Deep Learning for Computer Vision: A Crash Course

Yu-Chiang Frank Wang

Professor, Dept. Electrical Engineering, National Taiwan University

Research Director for Deep Learning & Computer Vision, NVIDIA

2024/03/15

About Myself

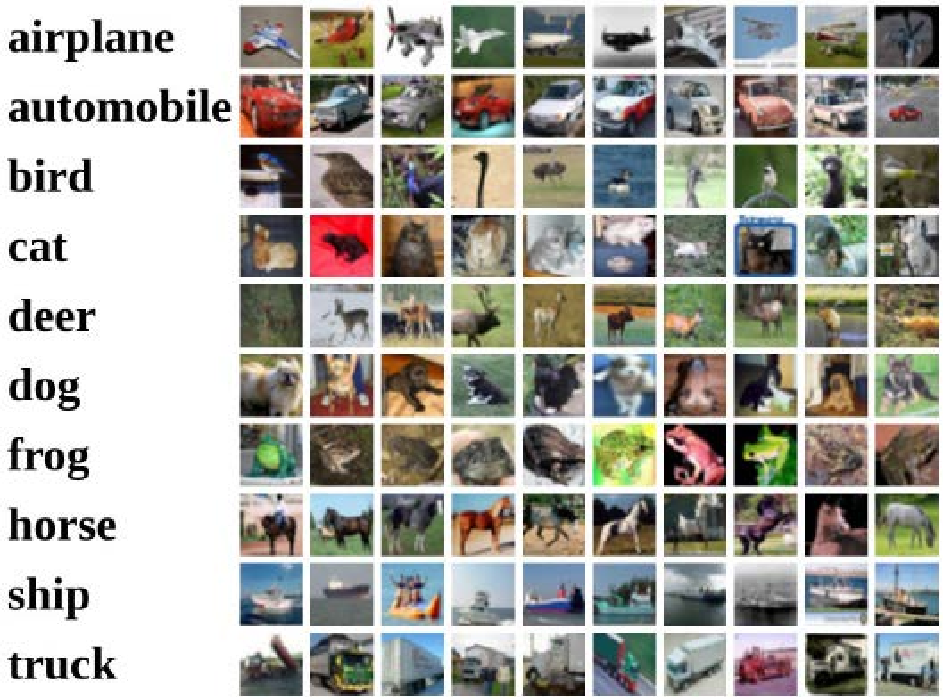
- **Education**
  - **Ph.D./M.S. in Electrical & Computer Engineering** 2002 – 2009
    Carnegie Mellon University, Pittsburgh, USA
  - **B.S. in Electrical Engineering** 1997 – 2001
    National Taiwan University, Taipei, Taiwan
- **Experiences**
  - **Research Director**, NVIDIA 2022/08 ~
  - **Professor**, Dept. EE, National Taiwan University
  - **Principal AI Consultant**, Inventec 2021 – 2022
  - **AI Advisory Consultant**, ASUS Intel. Cloud Services (AICS) 2019 – 2022
  - **Associate Professor**, National Taiwan University 2017 – 2019
  - **Deputy Director**, CITI, Academia Sinica 2015 – 2017
  - **Associate/Assistant Research Fellow** 2009 – 2017
    Research Center for IT Innovation (CITI), Academia Sinica

What'd Be Covered in This Crash Course…

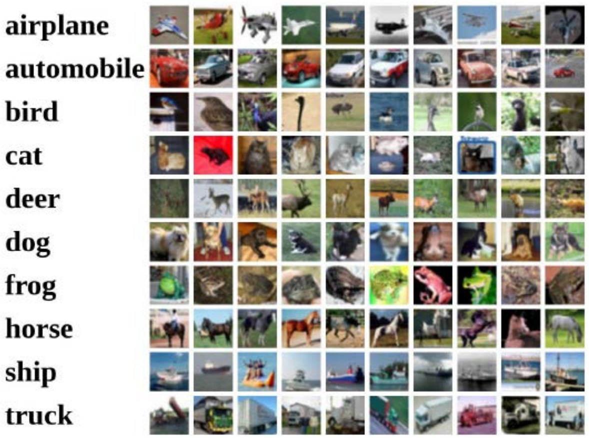
- **Learning-based Computer Vision**
  - From Linear to Non-Linear Classifiers
- **Start of Deep Learning for Computer Vision**
  - Convolutional Neural Networks
  - Self-Supervised Learning
  - Segmentation & Detection
- **Generative Models**
  - Autoencoder, Variational Autoencoder,
    Generative Adversarial Networks & Diffusion Models
- **Sequence-to-Sequence Learning**
  - Attention is All You Need: Transformer
- **Vision & Language Foundation Models**
  - Image-to-Text vs. Text-to-Image
  - Parameter-Efficient Fine-tuning

Linear Classification

- Linear Classifier
  - Can be viewed as a parametric or algebraic approach.
  - Consider that we have 10 object categories of interest
    - E.g., CIFAR10 with 50K training & 10K test images of 10 categories.
      Each image is of size 32 x 32 x 3 pixels.



Linear Classification (cont'd)



- Linear Classifier
  - Can be viewed as a parametric or algebraic approach. Why?
  - Consider that we have 10 object categories of interest
  - Let's take the input image as **x**, and the linear classifier as **W**.
    We need **y** = **Wx** + **b** as a 10-dimensional output vector, indicating the score for each class.

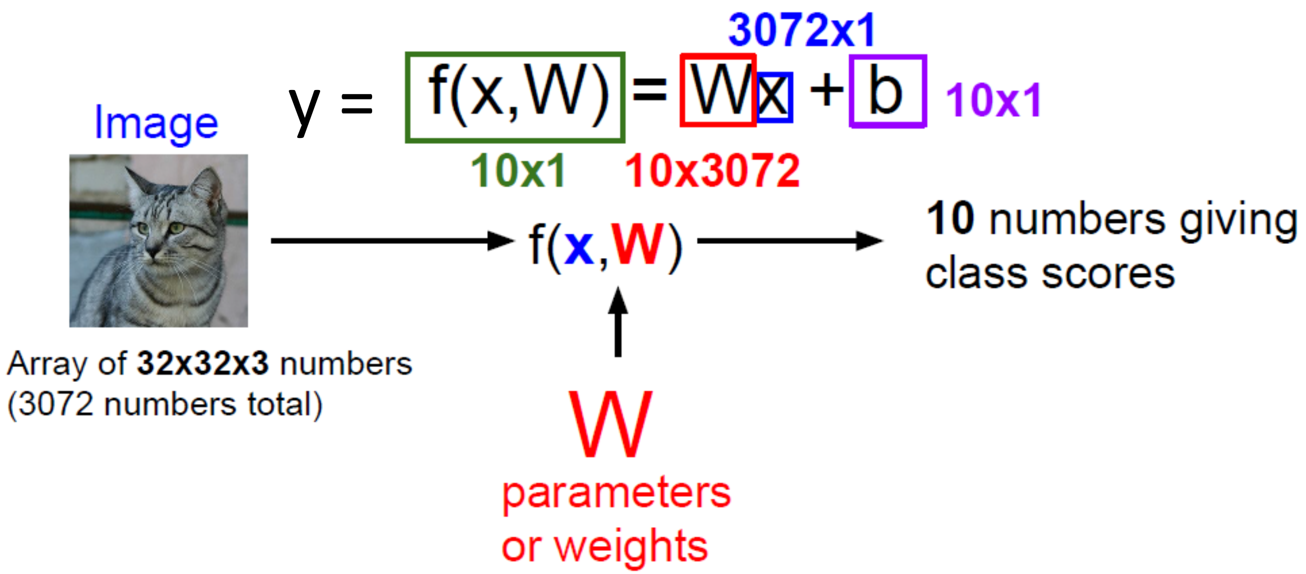


Image credit: Stanford CS231n

Linear Classification (cont'd)

- Linear Classifier
  - Can be viewed as a parametric or algebraic approach. Why?
  - Consider that we have 10 object categories of interest
  - Let's take the input image as **x**, and the linear classifier as **W**.
    We need **y** = **Wx** + **b** as a 10-dimensional output vector, indicating the score for each class.
  - For example, an image with 2 x 2 pixels & 3 classes of interest
    we need to learn a linear classifier **W** (plus a bias **b**),
    so that desirable outputs **y** = **Wx** + **b** can be expected.

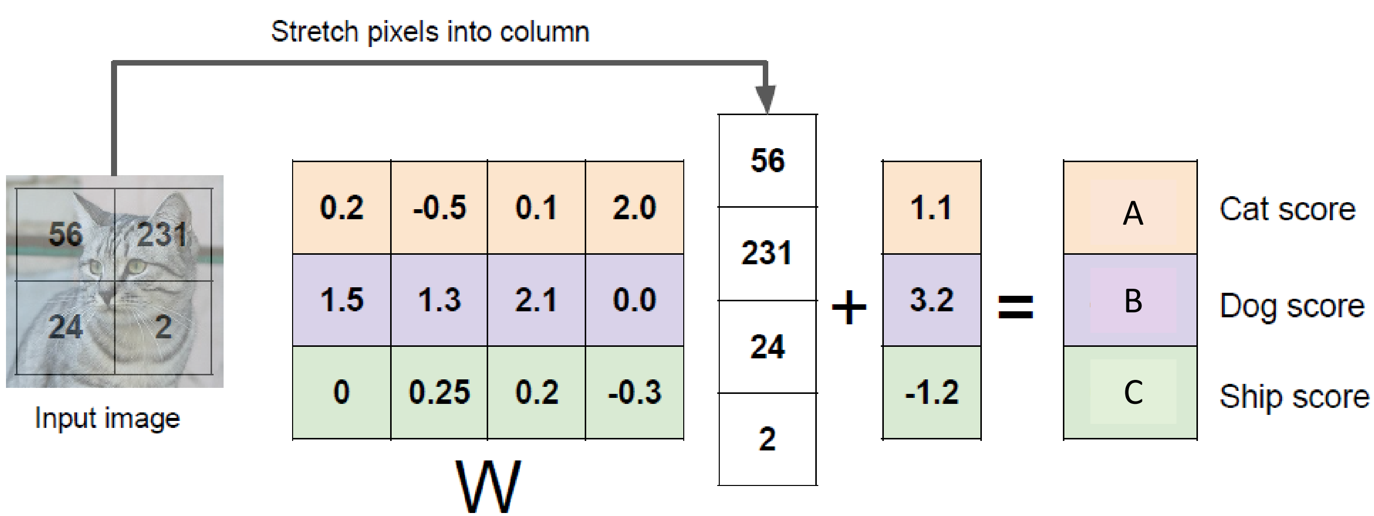
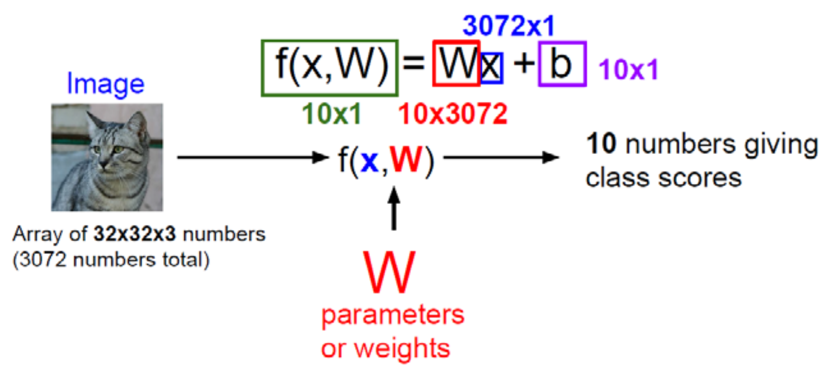


Image credit: Stanford CS231n

Remarks

- Interpreting **W** in **y** = **Wx** + **b**
  - Weights **W** are learned by observing (training) data **X** and their ground truth labels **Y**.
  - Each row in **W** can be viewed as an exemplar of the corresponding class.
  - Equivalently, we perform inner product between **x** and each row of **W** -> class similarity
  - How to determine a proper objective/loss function for deriving **W**?



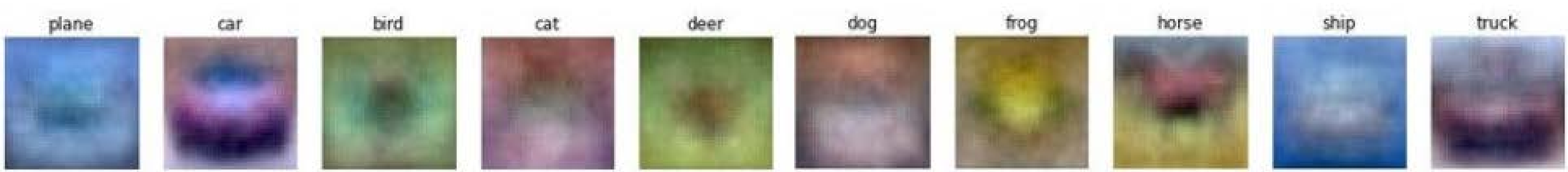


Image credit: Stanford CS231n

Loss Function

- **Cross-Entropy Loss** (Multinomial Logistic Regression)
  - Interpret classifier scores as **probabilities**
  - **Softmax function:**

$$P(Y = k \mid X = x_i) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$ with $s = f(x_i; W)$ as the classifier output for input $\mathbf{x}_i$

  - See example below

	Unnormalized log-probabilities / logits	$\xrightarrow{\text{exp}}$	unnormalized probabilities (Probabilities must be >= 0)	$\xrightarrow{\text{normalize}}$	probabilities (Probabilities must sum to 1)
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

$$\boxed{L_i = -\log P(Y = y_i \mid X = x_i)}$$

L_{cat} = -log(0.13) = 2.04

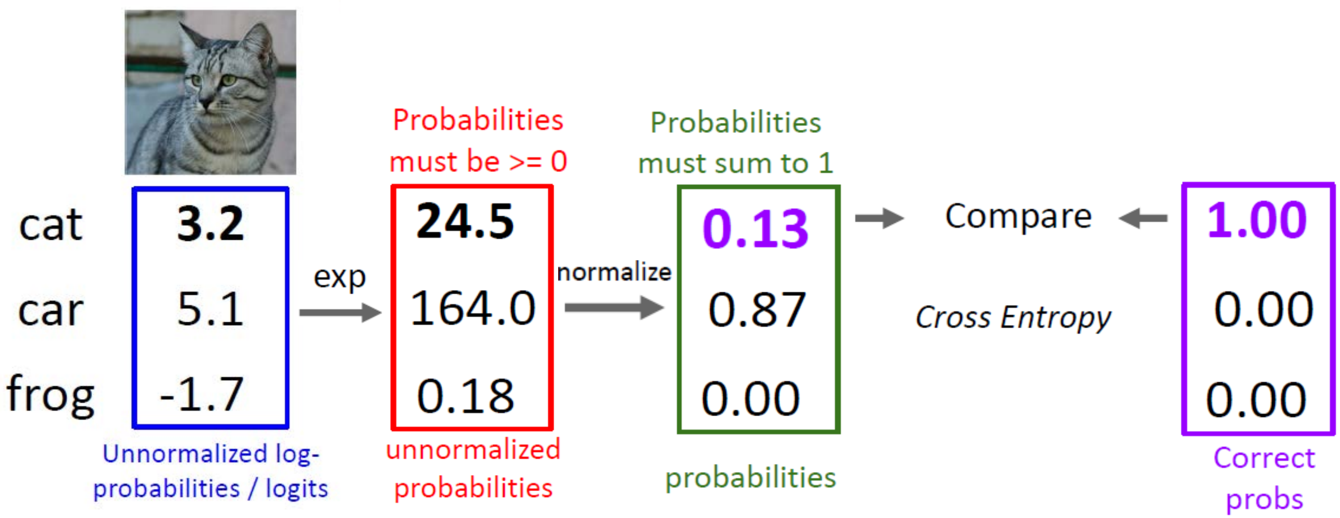
What about this L?
What are its possibly min/max value??

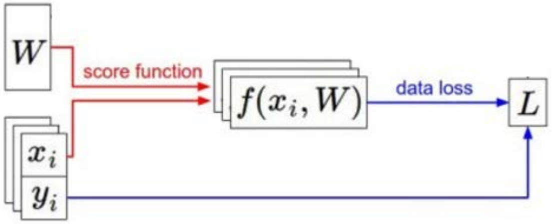
- **Cross-Entropy Loss** (cont'd)
  - **Softmax function:**

$$P(Y = k \mid X = x_i) = \frac{\exp(s_k)}{\sum_j \exp(s_j)}$$ with $s = f(x_i; W)$ as the classifier output for input $\mathbf{x}_i$

$$\Rightarrow \boxed{L_i = -\log P(Y = y_i \mid X = x_i)} \quad \text{or} \quad \boxed{L_i = -\log\left(\frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}\right)}$$

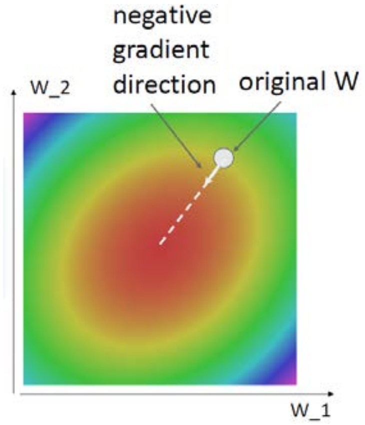
  - **(Binary) Cross Entropy Loss** (or $\mathbf{L}_{\mathbf{BCE}}$; see example below):





- **Searching for W from L_{BCE}**
  - Computing **gradients**:
    Following the slope to reach the (hopefully global) minimum for **W**.
  - Gradient Descent via numeric or analytic gradients:
    - Iteratively step in the direction of the negative gradient & search for **W**
    - Hyperparameters: weight initialization, # of steps, learning rate, etc.

```
# Vanilla gradient descent
w = initialize_weights()
for t in range(num_steps):
  dw = compute_gradient(loss_fn, data, w)
  w -= learning_rate * dw
```



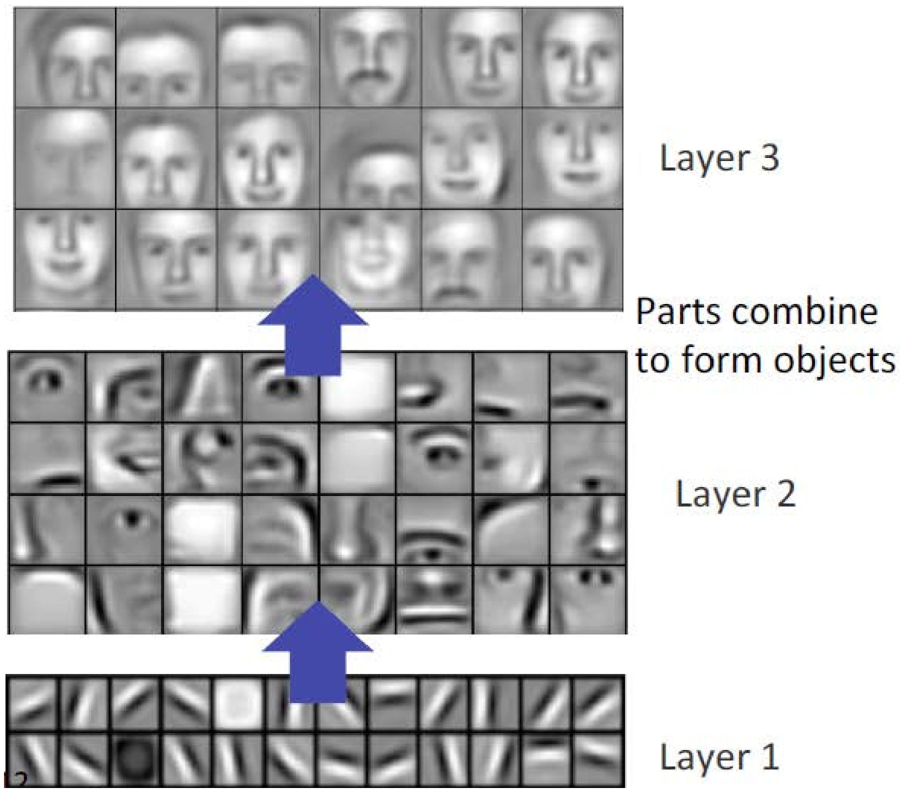
  - Stochastic Gradient Descent
    - Full sum in L is expensive when large N
    - *Approximate* sum using a minibatch of instances (e.g., 32, 64, 128, etc.)
    - Additional hyperparameters of batch size and data sampling

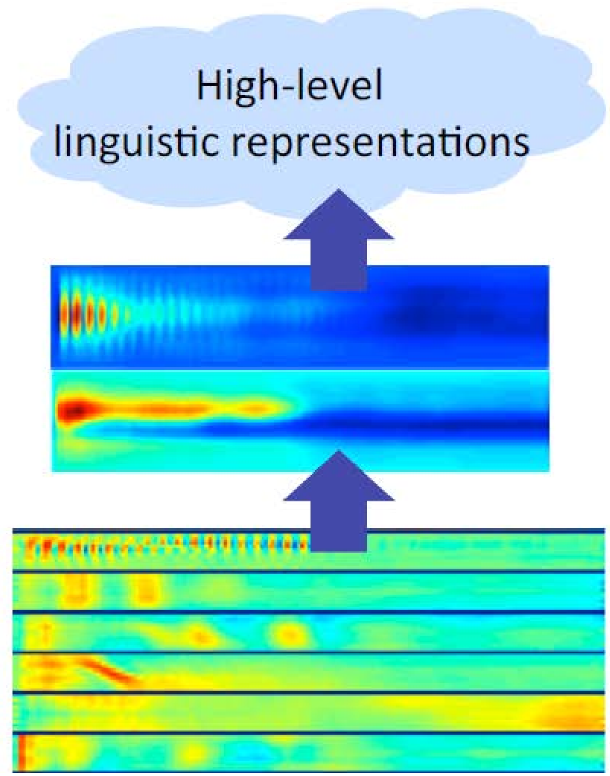
```
# Stochastic gradient descent
w = initialize_weights()
for t in range(num_steps):
  minibatch = sample_data(data, batch_size)
  dw = compute_gradient(loss_fn, minibatch, w)
  w -= learning_rate * dw
```

Slide credit: UMich EECS 498-007

Hierarchical Representation Learning

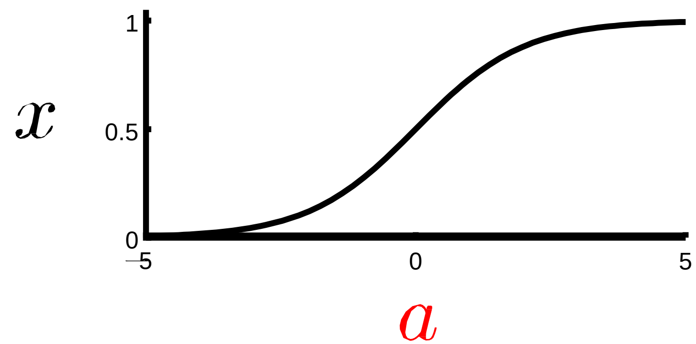
- Successive model layers learn deeper intermediate representations.





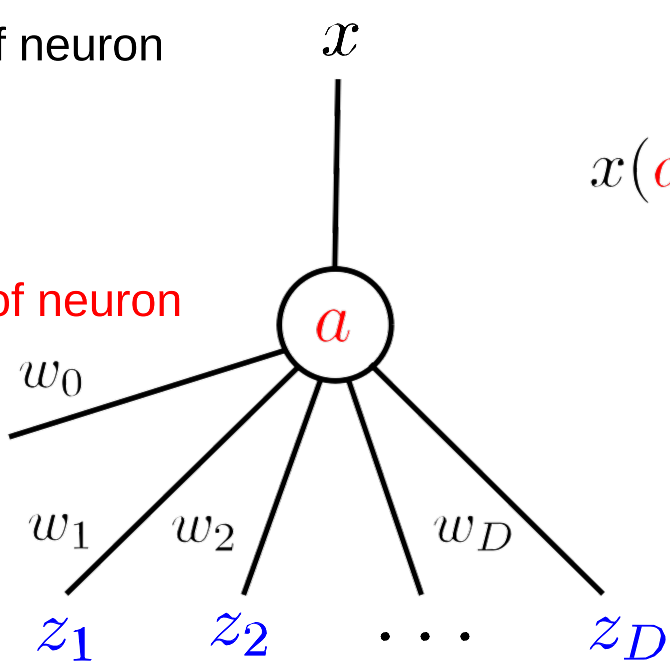
Let's Take a Closer Look…

- A single neuron in a single layer



output of neuron

activity of neuron

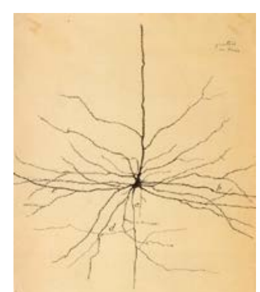


inputs to neuron

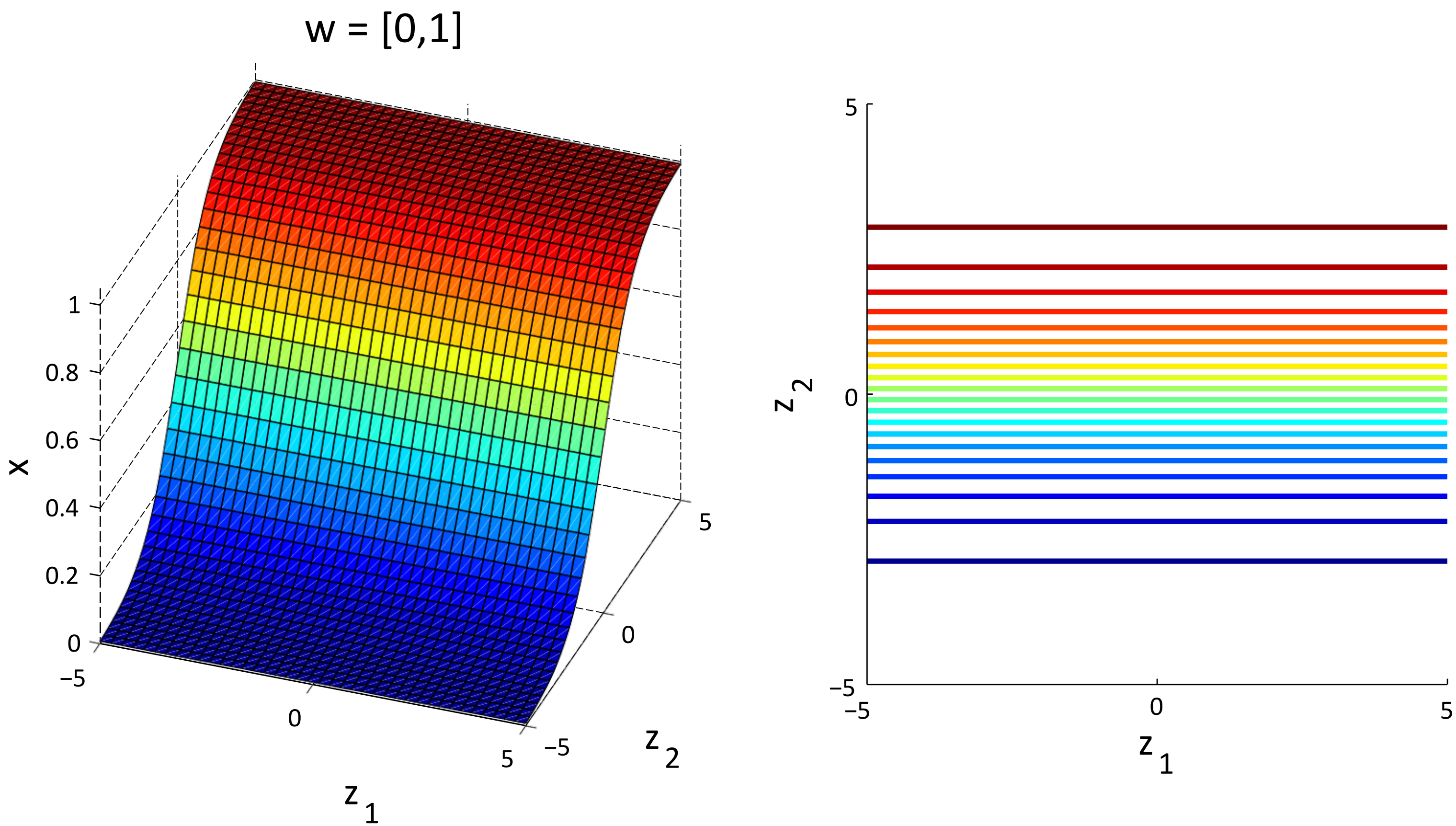
$$x(a) = \frac{1}{1+\exp(-a)} \qquad x \in (0,1)$$

