory/gans-awesome-applications>

# Week 08: Visualizing CNN

Dr. Hongping Cai  
Department of Computer Science  
University of Bath

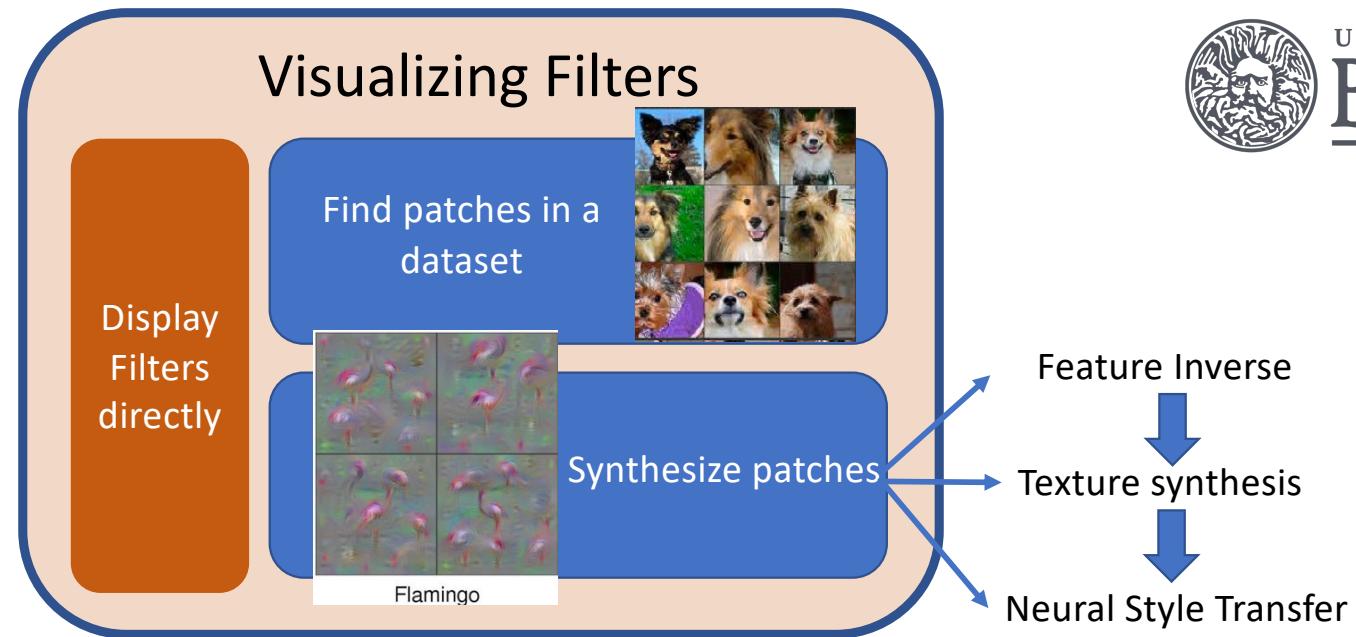
# Topic 1: Visualizing Activation Maps



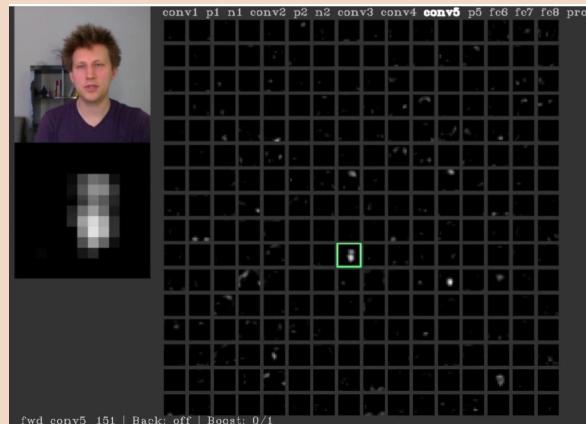
**Q:** What does CNN learn?

**A:** Visualizing CNN

# Different ways



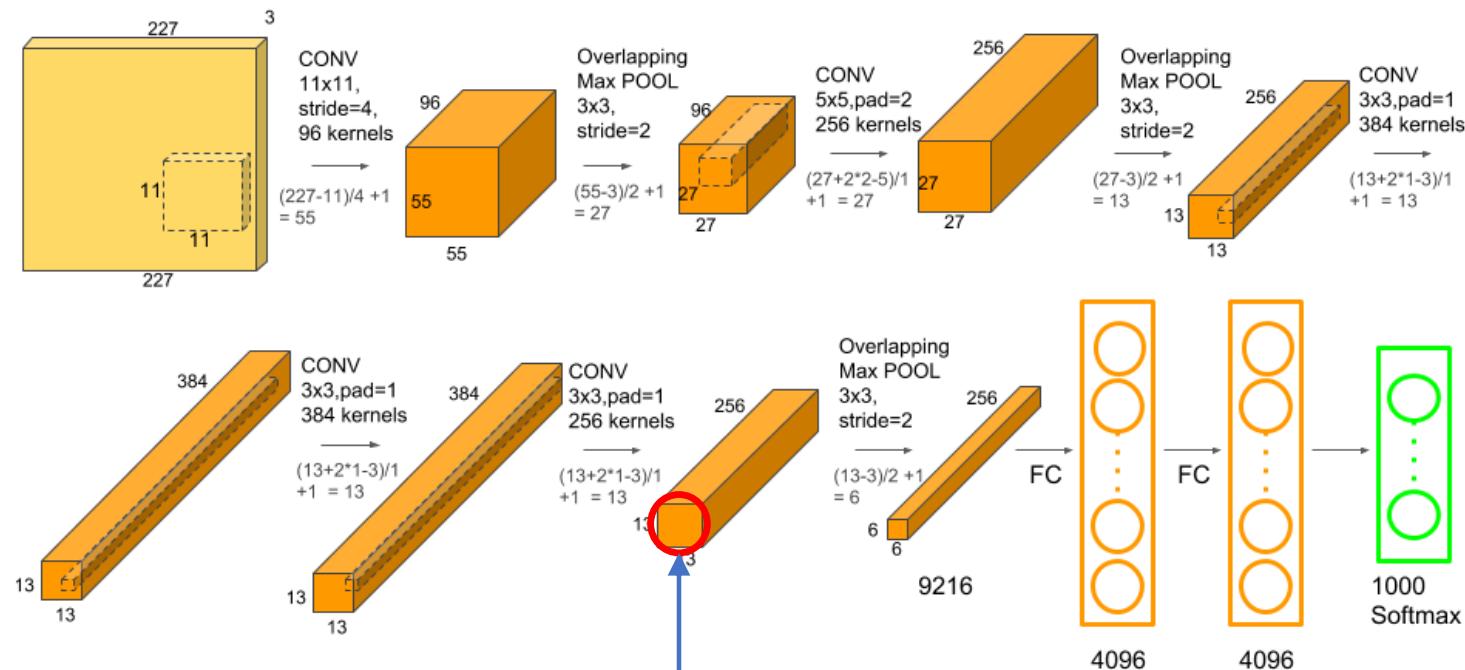
**Visualizing Activation Maps**



**Visualizing Heatmaps**

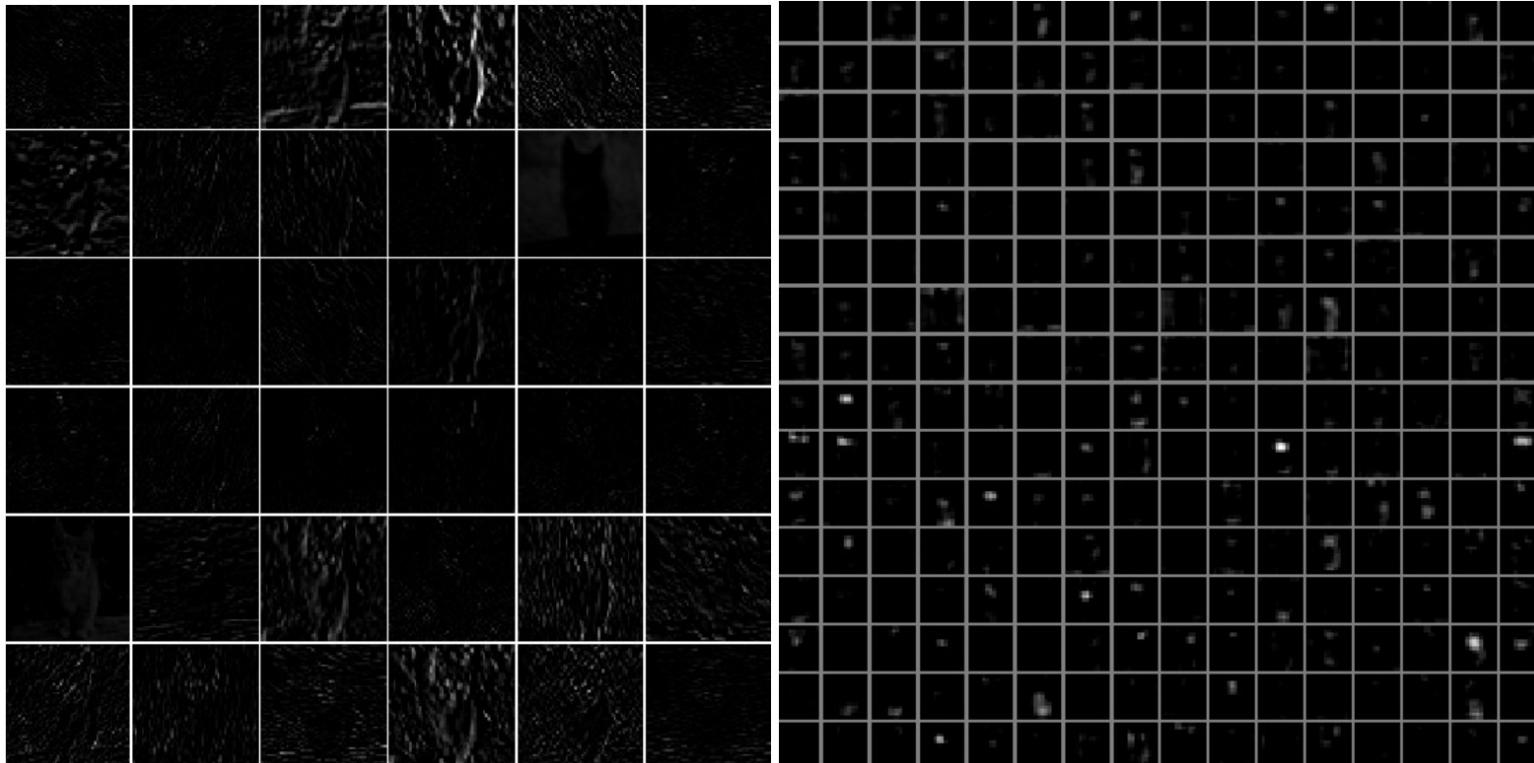


# Visualizing the activation maps



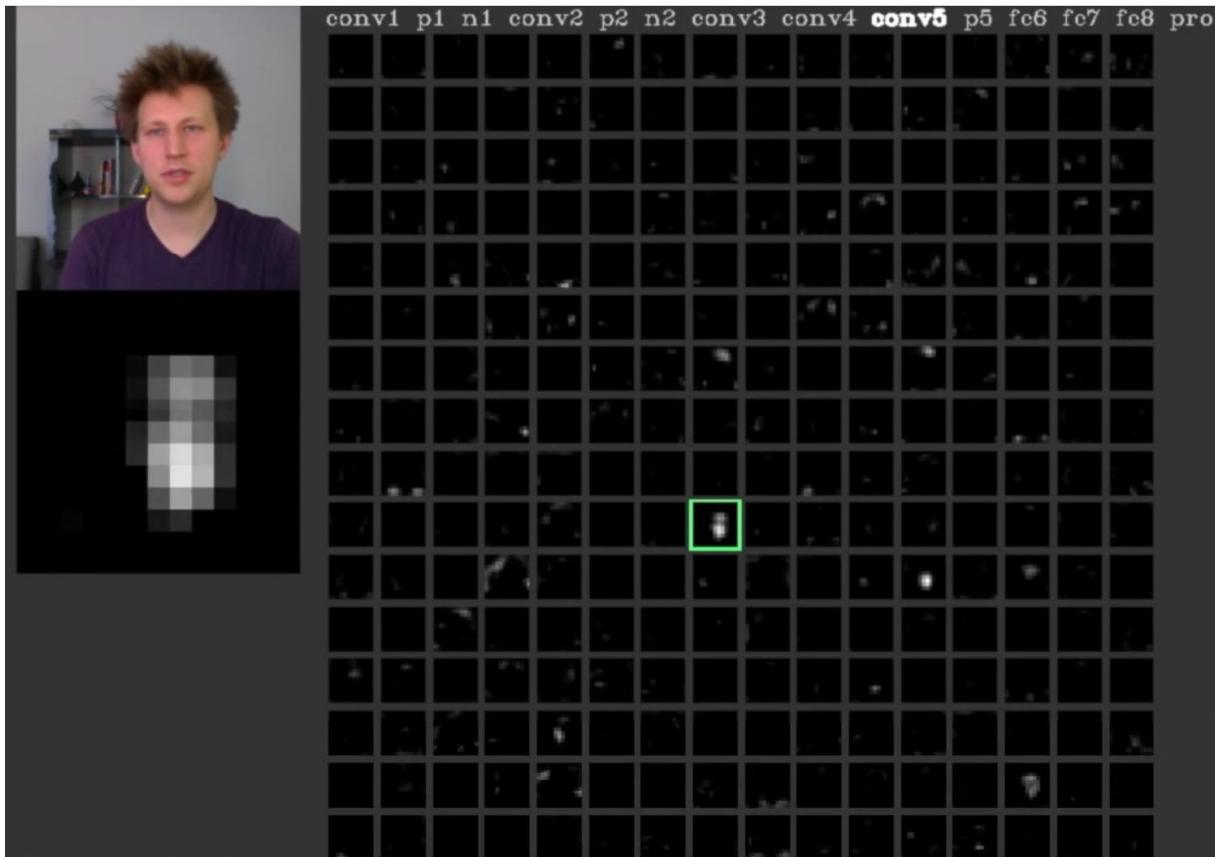
Visualizing the activation maps

# Visualizing the activation maps



Activation maps on the first CONV layer (left), and the 5th CONV layer (right)  
of a trained AlexNet of a cat image

# Visualizing the activation maps



256 Filters (each size: 13x13) in conv5 layer

**Q:** What does it mean that many higher-layer feature maps are blank?

**A:** The pattern encoded by these filters were not found in the input image.

For higher layers, the feature maps become less interpretable.

# References

- Online Lecture: “Visualizing what ConvNets learn”.  
<https://cs231n.github.io/understanding-cnn/>
- Video lecture in University of Stanford. “Visualizing and Understanding”. 2017.  
<https://www.youtube.com/watch?v=6wcs6szJWMY>
- Blog by Jason Yosinski. “Understanding Neural Networks Through Deep Visualization”. 2015. <https://yosinski.com/deepvis>

# Topic 2: Visualizing Filters

# Visualizing Filters

Visualizing the filters directly

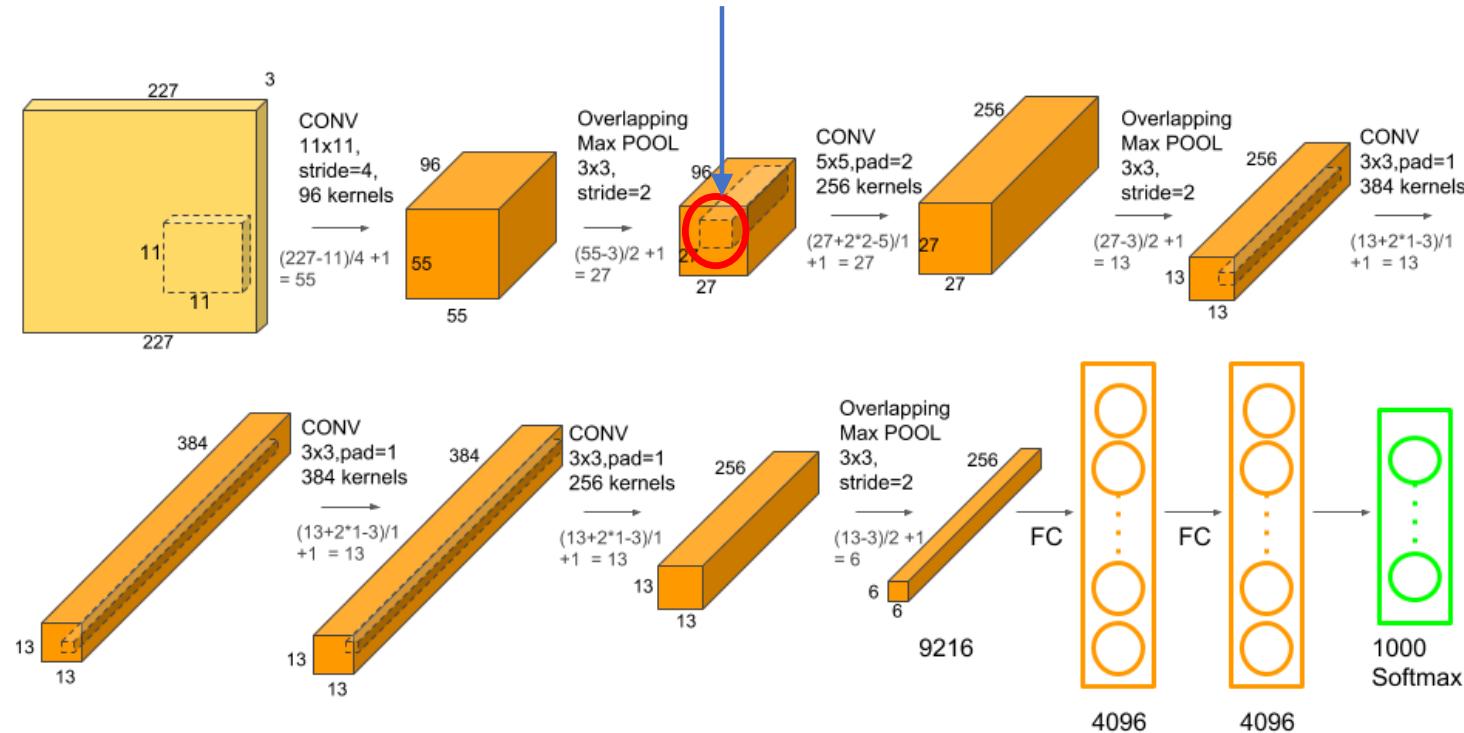
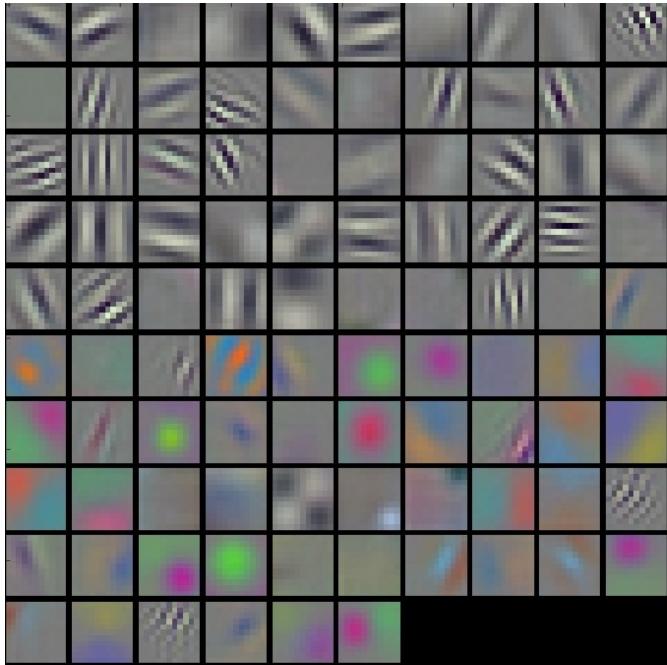


Image from: <https://learnopencv.com/understanding-alexnet/>

# Visualizing Filters

1<sup>st</sup> Conv Layer of AlexNet



2<sup>nd</sup> Conv Layer of AlexNet

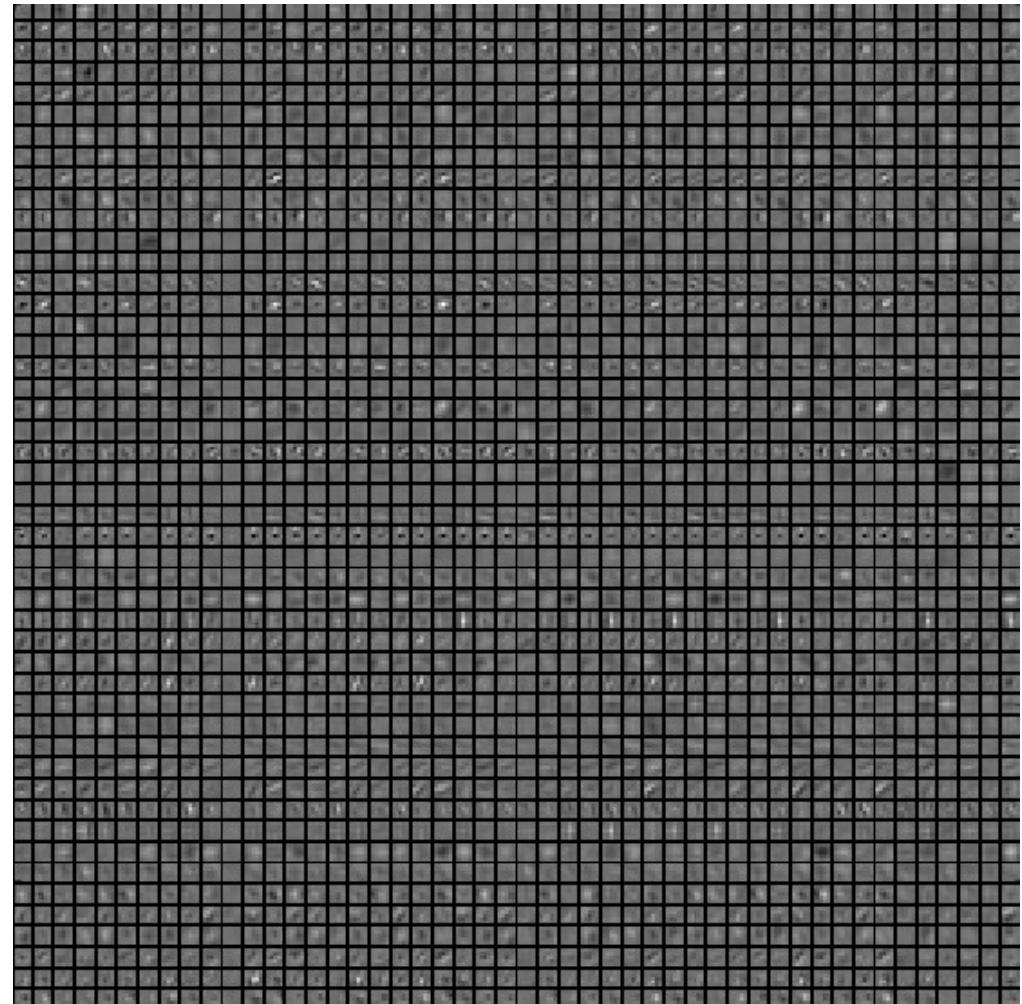
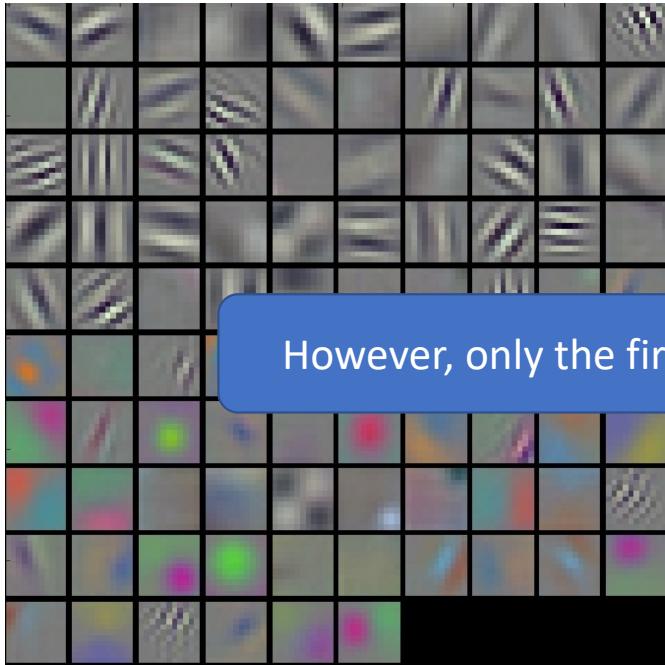


Image from: <https://cs231n.github.io/understanding-cnn/>

- Helpful to check if the network is trained properly (no noisy pattern)

# Visualizing Filters

1<sup>st</sup> Conv Layer of AlexNet



However, only the first layer is interpretable by visualizing filters directly.

- Helpful to check if the network is trained properly (no noisy pattern)

2<sup>nd</sup> Conv Layer of AlexNet

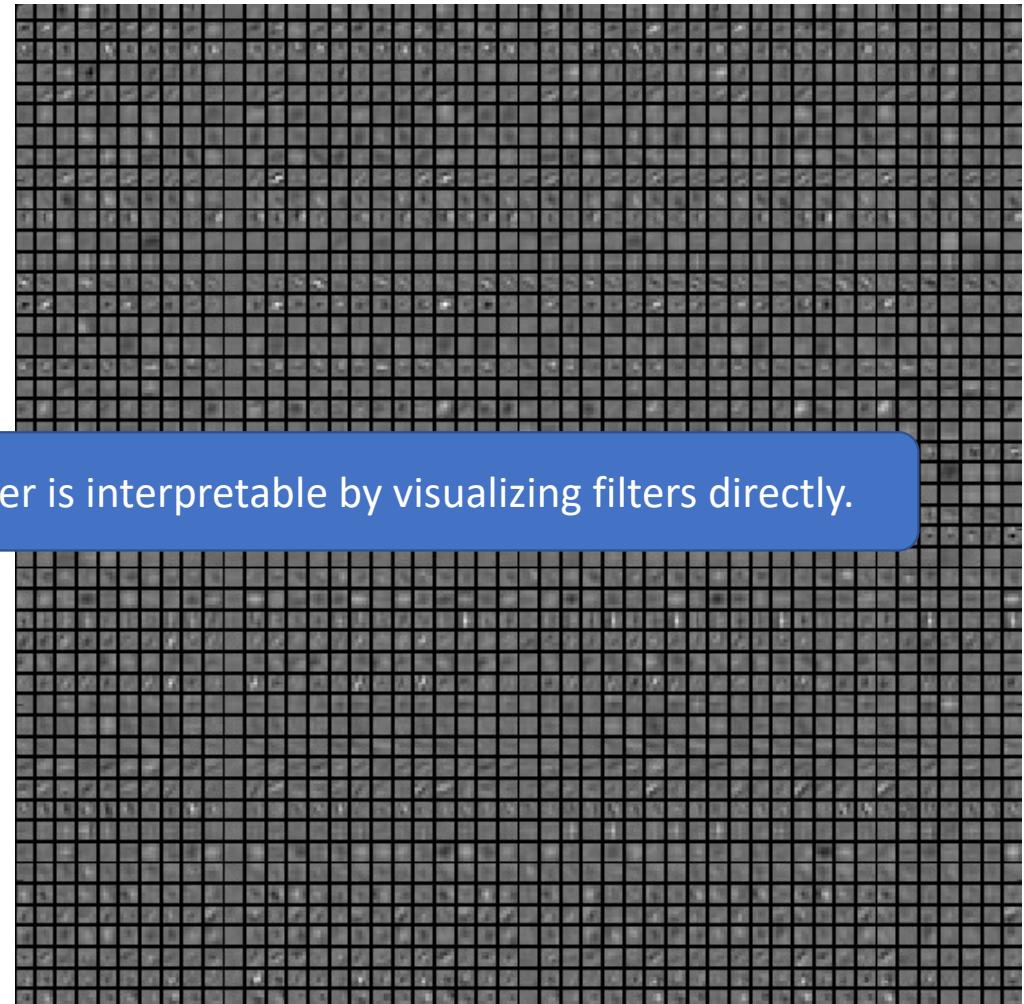
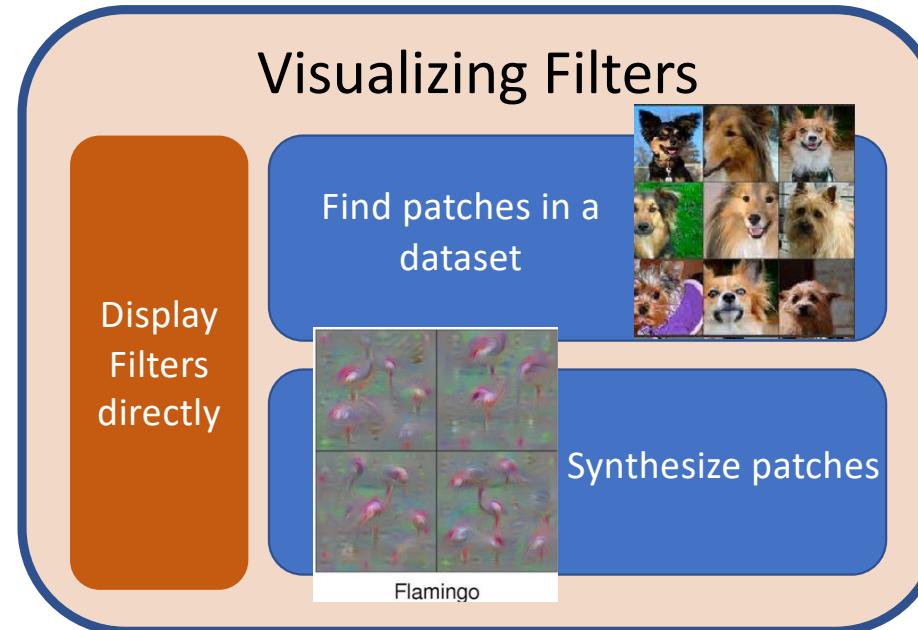


Image from: <https://cs231n.github.io/understanding-cnn/>

# Visualizing Filters implicitly



- Instead of visualizing filters directly, visualize what sort of input maximizes the activation of one filter.
- We get an idea of what pattern each filter has learnt to extract.

# Maximally Activating Patches

- Find patches produce maximal activation for a certain filter

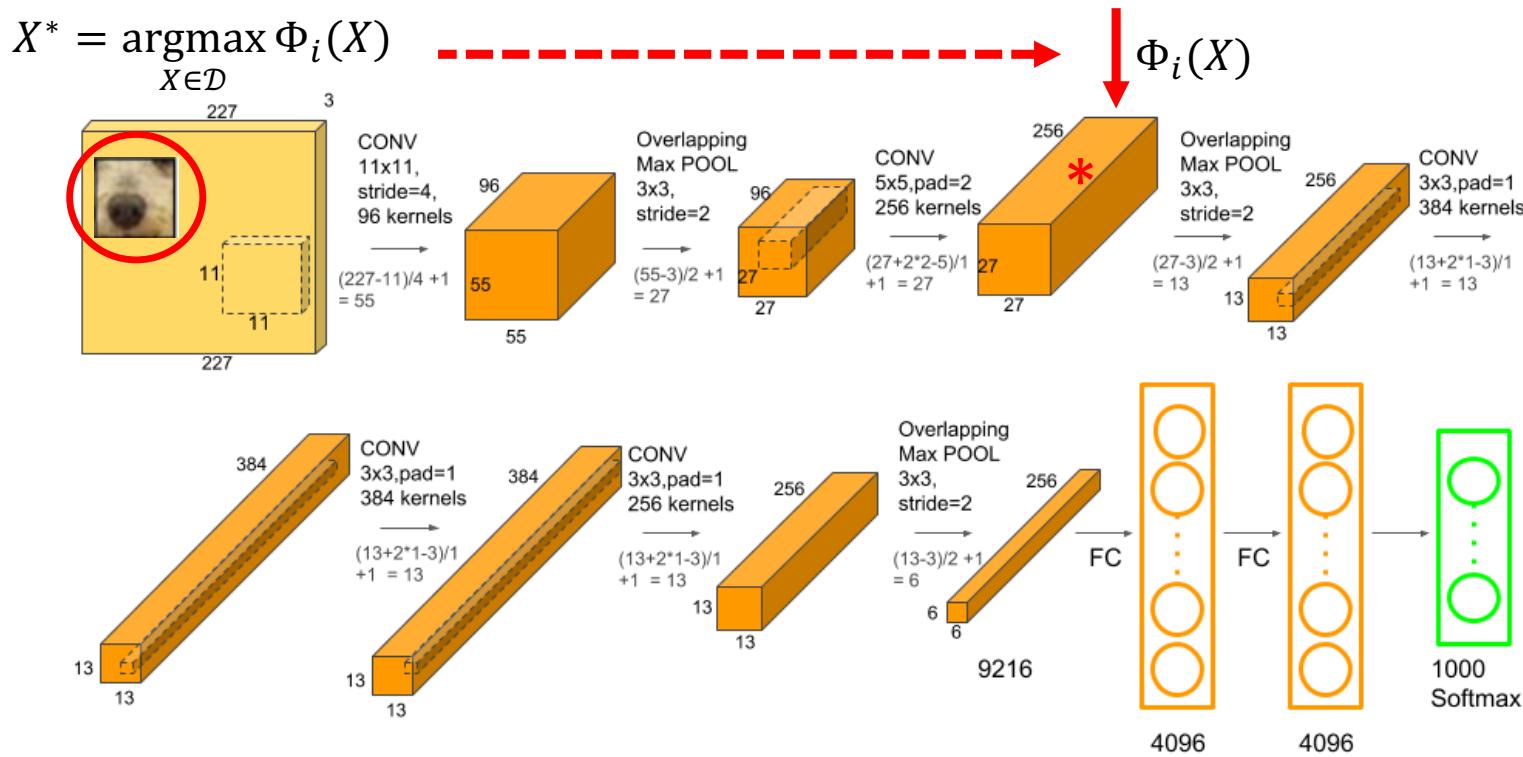
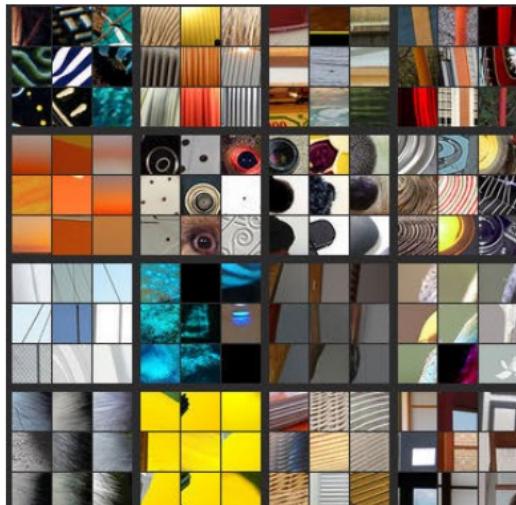


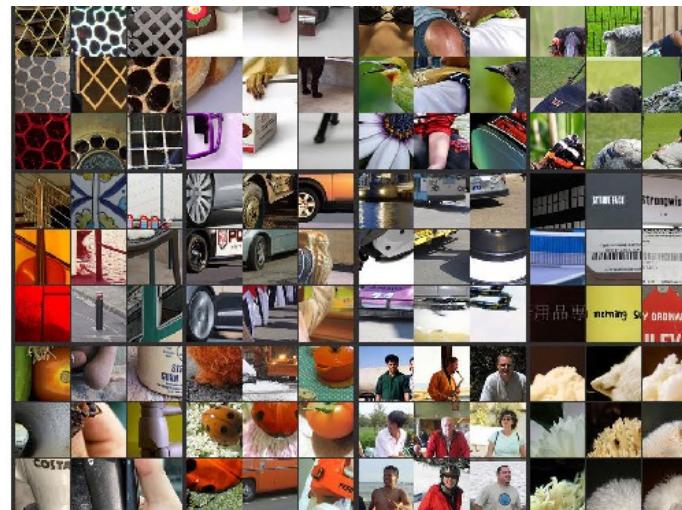
Image from: <https://learnopencv.com/understanding-alexnet/>

# Maximally Activating Patches

Top 9 image patches (from ImageNet) maximally activating each filter



Layer 2



Layer 3

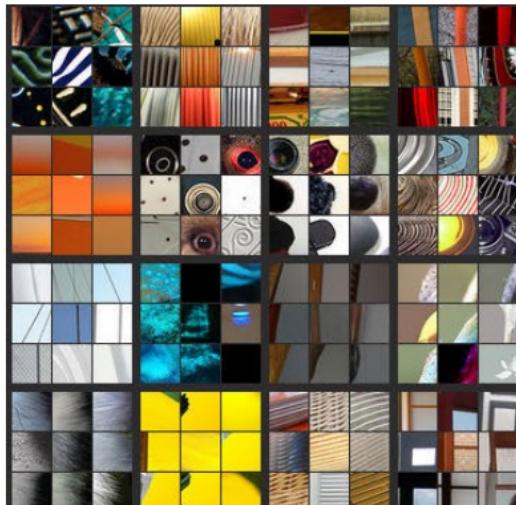


Layer 5

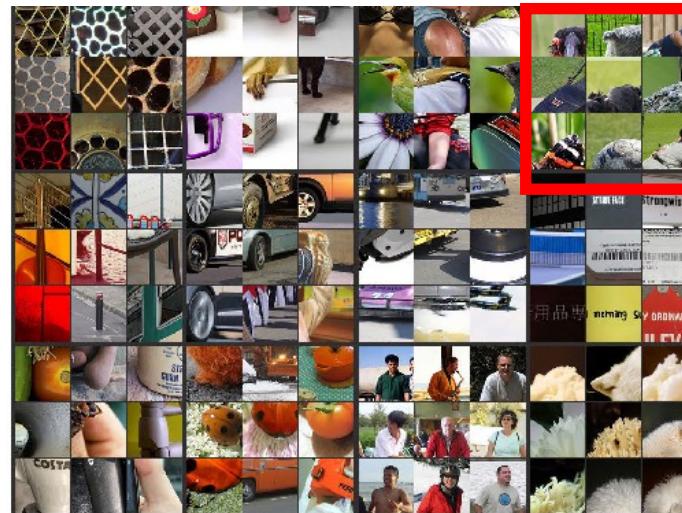
Zeiler, Fergus. "Visualizing and Understanding Convolutional Networks". ECCV14

# Maximally Activating Patches

Top 9 image patches (from ImageNet) maximally activating each filter



Layer 2



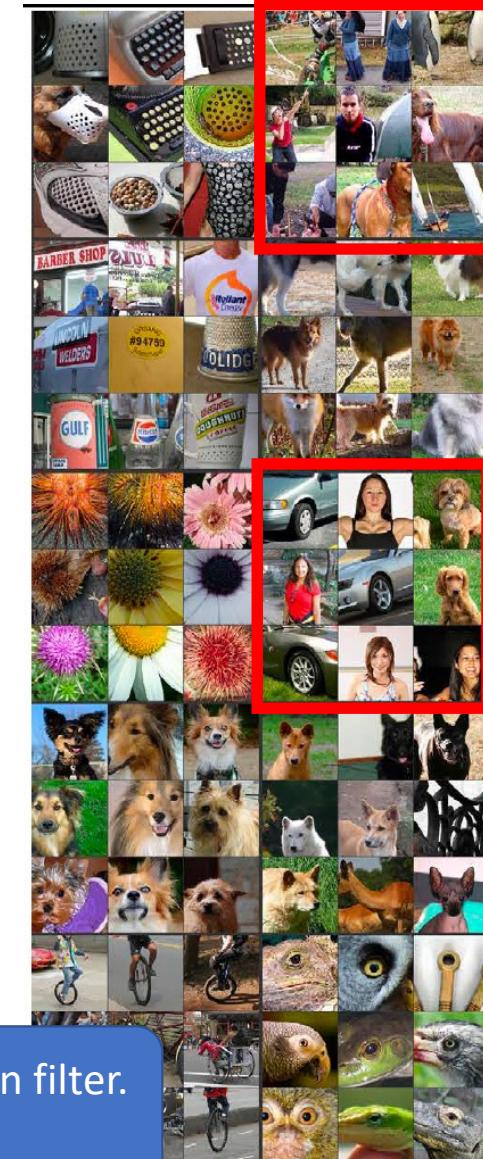
Layer 3

**Q1:** What are these 9 patches in common?

**Q2:** What is the discriminative part for these 9 patches?

We need also visualize which pixels in the patch “excite” a given filter.

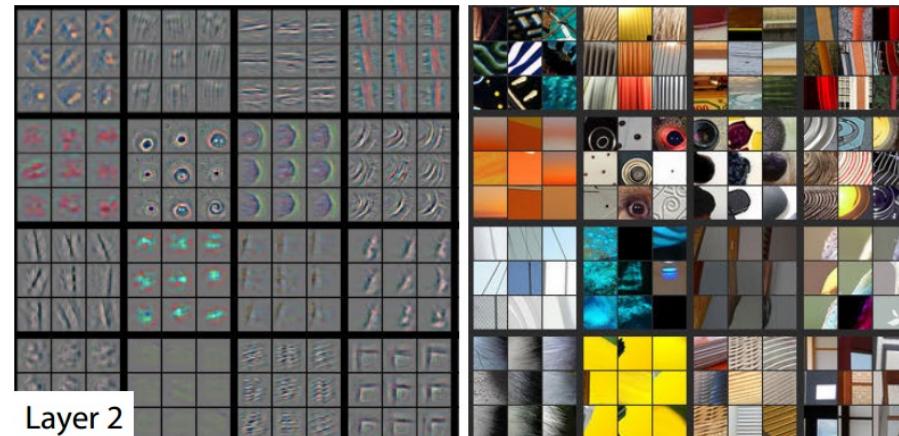
Solution: **Deconvnet; Guided-backpropagation**



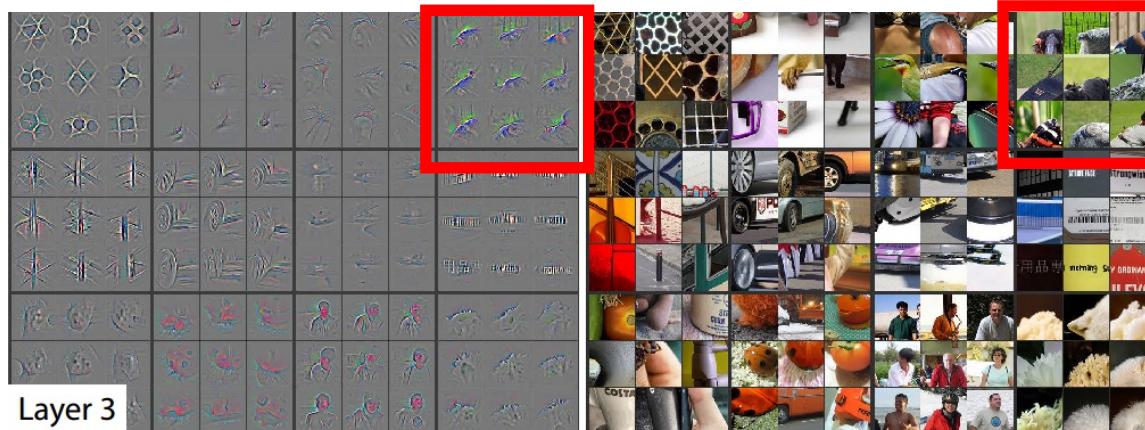
Layer 5

# Decovnet [Zeiler-ECCV14]

Deconvolutional Network (Decovnet) can be thought of as a ConvNet that uses the same components (conv, pooling) but in reverse, so instead of mapping pixels to features, deconvnet projects the feature activations back to the input pixel space.



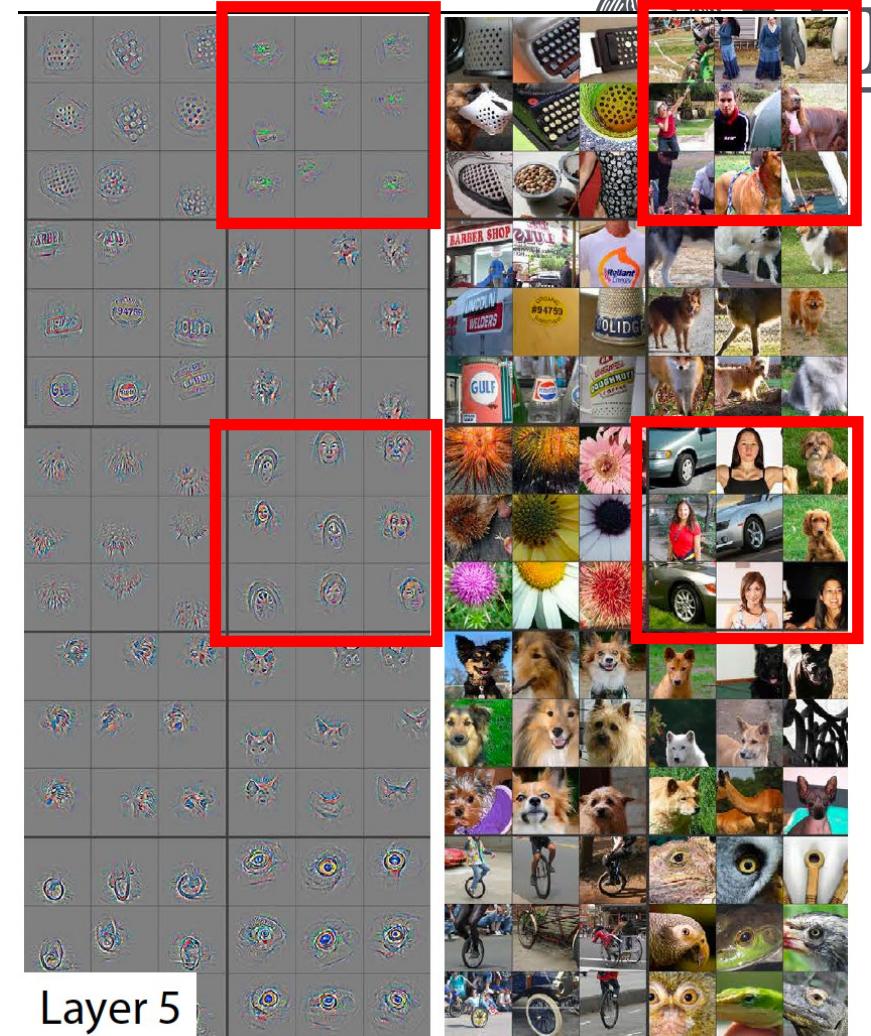
Layer 2



Layer 3

# Decovnet [Zeiler-ECCV14]

- Strong grouping within each feature map
- Greater invariance at higher layers
- Exaggeration of discriminative parts of the patch



# Synthesize an image to maximize the activation

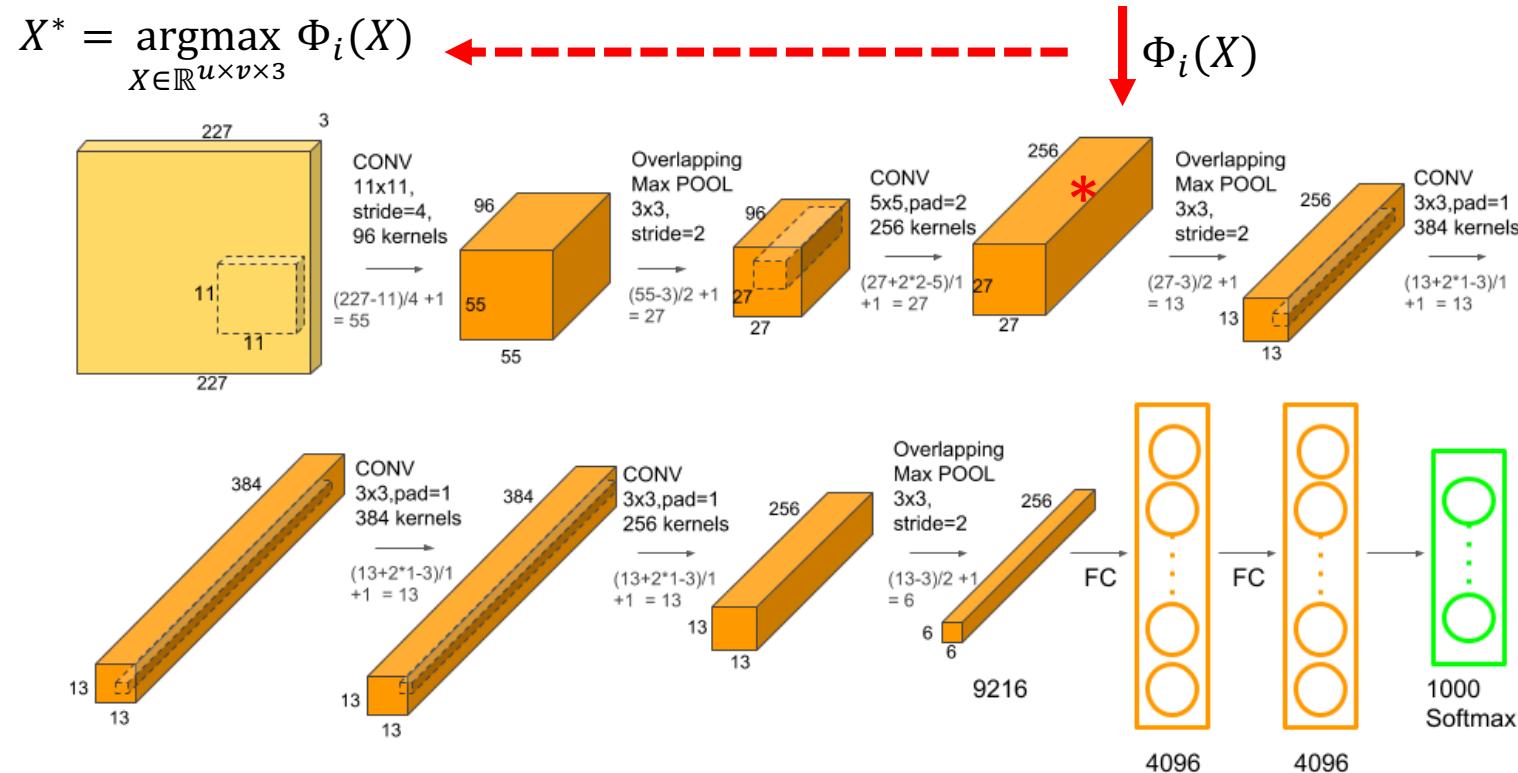


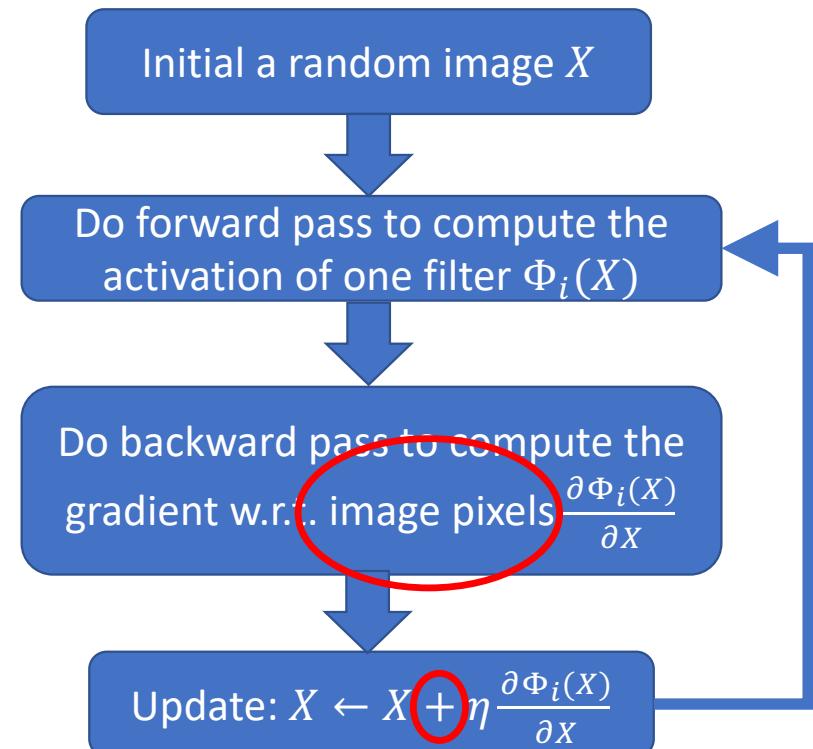
Image from: <https://learnopencv.com/understanding-alexnet/>

# Synthesize an image to maximize the activation

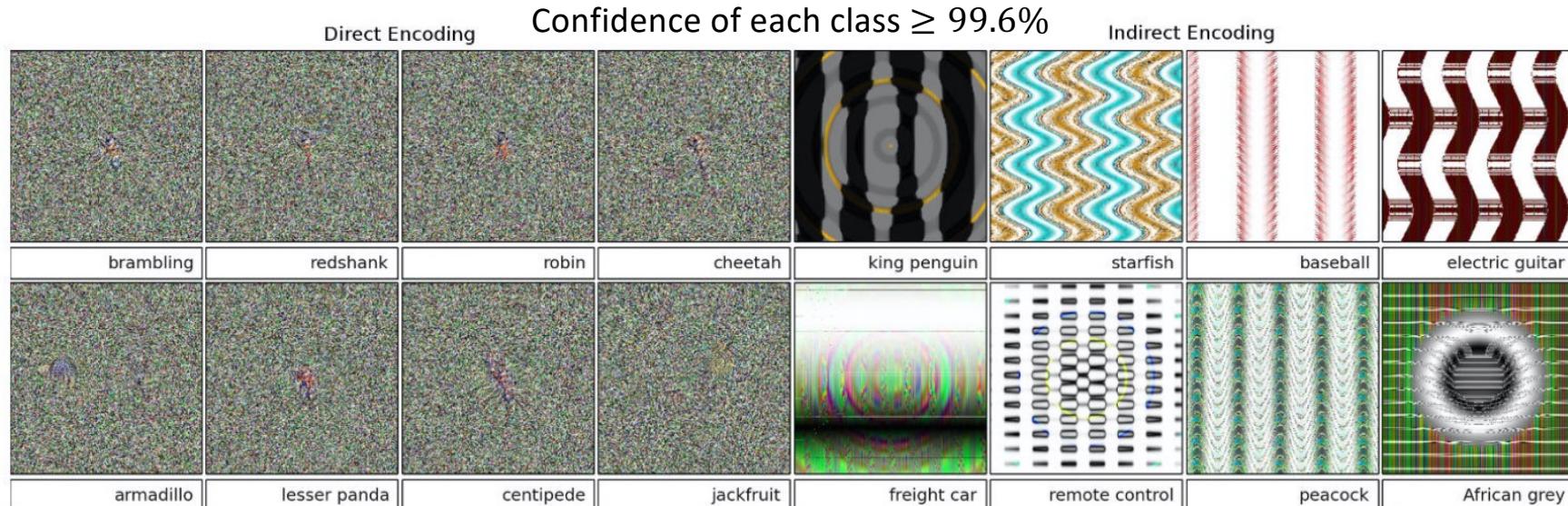
$$X^* = \underset{X \in \mathbb{R}^{u \times v \times 3}}{\operatorname{argmax}} \Phi_i(X)$$

Using **gradient ascent** to generate an image to maximize the activation of one filter.

**Q:** What is the difference from gradient descent of training neural networks?



# Synthesize an image to maximize the activation



But, these unrecognizable images do not help much for understanding filters.

That's because it is lack of **natural image prior**.

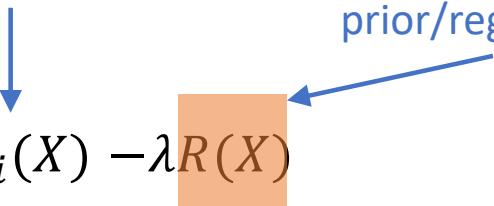
# Visualizing filters with regularizations

- To produce more recognizable images, we need

$$X^* = \underset{X \in \mathbb{R}^{u \times v \times 3}}{\operatorname{argmax}} \Phi_i(X) - \lambda R(X)$$

The activation value

Natural image prior/regularizer



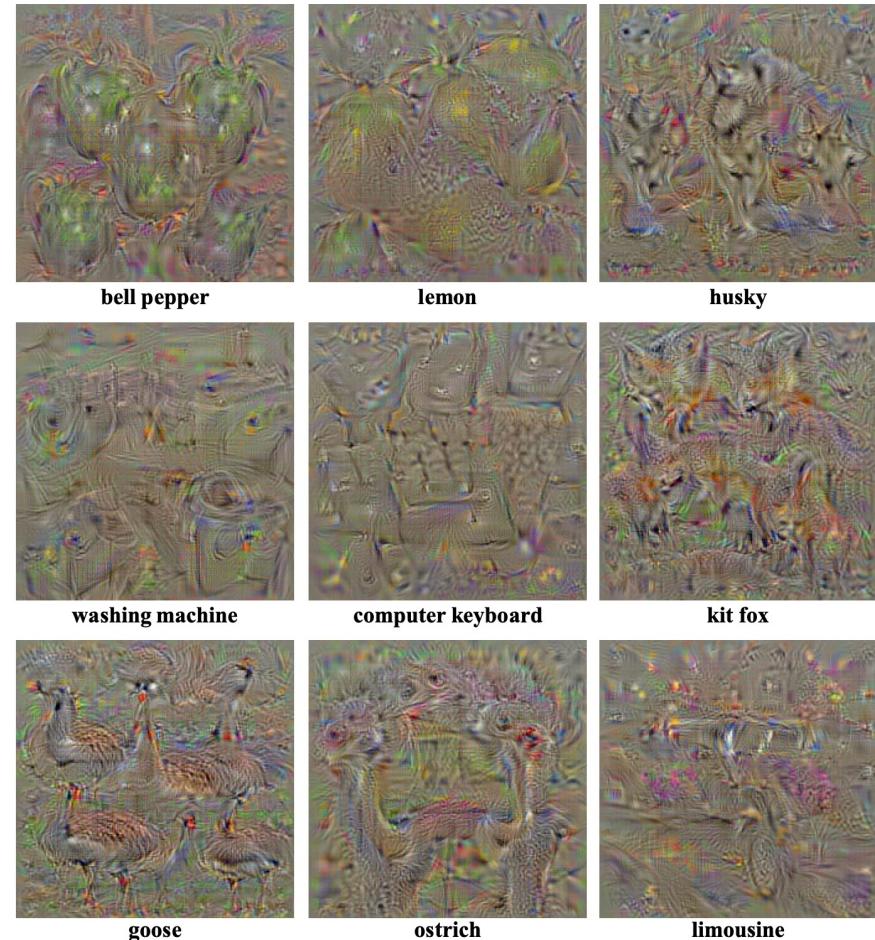
# Visualizing filters with regularizations

- $L_2$ -regularizer [Simonyan-2014]

The score of the class  $c$   
(before softmax)

$$I^* = \underset{I \in \mathbb{R}^{H \times W \times 3}}{\operatorname{argmax}} S_c(I) - \lambda \|I\|_2^2$$

$L_2$ -regularizer



# Visualizing filters with regularizations

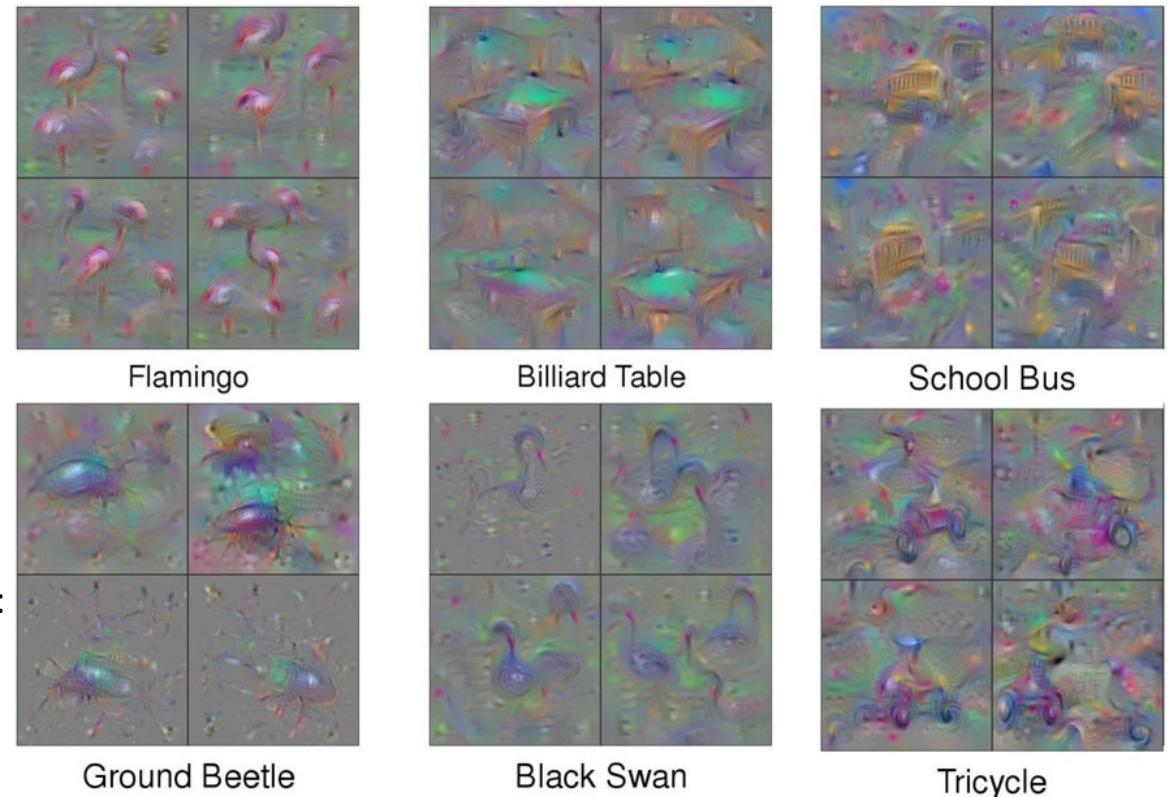
- Better regulariser [Yosinski-2015]

$$X^* = \underset{X \in \mathbb{R}^{u \times v \times 3}}{\operatorname{argmax}} \Phi_i(X) - \lambda \|X\|_2^2$$

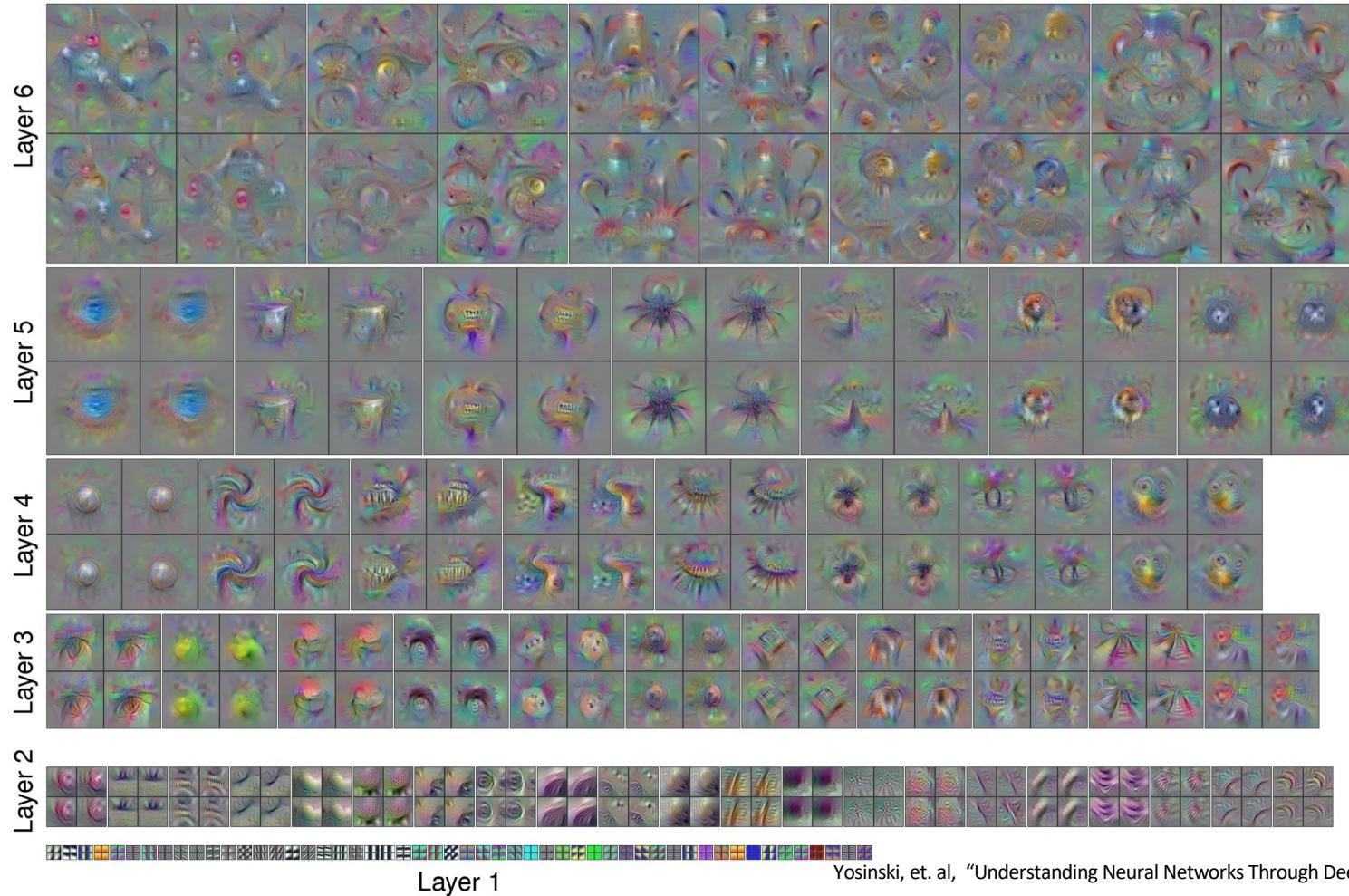
The activation value  
↓  
 $L_2$ -regularizer ↑

Not only penalize L2 norm of the image, also periodically:

- Gaussian blur image
- Clip pixels with small values to 0
- Clip pixels with small gradients to 0

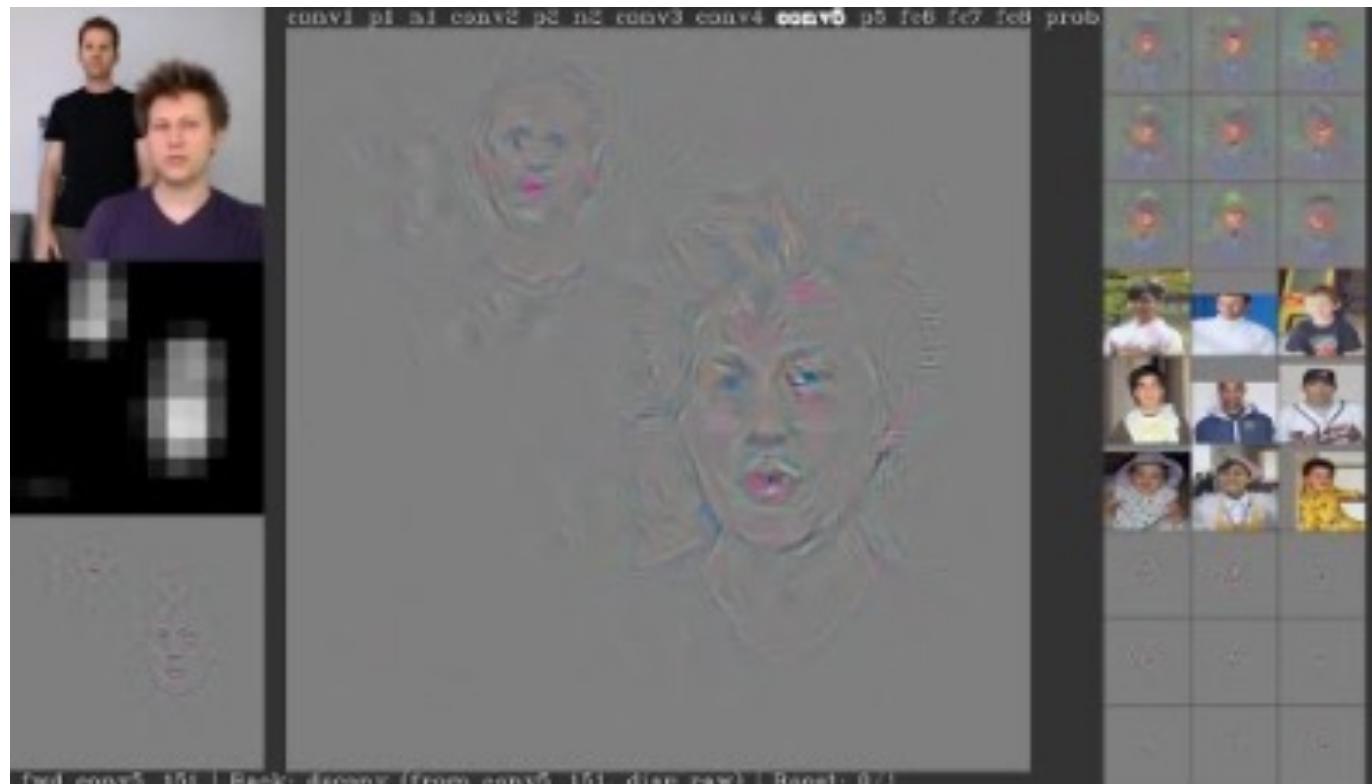


# Visualizing filters with regularizations



Yosinski, et. al, "Understanding Neural Networks Through Deep Visualization ", ICLR Workshop 2015.

# Deep Visualization Toolbox



Video from: <https://yosinski.com/deepvis>

# Summary

Find patches in the dataset



$$X^* = \operatorname{argmax}_{X \in \mathcal{D}} \Phi_i(X)$$

Synthesize patches



$$X^* = \operatorname{argmax}_{X \in \mathbb{R}^{u \times v \times 3}} \Phi_i(X)$$

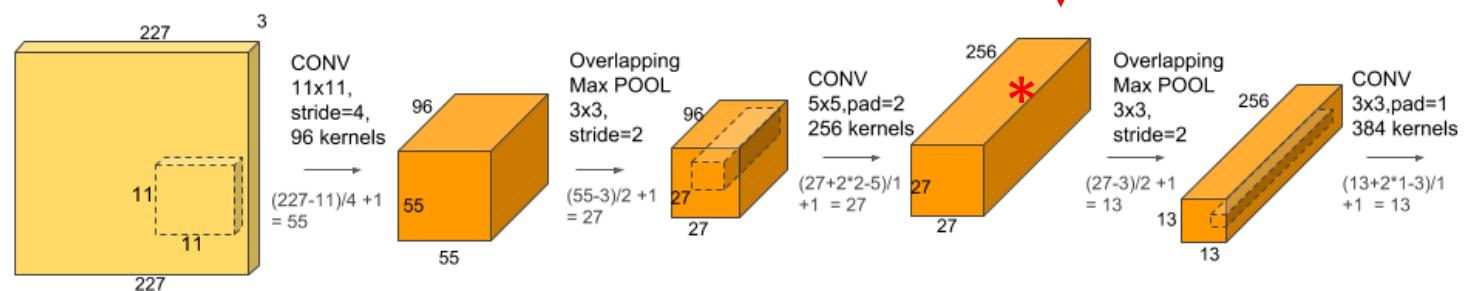
**Method:** Gradient accent

**Trick:** Adding regularizer

$$X^* = \operatorname{argmax}_X \Phi_i(X)$$



$$\Phi_i(X)$$



# Reference

- Online Lecture: “Visualizing what ConvNets learn”.  
<https://cs231n.github.io/understanding-cnn/>
- Video lecture in University of Stanford. “Visualizing and Understanding”. 2017.  
<https://www.youtube.com/watch?v=6wcs6szJWMY>
- Blog by Jason Yosinski. “Understanding Neural Networks Through Deep Visualization”. 2015. <https://yosinski.com/deepvis>

# Topic 3: Neural Style Transfer

The activation value

$$X^* = \underset{X \in \mathbb{R}^{u \times v \times 3}}{\operatorname{argmax}} \Phi_i(X) - \lambda R(X)$$

Natural image regularizer/prior

**Visualizing Filters**  
Visualize what sort of input maximizes the activation of each filter

$$X^* = \underset{X \in \mathbb{R}^{H \times W \times 3}}{\operatorname{argmin}} \ell(\Phi(X), \Phi(X_0)) + \lambda R(X)$$

Loss (matches the feature)  
 $\Phi(X): \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}^d$  is an image representation of a certain layer  
 (a feature vector)

Natural image regularizer/prior

**Feature Inversion**  
Reconstruct an image whose feature representation best matches the original image

# Feature Inversion

- Reconstruct an image whose feature representation best matches the original image's.

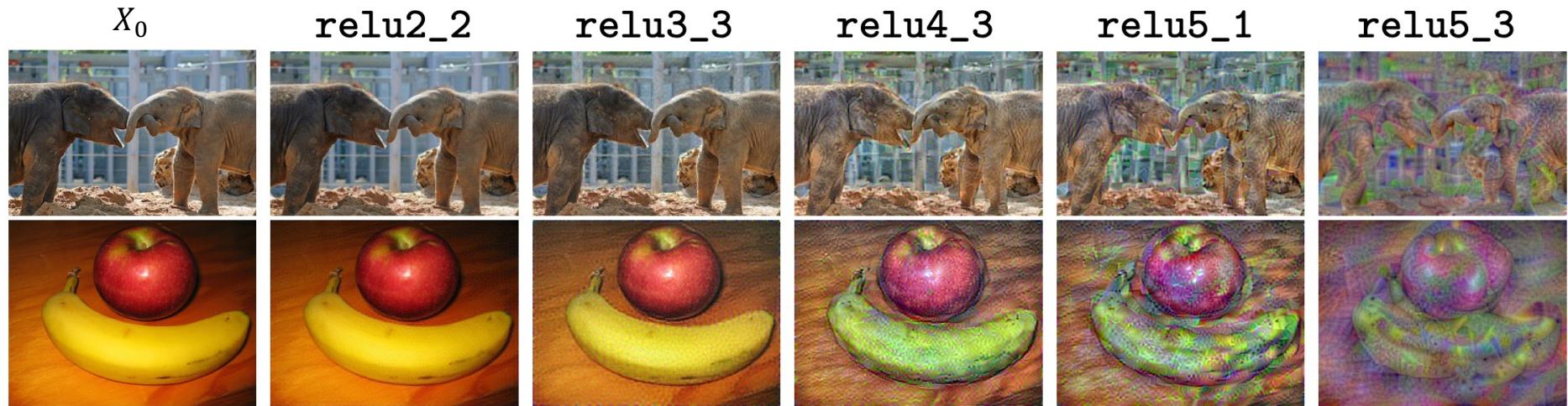
$$X^* = \underset{X \in \mathbb{R}^{H \times W \times 3}}{\operatorname{argmin}} \ell(\Phi(X), \Phi(X_0)) + \lambda R(X)$$

↗
↗

$\ell(\Phi(X), \Phi(X_0)) = \|\Phi(X) - \Phi(X_0)\|^2$   
Feature distance

$R_{V^\beta} = \sum_{i,j} \left( (x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$   
Total Variation regularizer  
(encourages spatial smoothness)

# Feature Inversion



As we reconstruct from higher layers, image content and overall spatial structure are preserved, but color, texture, and exact shape are not.

# Neural Texture Synthesis

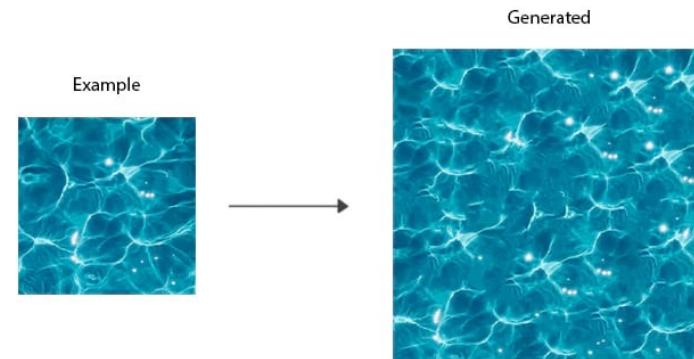
$$X^* = \underset{X \in \mathbb{R}^{H \times W \times 3}}{\operatorname{argmin}} \ell(\Phi(X), \Phi(X_0))$$



$$X^* = \underset{X \in \mathbb{R}^{H \times W \times 3}}{\operatorname{argmin}} \ell(G(X), G(X_0))$$



Texture representation using CNN



## Neural Texture Synthesis

# Gram Matrix

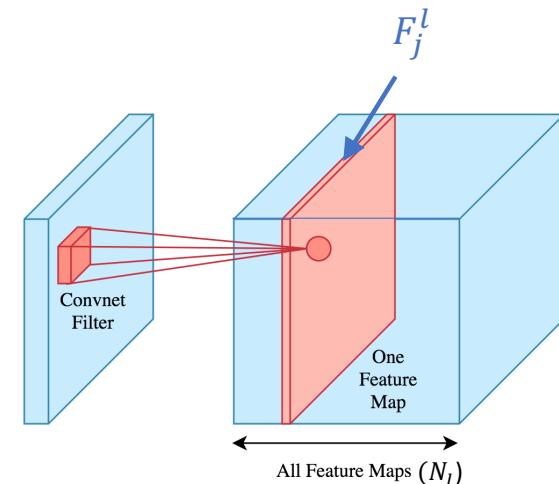
$F_j^l$ :  $j$ -th feature map at layer  $l$

$F_i^l$ :  $i$ -th feature map at layer  $l$

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

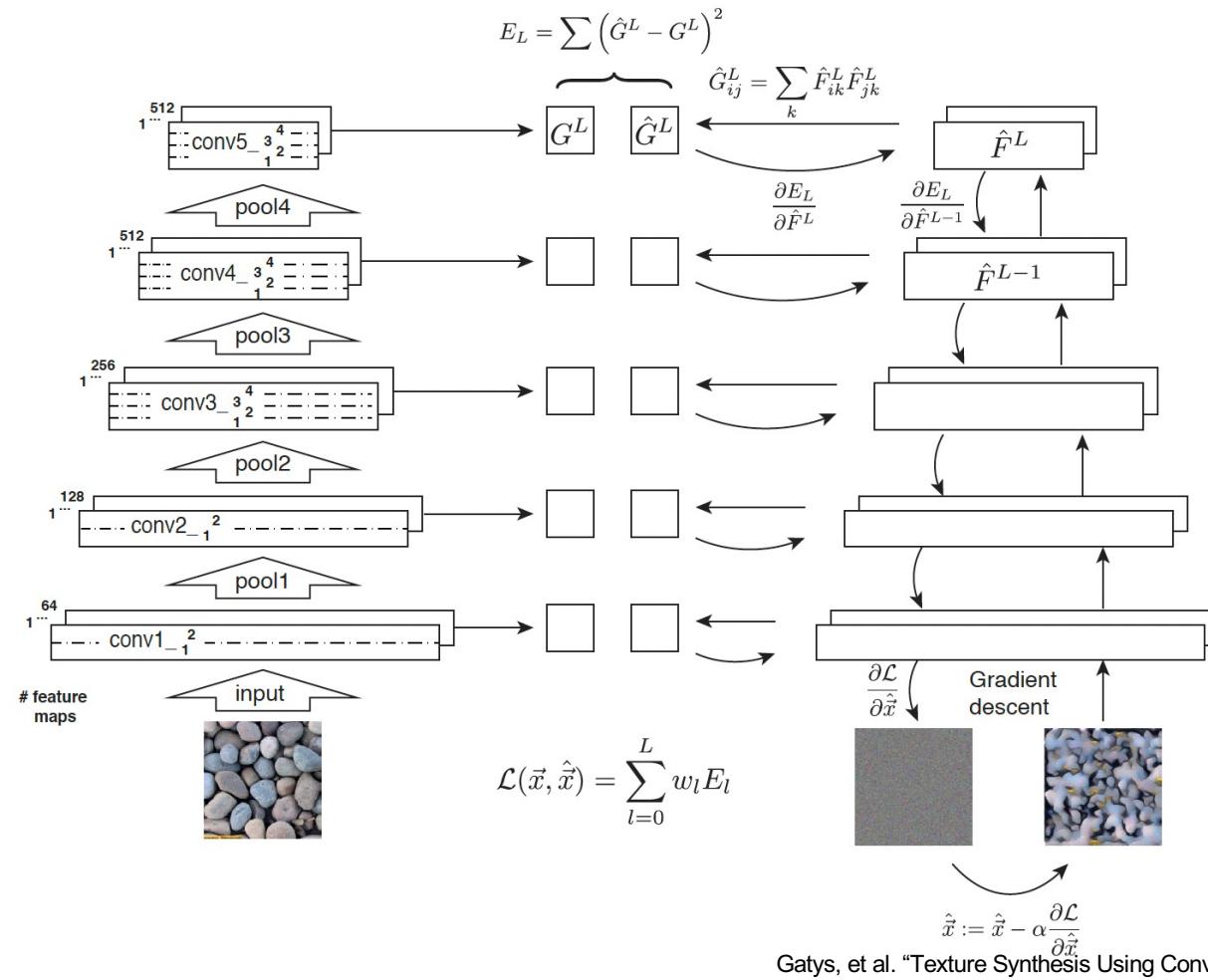
the inner product between two feature maps.

Gram matrix:  $G^l \in \mathbb{R}^{N_l \times N_l}$

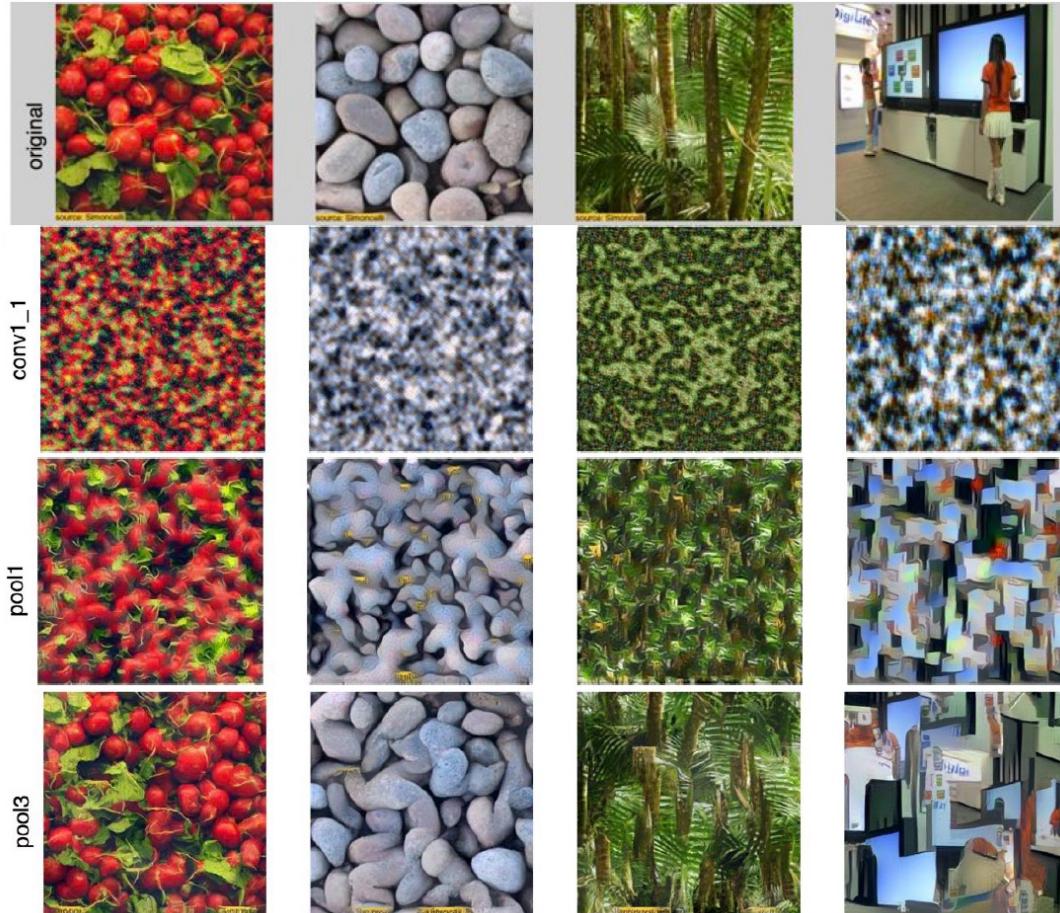


Texture features could be represented by a set of Gram Matrixes of conv layers:  $\{G^1, G^2, \dots, G^L\}$

# Neural Texture Synthesis

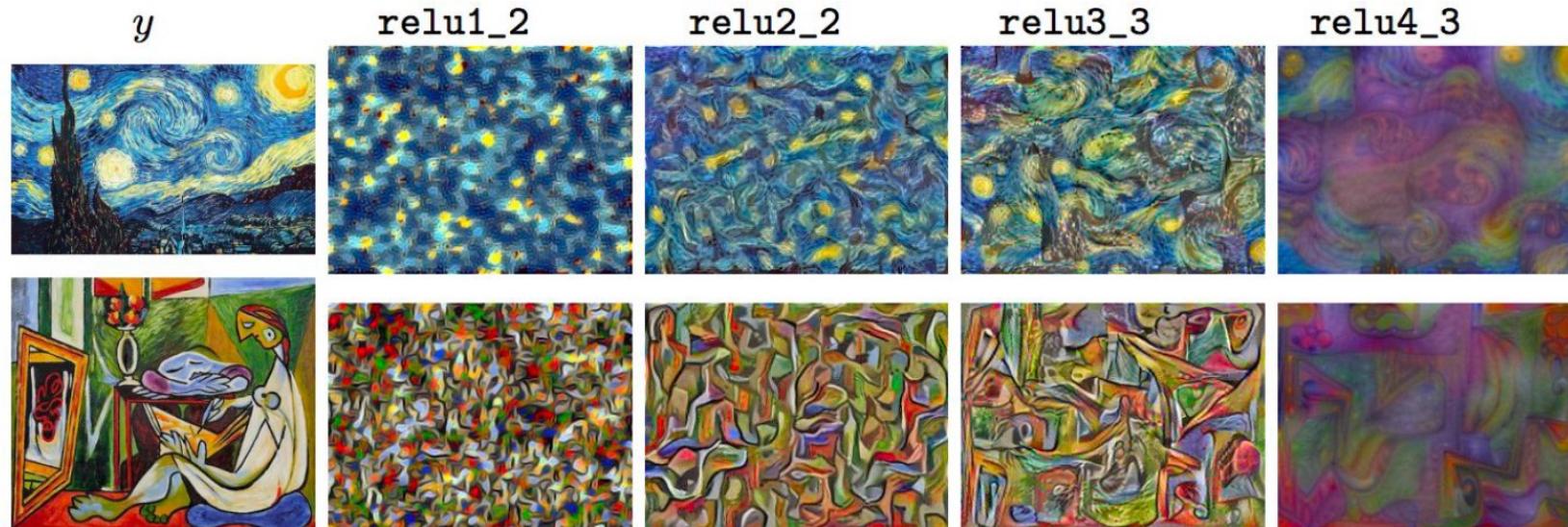


# Neural Texture Synthesis



Gatys, et al. "Texture Synthesis Using Convolutional Neural Networks". NIPS2015.

# When texture synthesis applies to art ...

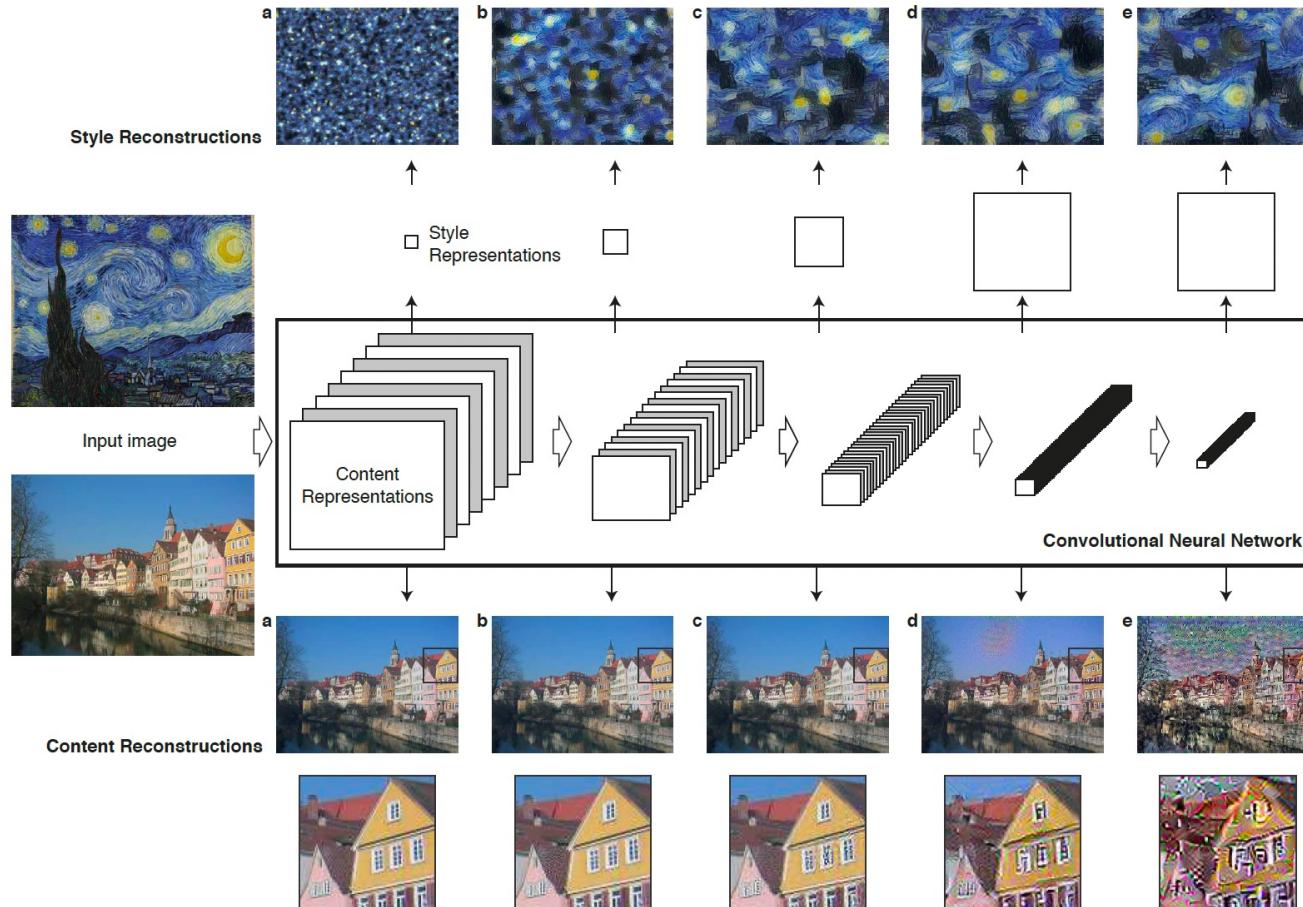


**Q:** Can we transfer a particular art style to a new image?



Gatys, et al. "Image style transfer using convolutional neural networks", CVPR 2016.  
 Gatys, et al. "Texture Synthesis Using Convolutional Neural Networks". NIPS2015.

# Neural Style Transfer



$$X^* = \underset{X \in \mathbb{R}^{H \times W \times 3}}{\operatorname{argmin}} \ell(G(X), G(X_0))$$

### Texture synthesis

Reconstruct an image whose texture representation best matches the original image

$$X^* = \underset{X \in \mathbb{R}^{H \times W \times 3}}{\operatorname{argmin}} \ell(\Phi(X), \Phi(X_0))$$

### Feature Inversion

Reconstruct an image whose feature representation best matches the original image

# Neural Style Transfer

Content Image



Style Image



Style Transfer!



$$P_{ij}^l$$

$$A_{ij}^l$$

$$G_{ij}^l$$

$$\mathcal{L}_{\text{content}}$$

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{style}} + \beta \mathcal{L}_{\text{content}}$$

Match the texture (Gram-matrix):

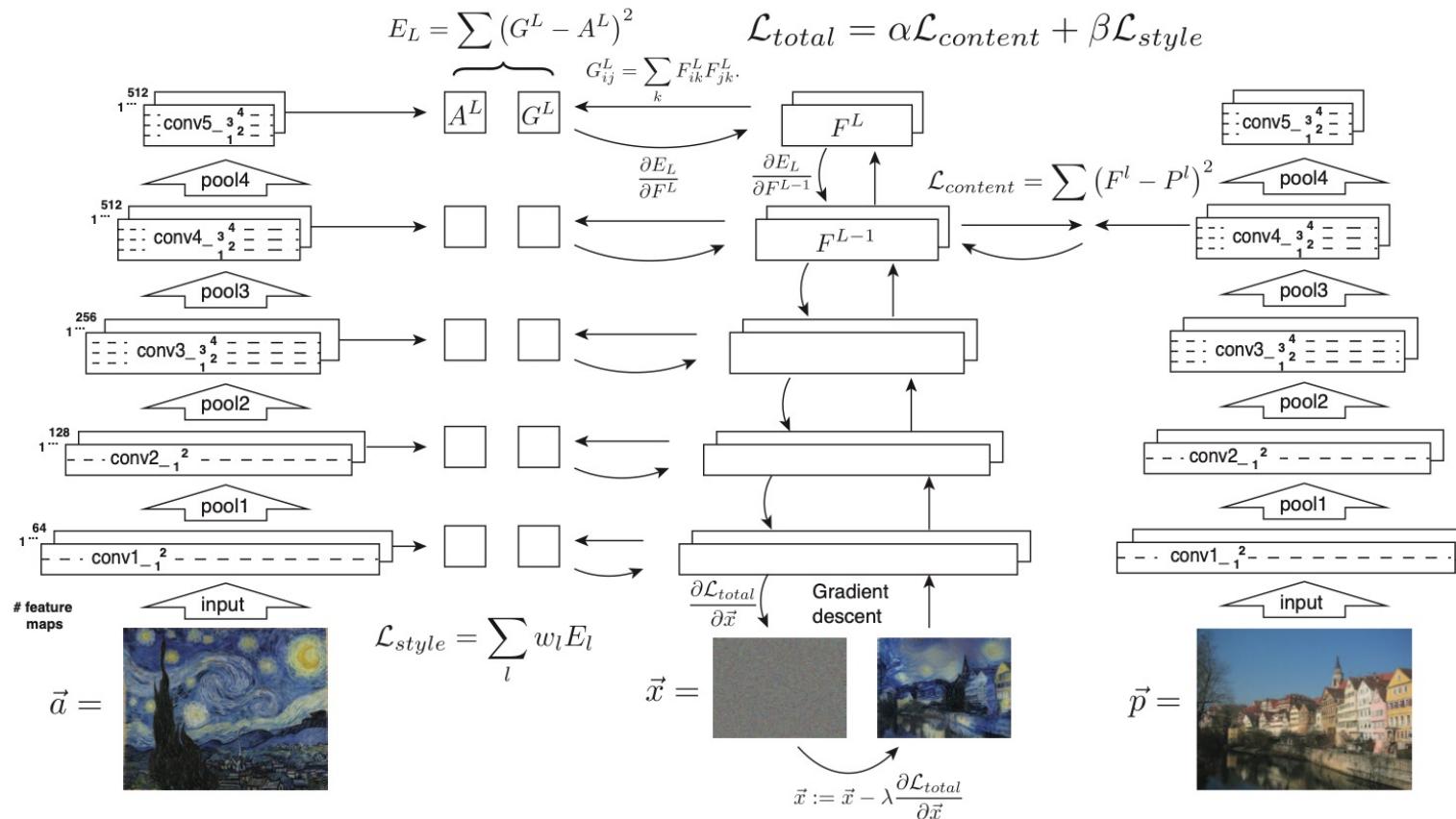
$$\sum_{l=1}^L \sum_{ij} (G_{ij}^l - A_{ij}^l)^2$$

Match the feature:

$$\sum_{ij} (F_{ij}^l - P_{ij}^l)^2$$

Gatys, et al. "Image style transfer using convolutional neural networks", CVPR 2016.  
Gatys, et al. "Texture Synthesis Using Convolutional Neural Networks". NIPS2015.

# Neural Style Transfer



Gatys, et al. "Image style transfer using convolutional neural networks", CVPR 2016.

# Neural Style Transfer

A



B



C



D



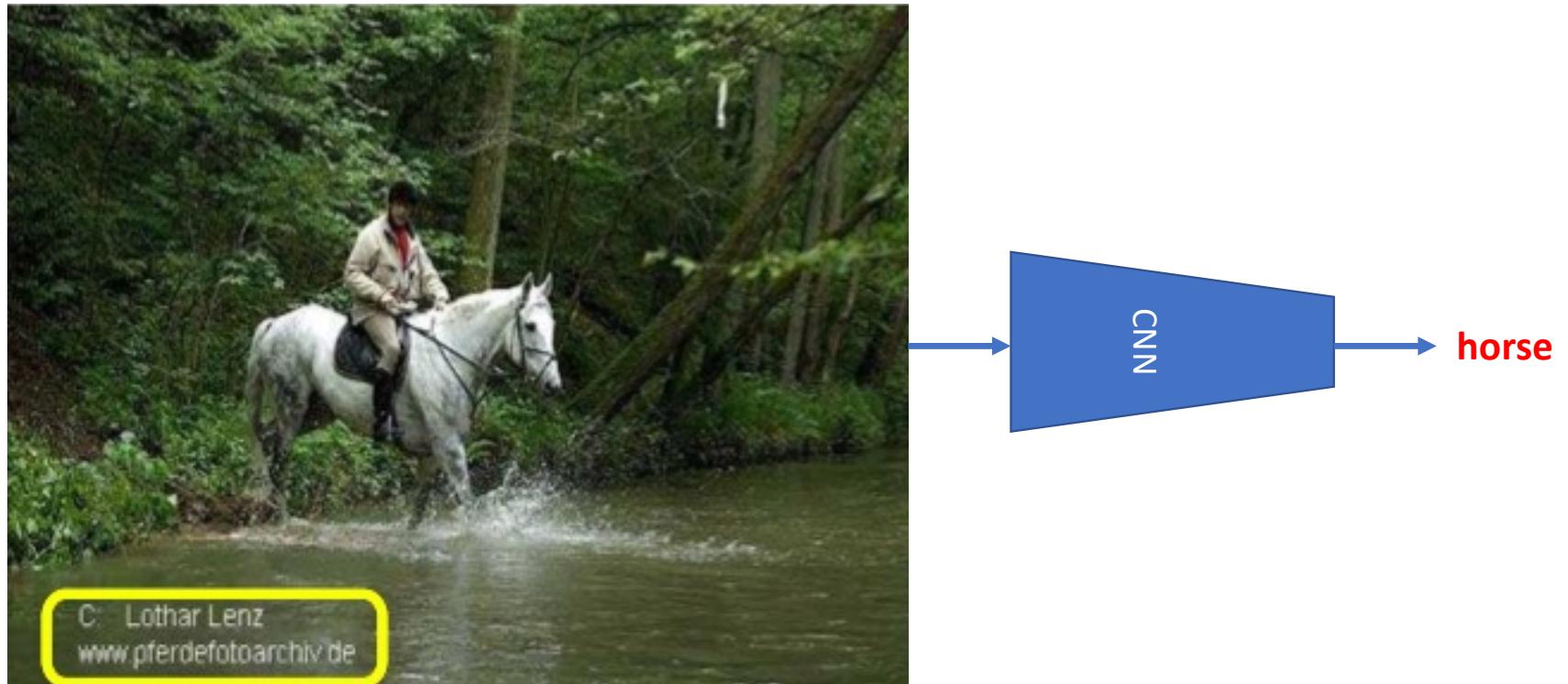
Gatys, et al. "Image style transfer using convolutional neural networks", CVPR 2016.

# References

- Video lecture in University of Stanford. “Visualizing and Understanding”. 2017.  
<https://www.youtube.com/watch?v=6wcs6szJWMY>

# Topic 4: Visualizing Class-Discriminative Heatmap

**Q:** When a CNN model outputs a class for an image, which parts of the image contribute high to the decision?



# Visualizing heat maps



CAM for the cat class

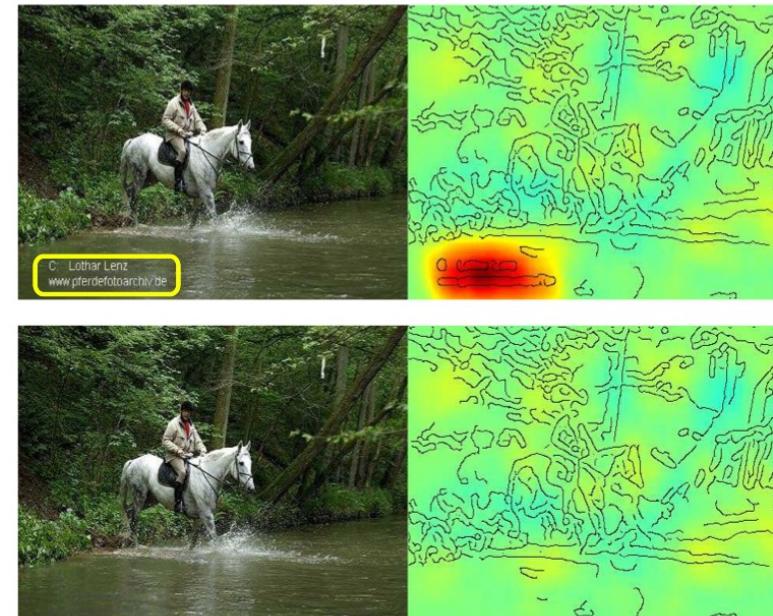
- **Class Activation Mapping (CAM)** is to produce heat maps to highlight class specific region of images, to indicate how important each location is with respect to the given class.

Image from: <https://glassboxmedicine.com/2019/06/11/cnn-heat-maps-class-activation-mapping-cam/>

# Why does this matter?

- Help investigate the traits or features of the input images which had major contribution in producing a certain class
- Identify the weak-points of our models
- Identify dataset bias

Horse-picture from Pascal VOC data set



Source tag present

↓  
Classified as horse

No source tag present

↓  
Not classified as horse

# Visualizing heat maps

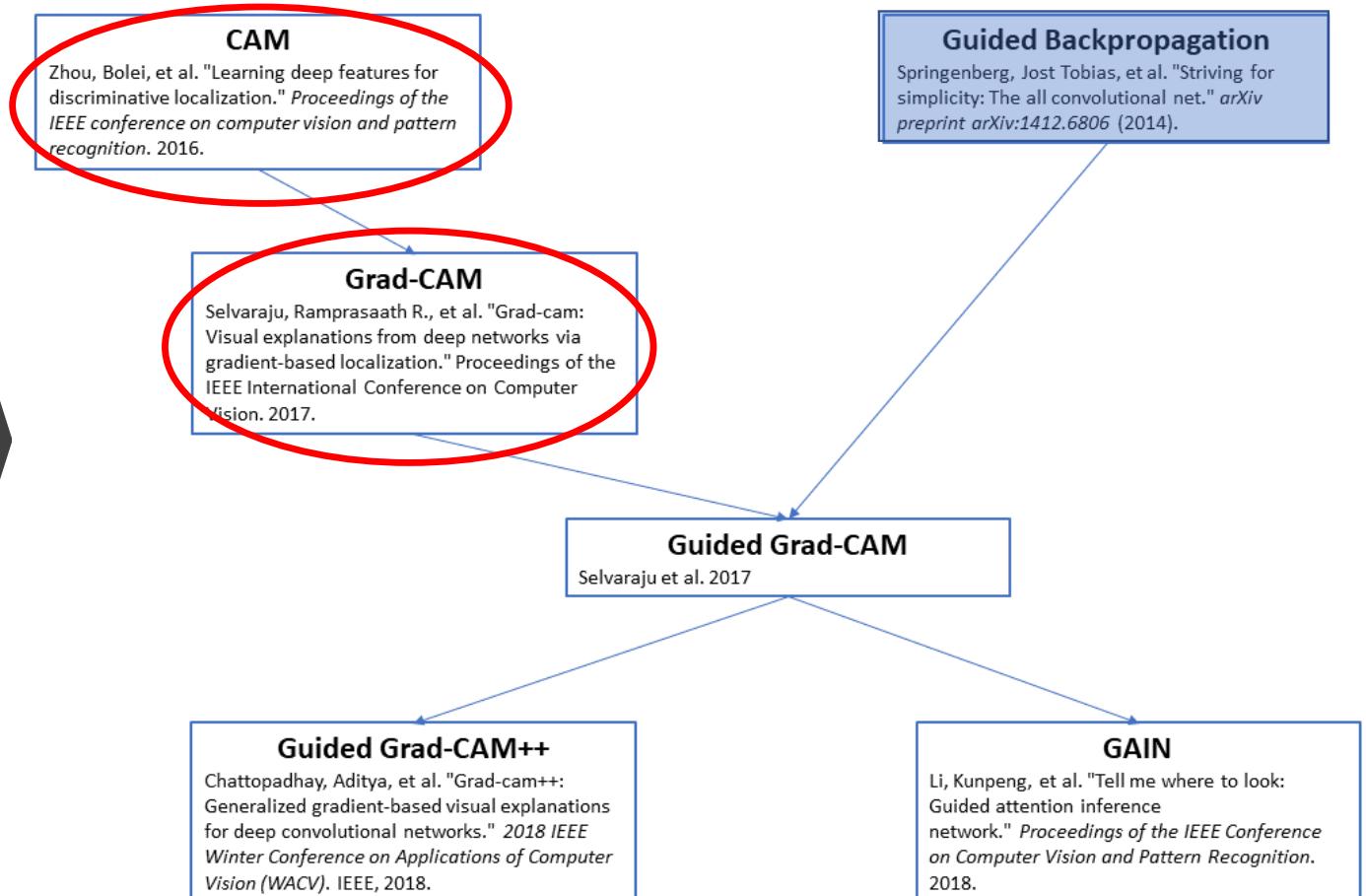
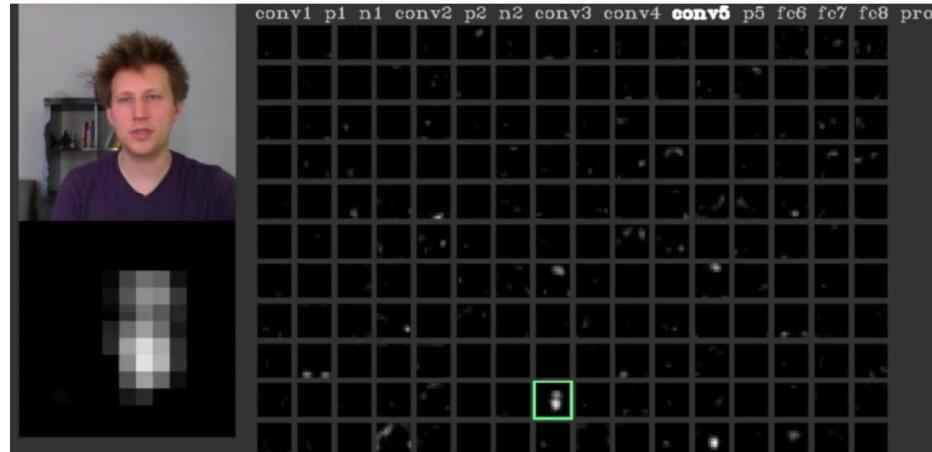
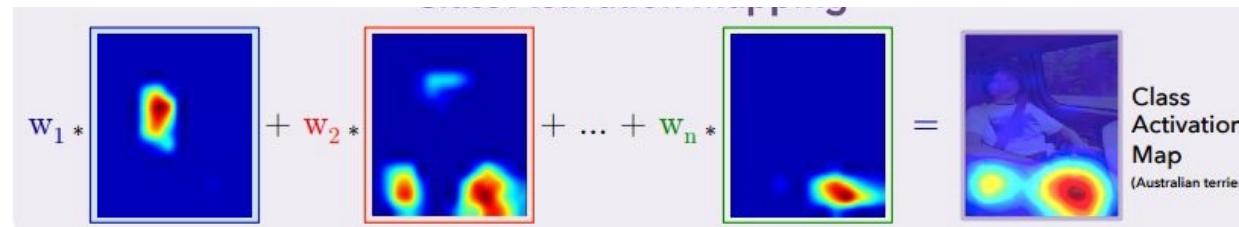


Image from: <https://glassboxmedicine.com/2019/06/11/cnn-heat-maps-class-activation-mapping-cam/>



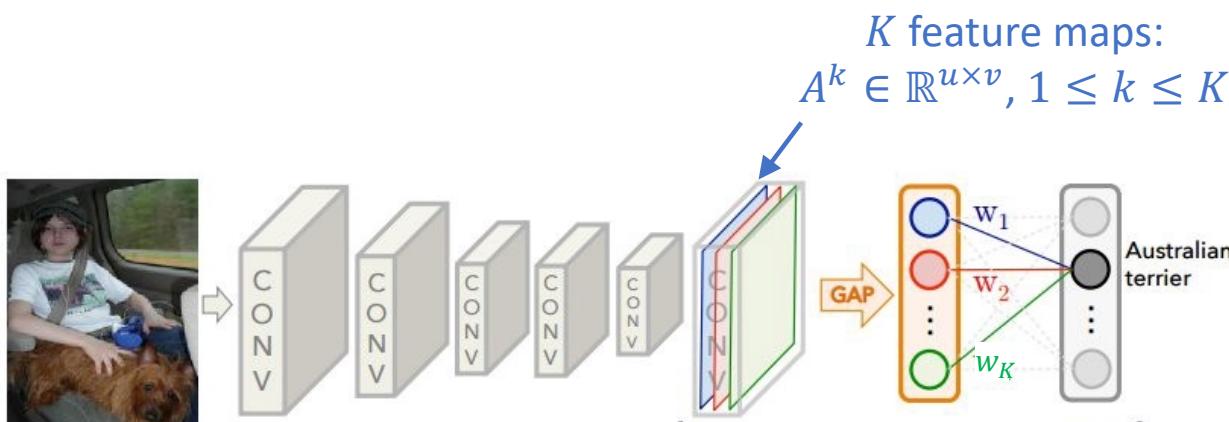
**Main principle of CAM and Grad-CAM:** Assign a significance value (weight) to each activation map, then sum up the weighted activation maps.

fwd conv5\_151 | Back off | Boost: 0/1



**Q:** How to calculate the weights?

# CAM [Zhou-CVPR16]



GAP (Global Average Pooling):

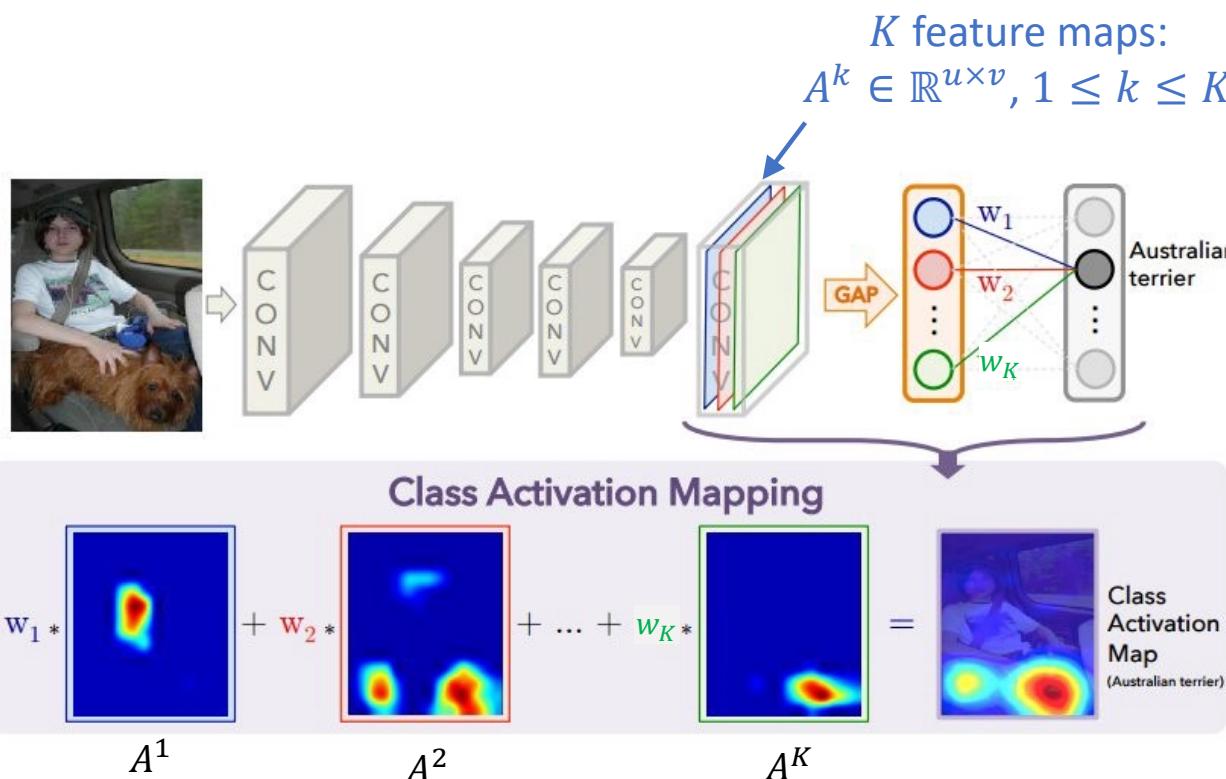
$$\alpha_k = \frac{1}{uv} \sum_{i=1}^u \sum_{j=1}^v A_{i,j}^k$$

Class score for  $c$ -th class:

$$\hat{y}^c = \sum_{k=1}^K w_k^c \alpha_k$$

$w_k^c$  is learnt by  
backpropagation

# CAM [Zhou-CVPR16]



GAP (Global Average Pooling):

$$\alpha_k = \frac{1}{uv} \sum_{i=1}^u \sum_{j=1}^v A_{i,j}^k$$

Class score for  $c$ -th class:

$$\hat{y}^c = \sum_{k=1}^K w_k^c \alpha_k$$

$w_k^c$  is learnt by backpropagation

Class Activation Map for  $c$ -th class:

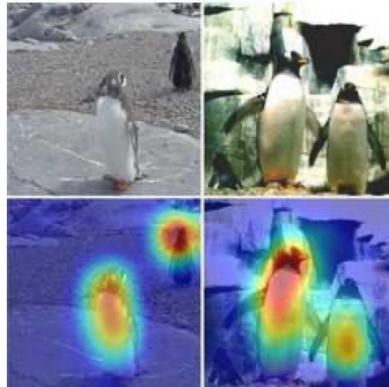
$$L_{\text{CAM}}^c = \sum_{k=1}^K w_k^c A^k$$

# CAM [Zhou-CVPR16]

Mushroom



Penguin



Teapot



Caltech256

Cleaning the floor



Cooking



Fixing a car



Stanford Action40

Zhou et. Al. "Learning deep features for discriminative localization" CVPR 2016.

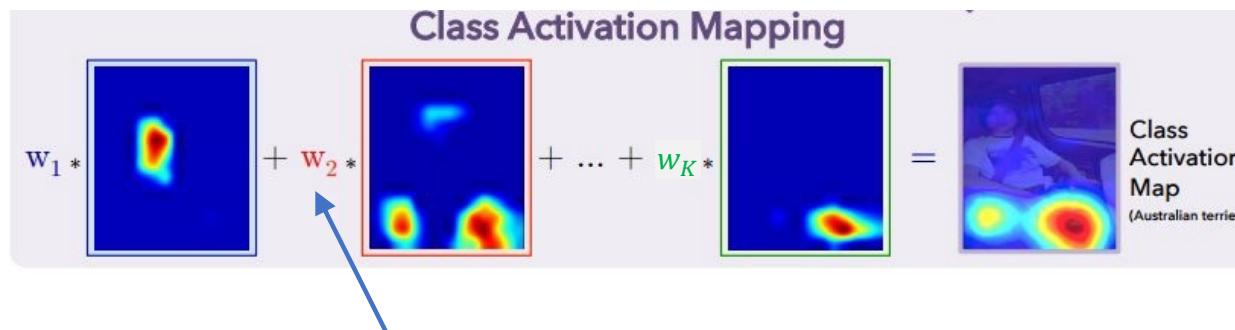
# Limitations of CAM

- Need to have a GAP layer in the model architecture
  - Only applicable to this particular kind of CNN architecture (conv layers → GAP → FC layer (with Softmax)), leading to inferior accuracies.
- Can only visualize the final layer heatmap

**Solution:** A generalization of CAM: **Grad-CAM**  
Compatible with any kind of CNN architecture

# Grad-CAM [Selvaraju-ICCV17]

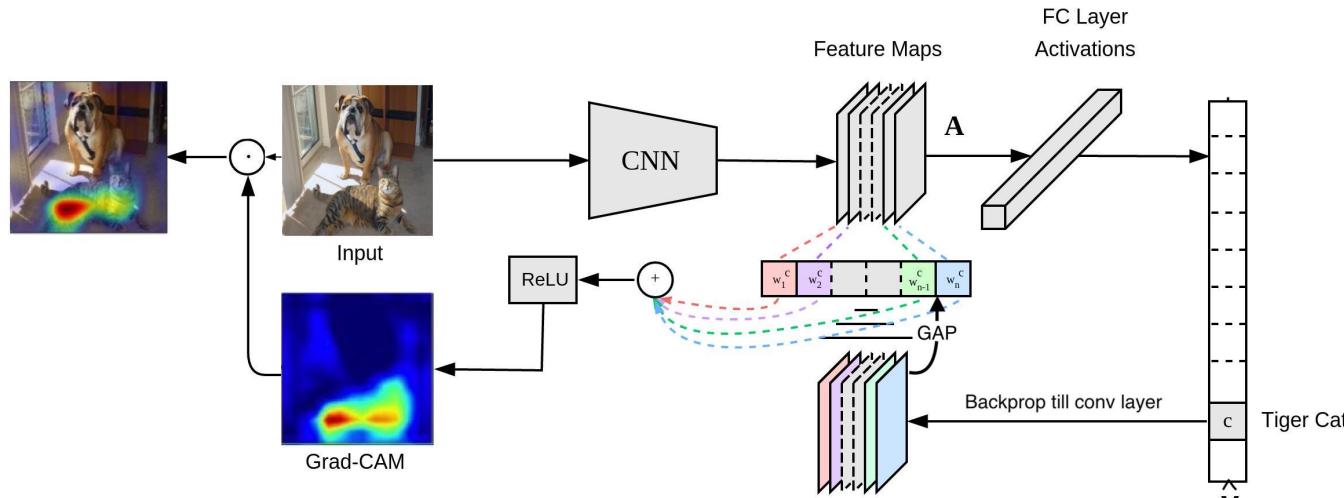
- **Intuition:** Gradients indicates its importance for making a decision by a model.



Q: How to calculate  $w_k$ ?

A: Using average of gradients w.r.t. a certain feature map.

# Grad-CAM [Selvaraju-ICCV17]



$K$  gradient maps:  
 $\frac{\partial \hat{y}}{\partial A^k} \in \mathbb{R}^{u \times v}, 1 \leq k \leq K$

Gradient via backpropagation

$$\frac{\partial \hat{y}^c}{\partial A_{i,j}^k}$$

GAP (Global Average Pooling):

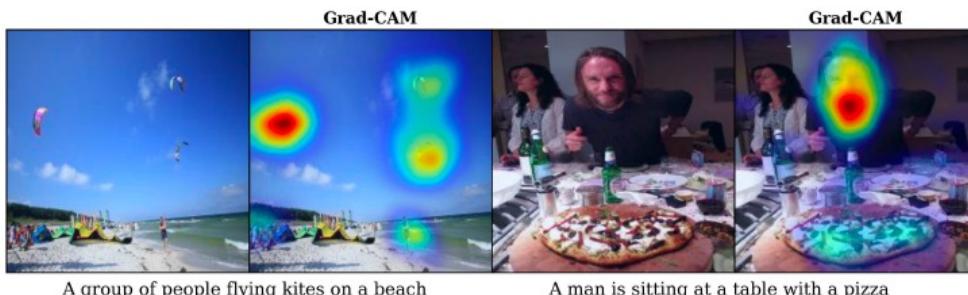
$$w_k^c = \frac{1}{uv} \sum_{i=1}^u \sum_{j=1}^v \frac{\partial \hat{y}^c}{\partial A_{i,j}^k}$$

Grad-CAM for  $c$ -th class:

$$L_{\text{Grad-CAM}}^c = \text{ReLU} \left( \sum_{k=1}^K w_k^c A^k \right)$$

# Beyond classification

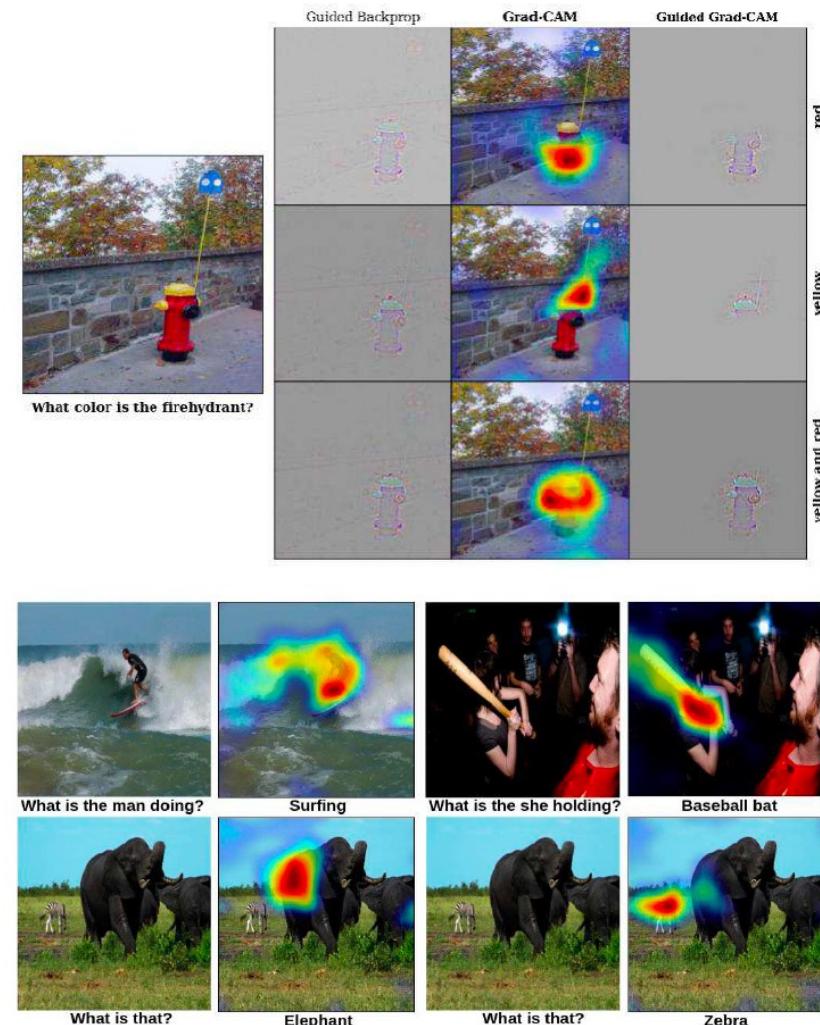
- Grad-CAM can be also used for other CNN-based models.



## Image captioning

More on Grad-CAM Demo: <http://gradcam.cloudcv.org>

## VQA (Visual Question Answering)



## Visualizing Filters

Display  
Filters  
directly

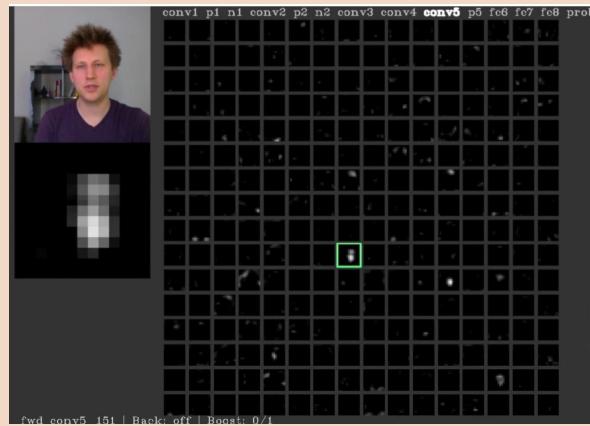
Find patches in a  
dataset



Synthesize patches

Feature Inverse  
Texture synthesis  
Neural Style Transfer

## Visualizing Activation Maps



## Visualizing Heatmaps

CAM, Grad-CAM, ...



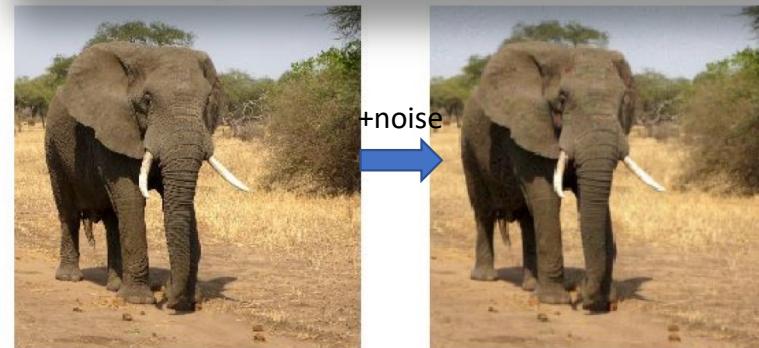
# Benefits of visualizing CNN

- Helps to understand why a model they build works or does not work.
- Helps to debug models and improve training and testing strategies.
- Helps to do architectural changes or build new models
- Helps to identify dataset bias

### Style transfer

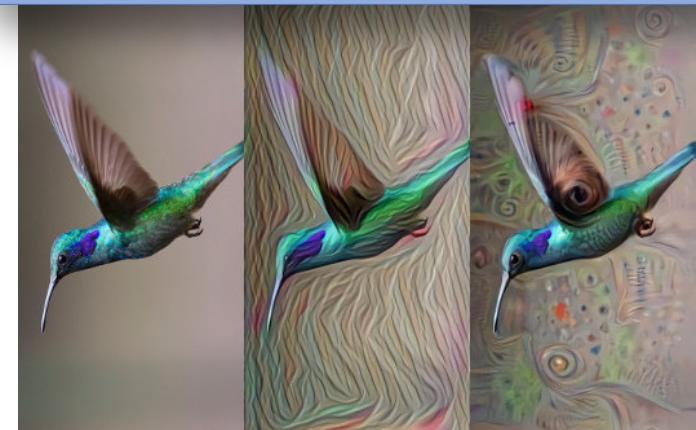


### Fooling images



Prediction: Koala

### Deep Dream



<https://deeplearning.net/deepdream/>

More than just visualization ...

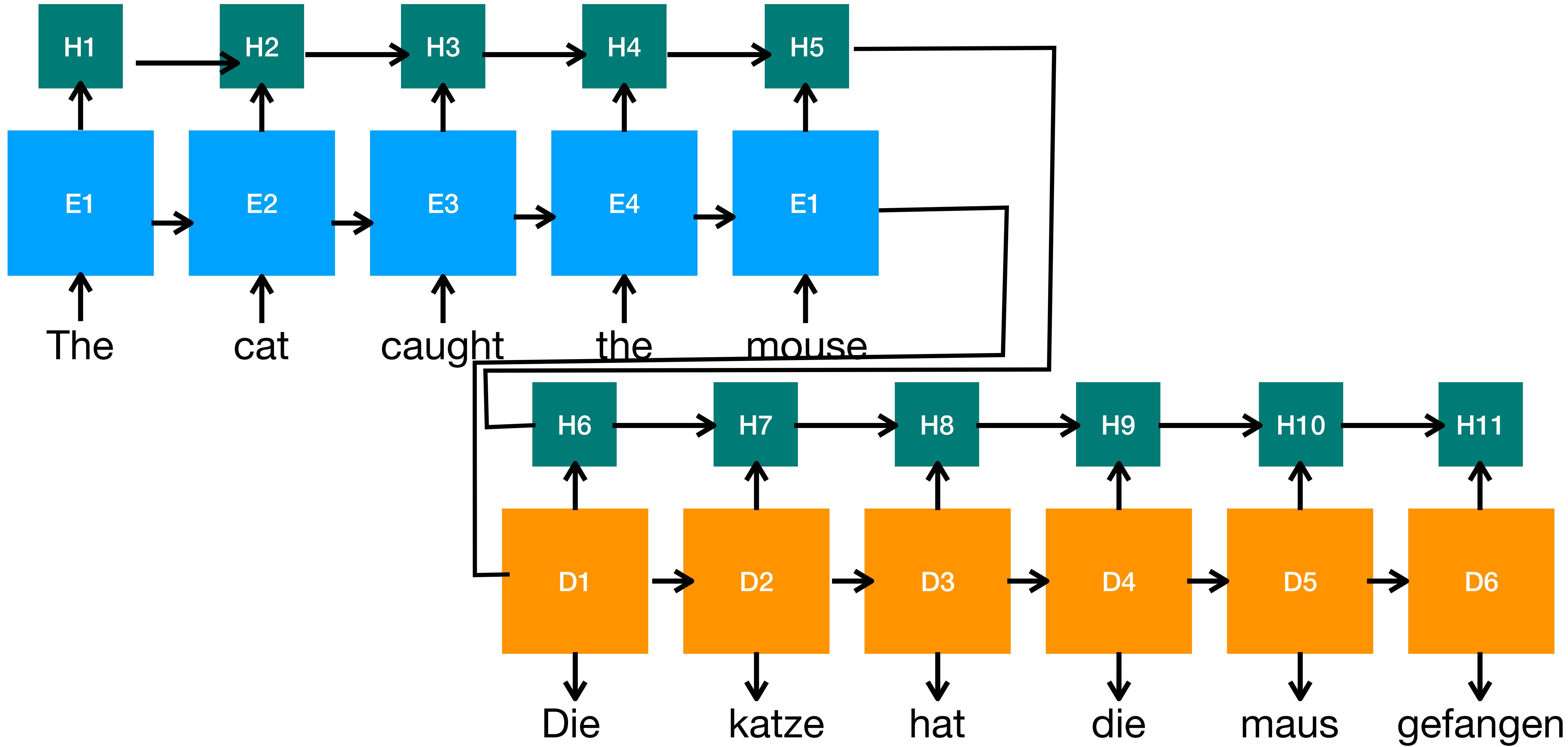
# References

- Blog by Rachel Draelos. “CNN Heat maps: class activation mapping (GAM)”. <https://glassboxmedicine.com/2019/06/11/cnn-heat-maps-class-activation-mapping-cam/>
- Blog by Anil Matcha. “Class activation maps: visualizing neural network decision-making”, 2019. <https://heartbeat.comet.ml/class-activation-maps-visualizing-neural-network-decision-making-92efa5af9a33>
- Blog by Shubham Panchal. “Grab-CAM: A camera for your model’s decision”, 2021. <https://towardsdatascience.com/grad-cam-camera-for-your-models-decision-1ef69aae8fe7>

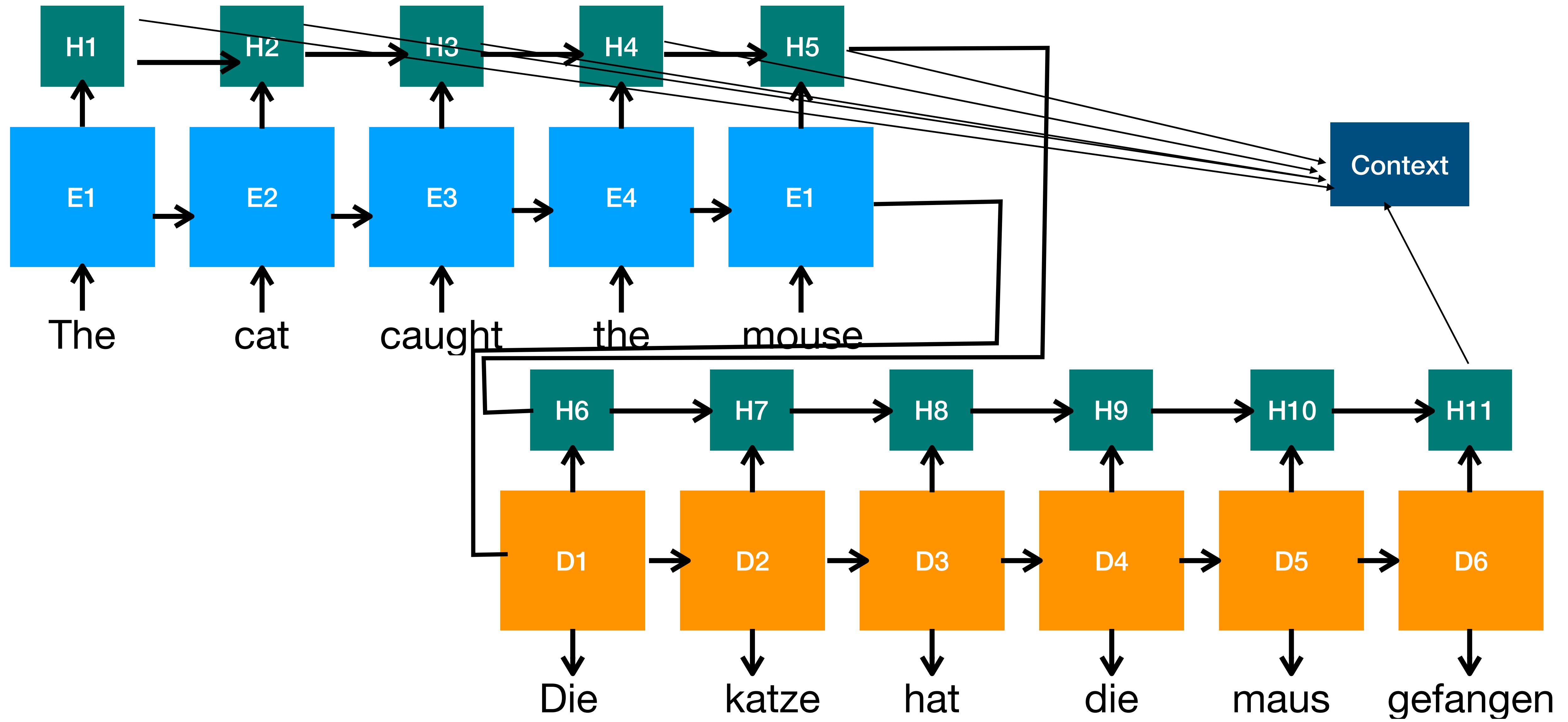
# Understanding Transformers

Vinay P. Namboodiri

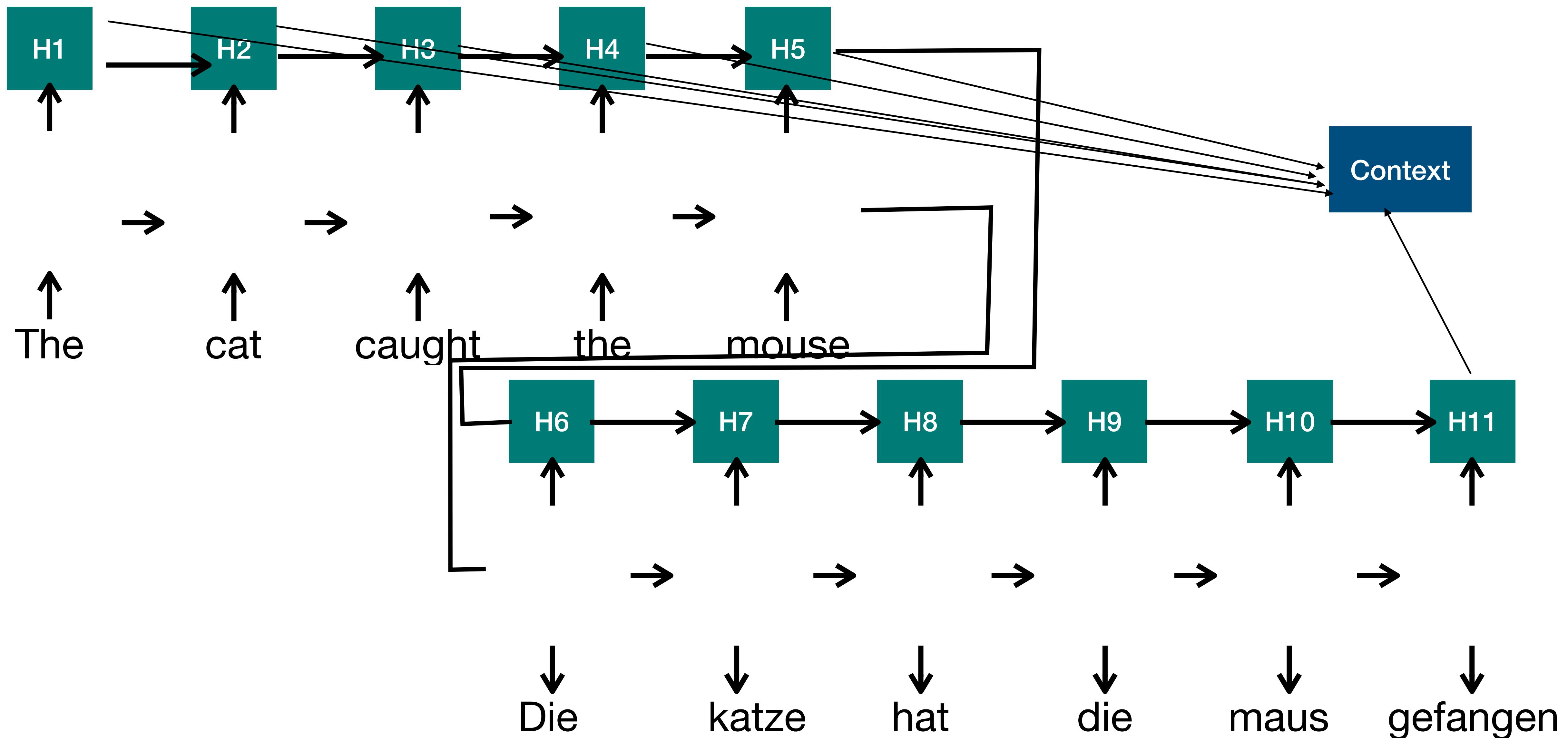
# Machine Translation using LSTM



# LSTM with attention



# Main idea of transformers - Attention is all you need



# Why Transformers?

# Attention Is All You Need

**Ashish Vaswani\***

Google Brain

avaswani@google.com

**Noam Shazeer\***

Google Brain

noam@google.com

**Niki Parmar\***

Google Research

nikip@google.com

**Jakob Uszkoreit\***

Google Research

usz@google.com

**Llion Jones\***

Google Research

llion@google.com

**Aidan N. Gomez\*** †

University of Toronto

aidan@cs.toronto.edu

**Łukasz Kaiser\***

Google Brain

lukaszkaiser@google.com

**Illia Polosukhin\*** ‡

illia.polosukhin@gmail.com

## Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-

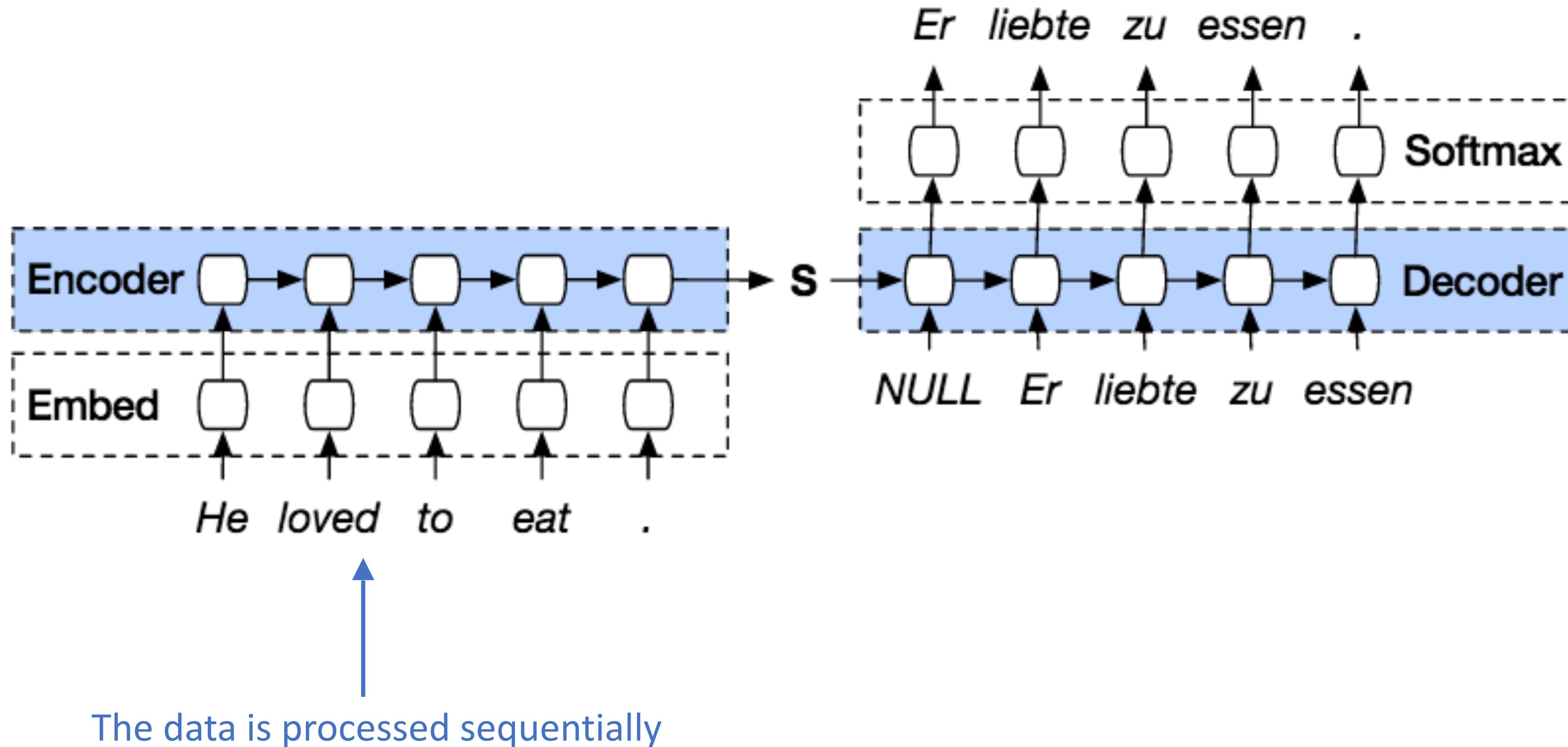
## Attention is all you need

[A Vaswani](#), [N Shazeer](#), [N Parmar](#)... - Advances in neural ... , 2017 - proceedings.neurips.cc

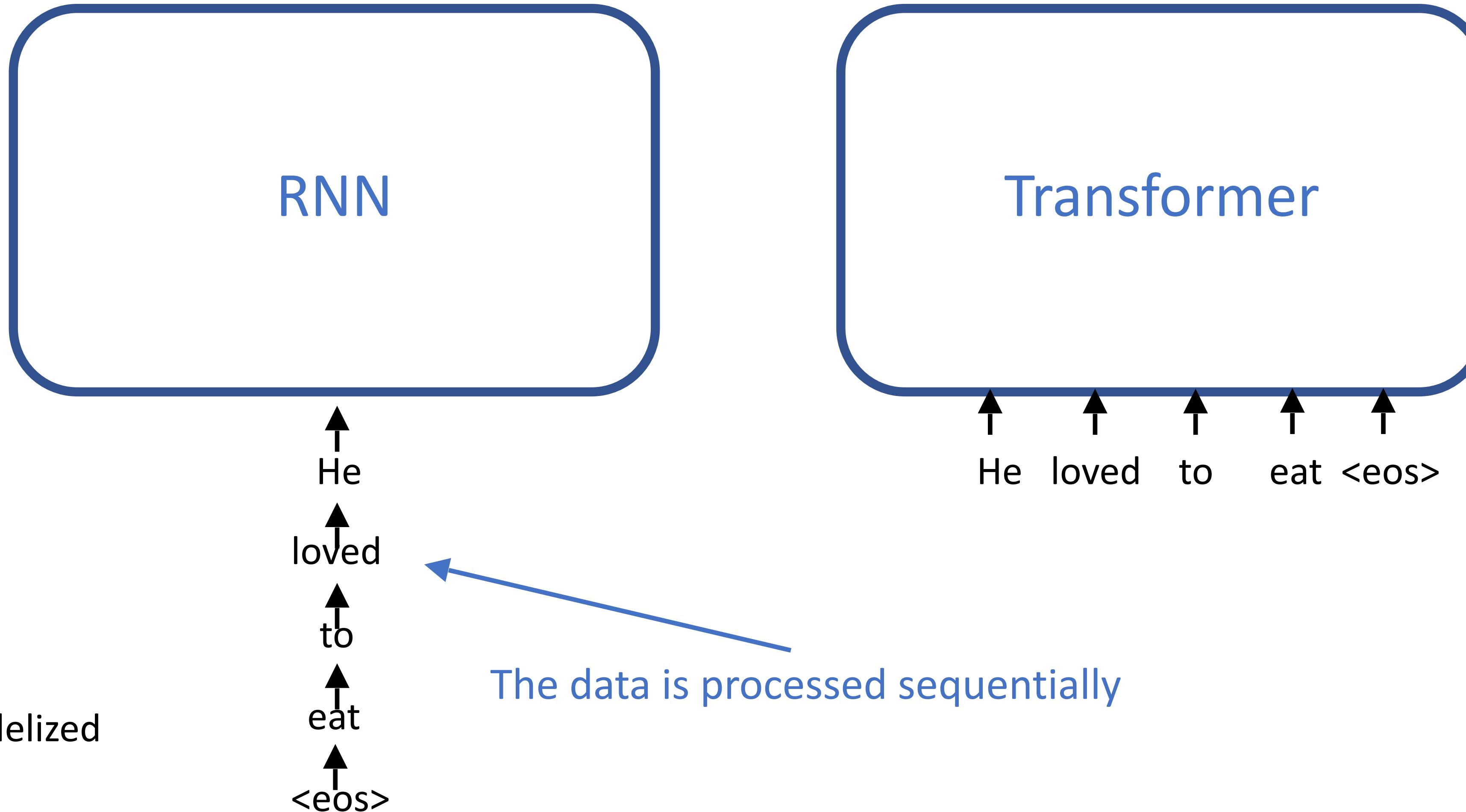
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder and decoder configuration. The best ...