$$a = w_0 + \sum_{d=1}^{D} w_d z_d$$

$$= \sum_{d=0}^{D} w_d z_d$$

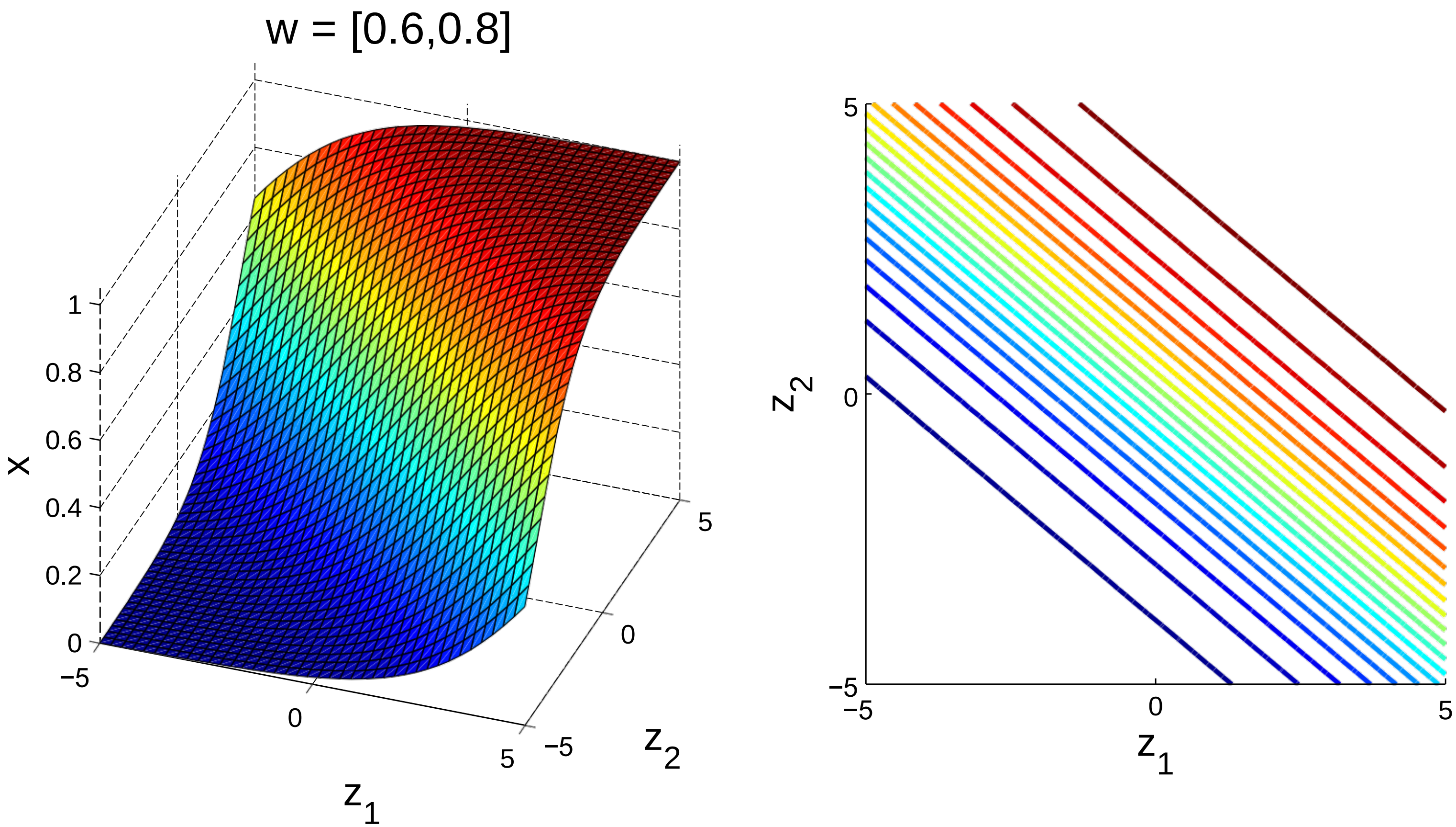


Input-Output Function of a Single Neuron



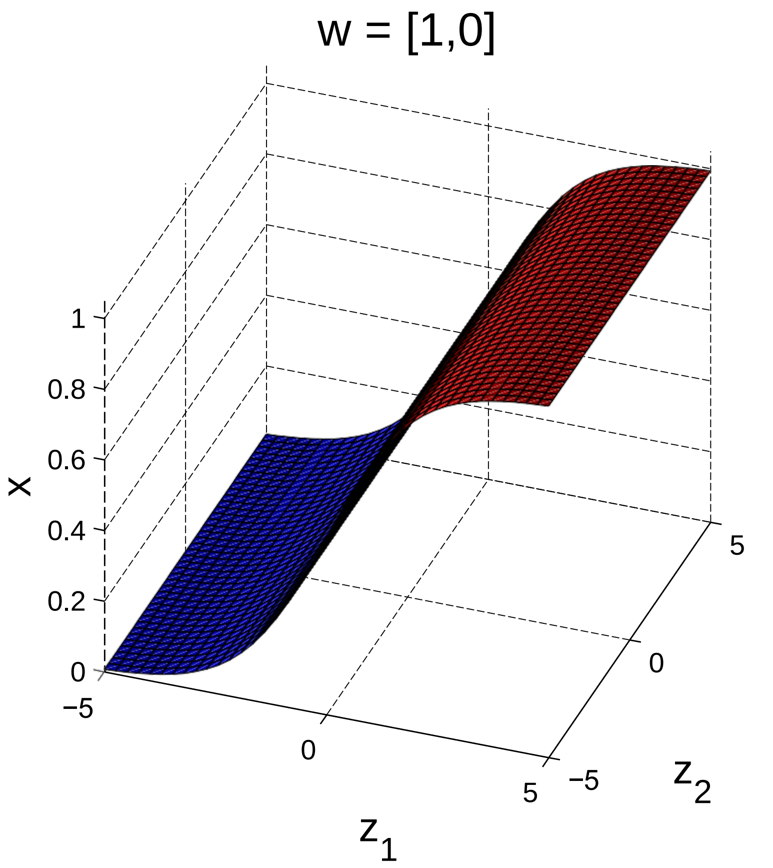
$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

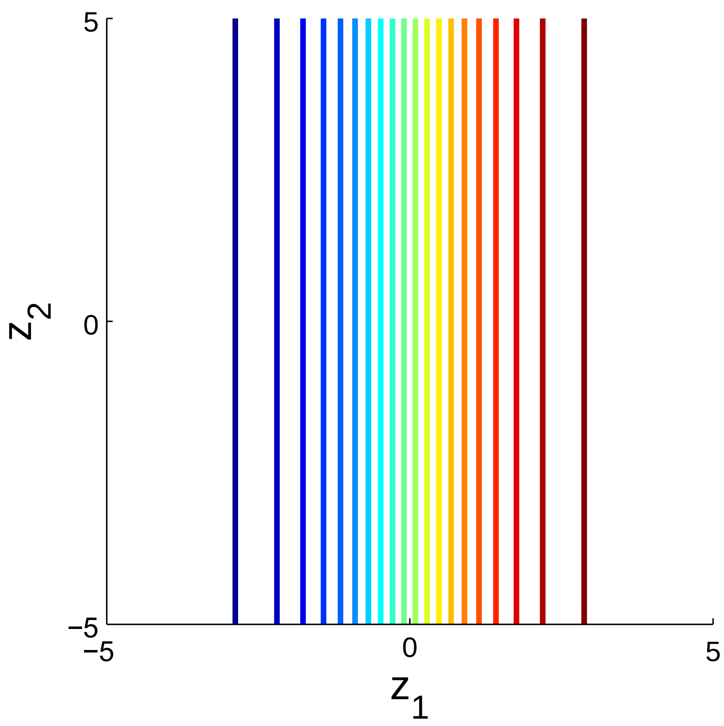
Input-Output Function of a Single Neuron



$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

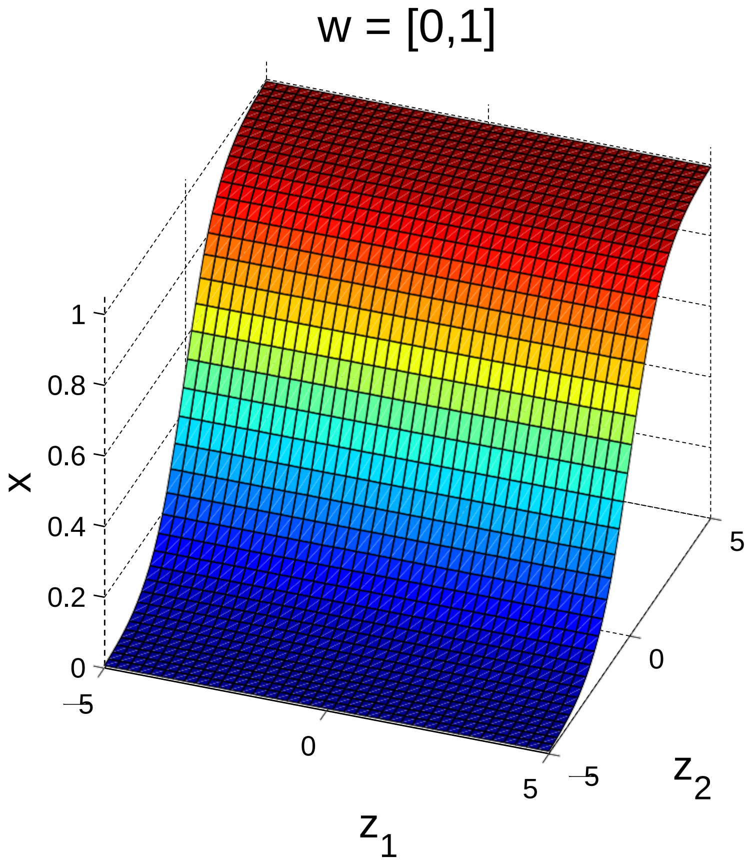
Input-Output Function of a Single Neuron





$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

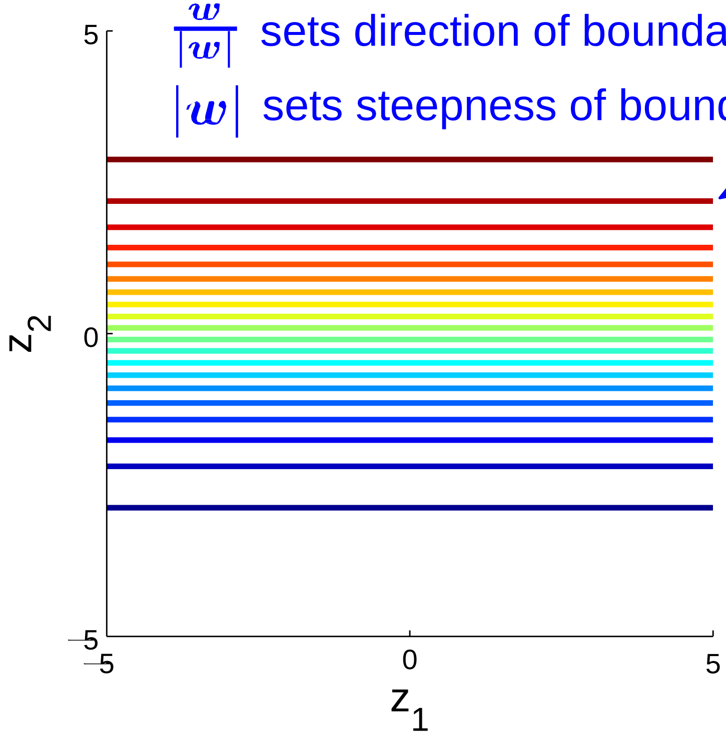
Input-Output Function of a Single Neuron (cont'd)



contours $w_1 z_1 + w_2 z_2 = c = \boldsymbol{w}^\mathsf{T}\boldsymbol{z}$

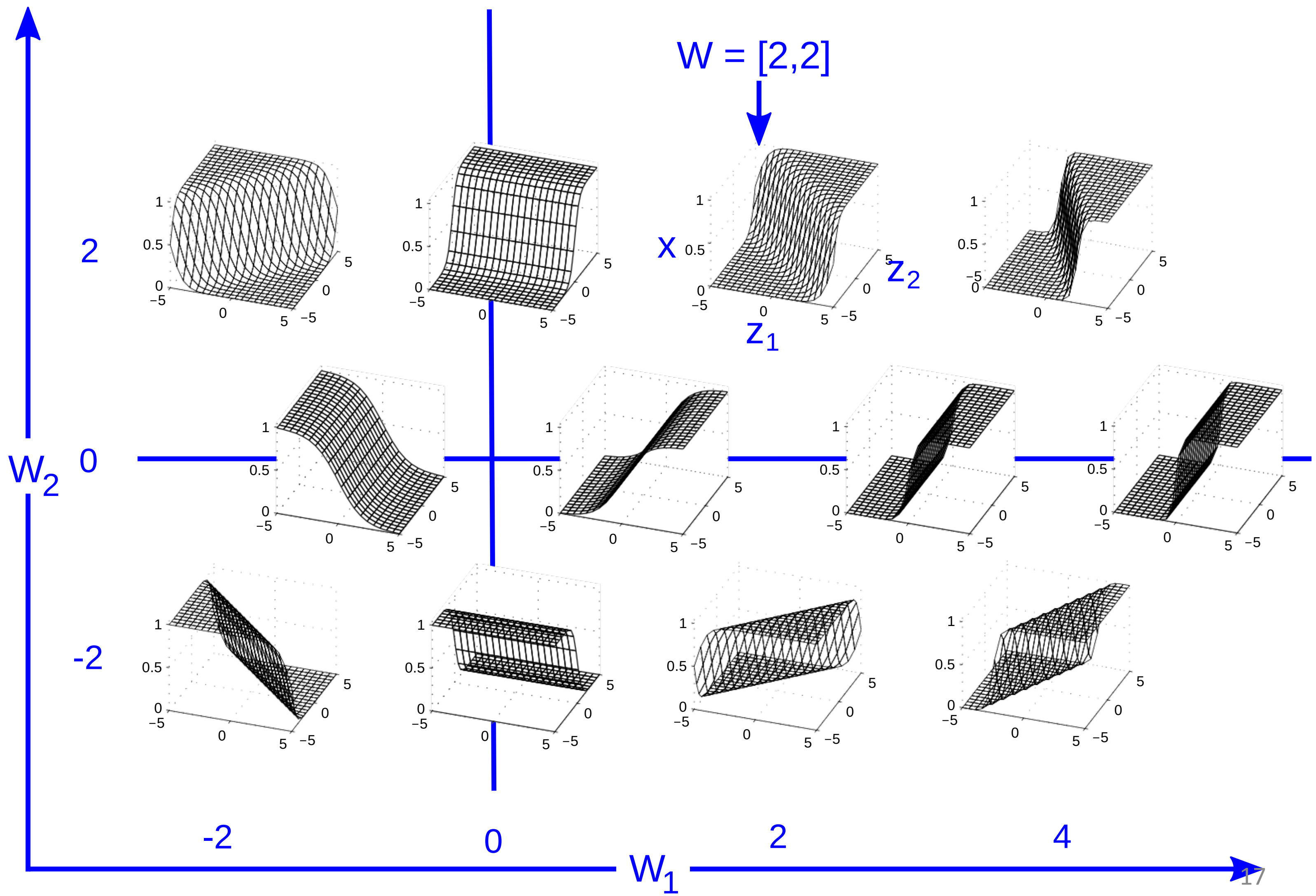
$\frac{\boldsymbol{w}}{|\boldsymbol{w}|}$ sets direction of boundary

$|\boldsymbol{w}|$ sets steepness of boundary

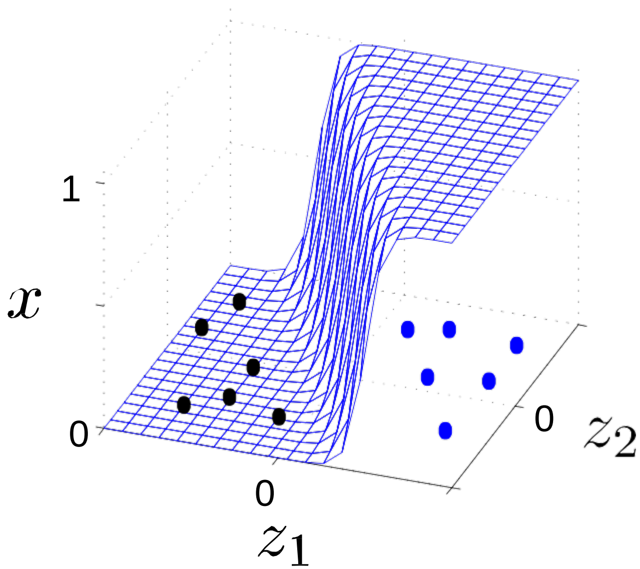


$$x(z_1, z_2) = \frac{1}{1+\exp(-w_1 z_1 - w_2 z_2)}$$

Weight Space of a Single Neuron



Training a single neuron



training data

$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

objective function:

$$\boxed{G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w}) + (1 - t^n) \log \left(1 - \mathrm{x}(\boldsymbol{z}^{(n)}; \boldsymbol{w})\right)\right]}$$

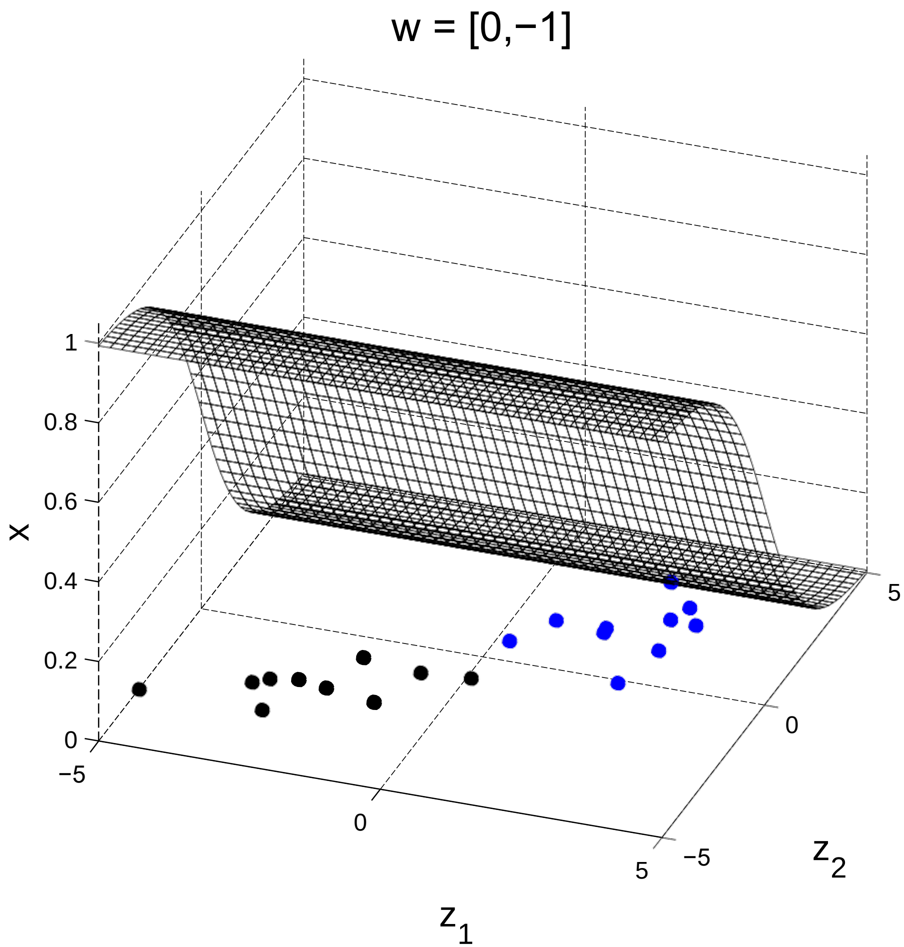
$E(\boldsymbol{w}) = \frac{1}{2} \sum_i w_i^2$ regulariser discourages the network using extreme weights

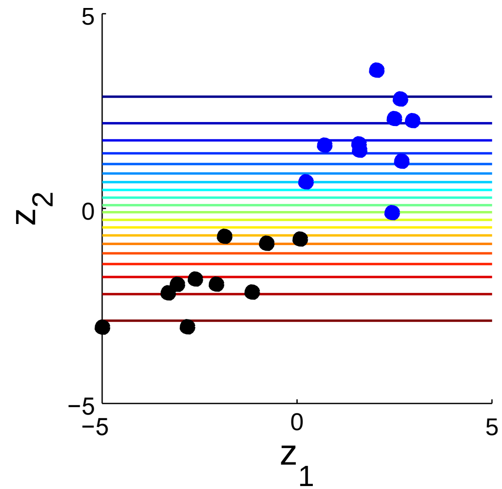
$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} [G(\boldsymbol{w}) + \alpha E(\boldsymbol{w})]$$

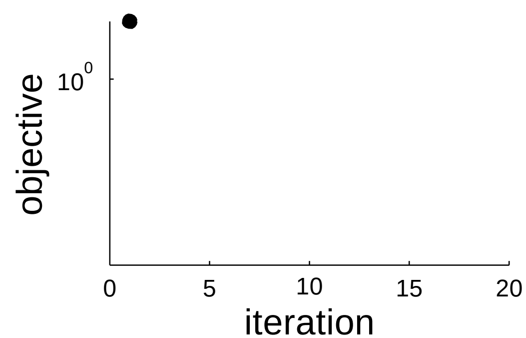
$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} M(\boldsymbol{w}) = -\sum_n (t^{(n)} - x^{(n)}) \boldsymbol{z}^{(n)} + \alpha \boldsymbol{w}$$