☆ Save ⚏ Cite Cited by 116698 Related articles ➞

# Neural Machine Translation with RNNs (LSTM, GRU)

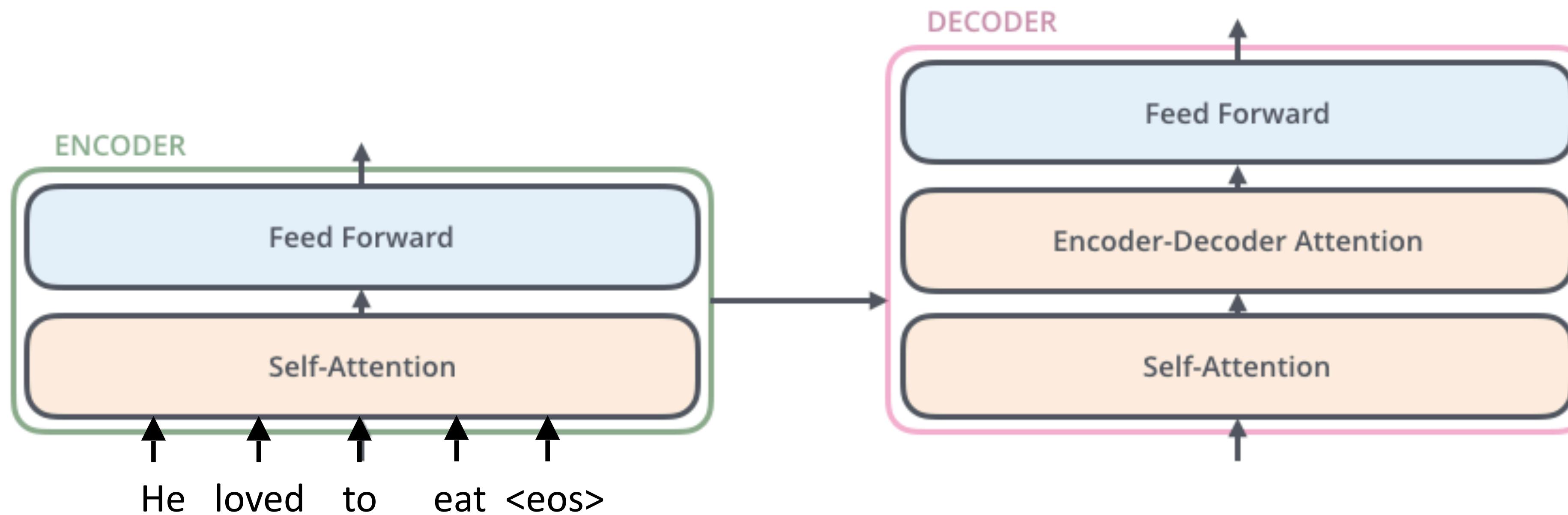


# Limitation of RNNs (LSTM, GRU)



- Can't be parallelized
- Slow to train
- Transfer learning can hardly be used, i.e., need specific labelled dataset to your task

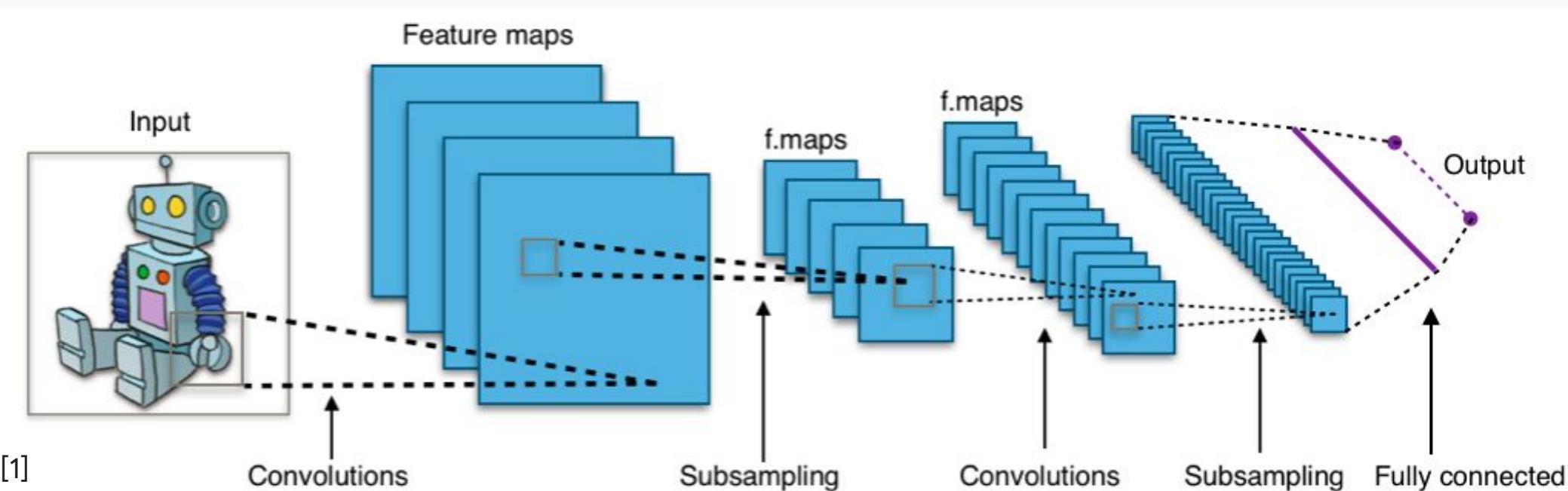
# Neural Machine Translation with Transformer



- Discard the recurrent structure
- Solely based on attention mechanism
- Parallelizable, train faster
- Can be pre-trained

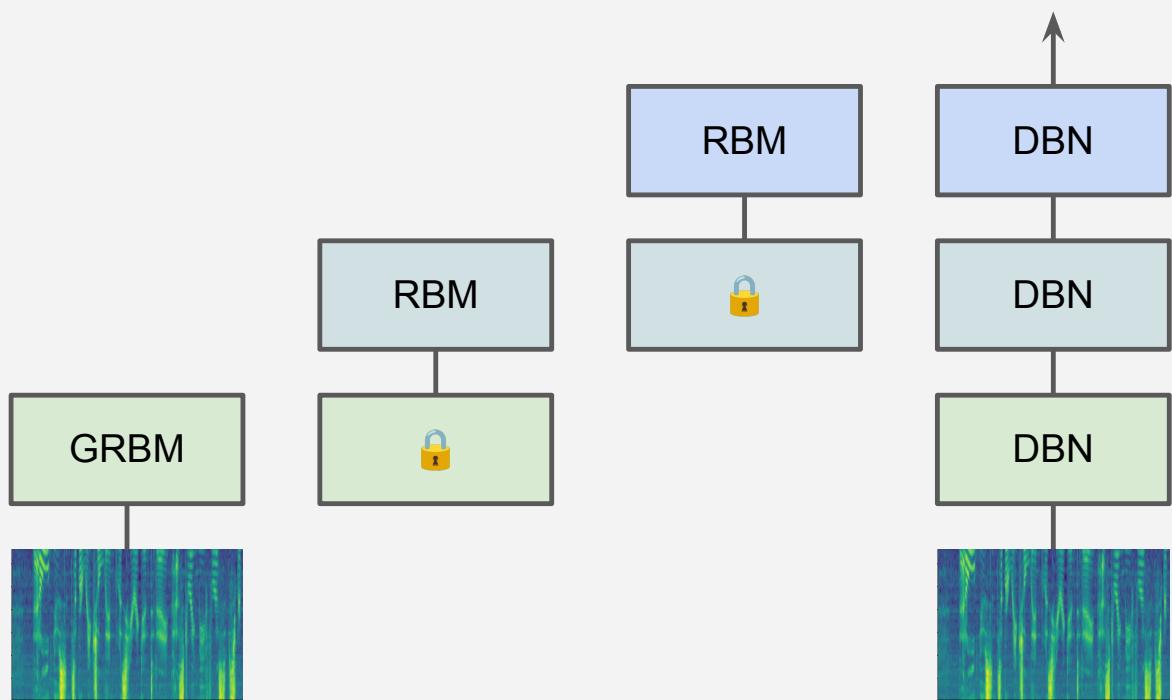
# Computer Vision

## Convolutional NNs (+ResNets)



# Speech

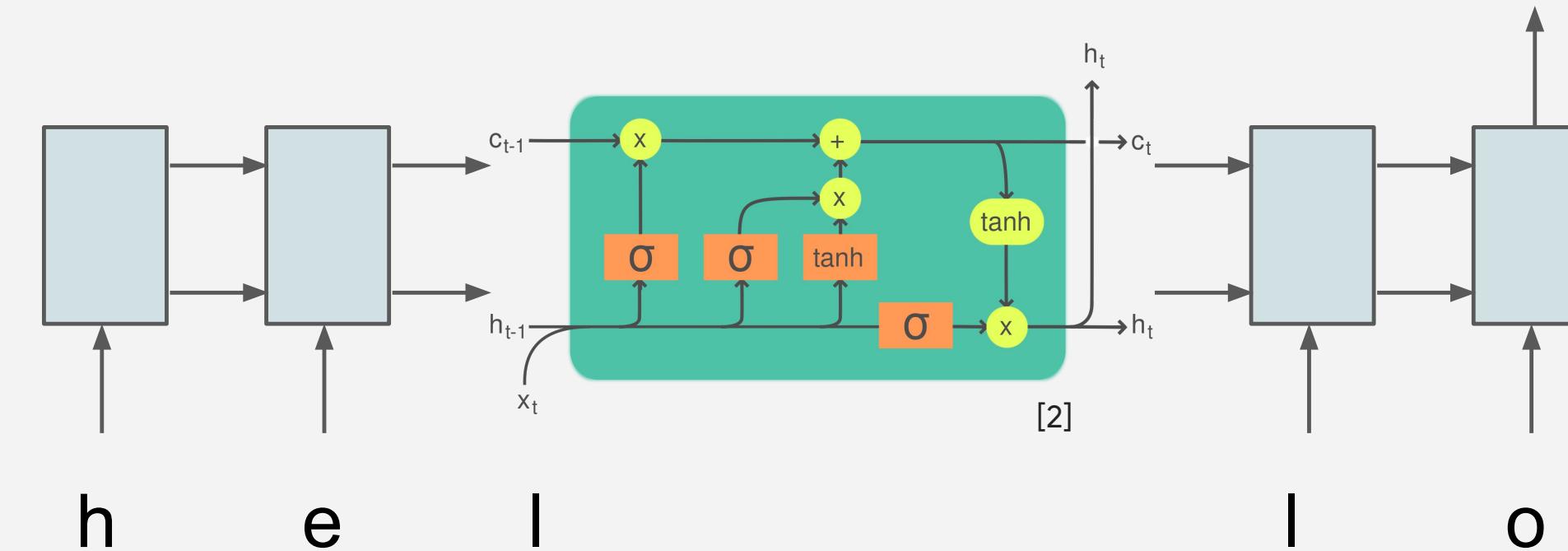
## Deep Belief Nets (+non-DL)



[1] CNN image CC-BY-SA by Aphex34 for Wikipedia [https://commons.wikimedia.org/wiki/File:Typical\\_cnn.png](https://commons.wikimedia.org/wiki/File:Typical_cnn.png)  
[2] RNN image CC-BY-SA by GChe for Wikipedia [https://commons.wikimedia.org/wiki/File:The\\_LSTM\\_Cell.svg](https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg)

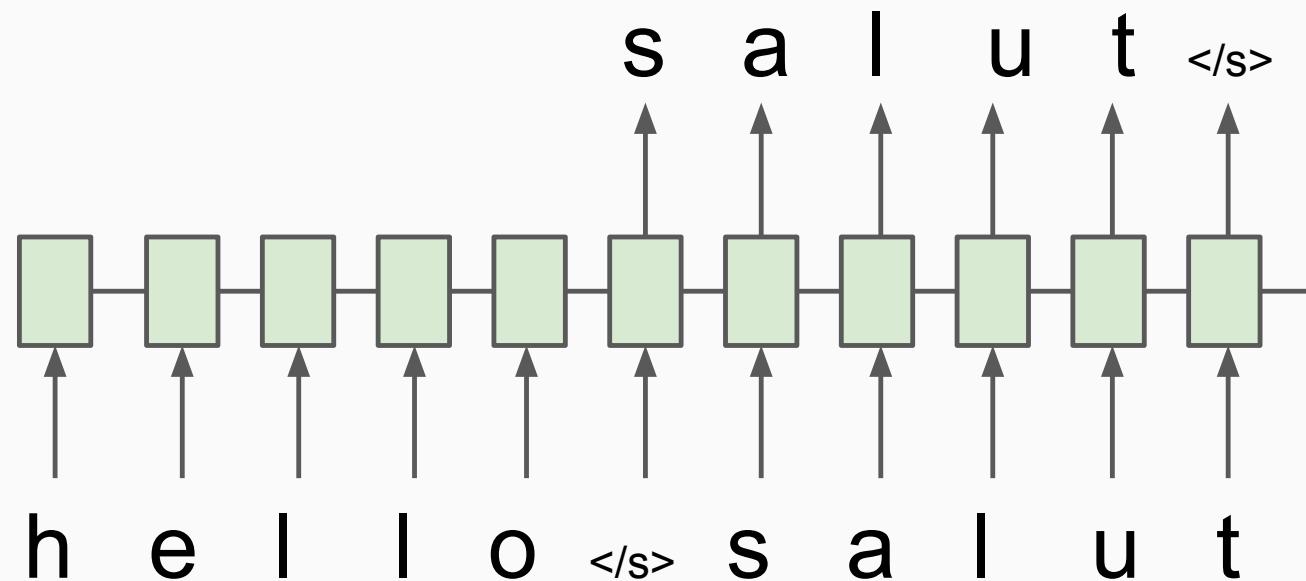
# Natural Lang. Proc.

## Recurrent NNs (+LSTMs)



# Translation

## Seq2Seq



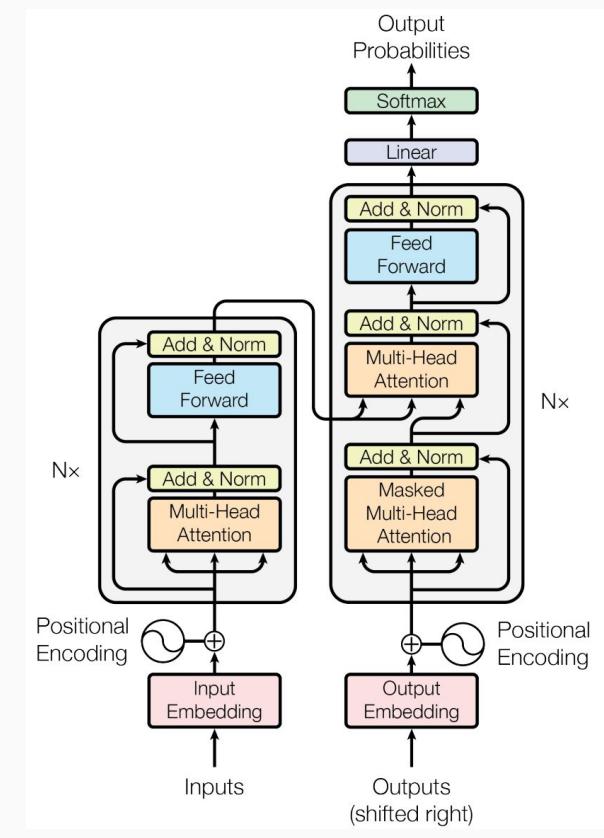
# RL

## BC/GAIL

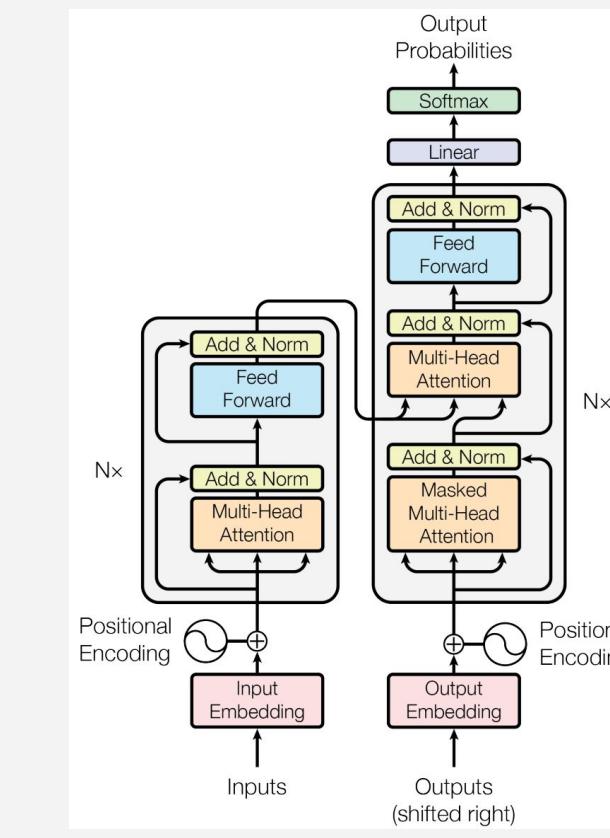
### Algorithm 1 Generative adversarial imitation learning

- 1: **Input:** Expert trajectories  $\tau_E \sim \pi_E$ , initial policy and discriminator parameters  $\theta_0, w_0$
  - 2: **for**  $i = 0, 1, 2, \dots$  **do**
  - 3:     Sample trajectories  $\tau_i \sim \pi_{\theta_i}$
  - 4:     Update the discriminator parameters from  $w_i$  to  $w_{i+1}$  with the gradient
- $$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s, a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s, a))] \quad (17)$$
- 5:     Take a policy step from  $\theta_i$  to  $\theta_{i+1}$ , using the TRPO rule with cost function  $\log(D_{w_{i+1}}(s, a))$ . Specifically, take a KL-constrained natural gradient step with
- $$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s)Q(s, a)] - \lambda \nabla_\theta H(\pi_\theta), \quad (18)$$
- $$\text{where } Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s, a)) | s_0 = \bar{s}, a_0 = \bar{a}]$$
- 6: **end for**

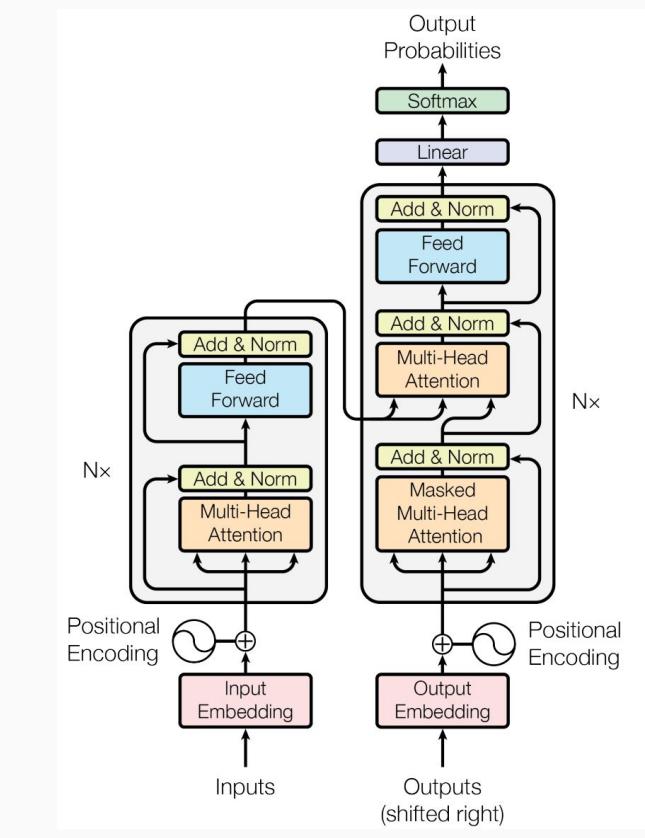
# Computer Vision



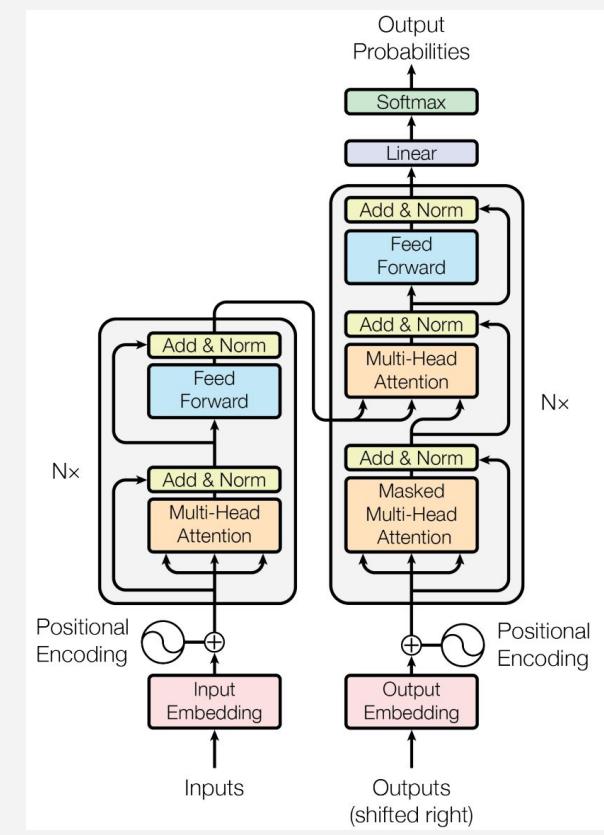
# Natural Lang. Proc.



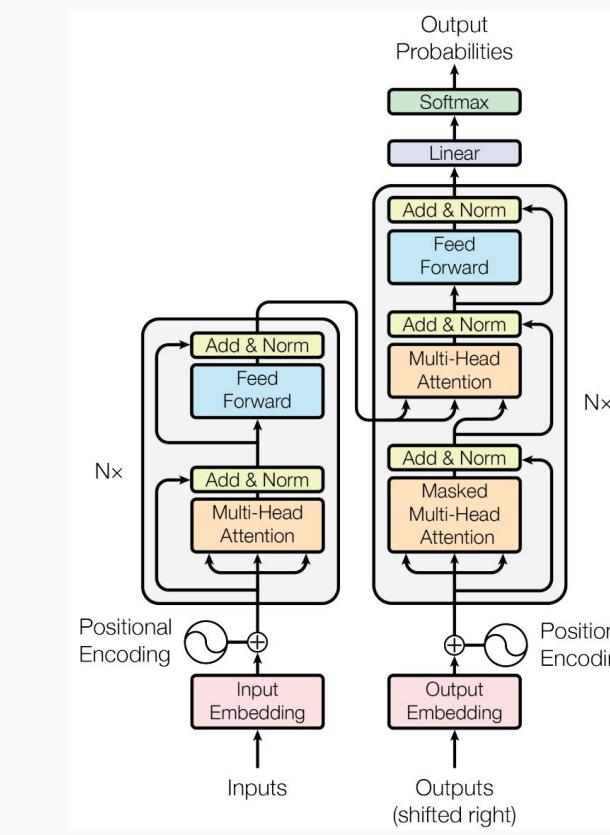
# Reinf. Learning



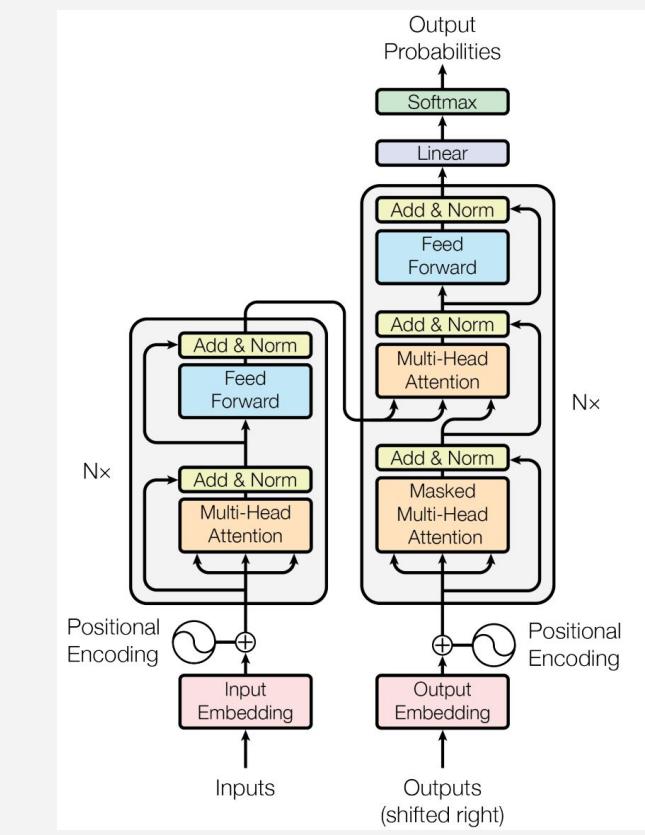
# Speech



# Translation



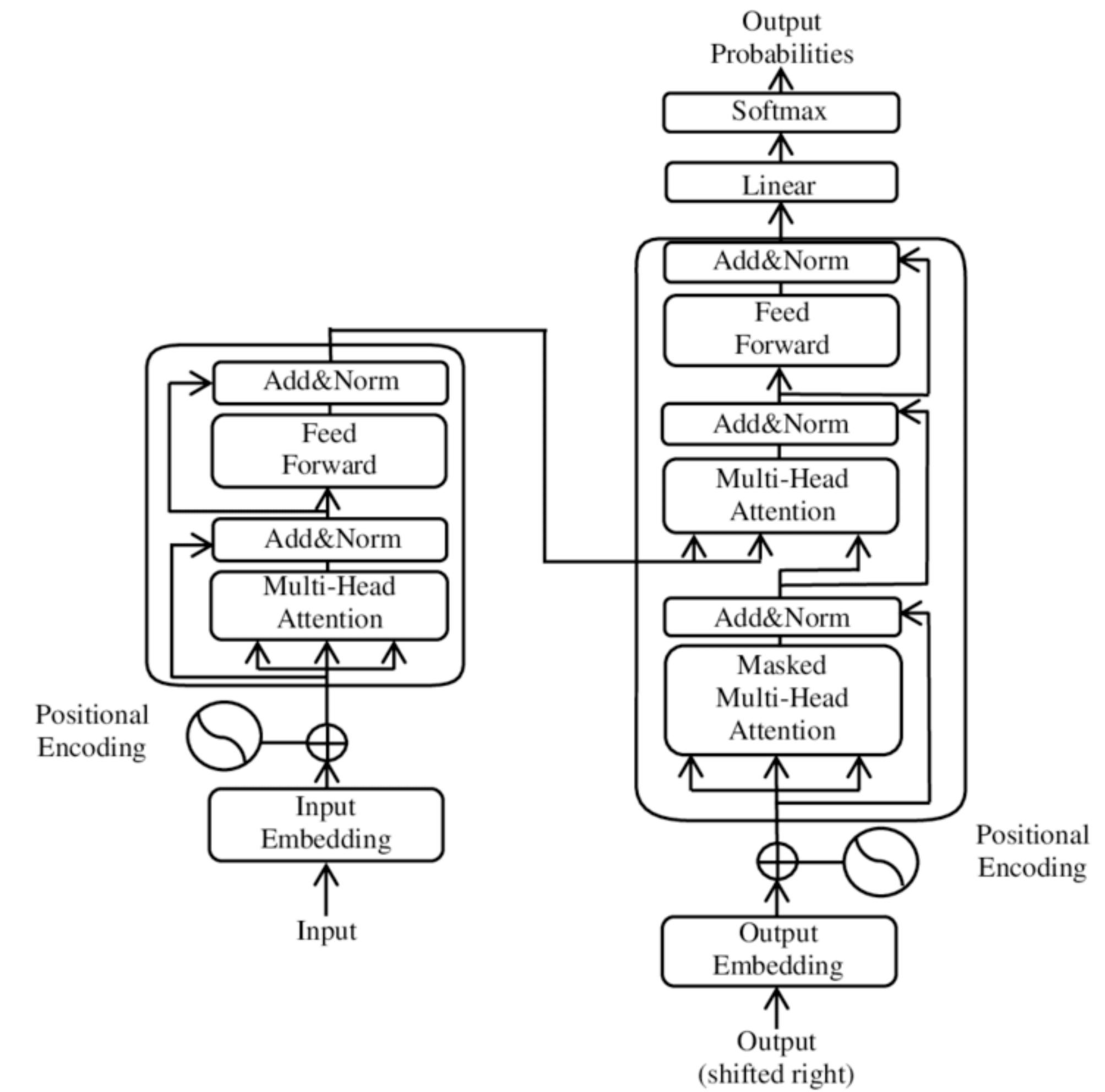
# Graphs/Science



former image source: "Attention Is All You Need" paper

# What is the transformer?

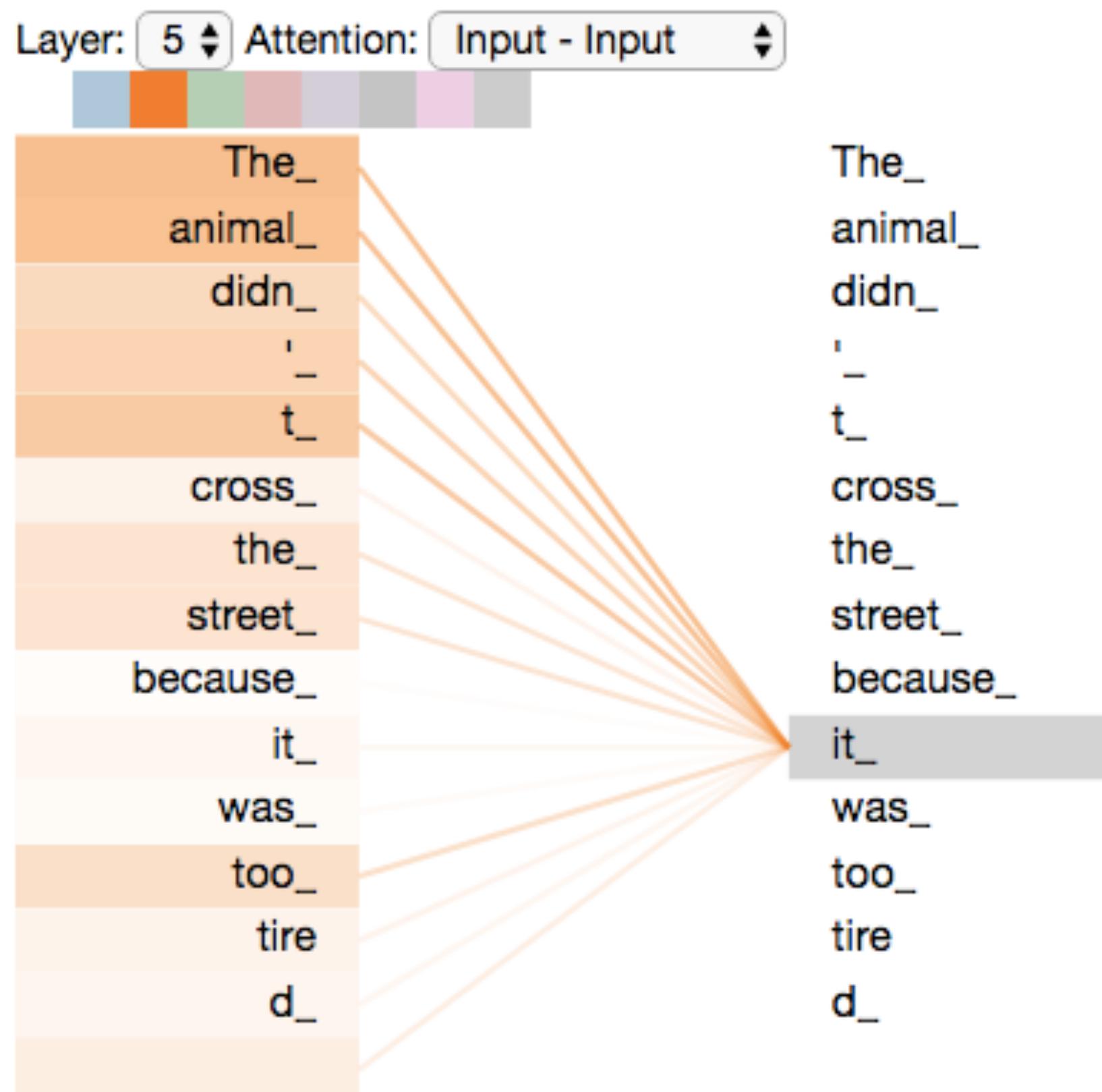
- A ‘simple’ architecture with few operation
- Embedding
- Positional Encoding
- Layer-normalization
- Multi-headed attention
- MLP
- Residual Layer



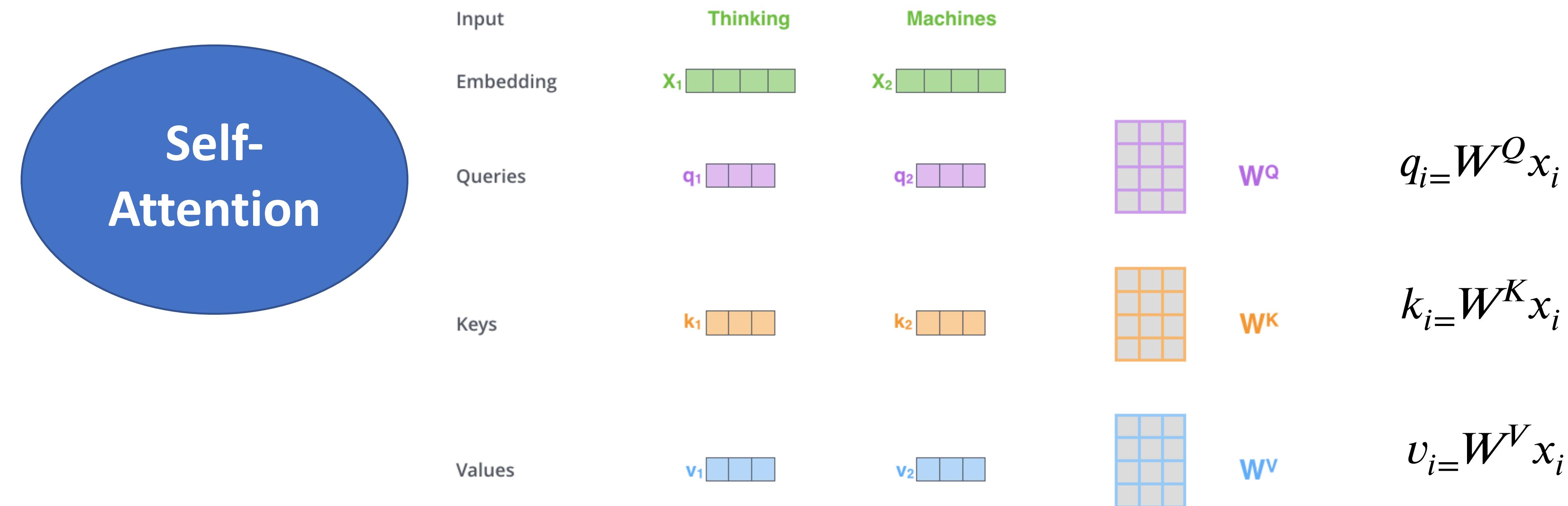
# The Attention Layer

# Self-Attention

- **Intuition:** to bake the “understanding” of other relevant words into the one we are currently processing.

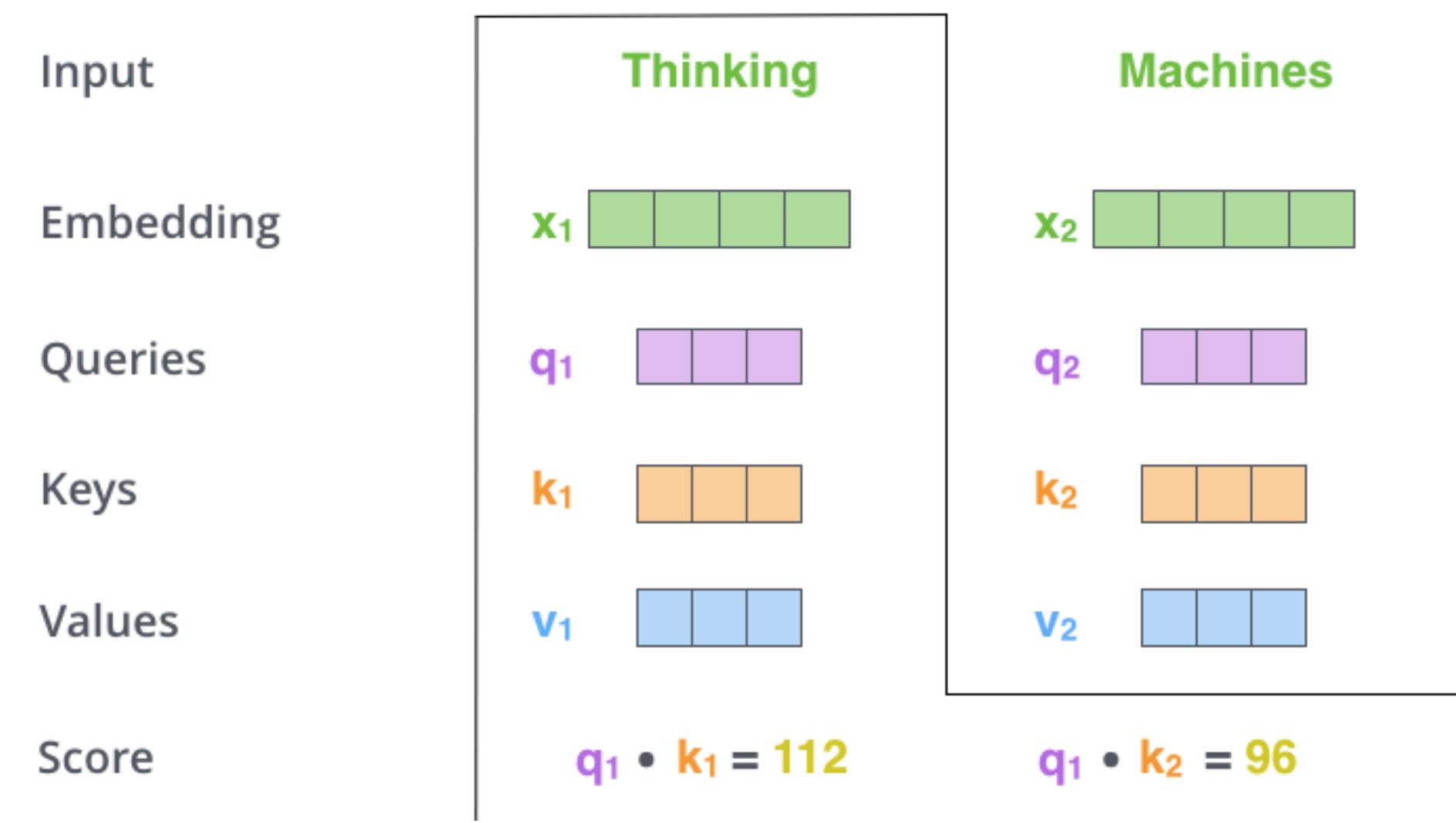


# STEP 1: create three vectors by multiplying the embedding with three weight matrices (will be learnt during training)



# Self-Attention

**STEP 2:** Calculate the **attention score** which determines how much focus to place on other words as we encode one word.



$$\text{score}(x_i, x_j) = q_i^T \bullet k_j$$

# Self-Attention

**STEP 3:** Convert the score into a number in  $[0, 1]$  using softmax function.

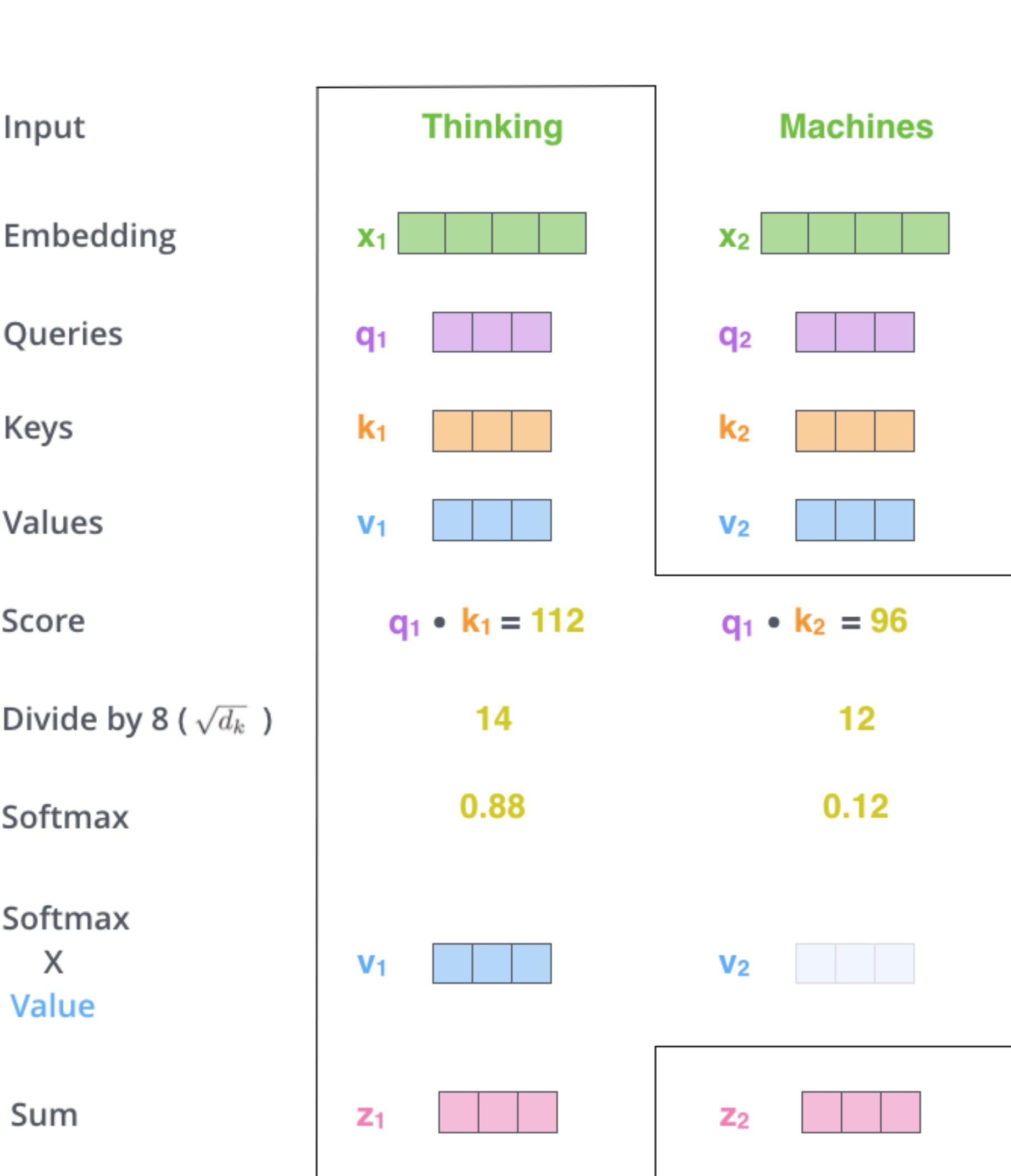
Input	<b>Thinking</b> $x_1$		<b>Machines</b> $x_2$	
Embedding				
Queries	$q_1$		$q_2$	
Keys	$k_1$		$k_2$	
Values	$v_1$		$v_2$	
Score	$q_1 \cdot k_1 = 112$		$q_1 \cdot k_2 = 96$	
Divide by 8 ( $\sqrt{d_k}$ )	14		12	
Softmax	0.88		0.12	

$$\alpha_{ij} = \frac{\exp(\text{score}(x_i, x_j) / \sqrt{d_k})}{\sum_{j'} \exp(\text{score}(x_i, x_{j'}) / \sqrt{d_k})}$$

$d_k$  : the dimension of the three vectors ( $q_i$ ,  $k_i$ ,  $v_i$ ), is 64 in the paper.

# Self-Attention

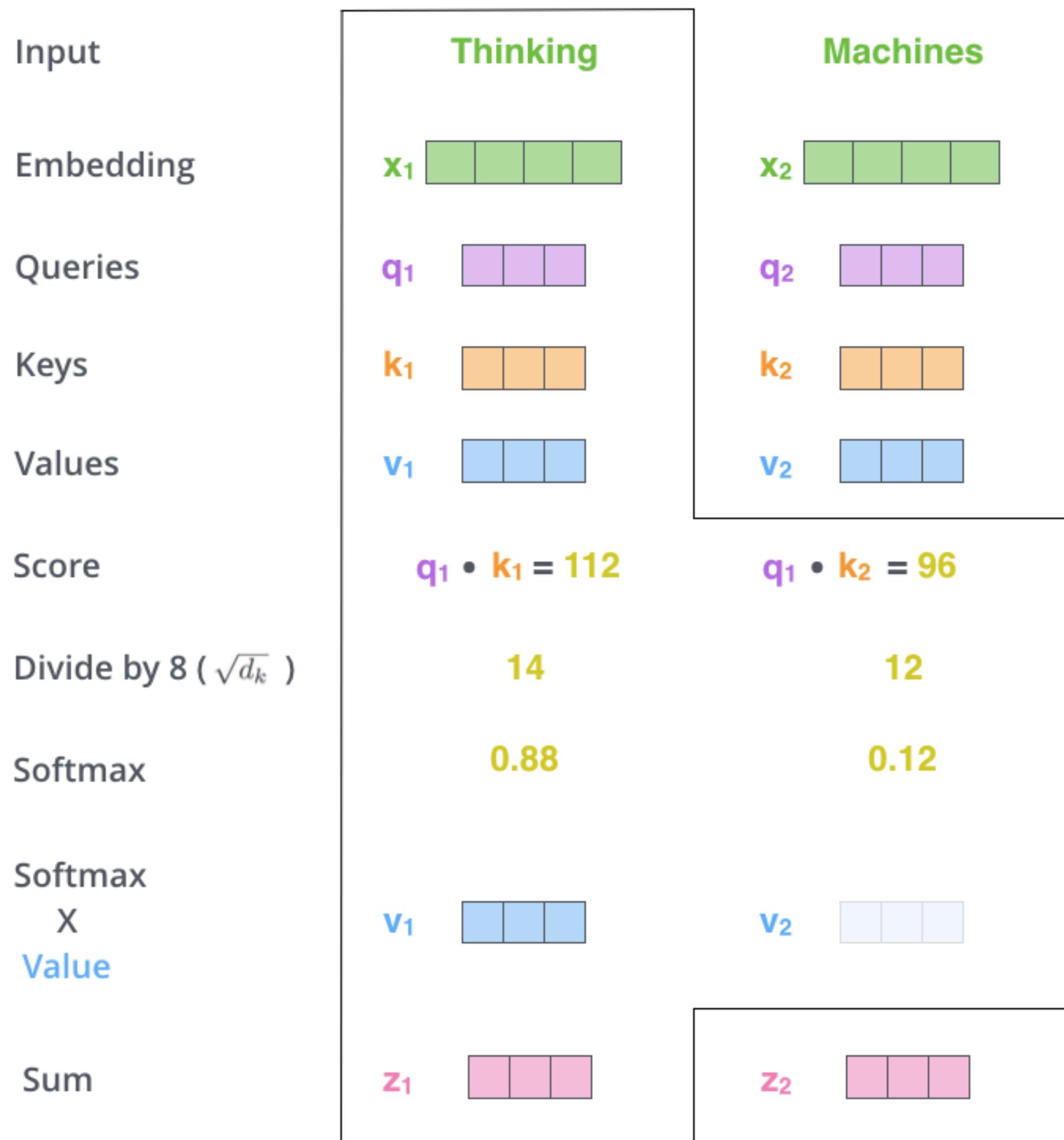
**STEP 4:** sum up the weighted value vectors.



$$z_i = \sum_j \alpha_{ij} v_j$$

# Self-Attention

**STEP 4:** sum up the weighted value vectors.



$$z_i = \sum_j \alpha_{ij} v_j$$



creature

$$\downarrow \vec{E}_4$$

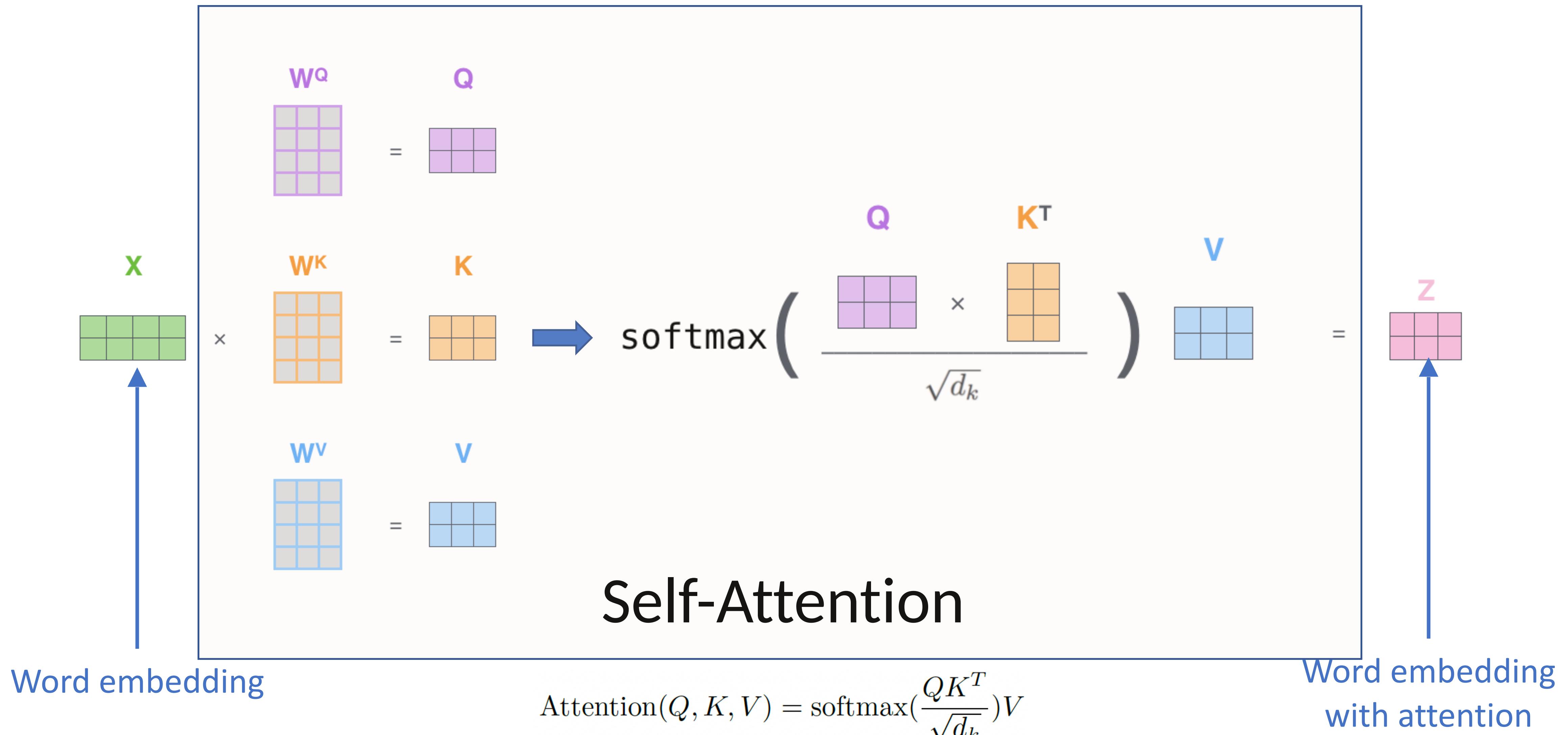
$$+ \Delta \vec{E}_4$$

$$|| \vec{E}'_4$$



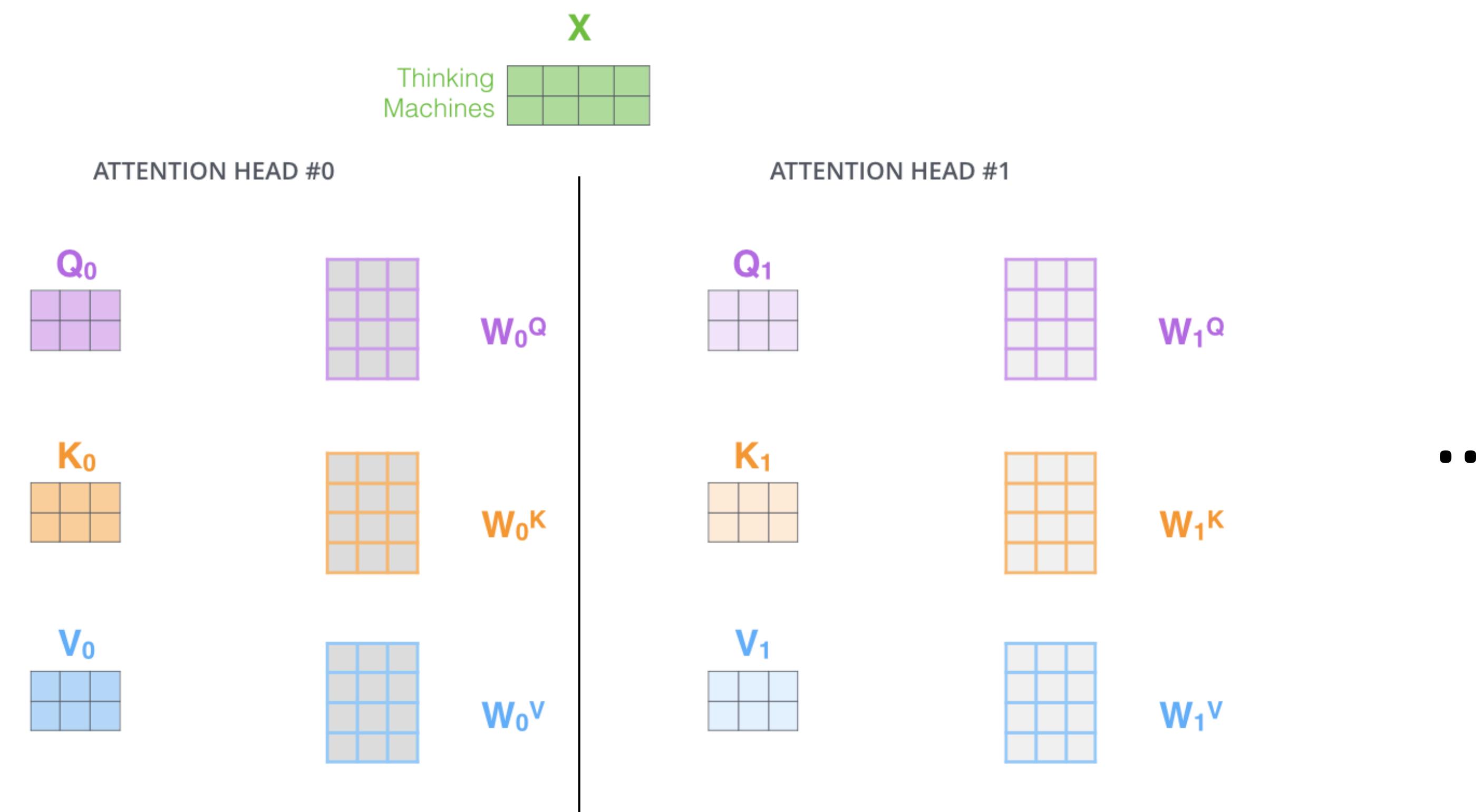
Attending to “fluffy blue”

Image by 3Blue1Brown



# Multi-head Attention

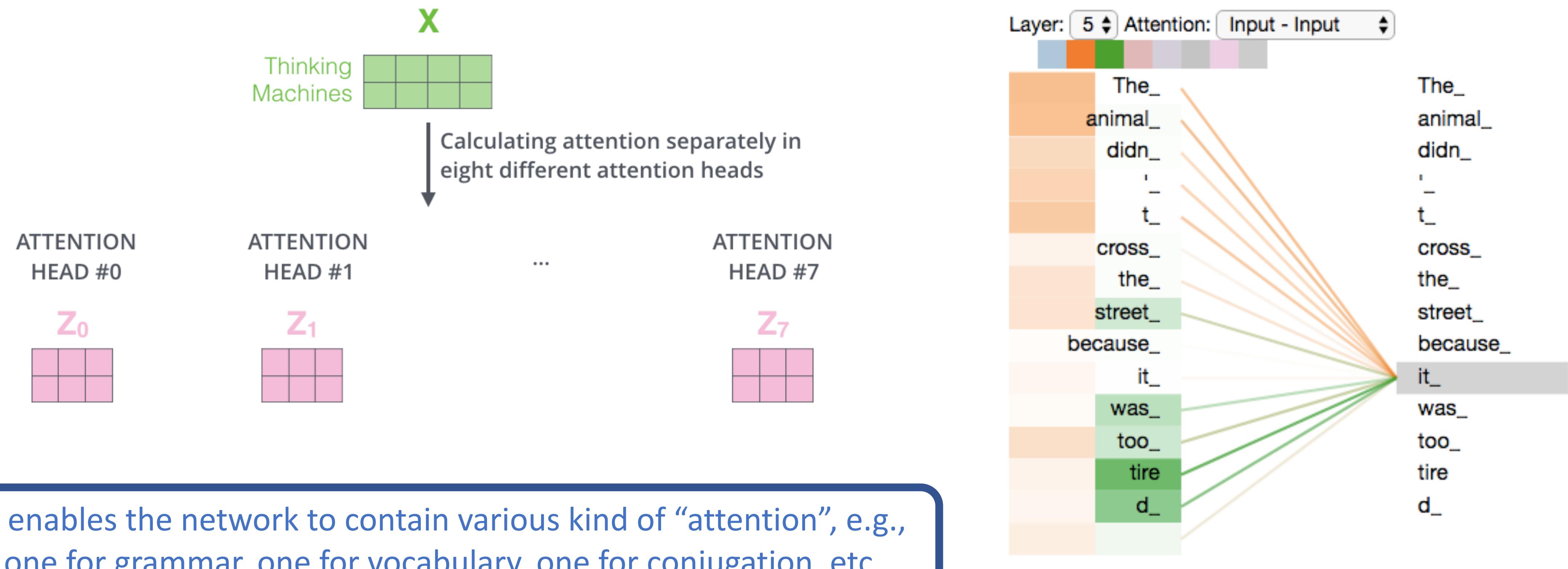
- **Multi-head Attention:** multiple self-attentions, running in parallel



Multiple sets of Query/Key/Value weight matrices are randomly initialized and then trained.

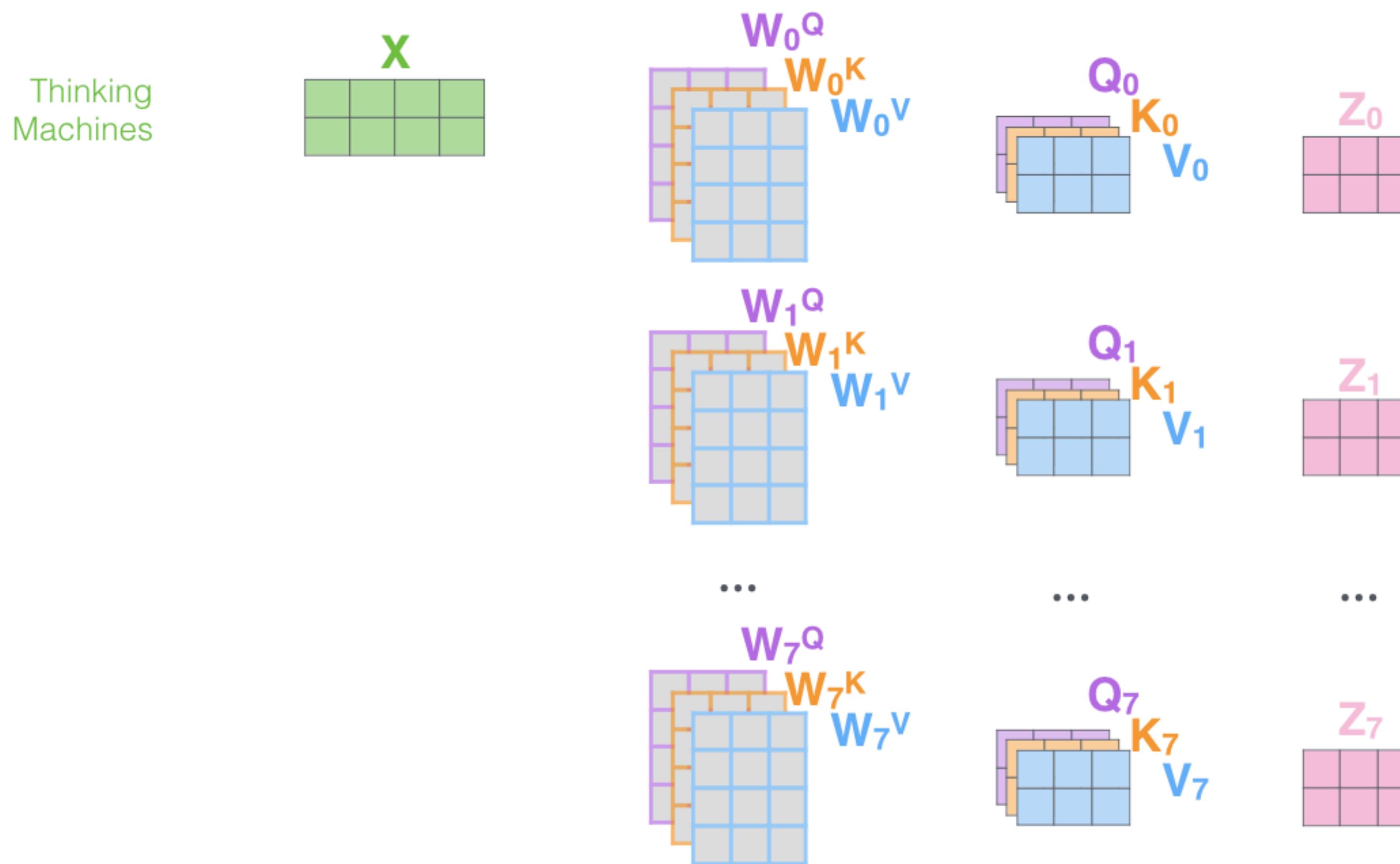
# Multi-head Attention

- **Multi-head Attention:** multiple self-attentions, running in parallel



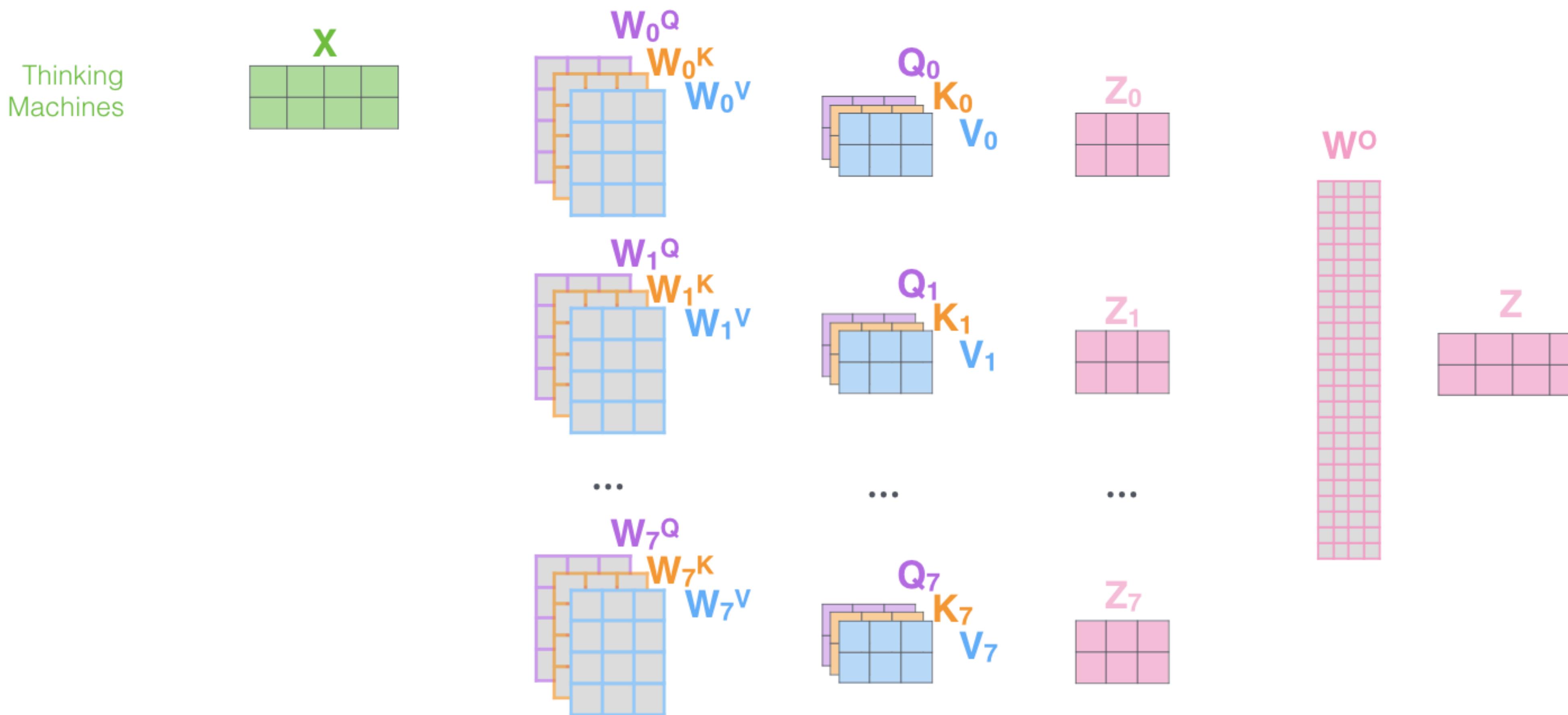
# Put all together

- 1) This is our input sentence\*
- 2) We embed each word\*
- 3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices
- 4) Calculate attention using the resulting  $Q/K/V$  matrices



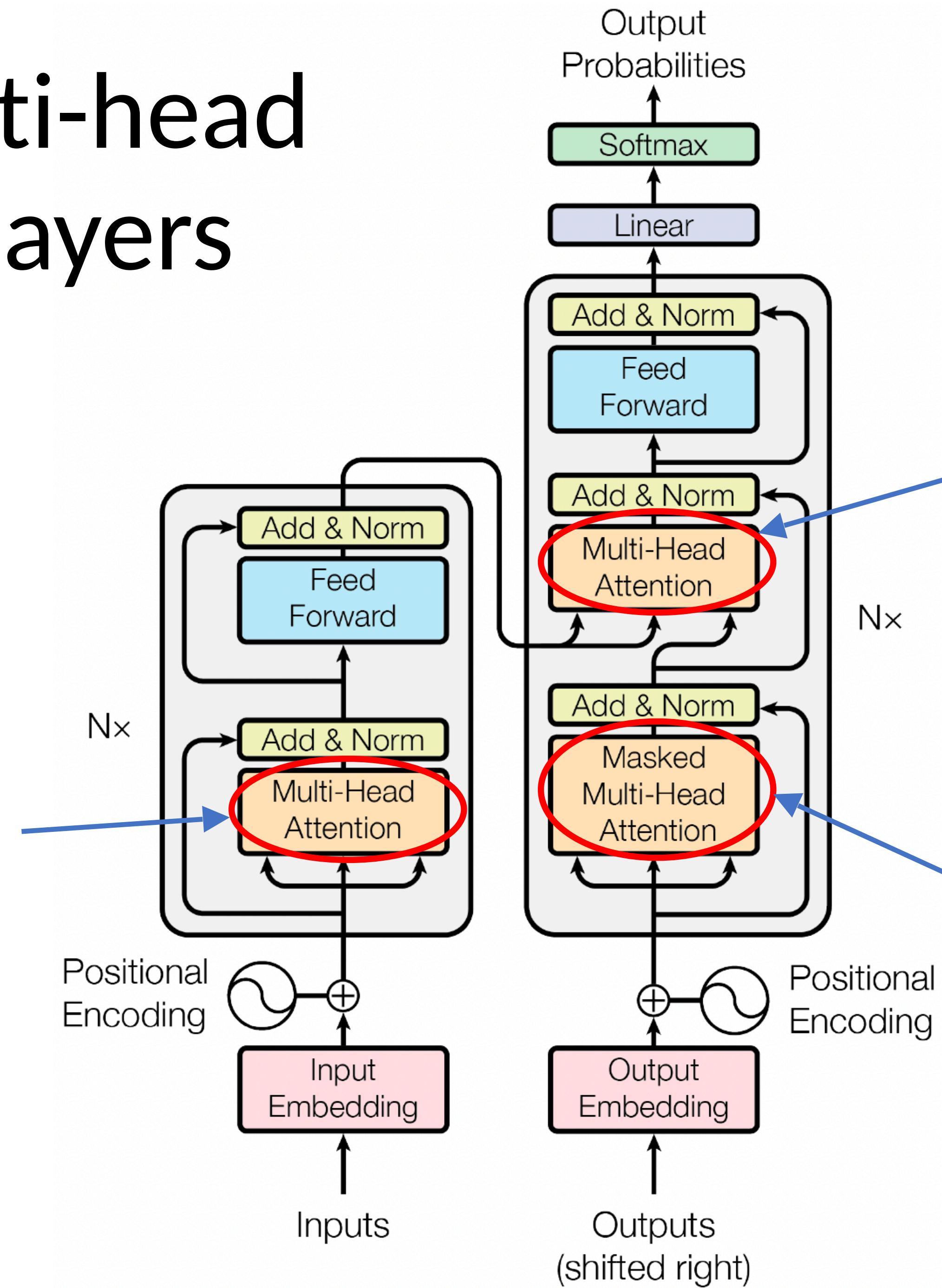
# Put all together

- 1) This is our input sentence\*
- 2) We embed each word\*
- 3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices
- 4) Calculate attention using the resulting  $Q/K/V$  matrices
- 5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer



# Three multi-head attention layers

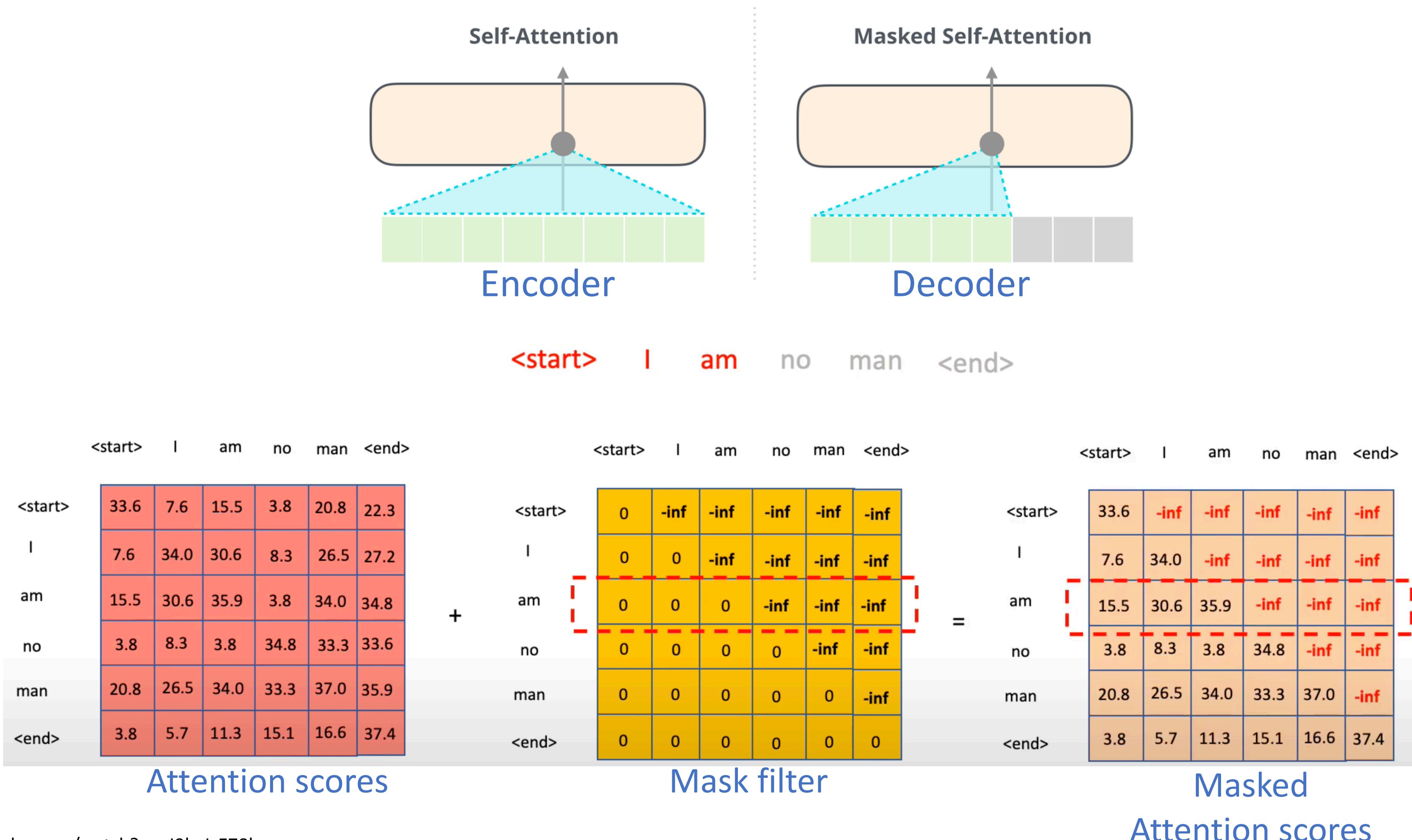
**“Encoder self-attention”:**  
Keys, values, queries all from  
input sequence.



**“Encoder-decoder attention”:**  
Keys, values from the encoder;  
Queries from the previous decoder  
layer.

**“Decoder self-attention”:**  
Keys, values, queries all from the  
decoder; but only calculate the  
attention till the current  
position.

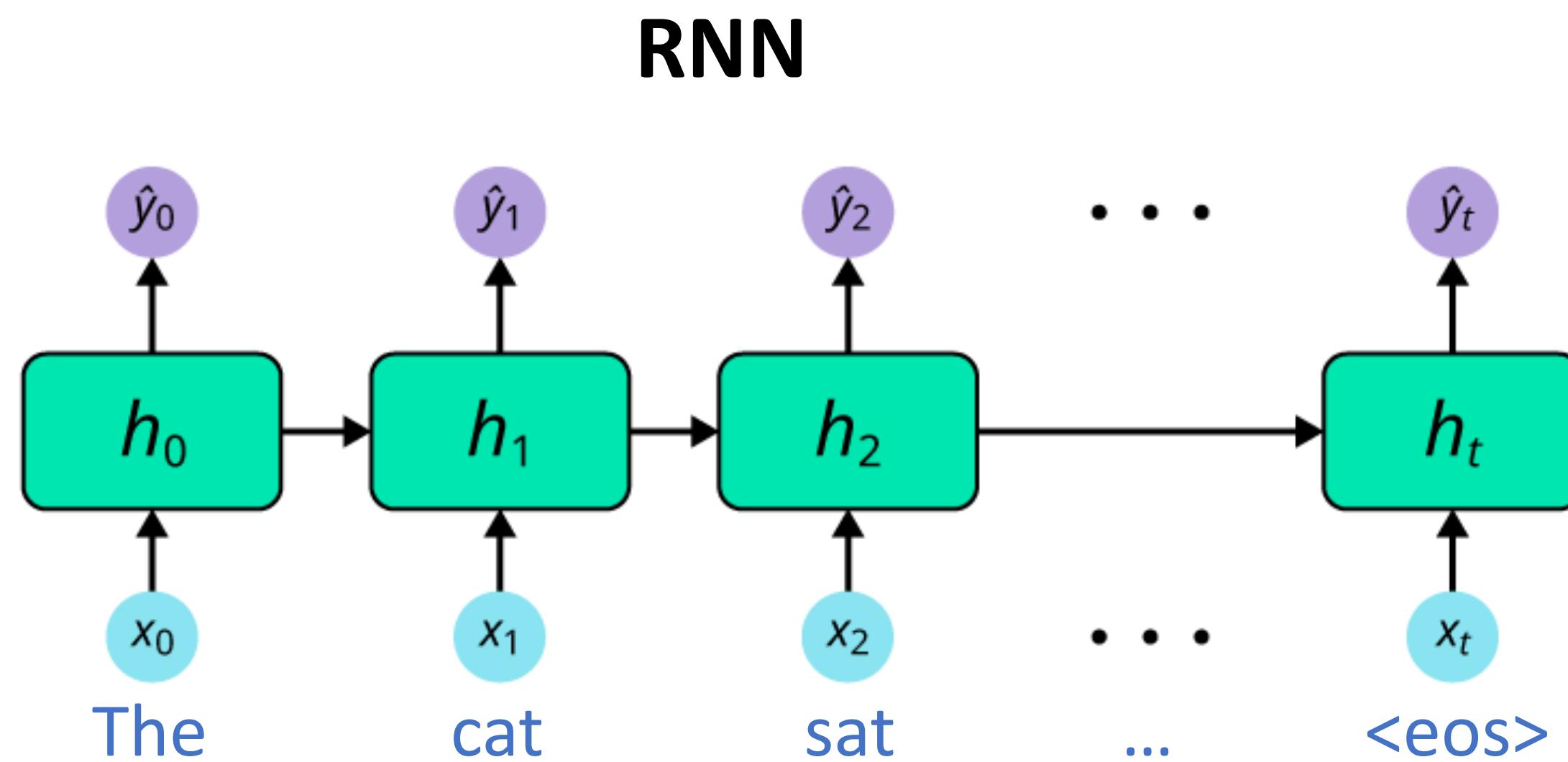
# Masked Multi-Head Attention



# **Positional Encoding**

# The order matters

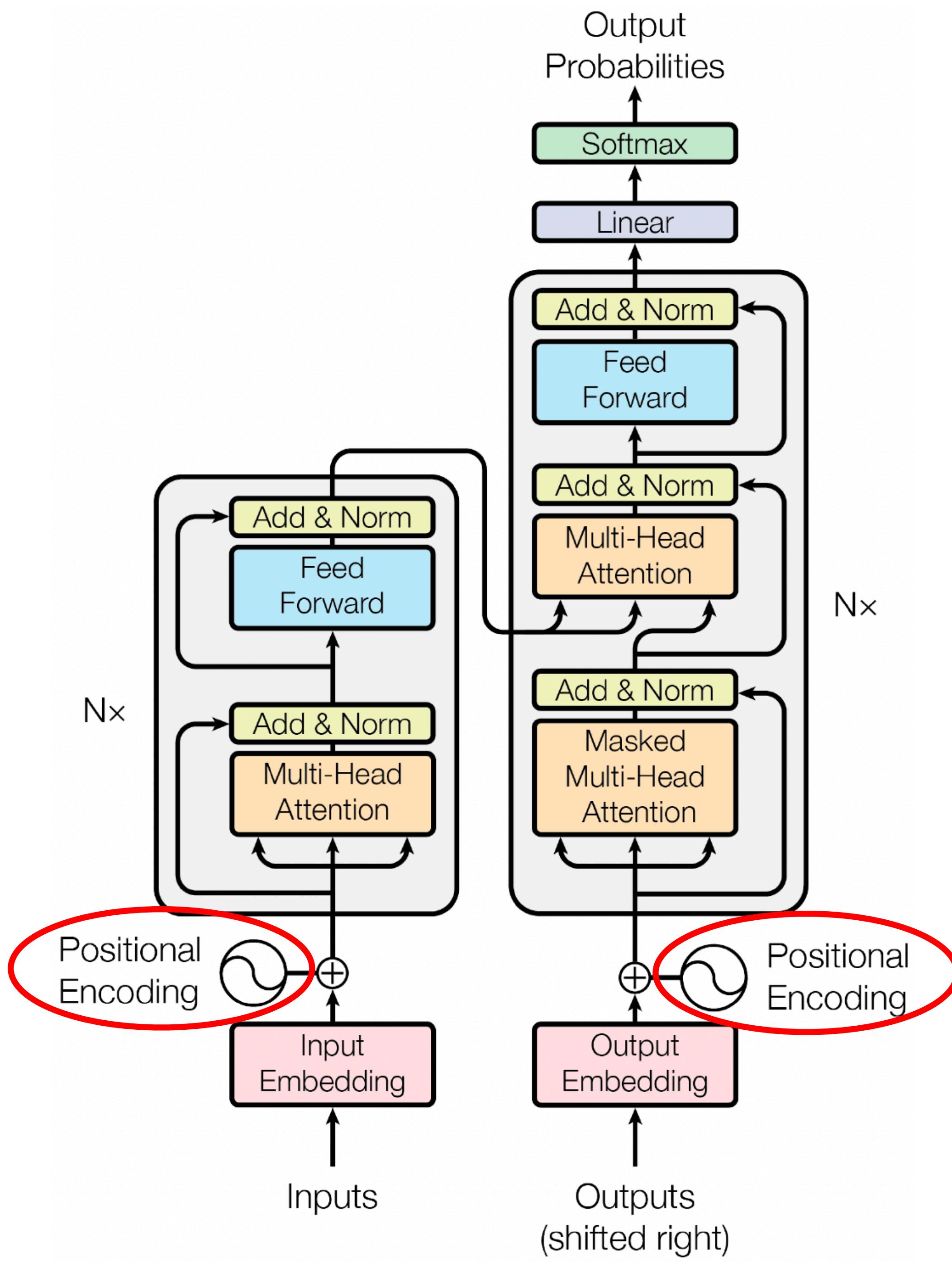
“The cat sat on the mat.”



**Transformer?**

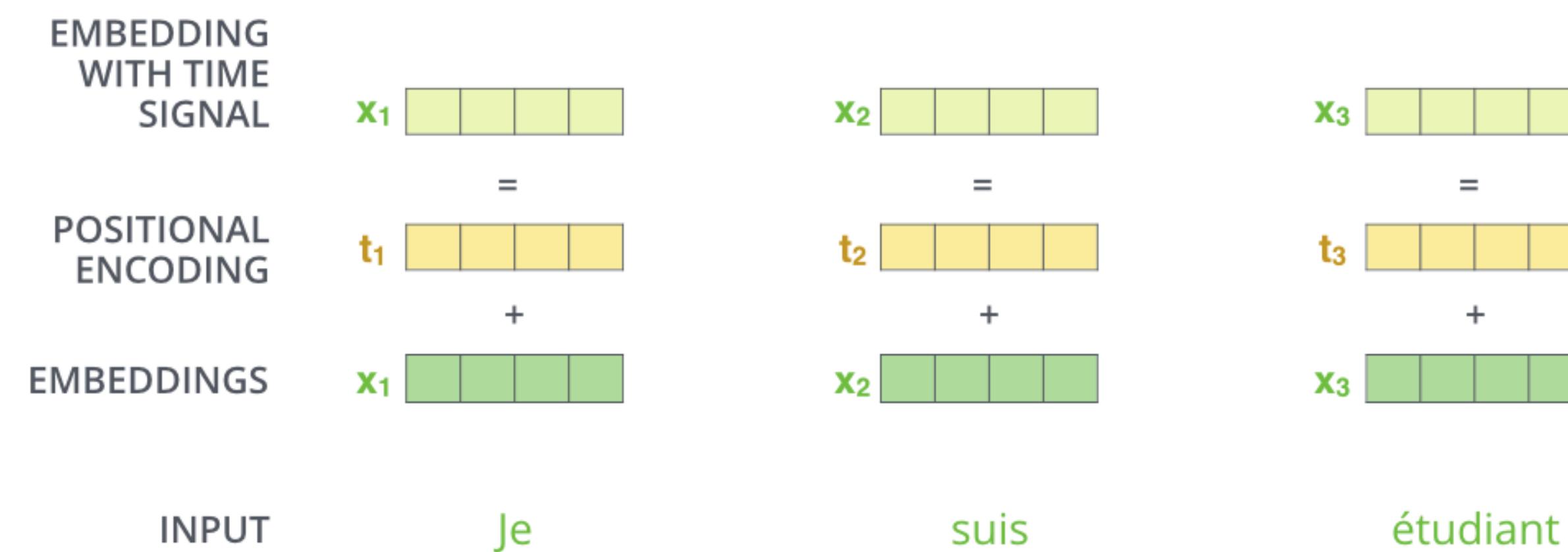
**Q:** Since transformer abandons the recurrent structure,  
how could it embed the positional information?

**A: Positional Encoding.**



# Positional Encoding

**Key idea:** add a new vector (same length) containing positional information to the word embedding.



# Criteria of the Positional Encoding Vector

- Unique encoding for each time-step
- Consistent distance between any two time-steps
- Should be bounded, generalize to longer sentences
- Deterministic

# Criteria of the Positional Encoding Vector

- Unique encoding for each time-step ✓
- Consistent distance between any two time-steps ✓
- Should be bounded, generalize to longer sentences ✗
- Deterministic ✓

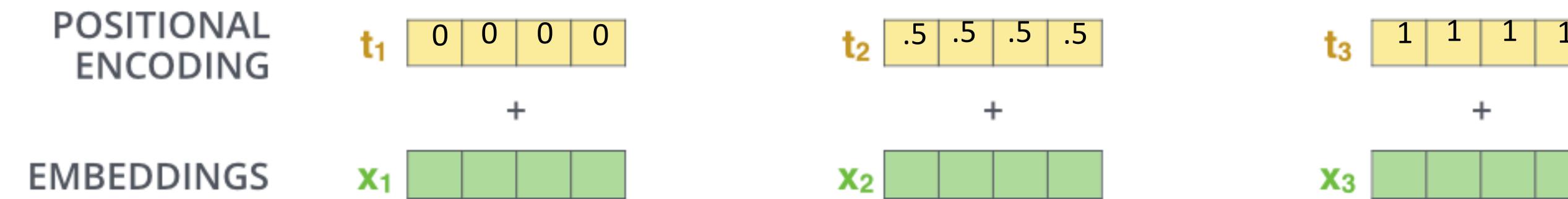
Option 1: How about using the order number?



# Criteria of the Positional Encoding Vector

- Unique encoding for each time-step ✓
- Consistent distance between any two time-steps ✗
- Should be bounded, generalize to longer sentences ✓
- Deterministic ✗

Option 2: How about using fractions ( $1/(n-1)$ )?



# Positional Embedding in Transformer

$i$ :  $i$ -th dimension of the vector

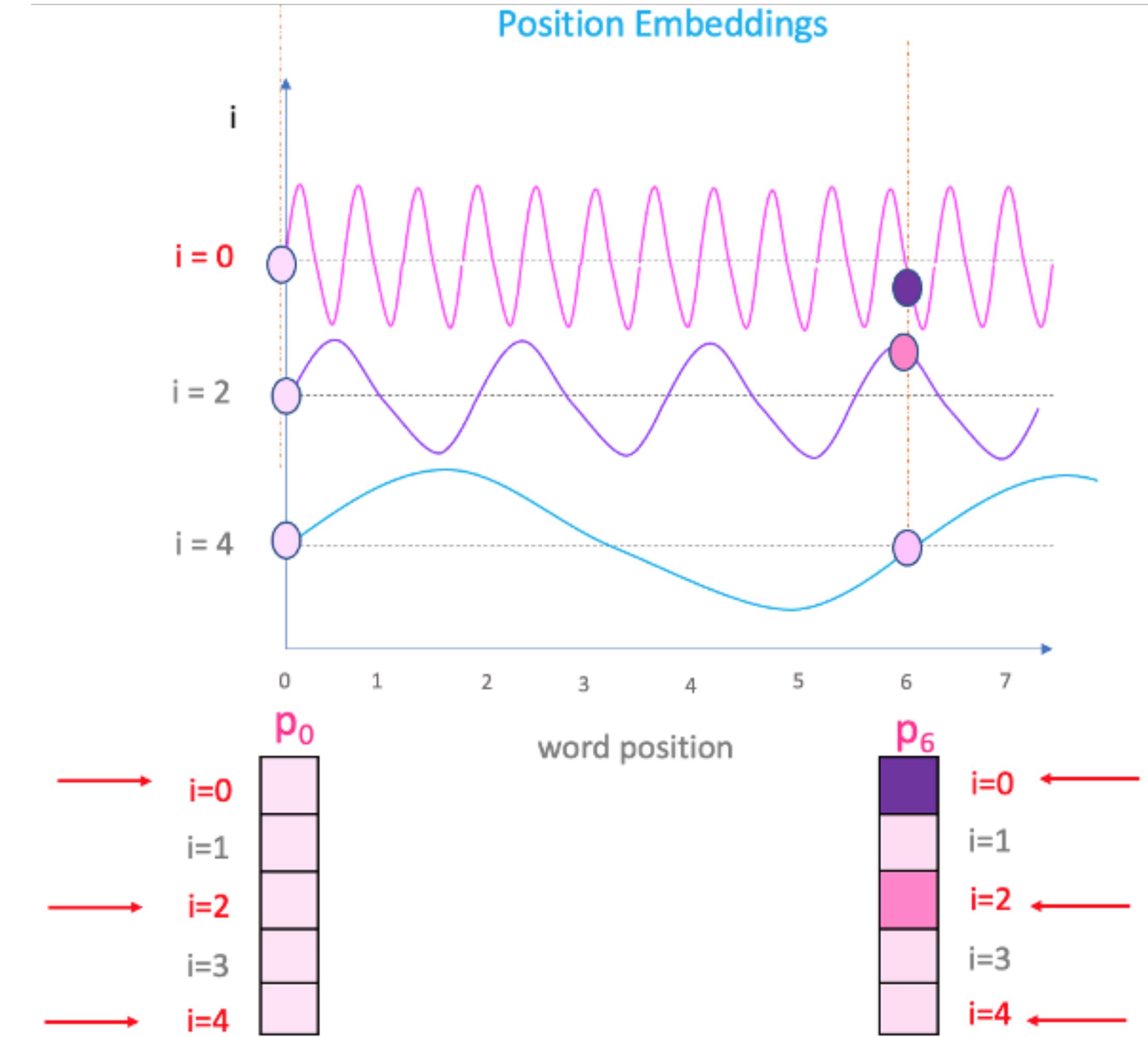
**Key idea:** using frequencies.

$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$

$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$

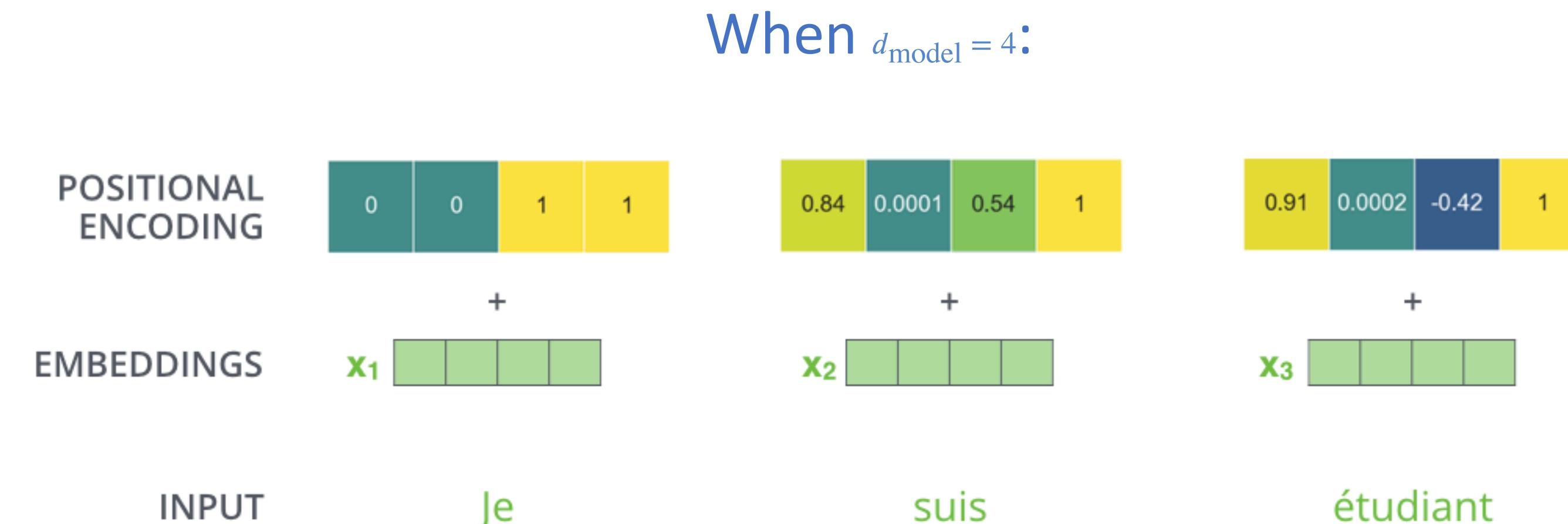
$pos$ : the position of the word in the sentence.

$d_{model}$ : the dimension of the word embedding (=512 in the paper)



# Positional Embedding in Transformer

- Unique encoding for each time-step ✓
- Consistent distance between any two time-steps ✓
- Should be bounded, generalize to longer sentences ✓
- Deterministic ✓



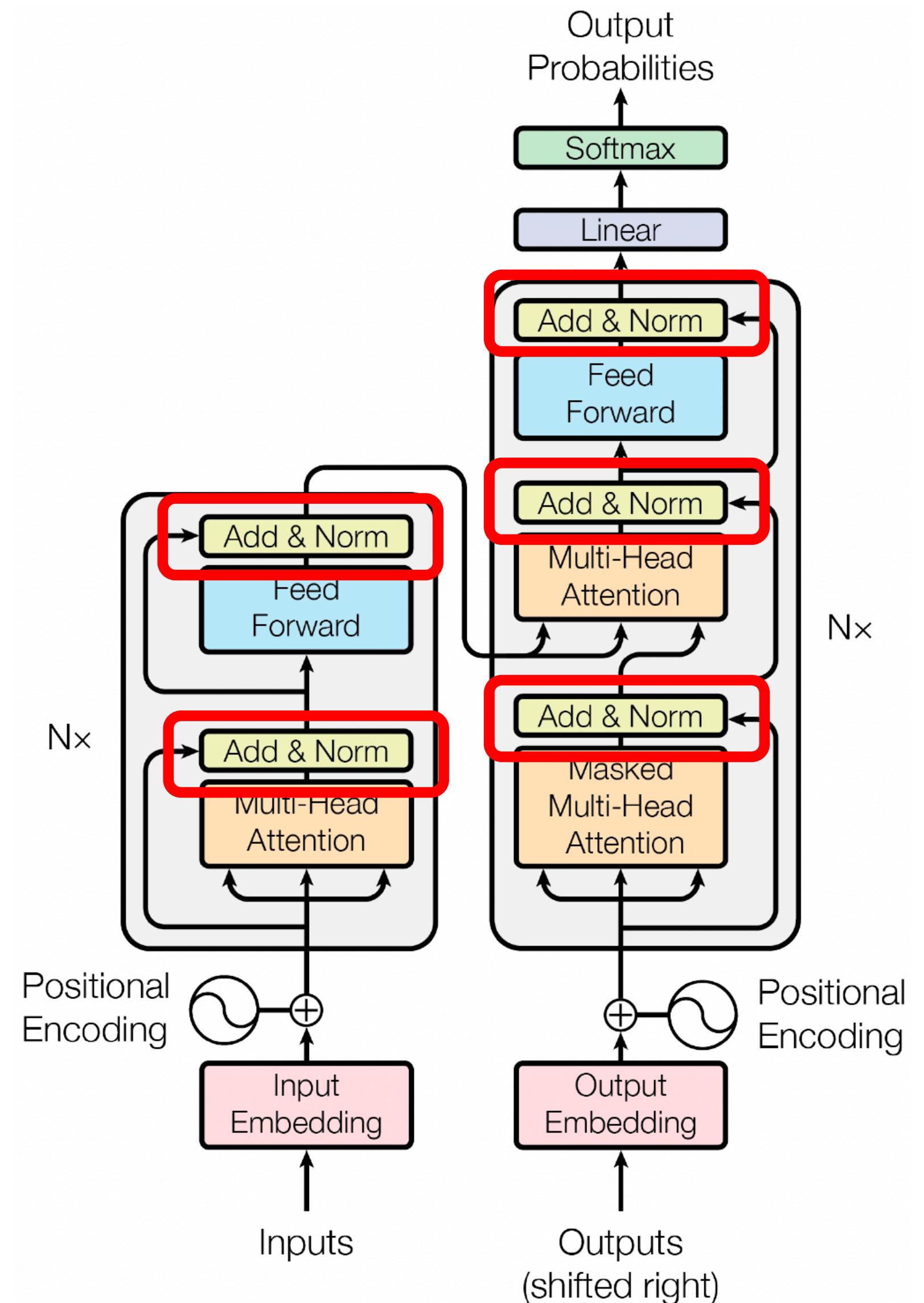
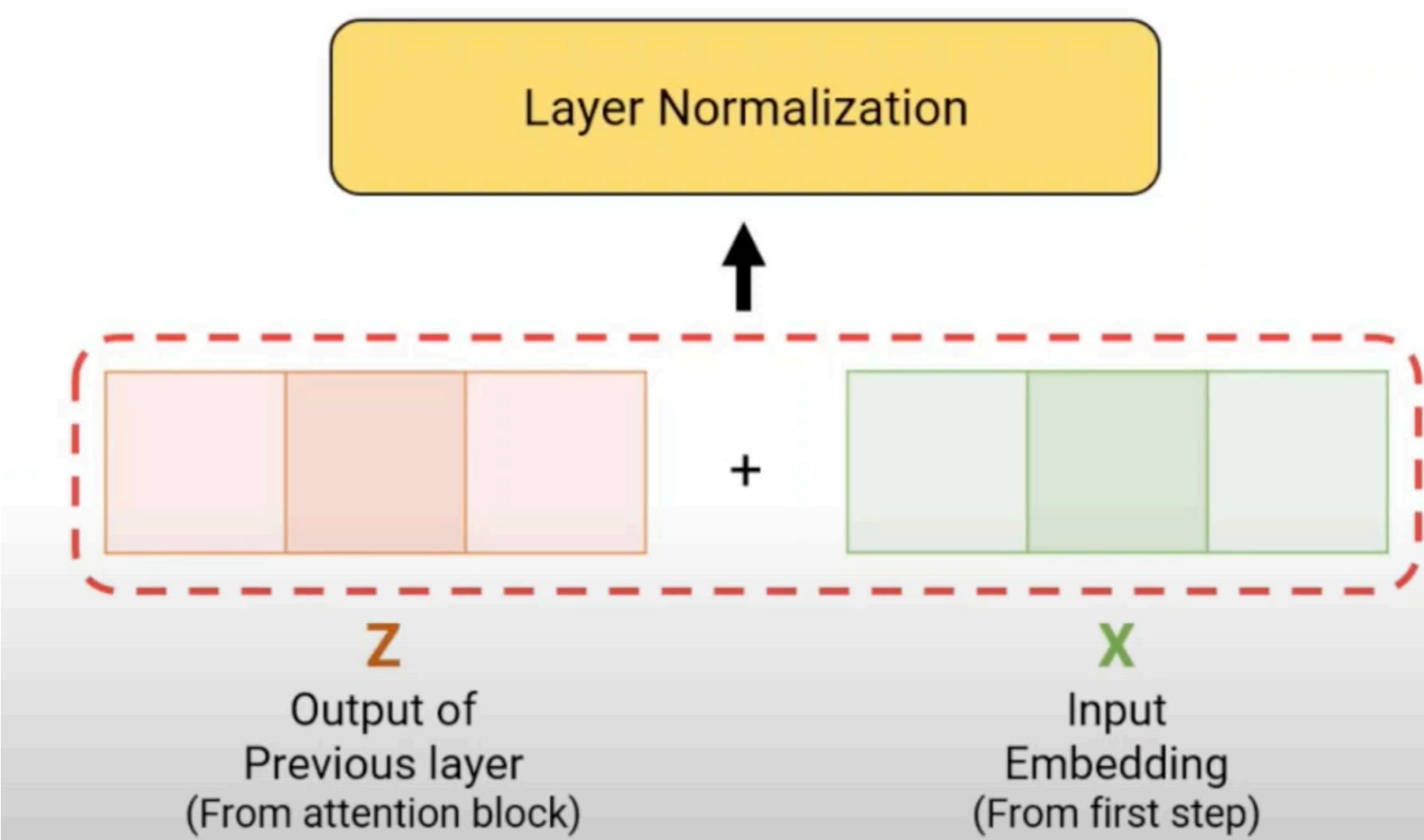
# References

- Video by Hedu. “Visual Guide to Transformer Neural Networks – (Episode 1) Positional Embeddings”, 2021. <https://www.youtube.com/watch?v=dichIcUZfOw>
- Video by Halfling Wizard. “Attention is All You Need – Paper Explained”, 2021. <https://www.youtube.com/watch?v=XowwKOAWYoQ>

# Other Layers

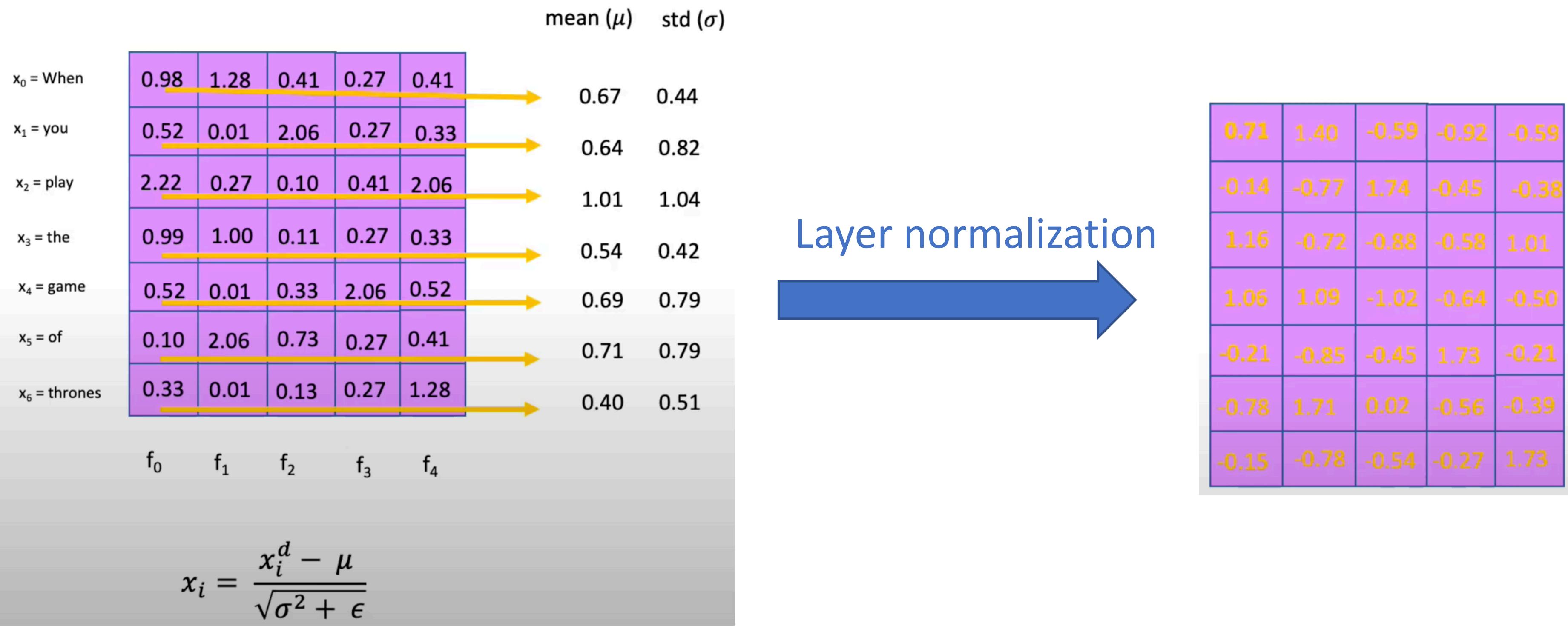
# Add & Normalization

- Residual connection
  - Preserve (reuse) the earlier information
  - Prevent vanishing gradient problem



# Add & Normalization

- Layer normalization
  - Stabilize training
  - Faster training
  - Prevent weight explosion

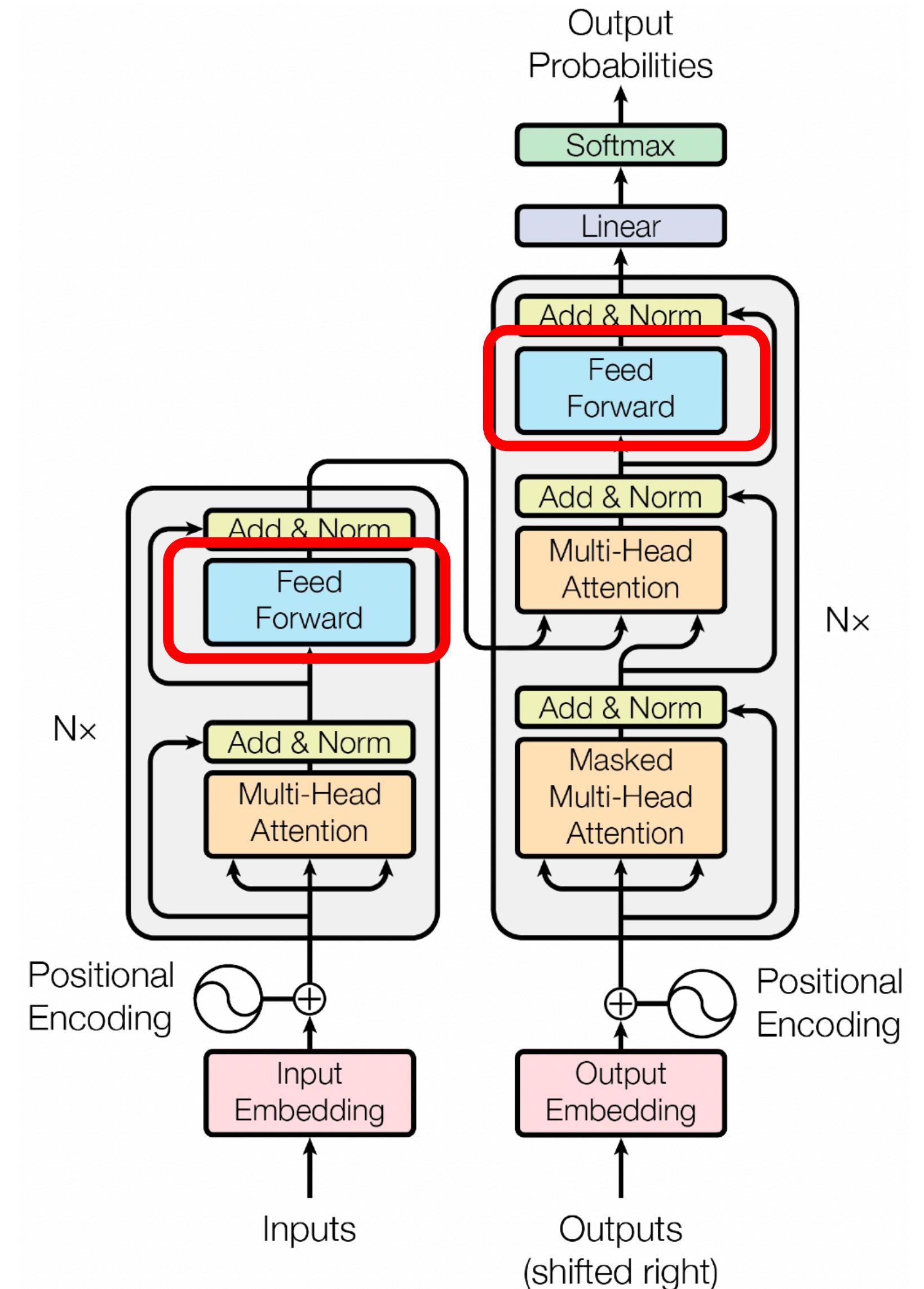


# Feed Forward

- Two fully-connected layers

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- The exact same feed-forward network (with different parameters) is independently applied to the encoder and decoder.

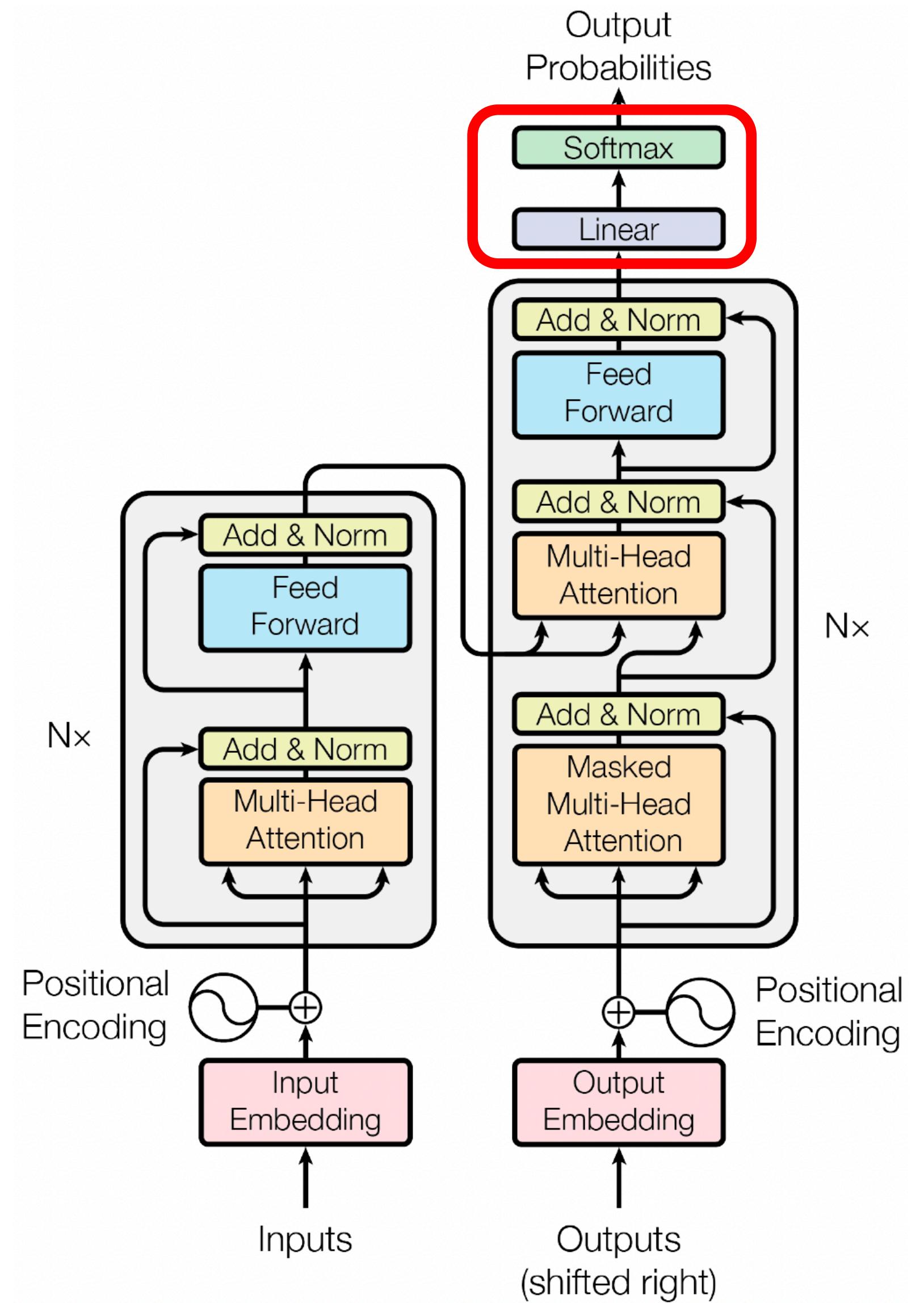
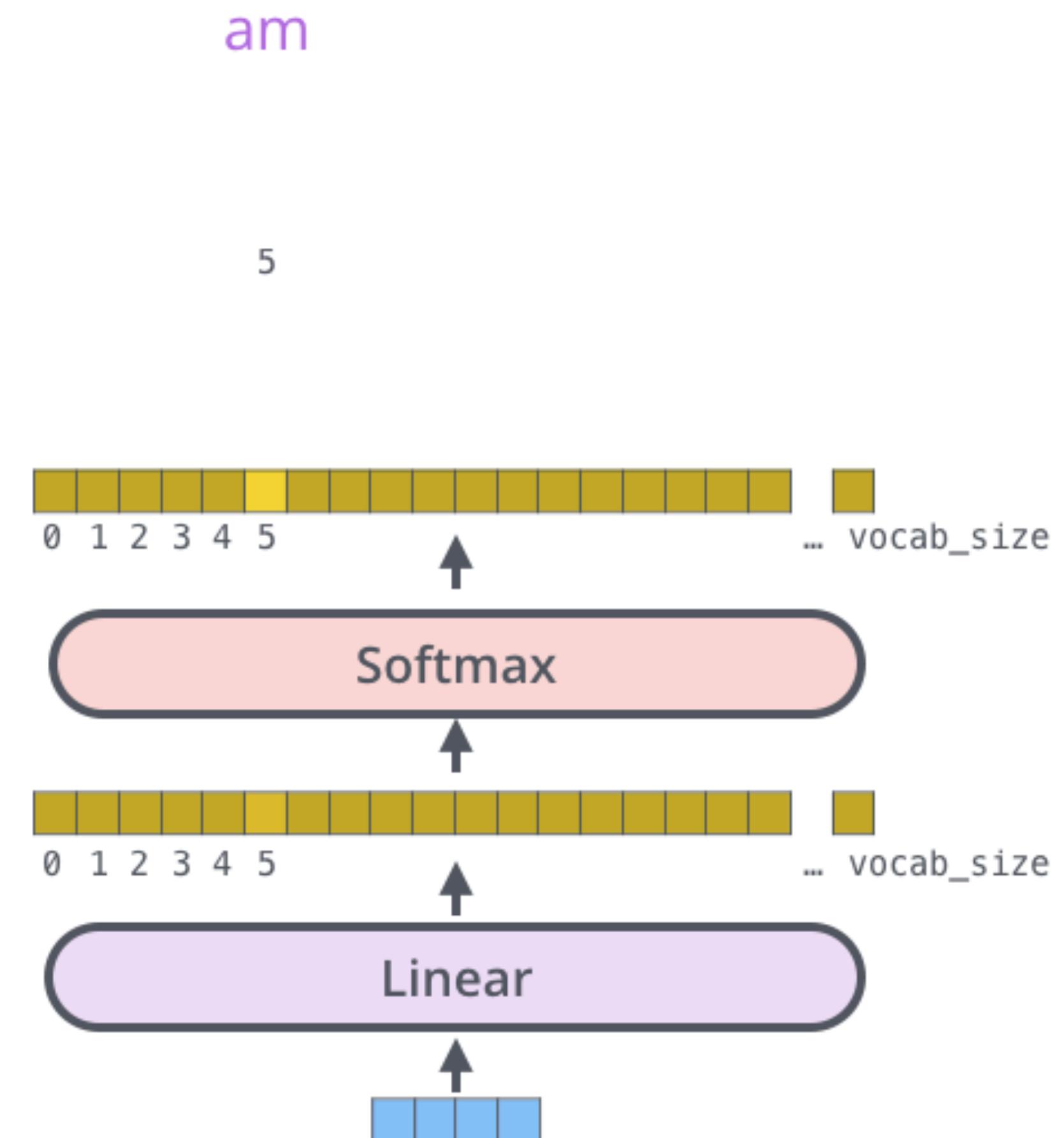


# Final linear and Softmax Layer

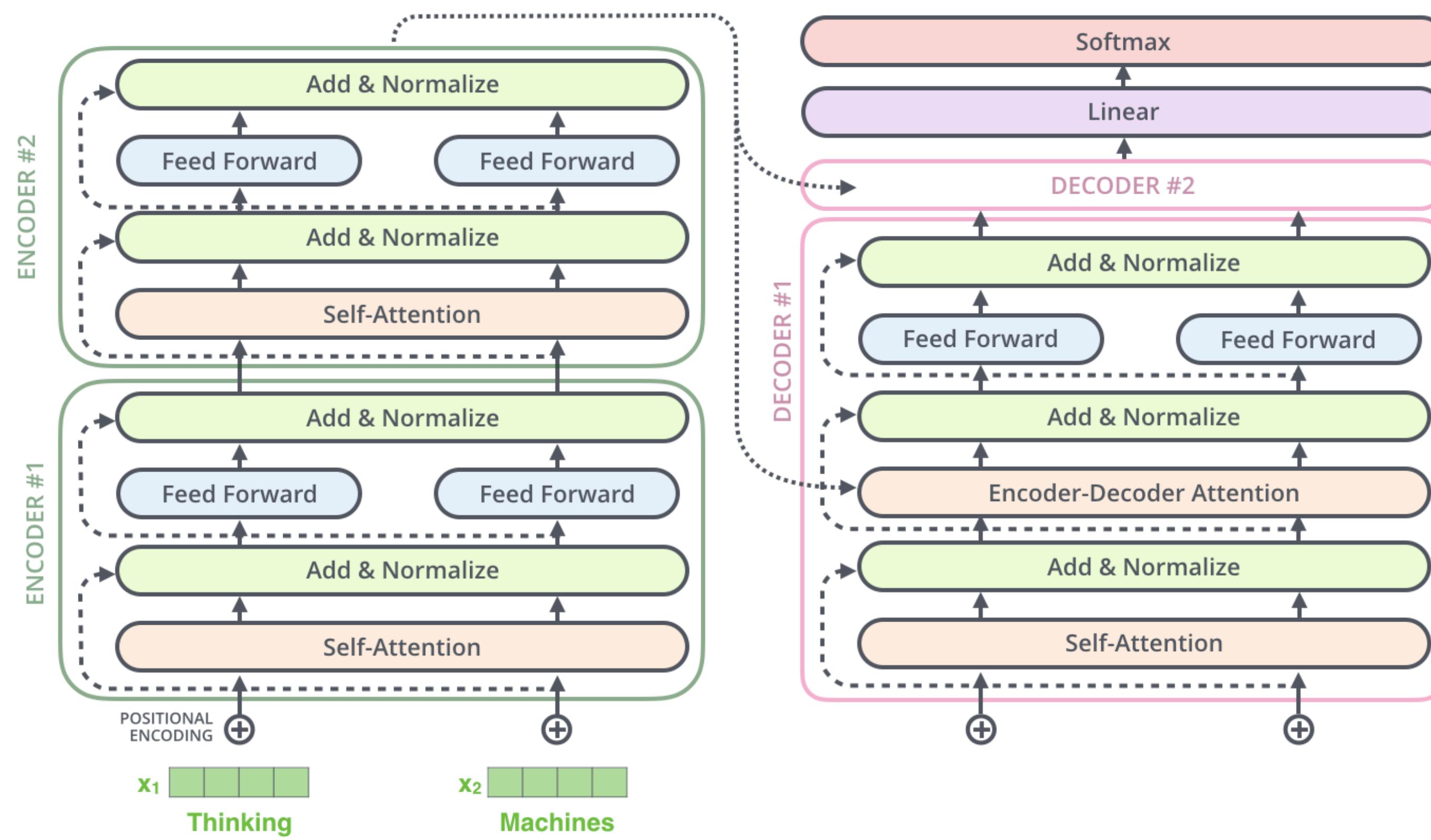
Which word in our vocabulary  
is associated with this index?

Get the index of the cell  
with the highest value  
(`argmax`)

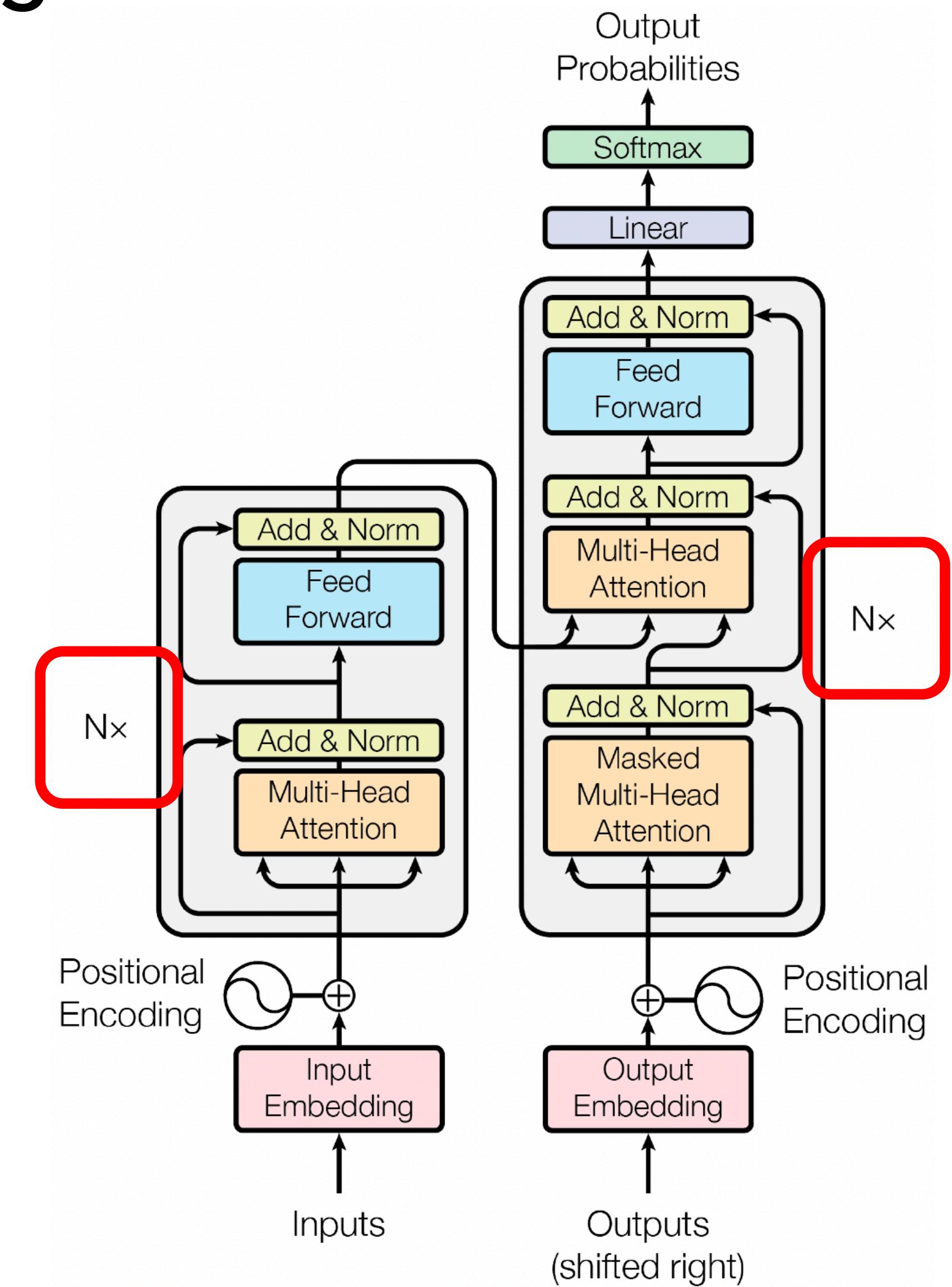
`log_probs`

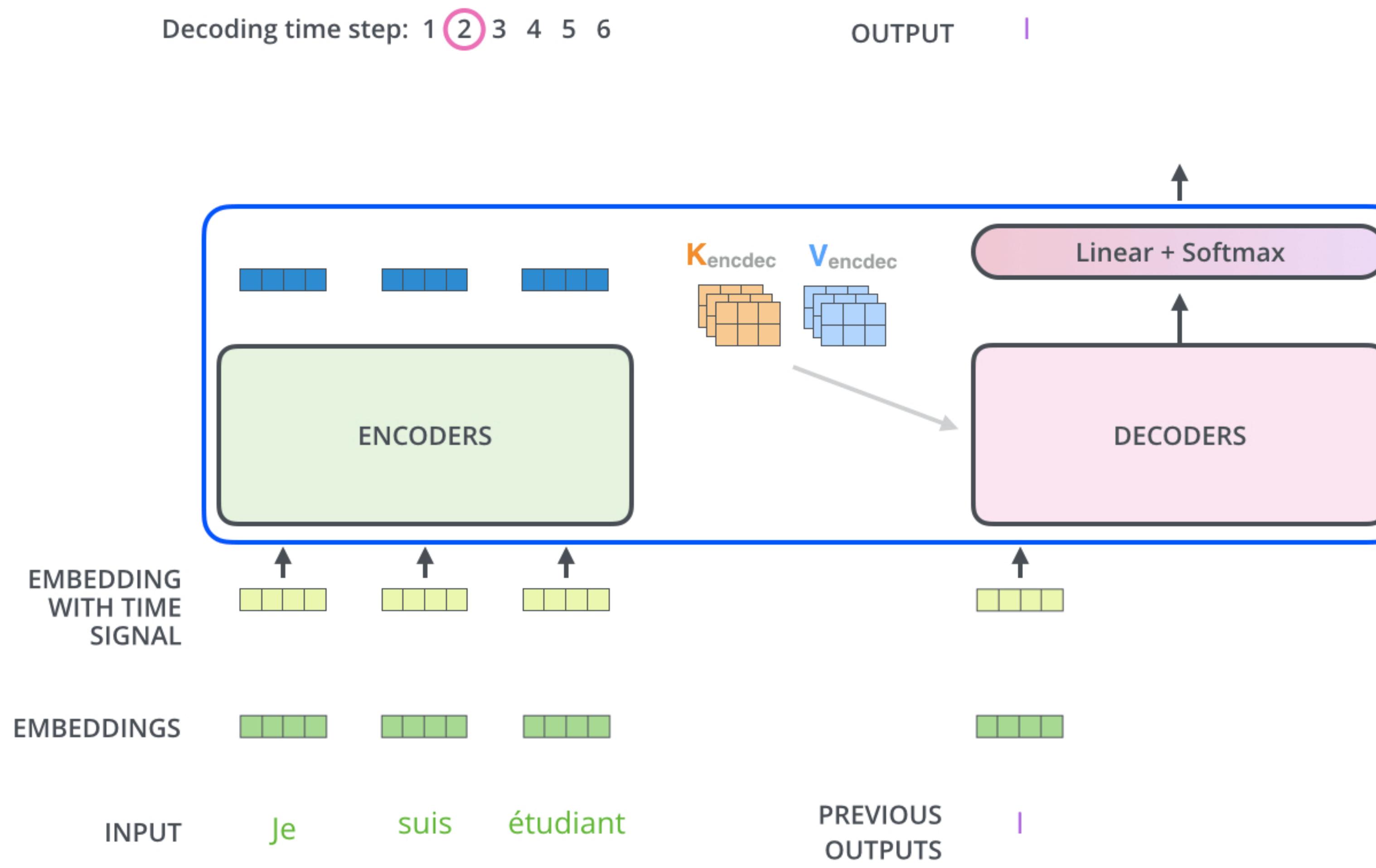


# Stack of Encoders and Decoders



N=6 in the paper



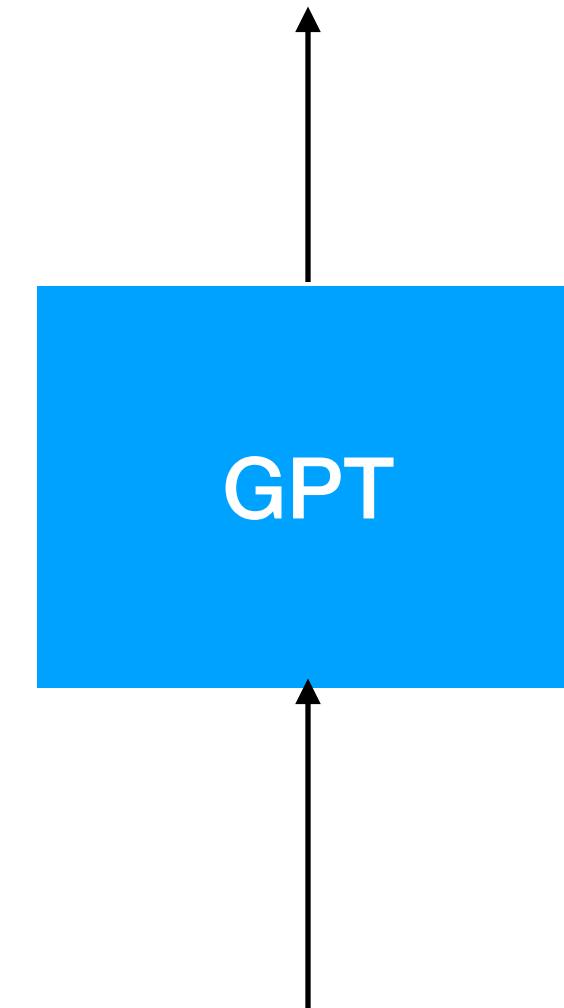


# Putting it together

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers

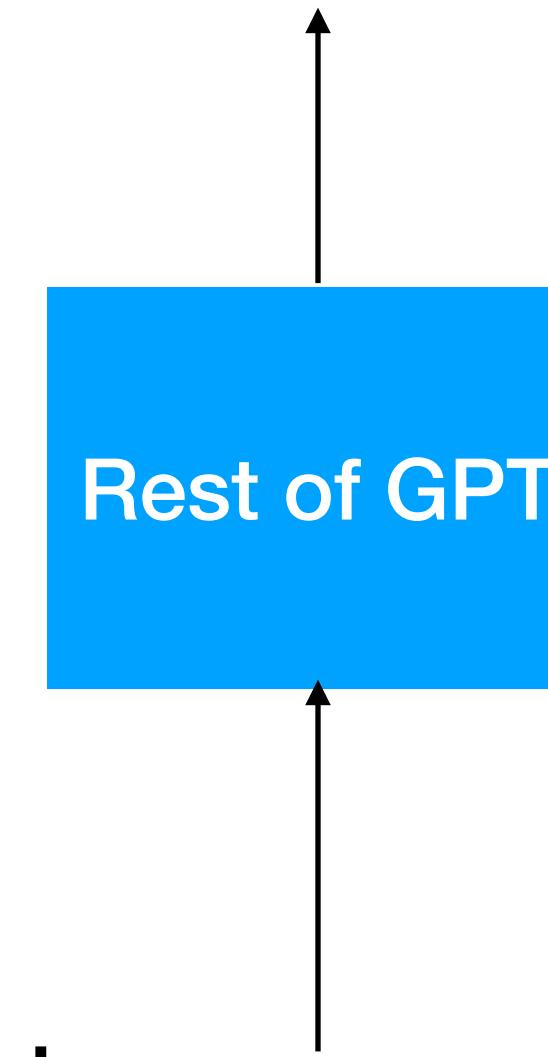


Once upon a time

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers



The text is converted into token vectors -

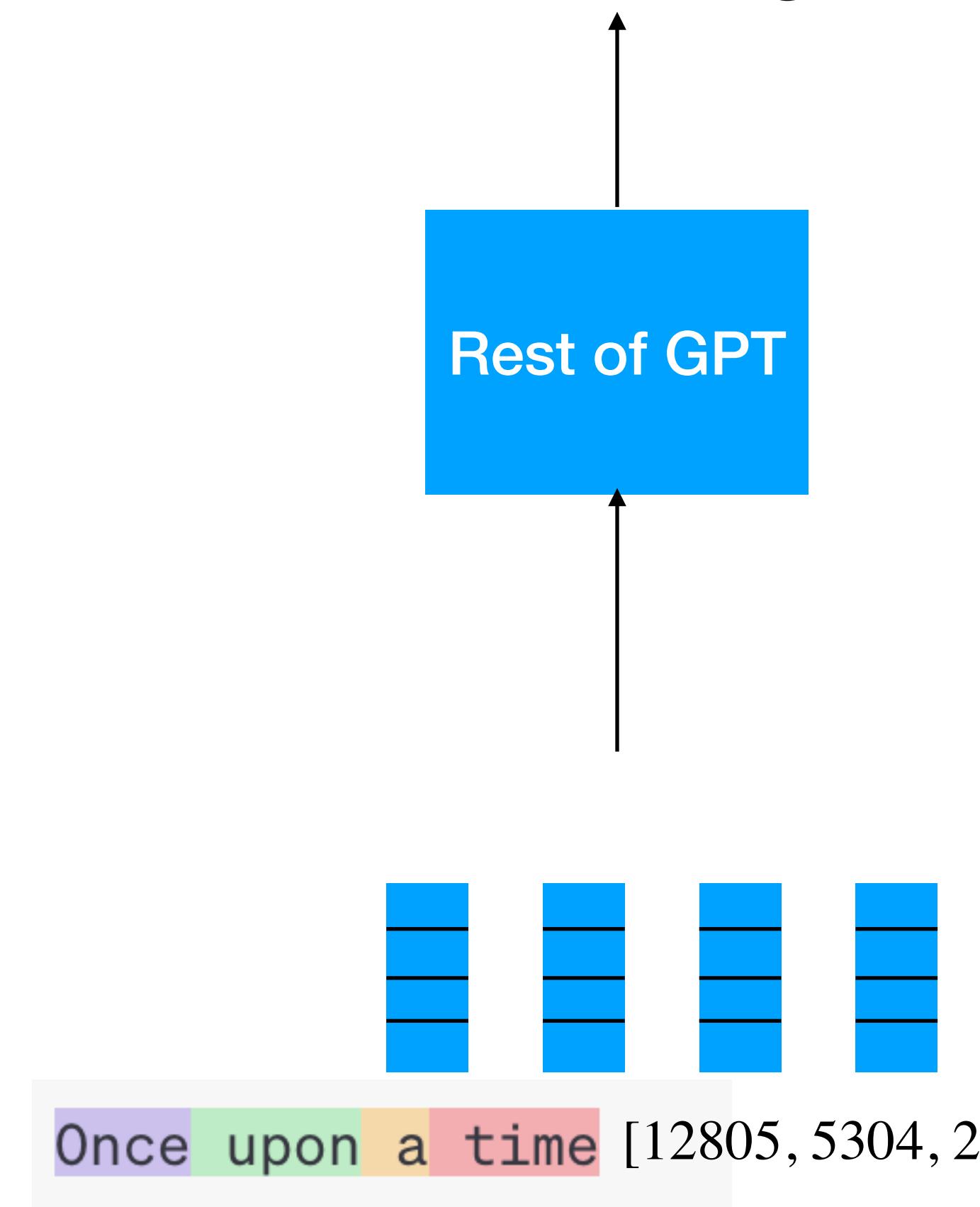
Once upon a time [12805, 5304, 264, 892]

Once upon a time

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers

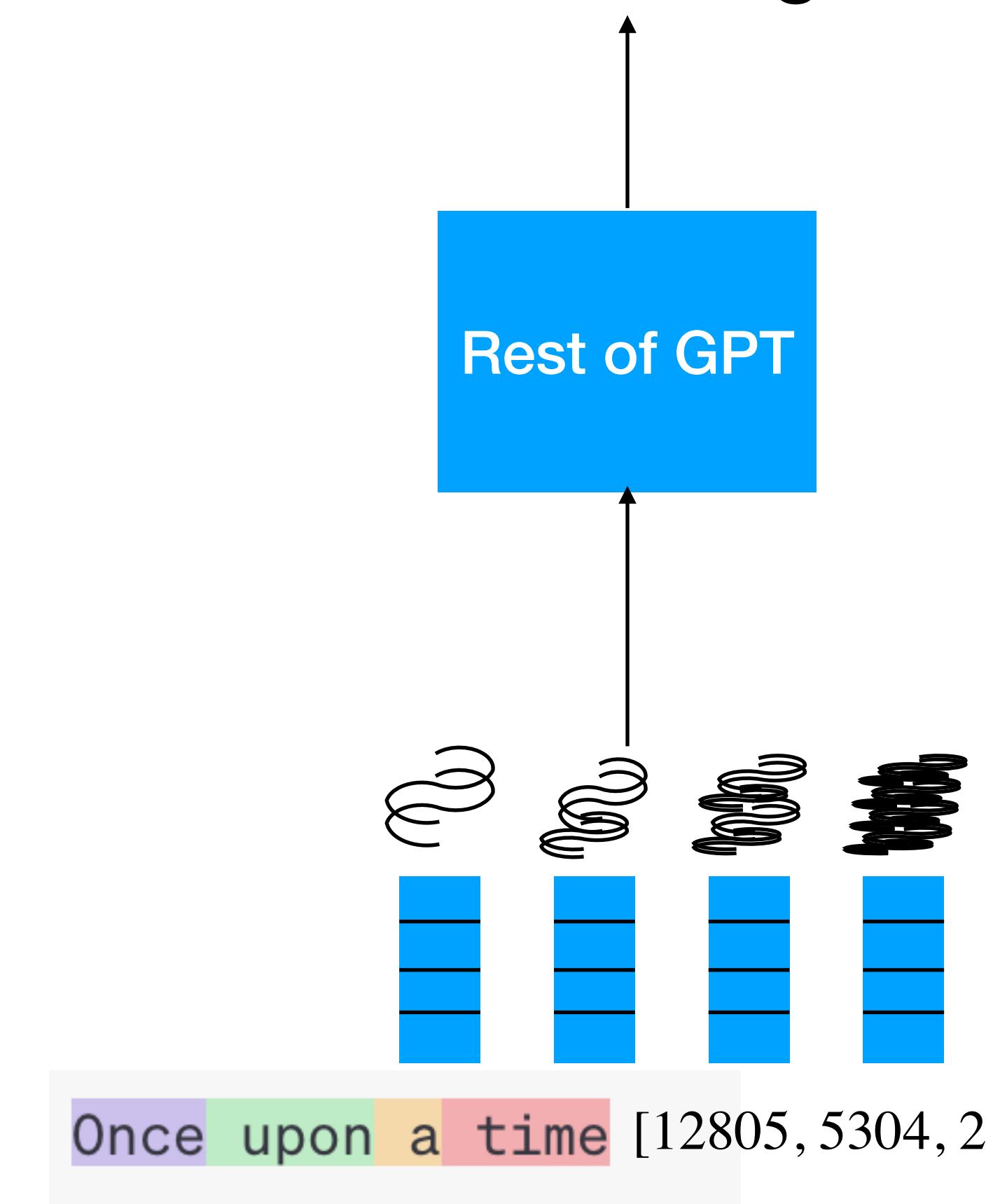


Sequence of integers is mapped to a set of embedding vectors by an embedding matrix  $W_E$

First we will try to understand GPT

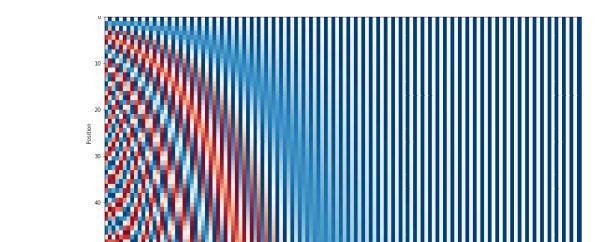
We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers



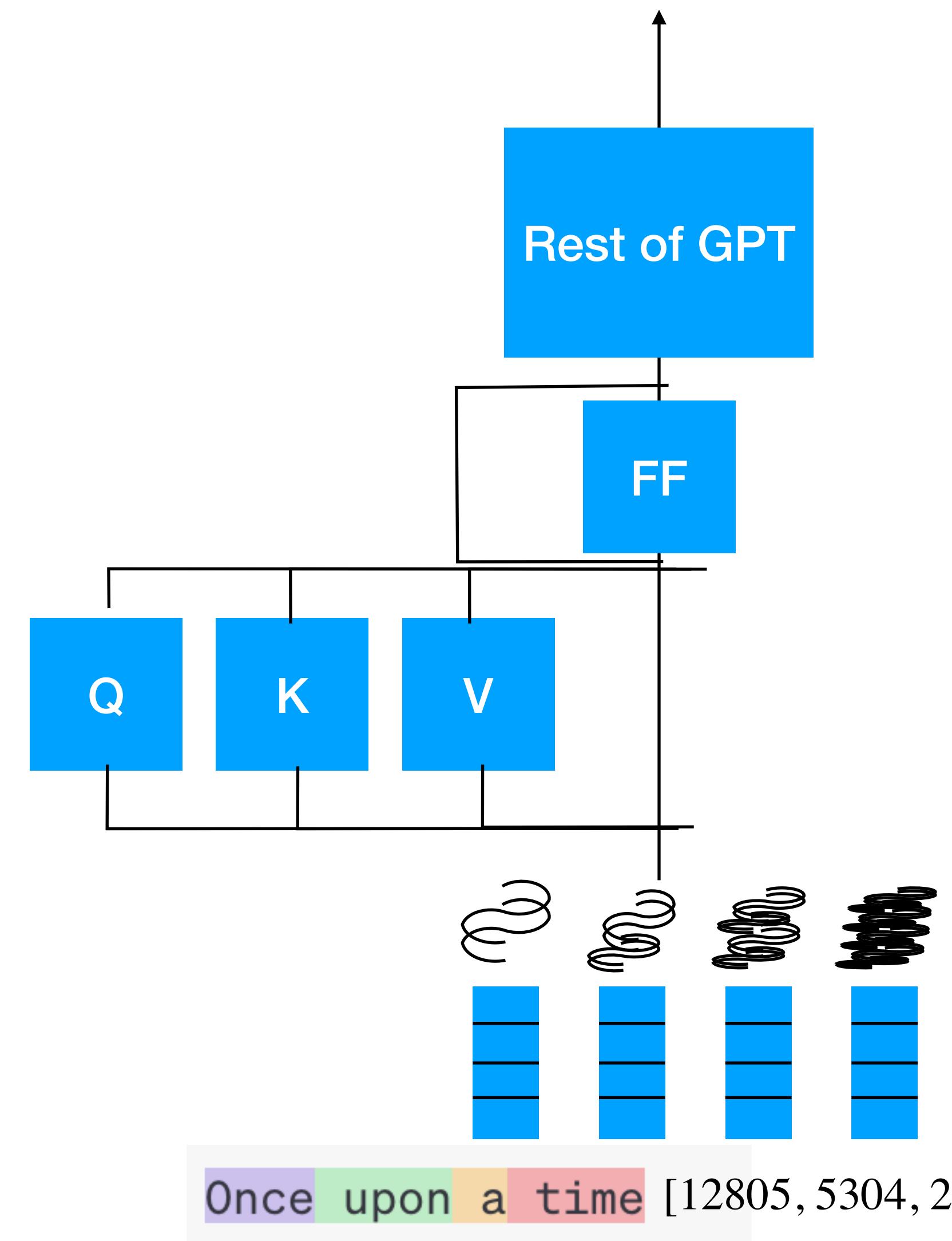
Positional encoding is added to the embedding -

$$\psi'(w_t) = \psi(w_t) + \vec{p}_t$$



First we will try to understand GPT

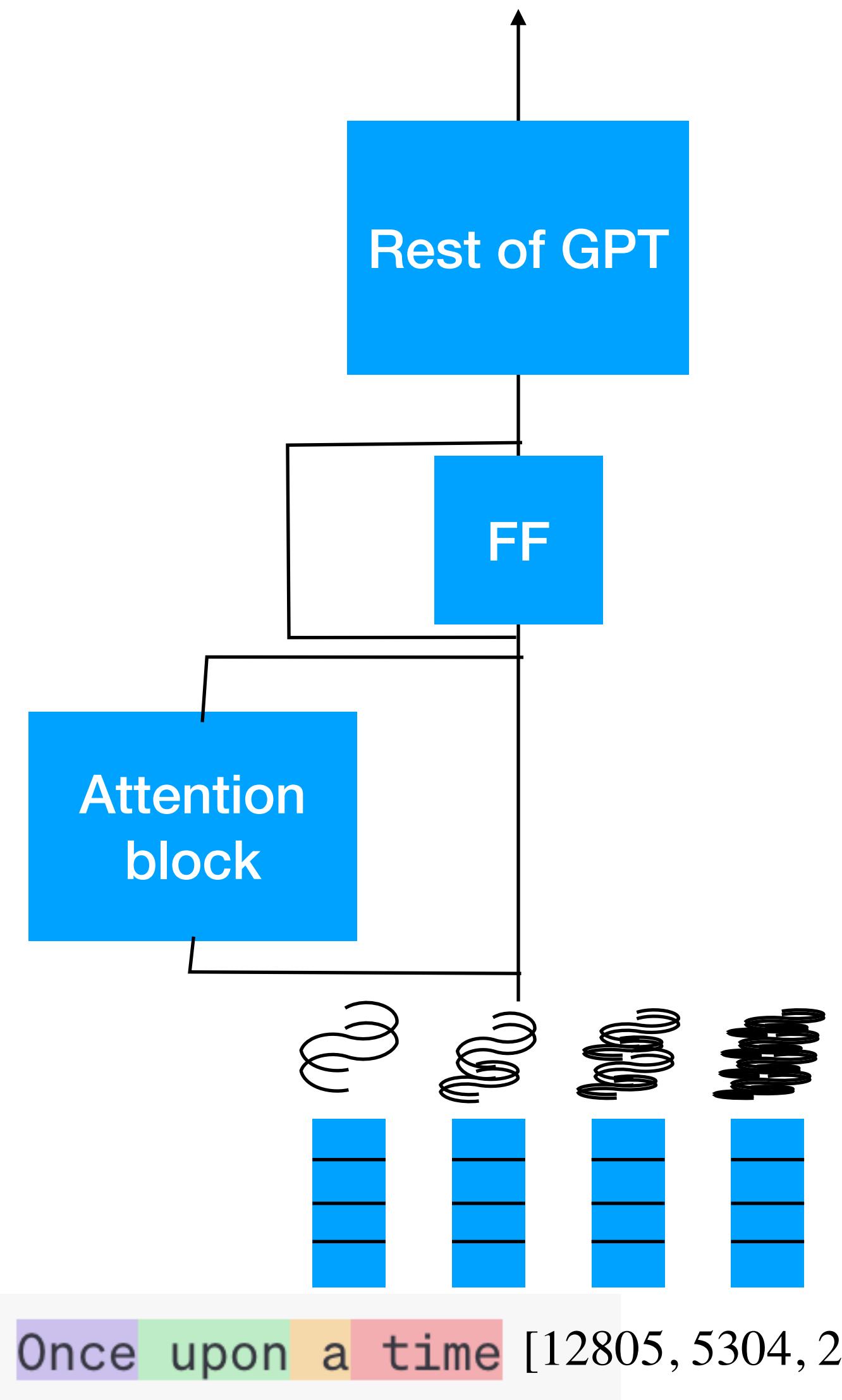
We want to understand what happens when we provide a prompt to GPT



This residual stream is then passed through a ‘self attention + FF layer block’

First we will try to understand GPT

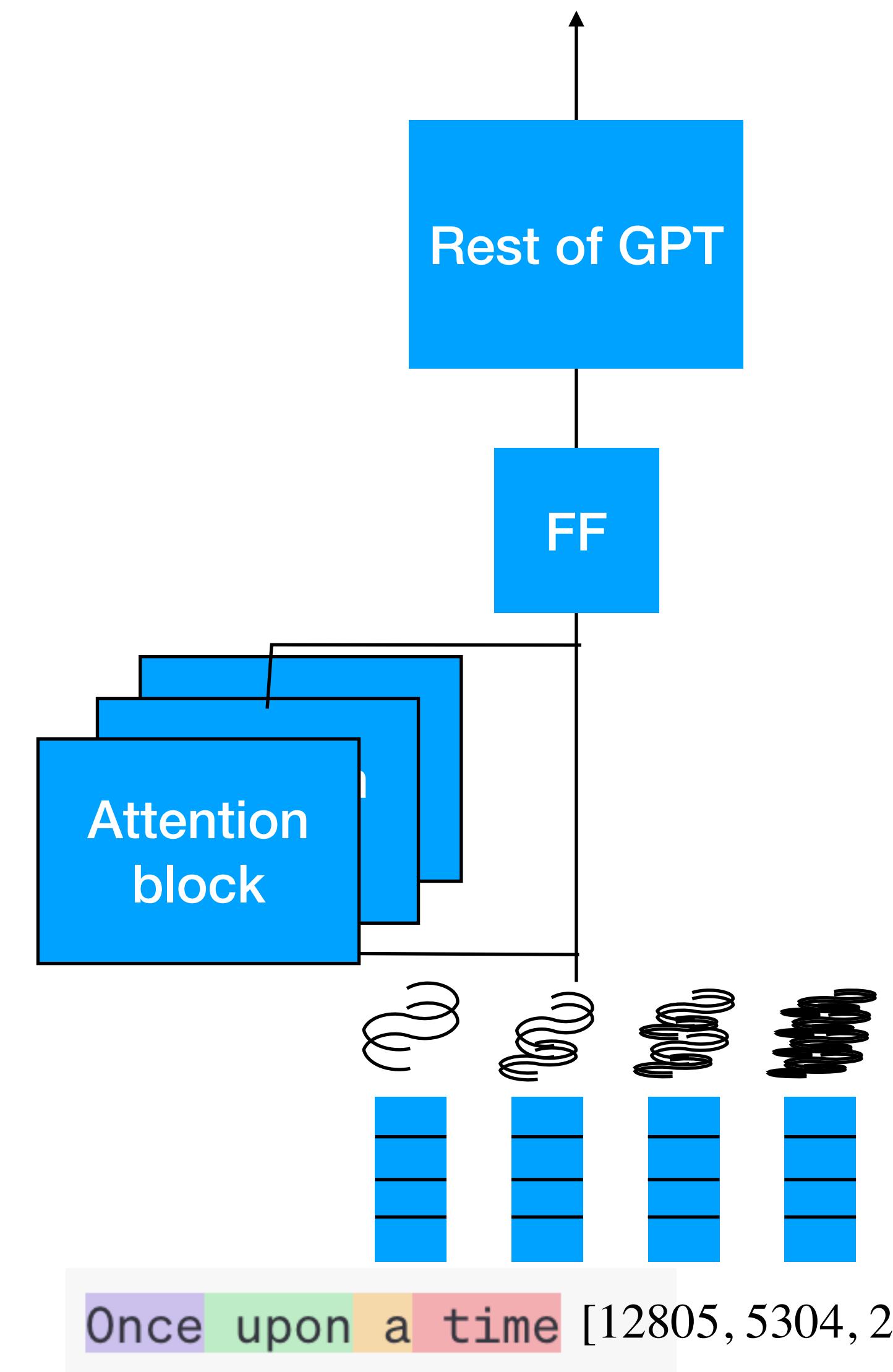
We want to understand what happens when we provide a prompt to GPT



This residual stream is then passed through a ‘self attention + FF layer block’

First we will try to understand GPT

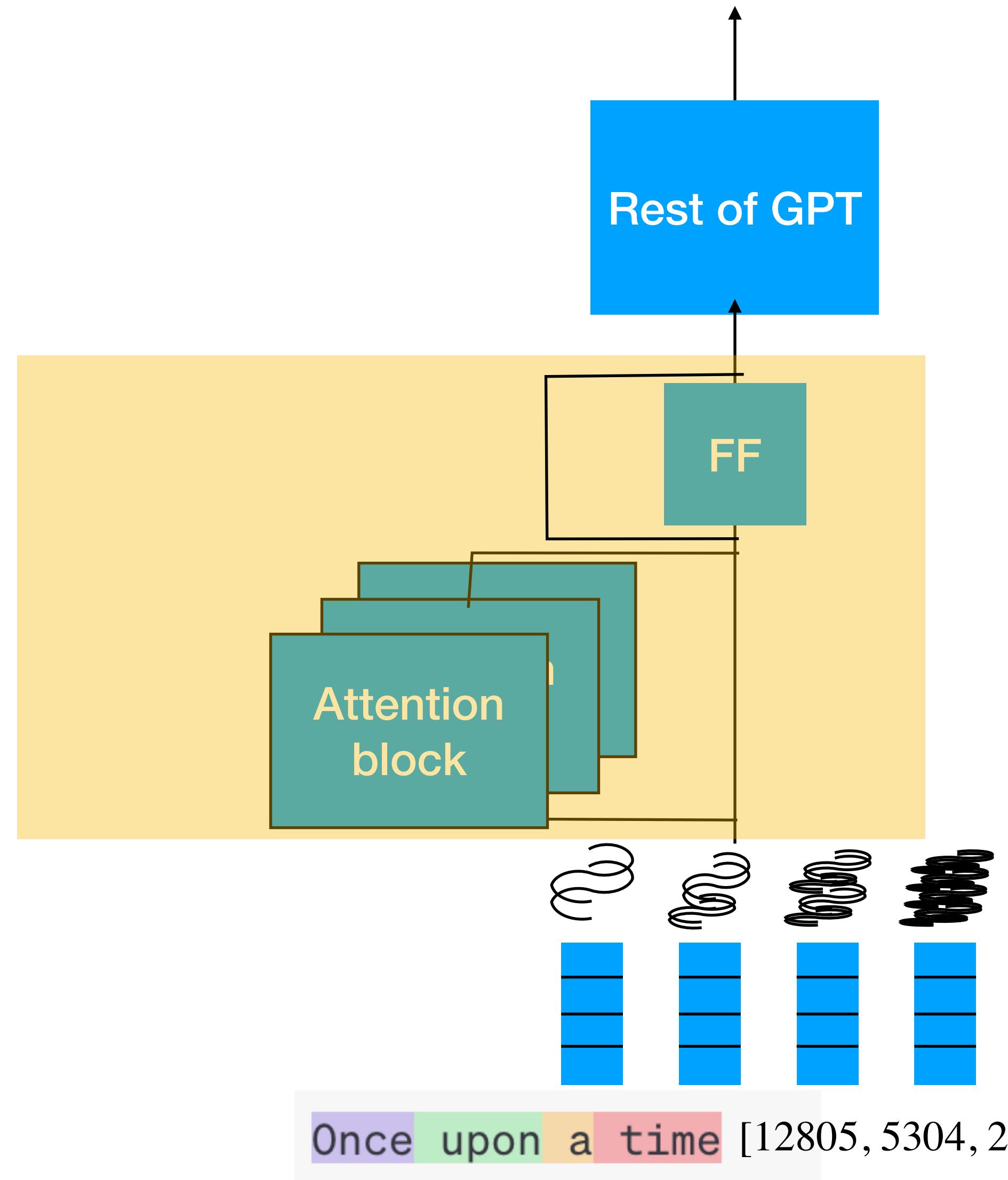
We want to understand what happens when we provide a prompt to GPT



In practice, there are multiple heads running in parallel

First we will try to understand GPT

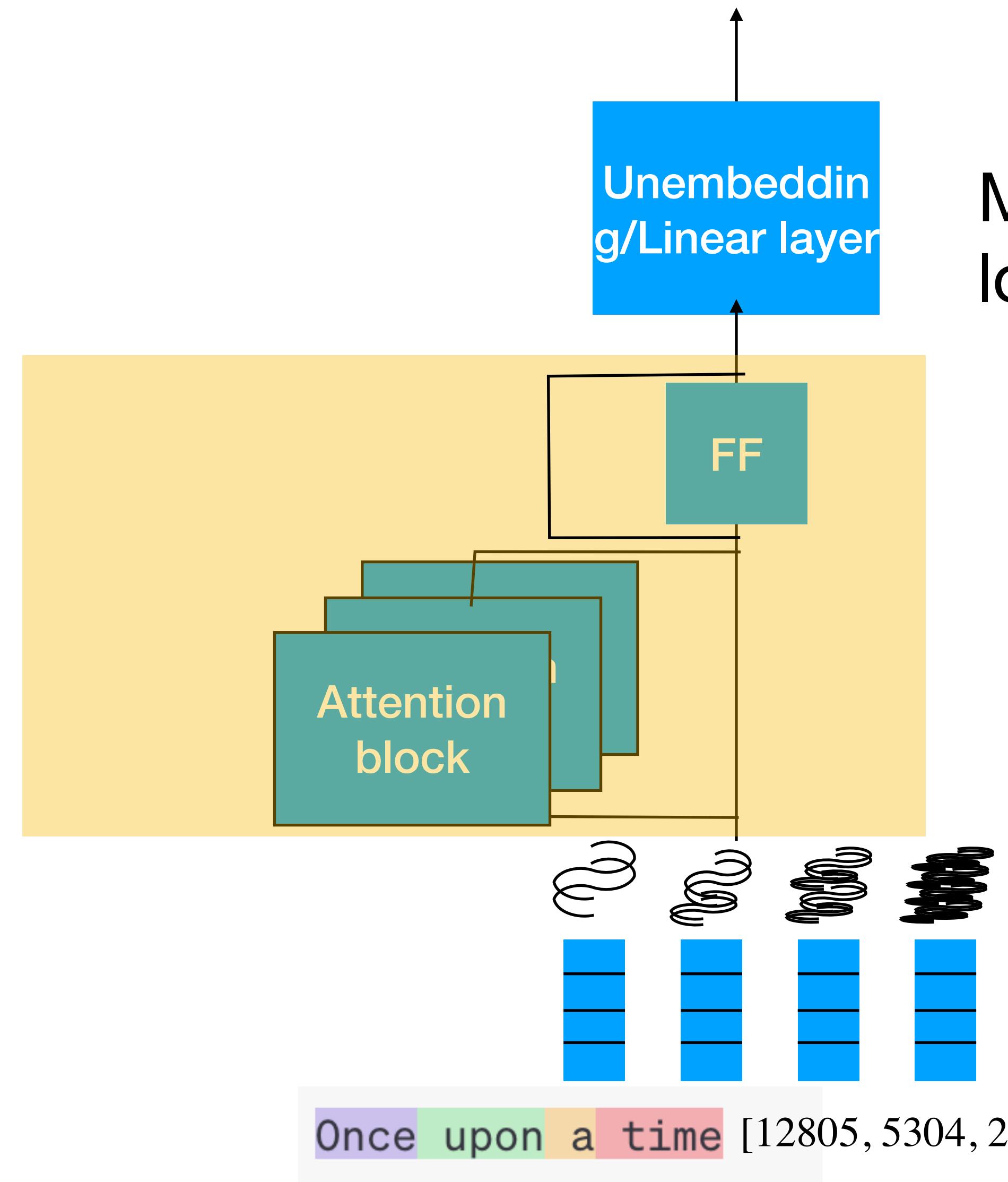
We want to understand what happens when we provide a prompt to GPT



This is one layer of the transformer - this structure is repeated many times

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT



Maps from embedding space to logits for the token space

Many layers of this structure

# References

- 3Blue1Brown - [Video 1](#), [Video2](#)
- Andrej Karpathy - [NanoGPT implementation](#), [Video Tutorial](#)



That's all  
folks

QUESTIONS?