weight decay - shrinks weights towards zero

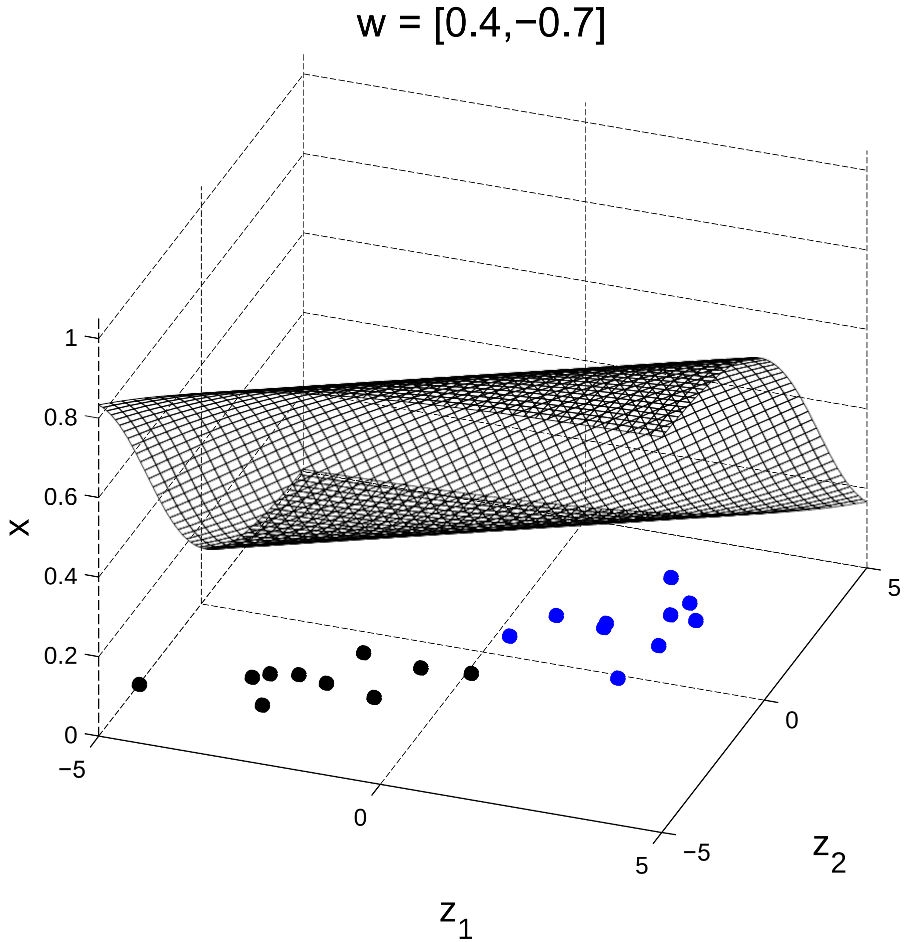
Training a Single Neuron

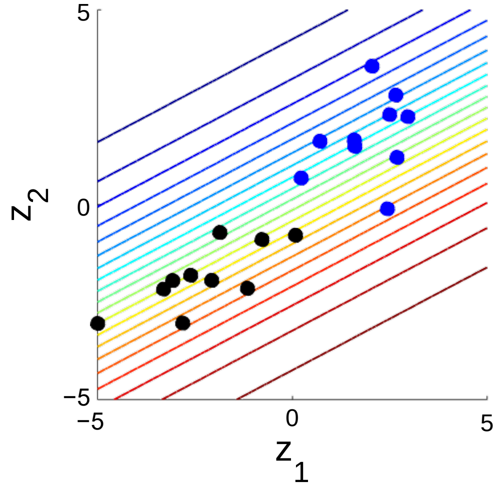


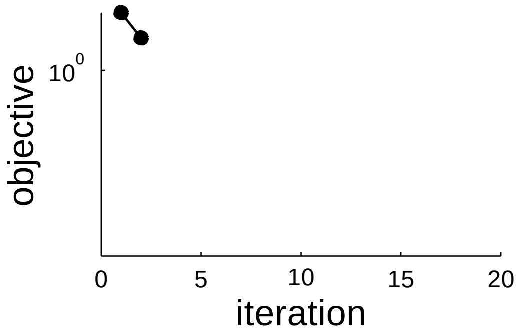




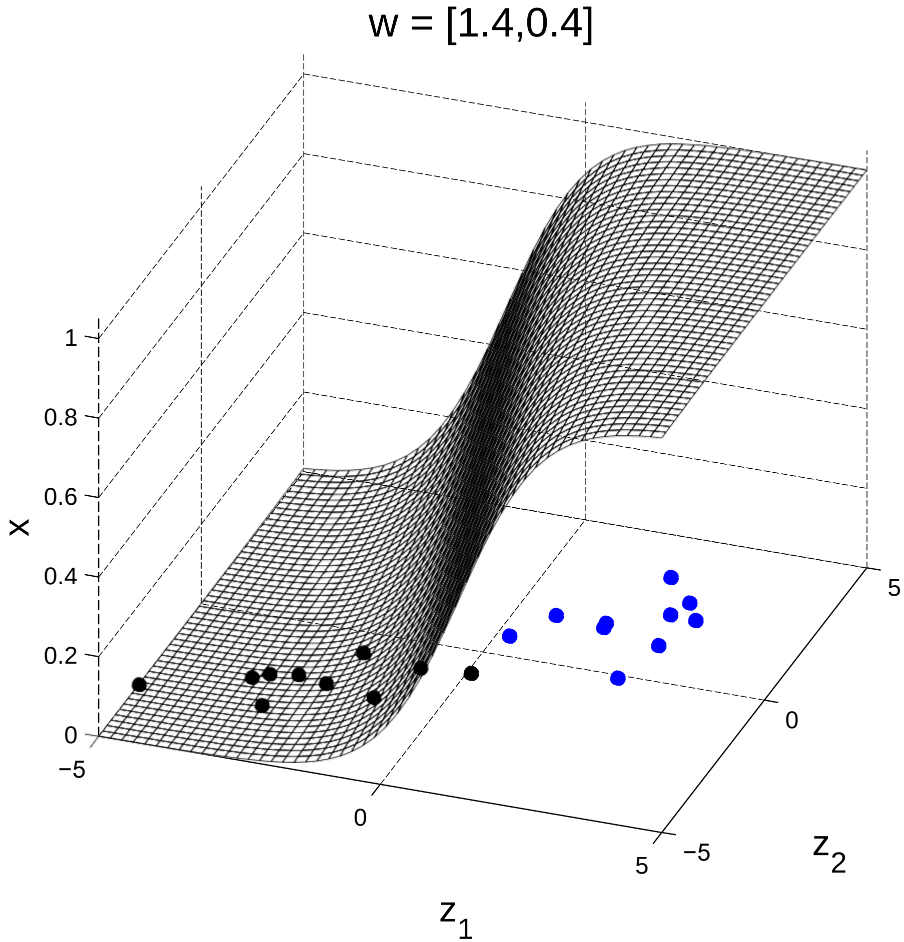
Training a Single Neuron

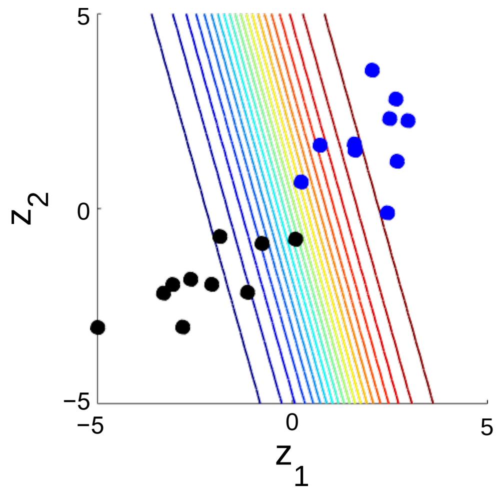


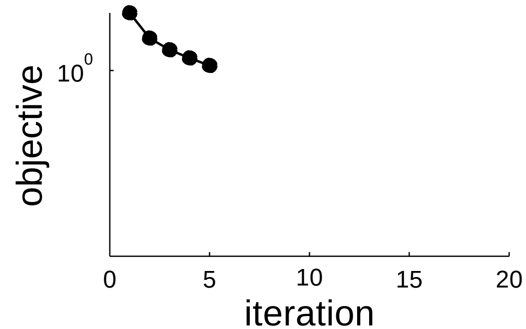




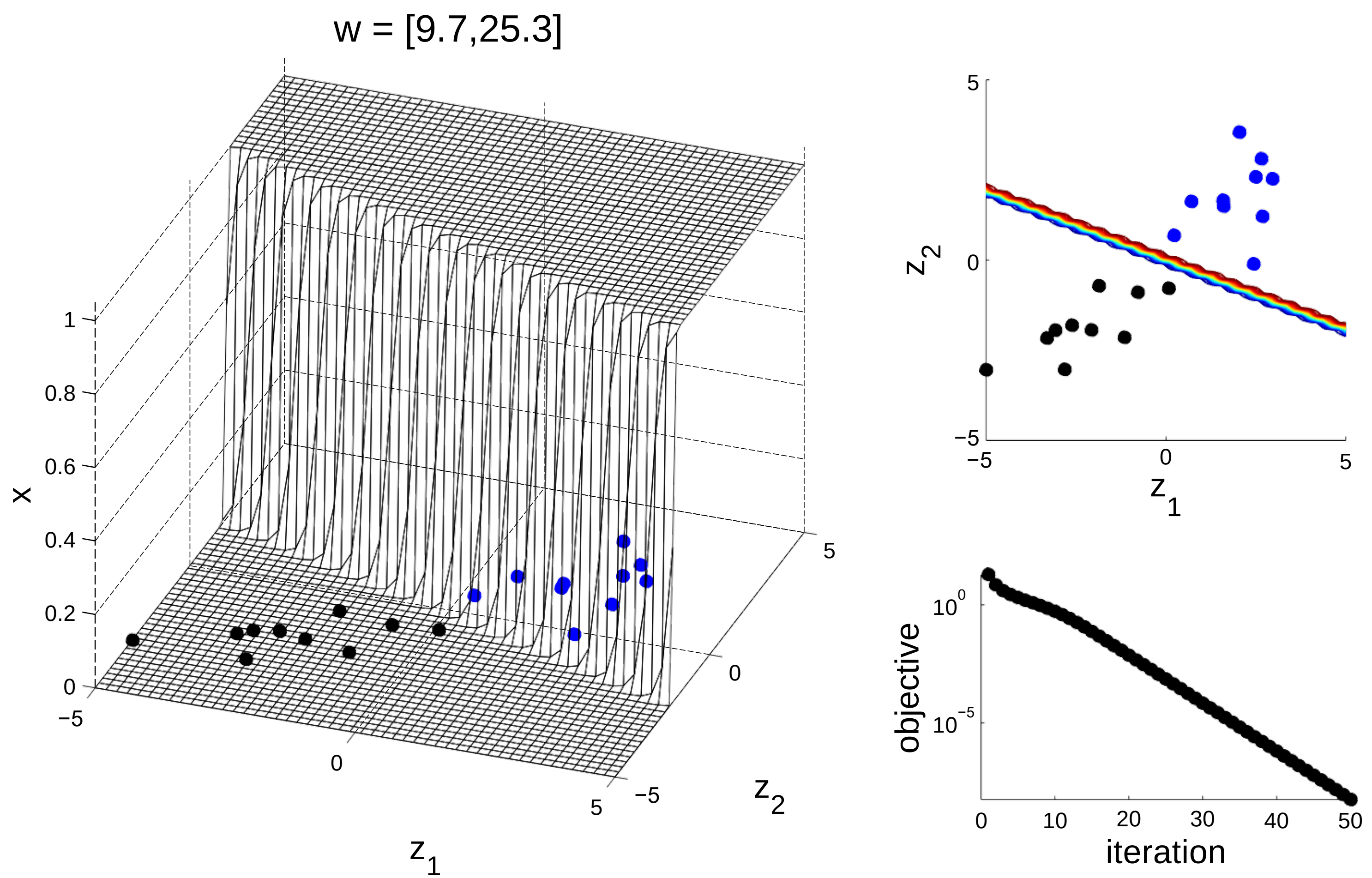
Training a Single Neuron



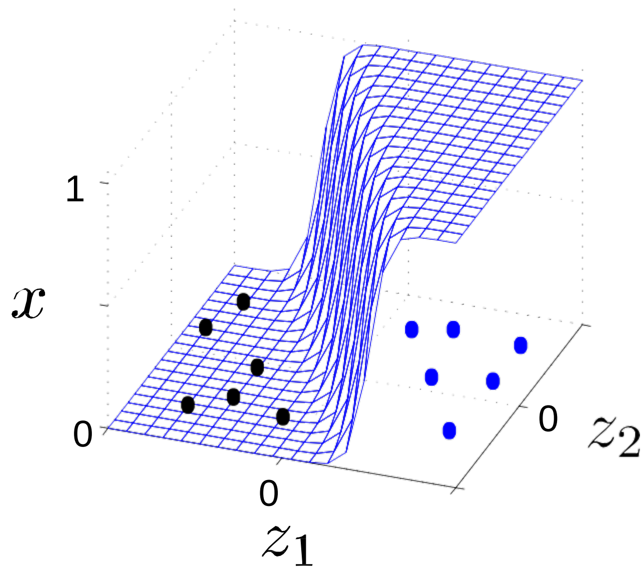




Training a Single Neuron



Overfitting and Weight Decay



training data

$\{\boldsymbol{z}^{(n)}\}_{n=1}^{N}$ $\{t^{(n)}\}_{n=1}^{N}$

inputs class labels

objective function:

$$G(\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}(\boldsymbol{z}^{(n)};\boldsymbol{w}) + (1-t^n)\log\left(1-\mathrm{x}(\boldsymbol{z}^{(n)};\boldsymbol{w})\right)\right]$$

$$\boxed{E(\boldsymbol{w}) = \tfrac{1}{2}\sum_i w_i^2}$$

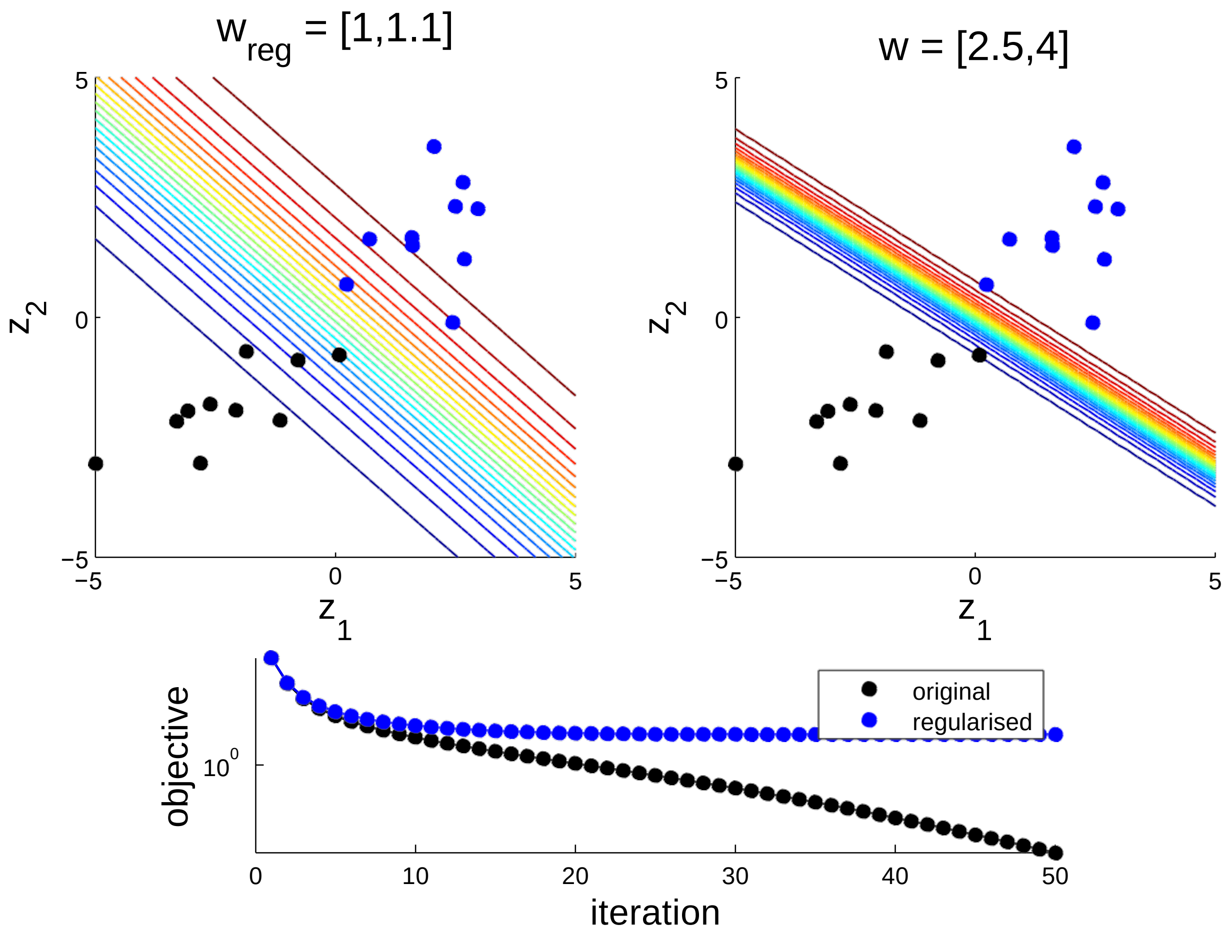
regulariser discourages the network using extreme weights

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} M(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \left[G(\boldsymbol{w}) + \alpha E(\boldsymbol{w})\right]$$

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} M(\boldsymbol{w}) = -\sum_n (t^{(n)} - x^{(n)})\boldsymbol{z}^{(n)} + \alpha \boldsymbol{w}$$

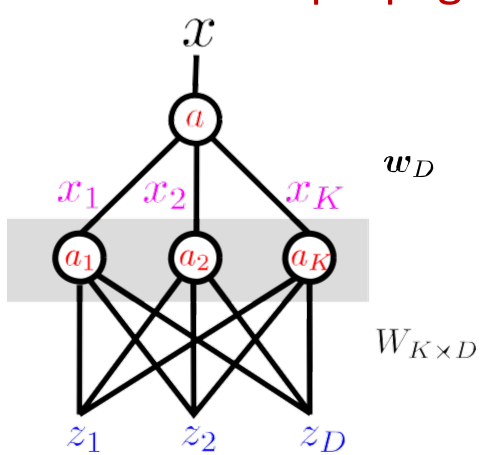
weight decay - shrinks weights towards zero

Training a Single Neuron (cont'd)



Training a Neural Network with Two Hidden Layers

Networks with hidden layers can be fit using gradient descent using an algorithm called back-propagation.



$$x(a) = \frac{1}{1+\exp(-a)}$$

$$a = \sum_{k=1}^{K} w_k x_k$$

$$x(a_k) = \frac{1}{1+\exp(-a_k)}$$

$$a_k = \sum_{d=1}^{D} W_{k,d} z_d$$

objective function:

$$G(W,\boldsymbol{w}) = -\sum_n \left[t^{(n)} \log \mathrm{x}^{(n)} + (1-t^n)\log\left(1-\mathrm{x}^{(n)}\right)\right]$$ likelihood same as before

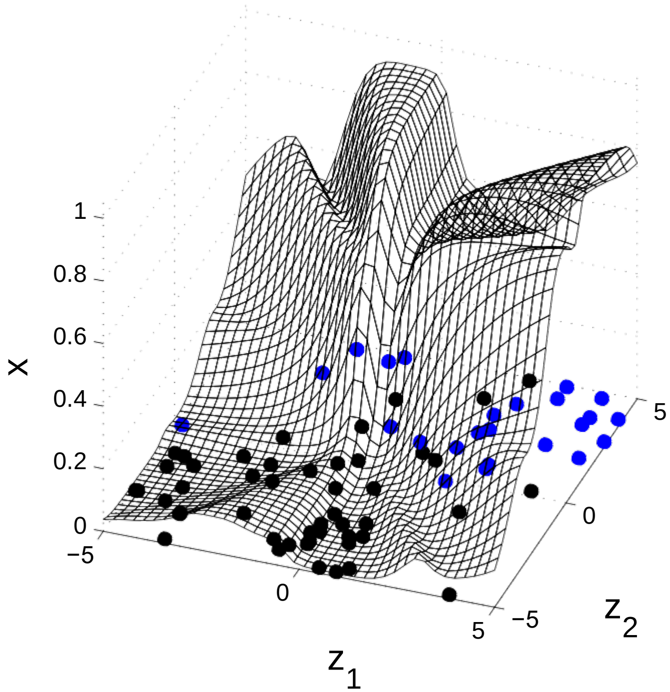
$$E(W,\boldsymbol{w}) = \frac{1}{2}\sum_i w_i^2 + \frac{1}{2}\sum_{ij} W_{ij}^2$$ regulariser discourages extreme weights

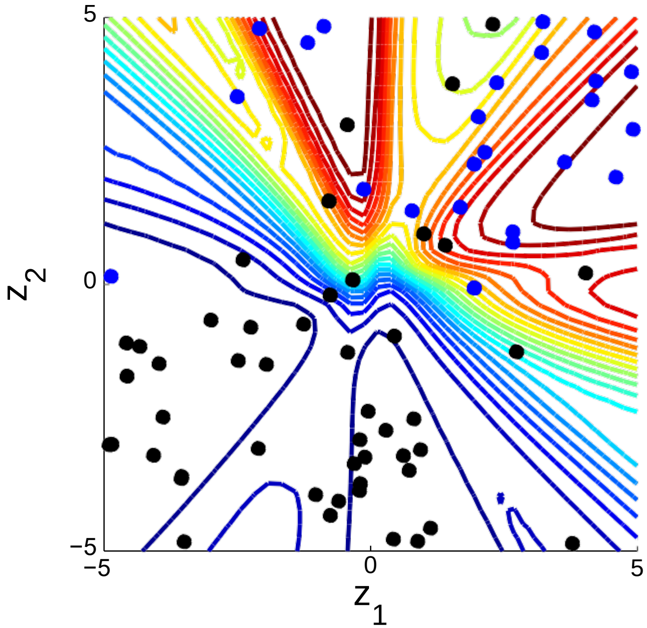
$$\{W,\boldsymbol{w}^*\} = \underset{W,\boldsymbol{w}}{\arg\min}\, M(W,\boldsymbol{w}) = \underset{W,\boldsymbol{w}}{\arg\min}\left[G(W,\boldsymbol{w}) + \alpha E(W,\boldsymbol{w})\right]$$

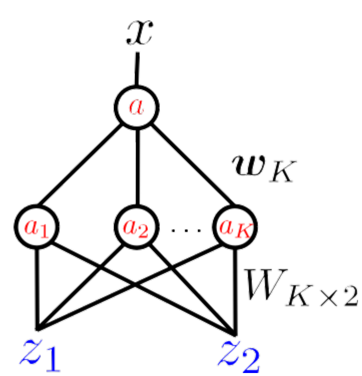
$$\frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}}\frac{\mathrm{d}x^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}}\frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}}\frac{\mathrm{d}a^{(n)}}{\mathrm{d}W_{ij}}$$

$$= \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}}\frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}}\frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}}\frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}W_{ij}} = \sum_n \frac{\mathrm{d}G(W,\boldsymbol{w})}{\mathrm{d}x^{(n)}}\frac{\mathrm{d}x^{(n)}}{\mathrm{d}a^{(n)}}\frac{\mathrm{d}a^{(n)}}{\mathrm{d}x_i^{(n)}}\frac{\mathrm{d}x_i^{(n)}}{\mathrm{d}a_i^{(n)}}\frac{\mathrm{d}a_i^{(n)}}{\mathrm{d}W_{ij}}$$

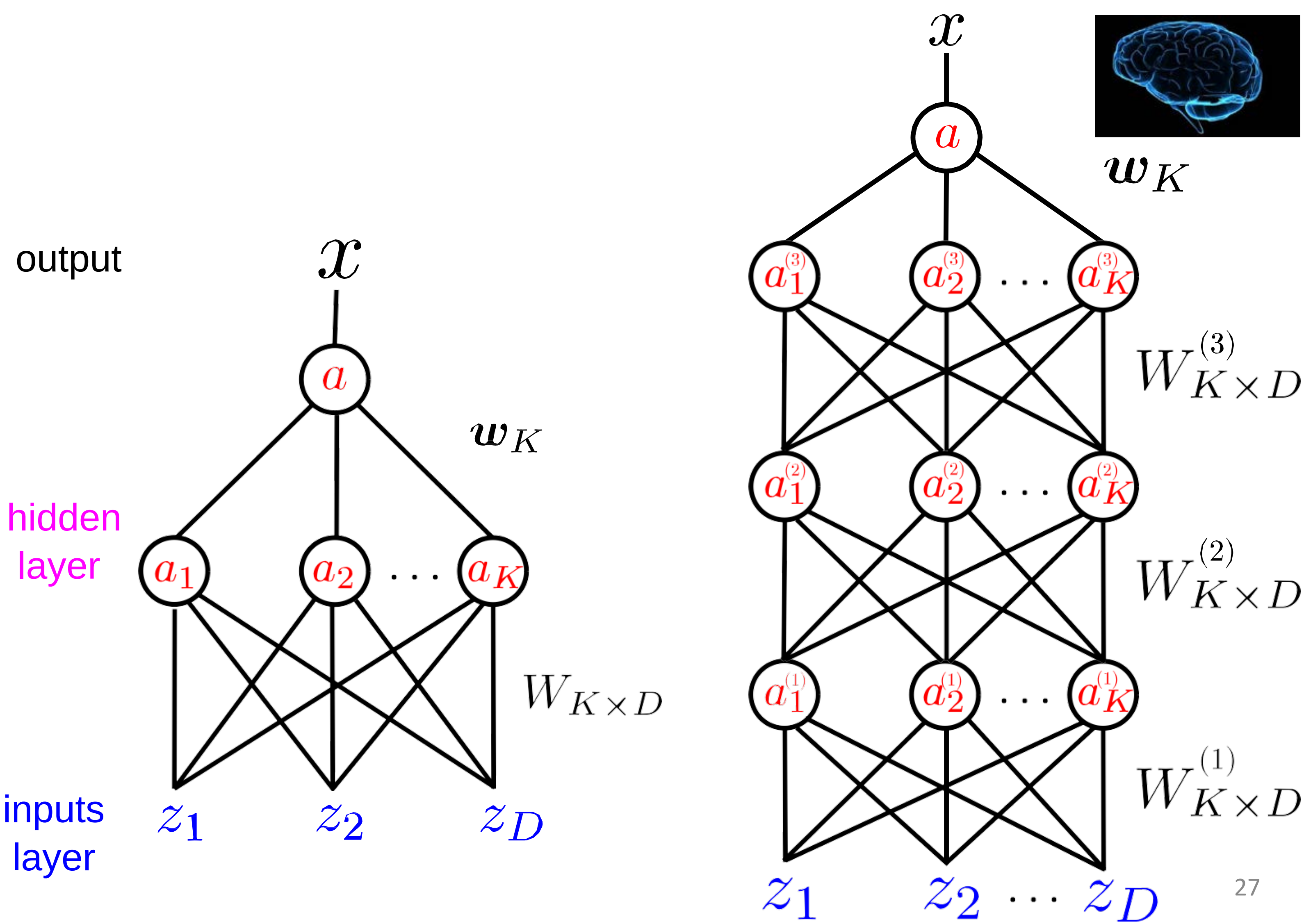
Training a Neural Network with a Single Hidden Layer







Hierarchical Models with Many Layers

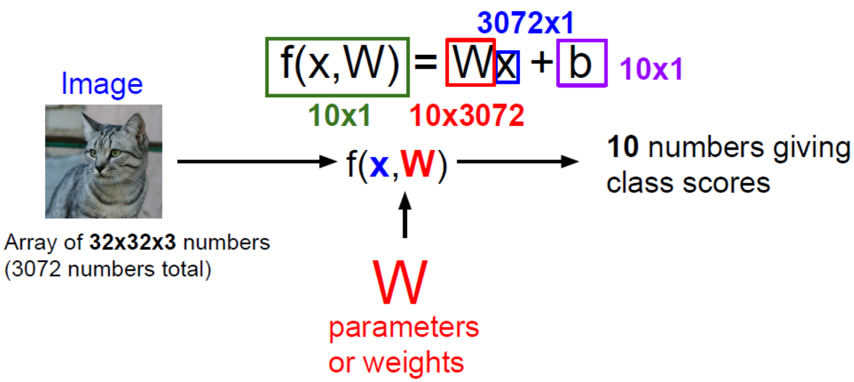


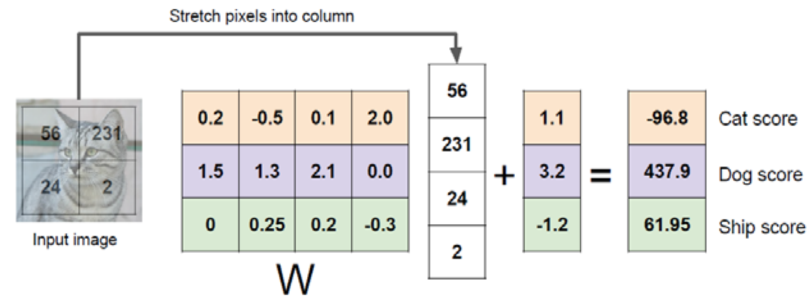
What'd Be Covered in This Crash Course…

- **Learning-based Computer Vision**
  - From Linear to Non-Linear Classifiers
- **Start of Deep Learning for Computer Vision**
  - Convolutional Neural Networks
  - Self-Supervised Learning
  - Segmentation & Detection
- **Generative Models**
  - Autoencoder, Variational Autoencoder,
    Generative Adversarial Networks & Diffusion Models
- **Sequence-to-Sequence Learning**
  - Attention is All You Need: Transformer
- **Vision & Language Foundation Models**
  - Image-to-Text vs. Text-to-Image
  - Parameter-Efficient Fine-tuning

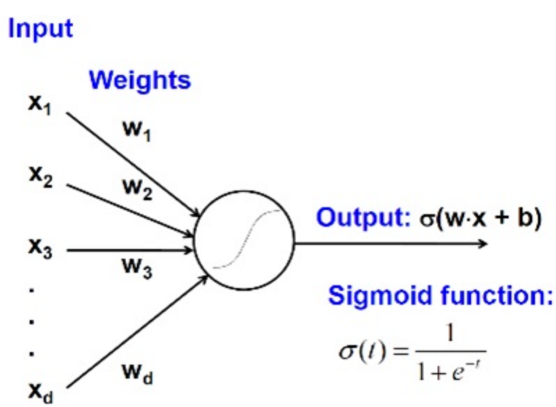
Recap: Linear Classification to Neural Nets

- Linear Classifier





- Neural Network (Multilayer Perceptron)