# Understanding Transformers - 2

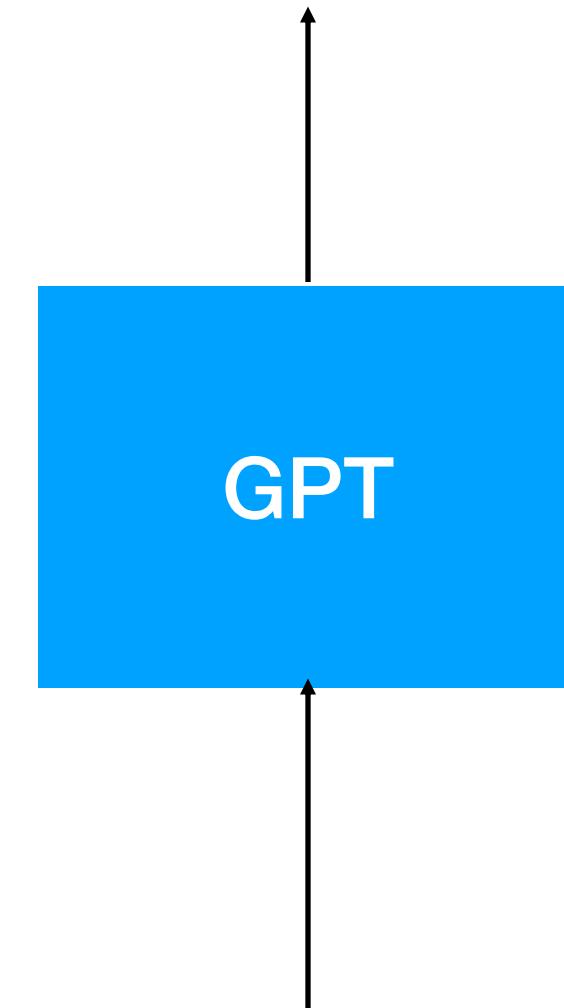
Vinay P. Namboodiri

# Using Transformers – an example

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers

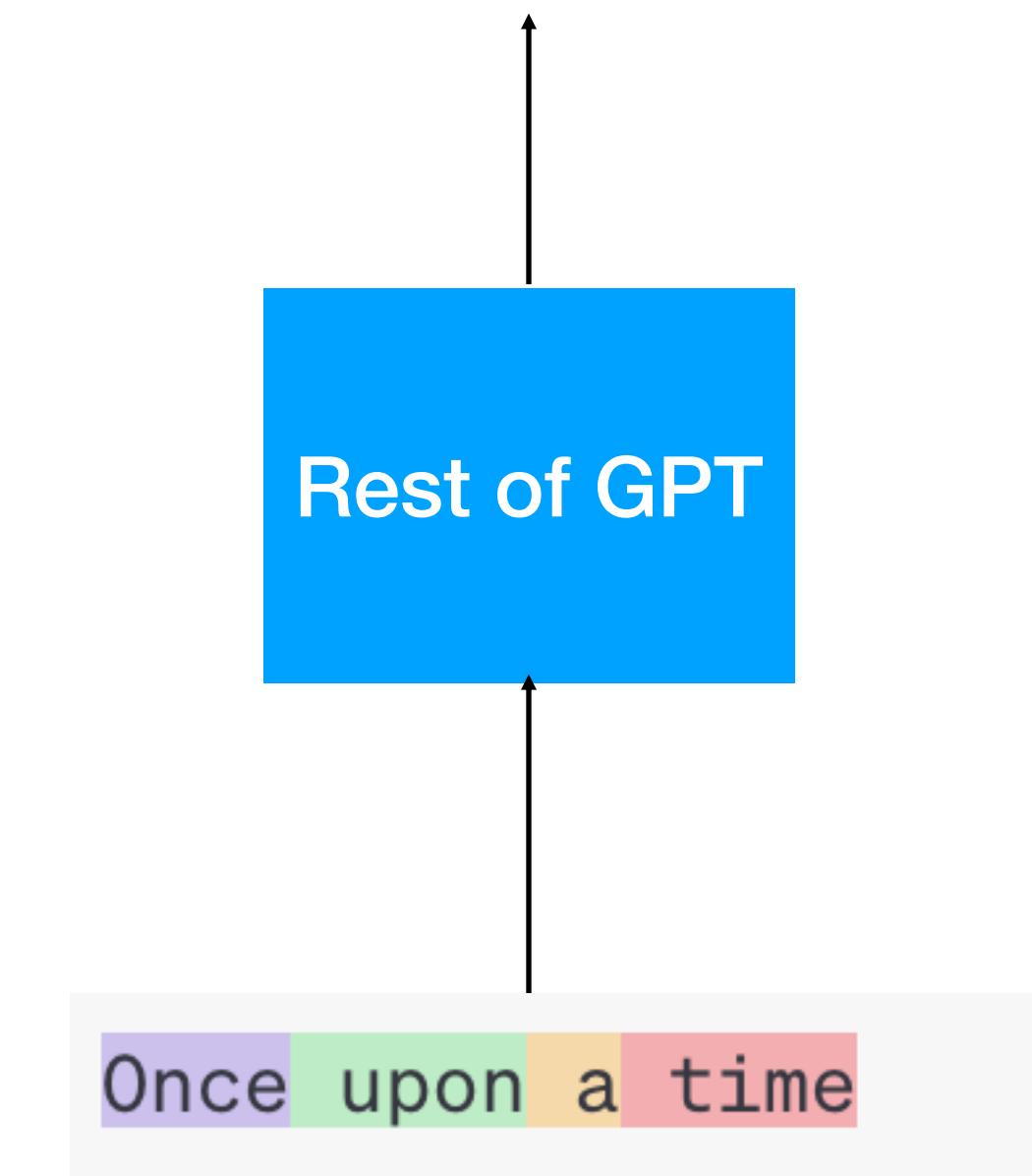


Once upon a time

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers



The text is converted into  
token vectors -

[12805, 5304, 264, 892]

Once upon a time

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers



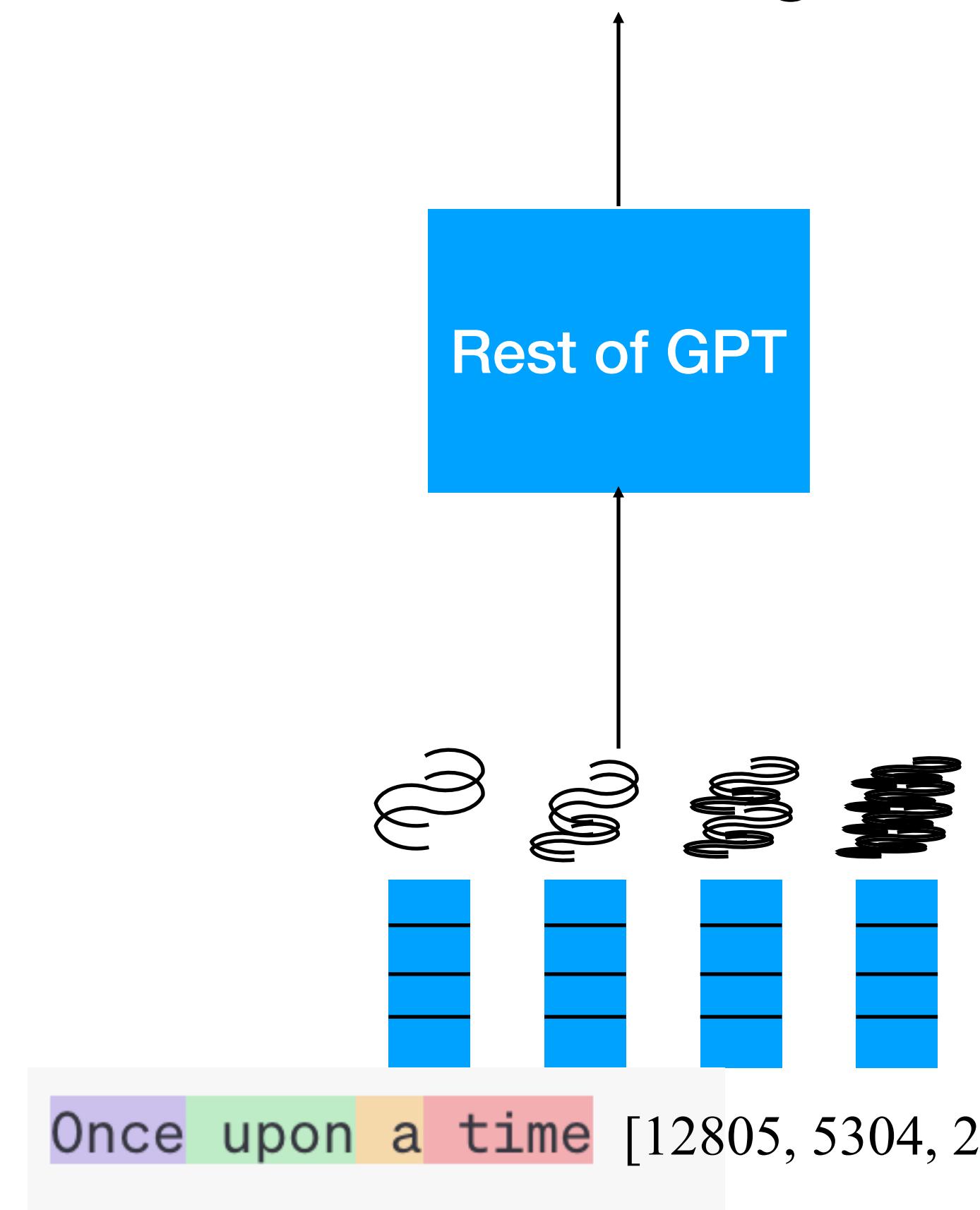
Sequence of integers is mapped to a set of embedding vectors by an embedding matrix  $W_E$

Once upon a time [12805, 5304, 264, 892]

First we will try to understand GPT

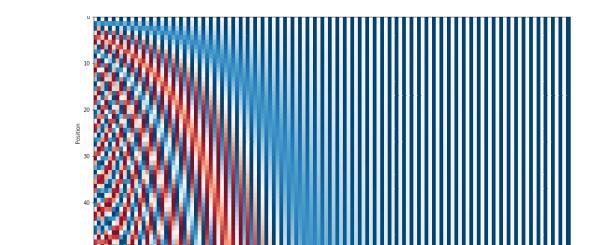
We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers



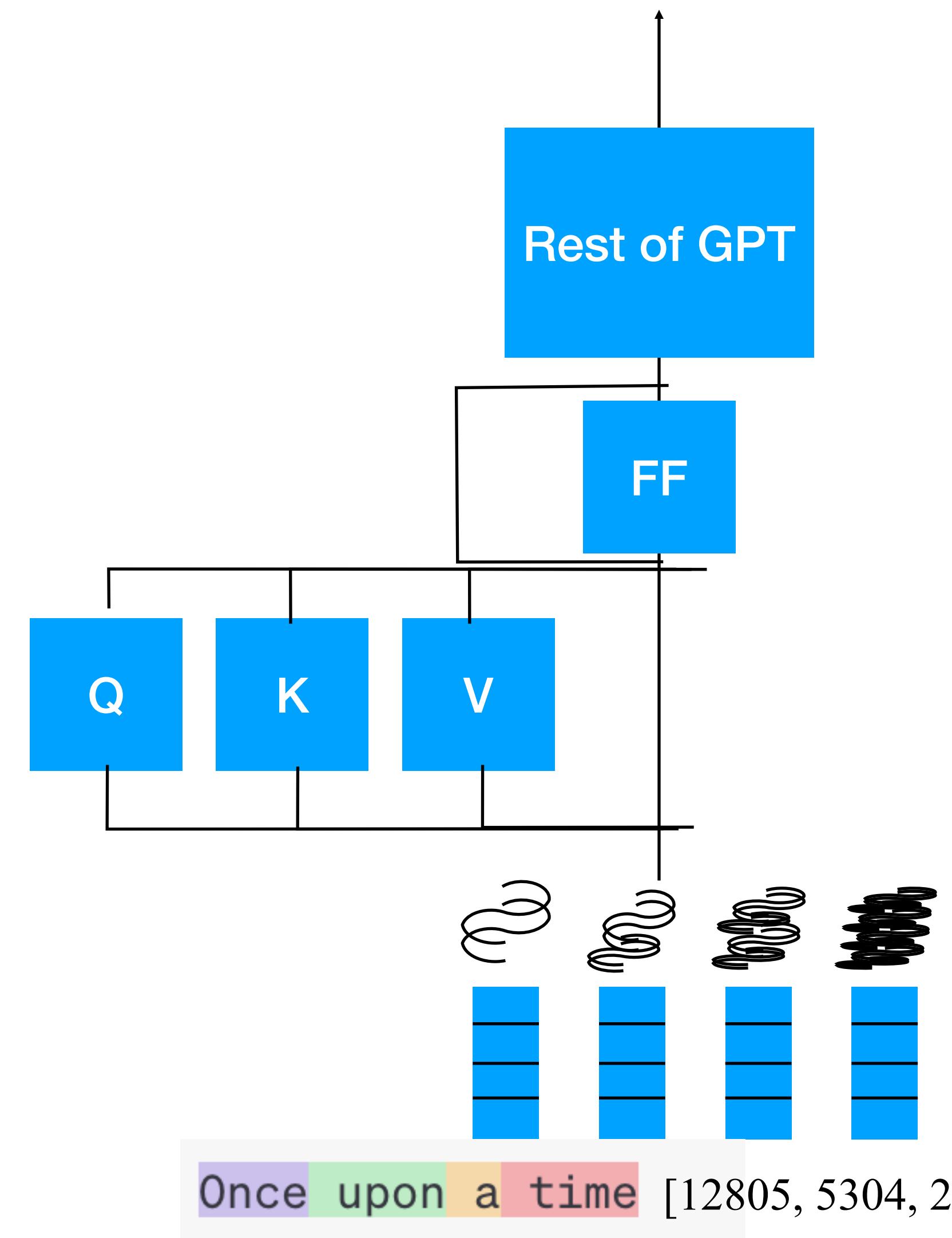
Positional encoding is added to the embedding -

$$\psi'(w_t) = \psi(w_t) + \vec{p}_t$$



First we will try to understand GPT

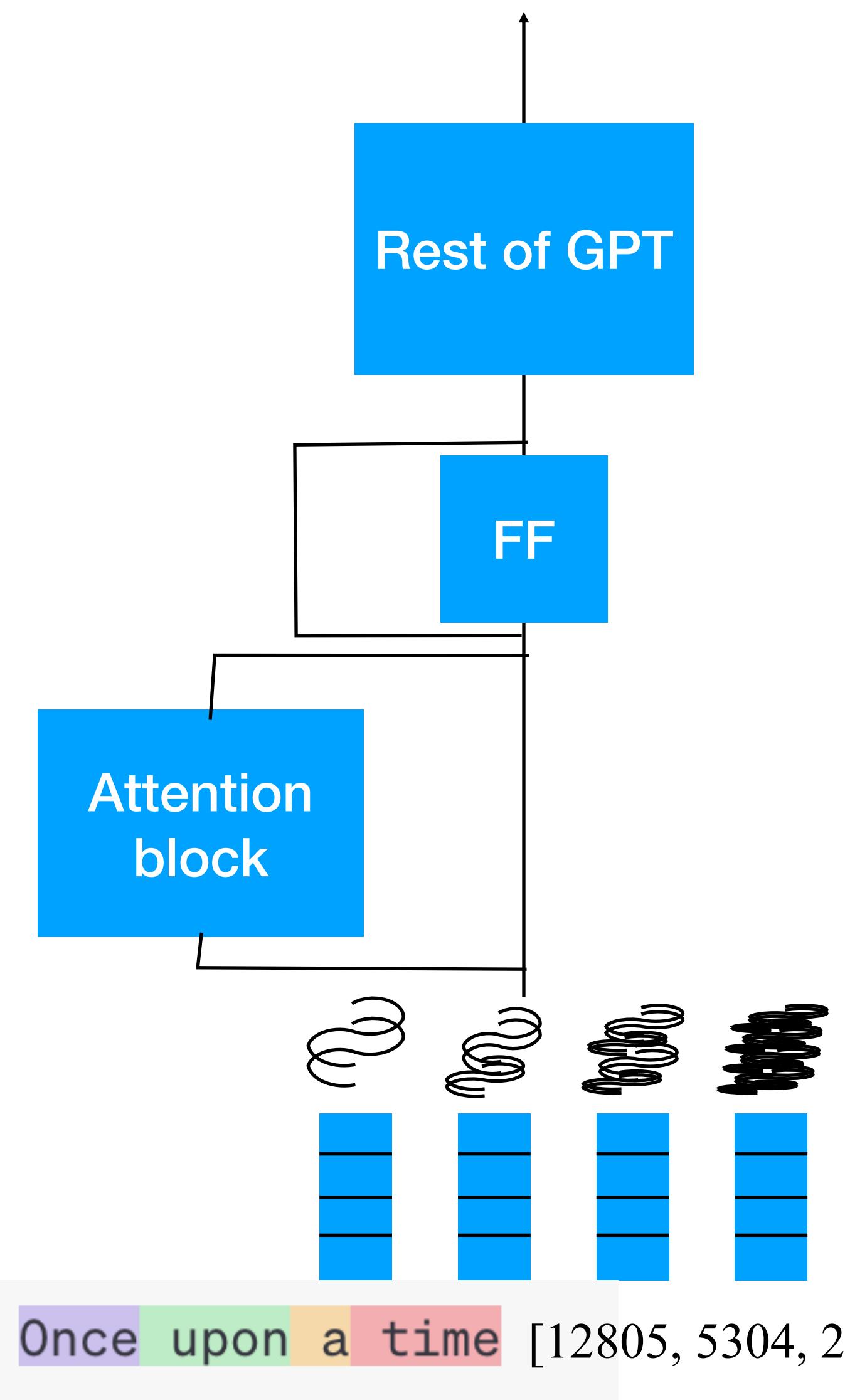
We want to understand what happens when we provide a prompt to GPT



This residual stream is then passed through a ‘self attention + FF layer block

First we will try to understand GPT

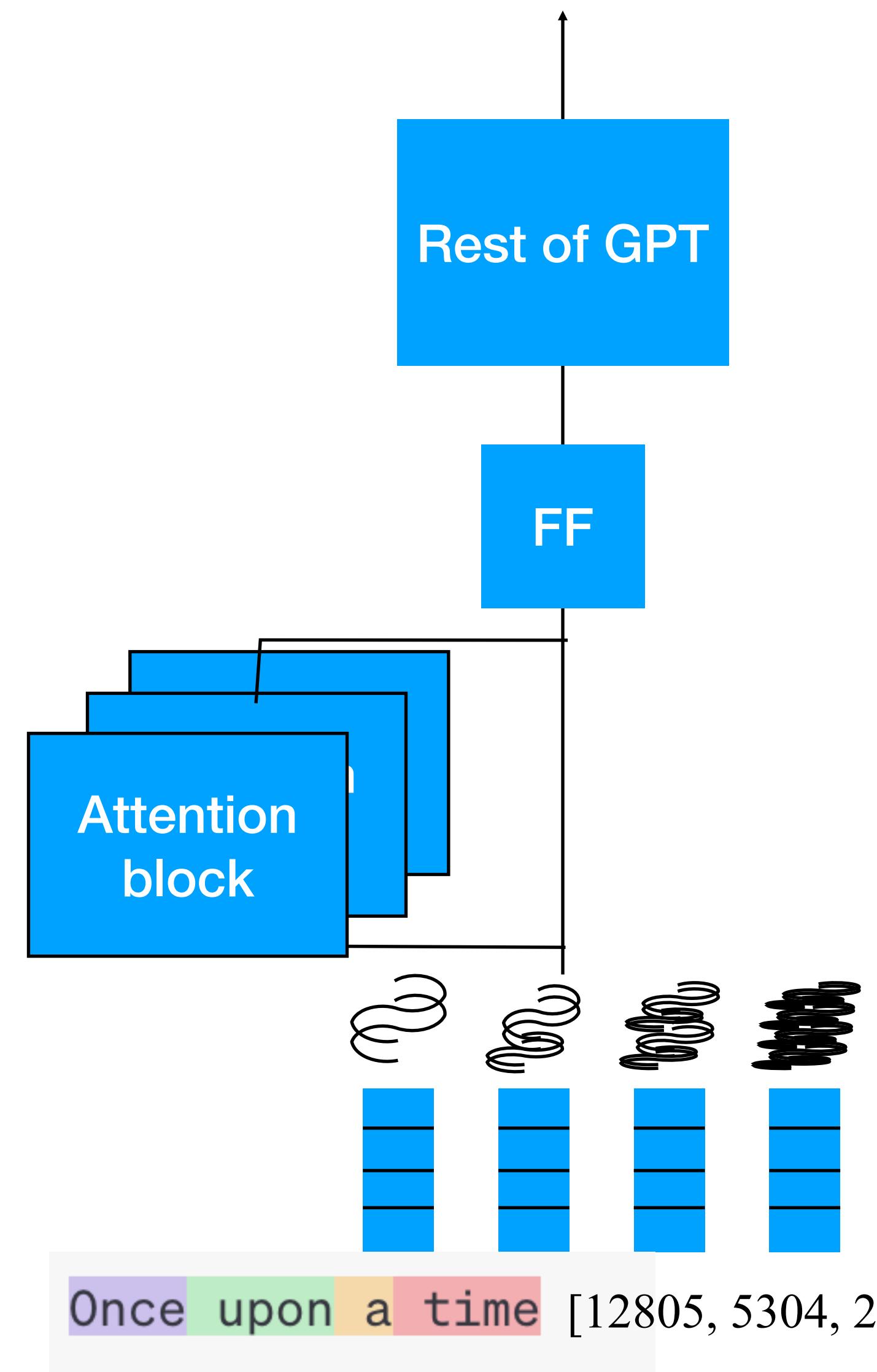
We want to understand what happens when we provide a prompt to GPT



This residual stream is then passed through a ‘self attention + FF layer block’

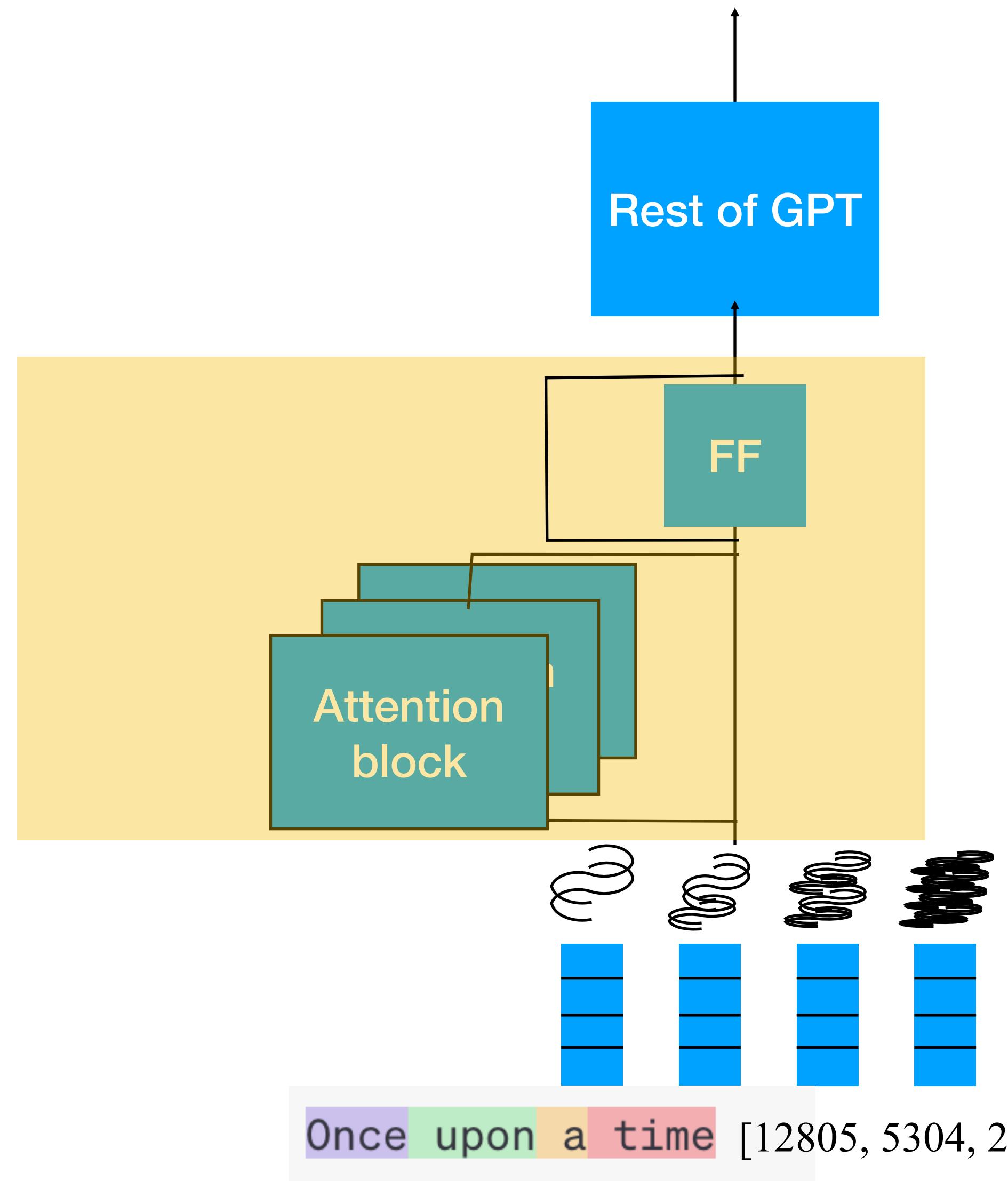
First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT



First we will try to understand GPT

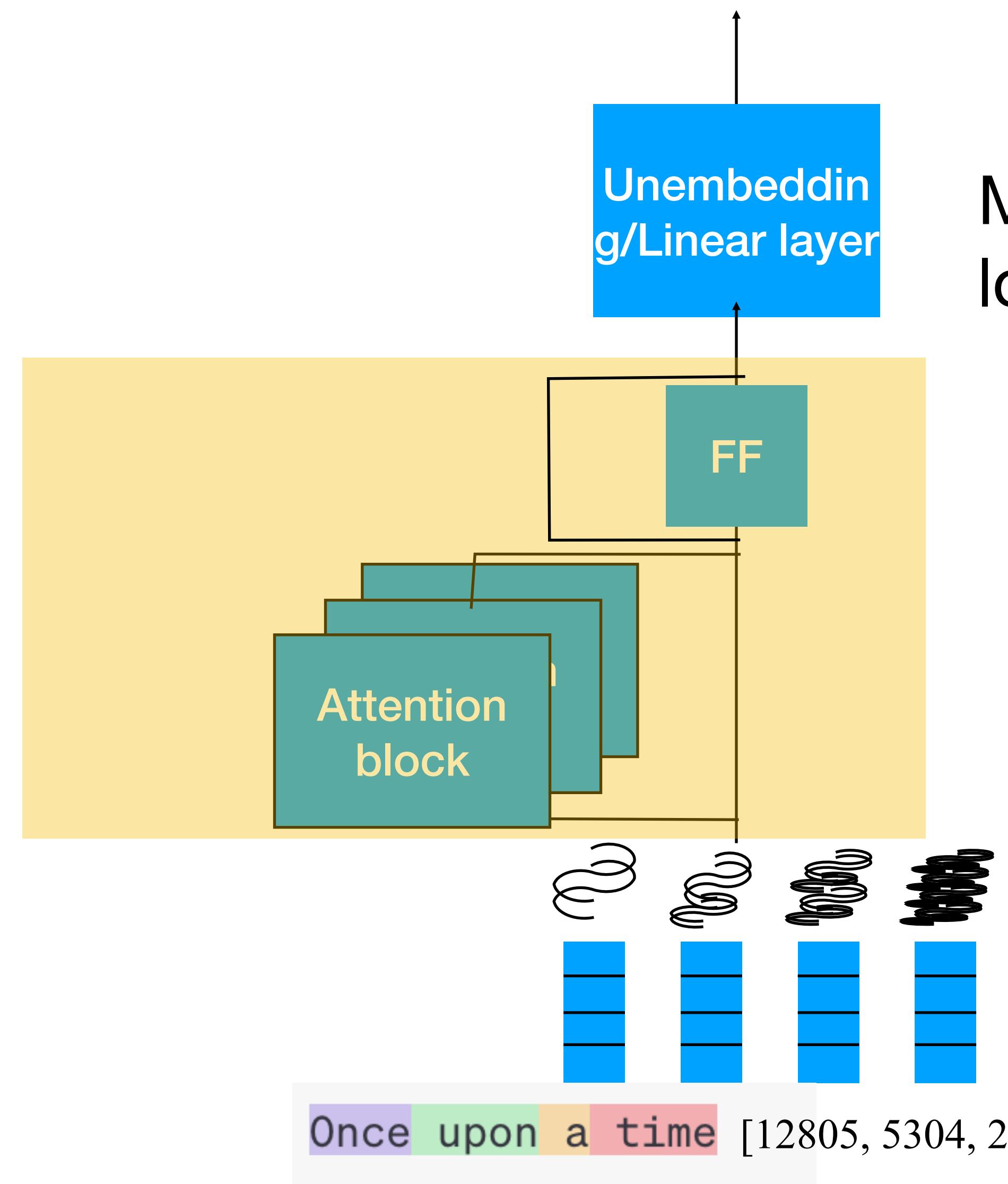
We want to understand what happens when we provide a prompt to GPT



This is one layer of the transformer - this structure is repeated many times

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT



Maps from embedding space to logits for the token space

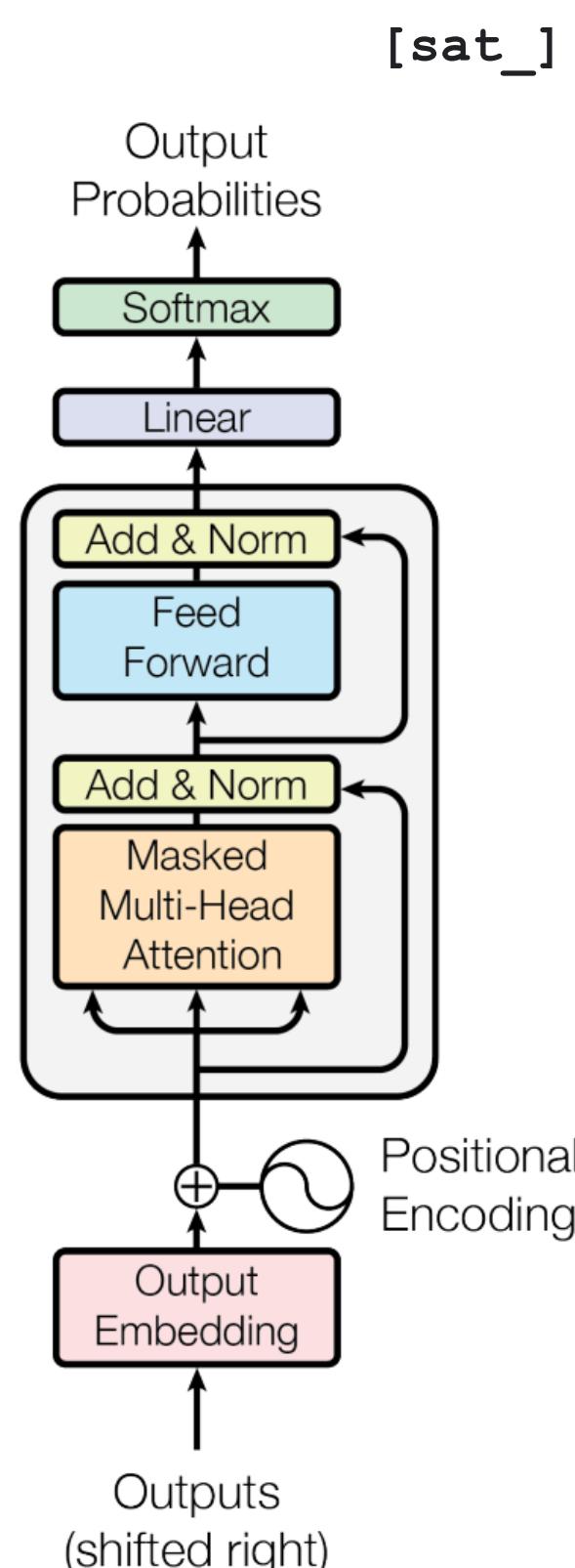
Many layers of this structure

# Transformer architectures

# NLP architectures

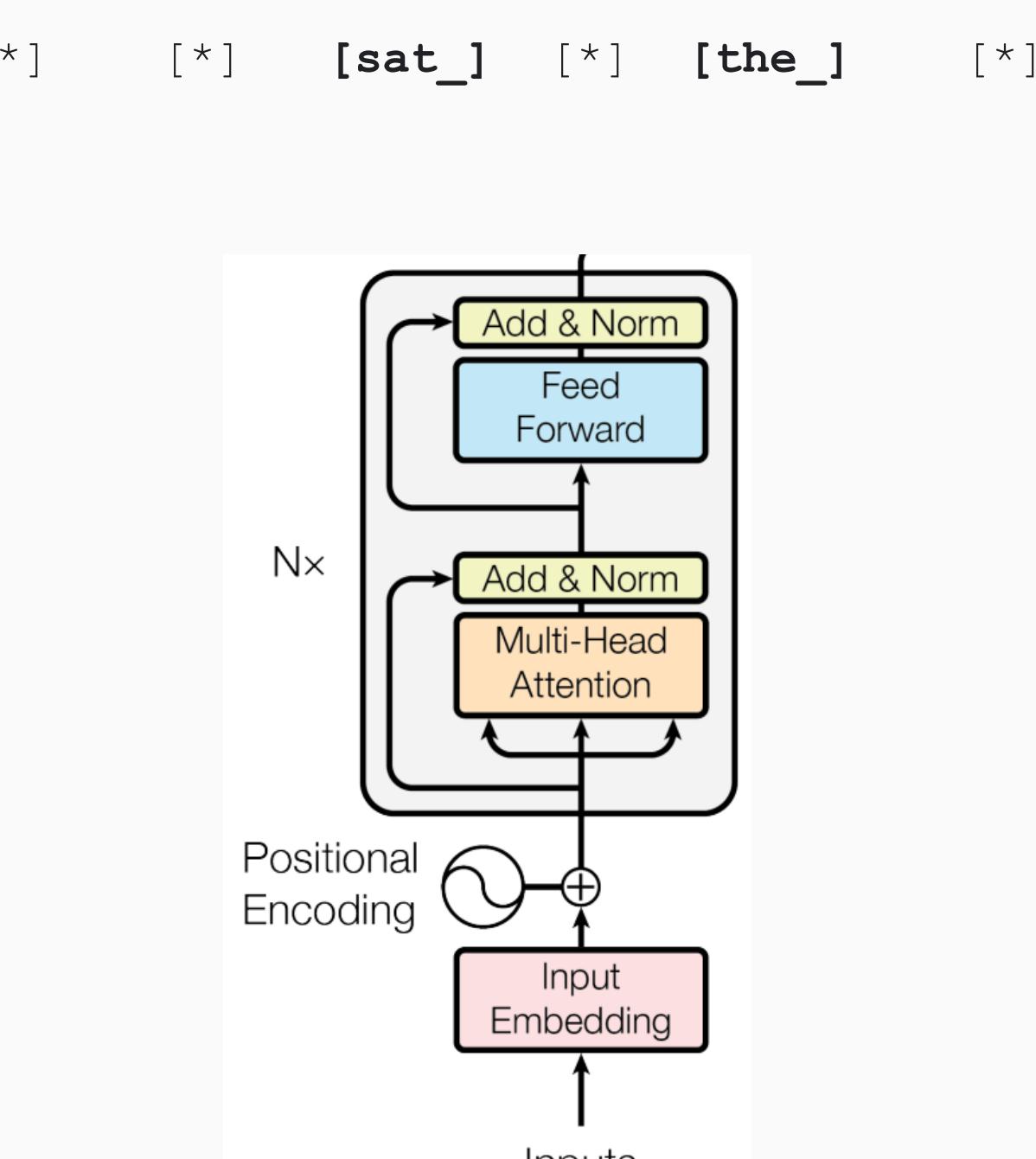
# Decoder-only

## GPT



# Encoder-only

## BERT



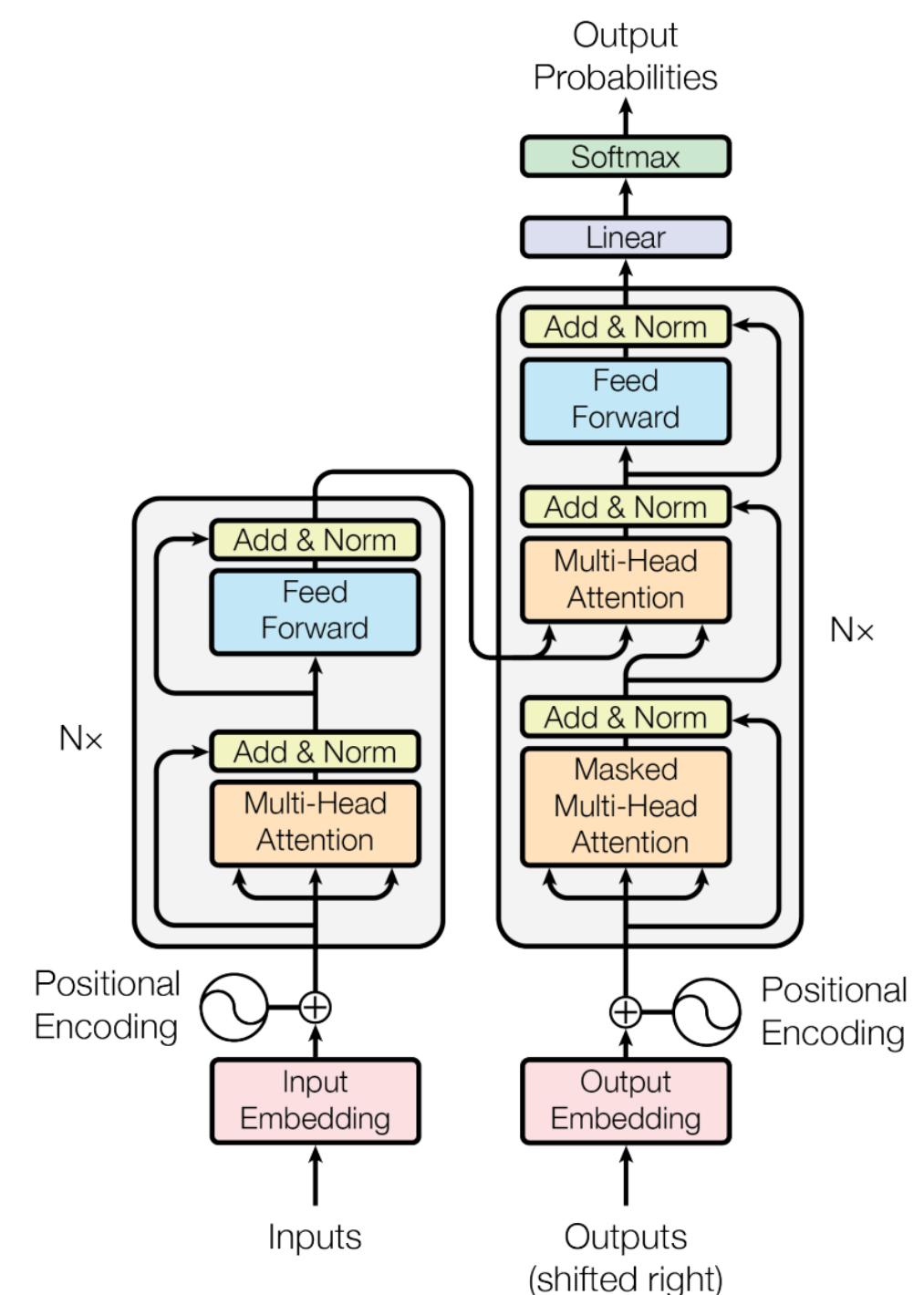
[START] [The\_] [cat\_]

[The\_] [cat\_] [MASK] [on\_] [MASK] [mat\_]

# Enc-Dec

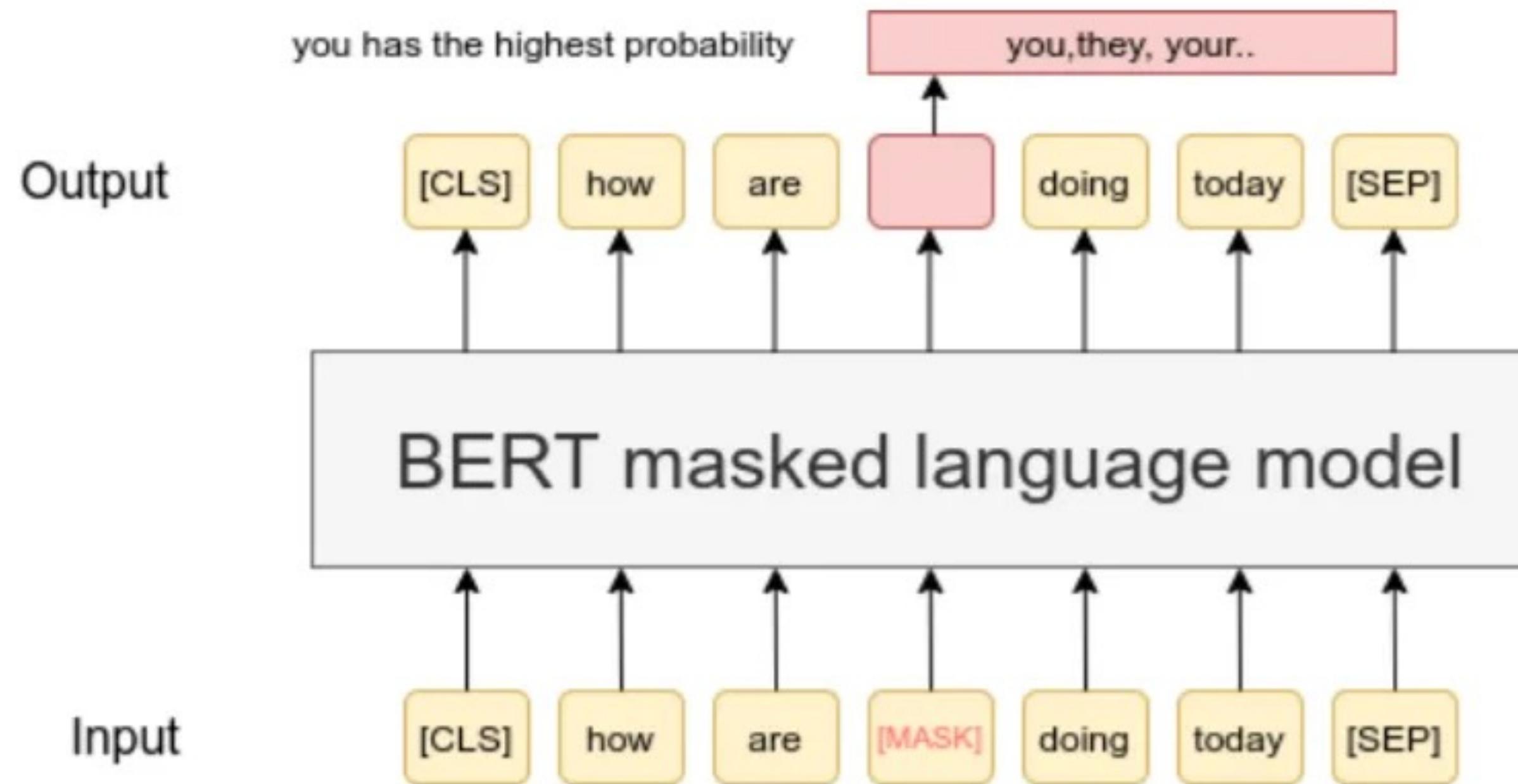
## T5/M2M

Das ist gut.  
A storm in Attala caused 6 victims.  
This is not toxic.

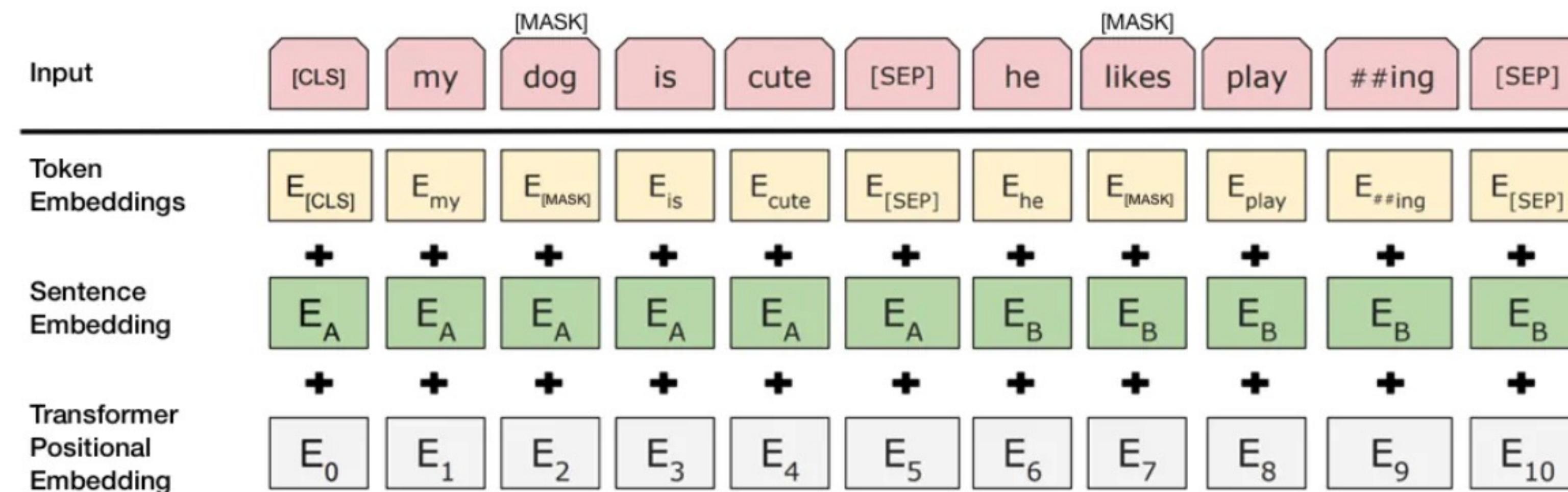


Translate EN-DE: This is good.  
Summarize: state authorities dispatched...  
Is this toxic: You look beautiful today!

# BERT - Pretraining Task MLM



# BERT - Pretraining next sentence prediction



# BERT - usage for various tasks

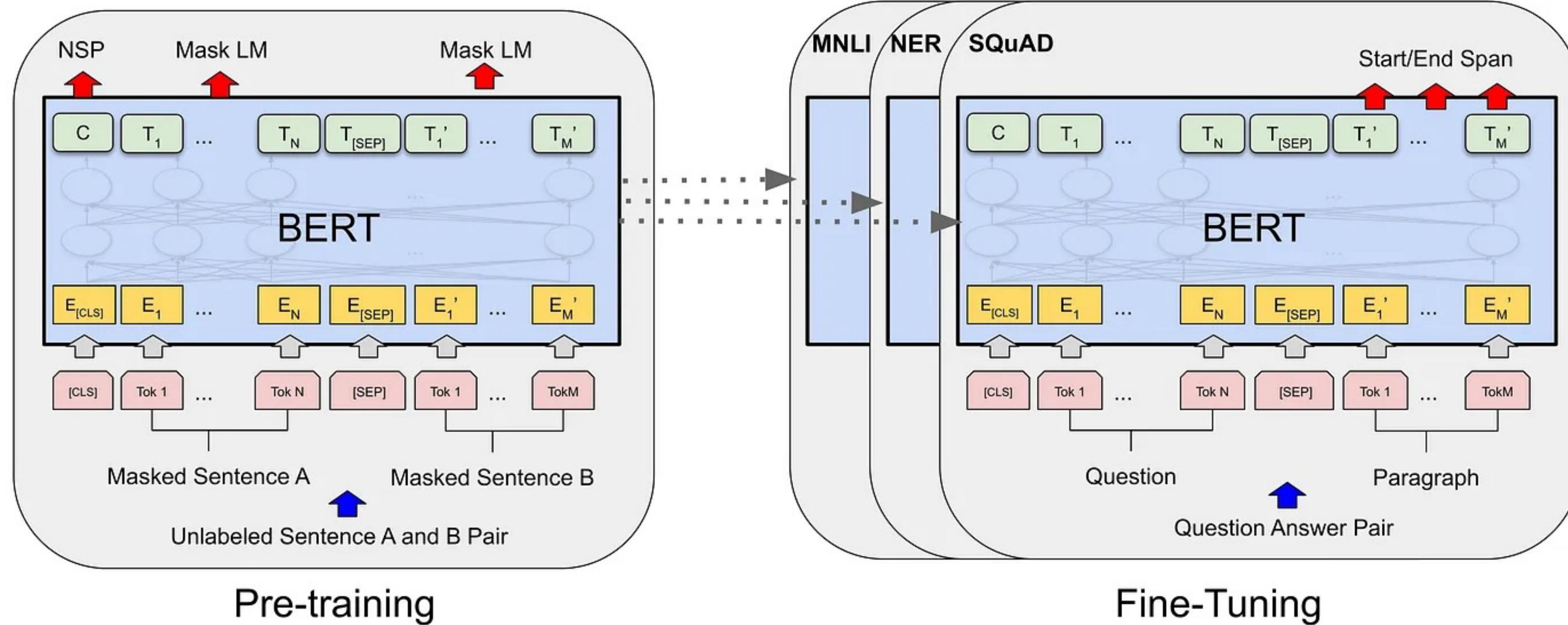


Fig from BERT paper - BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

# GPT-3 Language Models are Few Shot Learners

Model Name	$n_{\text{params}}$	$n_{\text{layers}}$	$d_{\text{model}}$	$n_{\text{heads}}$	$d_{\text{head}}$	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	$6.0 \times 10^{-4}$
GPT-3 Medium	350M	24	1024	16	64	0.5M	$3.0 \times 10^{-4}$
GPT-3 Large	760M	24	1536	16	96	0.5M	$2.5 \times 10^{-4}$
GPT-3 XL	1.3B	24	2048	24	128	1M	$2.0 \times 10^{-4}$
GPT-3 2.7B	2.7B	32	2560	32	80	1M	$1.6 \times 10^{-4}$
GPT-3 6.7B	6.7B	32	4096	32	128	2M	$1.2 \times 10^{-4}$
GPT-3 13B	13.0B	40	5140	40	128	2M	$1.0 \times 10^{-4}$
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	$0.6 \times 10^{-4}$

# GPT-3 Language Models are Few Shot Learners

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

# GPT-3 Language Models are Few Shot Learners

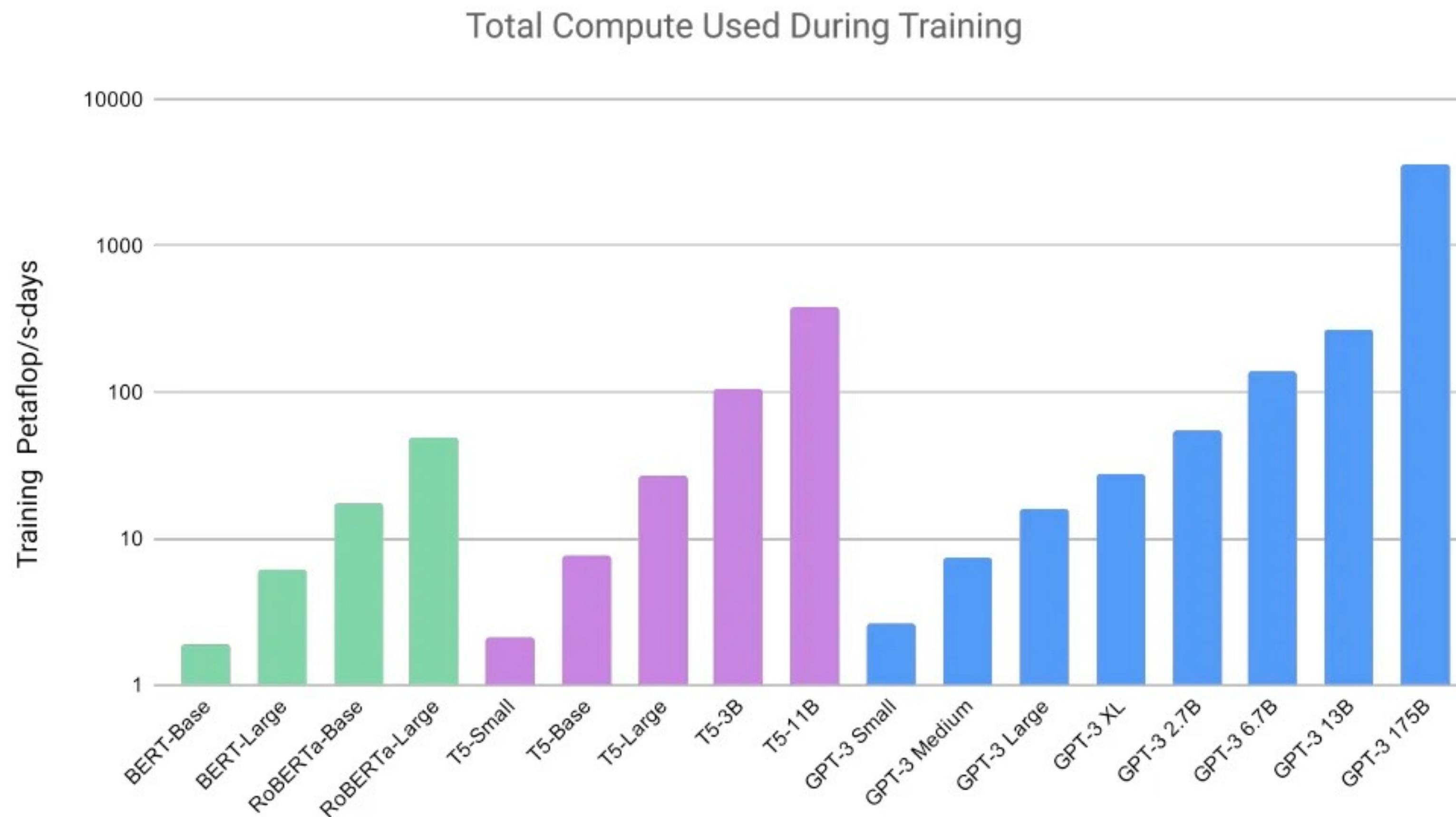


Fig from GPT3 paper - Language Models are Few Shot Learners

# GPT-3 Language Models are Few Shot Learners

The three settings we explore for in-context learning

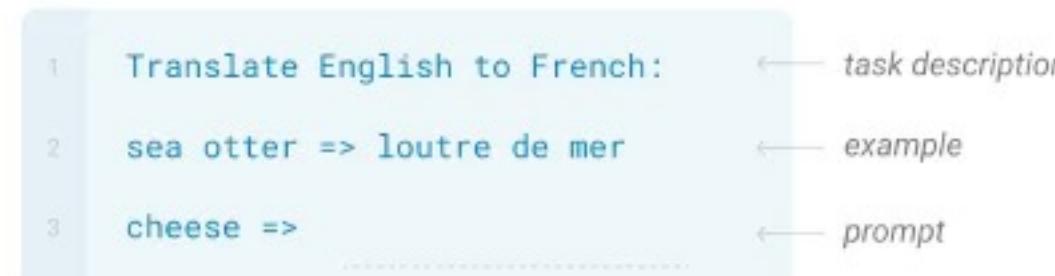
## Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



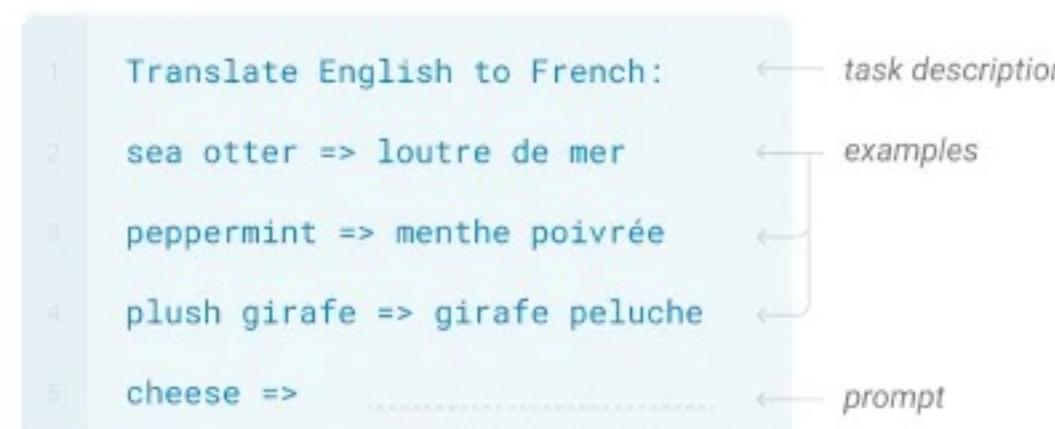
## One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.



## Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Traditional fine-tuning (not used for GPT-3)

## Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

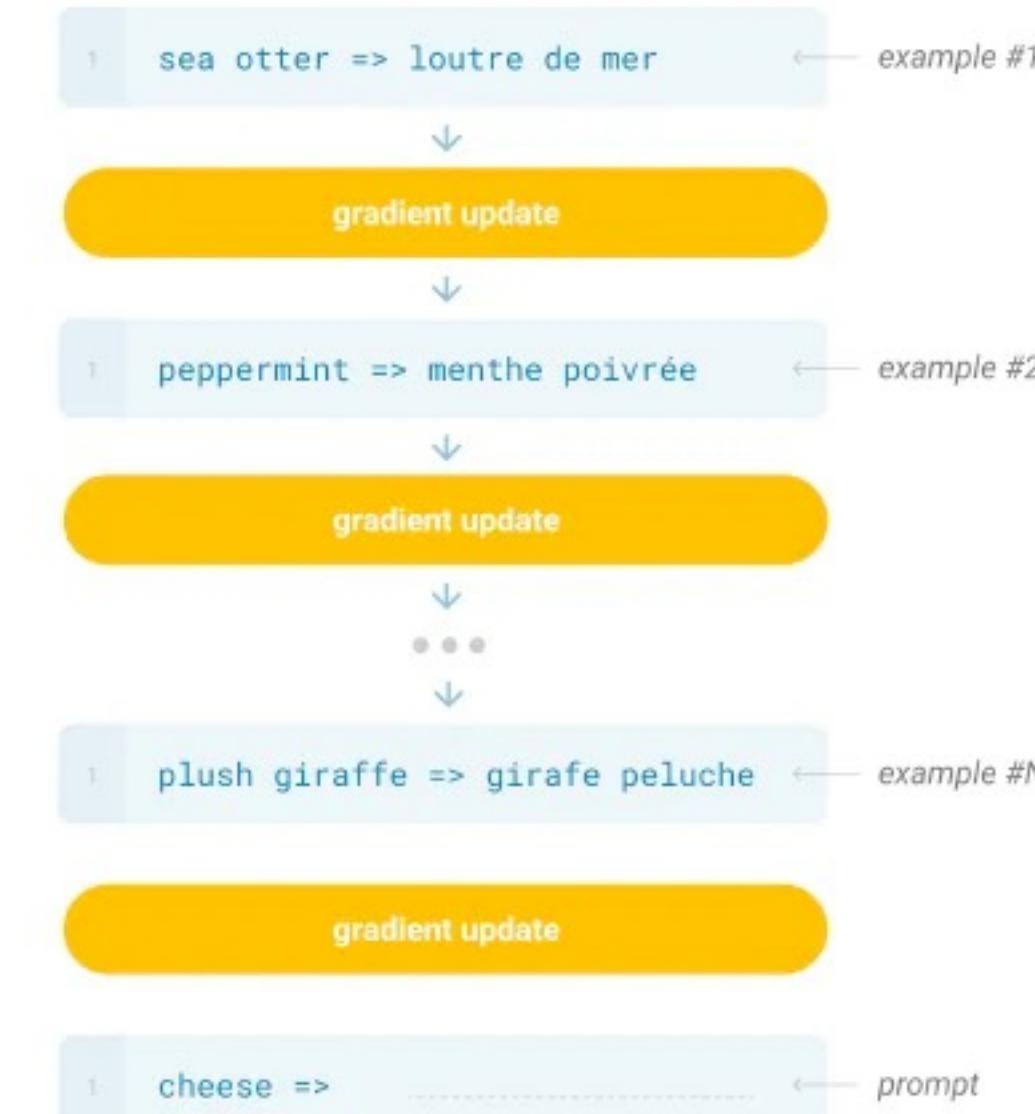


Fig from GPT3 paper - Language Models are Few Shot Learners

# GPT-3 Language Models are Few Shot Learners

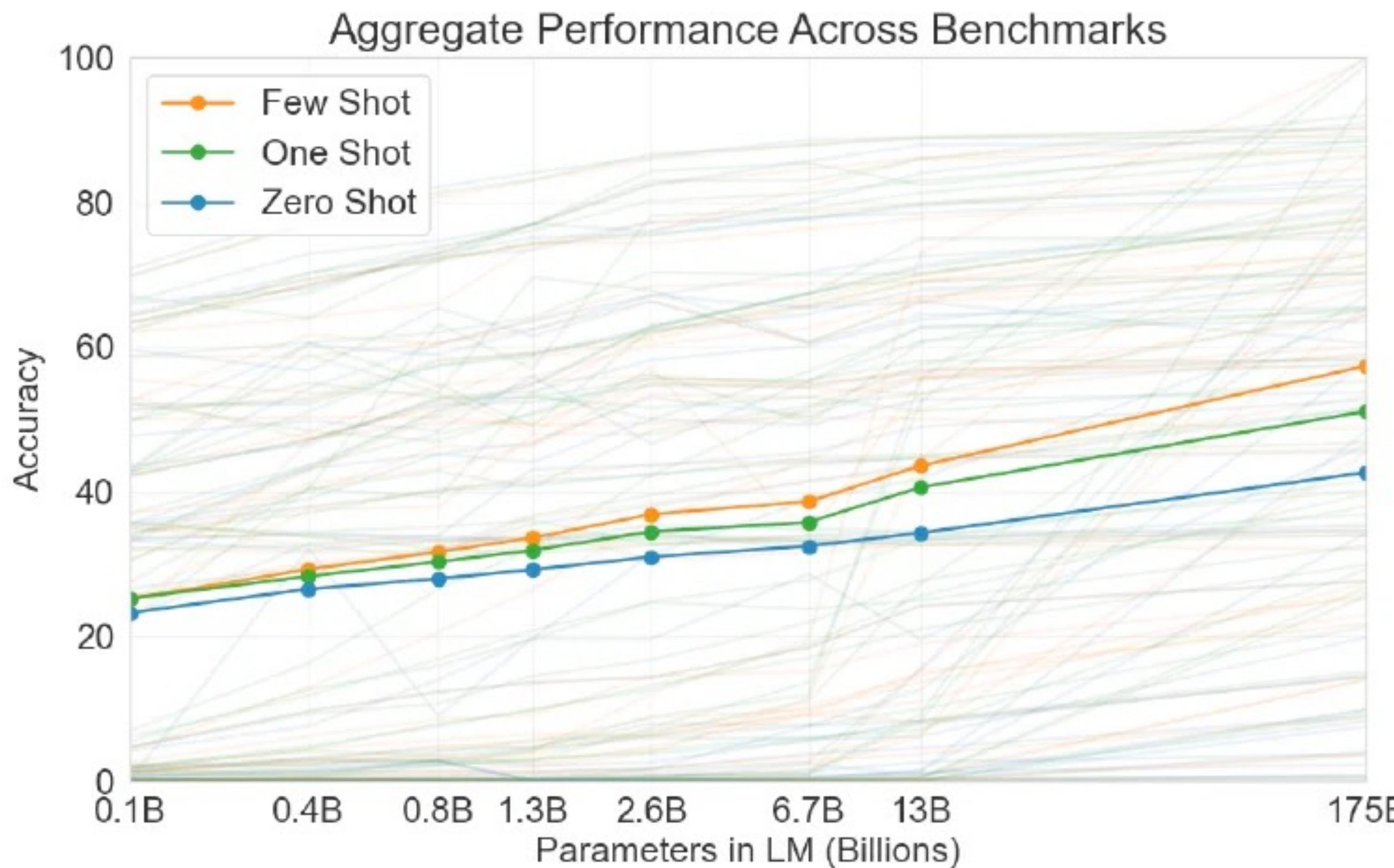
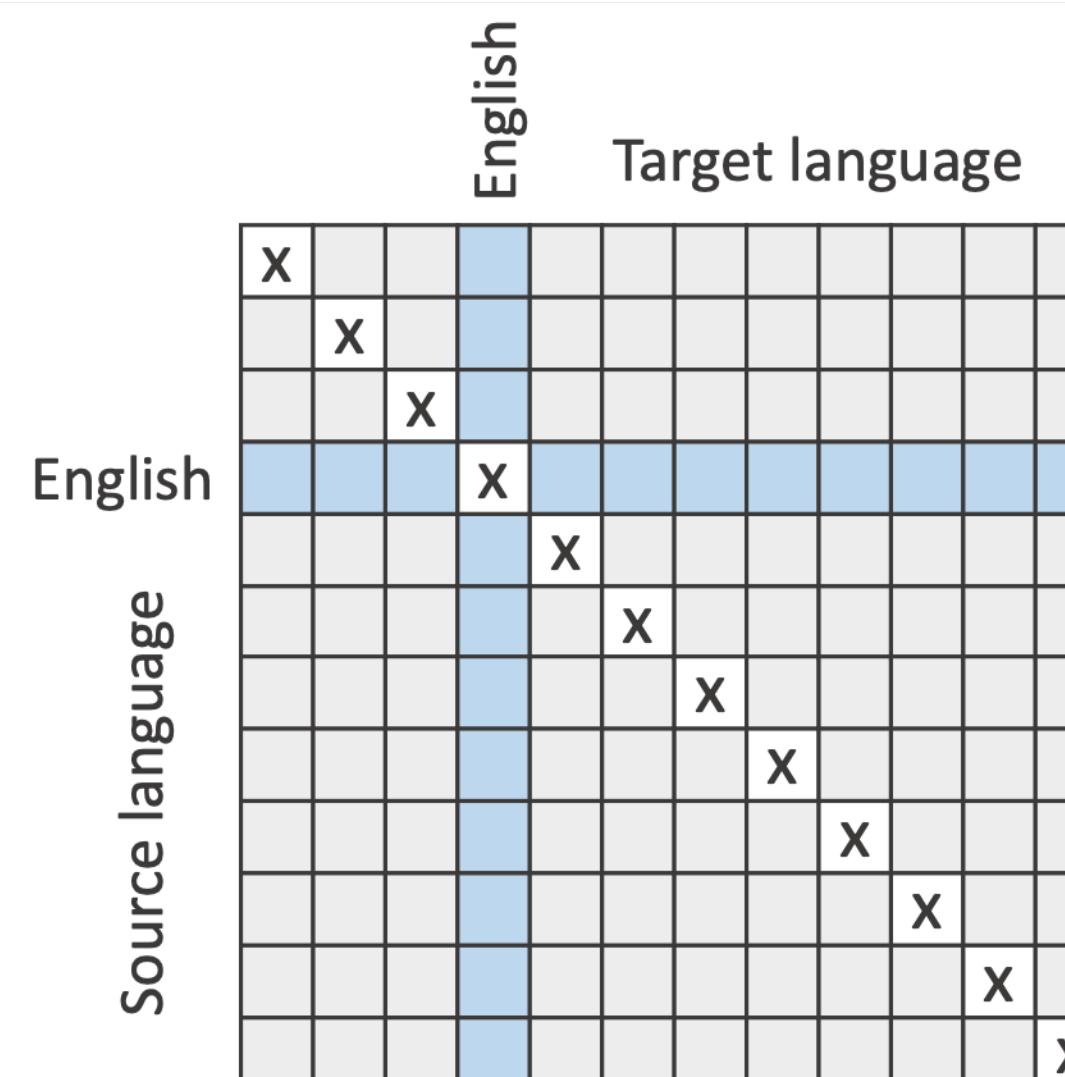
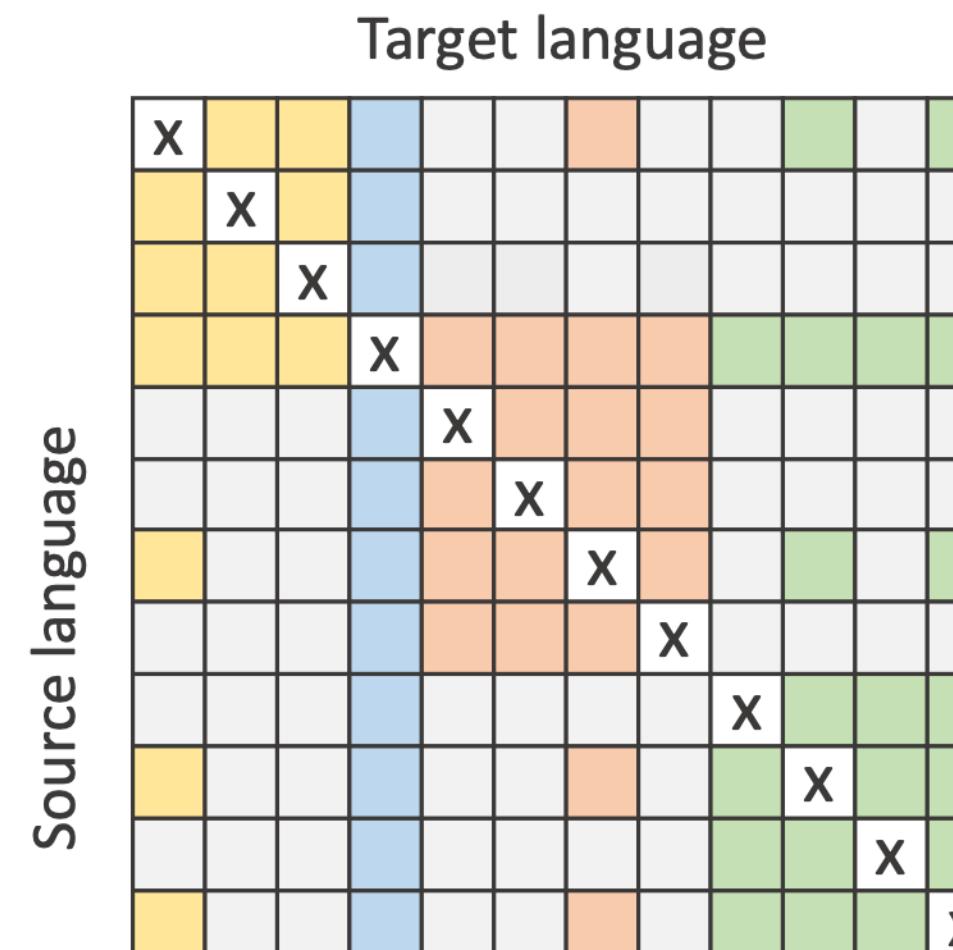


Fig from GPT3 paper - Language Models are Few Shot Learners

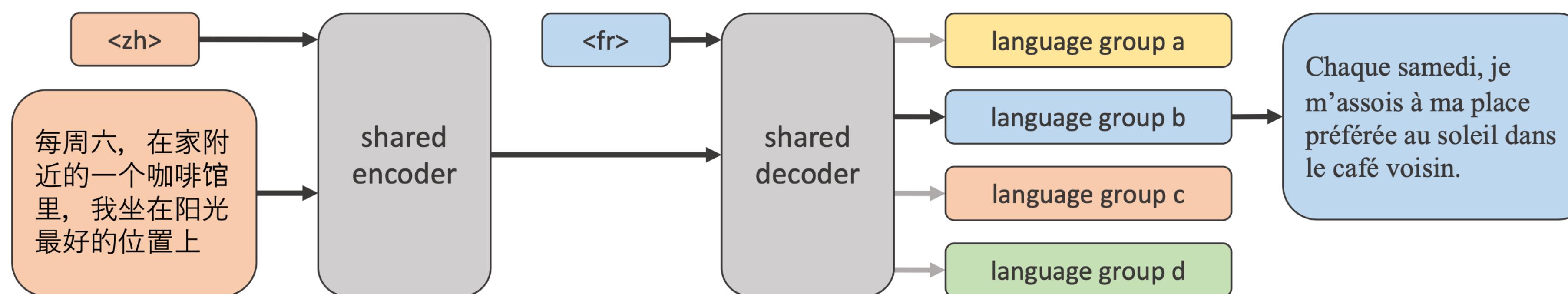
# M2M100



(a) English-Centric Multilingual



(b) M2M-100: Many-to-Many Multilingual Model



(c) Translating from Chinese to French with Dense + Language-Specific Sparse Model

ISO Language	Family	Script	ISO Language	Family	Script
af Afrikaans	Germanic	Latin	ja Japanese	Japonic	Kanji; Kana
da Danish	Germanic	Latin	ko Korean	Koreanic	Hangul
nl Dutch	Germanic	Latin	vi Vietnamese	Vietic	Latin
de German	Germanic	Latin	zh Chinese Mandarin	Chinese	Chinese
en English	Germanic	Latin	bn Bengali	Indo-Aryan	Eastern-Nagari
is Icelandic	Germanic	Latin	gu Gujarati	Indo-Aryan	Gujarati
lb Luxembourgish	Germanic	Latin	hi Hindi	Indo-Aryan	Devanagari
no Norwegian	Germanic	Latin	kn Kannada	Tamil	Kannada
sv Swedish	Germanic	Latin	mr Marathi	Indo-Aryan	Devanagari
Germanic	Latin	ne Nepali	Indo-Aryan	Devanagari	
yi Yiddish	Germanic	Hebrew	or Oriya	Indo-Aryan	Odia
ast Asturian	Romance	Latin	pa Panjabi	Indo-Aryan	Gurmukhi
ca Catalan	Romance	Latin	sd Sindhi	Indo-Aryan	Persian
fr French	Romance	Latin	si Sinhala	Indo-Aryan	Sinhala
gl Galician	Romance	Latin	ur Urdu	Indo-Aryan	Arabic
it Italian	Romance	Latin	ta Tamil	Dravidian	Tamil
oc Occitan	Romance	Latin	ceb Cebuano	Malayo-Polyn.	Latin
pt Portuguese	Romance	Latin	ilo Iloko	Philippine	Latin
ro Romanian	Romance	Latin	id Indonesian	Malayo-Polyn.	Latin
es Spanish	Romance	Latin	jv Javanese	Malayo-Polyn.	Latin
be Belarusian	Slavic	Cyrillic	mg Malagasy	Malayo-Polyn.	Latin
bs Bosnian	Slavic	Latin	ms Malay	Malayo-Polyn.	Latin
bg Bulgarian	Slavic	Cyrillic	ml Malayalam	Dravidian	Malayalam
hr Croatian	Slavic	Latin	su Sundanese	Malayo-Polyn.	Latin
cs Czech	Slavic	Latin	tl Tagalog	Malayo-Polyn.	Latin
mk Macedonian	Slavic	Cyrillic	my Burmese	Sino-Tibetan	Burmese
pl Polish	Slavic	Latin	km Central Khmer	Khmer	Khmer
ru Russian	Slavic	Cyrillic	lo Lao	Kra-Dai	Thai; Lao
sr Serbian	Slavic	Cyrillic; Latin	th Thai	Kra-Dai	Thai
sk Slovak	Slavic	Latin	mn Mongolian	Mongolic	Cyrillic
sl Slovenian	Slavic	Latin	ar Arabic	Arabic	Arabic
uk Ukrainian	Slavic	Cyrillic	he Hebrew	Semitic	Hebrew
et Estonian	Uralic	Latin	ps Pashto	Iranian	Arabic
fi Finnish	Uralic	Latin	fa Farsi	Iranian	Arabic
hu Hungarian	Uralic	Latin	am Amharic	Ethopian	Ge'ez
lv Latvian	Baltic	Latin	ff Fulah	Niger-Congo	Latin
lt Lithuanian	Baltic	Latin	ha Hausa	Afro-Asiatic	Latin
sq Albanian	Albanian	Latin	ig Igbo	Niger-Congo	Latin
hy Armenian	Armenian	Armenian	ln Lingala	Niger-Congo	Latin
ka Georgian	Kartvelian	Georgian	lg Luganda	Niger-Congo	Latin
el Greek	Hellenic	Greek	nso Northern Sotho	Niger-Congo	Latin
br Breton	Celtic	Latin	so Somali	Cushitic	Latin
ga Irish	Irish	Latin	sw Swahili	Niger-Congo	Latin
gd Scottish Gaelic	Celtic	Latin	ss Swati	Niger-Congo	Latin
cy Welsh	Celtic	Latin-Welsch	tn Tswana	Niger-Congo	Latin
az Azerbaijani	Turkic	Latin; Cyrillic	wo Wolof	Niger-Congo	Latin
ba Bashkir	Turkic	Persian	xh Xhosa	Niger-Congo	Latin
kk Kazakh	Turkic	Cyrillic	yo Yoruba	Niger-Congo	Latin
tr Turkish	Turkic	Cyrillic	zu Zulu	Niger-Congo	Latin
uz Uzbek	Turkic	Latin; Cyrillic	ht Haitian Creole	Creole	Latin

Table 1: **100 Languages grouped by family.** For each language, we display the ISO code, language name, language family, and script. Languages in bold are *bridge languages* (*Malayo-Polyn.* stands for *Malayo-Polynesian*).

# M2M100

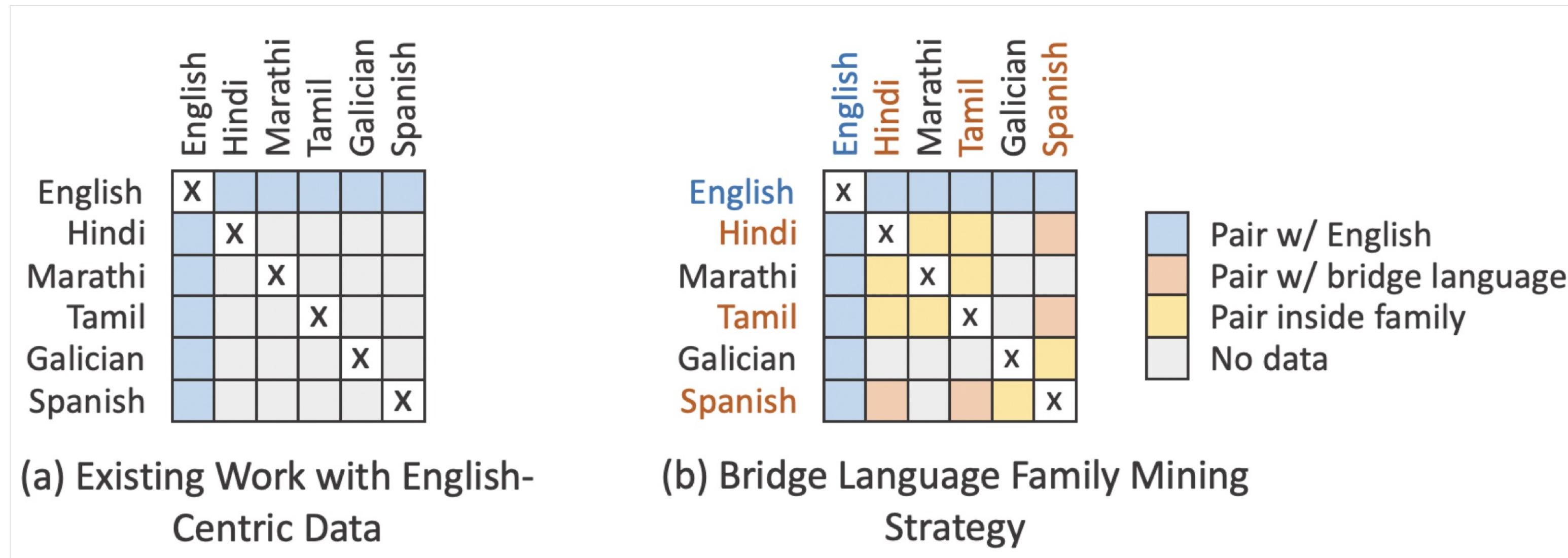


Figure 2: **Depiction of an English-Only data mining setting compared to the Bridge Language Mining Strategy.** We display a data matrix, where languages are shown on the X and Y axes. Data is mined in one direction (such as Hindi to Marathi) and used to train bidirectionally.

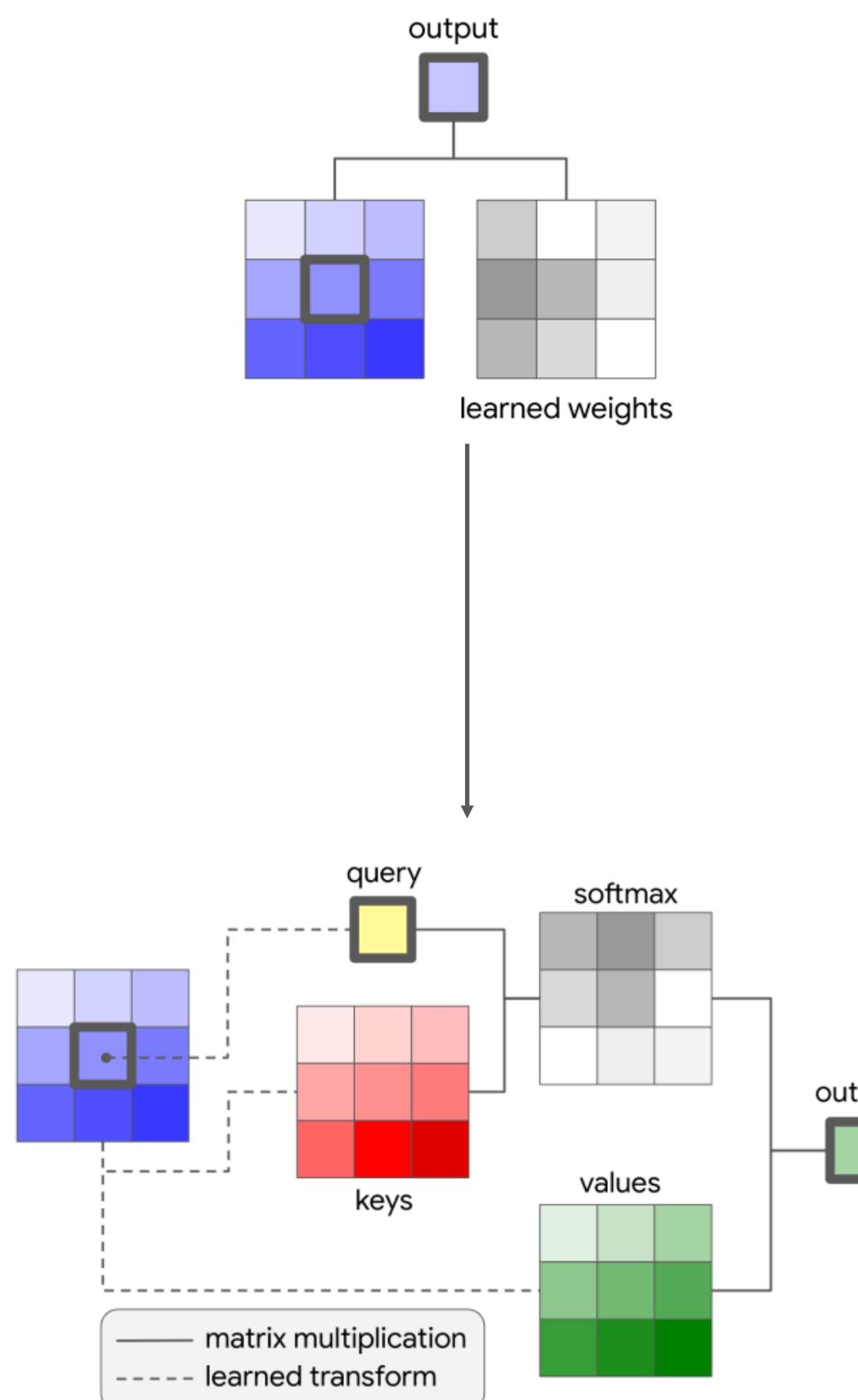
# Computer Vision

Many prior works attempted to introduce self-attention at the pixel level.

## Previous approaches

For  $224\text{px}^2$ , that's 50k sequence length, too much!

### 1. On pixels, but locally or factorized



Usually replaces 3x3 conv in ResNet:

stage	output	ResNet-50	<b>LR-Net-50 (<math>7 \times 7</math>, <math>m=8</math>)</b>
res1	$112 \times 112$	$7 \times 7$ conv, 64, stride 2	<b><math>1 \times 1</math>, 64 <math>7 \times 7</math> LR, 64, stride 2</b>
res2	$56 \times 56$	$3 \times 3$ max pool, stride 2 $\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3 \text{ conv}, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$3 \times 3$ max pool, stride 2 $\begin{bmatrix} 1 \times 1, 100 \\ 7 \times 7 \text{ LR, 100} \\ 1 \times 1, 256 \end{bmatrix} \times 3$
res3	$28 \times 28$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3 \text{ conv}, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 200 \\ 7 \times 7 \text{ LR, 200} \\ 1 \times 1, 512 \end{bmatrix} \times 4$
res4	$14 \times 14$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3 \text{ conv}, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 400 \\ 7 \times 7 \text{ LR, 400} \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
res5	$7 \times 7$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3 \text{ conv}, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 800 \\ 7 \times 7 \text{ LR, 800} \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	$1 \times 1$	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
	# params	<b><math>25.5 \times 10^6</math></b>	<b><math>23.3 \times 10^6</math></b>
	FLOPs	<b><math>4.3 \times 10^9</math></b>	<b><math>4.3 \times 10^9</math></b>

Results:

Are not great

Do not justify increased complexity

Do not justify slowdown over convolutions

Examples:

Non-local NN (Wang et.al. 2017)

SASANet (Stand-Alone Self-Attention in Vision Models)

HaloNet (Scaling Local Self-Attn for Parameter Efficient...)

LR-Net (Local Relation Networks for Image Recognition)

SANet (Exploring Self-attention for Image Recognition)

...

# An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

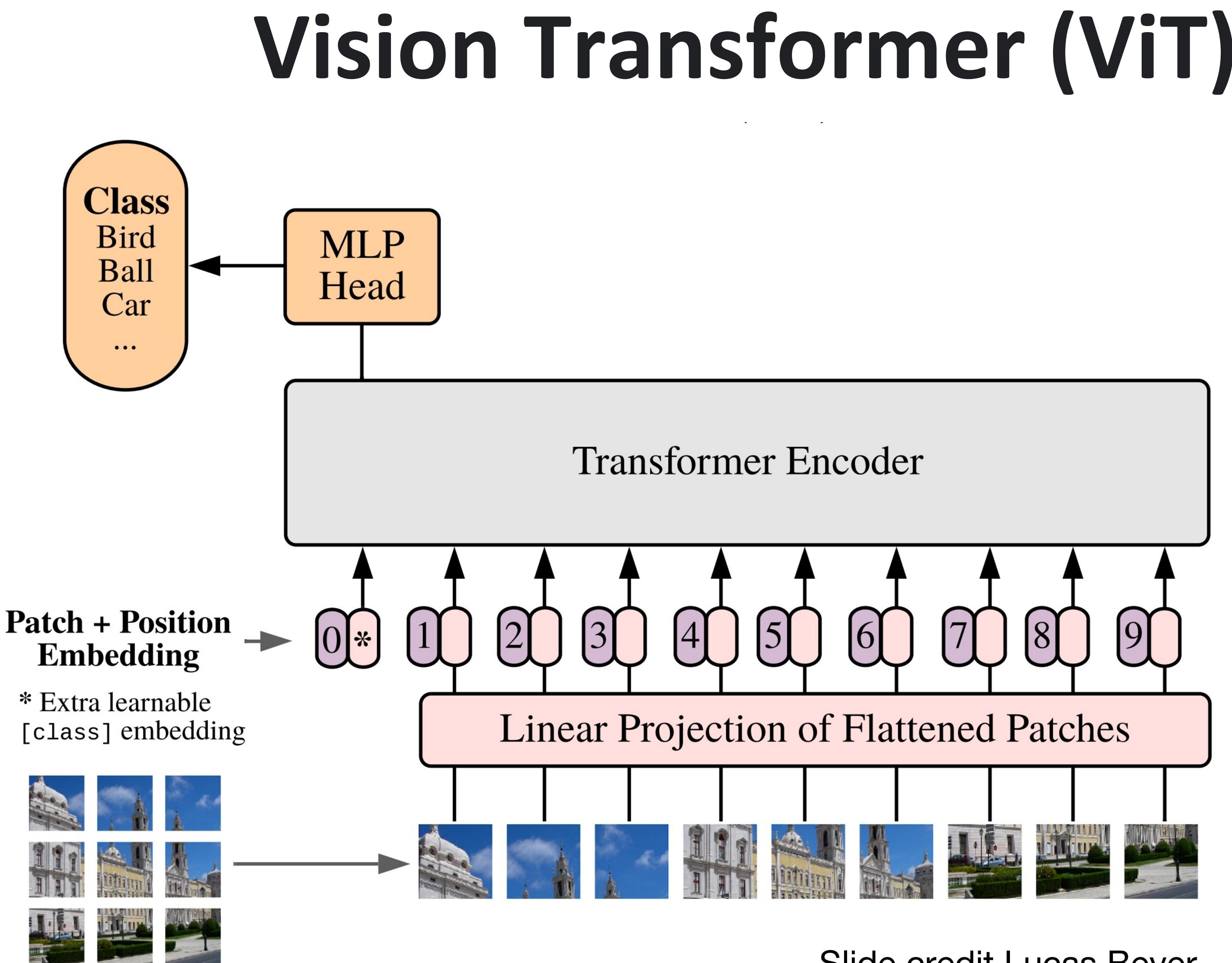
2020, A Dosovitskiy, L Beyer, A Kolesnikov, D Weissenborn, X Zhai, T Unterthiner, M Dehghani, M Minderer, G Heigold, S Gelly, J Uszkoreit, N Houlsby

Many prior works attempted to introduce self-attention at the pixel level.

For 224px<sup>2</sup>, that's 50k sequence length, too much!

Thus, most works restrict attention to local pixel neighborhoods, or as high-level mechanism on top of detections.

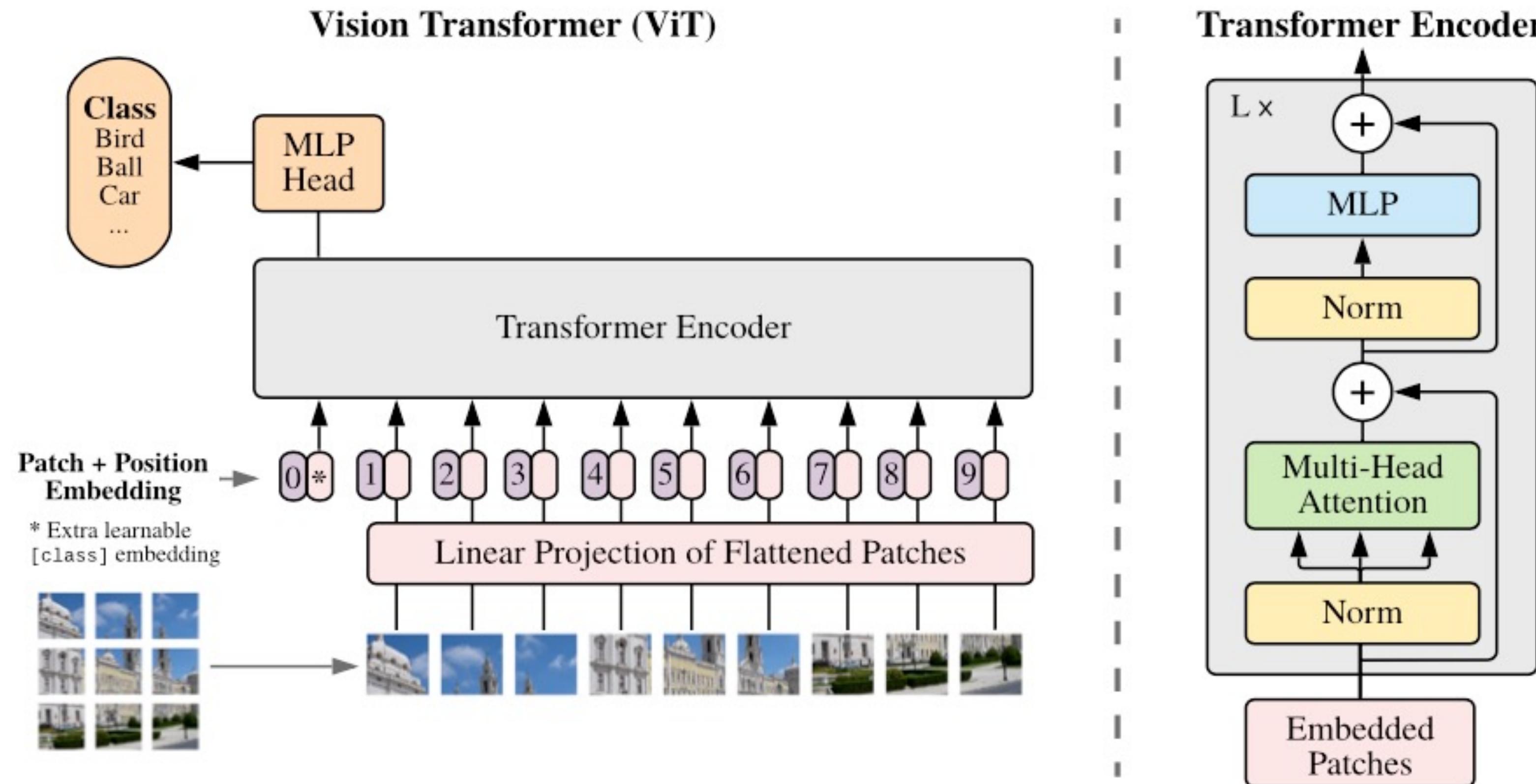
The **key breakthrough** in using the full Transformer architecture, standalone, was to "**tokenize**" the image by **cutting it into patches** of 16px<sup>2</sup>, and treating each patch as a token, e.g. embedding it into input space.



Slide credit Lucas Beyer

# An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

2020, A Dosovitskiy, L Beyer, A Kolesnikov, D Weissenborn, X Zhai, T Unterthiner, M Dehghani, M Minderer, G Heigold, S Gelly, J Uszkoreit, N Houlsby



# An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

2020, A Dosovitskiy, L Beyer, A Kolesnikov, D Weissenborn, X Zhai, T Unterthiner, M Dehghani, M Minderer, G Heigold, S Gelly, J Uszkoreit, N Houlsby

Model	Layers	Hidden size $D$	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

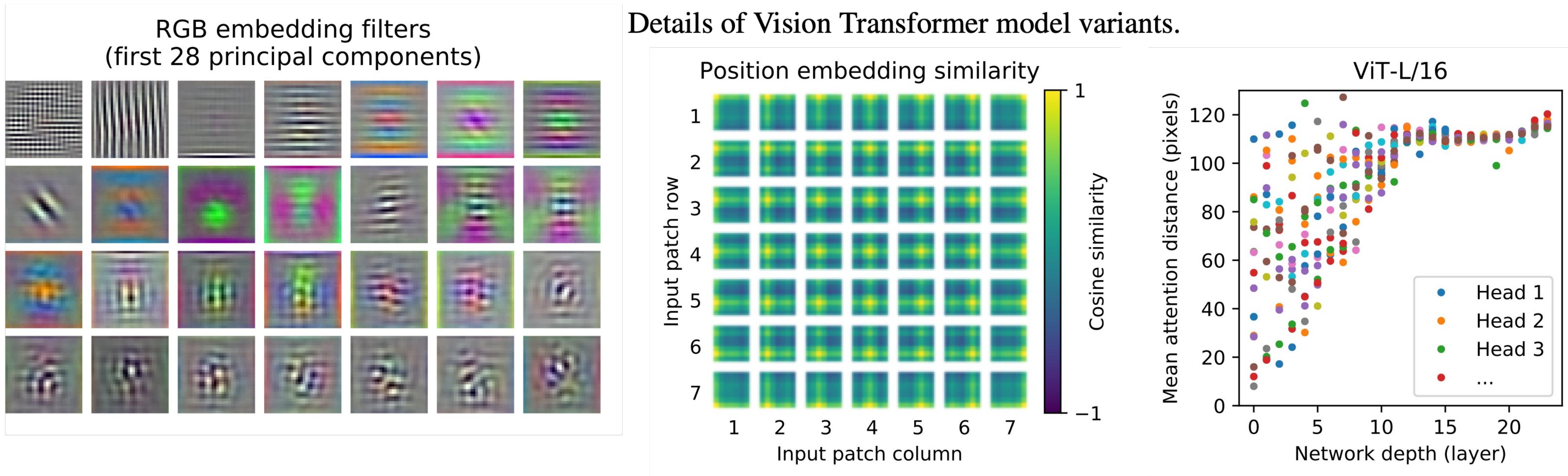


Figure 7: **Left:** Filters of the initial linear embedding of RGB values of ViT-L/32. **Center:** Similarity of position embeddings of ViT-L/32. Tiles show the cosine similarity between the position embedding of the patch with the indicated row and column and the position embeddings of all other patches. **Right:** Size of attended area by head and network depth. Each dot shows the mean attention distance across images for one of 16 heads at one layer. See Appendix D.7 for details.

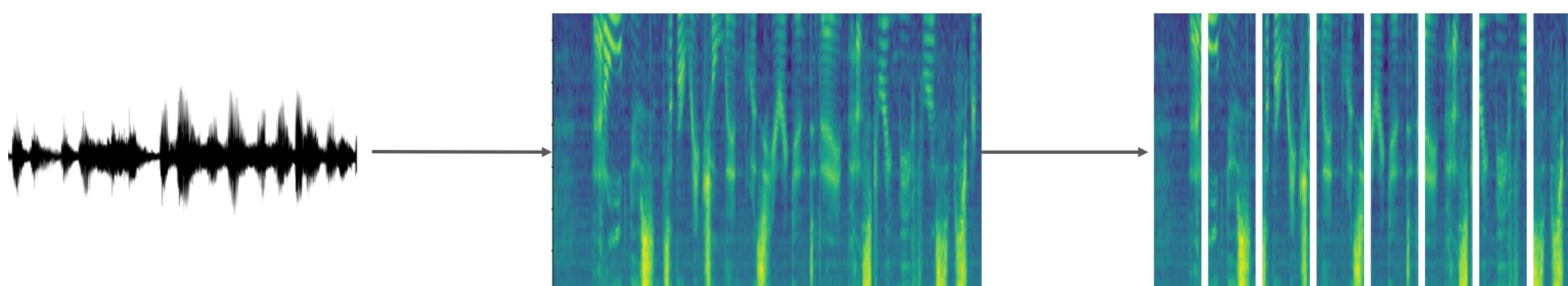
# Speech

# Conformer: Convolution-augmented Transformer for Speech Recognition

2020, A Gulati, J Qin, C-C Chiu, N Parmar, Y Zhang, J Yu, W Han, S Wang, Z Zhang, Y Wu, R Pang

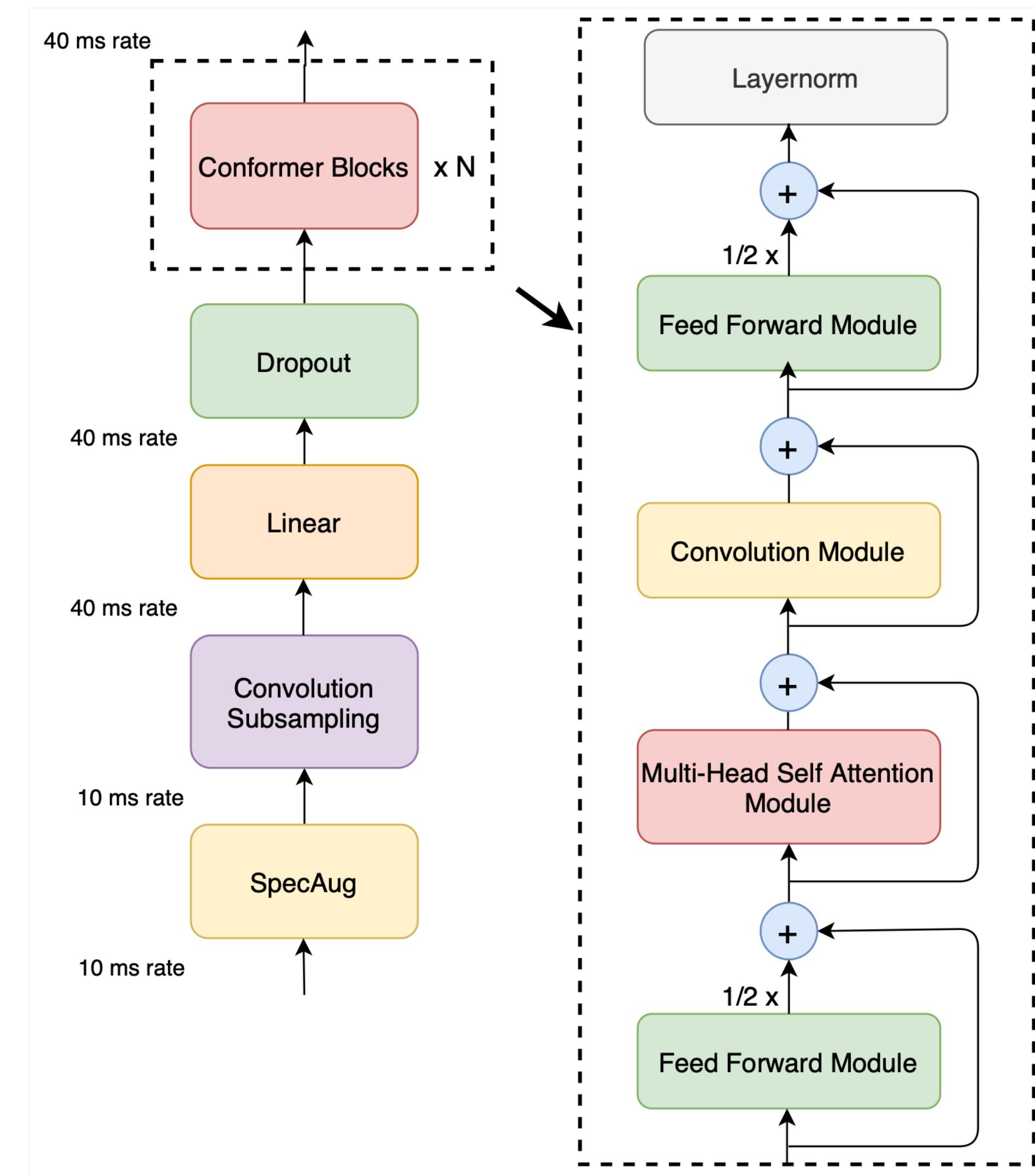
Largely the same story as in computer vision.

But with spectrograms instead of images.

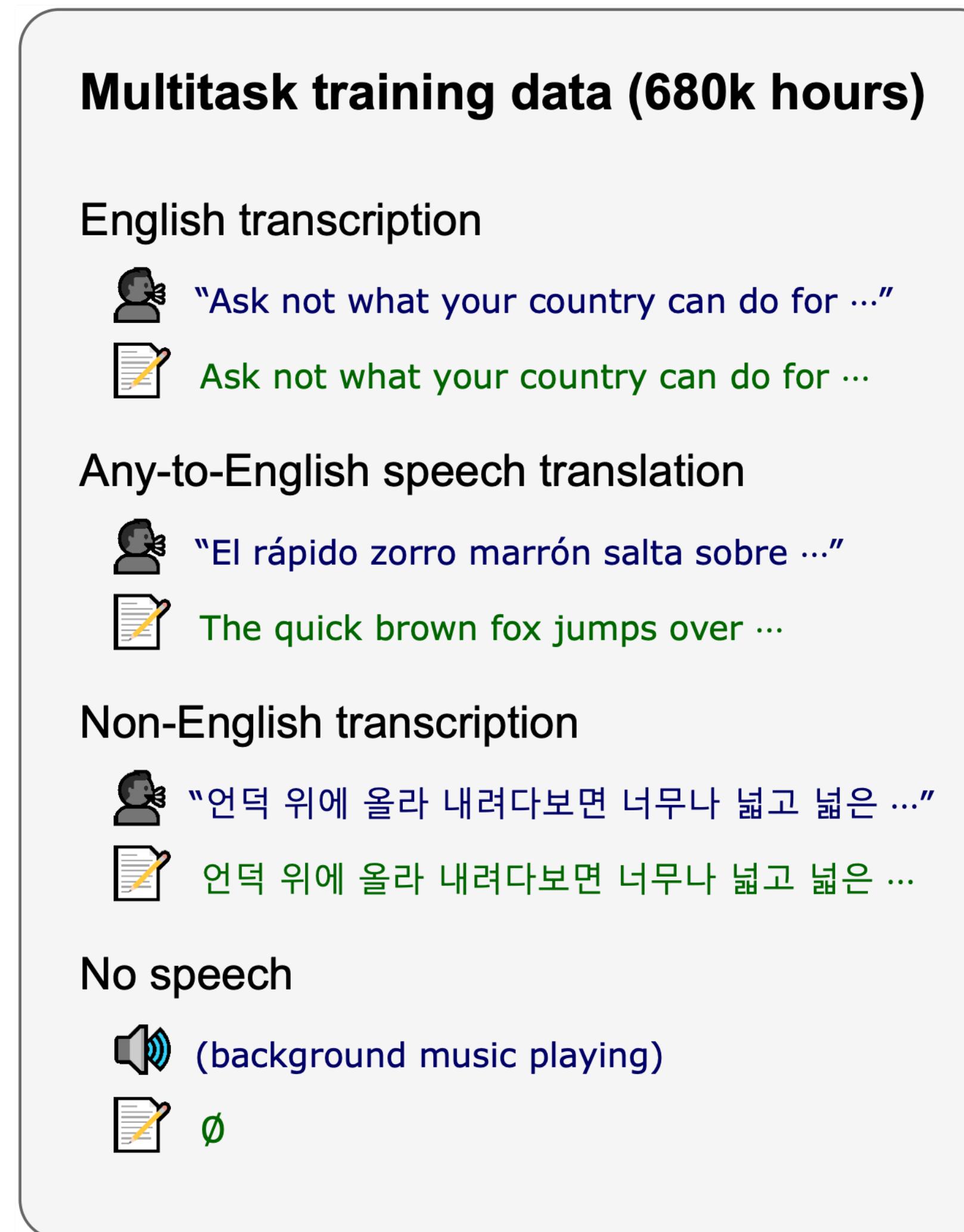


Conformer adds a third type of block using convolutions, and slightly reorder blocks, but overall very transformer-like.

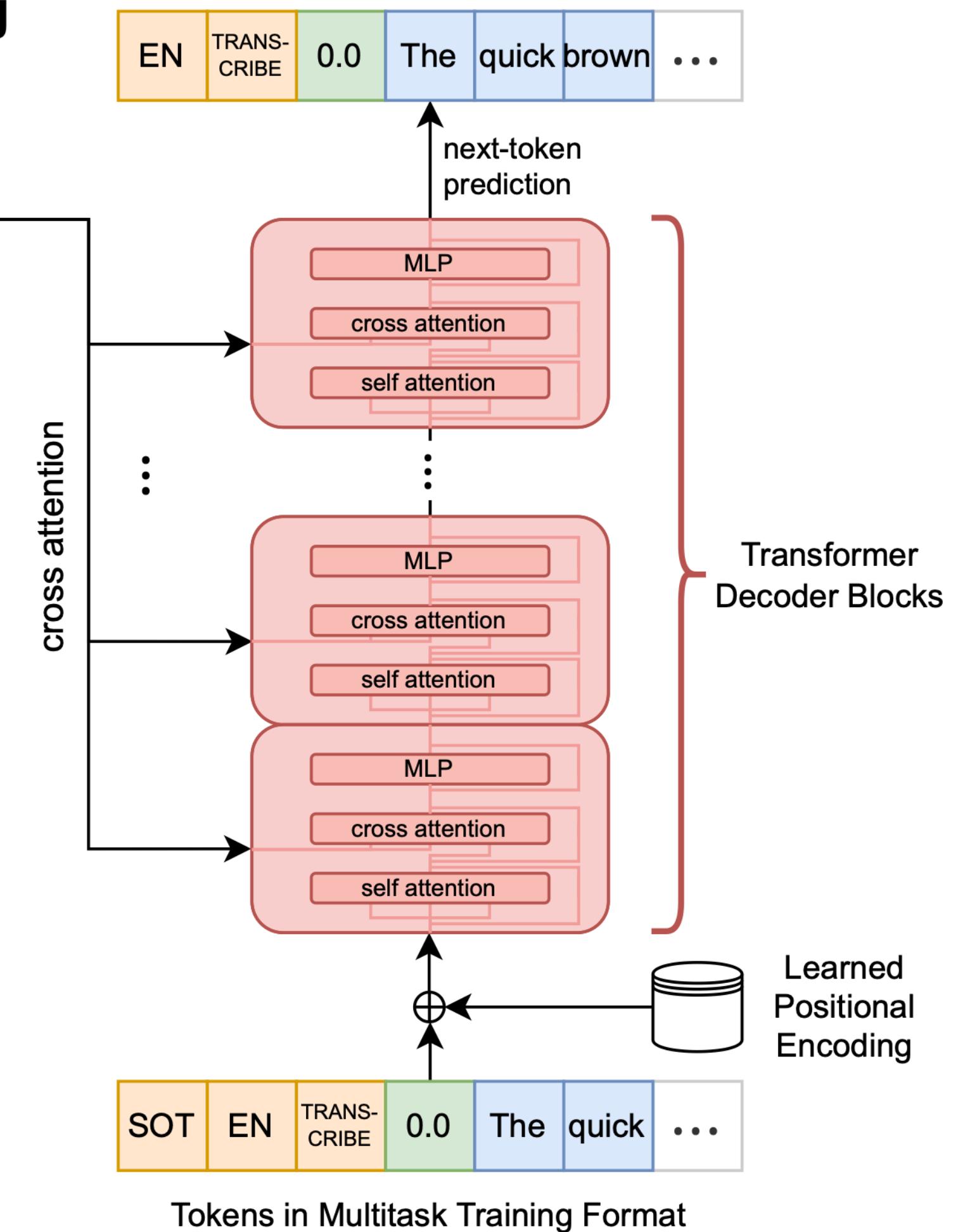
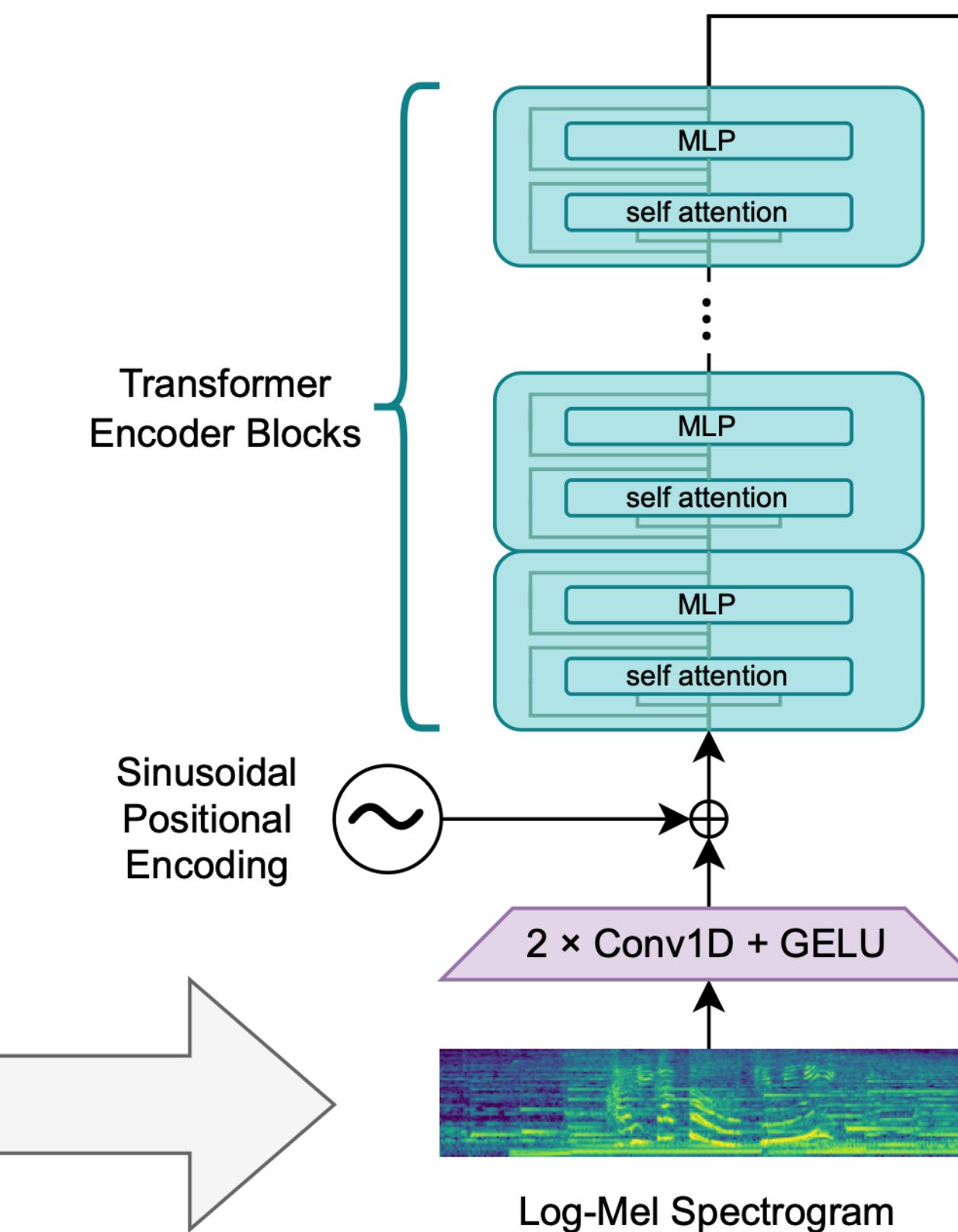
Exists as encoder-decoder variant, or as encoder-only variant with CTC loss.



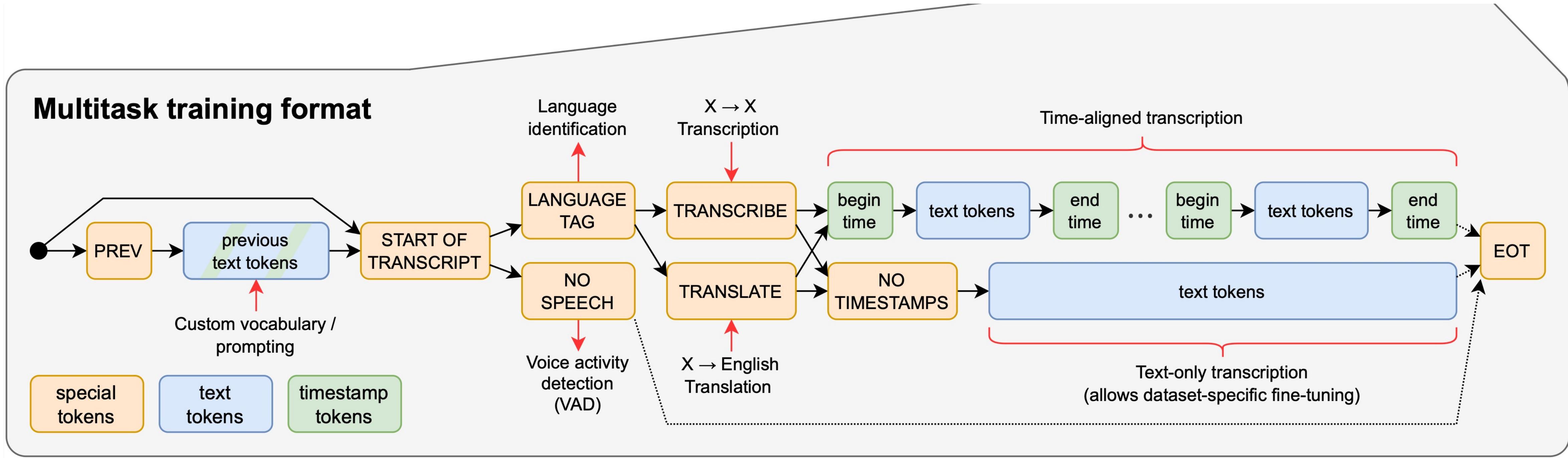
# Robust Speech Recognition via Large-Scale Weak Supervision - Whisper Model from OpenAI



## Sequence-to-sequence learning



## - Whisper Model from OpenAI

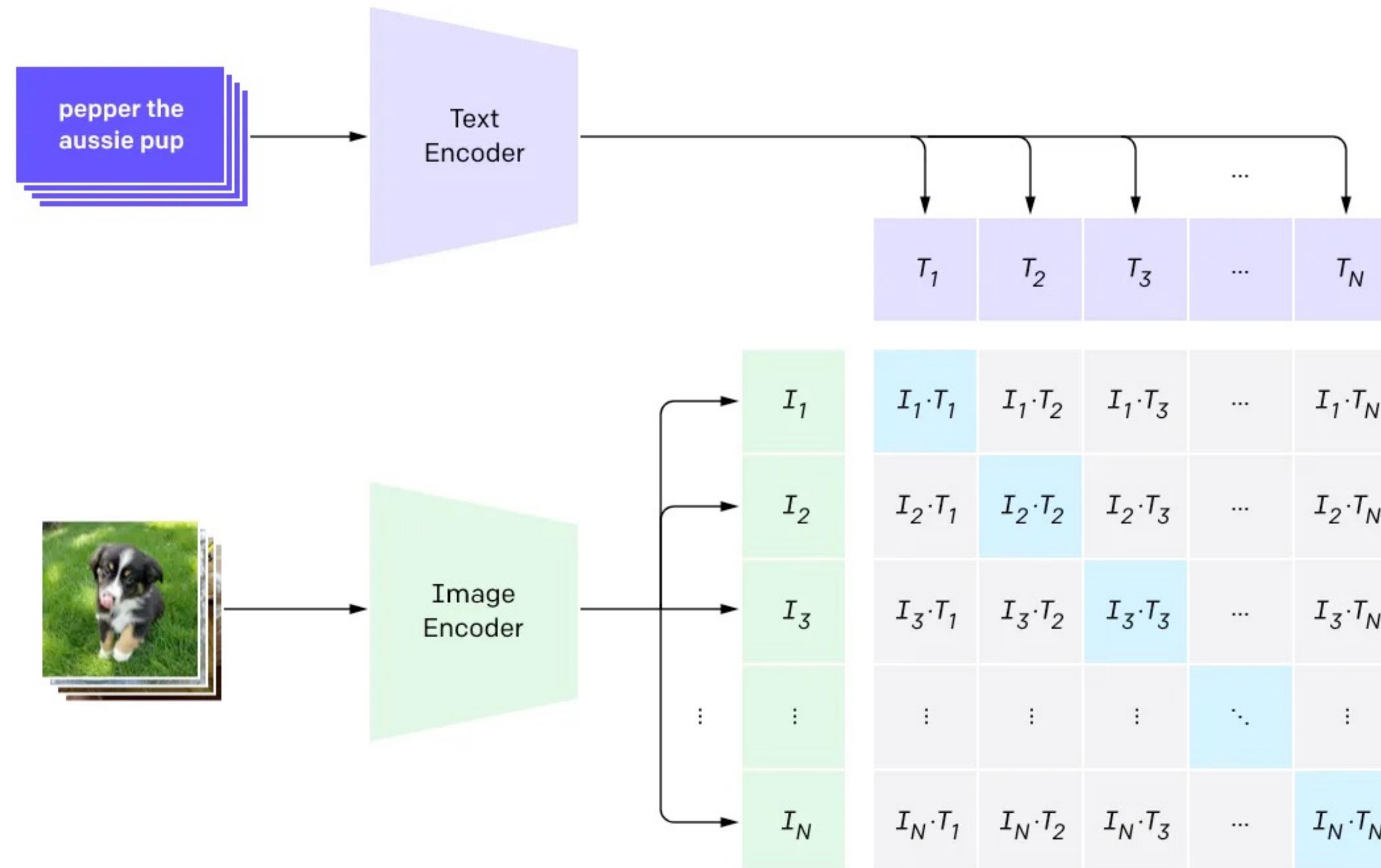


**Figure 1. Overview of our approach.** A sequence-to-sequence Transformer model is trained on many different speech processing tasks, including multilingual speech recognition, speech translation, spoken language identification, and voice activity detection. All of these tasks are jointly represented as a sequence of tokens to be predicted by the decoder, allowing for a single model to replace many different stages of a traditional speech processing pipeline. The multitask training format uses a set of special tokens that serve as task specifiers or classification targets, as further explained in Section 2.3.

# Vision and Language Models

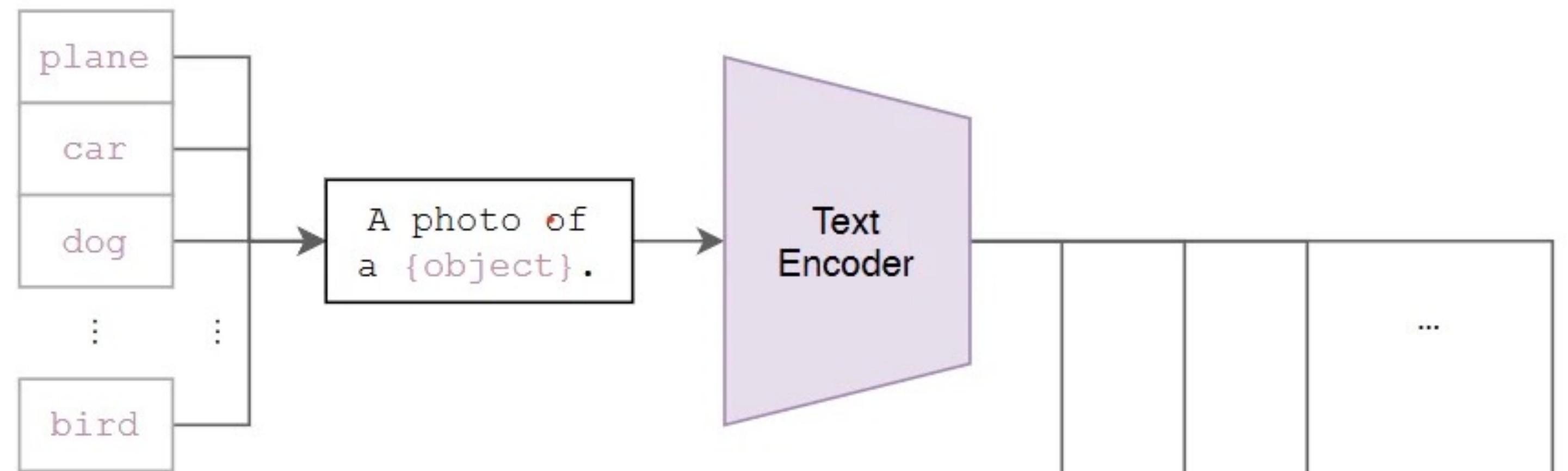
# CLIP Model - OpenAI

## 1. Contrastive pre-training

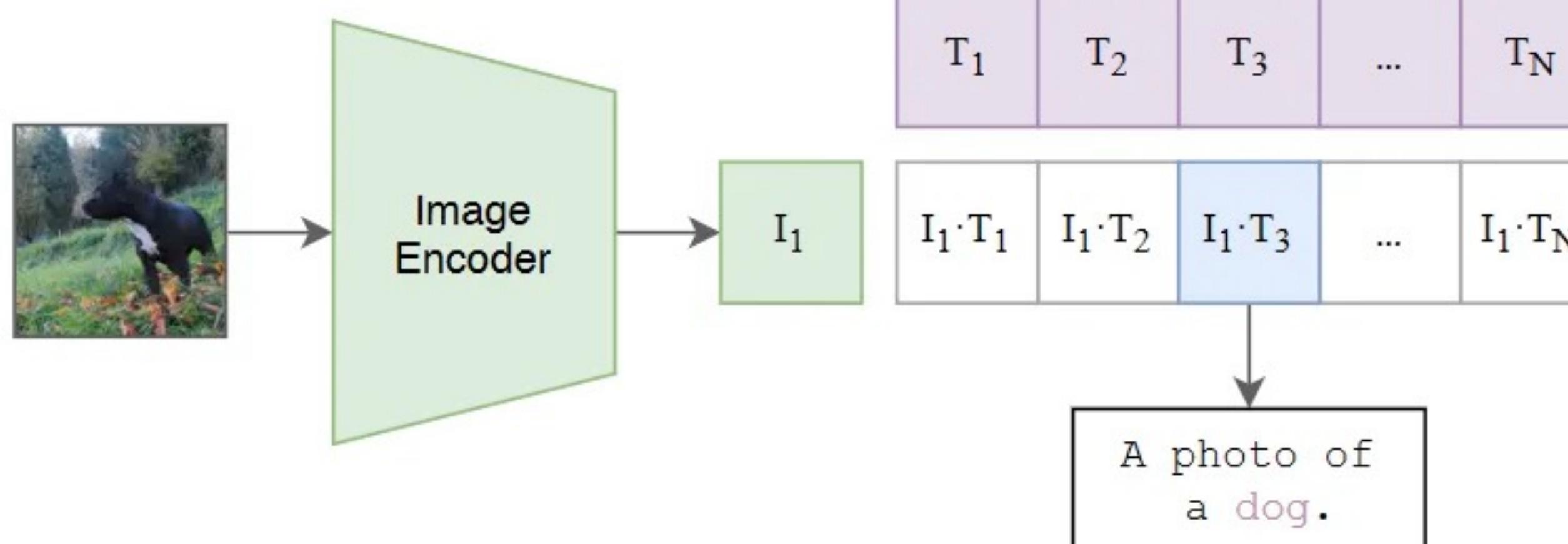


# CLIP Model - OpenAI

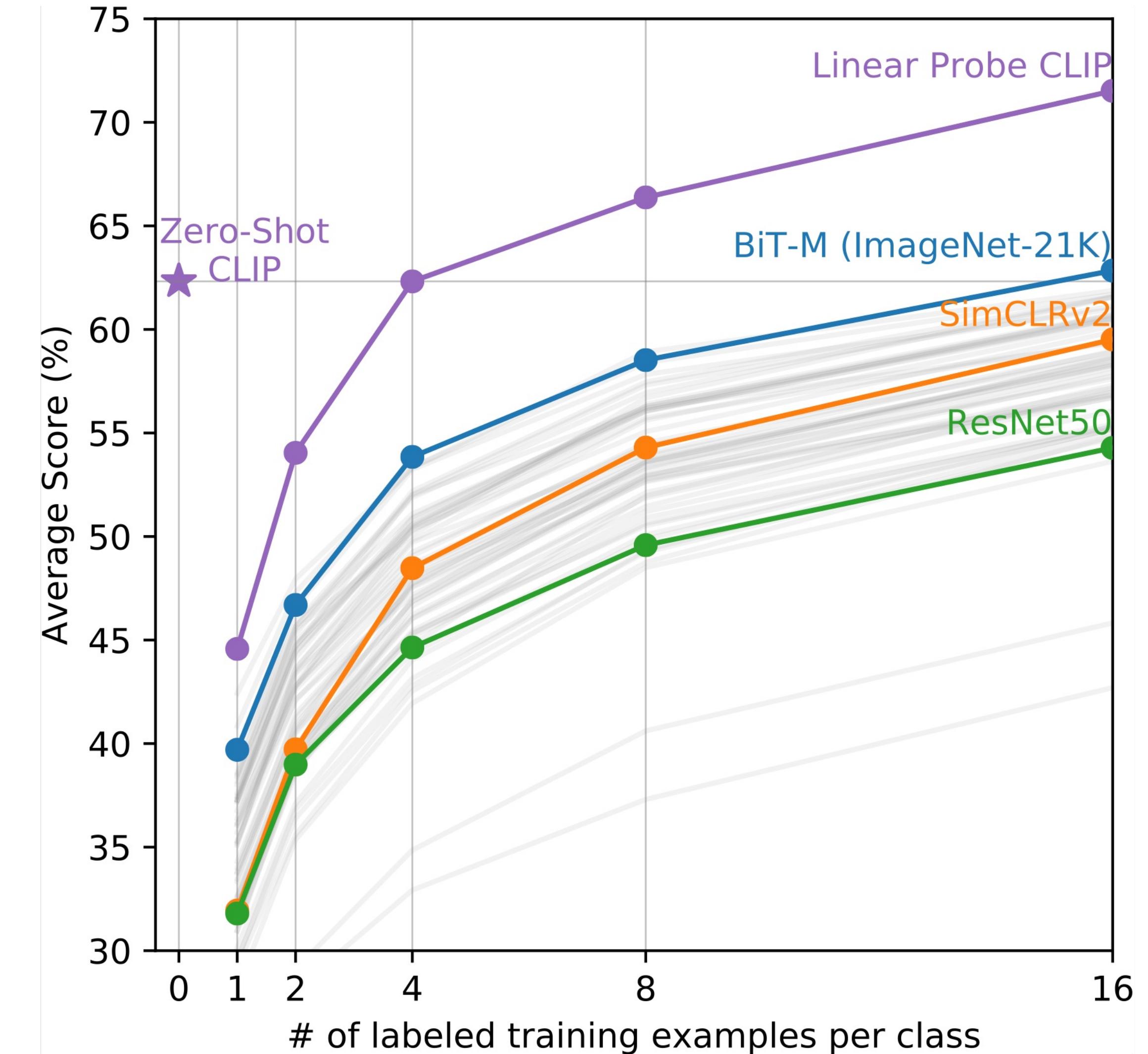
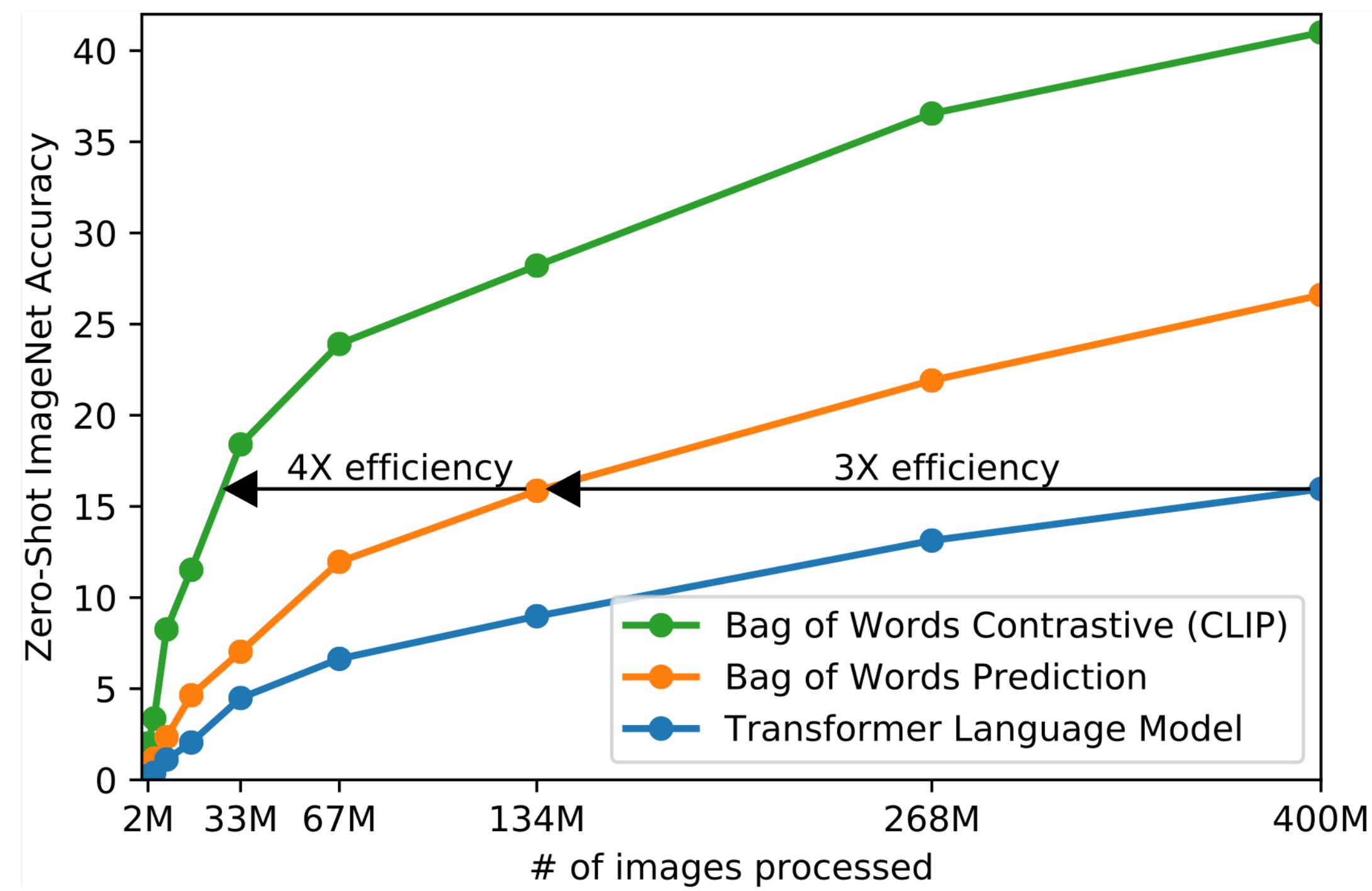
(2) Create dataset classifier from label text



(3) Use for zero-shot prediction



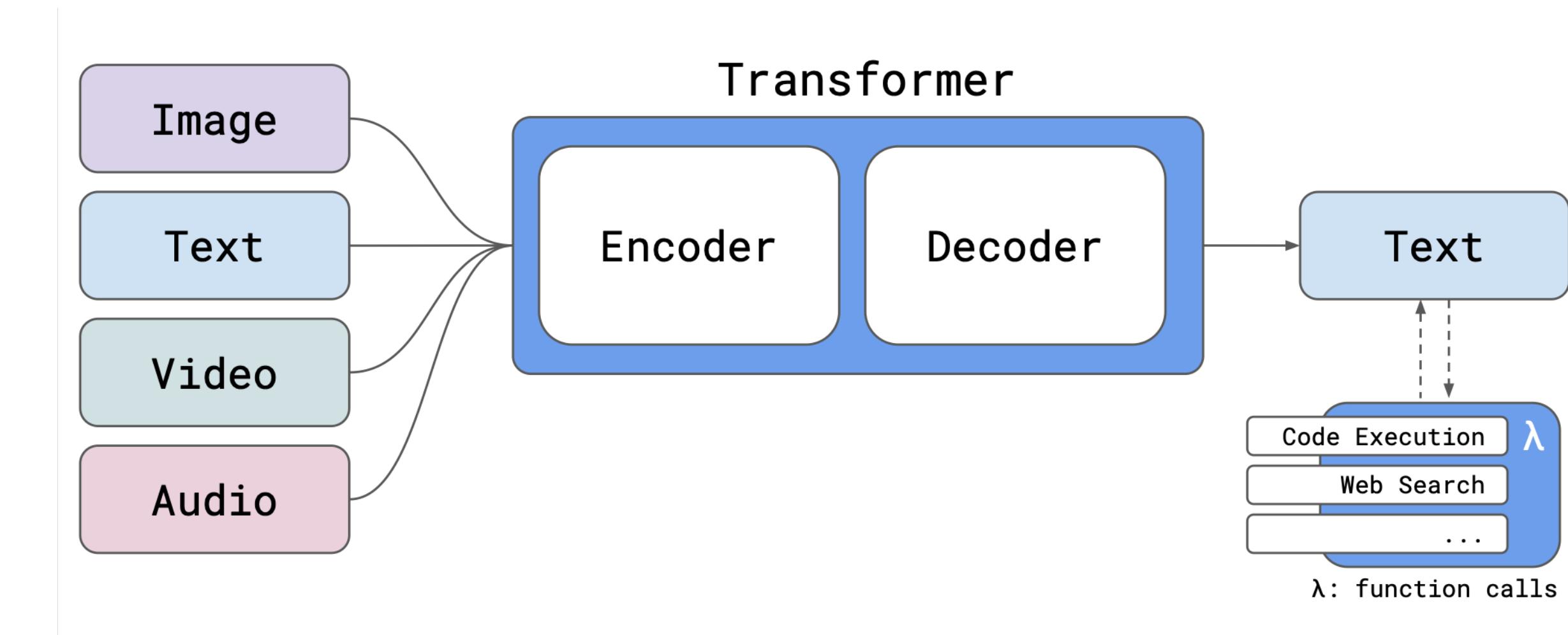
# CLIP Model - OpenAI



# **The Transformer's Unification of communities**

# Reka Model - One of the recent GPT competitors

Figure 2: **Architectural overview for Reka Core, Flash & Edge models:** a modular encoder-decoder transformer supporting multimodal input (image, text, video & audio). The text output can invoke function calls, such as web search and code execution, then return the results.