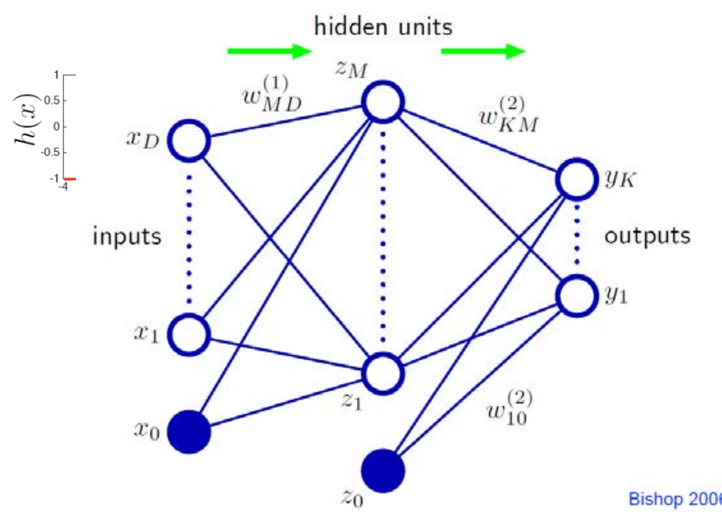
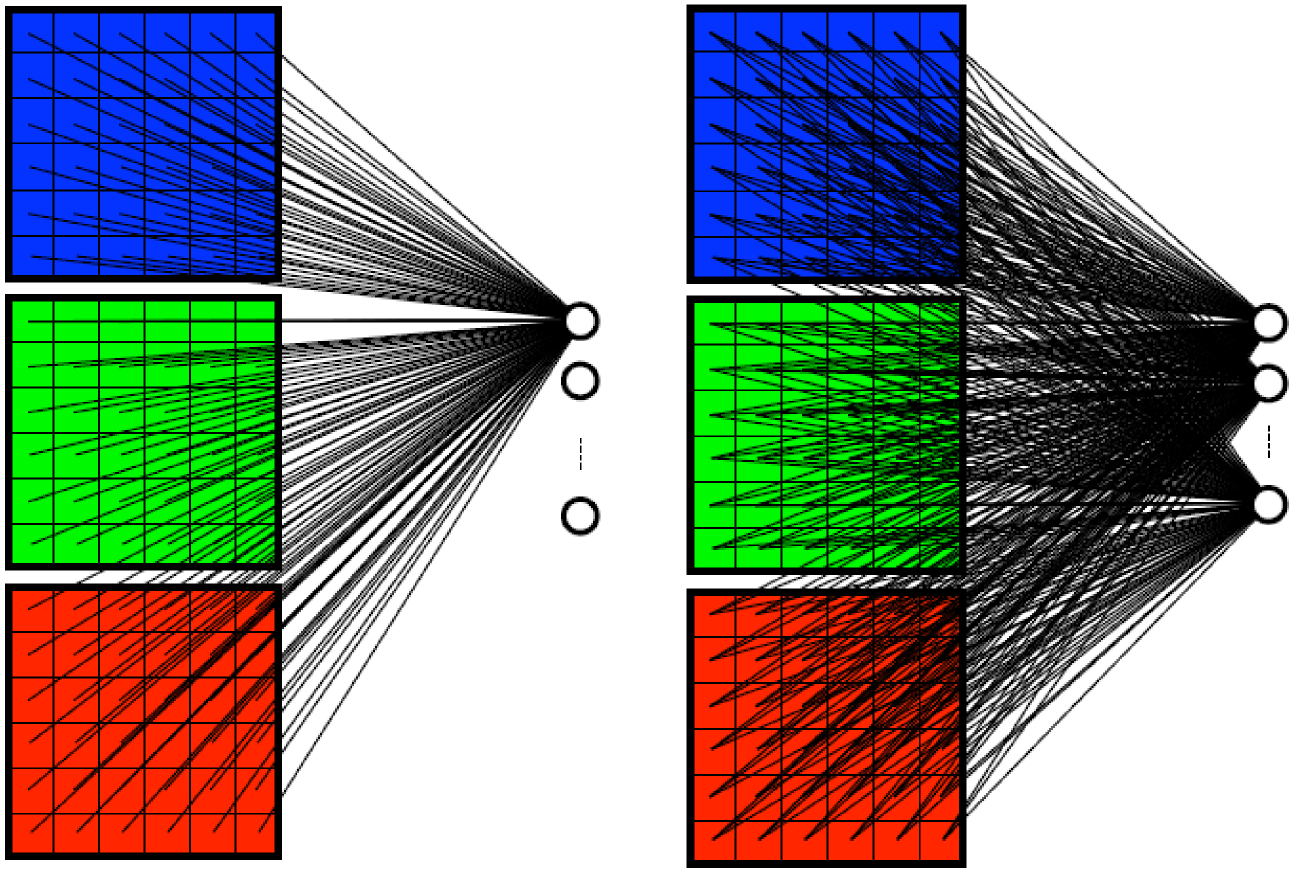


Image credit: Stanford CS231n

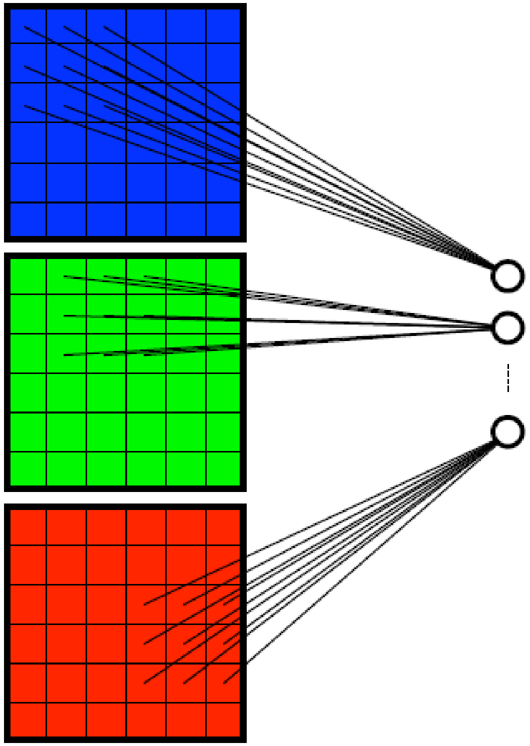
Convolutional Neural Networks

- How many weights for MLPs for images?



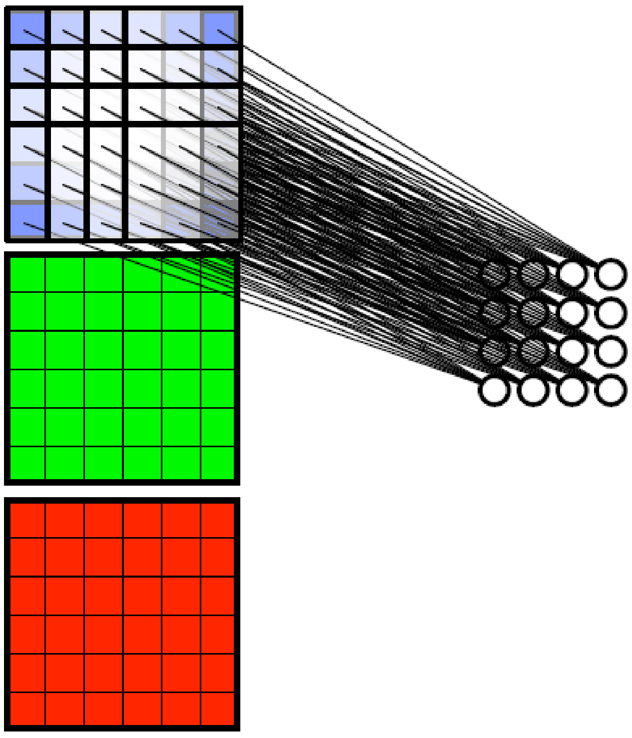
Convolutional Neural Networks

- Property I of CNN: Local Connectivity
  - Each neuron takes info only from a neighborhood of pixels.



Convolutional Neural Networks

- Property II of CNN: Weight Sharing
  - Neurons connecting all neighborhoods have identical weights.



Putting them together → CNN

- Local connectivity
- Weight sharing
- Handling multiple input channels
- Handling multiple output maps

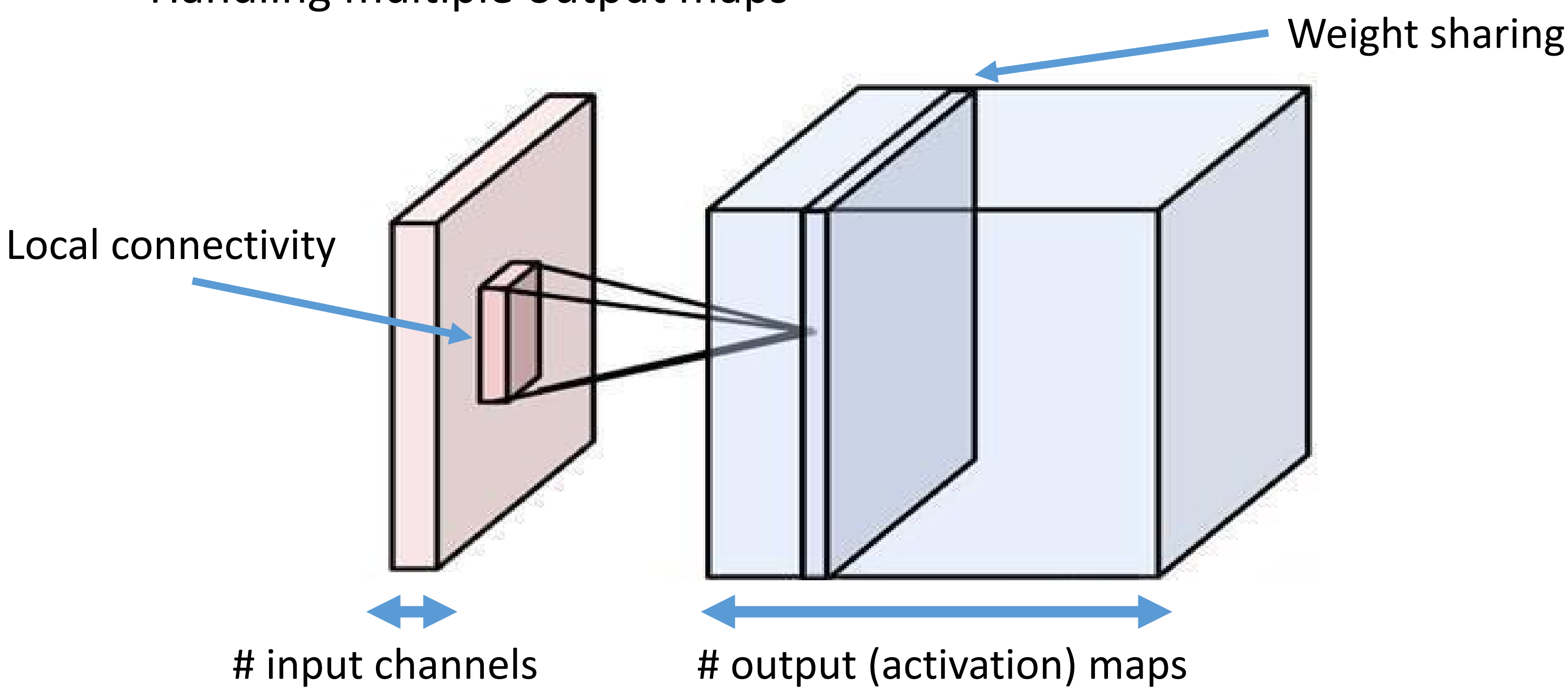
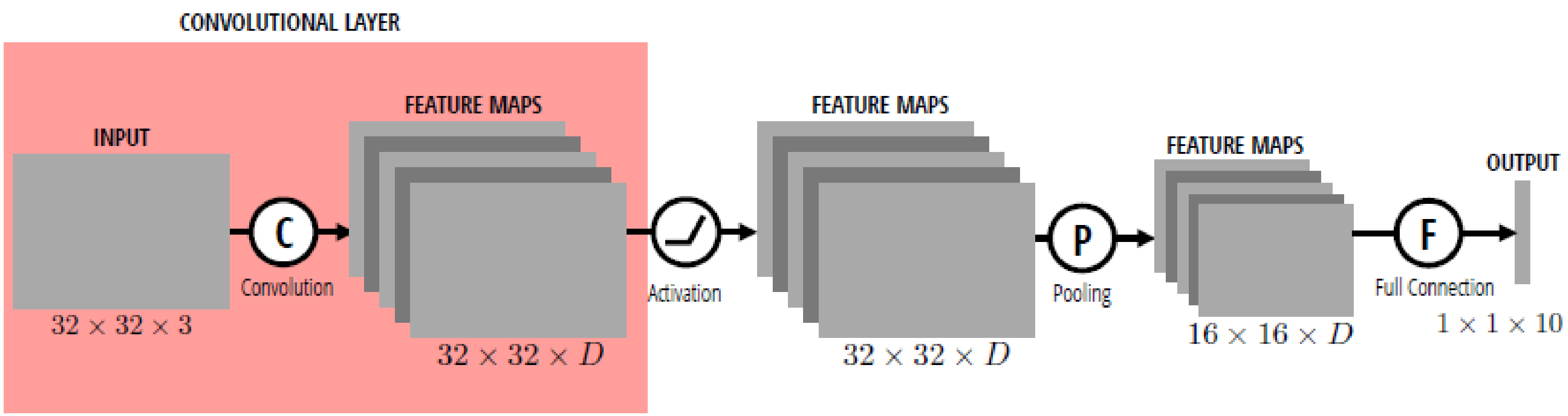
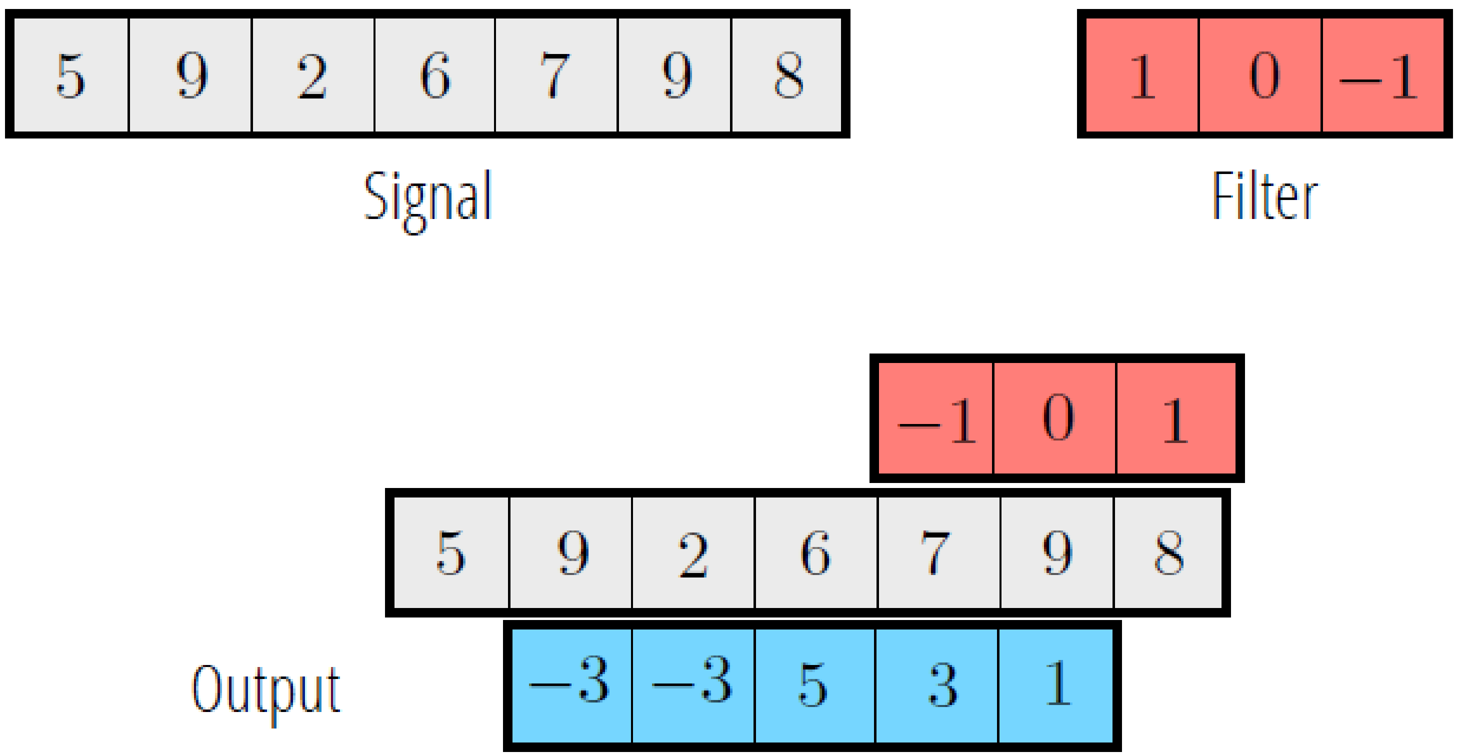


Image credit: A. Karpathy

Convolution Layer in CNN



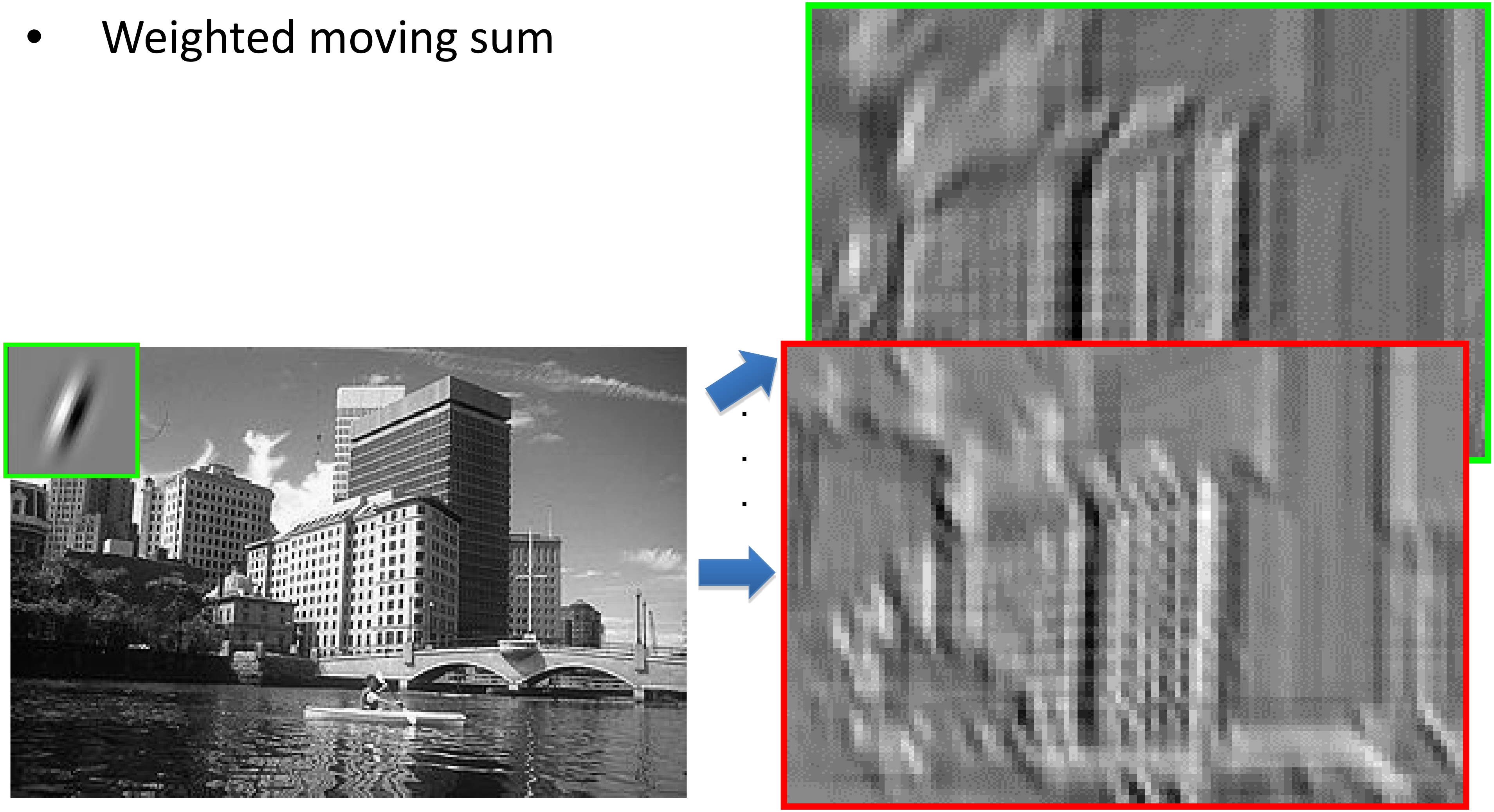
What is a Convolution?



Convolution is a local linear operator

What is a Convolution?

- Weighted moving sum



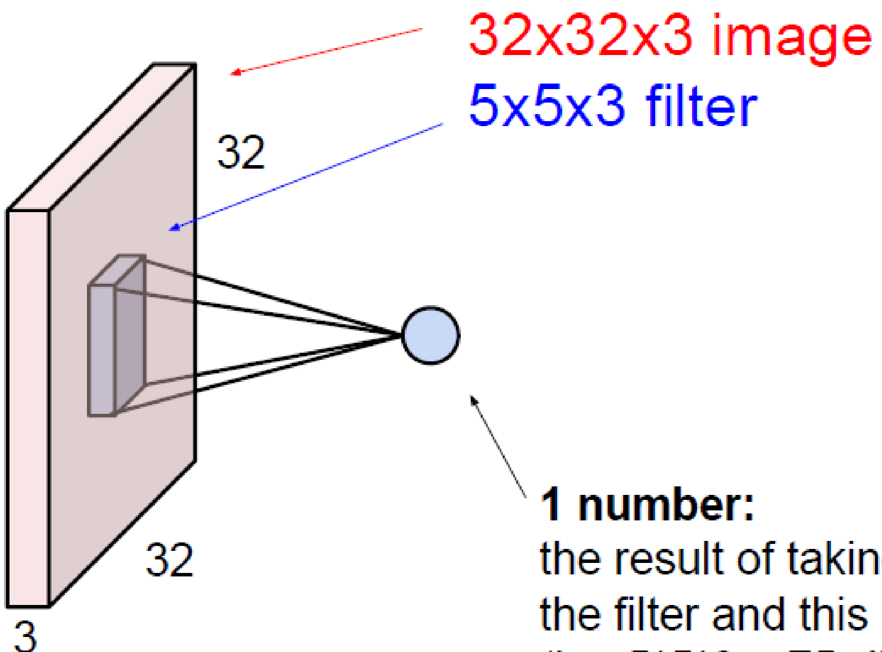
Input

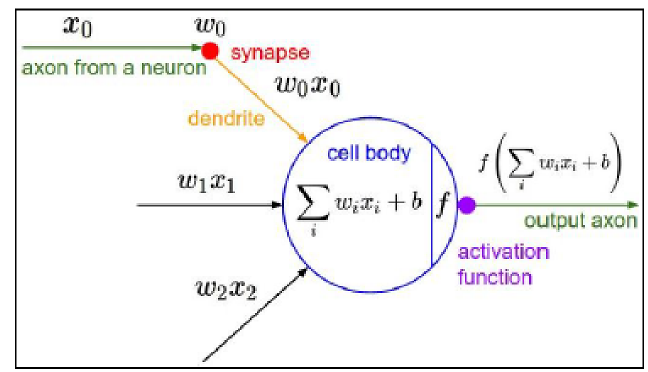
Feature Activation Map

slide credit: S. Lazebnik

Putting them together

- The brain/neuron view of CONV layer



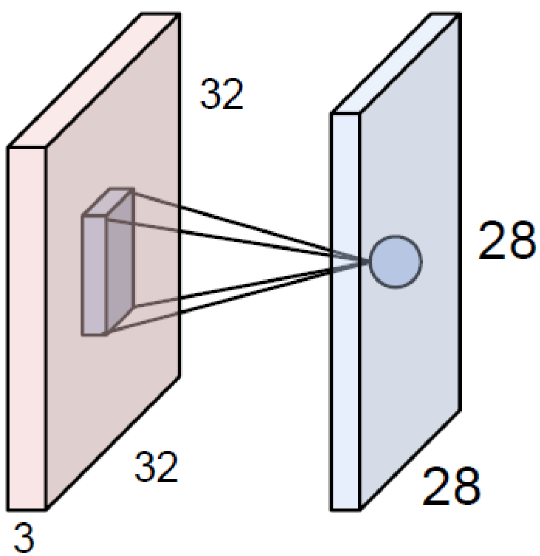


It's just a neuron with local connectivity...

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

Putting them together (cont'd)

- The brain/neuron view of CONV layer



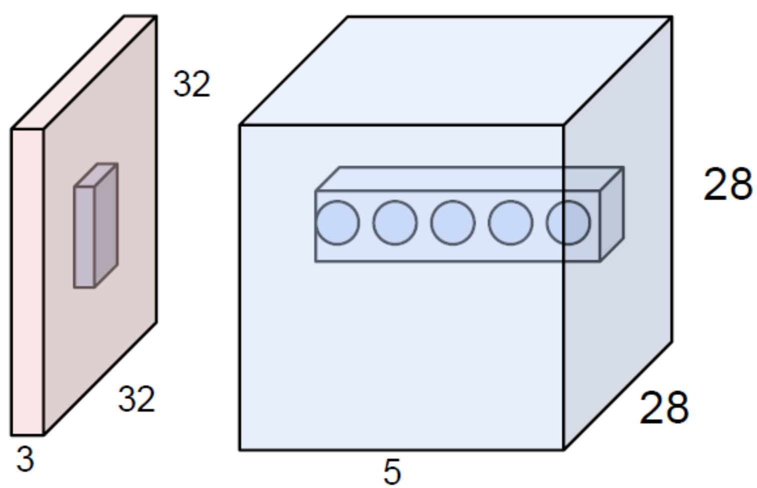
An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

Putting them together (cont'd)

- The brain/neuron view of CONV layer

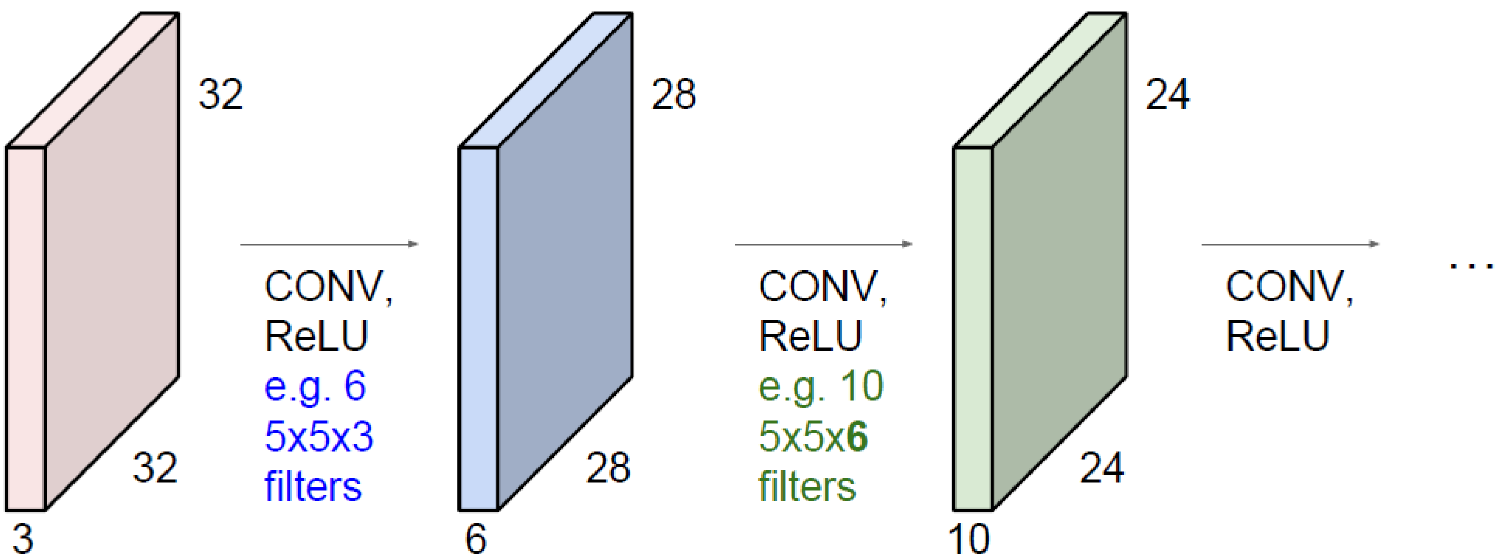


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

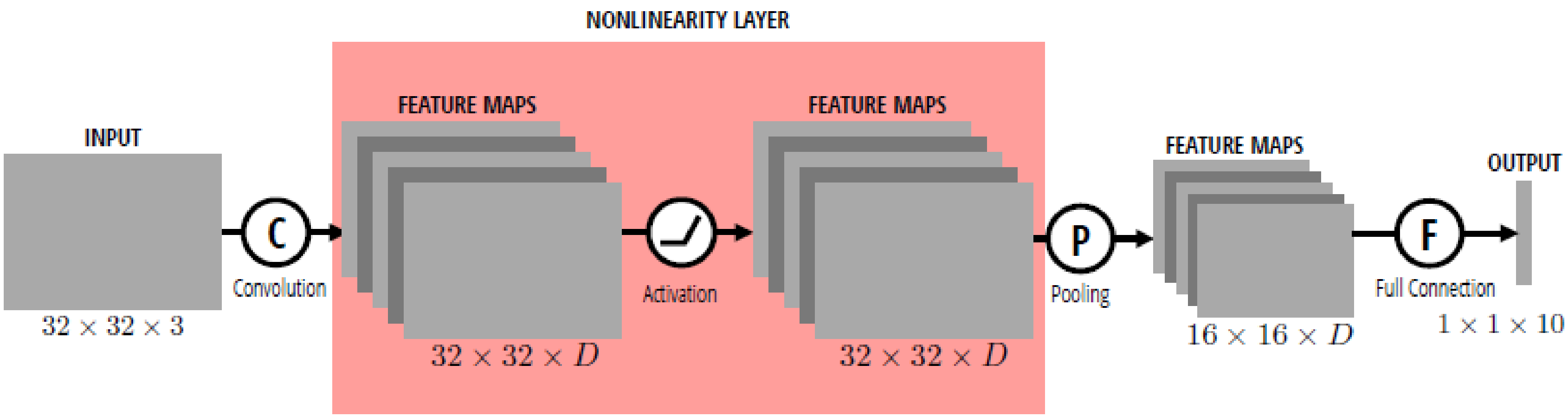
There will be 5 different
neurons all looking at the same
region in the input volume

Putting them together (cont'd)

- Image input with 32 x 32 pixels convolved repeatedly with 5 x 5 x 3 filters shrinks volumes spatially (32 -> 28 -> 24 -> ...).

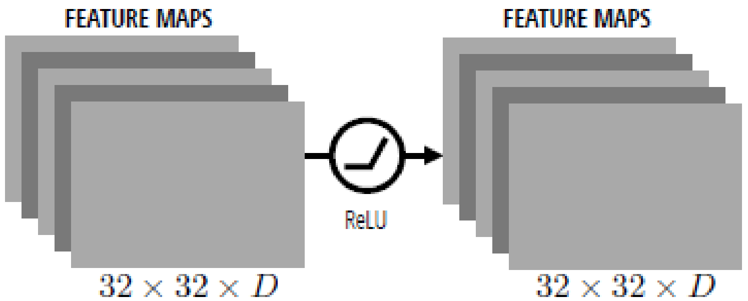


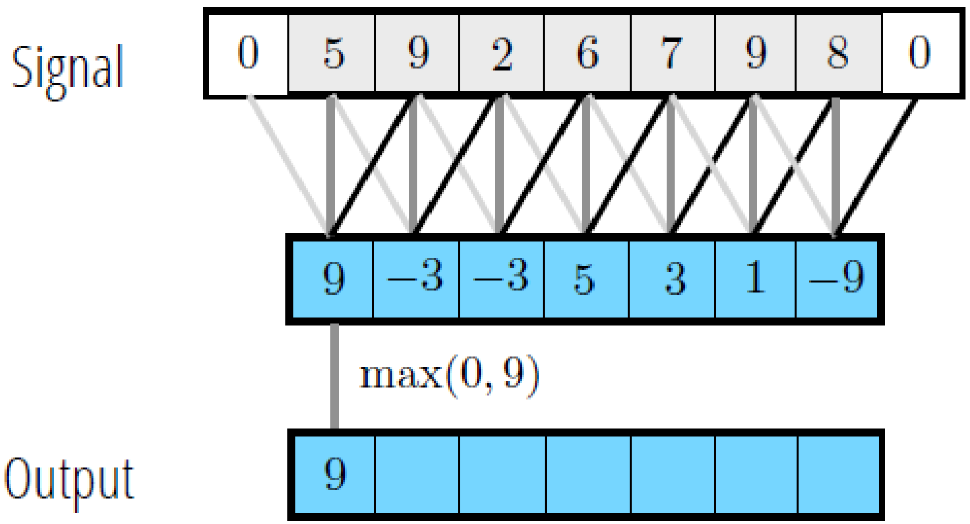
Nonlinearity Layer in CNN



Nonlinearity Layer

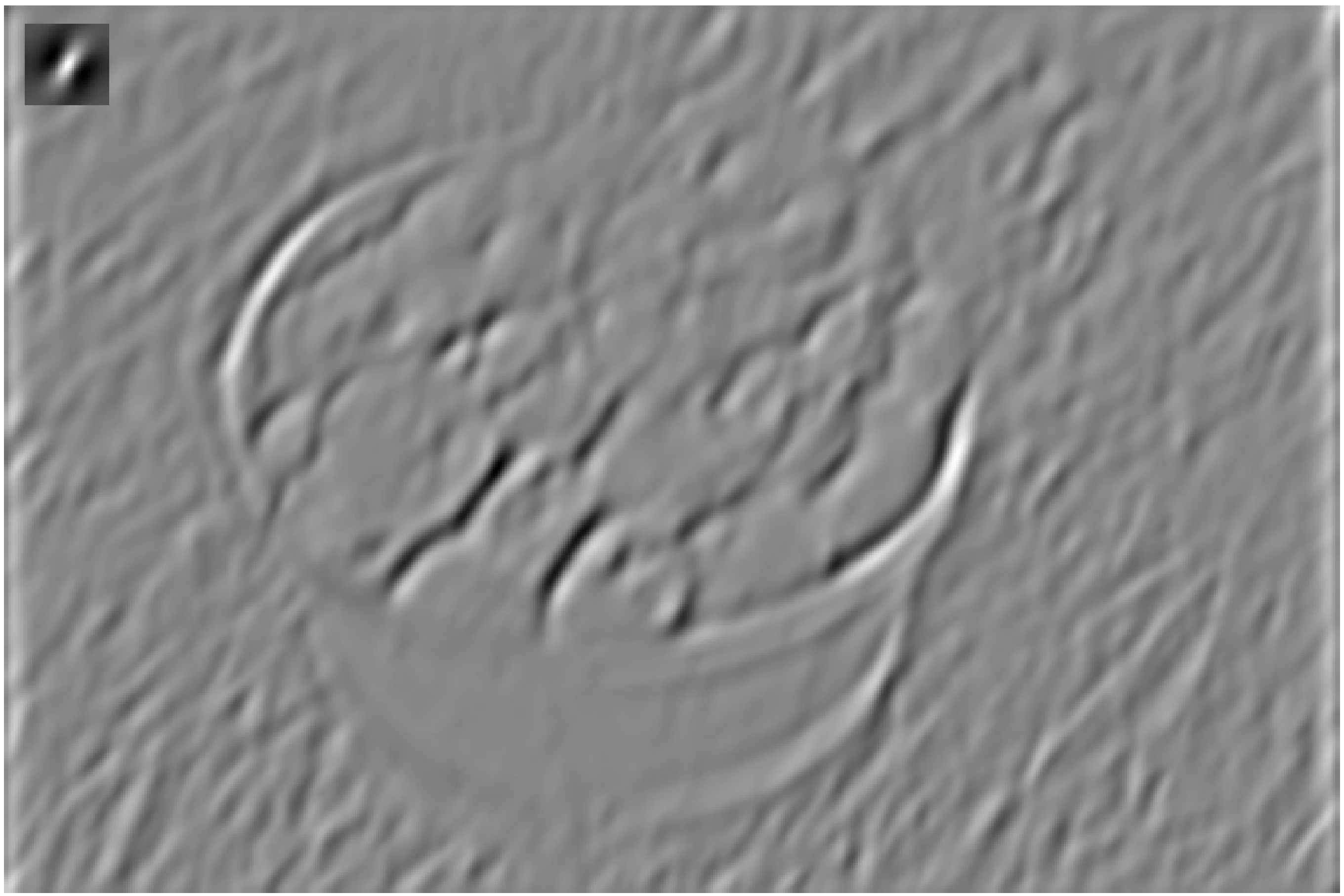
- E.g., ReLU (Rectified Linear Unit)
  - Pixel by pixel computation of max(0, x)





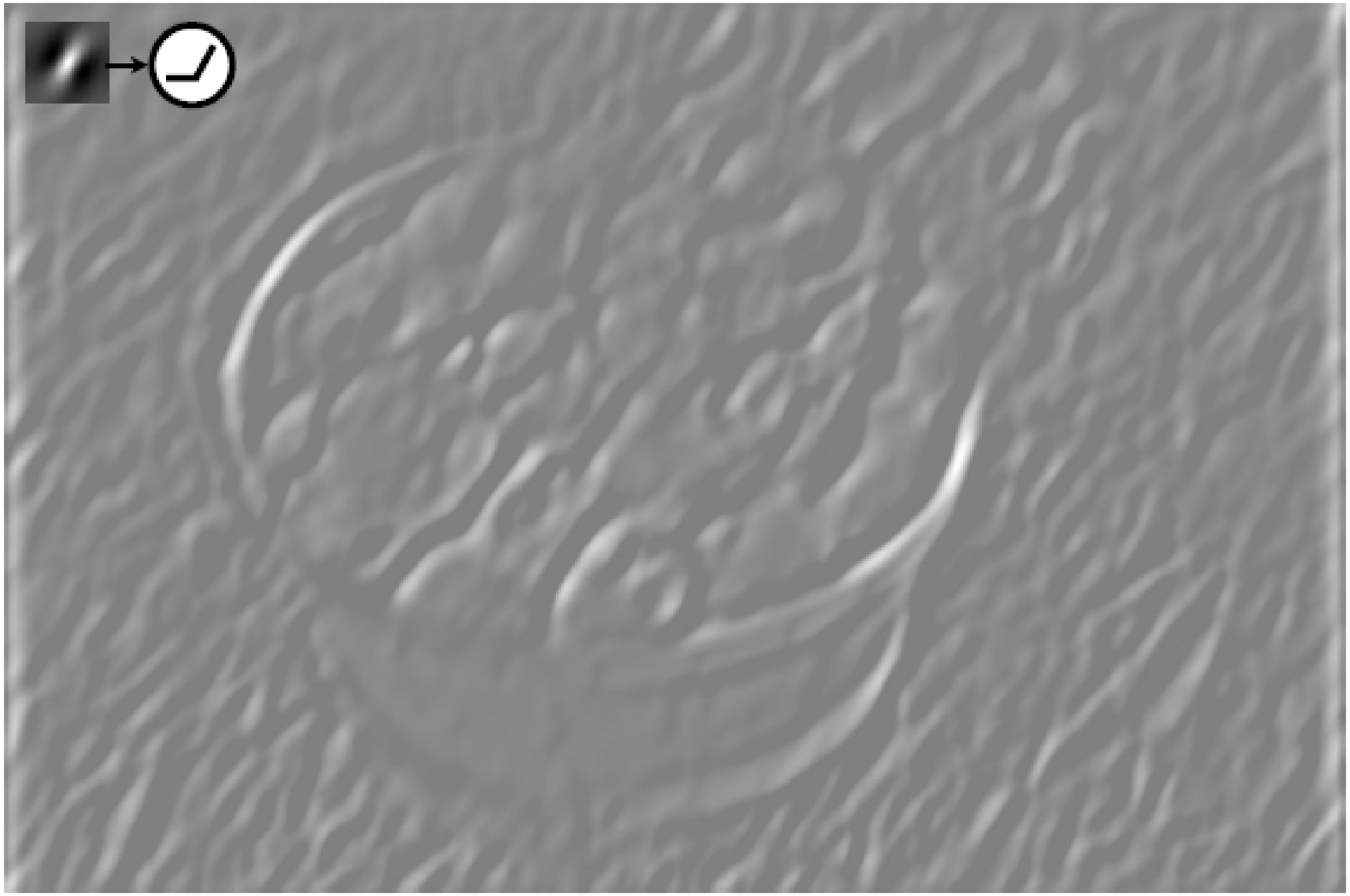
Nonlinearity Layer

- E.g., ReLU (Rectified Linear Unit)
  - Pixel by pixel computation of max(0, x)

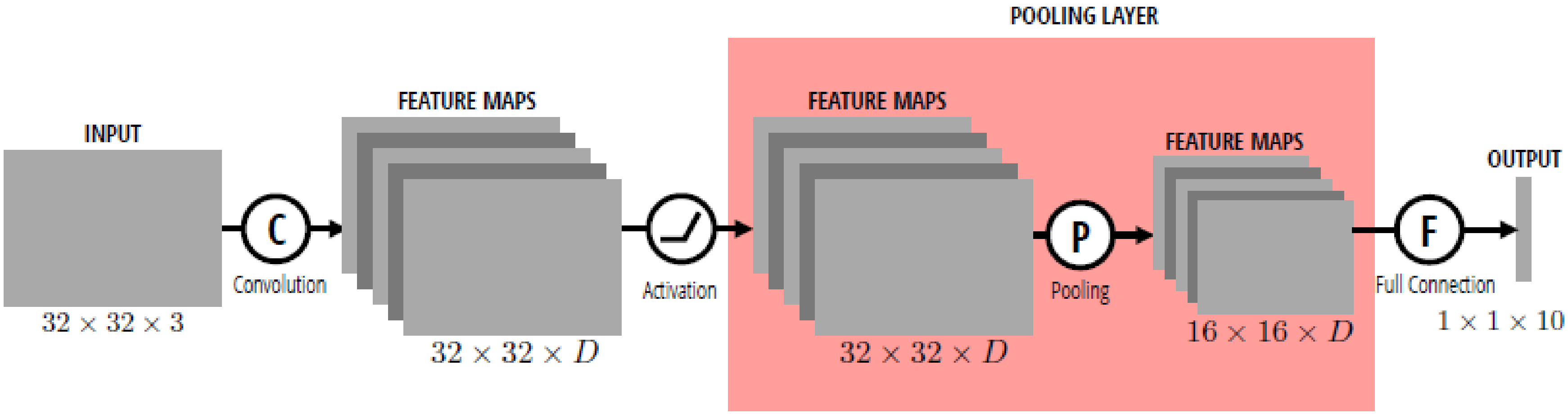


Nonlinearity Layer

- E.g., ReLU (Rectified Linear Unit)
  - Pixel by pixel computation of max(0, x)

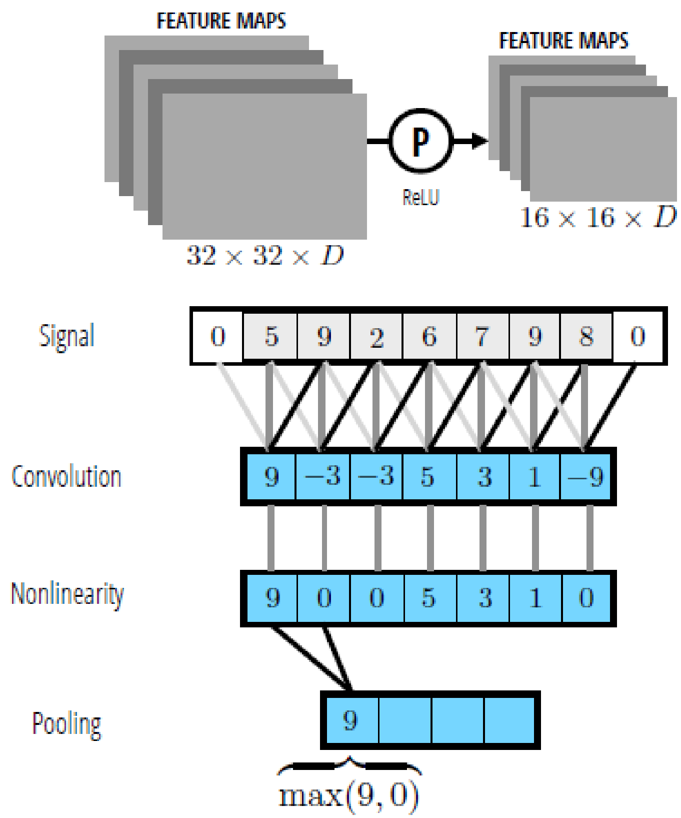


Pooling Layer in CNN



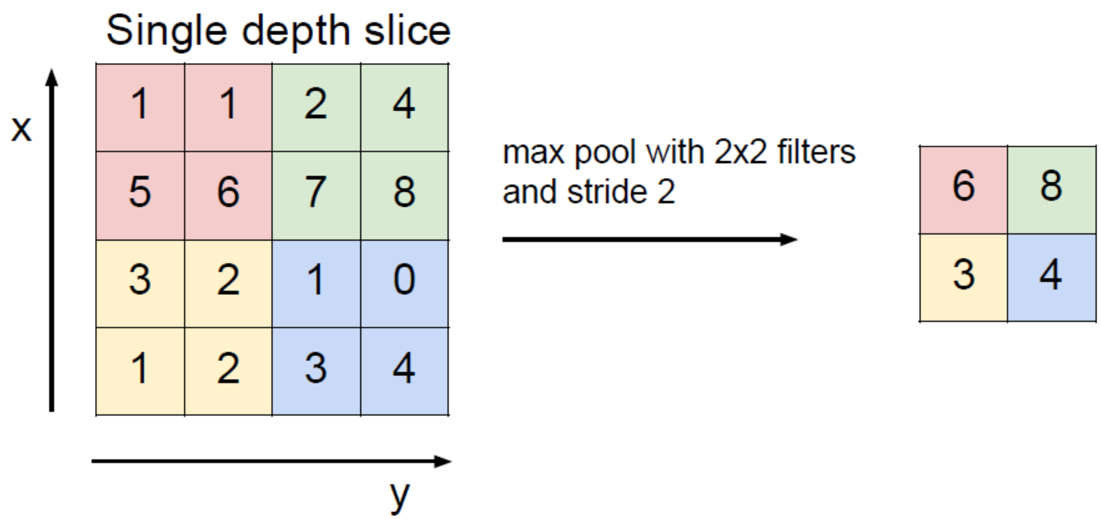
Pooling Layer

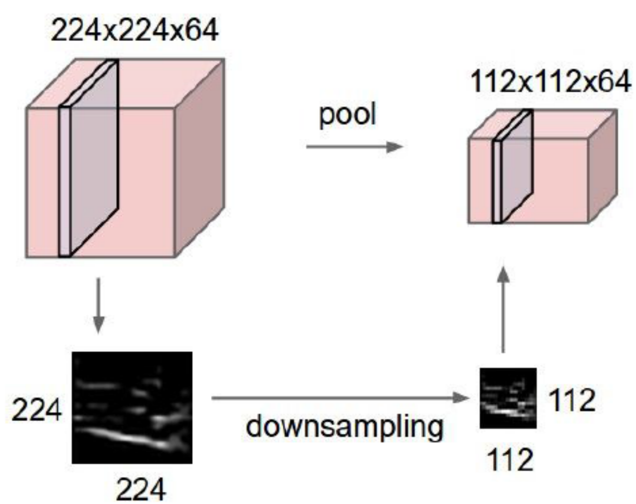
- Makes the representations smaller and more manageable
- Operates over each activation map independently
- E.g., Max Pooling