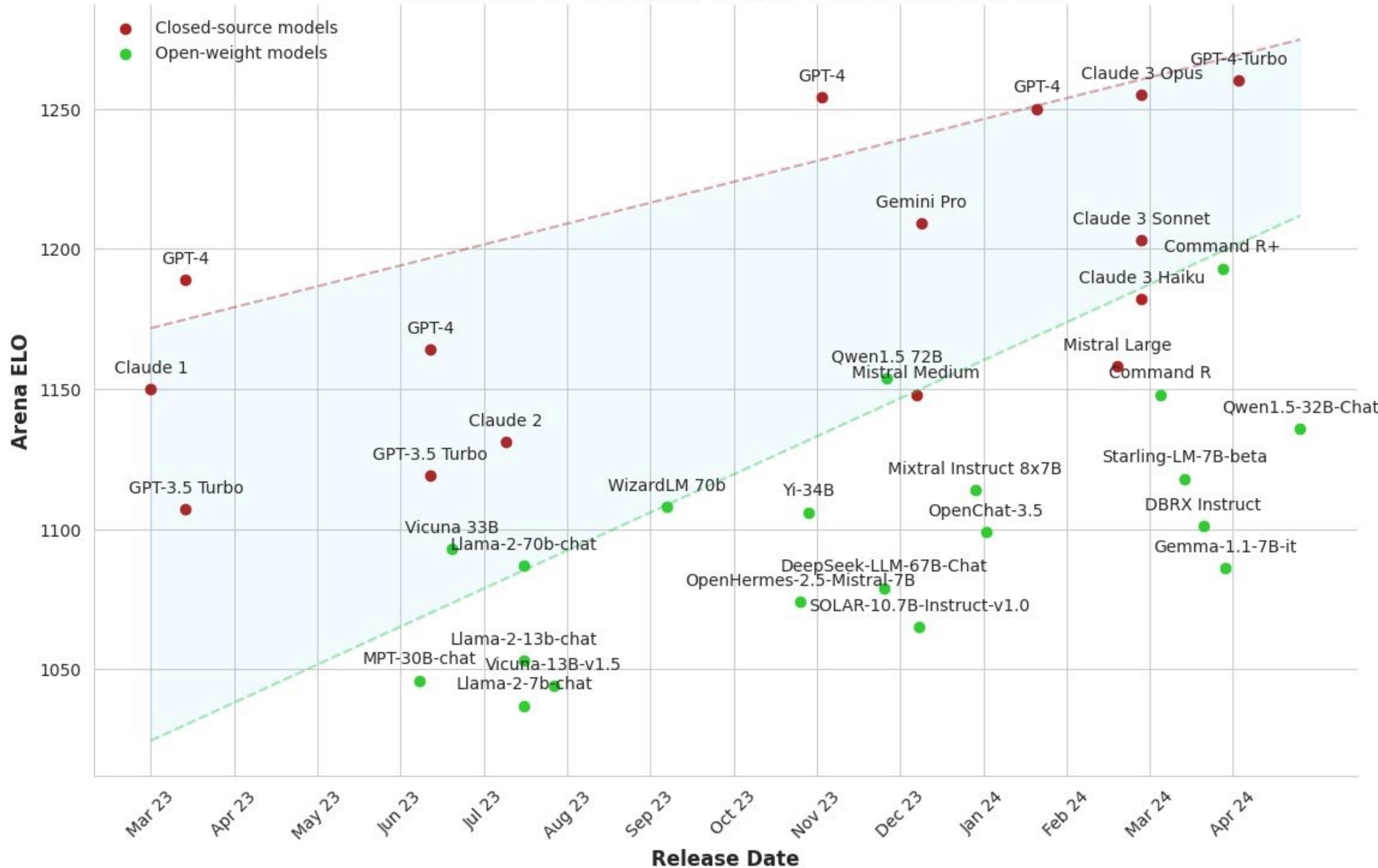
Similar approach is followed by most large models

# Reka Model - One of the recent GPT competitors

Table 5: Comparisons of our Reka Flash and Reka Core against other frontier models. Dashes (-) refer to either model not supporting modality or unavailable benchmark scores.

Model / Eval	Reka Core v0.5	Reka Flash v1.5	GPT-4	Claude 3 Opus	Claude 3 Sonnet	Gemini Ultra	Gemini Pro 1.5
MMLU <i>(Knowledge)</i>	83.2	75.9	86.4	86.8	79.0	83.7	81.9
GSM8K <i>(Reasoning)</i>	92.2	85.8	92.0	95.0	92.3	94.4	91.7
HumanEval <i>(Coding)</i>	76.8	72.0	76.5	84.9	73.0	74.4	71.9
GPQA ( <i>main</i> ) <i>(Hard QA)</i>	38.2	34.0	38.1	50.2	39.1	35.7	41.5
MMMU <i>(Image QA)</i>	56.3	53.3	56.8	59.1	53.1	59.4	58.5
VQAv2 <i>(Image QA)</i>	78.1	78.4	77.2	-	-	77.8	73.2
Perception-test <i>(Video QA)</i>	59.3	56.4	-	-	-	54.7	51.1 <sup>3</sup>

## Closed-source vs. Open-weight models (Arena ELO)



A note on

# Efficient Transformers

# A note on Efficient Transformers

The self-attention operation complexity is  $O(N^2)$  for sequence length  $N$ .

We'd like to use large  $N$ :

- Whole articles or books
- Full video movies
- High resolution images

Many  $O(N)$  approximations to the full self-attention have been proposed in the past two years.

Unfortunately, none provides a clear improvement.  
They always trade-off between speed and quality.

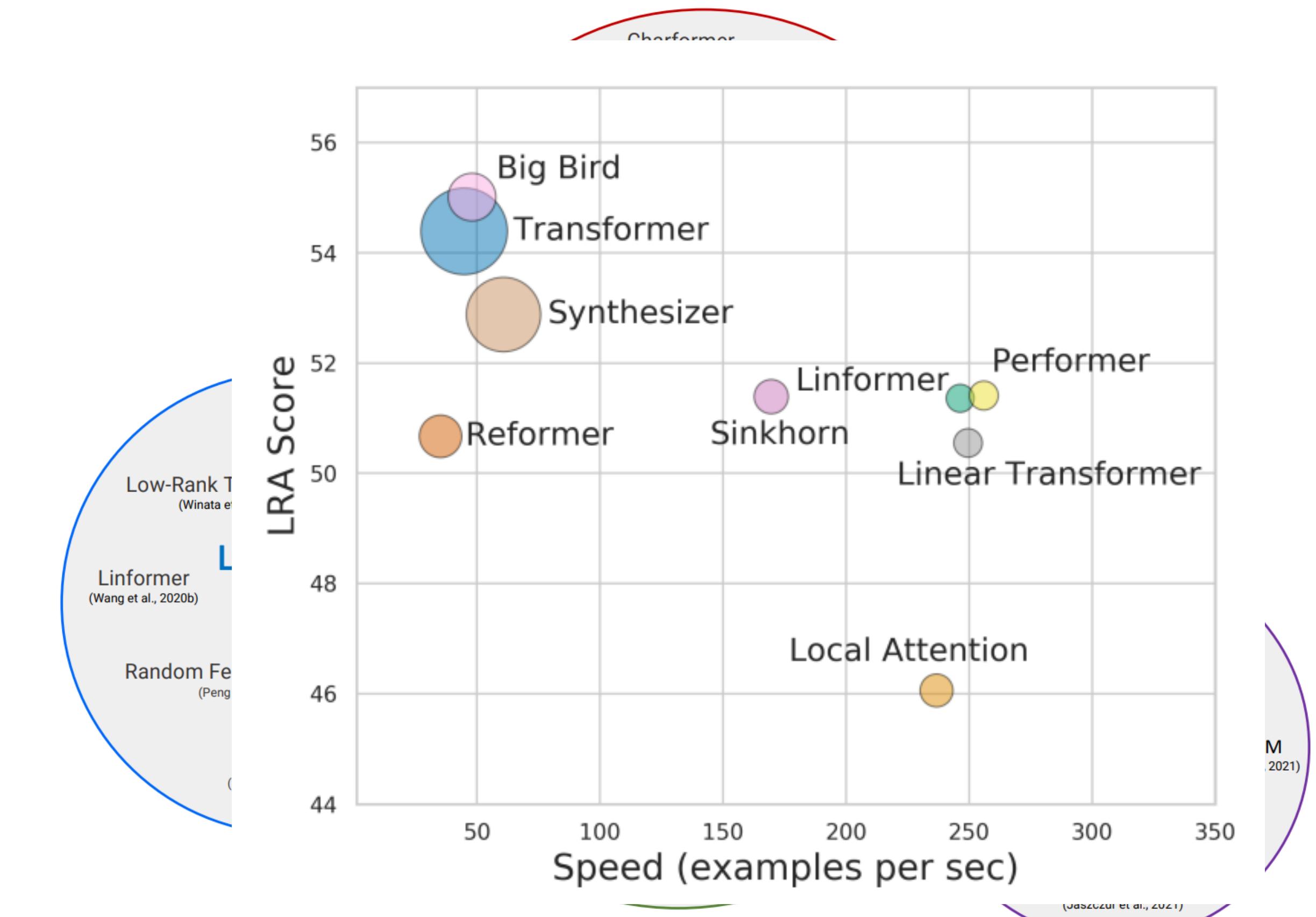


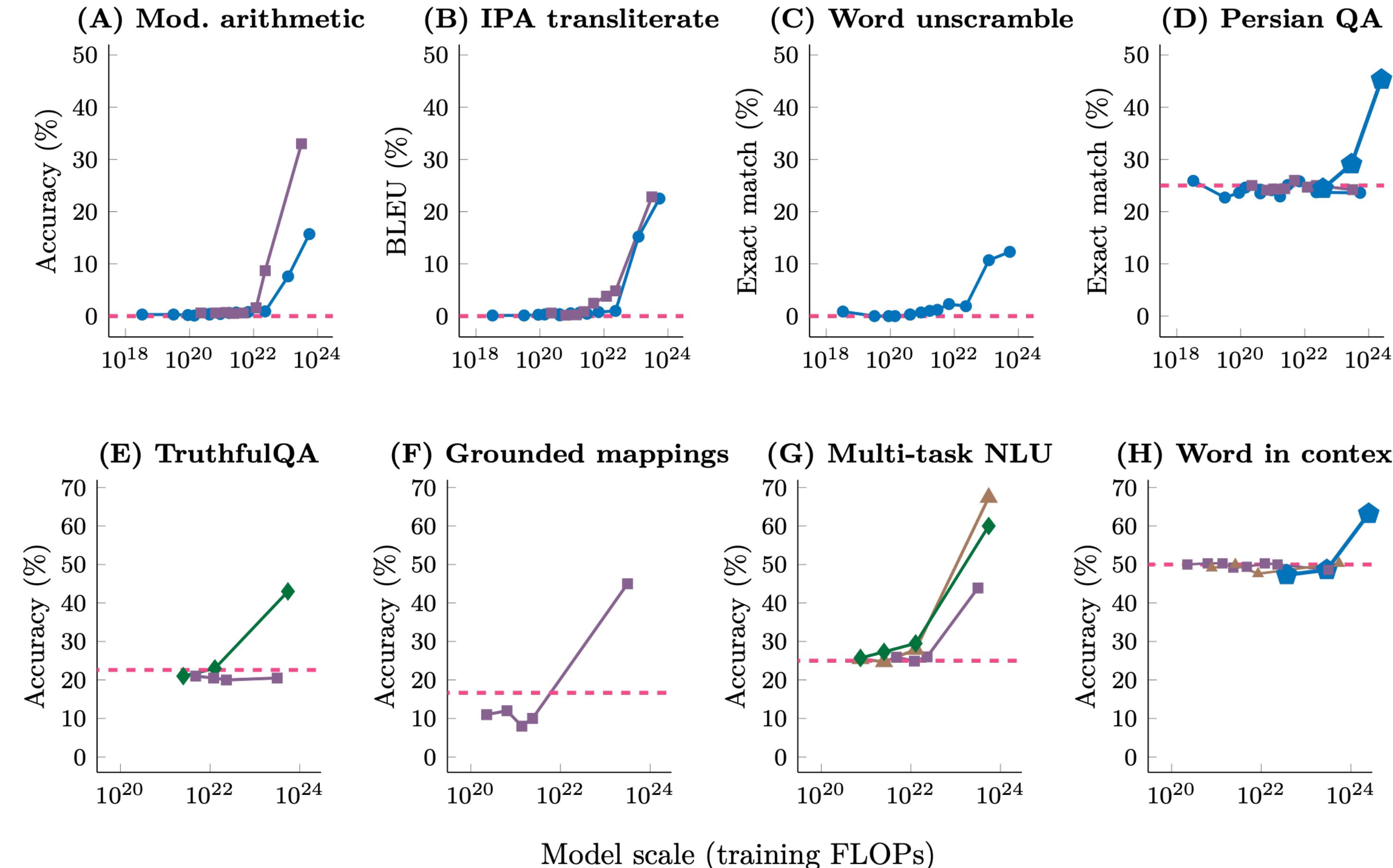
Figure 2: Taxonomy of Efficient Transformer Architectures.

# **Emergence in LLMs**

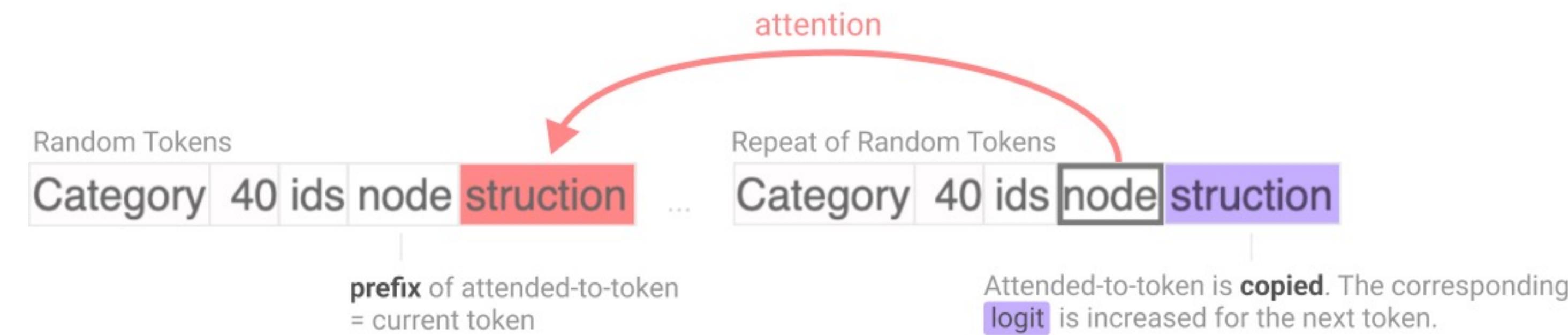
# Emergent Abilities of Large Language Models

*Emergence is when quantitative changes in a system result in qualitative changes in behavior.*

*An ability is emergent if it is not present in smaller models but is present in larger models.*



# In-context Learning and Induction Heads

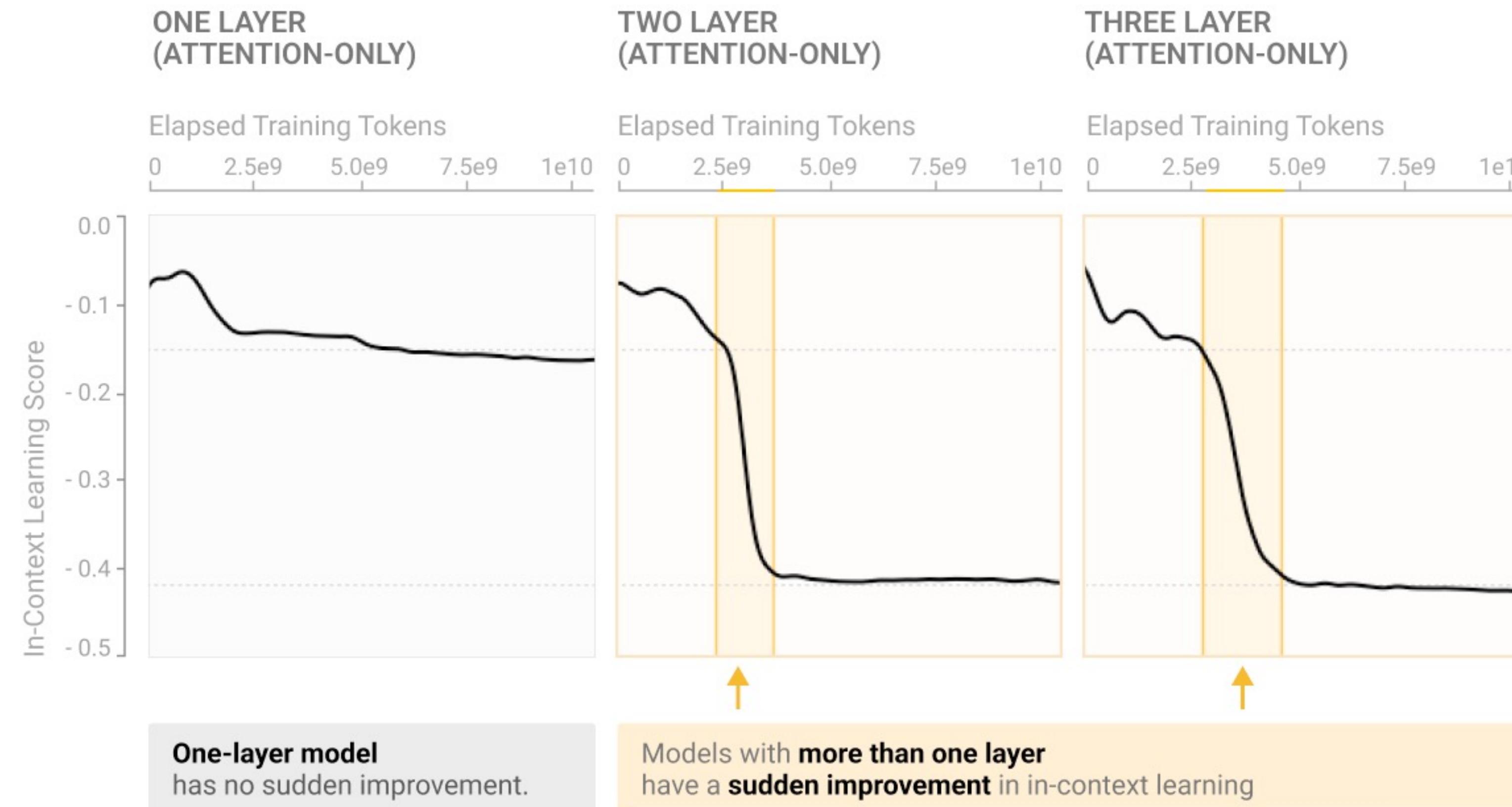


“We observed that induction heads could be seen as a kind of "in-context nearest neighbor" algorithm”

<https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html>

# In-context Learning and Induction Heads

**MODELS WITH MORE THAN ONE LAYER HAVE AN ABRUPT IMPROVEMENT IN IN-CONTEXT LEARNING**

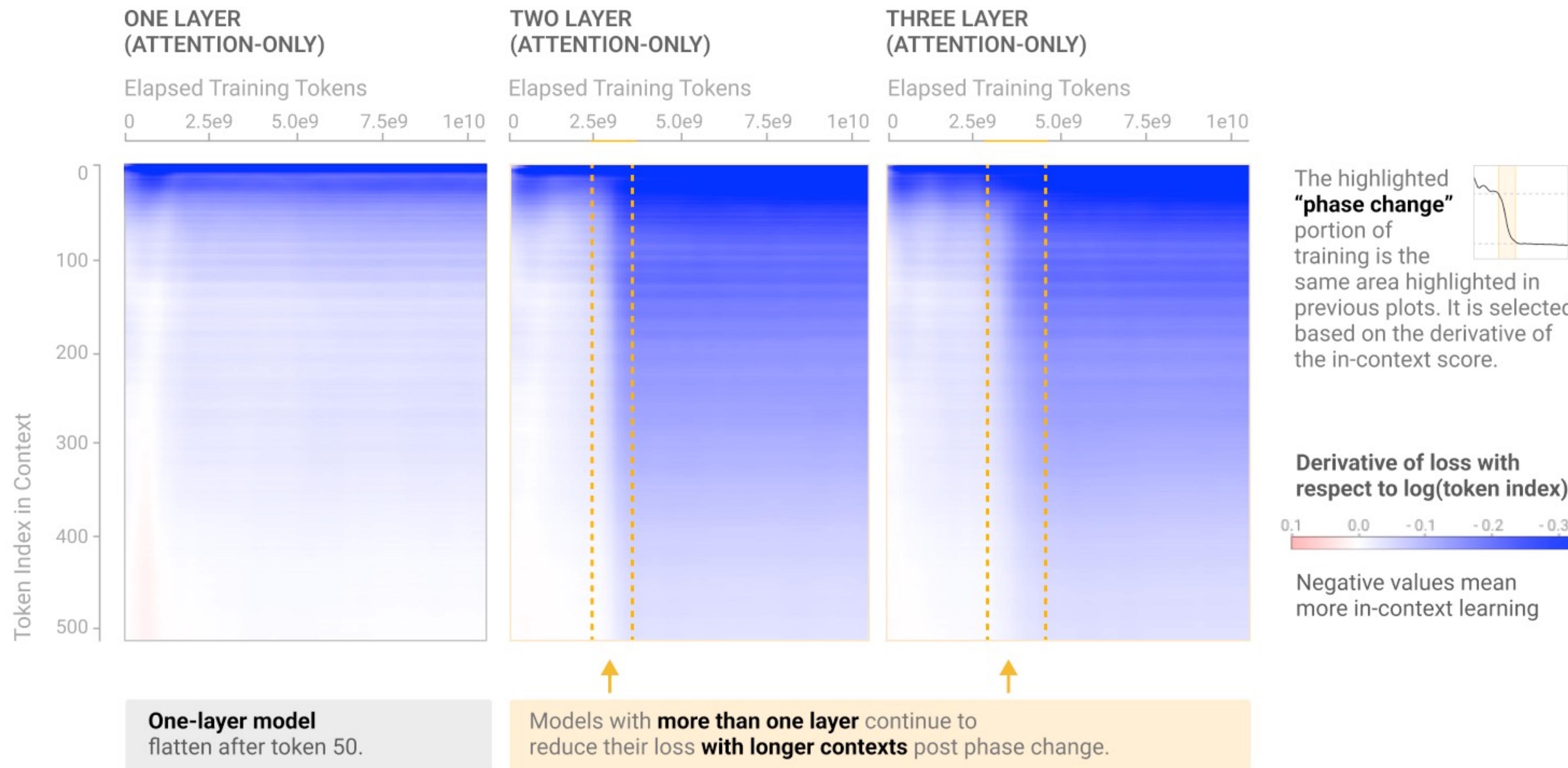


We highlight the **"phase change"** period of training in plots to make visual comparison between plots easier. The highlighted region is selected for each model based on the derivative of in-context learning.

# In-context Learning and Induction Heads

## DERIVATIVE OF LOSS WITH RESPECT TO LOG TOKEN INDEX

The rate at which loss decreases with increasing token index can be thought of as something like “in-context learning per token”. This appears to be most naturally measured with respect to the log number of tokens.



# References

- 3Blue1Brown - [Video 1](#), [Video2](#)
- Andrej Karpathy - [NanoGPT implementation](#), [Video Tutorial](#)



That's all  
folks

QUESTIONS?

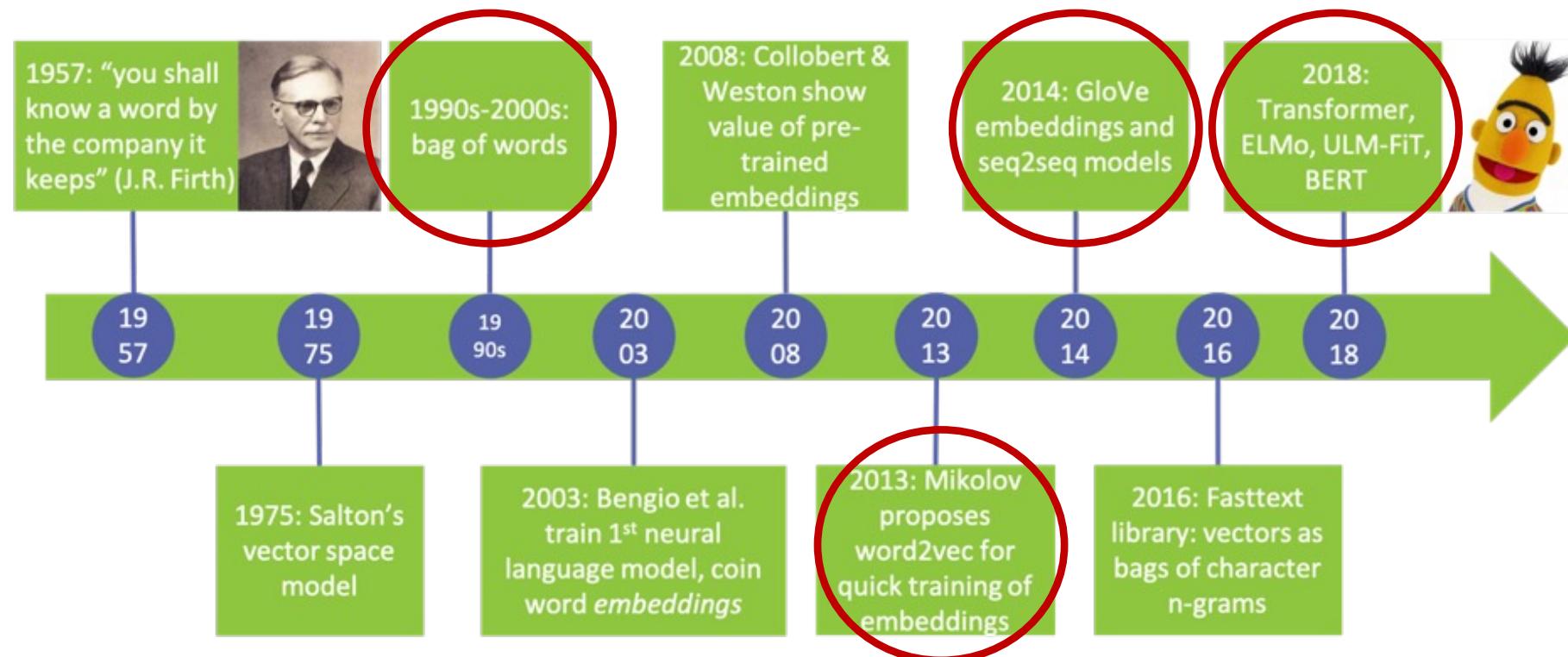
# Week 10: Natural Language Processing (NLP)

Vinay P. Namboodiri

# Topic 1: Introduction to NLP

# What is NLP?

- **Natural language processing (NLP)** is a subfield of computer science that focuses on developing algorithms and computational models to analyze, understand, and generate human language.



# Challenges of NLP

- Variable input size:
  - “The alien mothership is in orbit here! If we hit that bullseye, the rest of the dominoes will fall like a house of cards! Checkmate!” – 25 words
  - “Stop exploding you cowards!” - 4 words
- Sensitive: Small changes can have large effects
  - “Let’s eat, Jack.” vs “Let’s eat Jack.” (comma)
  - “Dog bites man.” vs “Man bites dog.” (word order)
  - “I miss home.” vs “We might miss the train.” (same word, different meaning)
  - “I hit the man with a stick.” (who is holding the stick?)
- Redundant: Many ways to say the same thing
  - “The same thing can be said in many different ways.”
  - “There are a plurality of methods for communicating an identical concept.”
- ...
- More difficult: common sense, culture information

# NLP applications

- Text Classification
- Named Entity Recognition
- Document search
- Sentiment Analysis
- Text Summarization
- Topic Modelling
- Machine Translation
- Question Answering
- Chatbot
- ....

# Early machine translation

- Georgetown-IBM experiment (1954): The Georgetown-IBM experiment was the first demonstration of machine translation, where researchers attempted to translate Russian sentences into English.

# Early chatbot

- ELIZA (1966) was one of the first chatbots and one of the first programs capable of attempting the Turing test.
- Rule-based

```
Welcome to
      EEEEEE  LL      IIII      ZZZZZZ      AAAAAA
      EE      LL      II       ZZ      AA      AA
      EEEEEE  LL      II       ZZZ      AAAAAAAA
      EE      LL      II       ZZ      AA      AA
      EEEEEE  LLLLLL  IIII  ZZZZZZ  AA      AA

Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.

ELIZA: Is something troubling you ?
YOU: Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU: They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU: Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU: He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU: It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

# Topic 2: Text Pre-Processing

# Tokenization

- **Tokenization** is the process of breaking down the given text in MLP into smallest unit in a sentence called a **token**.

```
import nltk
from nltk import word_tokenize
text = "Let's first tokenize the sentence into words using nltk.word_tokenize()."
word_list = nltk.word_tokenize(text)
print(word_list)
```

```
['Let', "'", 's', 'first', 'tokenize', 'the', 'sentence', 'into', 'words',
 'using', 'nltk.word_tokenize', '(', ')', '.']
```

**Q:** why do we need tokenization?

# Handling unknown words

- What happens when we encounter a word at test time that we've never seen in our training data?
  - With word level tokenization, we have no way of assigning an index to an unseen word!
  - This means we don't have a word embedding for that word and thus cannot process the input sequence
- Solution: replace low-frequency words in training data with a special <UNK> token, use this token to handle unseen words at test time too
  - Why use <UNK> tokens during training?

# Limitations of <UNK>

- We lose lots of information about texts with a lot of rare words / entities

The chapel is sometimes referred to as "Hen Gapel Lligwy" ("hen" being the Welsh word for "old" and "capel" meaning "chapel").

The chapel is sometimes referred to as " Hen <unk> <unk> " (" hen " being the Welsh word for " old " and " <unk> " meaning " chapel ").

# Other limitations

- Word-level tokenization treats different forms of the same word (e.g., “open”, “opened”, “opens”, “opening”, etc) as separate types –> separate embeddings for each

This can be problematic especially when training over smaller datasets, why?

# An alternative: character tokenization

- Small vocabulary, just the number of unique characters in the training data!
- However, you pay for this with longer input sequences.  
Why is this a problem for the models we've discussed?

# An alternative: character tokenization

- Small vocabulary, just the number of unique characters in the training data!
- However, you pay for this with longer input sequences.  
Why is this a problem for the models we've discussed?

# Byte pair encoding

- Form base vocabulary (all characters that occur in the training data)

word	frequency
hug	10
pug	5
pun	12
bun	4
hugs	5

- Base vocab: **b, g, h, n, p, s, u**

# Byte pair encoding

- Now, count up the frequency of each character *pair* in the data, and choose the one that occurs most frequently

word	frequency	character pair	frequency
h+u+g	10	ug	20
p+u+g	5	pu	17
p+u+n	12	un	16
b+u+n	4	hu	15
h+u+g+s	5	gs	5
...			

# Byte pair encoding

- Now, choose the most common pair ( $ug$ ) and then merge the characters together into one symbol. Add this new symbol to the vocabulary. Then, retokenize the data

word	frequency	character pair	frequency
$h+ug$	10	$un$	16
$p+ug$	5	$h+ug$	15
$p+u+n$	12	$pu$	12
$b+u+n$	4	$p+ug$	5
$h+ug+s$	5	$ug+s$	5
...			

# Byte pair encoding

- Keep repeating this process! This time we choose *un* to merge, next time we choose *h+ug*, etc.

word	frequency	character pair	frequency
<i>h+ug</i>	10	<i>un</i>	16
<i>p+ug</i>	5	<i>h+ug</i>	15
<i>p+u+n</i>	12	<i>pu</i>	12
<i>b+u+n</i>	4	<i>p+ug</i>	5
<i>h+ug+s</i>	5	<i>ug+s</i>	5
...			

# Byte pair encoding

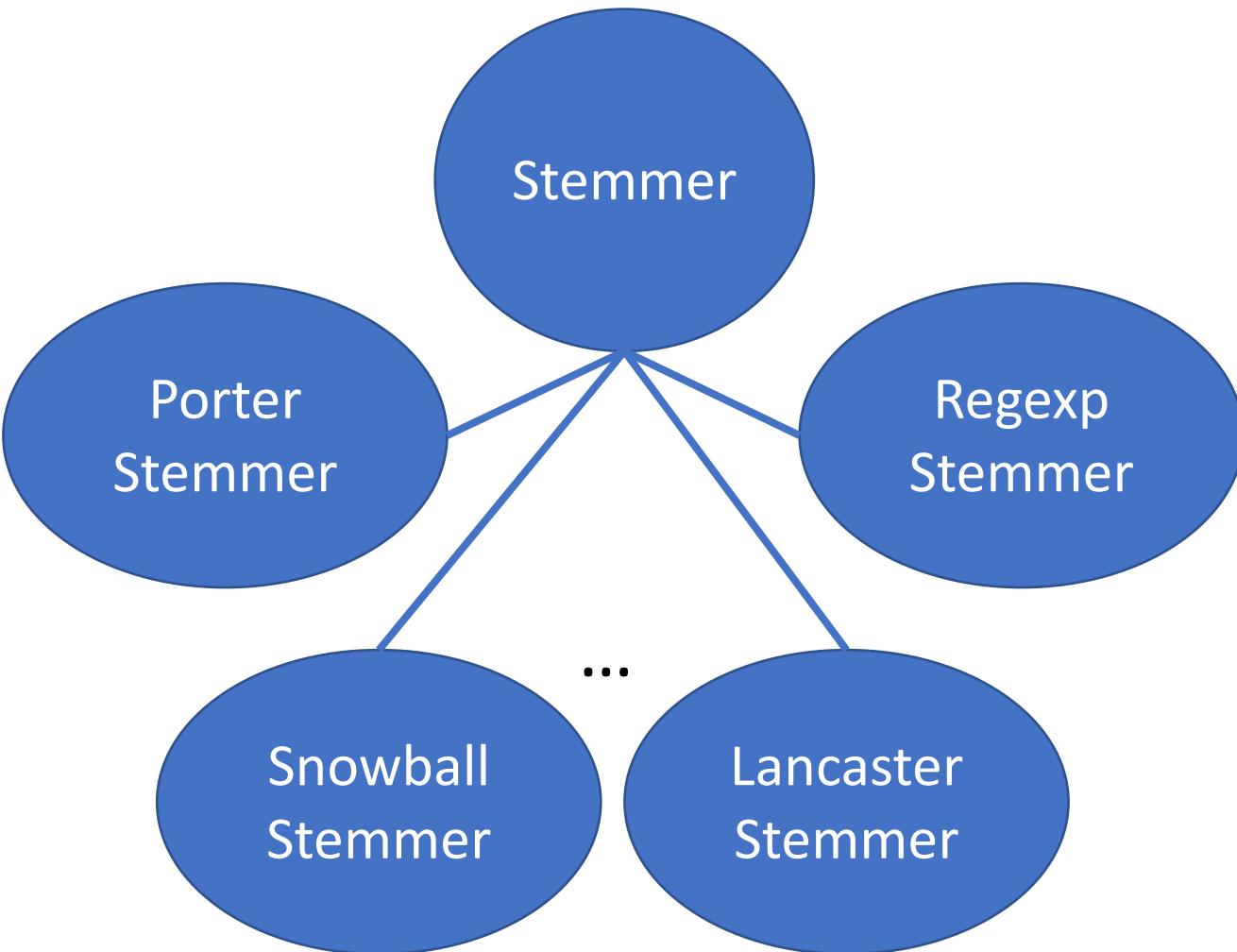
- Eventually, after a fixed number of merge steps, we stop

word	frequency
<i>hug</i>	10
p+ug	5
p+un	12
b+un	4
<i>hug + s</i>	5

- new vocab: **b, g, h, n, p, s, u, ug, un, hug**

# Stemming

- Problem: lots of words! (150– 500k, many ways to account)
- **Stemming:** is the process of finding the root of words.
- Example:
  - “like”, “likes”, “liked”, “likely”, “liking” → “like”
  - “warm”, “warmer”, “warmed” → “warm”



- Rules based (E.g., Porter stemmer: 5-step rules)

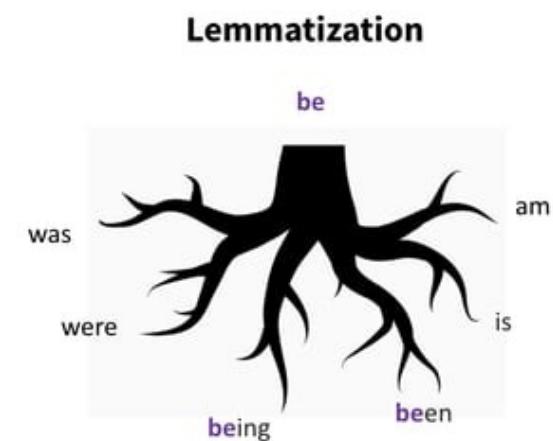
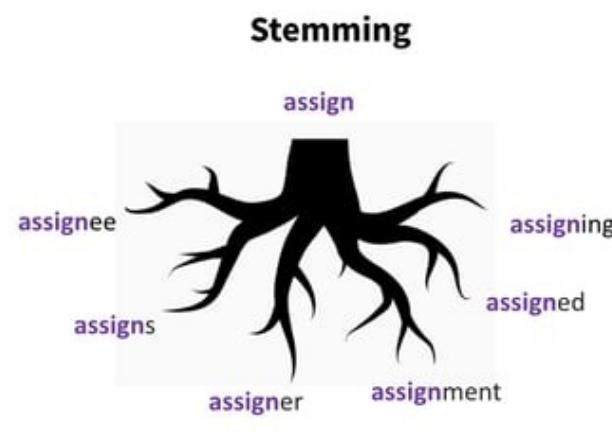
```

import nltk
from nltk.stem import PorterStemmer
ps = PorterStemmer()
words = ['caresses', 'flies', 'dies', 'mules', 'denied',
'died', 'agreed', 'owned', 'humbled', 'sized',
'meeting', 'stating', 'siezing', 'itemization',
'sensational', 'traditional', 'reference', 'plotted']
words_after_stem = [ps.stem(word) for word in words]
print(' '.join(words_after_stem))
  
```

caress fli die mule deni die agre own  
humbl size meet state siez item sensat  
tradit refer plot

# Lemmatization

- Also reduce words to their base or root form.
- **Lemmatization** produces a valid base word that is a morphological variant of the original word, while stemming simply chops off the suffixes of the word.
- Examples:
  - “ran” → “run”, “better” → “good”, “am”, “is”, “was” → “be”



# Stop word removal

- For some of the applications it is also useful to omit the common stop words that are the most frequent words such as 'a', 'an', 'the'.

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
 "you'll", "you'd", "your", 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself',
 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if',
 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against',
 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to',
 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',
 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own',
 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't",
 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
 "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',
 "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
 "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren',
 "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

# Part-Of-Speech Tagging (POS Tag)

- **Part of Speech Tagging (POS-Tag)** is the labelling of the words in a text according to their word types (noun, adjective, adverb, verb, etc.).



# Part-Of-Speech Tagging (POS Tag)

- **Part of Speech Tagging (POS-Tag)** is the labelling of the words in a text according to their word types (noun, adjective, adverb, verb, etc.).

```
import nltk
from nltk import word_tokenize
text = "The striped bats are hanging on their feet for best"
tokens = nltk.word_tokenize(text)
print("Parts of Speech: ",nltk.pos_tag(tokens))
```

Determiner                      Adjective                      Noun, plural

Verb, non-3rd person  
singular present