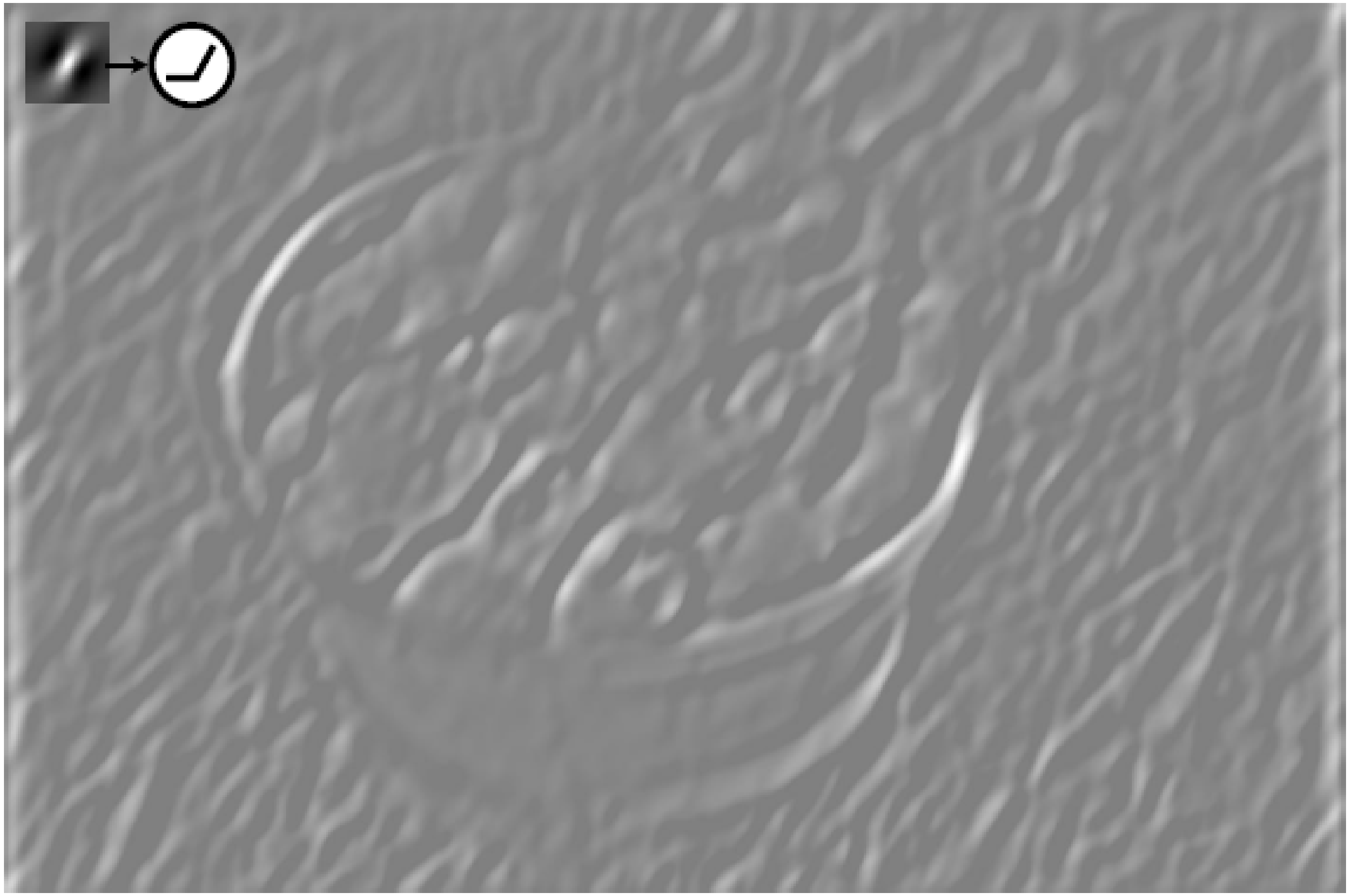
Pooling Layer

- Reduces the spatial size and provides spatial invariance

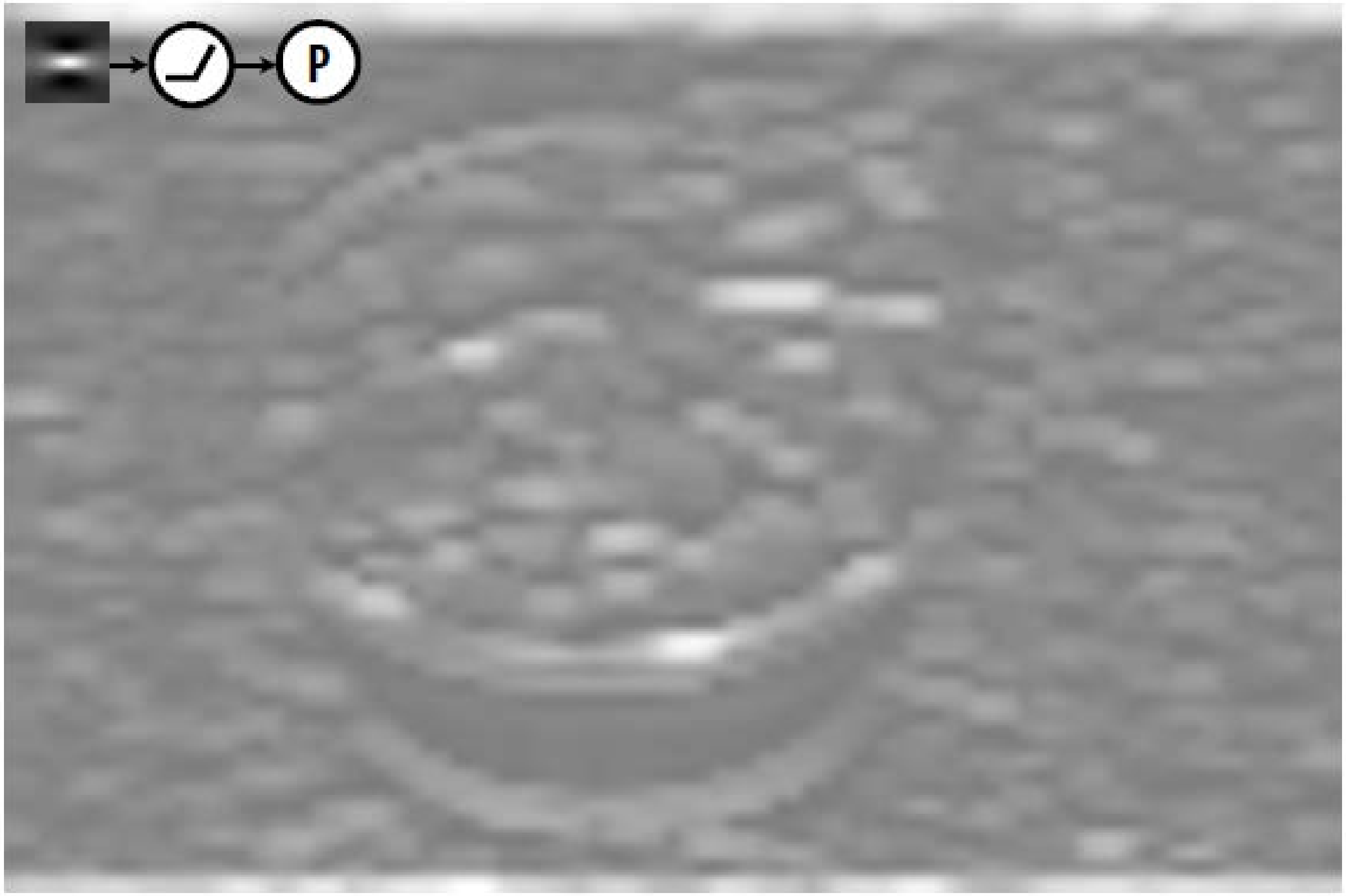




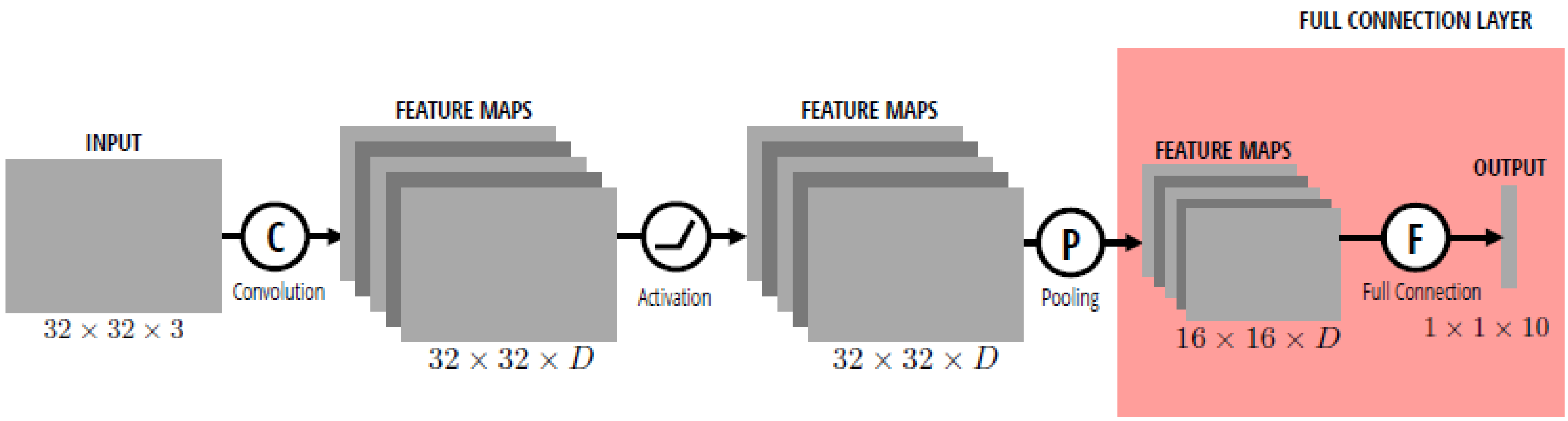
- Example
  - Nonlinearity by ReLU



- Example
  - Max pooling

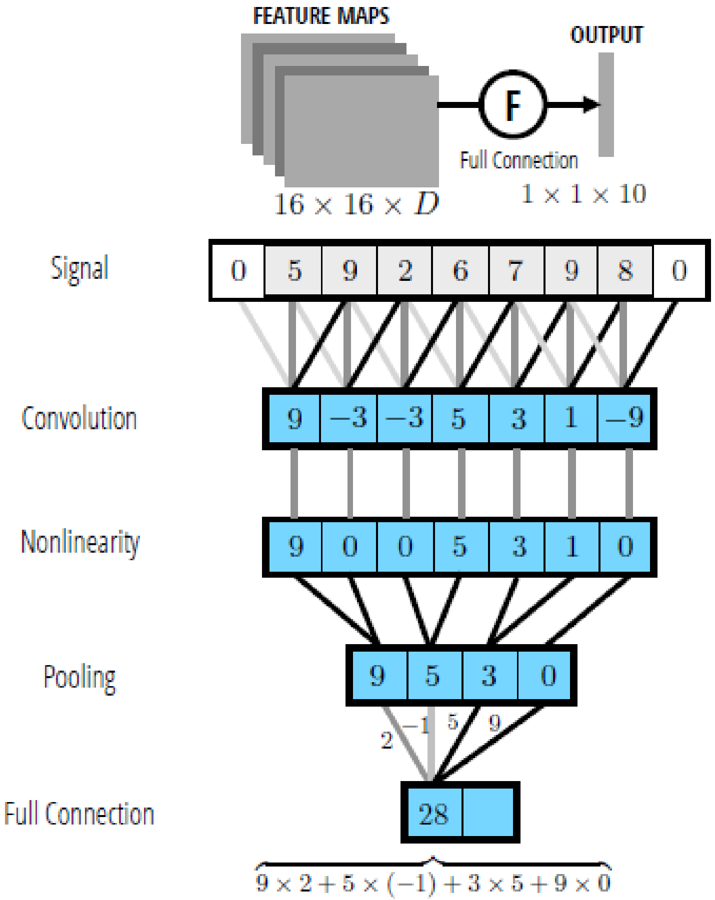


Fully Connected (FC) Layer in CNN



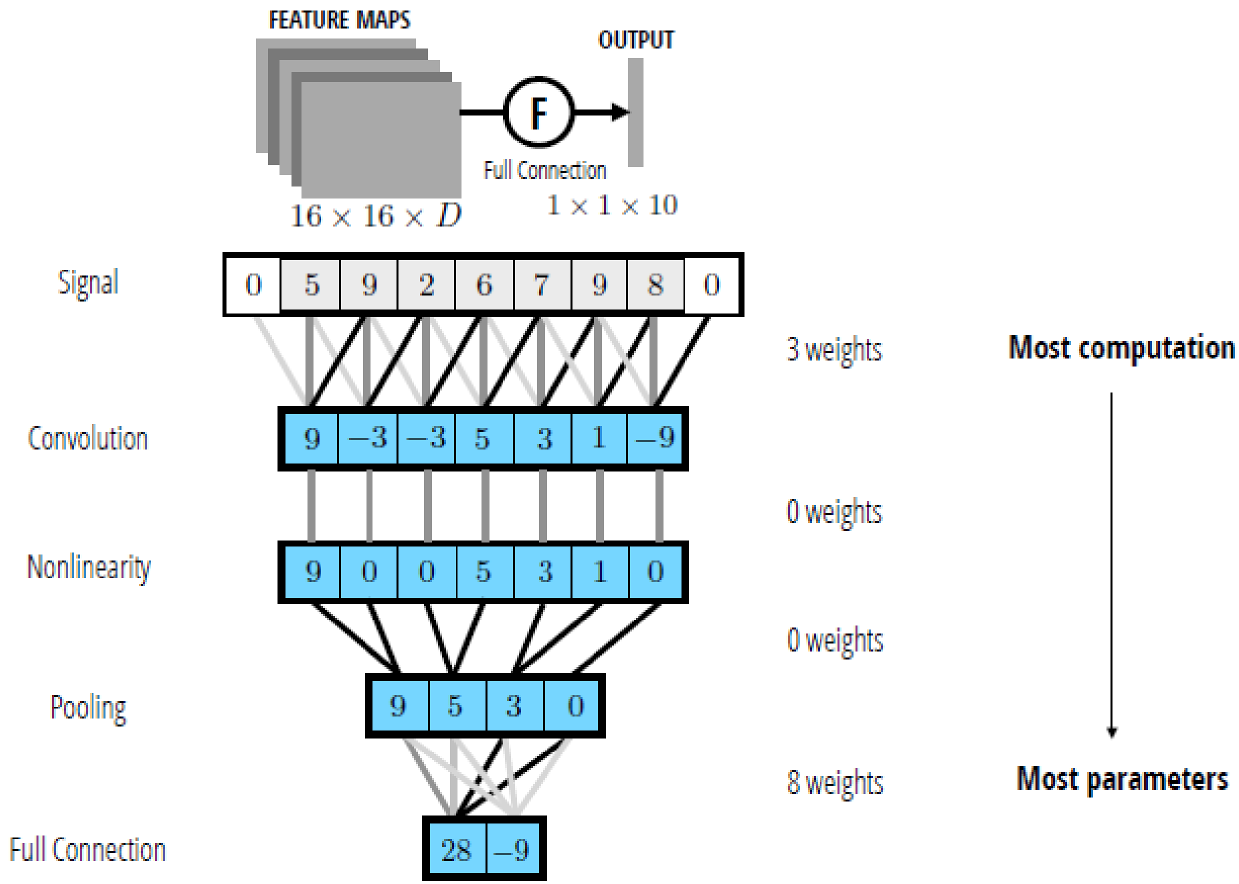
FC Layer

- Contains neurons that connect to the entire input volume, as in ordinary neural networks

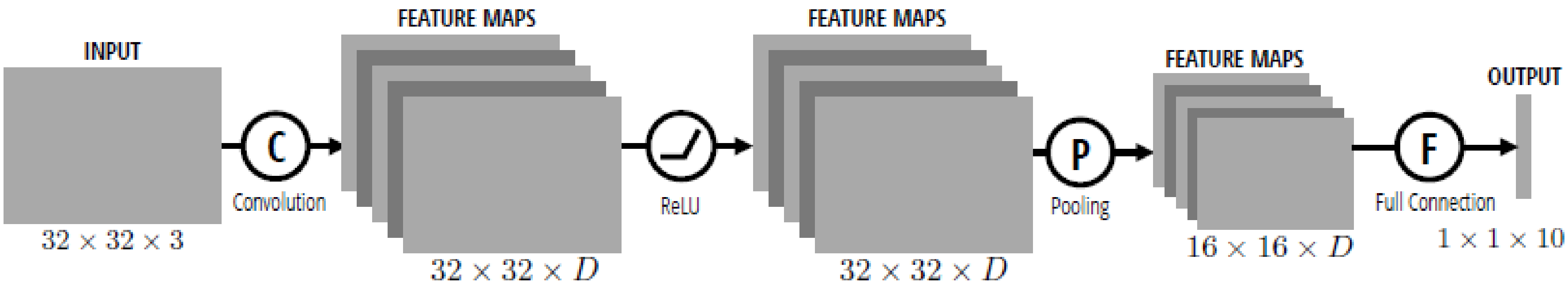


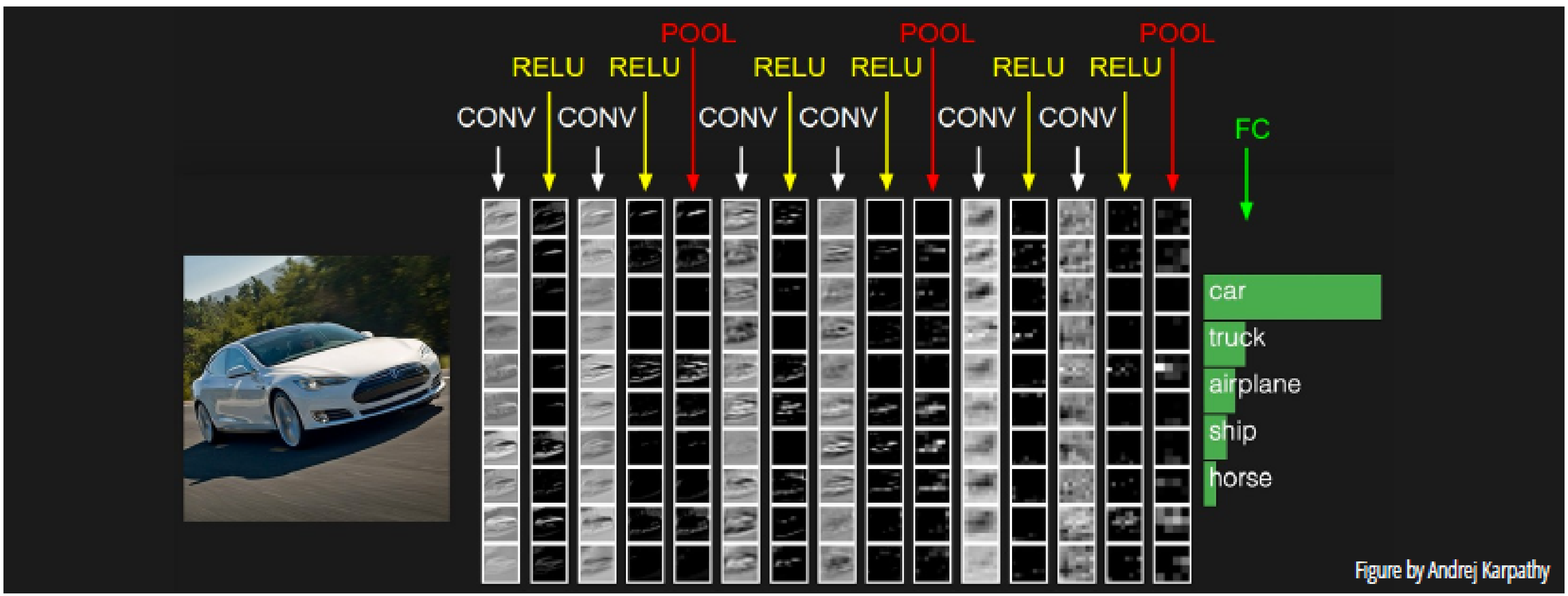
FC Layer

- Contains neurons that connect to the entire input volume, as in ordinary neural networks

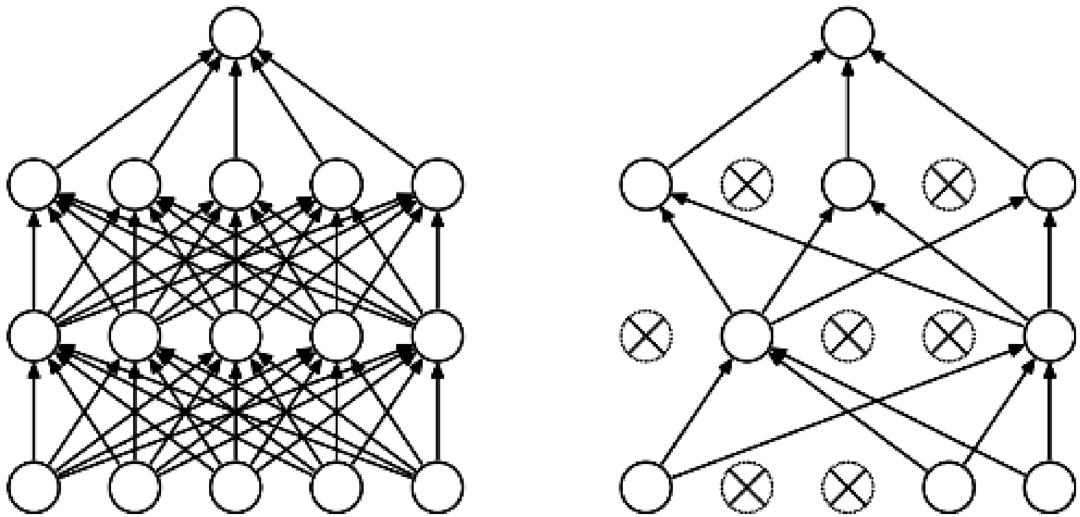


CNN


INPUT
Convolution
FEATURE MAPS
ReLU
FEATURE MAPS
Pooling
FEATURE MAPS
Full Connection
OUTPUT
$32 \times 32 \times 3$
$32 \times 32 \times D$
$32 \times 32 \times D$
$16 \times 16 \times D$
$1 \times 1 \times 10$


POOL
RELU
CONV
FC
car
truck
airplane
ship
horse
Figure by Andrej Karpathy

Training Technique #1: Dropout



(a) Standard Neural Net (b) After applying dropout.

Intuition: successful **conspiracies**

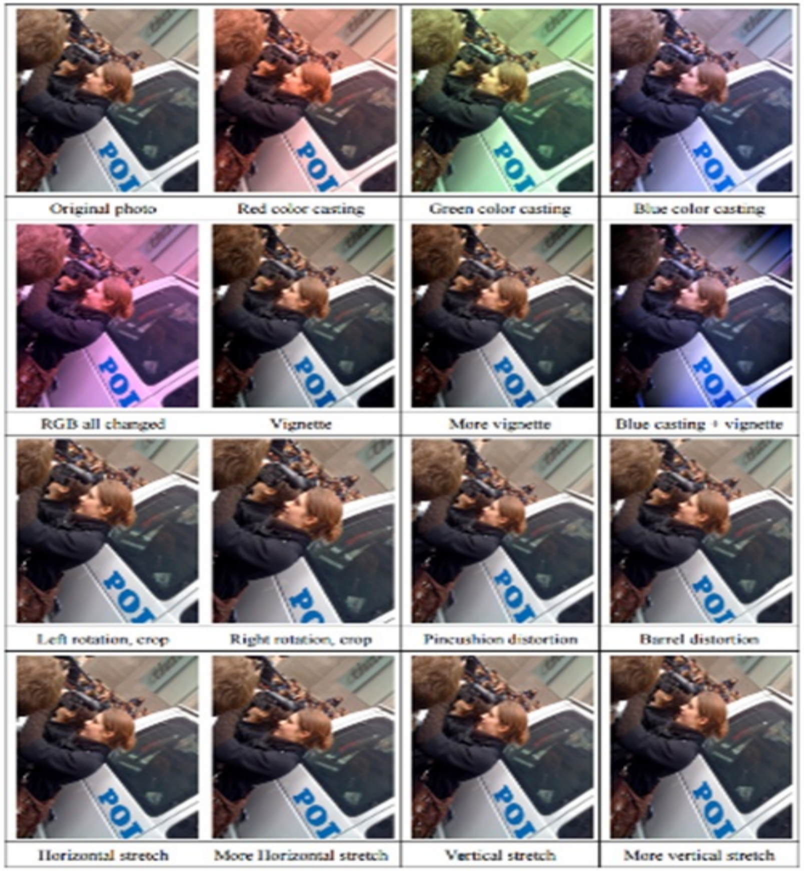
Example: 50 people planning a conspiracy

- Strategy A: plan a big conspiracy involving 50 people
  - Likely to fail. 50 people need to play their parts correctly.
- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

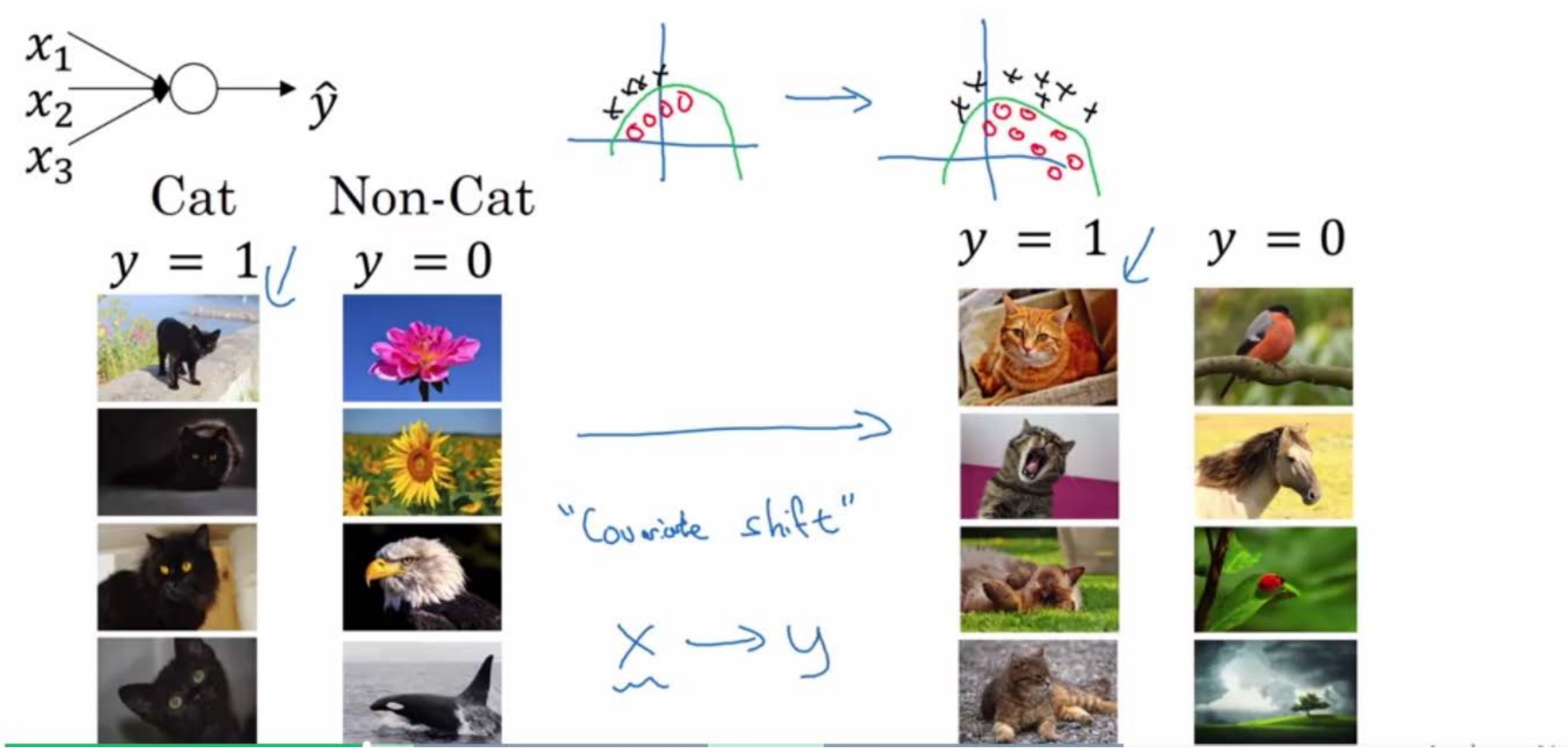
Training Technique #2: Data Augmentation

- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion



Deep Image [Wu et al. 2015]

Training Technique #3: Batch Normalization



Credit: Andrew Ng

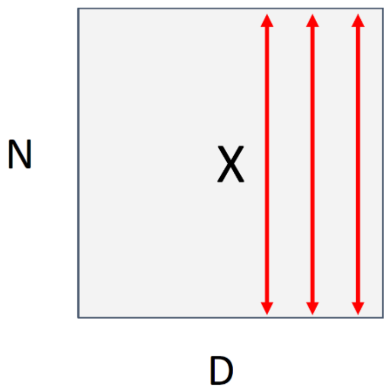
Batch Normalization (cont'd)

- Remarks
  - Differentiable function; back propagation OK

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

  - Procedure

Input: $x : N \times D$



$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$ Per-channel mean across N samples

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$ Per-channel std across N samples

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ Normalized x, Shape is N x D

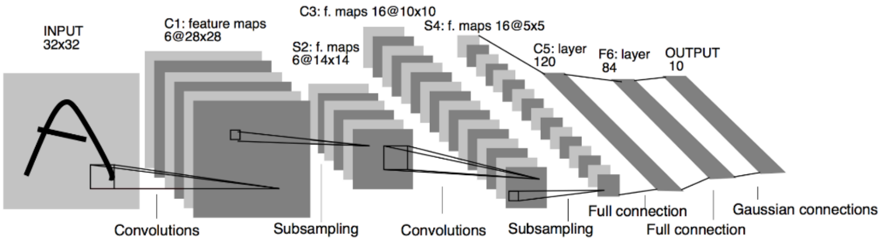
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

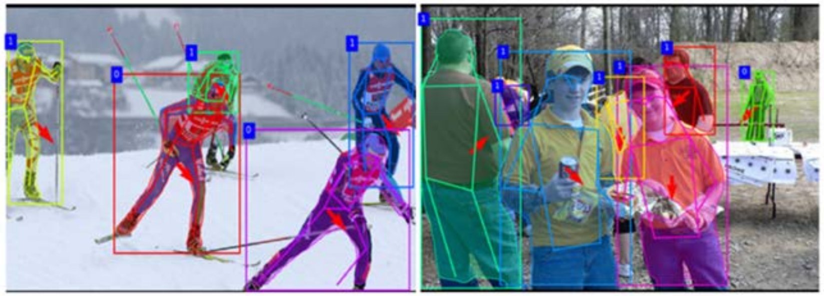
What'd Be Covered in This Crash Course…

- **Learning-based Computer Vision**
  - From Linear to Non-Linear Classifiers
- **Start of Deep Learning for Computer Vision**
  - Convolutional Neural Networks
  - Self-Supervised Learning
  - Segmentation & Detection
- **Generative Models**
  - Autoencoder, Variational Autoencoder,
    Generative Adversarial Networks & Diffusion Models
- **Sequence-to-Sequence Learning**
  - Attention is All You Need: Transformer
- **Vision & Language Foundation Models**
  - Image-to-Text vs. Text-to-Image
  - Parameter-Efficient Fine-tuning

Supervised Learning

- Deep learning plus supervised learning are rocking the world ...





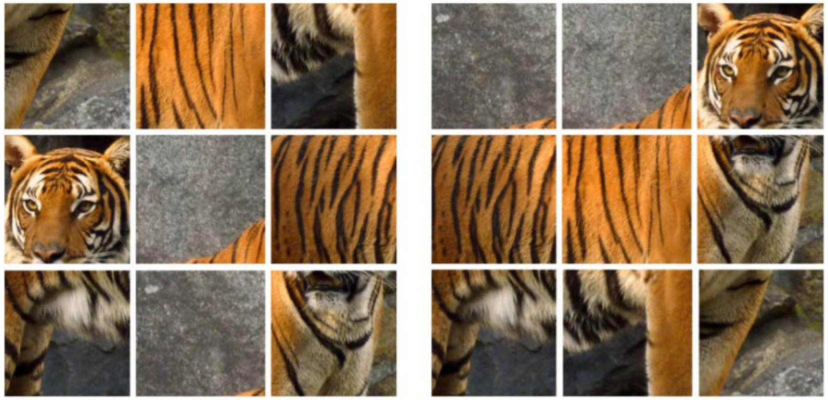
- In real world scenarios, data-annotation is quite **time-consuming**
- Could one exploit supervised signals from **unlabeled** data?

Self-Supervised Learning (SSL)

- Learning discriminative representations from **unlabeled** data
- Create self-supervised tasks via **data augmentation**

Colorization

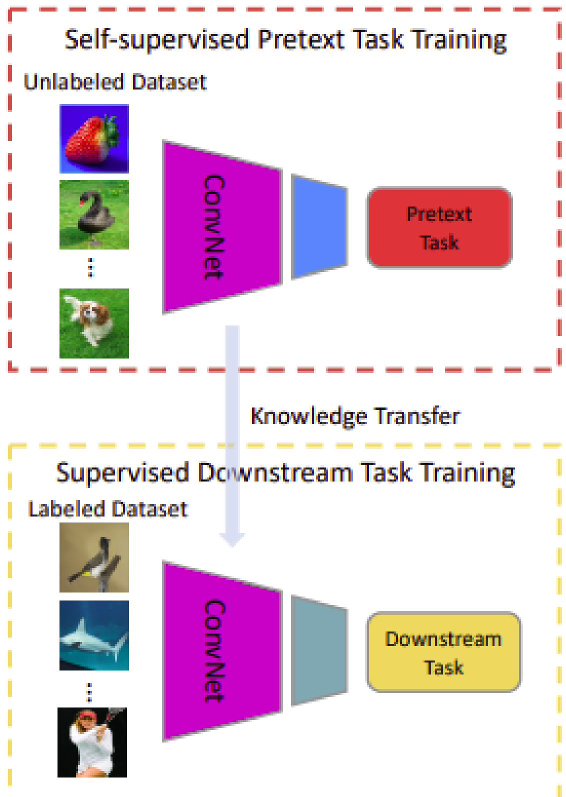
Rotation



Jigsaw Puzzle

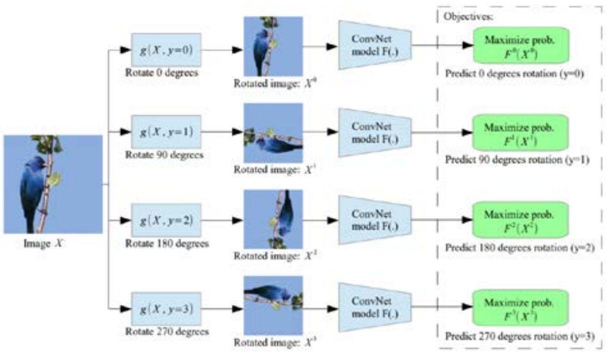
Self-Supervised Learning (SSL)

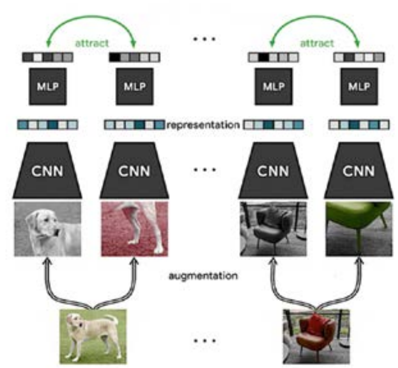
- Self-Supervised Pretraining
- Supervised Fine-tuning

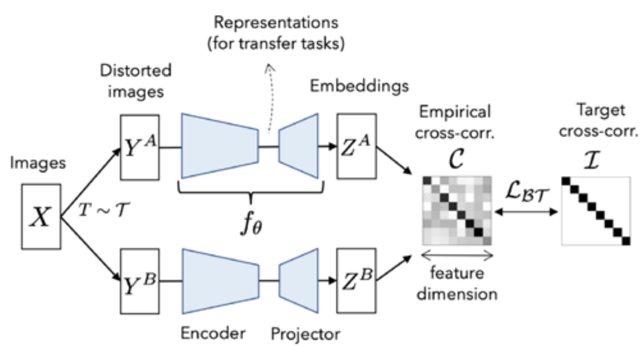


Self-Supervised Learning (SSL)

- Pretext Tasks
  - Jigsaw (ECCV’16)
  - RotNet (ICLR’18)
- Contrastive Learning
  - CPC (ICML’20)
  - SimCLR (ICML’20)
- Learning w/o negative samples
  - BYOL (NeurIPS’20)
  - Barlow Twins (ICML’21)

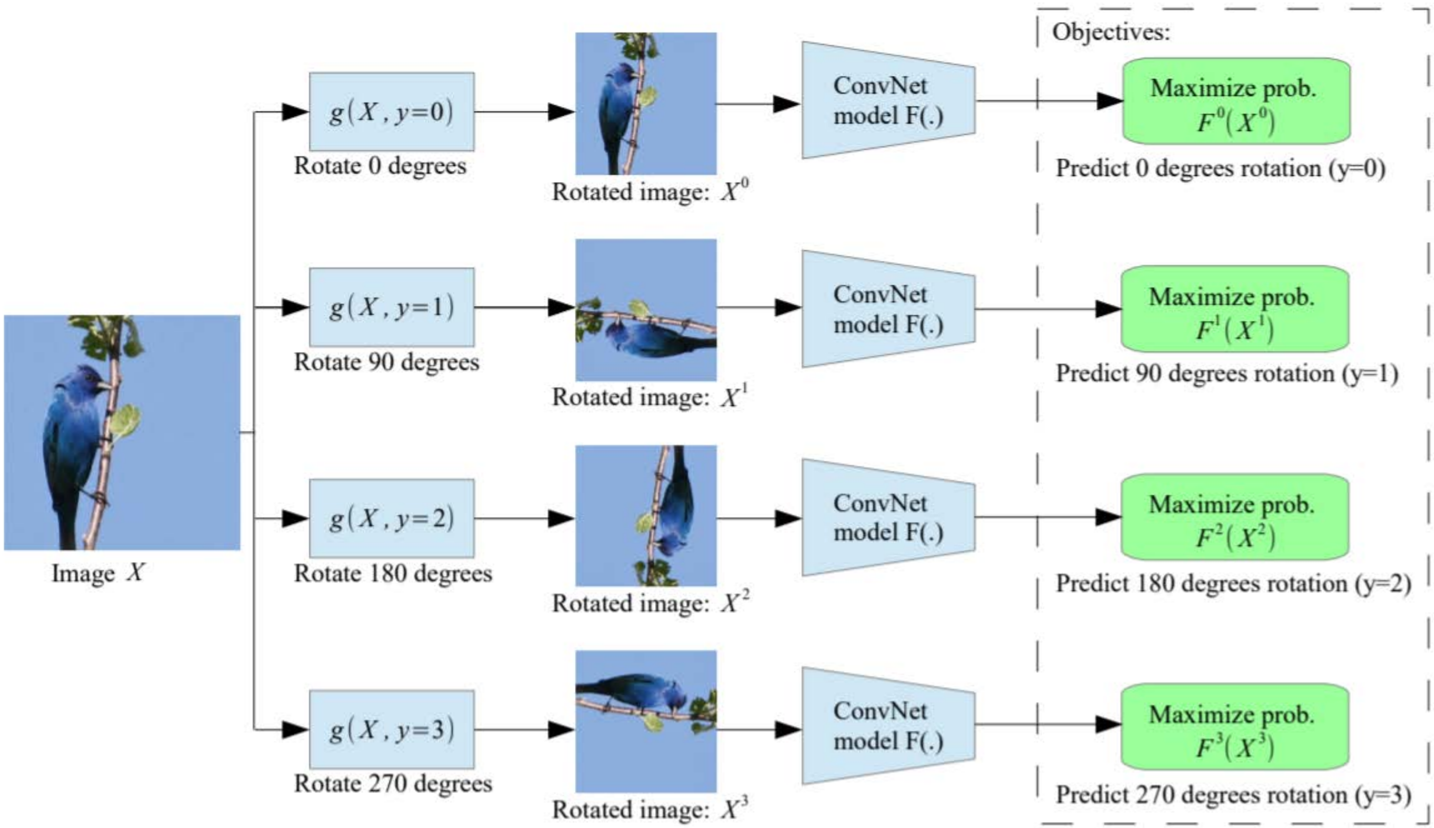






RotNet

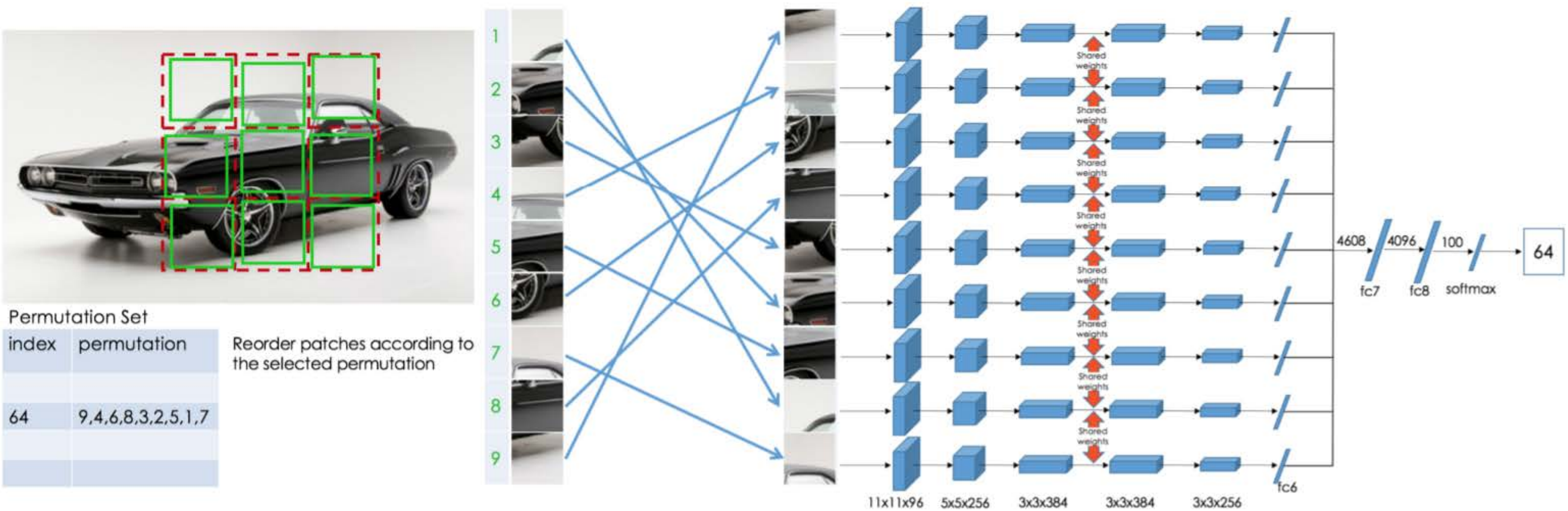
- Learning to predict the **rotation** angle



Gidaris et al. "Unsupervised Representation Learning by Predicting Image Rotations." ICLR 2018

Jigsaw Puzzle

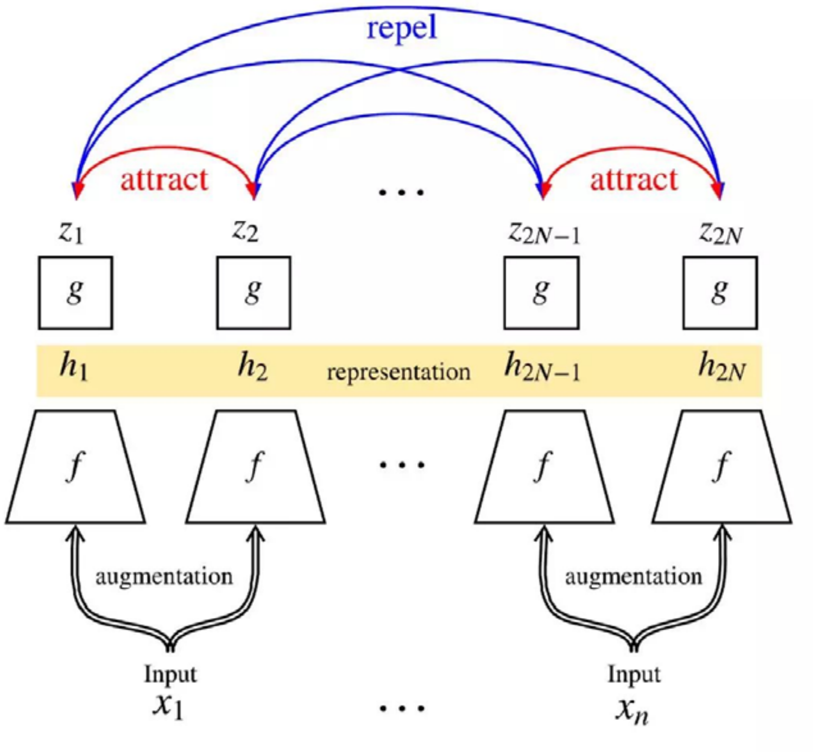
- Assign the **permutation index** and perform augmentation
- Solve jigsaw puzzle by predicting the permutation index



Noroozi et al. "Unsupervised learning of visual representations by solving jigsaw puzzles." ECCV 2016

SimCLR

- **Attract** augmented images and **repel** negative samples
- Improve the representation quality with **projection heads** (*g*)…why?



Chen et al. "A simple framework for contrastive learning of visual representations." ICML 2020

What'd Be Covered in This Crash Course…

- **Learning-based Computer Vision**
  - From Linear to Non-Linear Classifiers
- **Start of Deep Learning for Computer Vision**
  - Convolutional Neural Networks
  - Self-Supervised Learning
  - Segmentation & Detection
- **Generative Models**
  - Autoencoder, Variational Autoencoder,
    Generative Adversarial Networks & Diffusion Models
- **Sequence-to-Sequence Learning**
  - Attention is All You Need: Transformer
- **Vision & Language Foundation Models**
  - Image-to-Text vs. Text-to-Image
  - Parameter-Efficient Fine-tuning

Image Segmentation

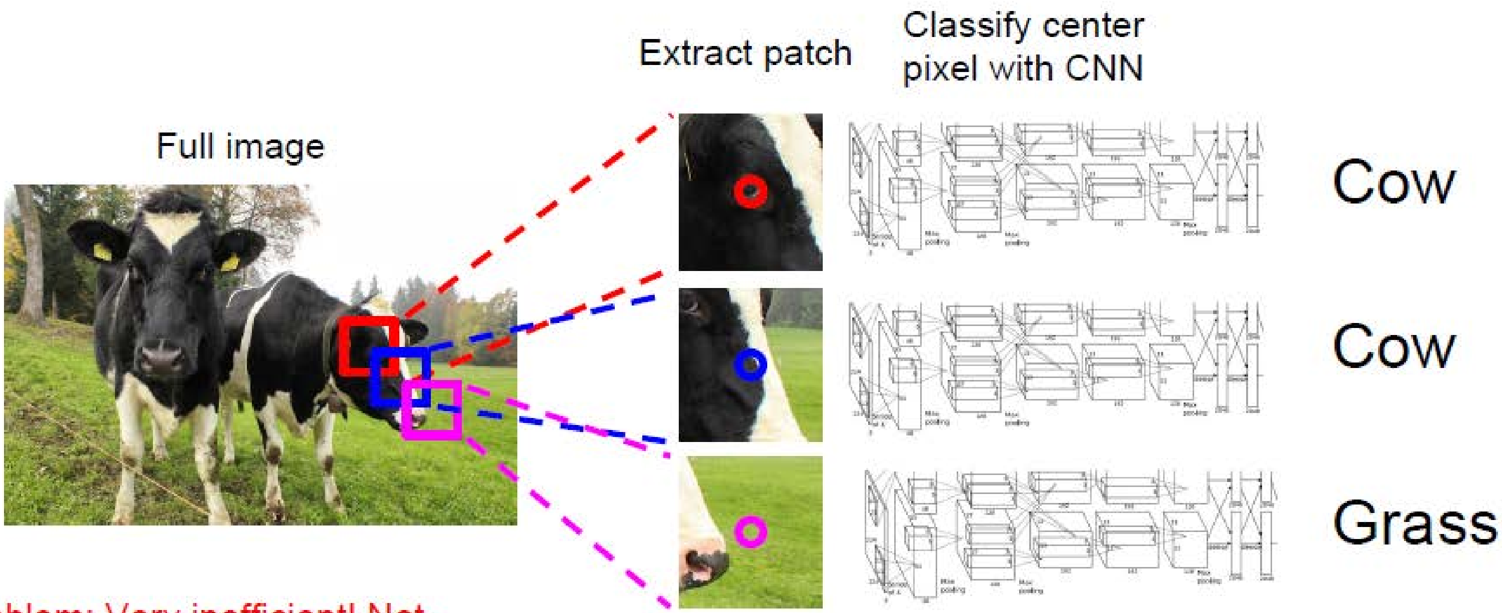
- Goal:
  Group pixels into meaningful or perceptually similar regions

A Practical Segmentation Task

- Semantic Segmentation
  - Supervised learning
  - Assign a class label to each pixel in the input image (i.e., pixel-level classification)
  - Not like instance segmentation, do not differentiate instances; only care about pixel labels

Semantic Segmentation

- Sliding Window

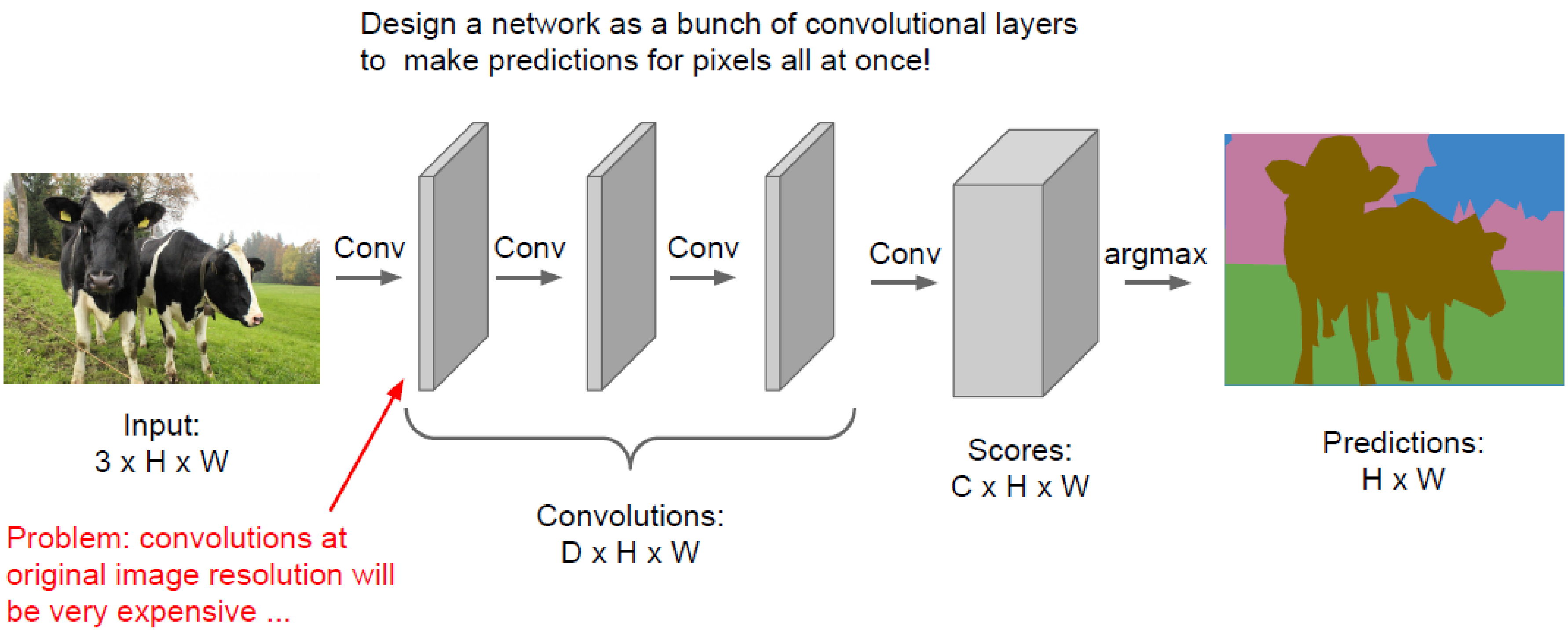


Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

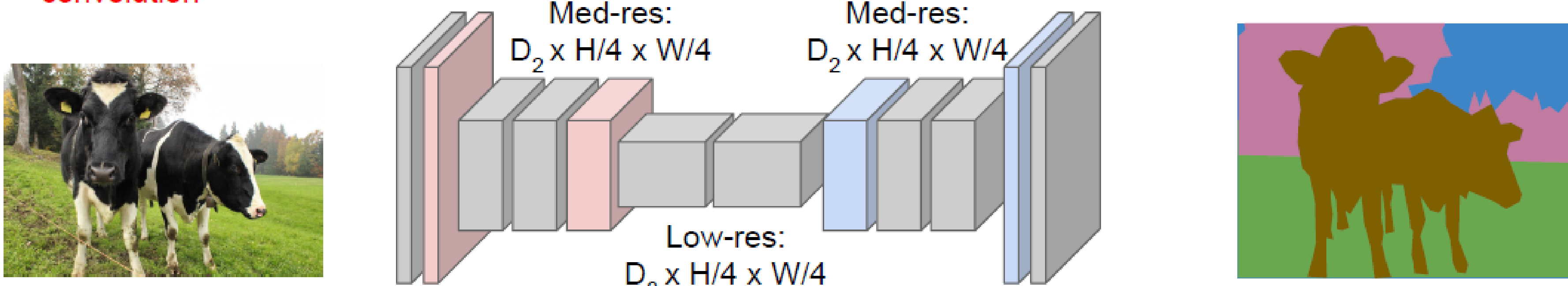
Semantic Segmentation

- Fully Convolutional Nets



Fully Convolutional Networks (FCN)

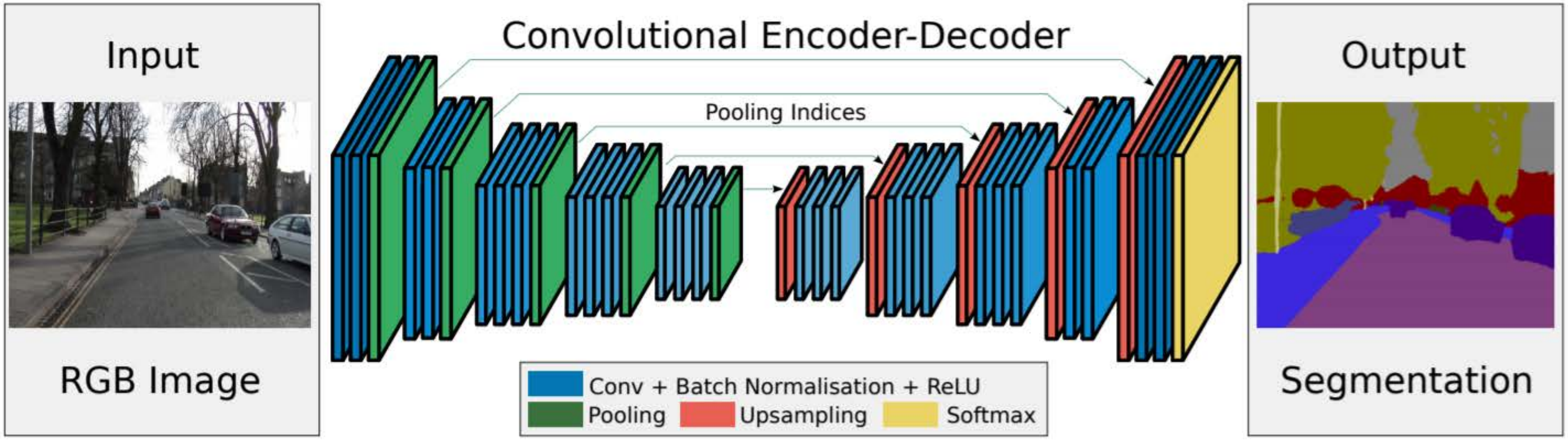
- Remarks
  - All layers are convolutional
  - End-to-end training



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

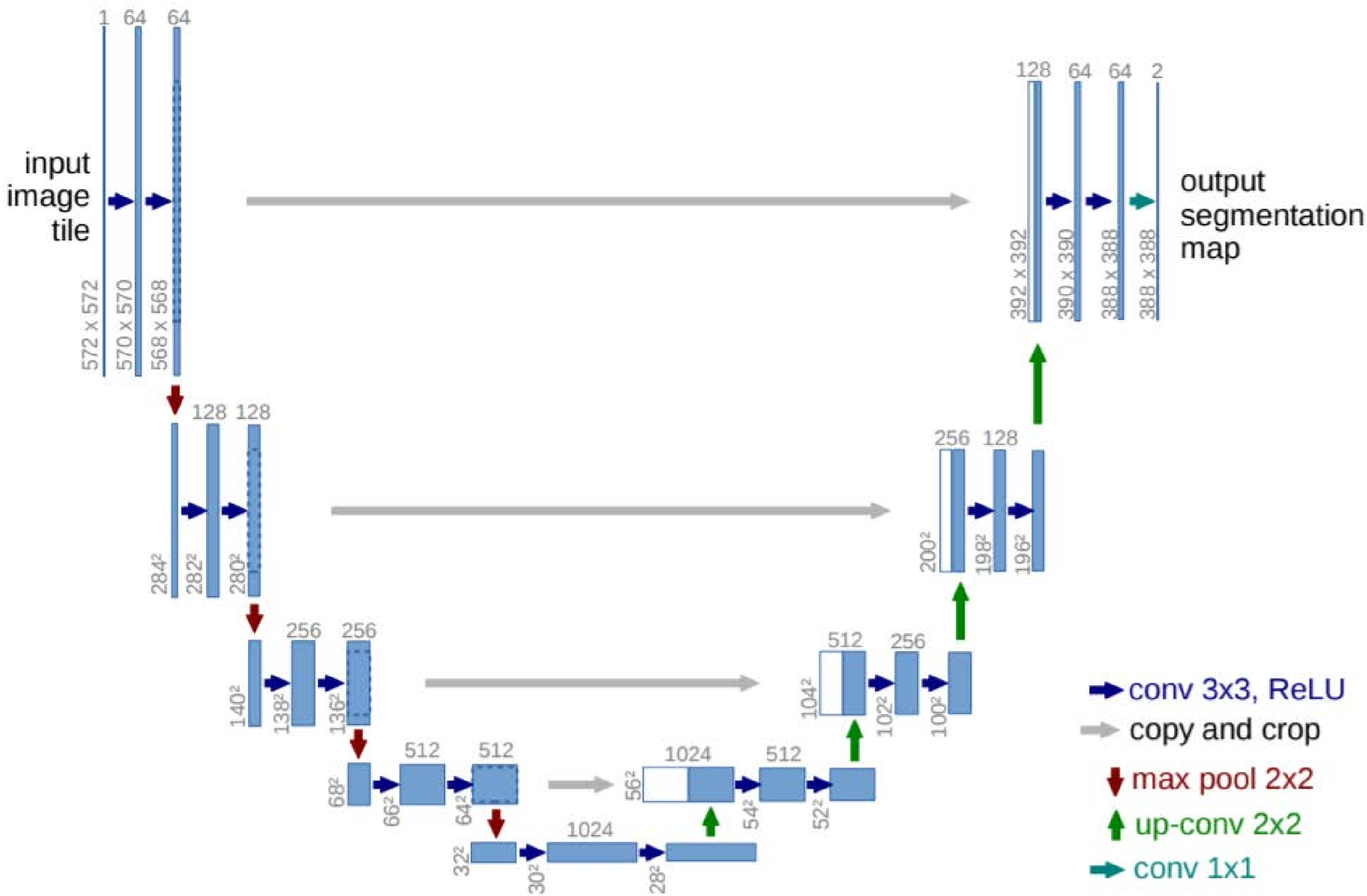
SegNet

- Efficient architecture (memory + computation time)
- Upsampling reusing max-unpooling indices
- Reasonable results without performance boosting addition
- Comparable to FCN