```
Parts of Speech: [('The', 'DT'), ('striped', 'JJ'), ('bats', 'NNS'), ('are', 'VBP'), ('hanging', 'VBG'), ('on', 'IN'), ('their', 'PRP$'), ('feet', 'NNS'), ('for', 'IN'), ('best', 'JJS')]
```

# Reference

- <https://medium.com/mlearning-ai/nlp-tokenization-stemming-lemmatization-and-part-of-speech-tagging-9088ac068768>

# Topic 3: BoW and TF-IDF

# Feature representation of a document

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



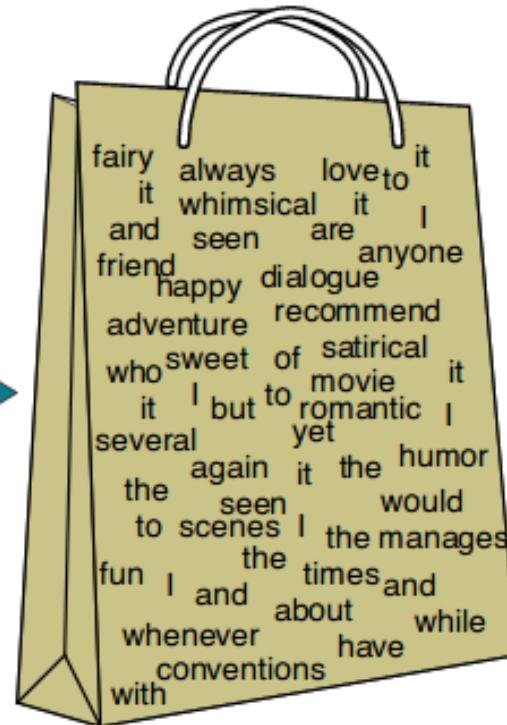
A sequence of words

$$\begin{bmatrix} 0.2 \\ 0 \\ 0.71 \\ -0.54 \\ 0.4 \\ 0.22 \\ 0.43 \\ \dots \\ -0.81 \end{bmatrix}$$

A fix-length vector

# Bag-of-words (BoW)

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

- Feature vector of a document = word frequency histogram
- A very long vector, mostly zeros

**Q:** Which of the following words are helpful for you to predict a document's topic?

“is”, “get”, “have”, “cosine”, “angle”, “equal”

More frequent words  $\neq$  More important

**Solution:** Higher weights are given to words which rarely occur in the corpus: **TF-IDF**

# Term Frequency – Inverse Document Frequency (TF-IDF)

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

Term Frequency:  $\text{TF}(t, d) = \frac{\text{Occurrence of term } t \text{ in document } d}{\text{Number of terms in } d} = \frac{n_{td}}{n_d}$

Inverse Document Frequency:  $\text{IDF}(t) = \log \frac{\text{Number of documents in the corpus}}{\text{Number of documents with term } t \text{ in it}} = \log \frac{N}{n_t}$

# Example

- In a corpus of 10000 documents, you pick a document D, which has a total of 2000 words.

Term (t)	Occurrence in D	Number of documents contains t	TF(t, D)	IDF(t)	TF-IDF
get	30	6000			
cosine	6	10			

$$TF(t, d) = \frac{\text{Occurrence of term } t \text{ in document } d}{\text{Number of terms in } d} = \frac{n_{td}}{n_d}$$

$$IDF(t) = \log \frac{\text{Number of documents in the corpus}}{\text{Number of documents with term } t \text{ in it}} = \log \frac{N}{n_t}$$

# Example

- In a corpus of 10000 documents you pick a document D, which has a total of 2000 words.

Term (t)	Occurrence in D	Number of documents contains t	TF(t, D)	IDF(t)	TF-IDF
get	30	6000	30/2000	$\log(10000/6000)$	0.0077
cosine	6	10	6/2000	$\log(10000/10)$	0.0207

$$TF(t, d) = \frac{\text{Occurrence of term } t \text{ in document } d}{\text{Number of terms in } d} = \frac{n_{td}}{n_d}$$

$$IDF(t) = \log \frac{\text{Number of documents in the corpus}}{\text{Number of documents with term } t \text{ in it}} = \log \frac{N}{n_t}$$

# Drawbacks of BoW and TF-IDF

- Very long vector
- Ignore the word order
- Treat words independently
  - Clearly naive
  - It works well for many applications. E.g., sentiment analysis, topic identification, spam filtering, etc.

# Reference

- Y. Hamdaoui's blog: "TF-IDF from scratch in python".  
<https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558>

# Topic 4: Introduction of Word Embedding

# What is word embedding?

- **Word Embedding:** converting a word/phrase into a fix-length vector.

I like playing football.



$$\begin{bmatrix} 0.1 \\ 0.9 \\ 1.2 \\ \dots \\ -1.3 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.9 \\ 1.2 \\ \dots \\ -1.3 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.9 \\ 1.2 \\ \dots \\ -1.3 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.9 \\ 1.2 \\ \dots \\ -1.3 \end{bmatrix}$$

# Simplest word embedding

- One-hot vector

I like playing football.



$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ \dots \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dots \\ 1 \end{bmatrix}$$

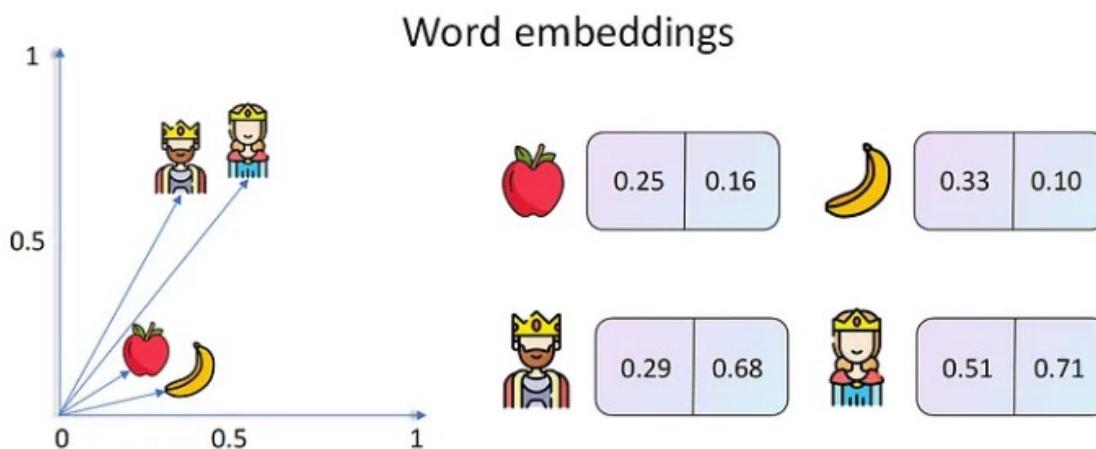
**Q:** What is the drawback of using one-hot vector?

# Drawbacks of one-hot vector

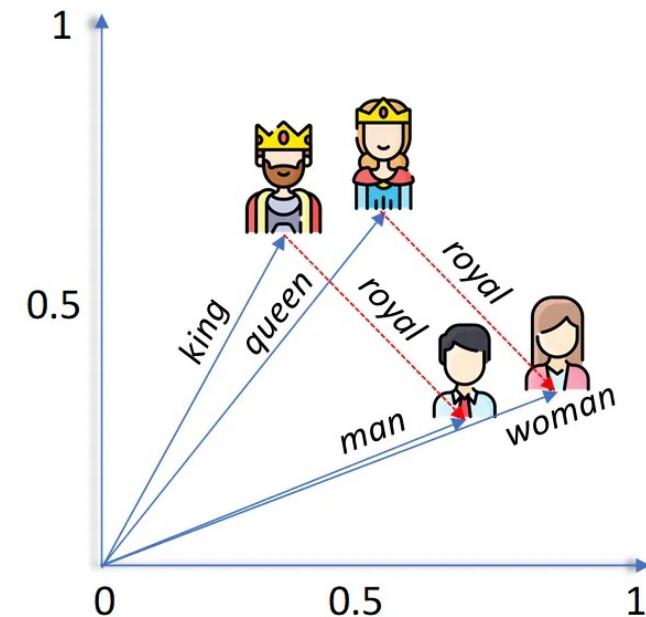
- The vector length is huge.
- The embedding is closely coupled to their application, requiring re-training the whole model, if the vocabulary changed.
- No context of words. All words have the same distance.

# What is a good word embedding?

- Word embedding aims at words that appear in similar contexts or have similar meaning are close together.



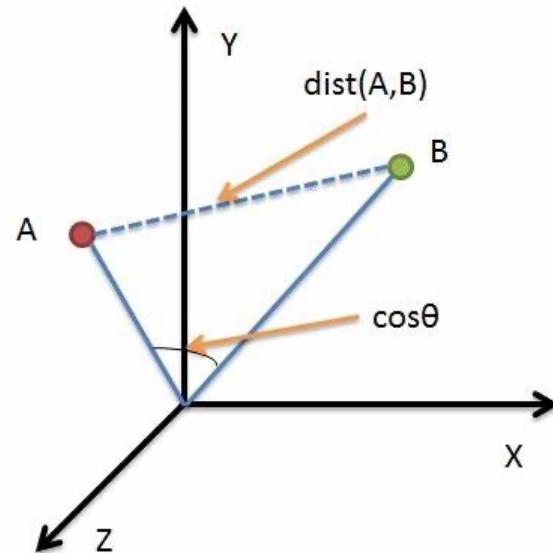
**Q:**  $v(\text{King}) - v(\text{man}) + v(\text{woman}) = ?$



# Similarity of words

- **Cosine similarity** is the most well known to measure how close two vectors are.

$$\text{Cos\_similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$



# Word embedding methods

## SVD-based methods

Based on matrix factorization of a global word co-occurrence matrix.

## Iteration based methods

### **Word2Vec (Mikolov et al. 2013)**

- learn the underlying word representation by using neural networks.
- Two models: CBOW and Skip-gram

### **GloVe (Pennington et al. 2014)**

- Learn word embedding by using a co-occurrence statistics of words in a corpus.

Word2Vec and GloVe are two most popular word embedding methods

# Reference

- Lecture notes CS224D: Deep Learning Part 1:  
[http://cs224d.stanford.edu/lecture\\_notes/notes1.pdf](http://cs224d.stanford.edu/lecture_notes/notes1.pdf)
- Lectures CS224n: NLP processing with Deep Learning.  
<https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/syllabus.html>

# Topic 5: Word Embedding – SVD-Based Methods

- **Step 1:** First loop over a massive dataset and accumulate word co-occurrence matrix  $X$
- **Step 2:** Perform Singular Value Decomposition on  $X$  to get a decomposition.
- **Step 3:** Use the rows of the singular matrix as the word embeddings for all words in our dictionary.

# Window-based co-occurrence matrix

$X_{ij}$ : The number of times each word appears inside a window of a particular size around the word of interest. We calculate this count for all the words in a corpus.

- Example corpus:  
(window size: 1)

- “I like deep learning.”
- “I like NLP.”
- “I enjoy flying.”

	<i>I</i>	<i>like</i>	<i>enjoy</i>	<i>deep</i>	<i>learning</i>	<i>NLP</i>	<i>flying</i>	.
<i>I</i>	0	2	1	0	0	0	0	0
<i>like</i>	2	0	0	1	0	1	0	0
<i>enjoy</i>	1	0	0	0	0	0	1	0
<i>deep</i>	0	1	0	0	1	0	0	0
<i>learning</i>	0	0	0	1	0	0	0	1
<i>NLP</i>	0	1	0	0	0	0	0	1
<i>flying</i>	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

- Single value decomposition (SVD) of co-occurrence matrix  $X$

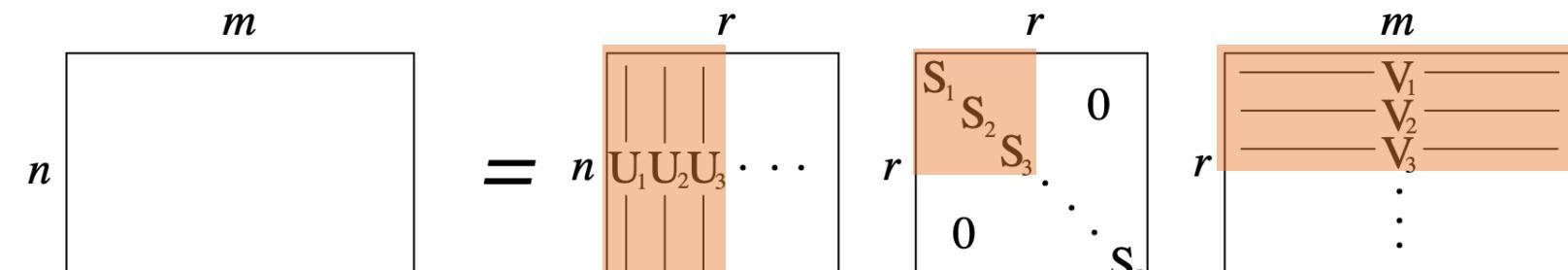
$$X = USV^T$$

$$\begin{matrix} m \\ n \end{matrix} \boxed{X} = \begin{matrix} r \\ n \end{matrix} \boxed{U_1 U_2 U_3 \dots} \begin{matrix} r \\ r \end{matrix} \boxed{S_1 S_2 S_3 \dots} \begin{matrix} 0 \\ \ddots \\ S_r \end{matrix} \begin{matrix} m \\ r \end{matrix} \boxed{V_1 V_2 V_3 \dots}$$

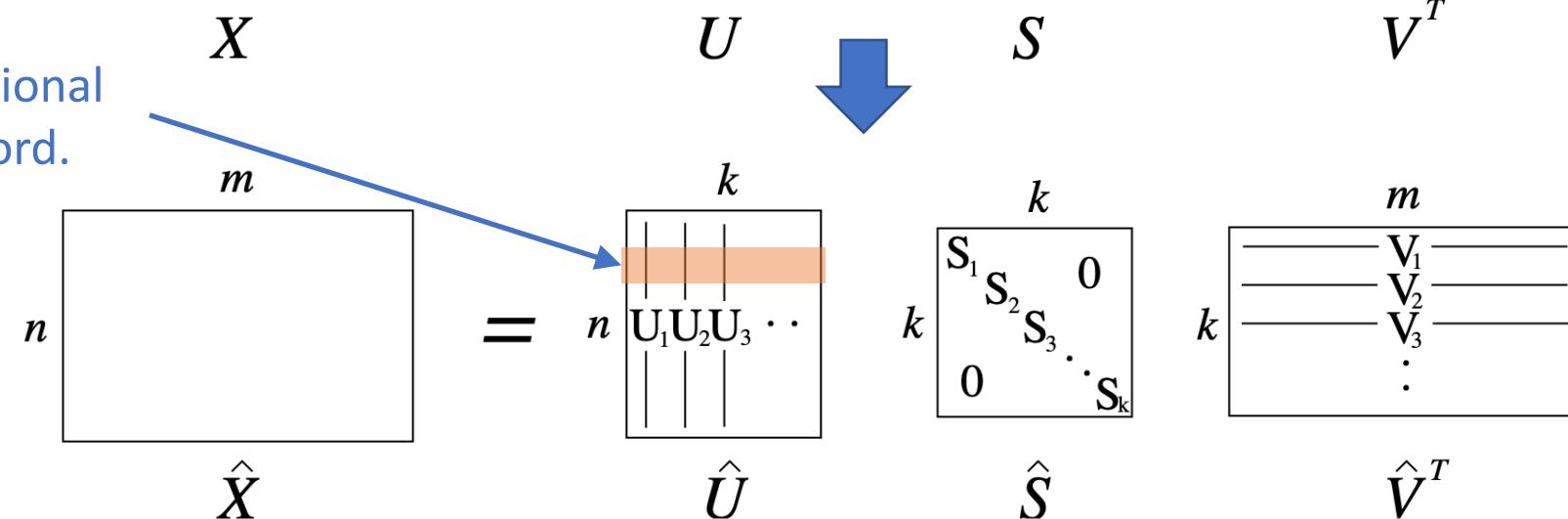
$X$                      $U$                      $S$                      $V^T$

- Single value decomposition (SVD) of co-occurrence matrix  $X$

$$X = USV^T$$



Each row is a  $k$ -dimensional word vector of one word.



$\hat{X}$  is the best  $k$ -rank approximation to  $X$

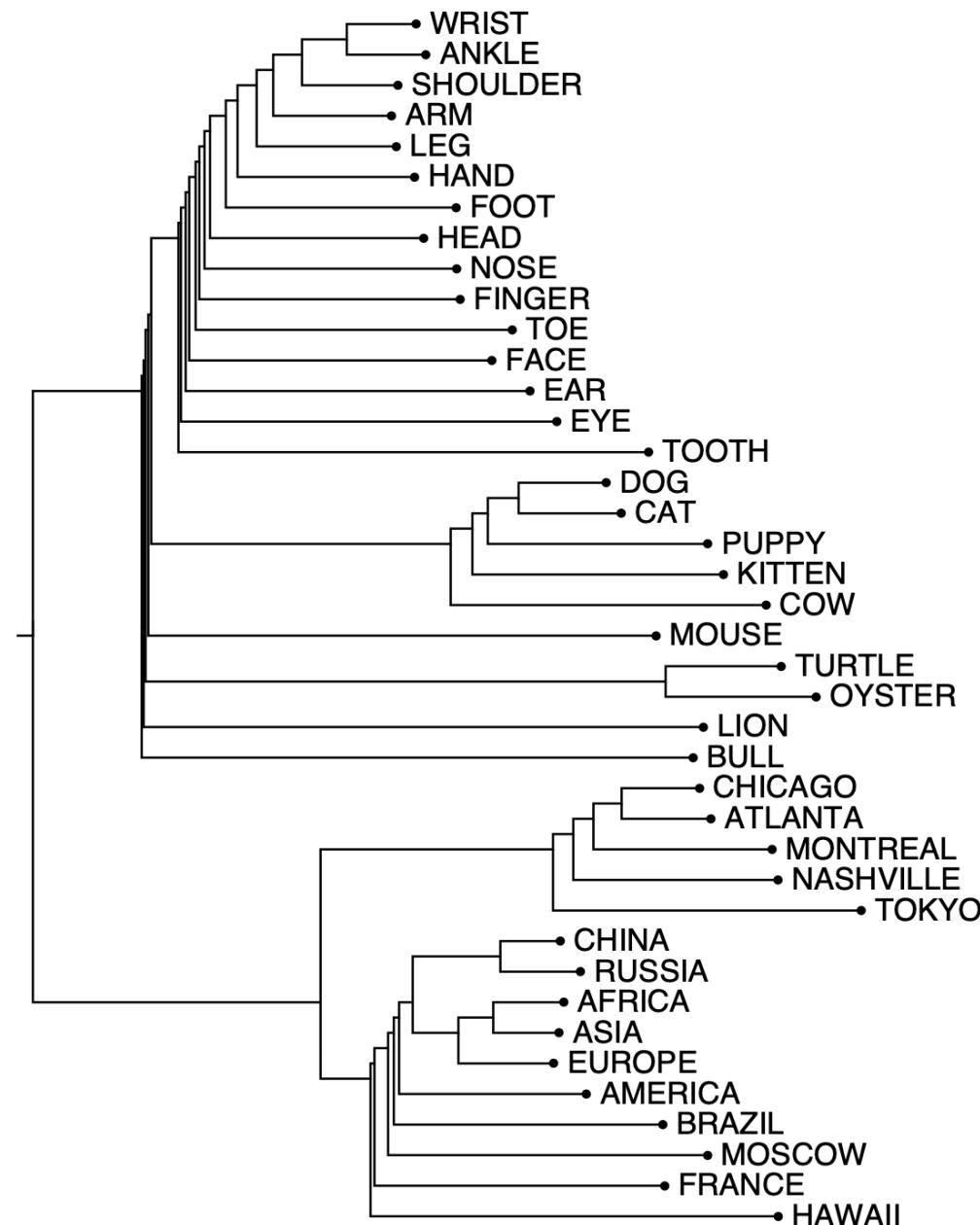
# Problems with SVD-based methods

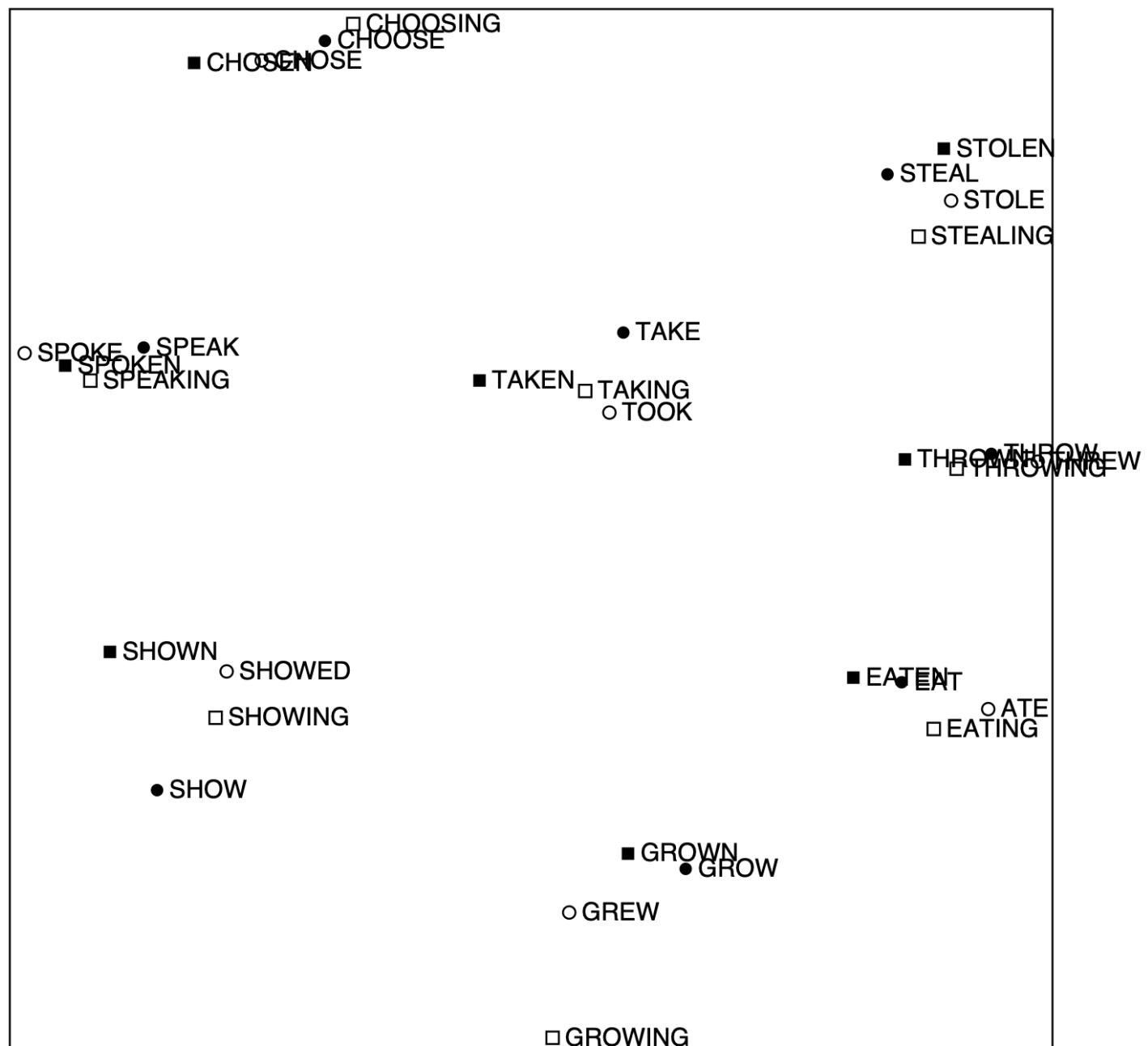
- The matrix is very high dimensional in general
- Quadratic cost to train (i.e. to perform SVD)
- Hard to incorporate new words or documents
- Requires the incorporation of some hacks on  $X$  to account for the drastic imbalance in word frequency

# Hacks on X

- Problem: function words ("the", "he", "has") are too frequent, leading the syntax has too much impact.
  - **Solution 1:** Cap the count  $\min(X, m)$ ,  $m \sim 100$ .
  - **Solution 2:** Ignore function words.
- Apply **a ramp window** – i.e. weight the co-occurrence count based on distance between the words in the document.

(1 2 3 4 0 4 3 2 1)
- Use **Pearson correlations** instead of counts, then set negative counts to 0





# Reference

- Lecture notes CS224D: Deep Learning Part 1:  
[http://cs224d.stanford.edu/lecture\\_notes/notes1.pdf](http://cs224d.stanford.edu/lecture_notes/notes1.pdf)
- Lectures CS224n: NLP processing with Deep Learning.  
<https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/syllabus.html>

# Topic 6: Word Embedding - Word2Vec (SkipGram & CBOW)

I like playing \_\_?

football  
basketball  
tennis  
table tennis  
golf  
chess  
game  
...

Words appear in similar contexts would have  
similar word embedding.

# Word pairs for training

**Example:** “The quick brown fox jumps over the lazy dog.”

Window size: 2

**Skip-gram:** predict the context given the current word

# Word pairs for training

**Example:** “The quick brown fox jumps over the lazy dog.”

Window size: 2

## Source Text

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

Input  $x$    Output  $y$

The quick brown fox jumps over the lazy dog.”

Training Samples

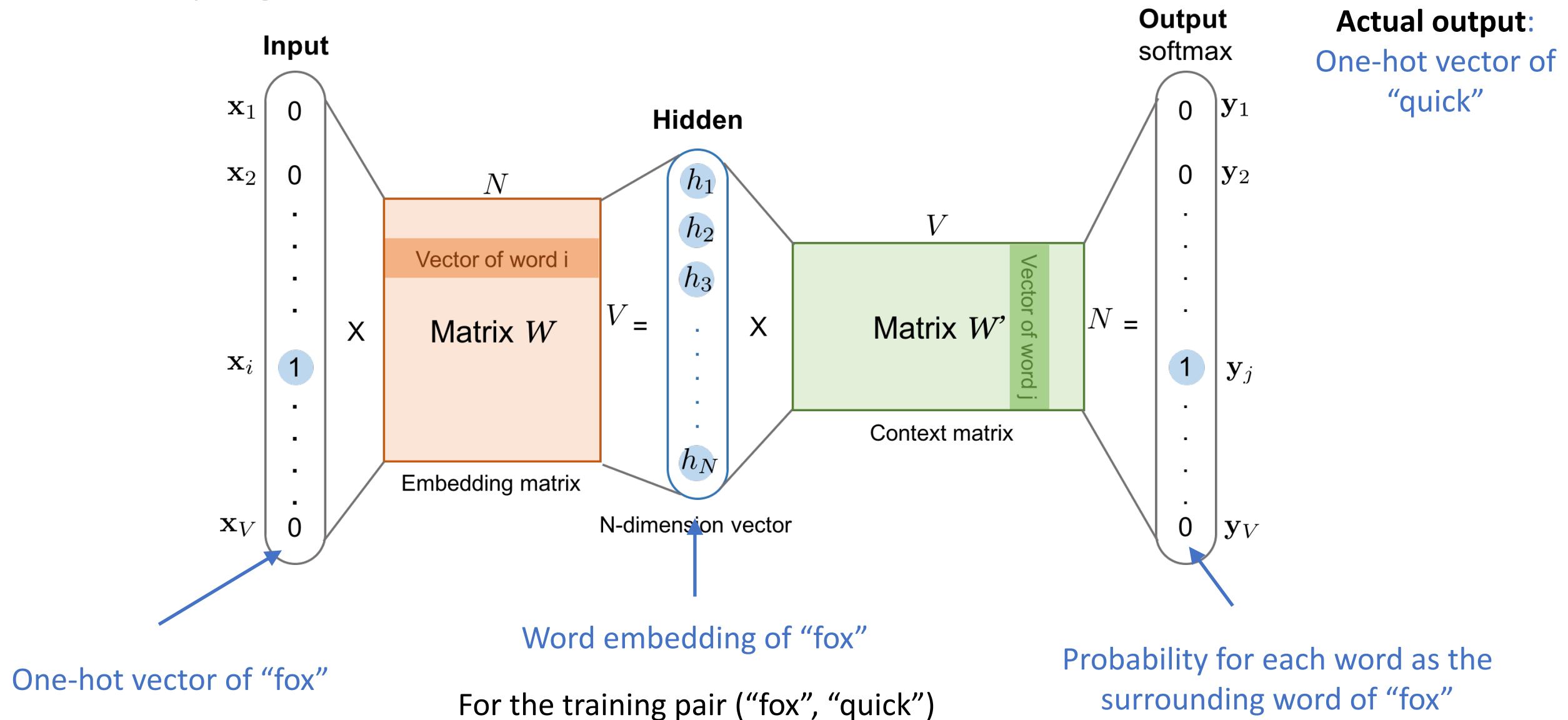
(the, quick)  
(the, brown)

(quick, the)  
(quick, brown)  
(quick, fox)

(brown, the)  
(brown, quick)  
(brown, fox)  
(brown, jumps)

(fox, quick)  
(fox, brown)  
(fox, jumps)  
(fox, over)

# Skip-gram model



# Hidden layer

- Fully connected layer with no activation function

Word embedding of the  
input word  $h \in R^{N \times 1}$

$$h = W^T x$$

Embedding matrix

$W \in R^{V \times N}$

$$\begin{bmatrix} 0 & 0 & 0 & \boxed{1} & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10} & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

# Hidden layer

*Word Vector  
Lookup Table!*

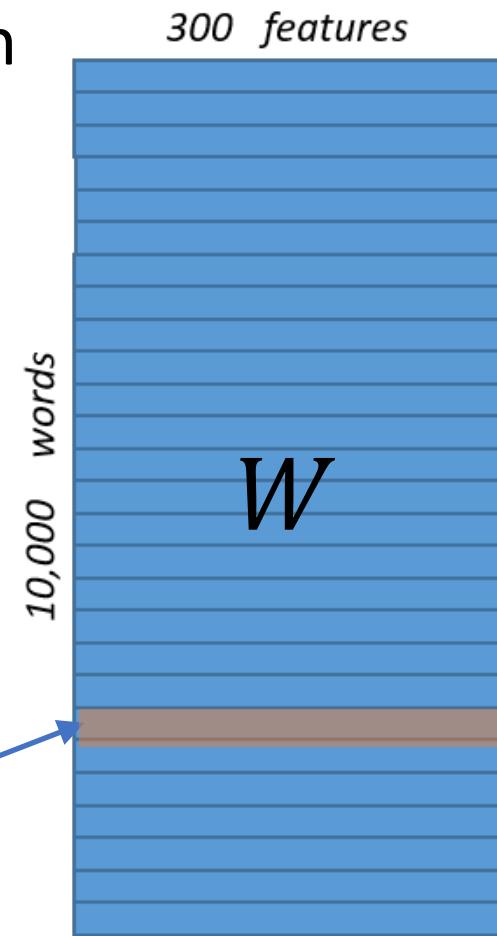
- Fully connected layer with no activation function

Word embedding of the  
input word  $h \in R^{N \times 1}$

$$h = W^T x$$

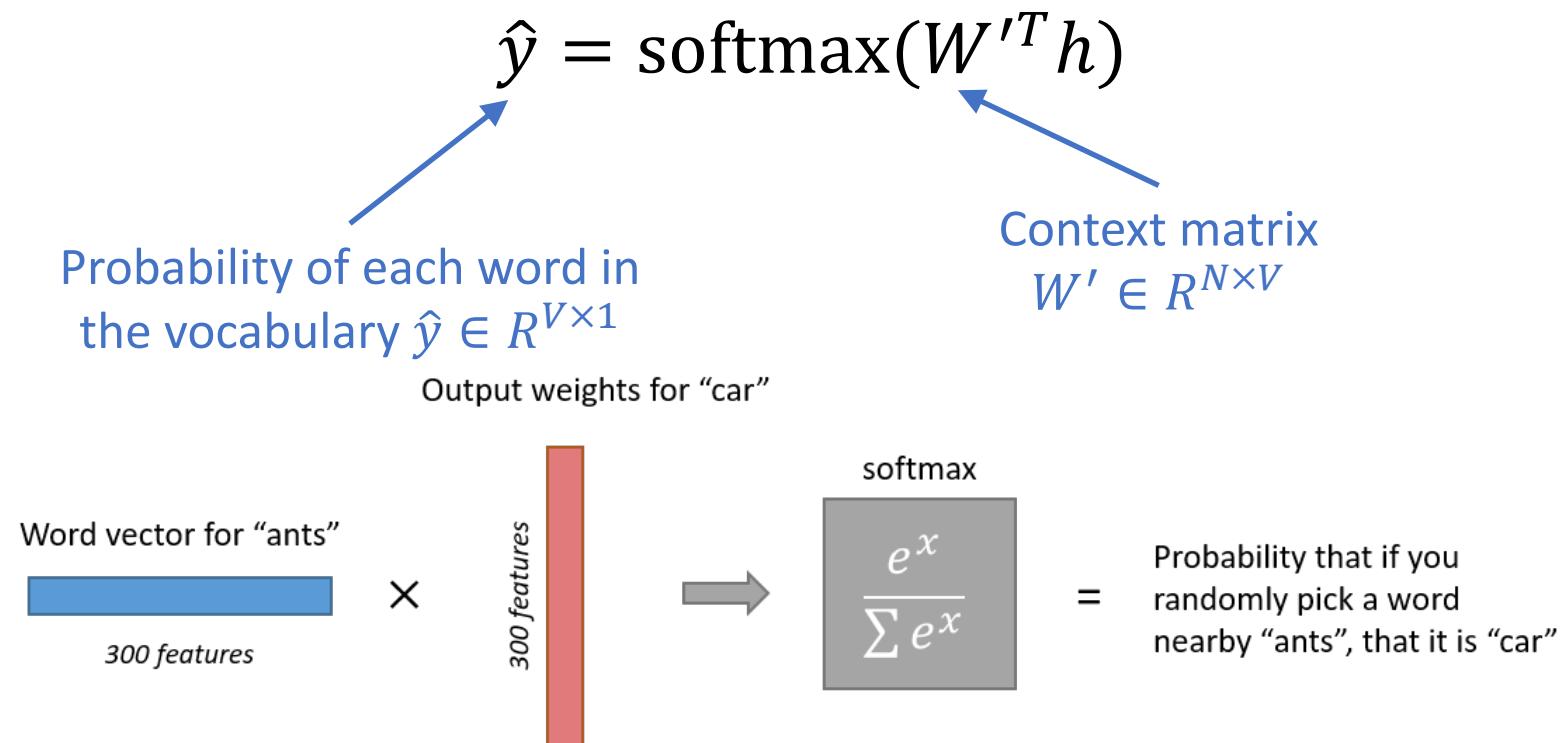
Embedding matrix  
 $W \in R^{V \times N}$

Each row is a  $N$ -dimensional  
word vector of one word.



# Output layer

- Fully connected layer with softmax activation function



# Output layer

- Fully connected layer with softmax activation function

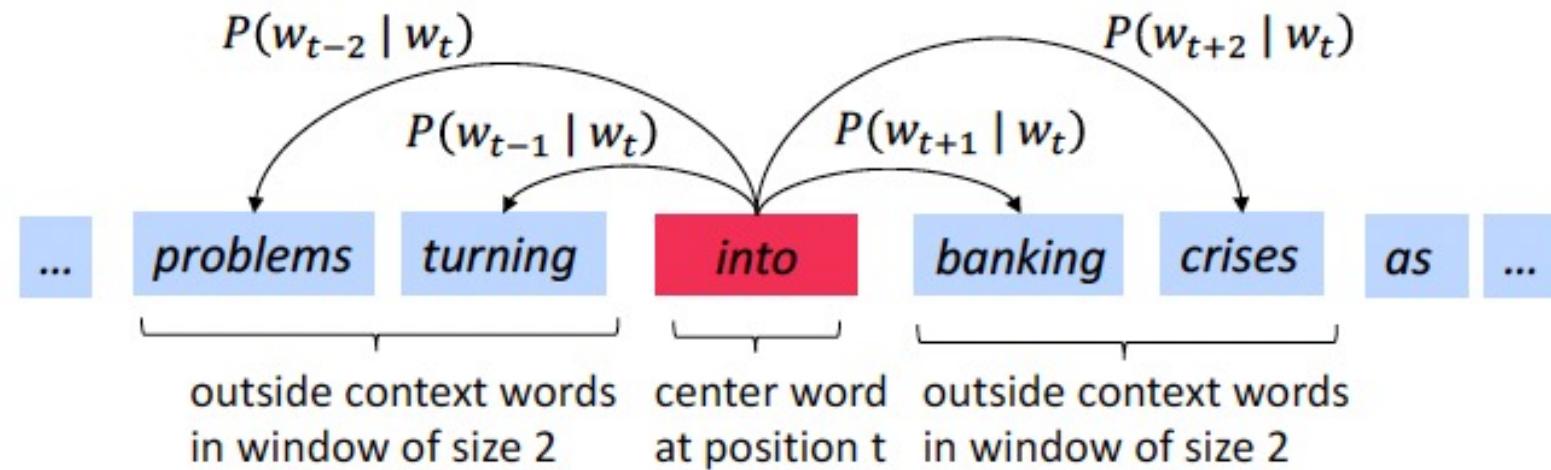
$$\hat{y} = \text{softmax}(W'^T h)$$

Probability of each word in the vocabulary  $\hat{y} \in R^{V \times 1}$

Context matrix  $W' \in R^{N \times V}$

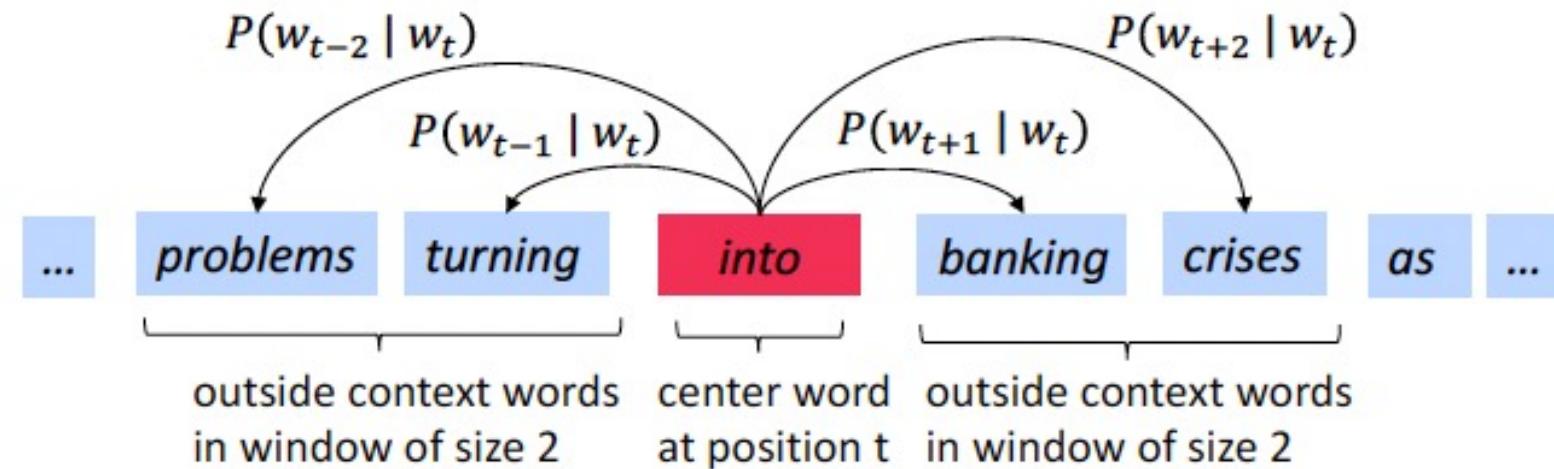
*Let's interpret this process in a statistical aspect ...*

- Try to maximize the probability  $P(w_{t+j} | w_t)$



- For each position  $t$ , maximize the likelihood of the context words within a window of size  $m$ , given the centre word  $w_t$ .

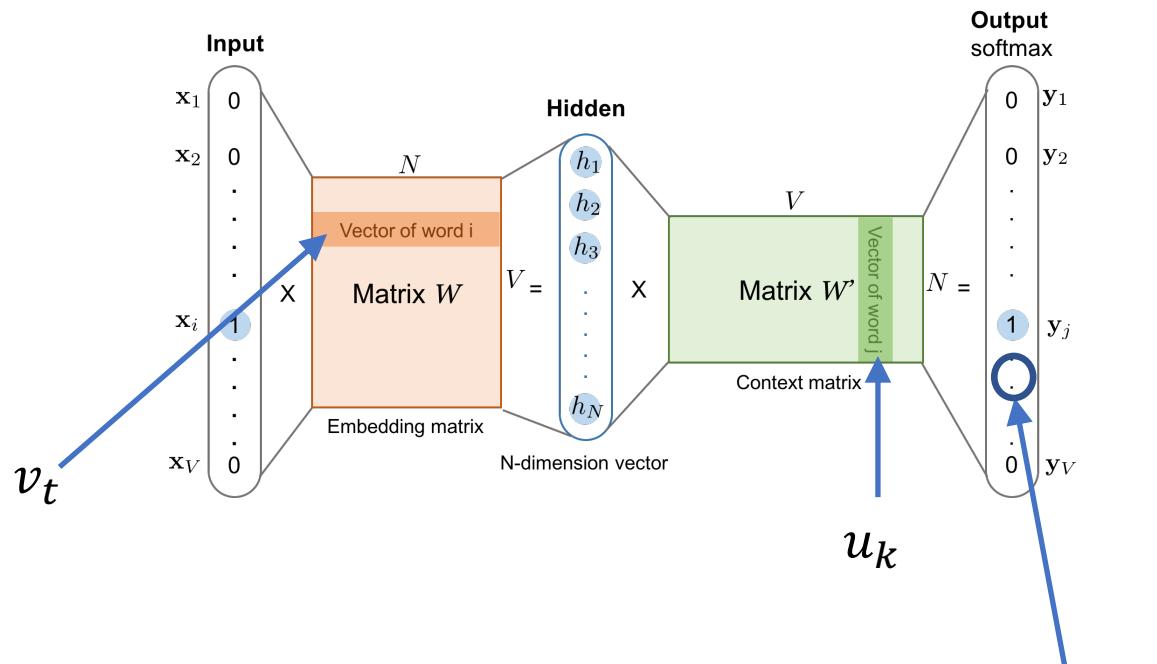
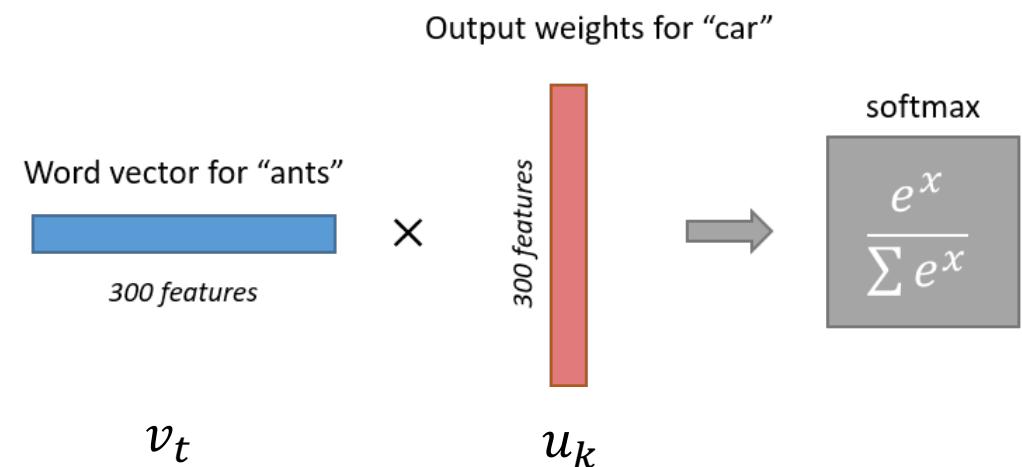
$$P(w_{t-m}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+m} | w_t) = \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} | w_t)$$



## How to calculate $P(w_{t+j}|w_t)$ ?

- $v_t$ : the word vector for the input word  $w_t$
- $u_k$ : the word vector for the output word  $w_k$

$$P(w_{t+j}|w_t) = \frac{\exp(u_{t+j}^T v_t)}{\sum_{k=1}^V \exp(u_k^T v_t)}$$



= Probability that if you randomly pick a word nearby "ants", that it is "car"

# Skip-gram loss function

- Maximize the likelihood of predicting the context words

$$P(w_{t-m}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+m}) = \prod_{-m \leq j \leq m, j \neq 0} \frac{\exp(u_{t+j}^T v_t)}{\sum_{k=1}^V \exp(u_k^T v_t)}$$

↓

- Minimize the negative log likelihood:

$$J = -\log P(w_{t-m}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+m})$$

$$= -\log \prod_{-m \leq j \leq m, j \neq 0} \frac{\exp(u_{t+j}^T v_t)}{\sum_{k=1}^V \exp(u_k^T v_t)}$$

$$= - \sum_{-m \leq j \leq m, j \neq 0} u_{t+j}^T v_t + 2m \log \sum_{k=1}^V \exp(u_k^T v_t)$$

# Skip-gram loss function

- All the parameters  $(v_t, u_t, t = 1, \dots, V)$  will be optimized via Stochastic Gradient Descent (SGD).

$$J = - \sum_{-m \leq j \leq m, j \neq 0} u_{t+j}^T v_t + 2m \log \sum_{k=1}^V \exp(u_k^T v_t)$$



However, since  $V$  is a huge number, it is computationally intensive for every training step.

# Topic 7: Word2Vec with Negative Sampling

# Skip-Grams with Negative Sampling (SGNS)

Marco saw a furry little wampimuk hiding in the tree.

“word2vec Explained...”  
Goldberg & Levy, arXiv 2014

# Skip-Grams with Negative Sampling (SGNS)

Marco saw a furry little **wampimuk** hiding in the tree.

“word2vec Explained...”  
Goldberg & Levy, arXiv 2014

# Skip-Grams with Negative Sampling (SGNS)

Marco saw a **furry little wampimuk hiding in** the tree.

## words

wampimuk

wampimuk

wampimuk

wampimuk

...

## contexts

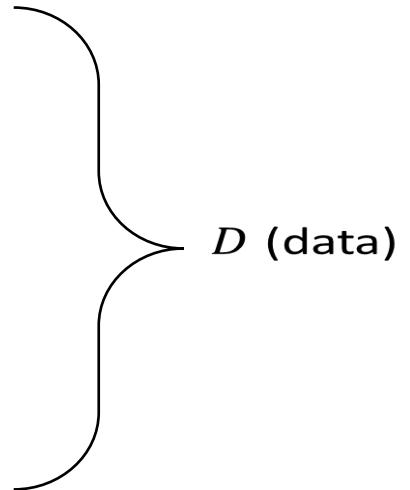
furry

little

hiding

in

...



“word2vec Explained...”  
Goldberg & Levy, arXiv 2014

# Skip-Grams with Negative Sampling (SGNS)

- **Maximize:**  $\sigma(\vec{w} \cdot \vec{c})$ 
  - $c$  was **observed** with  $w$

<u>words</u>	<u>contexts</u>
wampimuk	furry
wampimuk	little
wampimuk	hiding
wampimuk	in

“word2vec Explained...”  
Goldberg & Levy, arXiv 2014

# Skip-Grams with Negative Sampling (SGNS)

- **Maximize:**  $\sigma(\vec{w} \cdot \vec{c})$ 
  - $c$  was **observed** with  $w$

## words

wampimuk  
wampimuk  
wampimuk  
wampimuk

## contexts

furry  
little  
hiding  
in

- **Minimize:**  $\sigma(\vec{w} \cdot \vec{c}')$ 
  - $c'$  was **hallucinated** with  $w$

## words

wampimuk  
wampimuk  
wampimuk  
wampimuk

## contexts

Australia  
cyber  
the  
1985

# Negative Sampling

- **Idea:** randomly select just a small number of “negative” words (say 5) to update the weights for.

$300 \times 10,000$



$300 \times 5$

Context matrix  
 $W' \in R^{N \times V}$

Context matrix with  
NS:  $W' \in R^{N \times M}$

For every training step, only modify a small percentage of weights,  
rather than the big weight matrix for all training samples.

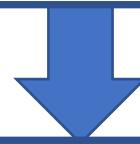
# Negative Sampling

## Basic Word2Vec

A multi-class classification problem

Predict probability of each word being a nearby word

$$P(\text{quick}|\text{fox}) = \text{softmax}\left(u_{\text{quick}}^T v_{\text{fox}}\right)$$



## Word2Vec with Negative Sampling

A binary classification problem

Predict probability of two words are neighbors

$$P(D = 1 | \text{fox}, \text{quick}) = \sigma\left(u_{\text{quick}}^T v_{\text{fox}}\right)$$

(fox, quick)	1
(fox, jumps)	1
(fox, brown)	1
(fox, chair)	0
(fox, artificial)	0
(fox, kiwi)	0
(fox, sign)	0

$$J_{NS} = -\log(\sigma(u_o^T v_c)) - \sum_{k=1}^K \log(\sigma(-u_k^T v_c))$$

$\sigma(\cdot)$  is the sigmoid function

$u_k$  is a negative sample of word  $v_c$

Small K (around 2 to 5) for large dataset. Large K (around 5 to 20) for smaller dataset.

# Selecting Negative Samples

- $u_k$  is sampled from  $P(w) = U(w)^{3/4}$ , where  $U(w)$  is a unigram distribution

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^n (f(w_j)^{3/4})}$$

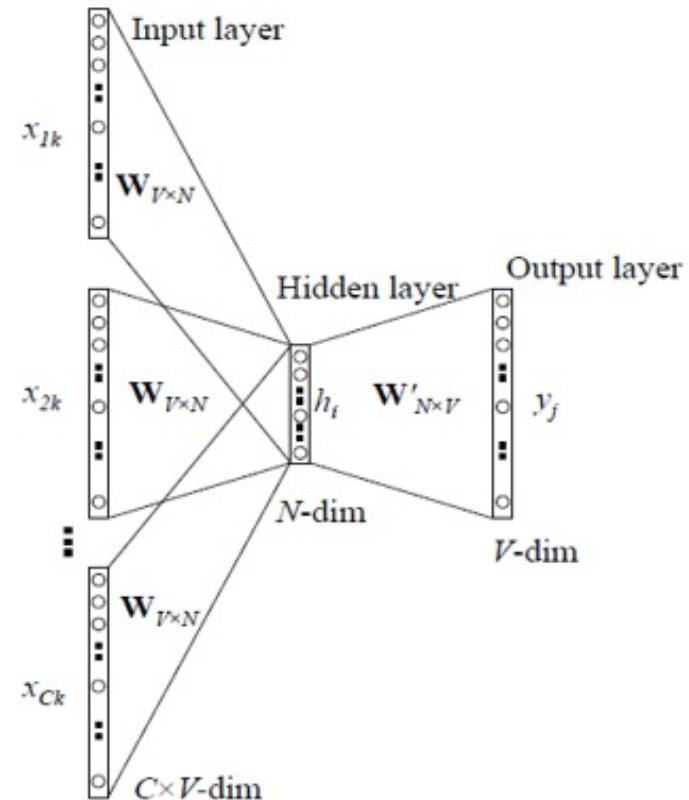
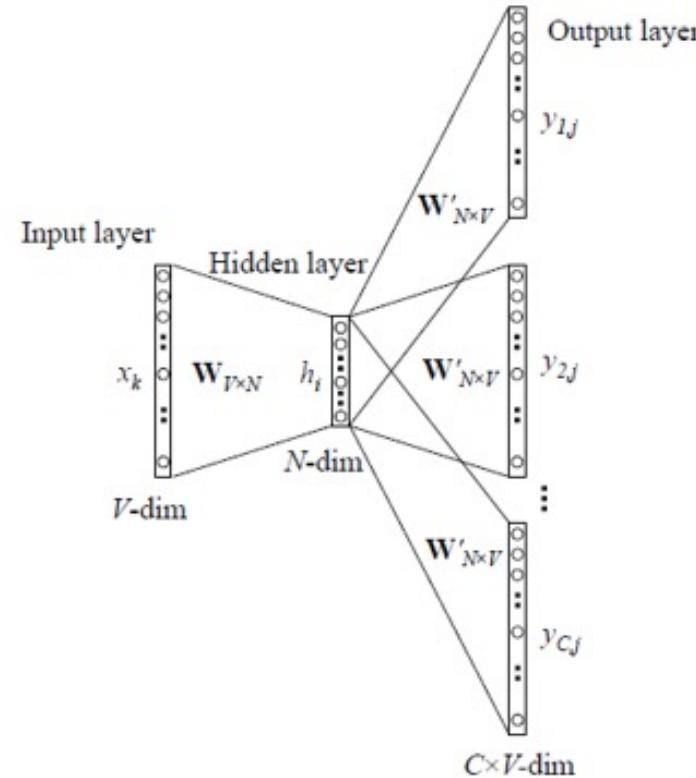
- The  $\frac{3}{4}$  power makes less frequent word be sampled more often.

“is”:  $0.9^{3/4} = 0.92$

“constitution”:  $0.09^{3/4} = 0.16$

“bombastic”:  $0.01^{3/4} = 0.032$

# Skip-gram vs CBOW (Continuous BoW)



**Skip-gram:** predict the context given the current word

I like playing football

**CBOW:** predict the current word using its context

I like playing football

# Reference

- Lecture notes CS224D: Deep Learning Part:  
[http://cs224d.stanford.edu/lecture\\_notes/notes1.pdf](http://cs224d.stanford.edu/lecture_notes/notes1.pdf)
- Ria Kushrestha's blog: NLP 102: Negative Sampling and GloVe  
<https://towardsdatascience.com/nlp-101-negative-sampling-and-glove-936c88f3bc68>
- Chris McCormick's blog: Word2Vec Tutorial – The Skip-Gram Model.  
<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>
- Chris McCormick's blog: Word2Vec Tutorial Part 2 – Negative sampling.  
<http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/>

# Topic 8: Word Embedding

## - GloVe

# GloVe

- Proposed by Pennington et al. 2014.
- Unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence matrix) to obtain word vectors.



## GloVe

Counts the co-occurrence between words  $w_k, w_i$ 

$$P(w_k|w_i) = \frac{C(w_k, w_i)}{C(w_i)}$$

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k steam)$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k ice)/P(k steam)$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

“solid” is more related to  
“ice” than “steam”

$$F(w_i, w_j, w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

“Water” is equally (ir)relevant to  
“ice” and “steam”

**Intuition:** co-occurrence probabilities ratios gathers more information than the raw probabilities and better capture relevant information about words’ relationship

# What is the function $F(\cdot)$ ?

$$F(w_i, w_j, w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

Since the goal is to learn meaningful word vectors,  $F(\cdot)$  is designed to be a function of the linear difference between two words  $w_i$  and  $w_j$

$$F((w_i - w_j)^T w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

The final solution is to  $F(\cdot)$  as an **exponential** function.

$$F((w_i - w_j)^T w_k) = \exp((w_i - w_j)^T w_k) = \frac{\exp(w_i^T w_k)}{\exp(w_j^T w_k)} = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

# Loss function of GloVe

$$w_i^T w_k = \log \frac{C(w_k, w_i)}{C(w_i)} = \log C(w_k, w_i) - \log C(w_i)$$

Replace it with a bias term  $b_i$

$$\log C(w_k, w_i) = w_i^T w_k + b_i + b_k$$

To keep the symmetric form, we also add in a bias term  $b_k$

The loss function for the GloVe model is designed to preserve the above formula by minimizing the sum of the squared errors

$$\mathcal{L} = \sum_{i=1, j=1}^V (w_i^T w_j + b_i + b_j - \log C(w_j, w_i))^2$$



Add a weighting function  $f()$  that is used to downweight the importance of very frequent co-occurrences

$$\mathcal{L} = \sum_{i=1, j=1}^V f(C(w_j, w_i))(w_i^T w_j + b_i + b_j - \log C(w_j, w_i))^2$$

# Loss function of GloVe

To penalize the difference between the dot product of two word vectors and the logarithm of the co-occurrence count, with the bias terms added.

$$\mathcal{L} = \sum_{i=1, j=1}^V f(C(w_j, w_i))(w_i^T w_j + b_i + b_j - \log C(w_j, w_i))^2$$

$w_i, w_j$  are the word vectors for words i and j

$b_i, b_j$  are bias terms

$$f(c) = \begin{cases} \left(\frac{c}{c_{\max}}\right)^{\alpha} & \text{if } c < c_{\max}, c_{\max} \text{ is adjustable.} \\ 1 & \text{if otherwise} \end{cases}$$

Nearest words to  
frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus



*litoria*



*rana*



*leptodactylidae*



*eleutherodactylus*

# Advantages of GloVe

- Computationally efficient
- Scales well to large datasets
- Produces embedding that capture both syntactic and semantic relationships between words.

# GloVe vs. Word2Vec

- Both are popular algorithms for generating word embeddings.
- Both are unsupervised learning algorithms
- Both are able to capture semantic relationships between words.

## **Word2Vec**

Use a neural network to learn embedding  
Focus more on local context  
Is able to handle larger corpora of text

## **GloVe**

Based on co-occurrence matrix  
Capture global relationships between words.  
Is generally faster than Word2Vec

# Reference

- Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).  
<https://nlp.stanford.edu/projects/glove/>
- Lectures of CS224n: NLP with Deep learning.  
<https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/syllabus.html>
- Matyas Amrouche's blog: Word embedding (Part II).  
<https://towardsdatascience.com/word-embedding-part-ii-intuition-and-some-maths-to-understand-end-to-end-glove-model-9b08e6bf5c06>
- Lilian Weng's blog: Learning Word Embedding.  
<https://lilianweng.github.io/posts/2017-10-15-word-embedding/>

# Week 10: Natural Language Processing (NLP)

Vinay P. Namboodiri

# Topic 8: Word Embedding

## - GloVe

# GloVe

- Proposed by Pennington et al. 2014.
- Unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence matrix) to obtain word vectors.



# GloVe

Counts the co-occurrence between words  $w_k, w_i$

$$P(w_k|w_i) = \frac{C(w_k, w_i)}{C(w_i)}$$

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k steam)$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k ice)/P(k steam)$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

“solid” is more related to  
“ice” than “steam”

$$F(w_i, w_j, w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

“Water” is equally (ir)relevant to  
“ice” and “steam”

**Intuition:** co-occurrence probabilities ratios gathers more information than the raw probabilities and better capture relevant information about words’ relationship

# What is the function $F(\cdot)$ ?

$$F(w_i, w_j, w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

Since the goal is to learn meaningful word vectors,  $F(\cdot)$  is designed to be a function of the linear difference between two words  $w_i$  and  $w_j$

$$F((w_i - w_j)^T w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

The final solution is to  $F(\cdot)$  as an **exponential** function.

$$F((w_i - w_j)^T w_k) = \exp((w_i - w_j)^T w_k) = \frac{\exp(w_i^T w_k)}{\exp(w_j^T w_k)} = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

# Loss function of GloVe

$$w_i^T w_k = \log \frac{C(w_k, w_i)}{C(w_i)} = \log C(w_k, w_i) - \log C(w_i)$$

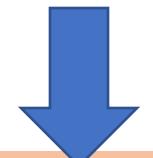
$$\log C(w_k, w_i) = w_i^T w_k + b_i + b_k$$

Replace it with a bias term  $b_i$   


To keep the symmetric form,  
 we also add in a bias term  $b_k$   


The loss function for the GloVe model is designed to preserve the above formula by minimizing the sum of the squared errors

$$\mathcal{L} = \sum_{i=1, j=1}^V (w_i^T w_j + b_i + b_j - \log C(w_j, w_i))^2$$



Add a weighting function  $f()$  that is used to downweight the importance of very frequent co-occurrences

$$\mathcal{L} = \sum_{i=1, j=1}^V f(C(w_j, w_i))(w_i^T w_j + b_i + b_j - \log C(w_j, w_i))^2$$

# Loss function of GloVe

To penalize the difference between the dot product of two word vectors and the logarithm of the co-occurrence count, with the bias terms added.

$$\mathcal{L} = \sum_{i=1, j=1}^V f(C(w_j, w_i))(w_i^T w_j + b_i + b_j - \log C(w_j, w_i))^2$$

$w_i, w_j$  are the word vectors for words i and j       $b_i, b_j$  are bias terms

$$f(c) = \begin{cases} \left(\frac{c}{c_{\max}}\right)^{\alpha} & \text{if } c < c_{\max}, c_{\max} \text{ is adjustable.} \\ 1 & \text{if otherwise} \end{cases}$$

Nearest words to  
frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus



*litoria*



*rana*



*leptodactylidae*



*eleutherodactylus*

# Advantages of GloVe

- Computationally efficient
- Scales well to large datasets
- Produces embedding that capture both syntactic and semantic relationships between words.

# GloVe vs. Word2Vec

- Both are popular algorithms for generating word embeddings.
- Both are unsupervised learning algorithms
- Both are able to capture semantic relationships between words.

## **Word2Vec**

Use a neural network to learn embedding  
Focus more on local context  
Is able to handle larger corpora of text

## **GloVe**

Based on co-occurrence matrix  
Capture global relationships between words.  
Is generally faster than Word2Vec

# Reference

- Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).  
<https://nlp.stanford.edu/projects/glove/>
- Lectures of CS224n: NLP with Deep learning.  
<https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/syllabus.html>
- Matyas Amrouche's blog: Word embedding (Part II).  
<https://towardsdatascience.com/word-embedding-part-ii-intuition-and-some-maths-to-understand-end-to-end-glove-model-9b08e6bf5c06>
- Lilian Weng's blog: Learning Word Embedding.  
<https://lilianweng.github.io/posts/2017-10-15-word-embedding/>

# Week 11:Domain Adaptation

Vinay P. Namboodiri

# Introduction

- So far, most of the techniques we have considered assume the availability of full supervision for training through a training dataset
- However, in many practical scenarios, this is not true
- For instance, for an autonomous car, you may have trained a pedestrian detection algorithm in summer
- Then it snows.....

# Problem

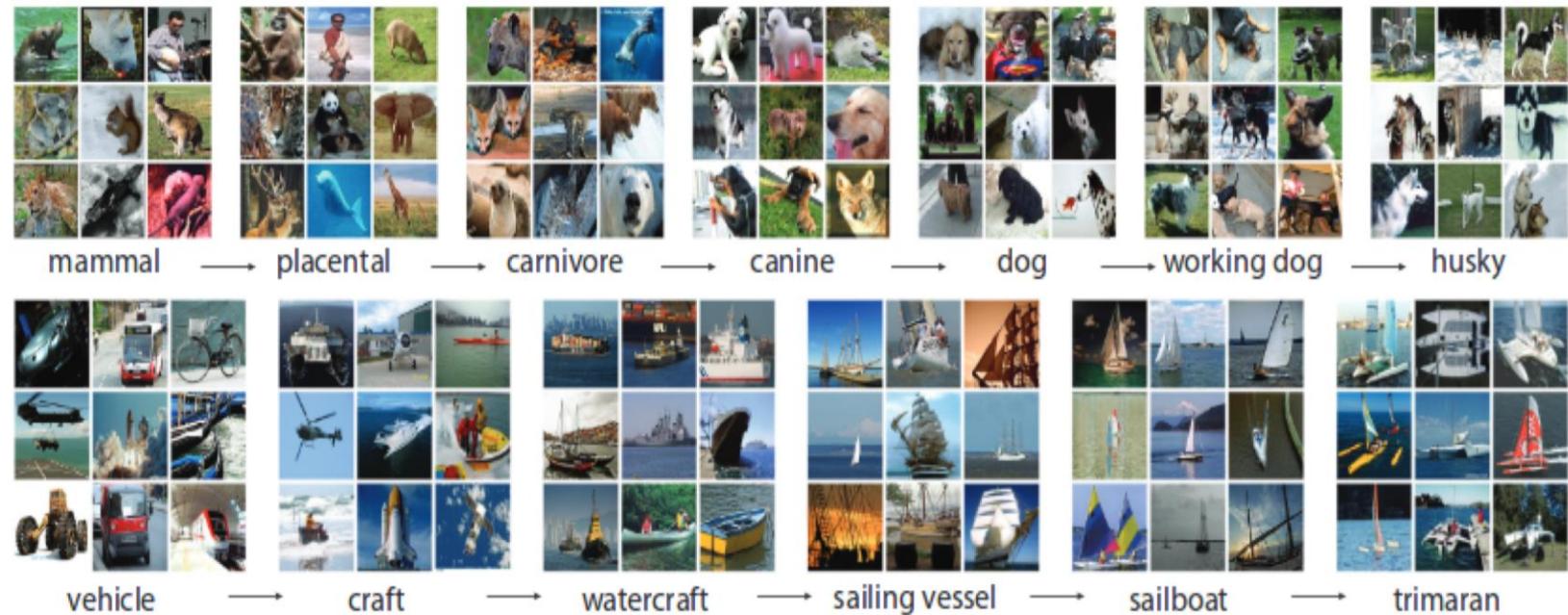


Figure : Typical Computer Vision Dataset

Generally, training and test instances are chosen from the same dataset and hence they are from same probability distribution. Also labels are free of noise, objects are centered and background clutter is less.

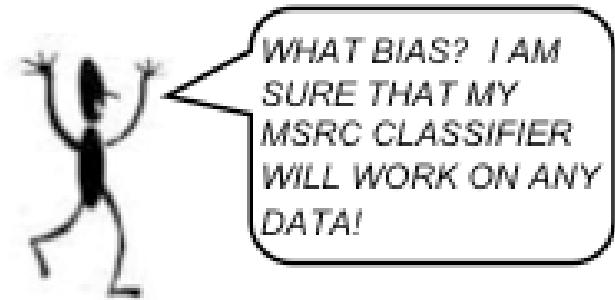
# Challenge



Figure : Real World Computer Vision Dataset

Real world data is noisy. Multiple instances of different object can be present in an image and also usually much more clutter is there.

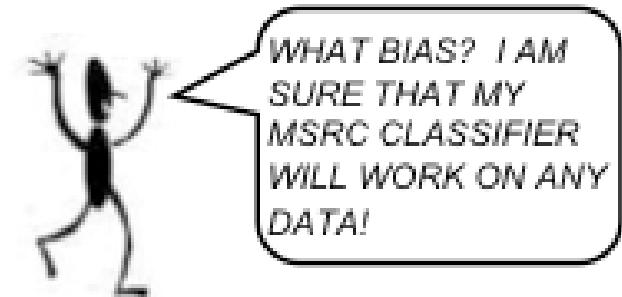
# Problem



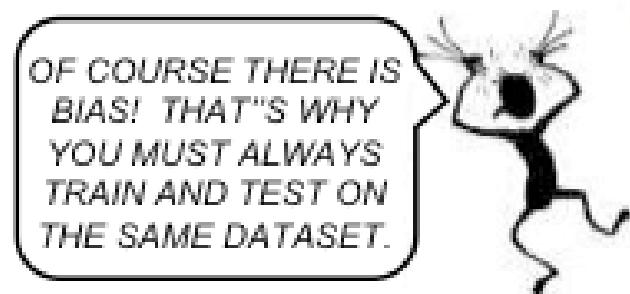
1. Denial

Source: Torralba and Efros  
<http://people.csail.mit.edu/torralba/research/bias/>

# Problem



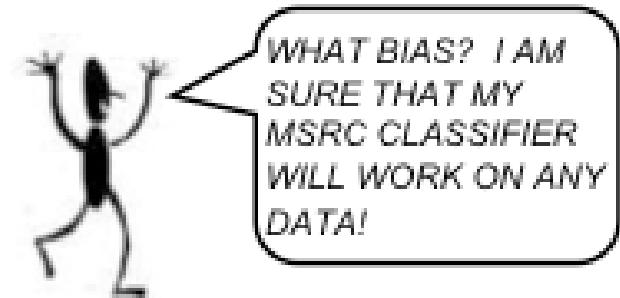
1. Denial



2. Machine Learning

Source: Torralba and Efros  
<http://people.csail.mit.edu/torralba/research/bias/>

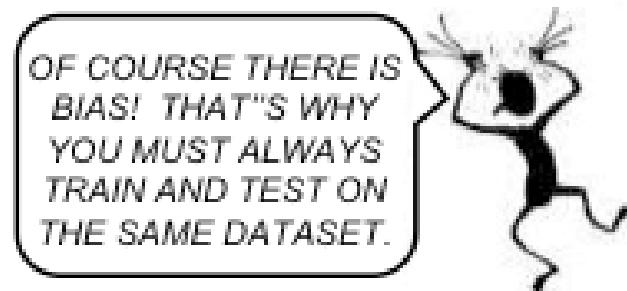
# Problem



1. Denial



3. Despair



2. Machine Learning

# Problem



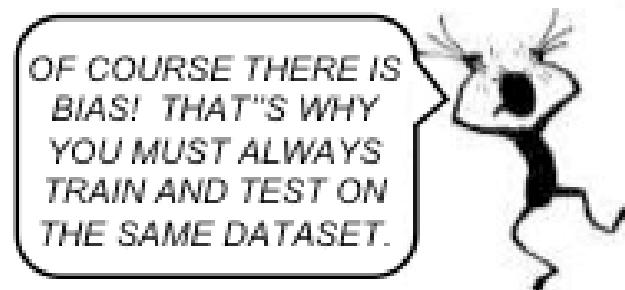
1. Denial

WHAT BIAS? I AM SURE THAT MY MSRC CLASSIFIER WILL WORK ON ANY DATA!

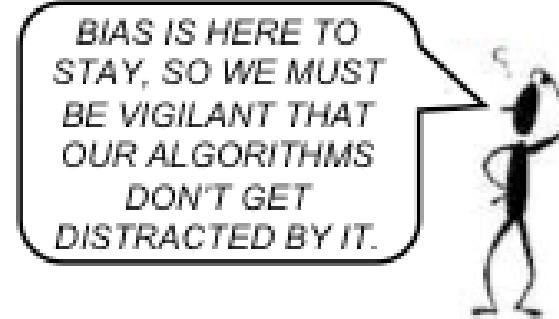


3. Despair

RECOGNITION IS HOPELESS., IT WILL NEVER WORK. WE WILL JUST KEEP OVERFITTING TO THE NEXT DATASET...



2. Machine Learning

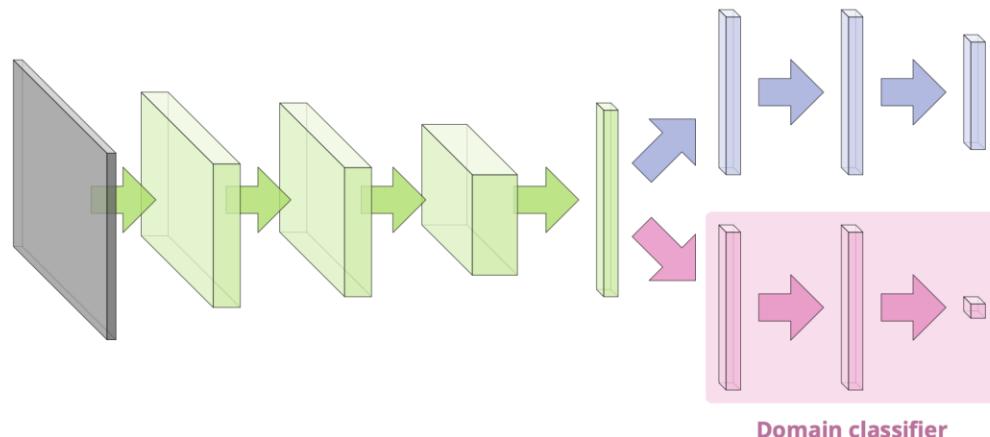


4. Acceptance

# Deep Learning-based Domain Adaptation



# Deep Unsupervised domain adaptation by back propagation



- Computes  $d = G_d(\mathbf{f}; \theta_d)$
- Is trained to predict **0** for **source** and **1** for **target**
- Therefore, the domain loss



Fig. credit: Yaroslav Ganin  
Yaroslav Ganin and Viktor Lempitsky  
Unsupervised domain adaptation by back propagation  
ICML 2015

# Assumptions

- There are
- lots of labeled examples in source domain
- lots of unlabelled examples in target domain
- We want a deep neural network that does well on target domain

# Deep Unsupervised domain adaptation

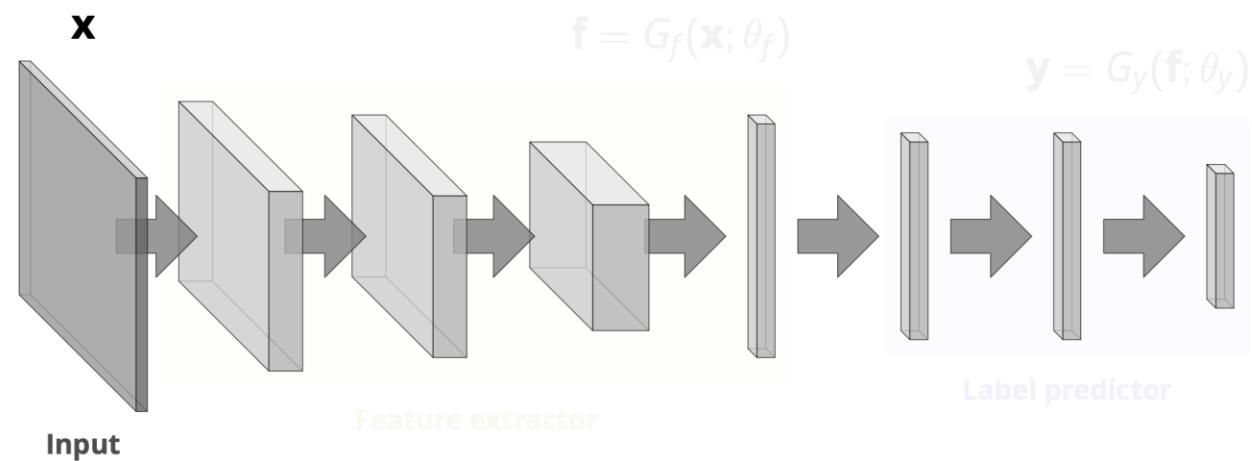


Fig. credit: Yaroslav Ganin  
Yaroslav Ganin and Viktor Lempitsky  
Unsupervised domain adaptation by back propagation  
ICML 2015

# Deep Unsupervised domain adaptation

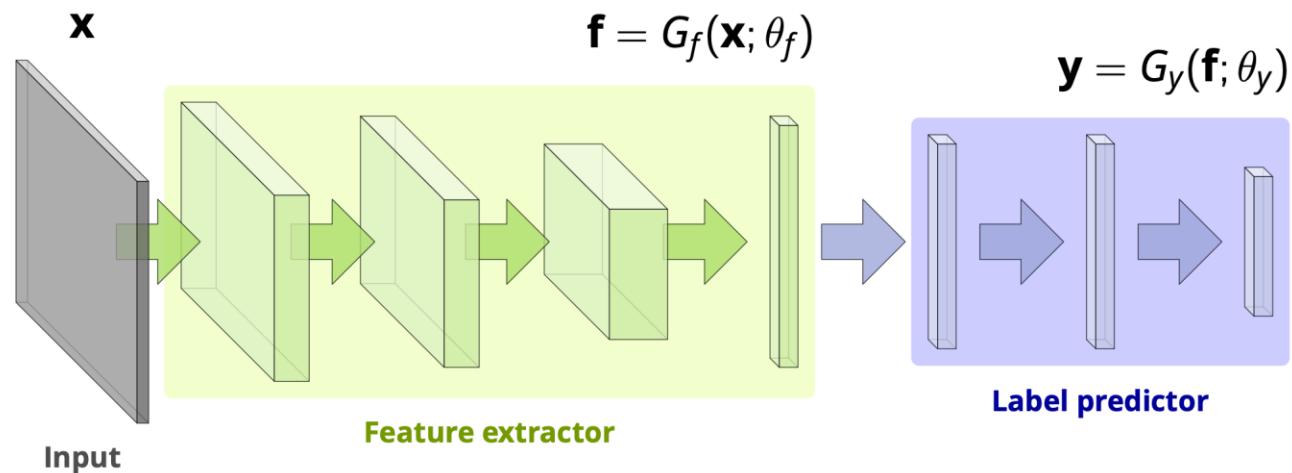
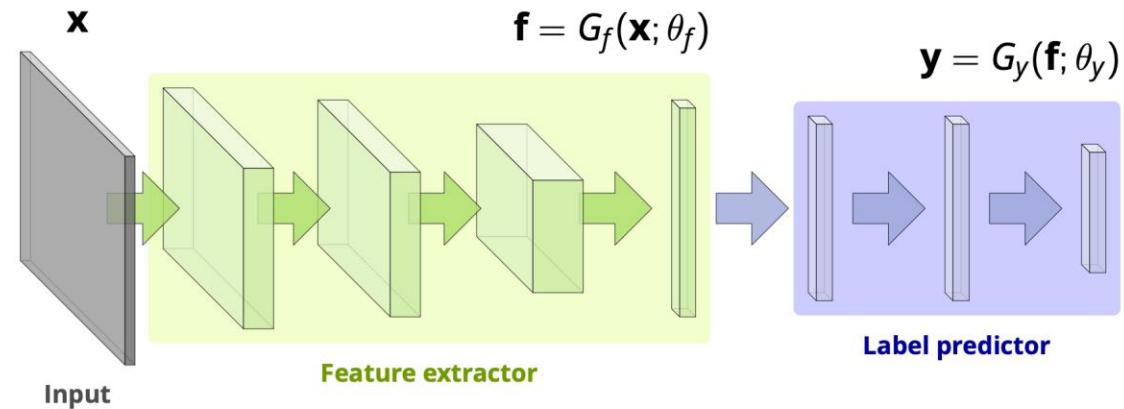


Fig. credit: Yaroslav Ganin  
 Yaroslav Ganin and Viktor Lempitsky  
 Unsupervised domain adaptation by back propagation  
 ICML 2015

# Deep Unsupervised domain adaptation



When trained on **source only**,  
feature distributions **do not**  
**match**.

$$S(\mathbf{f}) = \{G_f(\mathbf{x}; \theta_f) \mid \mathbf{x} \sim S(\mathbf{x})\}$$

$$T(\mathbf{f}) = \{G_f(\mathbf{x}; \theta_f) \mid \mathbf{x} \sim T(\mathbf{x})\}$$

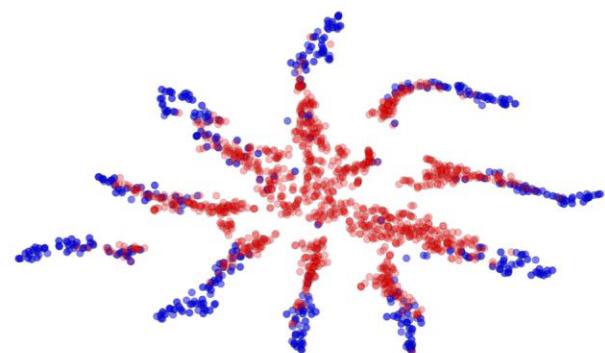


Fig. credit: Yaroslav Ganin  
 Yaroslav Ganin and Viktor Lempitsky  
 Unsupervised domain adaptation by back propagation  
 ICML 2015

# Deep Unsupervised domain adaptation



When trained on **source only**,  
feature distributions **do not**  
**match**.

**Our goal is to get this:**

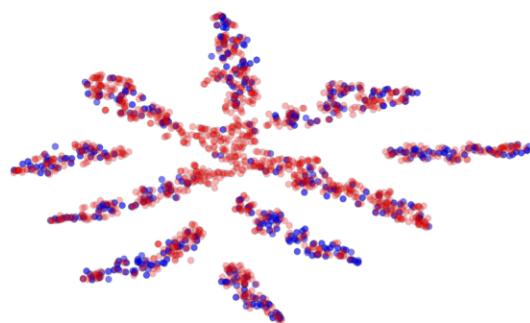


Fig. credit: Yaroslav Ganin  
Yaroslav Ganin and Viktor Lempitsky  
Unsupervised domain adaptation by back propagation  
ICML 2015

# Deep Unsupervised domain adaptation

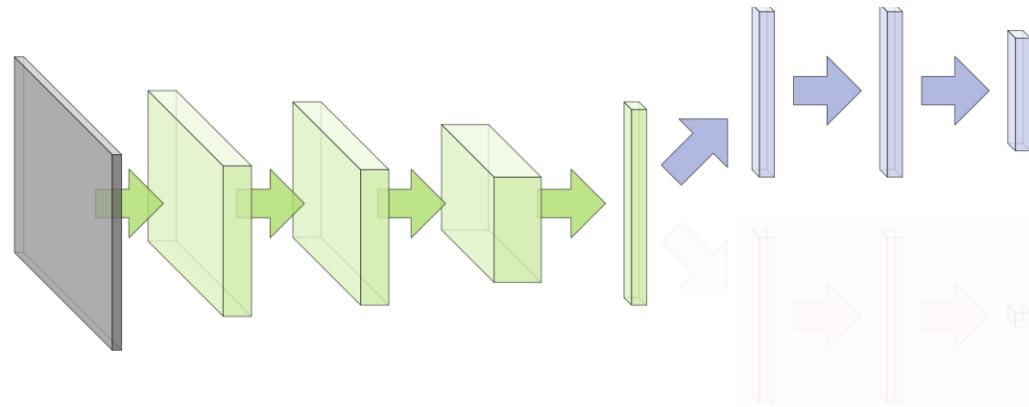
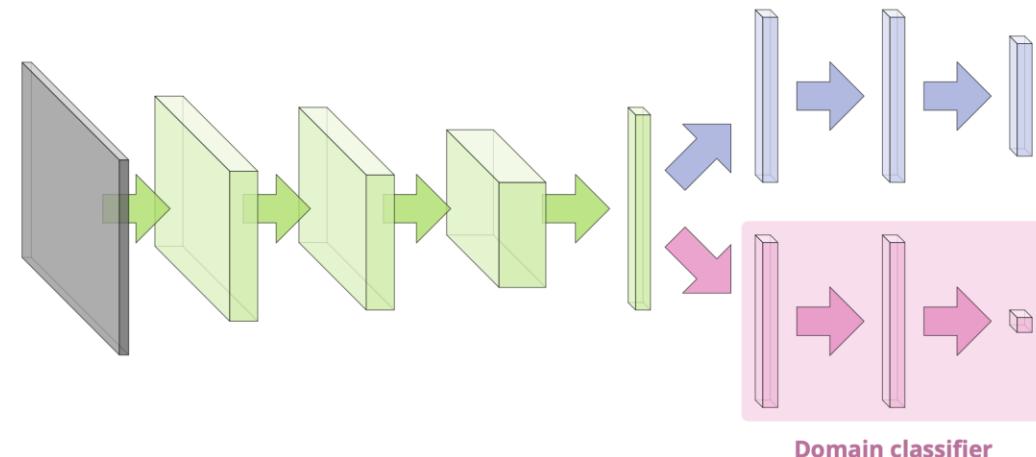


Fig. credit: Yaroslav Ganin  
Yaroslav Ganin and Viktor Lempitsky  
Unsupervised domain adaptation by back propagation  
ICML 2015

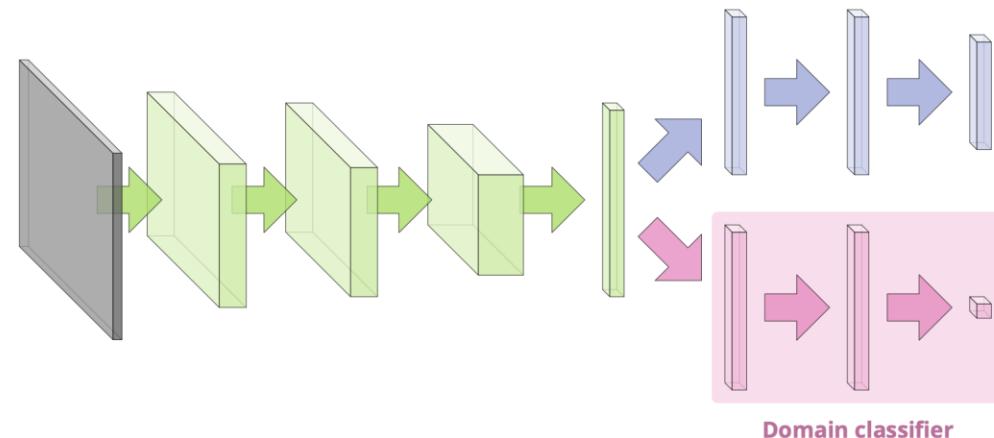
# Deep Unsupervised domain adaptation



■ Computes  $d = G_d(\mathbf{f}; \theta_d)$

Fig. credit: Yaroslav Ganin  
Yaroslav Ganin and Viktor Lempitsky  
Unsupervised domain adaptation by back propagation  
ICML 2015

# Deep Unsupervised domain adaptation

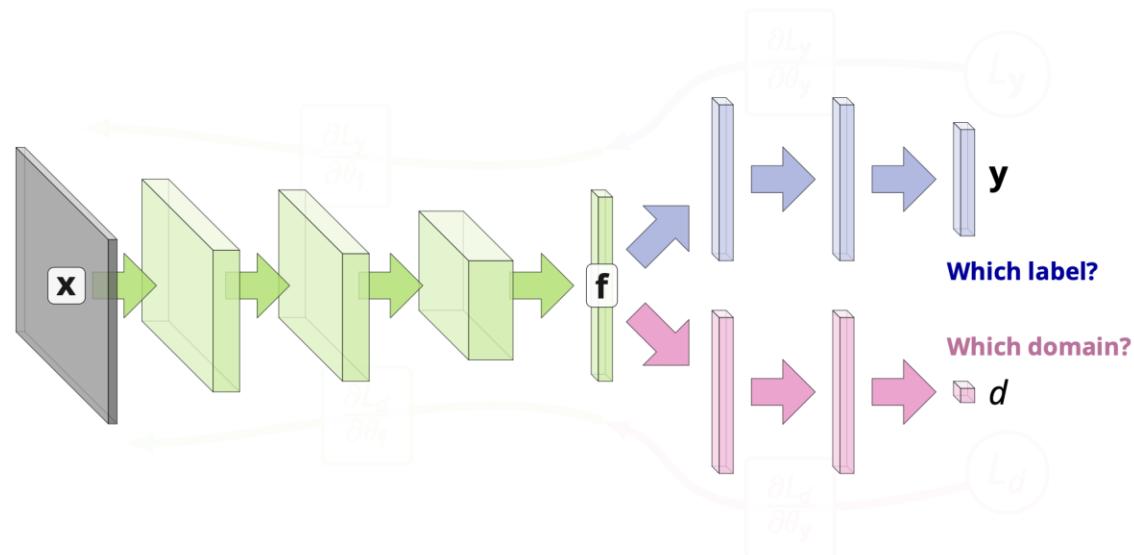


- Computes  $d = G_d(\mathbf{f}; \theta_d)$
- Is trained to predict **0** for **source** and **1** for **target**
- Therefore, the domain loss



Fig. credit: Yaroslav Ganin  
 Yaroslav Ganin and Viktor Lempitsky  
 Unsupervised domain adaptation by back propagation  
 ICML 2015

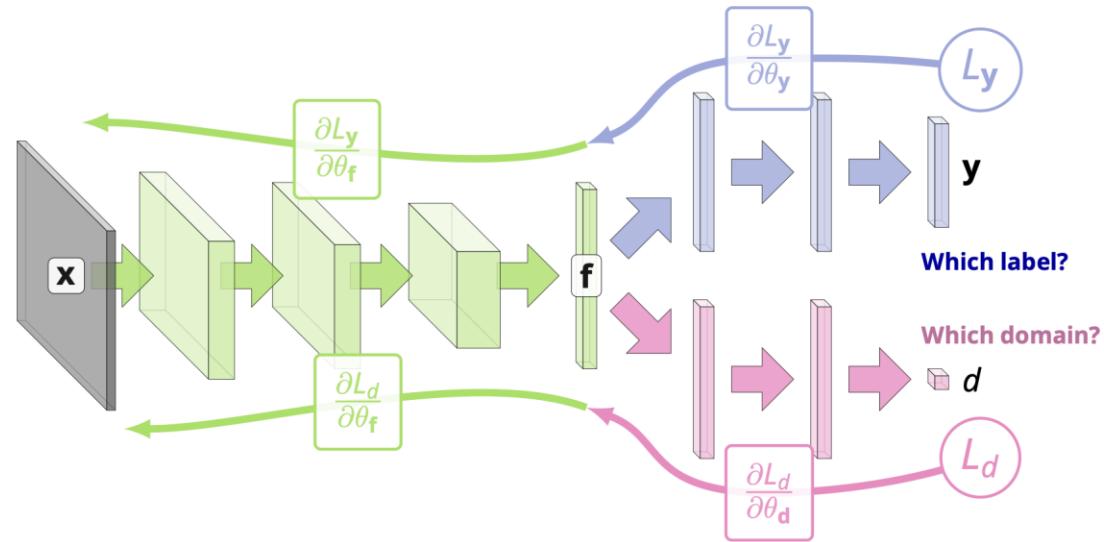
# Deep Unsupervised domain adaptation



It's trying standard backpropagation. Emerging features are:

Fig. credit: Yaroslav Ganin  
Yaroslav Ganin and Viktor Lempitsky  
Unsupervised domain adaptation by back propagation  
ICML 2015

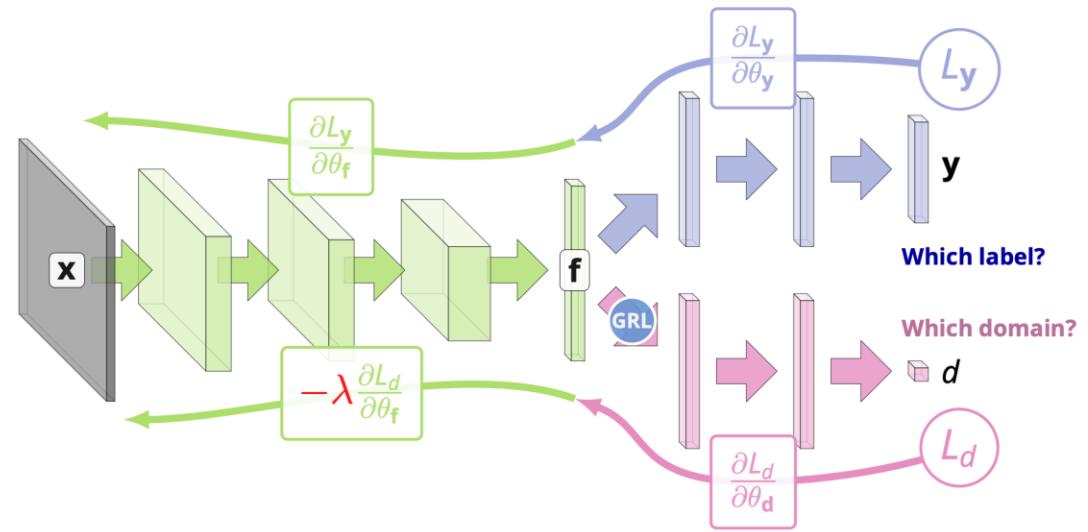
# Deep Unsupervised domain adaptation



Let's try *standard backpropagation*. **Emerging features** are:

- Discriminative (i.e. good for predicting  $y$ )
- Domain-discriminative (i.e. good for predicting  $d$ )

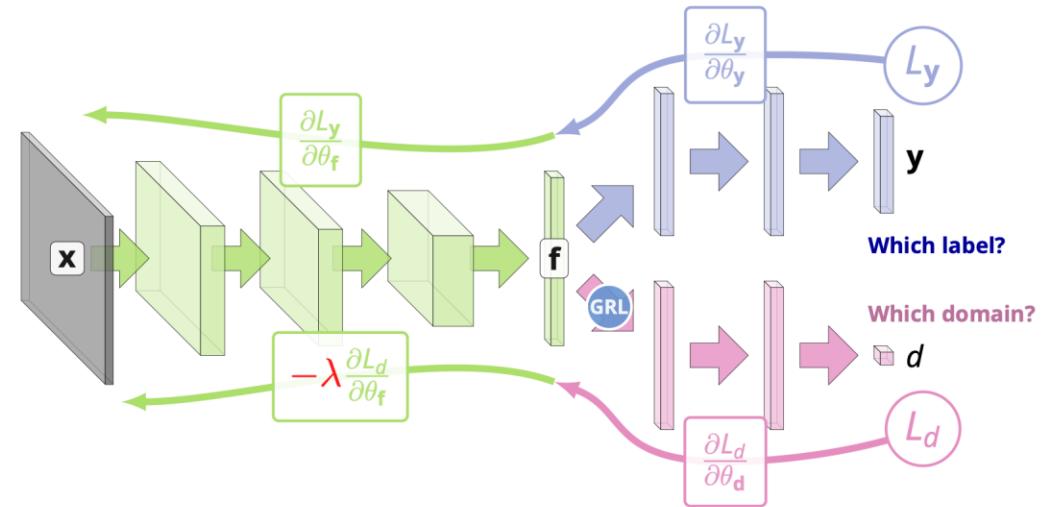
# Deep Unsupervised domain adaptation



Let's now inject the **Gradient Reversal Layer**:

- Copies data without change at *fprop*
- Multiplies deltas by  $-\lambda$  at *bprop*

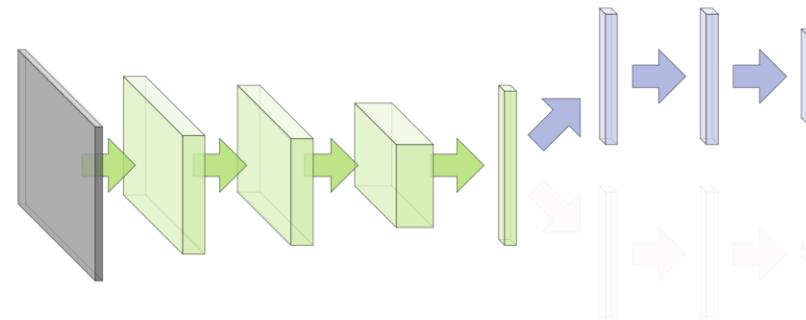
# Deep Unsupervised domain adaptation



**Emerging features** are now:

- Discriminative (i.e. good for predicting  $y$ )
- Domain-invariant (i.e. not good for predicting  $d$ )

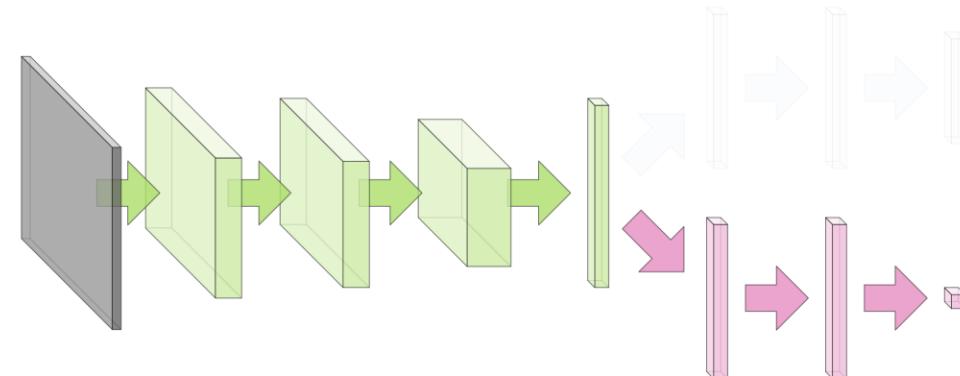
# Deep Unsupervised domain adaptation



- 1 Train **feature extractor + label predictor** on **source**
- 2 Train **feature extractor + domain classifier** on **source + target**
- 3 Use **feature extractor + label predictor** at test time

Fig. credit: Yaroslav Ganin  
Yaroslav Ganin and Viktor Lempitsky  
Unsupervised domain adaptation by back propagation  
ICML 2015

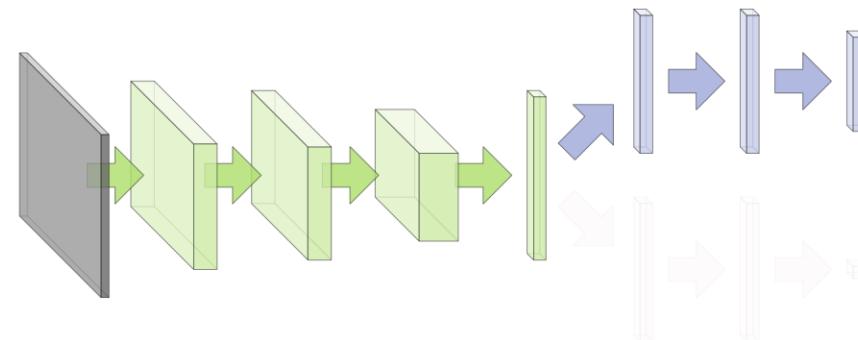
# Deep Unsupervised domain adaptation



- 1 Train **feature extractor + label predictor** on **source**
- 2 Train **feature extractor + domain classifier** on **source + target**
- 3 Use **feature extractor + label predictor** at test time

Fig. credit: Yaroslav Ganin  
Yaroslav Ganin and Viktor Lempitsky  
Unsupervised domain adaptation by back propagation  
ICML 2015

# Deep Unsupervised domain adaptation



- 1 Train **feature extractor + label predictor** on **source**
- 2 Train **feature extractor + domain classifier** on **source + target**
- 3 Use **feature extractor + label predictor** at **test time**

Fig. credit: Yaroslav Ganin  
Yaroslav Ganin and Viktor Lempitsky  
Unsupervised domain adaptation by back propagation  
ICML 2015

# Deep Unsupervised domain adaptation

METHOD	SOURCE	AMAZON	DSLR	WEBCAM
	TARGET	WEBCAM	WEBCAM	DSLR
GFK(PLS, PCA) ( <i>GONG ET AL., 2013</i> )		.197	.497	.631
SA ( <i>FERNANDO ET AL., 2013</i> )		.450	.648	.699
DLID ( <i>S. CHOPRA &amp; GOPALAN, 2013</i> )		.519	.782	.899
DDC ( <i>TZENG ET AL., 2014</i> )		.618	.950	.985
DAN ( <i>LONG &amp; WANG, 2015</i> )		.685	.960	.990
SOURCE ONLY		.642	.961	.978
PROPOSED APPROACH		<b>.730</b>	<b>.964</b>	<b>.992</b>

**Protocol:** all of the methods above use

- **all available labeled source** samples
- **all available unlabeled target** samples

# Domain Adaptation using Adaboost

# Domain Adaptation Setting

- Example: We want to obtain sentiment analysis for movies
- We have a small amount of data for movies for the year 2020
- This data is insufficient to train for sentiment analysis

Sentiment Analysis dataset for year 2020

Sentiment Analysis dataset for years 1990-1995

- We also have some additional training data for movies released in 1990-1995
- How can we best make use of the old training data

# Domain Adaptation Setting

Sentiment Analysis  
dataset for year  
2020

Option 1: Train only with Y2020 data

Problem: Data not enough

Sentiment Analysis  
dataset for year  
2020

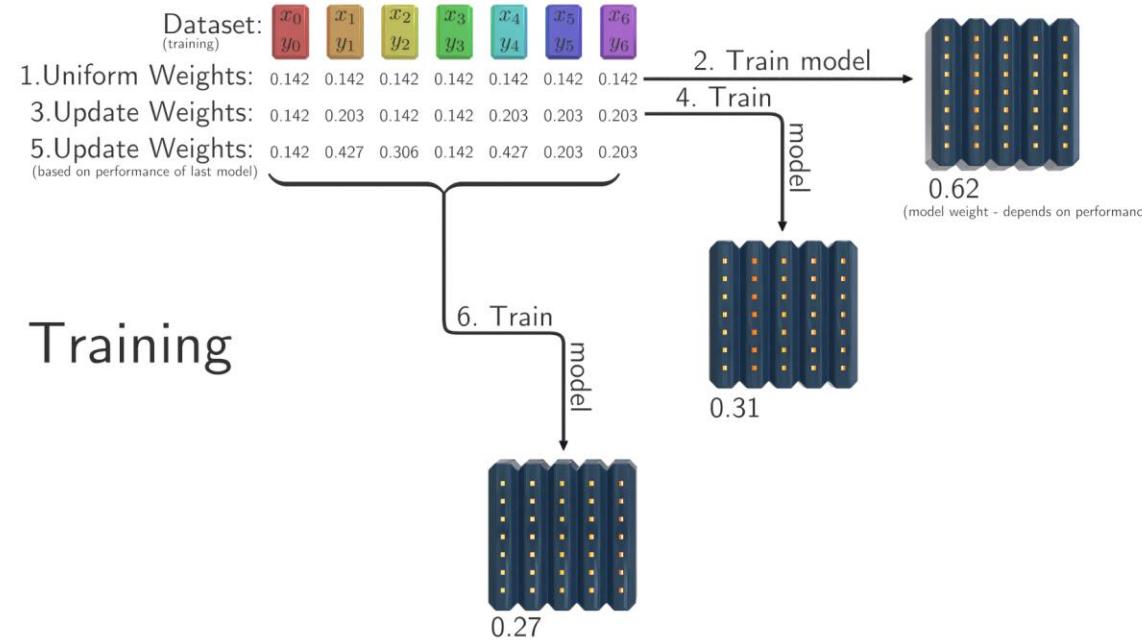
Sentiment Analysis  
dataset for years  
1990-1995

Option 2: Put all data together and train

Problem: While some of the old data is relevant, some of it is not very relevant

# How to Solve?

- Can we find out the data that is relevant from old data
- How do we go about it
- Consider that you have been taught Adaboost and you are a big fan of it :)
- Can you use Adaboost and solve the problem?



Training

# Solution: TrAdaboost

- Main idea:
- Let  $X_s$  be source dataset, (the old dataset)
- Let  $X_d$  be the new labeled target dataset (the new dataset)
- Train using all samples
- If a sample is misclassified, check which dataset it comes from
- If it is source dataset, then decrease its weight, if it is target dataset then increase its weight

---

## Boosting for Transfer Learning

---

**Wenyuan Dai** DWYAK@APEX.SJTU.EDU.CN  
Department of Computer Science and Engineering, Shanghai Jiao Tong University, China

**Qiang Yang** QYANG@CSE.UST.HK  
Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong

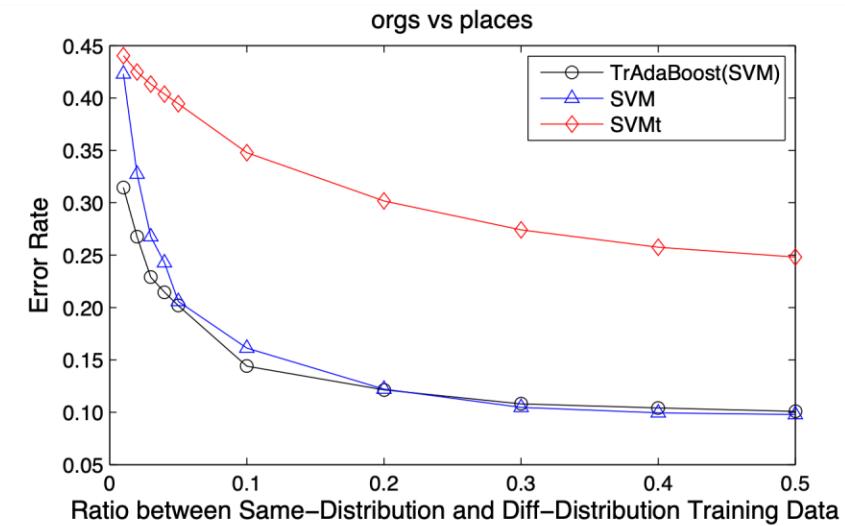
**Gui-Rong Xue** GRXUE@APEX.SJTU.EDU.CN  
**Yong Yu** YYU@APEX.SJTU.EDU.CN  
Department of Computer Science and Engineering, Shanghai Jiao Tong University, China

TrAdaBoost: ICML 2007

# And it works

**Table 4.** The error rates when semi-supervised learning

Data Set	TSVM	TSVMt	TrAdaBoost(TSVM)
rec vs talk	0.059	0.040	<b>0.021</b>
rec vs sci	0.067	0.062	<b>0.013</b>
sci vs talk	0.173	0.106	<b>0.075</b>
auto vs aviation	0.043	0.103	<b>0.038</b>
real vs simulated	0.144	0.131	<b>0.102</b>
orgs vs people	0.358	0.292	<b>0.248</b>
orgs vs places	0.424	0.436	<b>0.304</b>
people vs places	0.307	0.225	<b>0.179</b>
edible vs poisonous	0.439	0.179	<b>0.160</b>



**Figure 2.** The error rate curves on **orgs vs places** data set for three classifiers TrAdaBoost(SVM), SVM and SVMt

# TrAdaboost Algorithm

# TrAdaboost Algorithm

**Input:** The source dataset that is fully labeled  $(X_s, Y_s)$  consisting of  $n$  samples and the destination dataset  $X_d, Y_d$  consisting of  $m$  samples such that  $m < n$ , a baseline learner that provides a hypothesis  $h(x) \rightarrow Y$ . Initialise  $\hat{w}_0(n+1, \dots, n+m) = [\frac{1}{m}, \dots, \frac{1}{m}]$ . For  $t = 1$  to  $N$  do,

# TrAdaboost Algorithm

**Input:** The source dataset that is fully labeled  $(X_s, Y_s)$  consisting of  $n$  samples and the destination dataset  $X_d, Y_d$  consisting of  $m$  samples such that  $m < n$ , a baseline learner that provides a hypothesis  $h(x) \rightarrow Y$ . Initialise  $\hat{w}_{0(n+1, \dots, n+m)} = [\frac{1}{m}, \dots, \frac{1}{m}]$ . For  $t = 1$  to  $N$  do,

1. Train model  $M_t$  with  $\hat{w}_t$  weighted data

# TrAdaboost Algorithm

**Input:** The source dataset that is fully labeled  $(X_s, Y_s)$  consisting of  $n$  samples and the destination dataset  $X_d, Y_d$  consisting of  $m$  samples such that  $m < n$ , a baseline learner that provides a hypothesis  $h(x) \rightarrow Y$ . Initialise  $\hat{w}_0(n+1, \dots, n+m) = [\frac{1}{m}, \dots, \frac{1}{m}]$ . For  $t = 1$  to  $N$  do,

1. Train model  $M_t$  with  $\hat{w}_t$  weighted data
2. Calculate weighted error over  $X_d$

$$e_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=n+1}^{n+m} w_{ti} L_{ti}$$

where  $L_{ti}$  is zero-one loss of exemplar  $i$  for model  $M_t$

# TrAdaboost Algorithm

**Input:** The source dataset that is fully labeled  $(X_s, Y_s)$  consisting of  $n$  samples and the destination dataset  $X_d, Y_d$  consisting of  $m$  samples such that  $m < n$ , a baseline learner that provides a hypothesis  $h(x) \rightarrow Y$ . Initialise  $\hat{w}_0(n+1, \dots, n+m) = [\frac{1}{m}, \dots, \frac{1}{m}]$ . For  $t = 1$  to  $N$  do,

1. Train model  $M_t$  with  $\hat{w}_t$  weighted data
2. Calculate weighted error over  $X_d$

$$e_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=n+1}^{n+m} w_{ti} L_{ti}$$

where  $L_{ti}$  is zero-one loss of exemplar  $i$  for model  $M_t$

3. Model weight

$$\alpha_t = \log \left( \frac{1 - e_t}{e_t} \right), \text{ and } \alpha = \frac{1}{(1 + \sqrt{2 \ln n / N})}$$

# TrAdaboost Algorithm

**Input:** The source dataset that is fully labeled  $(X_s, Y_s)$  consisting of  $n$  samples and the destination dataset  $X_d, Y_d$  consisting of  $m$  samples such that  $m < n$ , a baseline learner that provides a hypothesis  $h(x) \rightarrow Y$ . Initialise  $\hat{w}_0(n+1, \dots, n+m) = [\frac{1}{m}, \dots, \frac{1}{m}]$ . For  $t = 1$  to  $N$  do,

1. Train model  $M_t$  with  $\hat{w}_t$  weighted data
2. Calculate weighted error over  $X_d$

$$e_t = \frac{1}{\|\hat{w}_t\|_1} \sum_{i=n+1}^{n+m} w_{ti} L_{ti}$$

where  $L_{ti}$  is zero-one loss of exemplar  $i$  for model  $M_t$

3. Model weight

$$\alpha_t = \log \left( \frac{1 - e_t}{e_t} \right), \text{ and } \alpha = \frac{1}{(1 + \sqrt{2 \ln n / N})}$$

4. Update weights (only changes weights of exemplars it got wrong)

$$w_{t+1,i} = w_{ti} \exp(\alpha_t L_{ti}) \text{ for } i \in (n+1, n+m)$$

$$w_{t+1,i} = w_{ti} \exp(-\alpha L_{ti}) \text{ for } i \in (1, n)$$

# Related Reading List

- Boosting for transfer learning, by Wenyuan Dai, Qiang Yang, Gui-Rong Xue, Yong Yu, ICML 2007
- Unsupervised domain adaptation by back propagation, Yaroslav Ganin and Viktor Lempitsky, ICML 2015