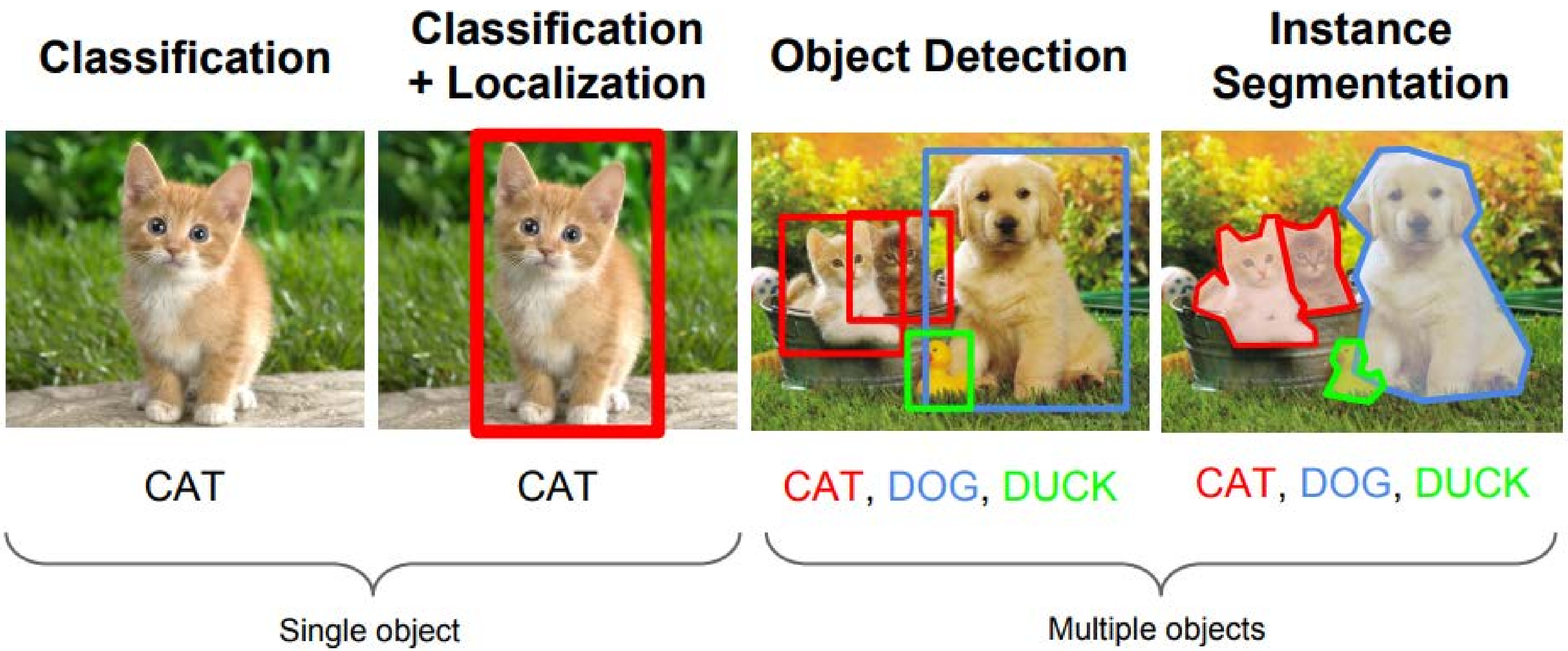
“SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation” [link]

U-Net



U-Net: Convolutional Networks for Biomedical Image Segmentation [link]

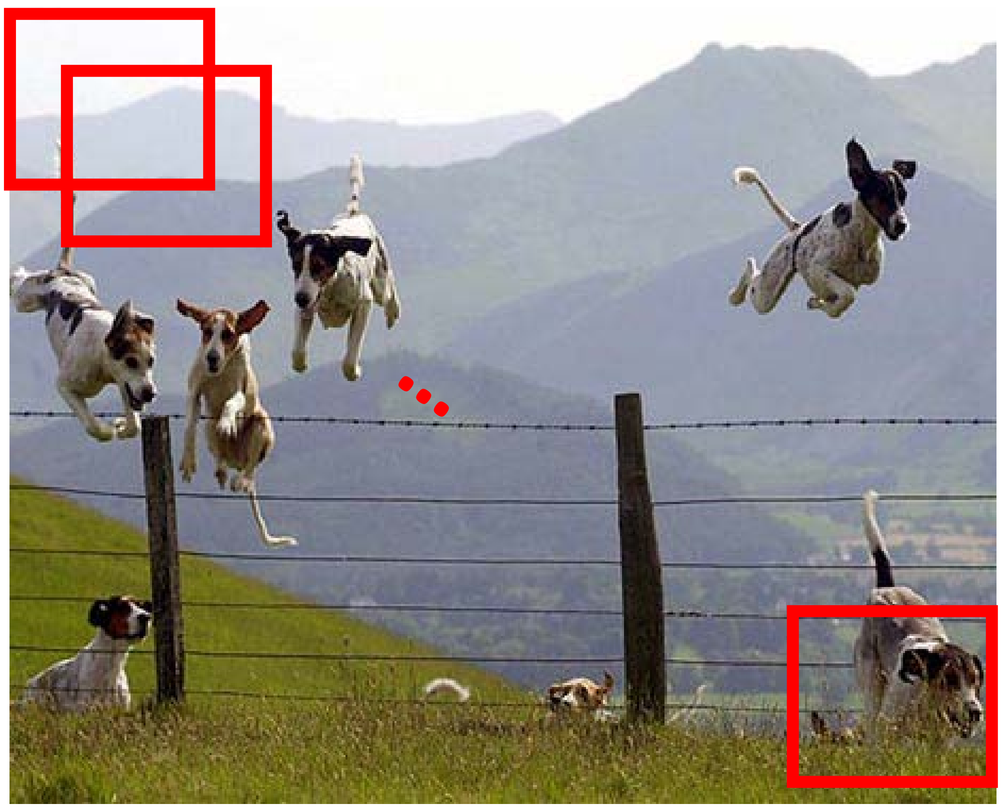
Roadmap

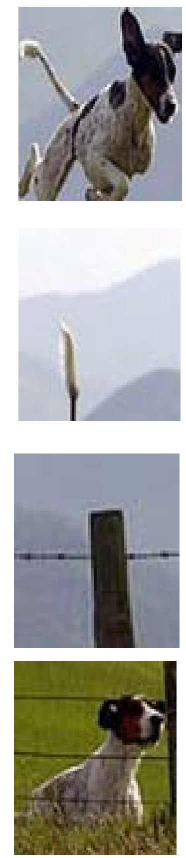


Slide from A. Karpathy

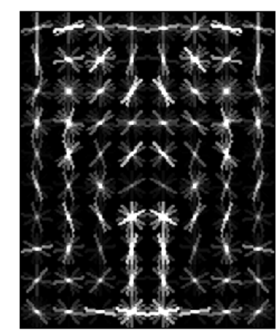
Object Detection

- Focus on object search: “Where is it?”
- Build templates that quickly differentiate object patch from background patch





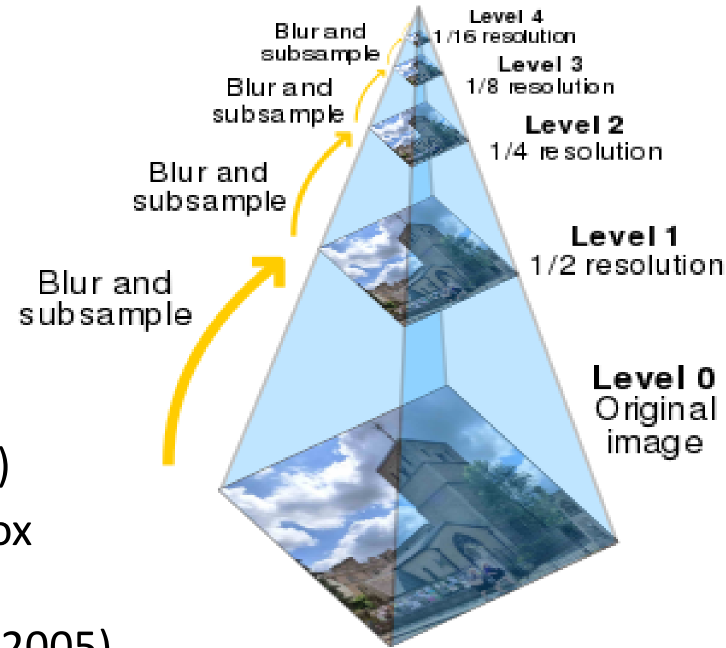
Dog Model



Object or

Non-Object?

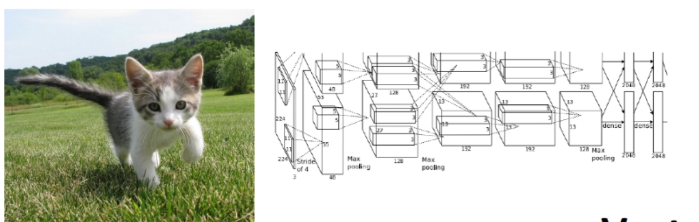
Type of Approaches



- Sliding Windows
  - “Slide” a box around the input image or even across image scales (i.e., image pyramid)
  - Classify each cropped image region inside the box and determine if it’s an object of interest or not
  - E.g., HOG (person) detector by Dalal and Triggs (2005)
    Deformable part-based model by Felzenswalb et al. (2010)
    Real-time (face) detector by Viola and Jones (2001)
- Region (Object) Proposals
  - Generate region (object) proposals
  - Classify each image region and determine it’s an object or not

Type of Approaches (cont'd)

- CNN-based Methods



This image is CC0 public domain

“What”

Correct label: Cat

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Weighted Sum

Loss

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

L2 Loss

Correct box:
(x', y', w', h')

“Where”

Treat localization as a regression problem!

Slide credit: UMich EECS 498-007

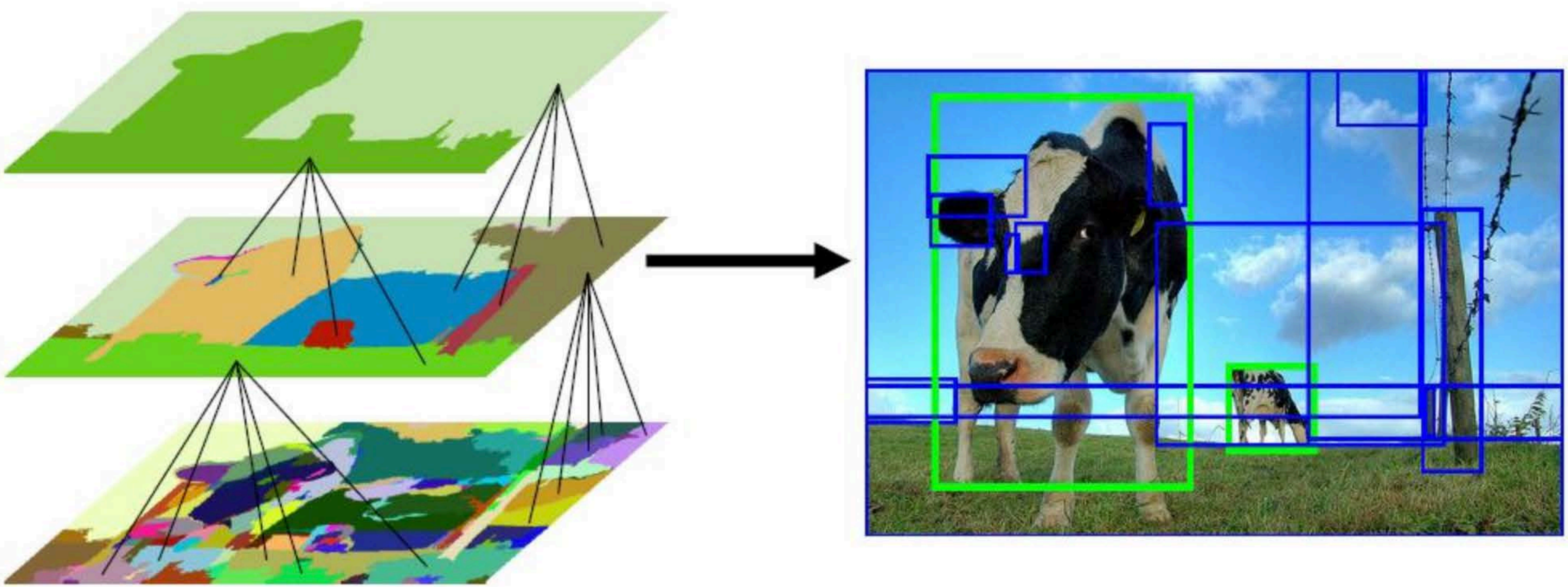
Two-Stage vs. One-Stage Object Detection

Methods

- Sliding Windows
- **Two–stage Frameworks**
  - R–CNN
  - Fast R–CNN
  - Mask R–CNN
  - ⋮
- **One–stage Frameworks**
  - YOLO
  - YOLOv2
  - YOLOv3
  - ⋮

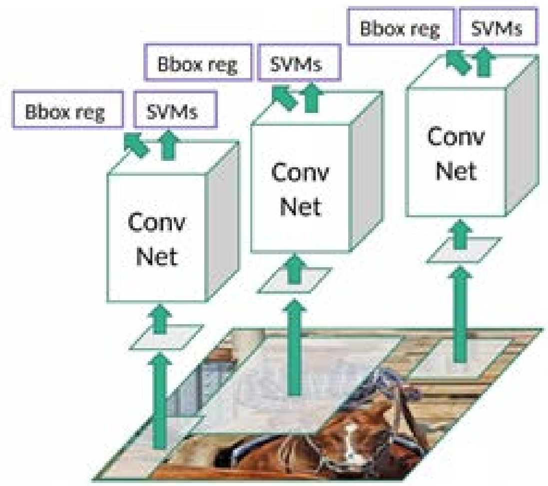
Region Proposal

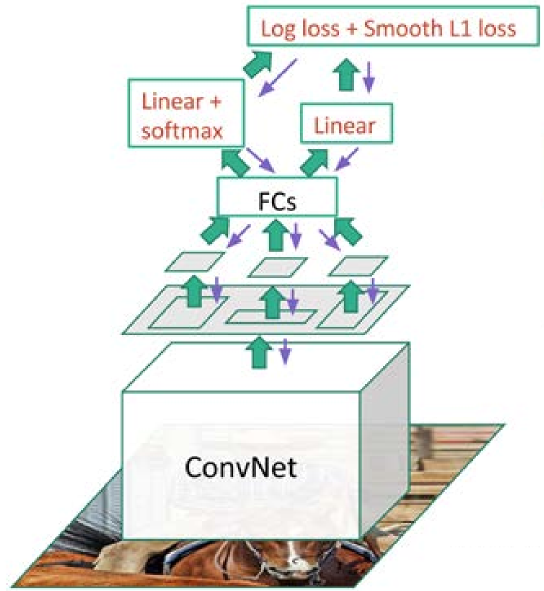
- Solution
  - Use pre-processing algorithms to filter out some regions first, and feed the regions of interest (i.e., region proposals) into CNN
  - E.g., selective search

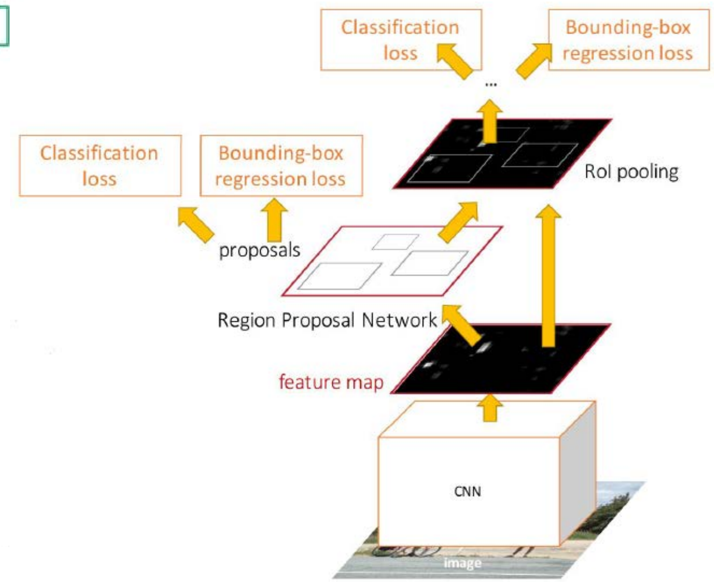


Uijilings et al. IJCV 2013

R-CNN, Fast R-CNN, & Fast*er* R-CNN





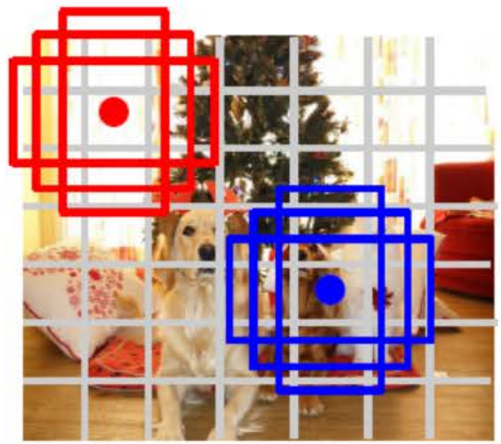


R-CNN (Ren et al. NIPS/NeurIPS 2015), Fast R-CNN (Girshick ICCV 2015), Faster R-CNN (arxiv. 2016)

One-Stage Object Detection: Detection without Proposals

Go from input image to tensor of scores with one big convolutional network!

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:

- Regress from each of the B base boxes to a final box with 5 numbers:
(dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)

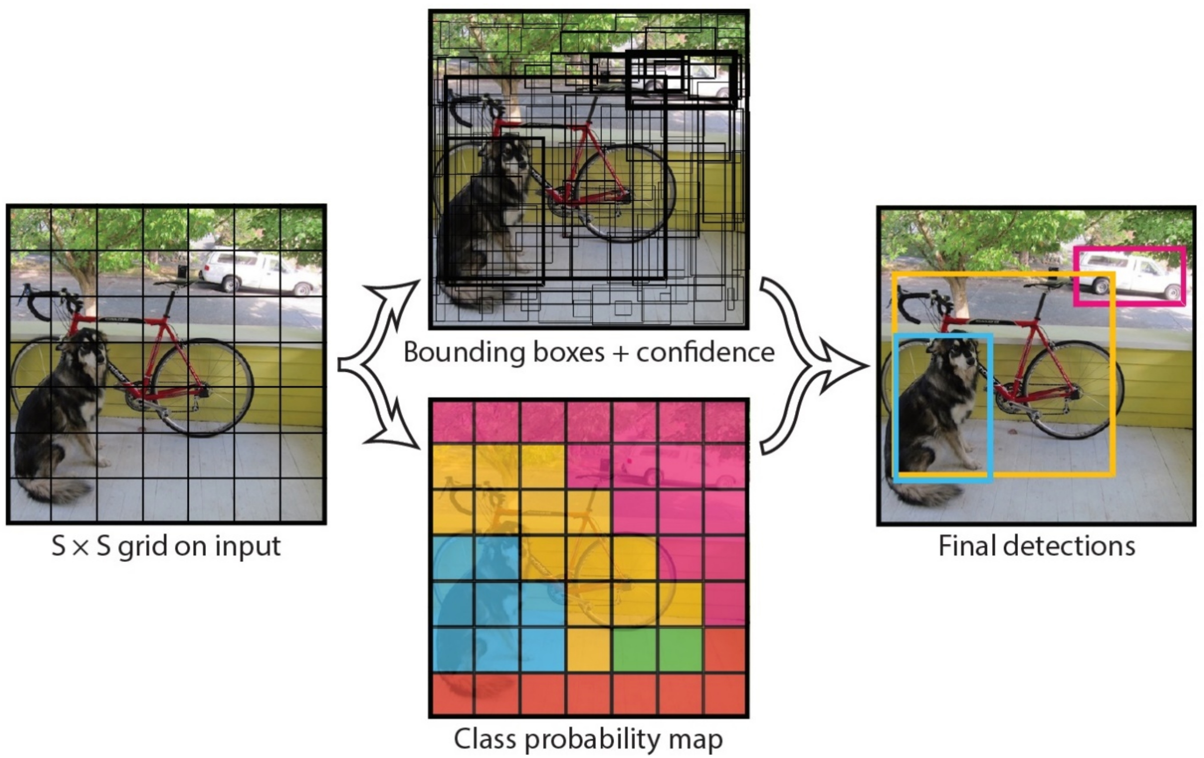
Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

You Only Look Once (YOLO)

Divide the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities.

These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.



What'd Be Covered in This Crash Course…

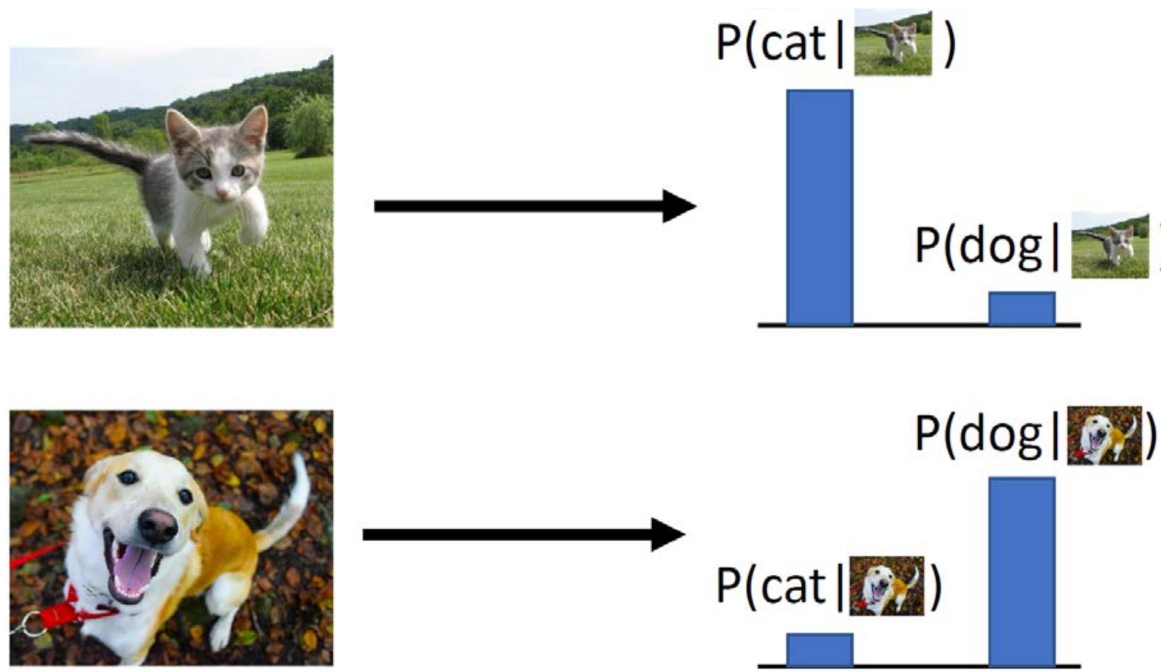
- **Learning-based Computer Vision**
  - From Linear to Non-Linear Classifiers
- **Start of Deep Learning for Computer Vision**
  - Convolutional Neural Networks
  - Self-Supervised Learning
  - Segmentation & Detection
- **Generative Models**
  - Autoencoder, Variational Autoencoder,
    Generative Adversarial Networks & Diffusion Models
- **Sequence-to-Sequence Learning**
  - Attention is All You Need: Transformer
- **Vision & Language Foundation Models**
  - Image-to-Text vs. Text-to-Image
  - Parameter-Efficient Fine-tuning

Discriminative vs. Generative Models (cont'd)

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

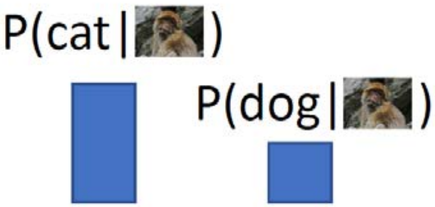
Conditional Generative Model: Learn p(x|y)



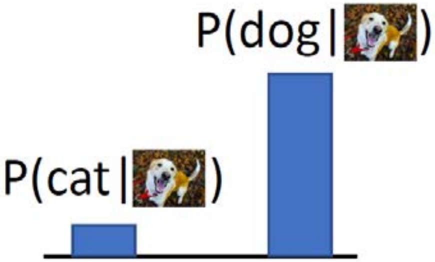
Discriminative model: the possible labels for each input "compete" for probability mass. But no competition between **images**

Discriminative vs. Generative Models (cont'd)

Discriminative Model:
Learn a probability distribution p(y|x)



Generative Model:
Learn a probability distribution p(x)



Conditional Generative Model: Learn p(x|y)

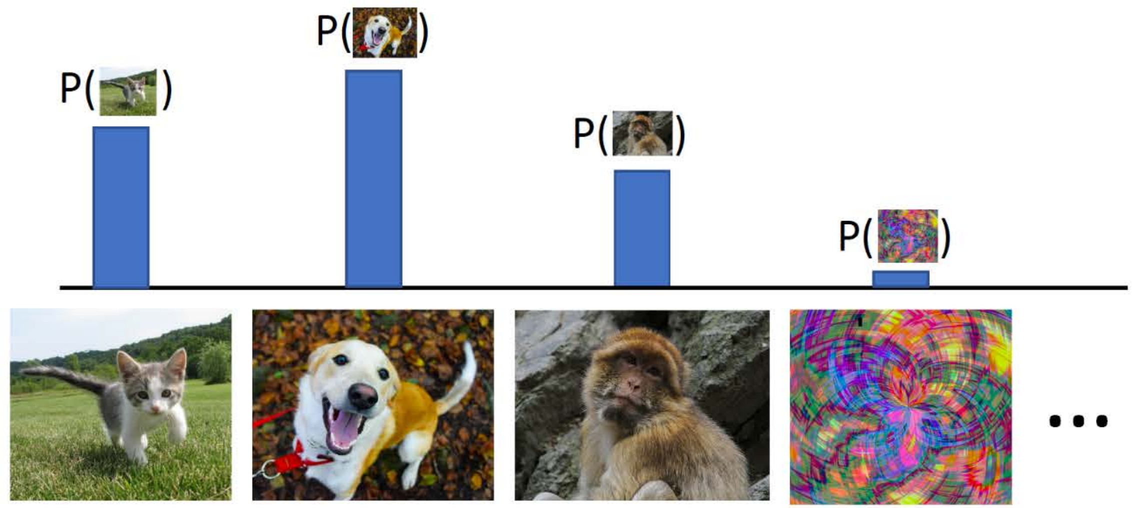
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Discriminative vs. Generative Models (cont’d)

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Generative model: All possible images compete with each other for probability mass

Model can “reject” unreasonable inputs by assigning them small values

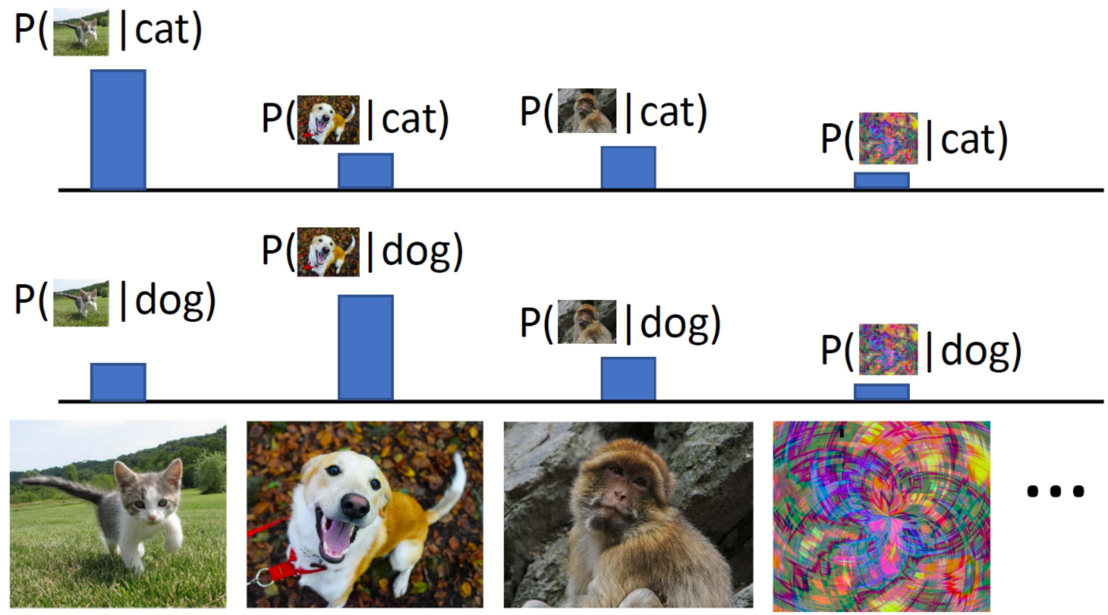
Slide credit: UMich EECS 498-007

Discriminative vs. Generative Models (cont'd)

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

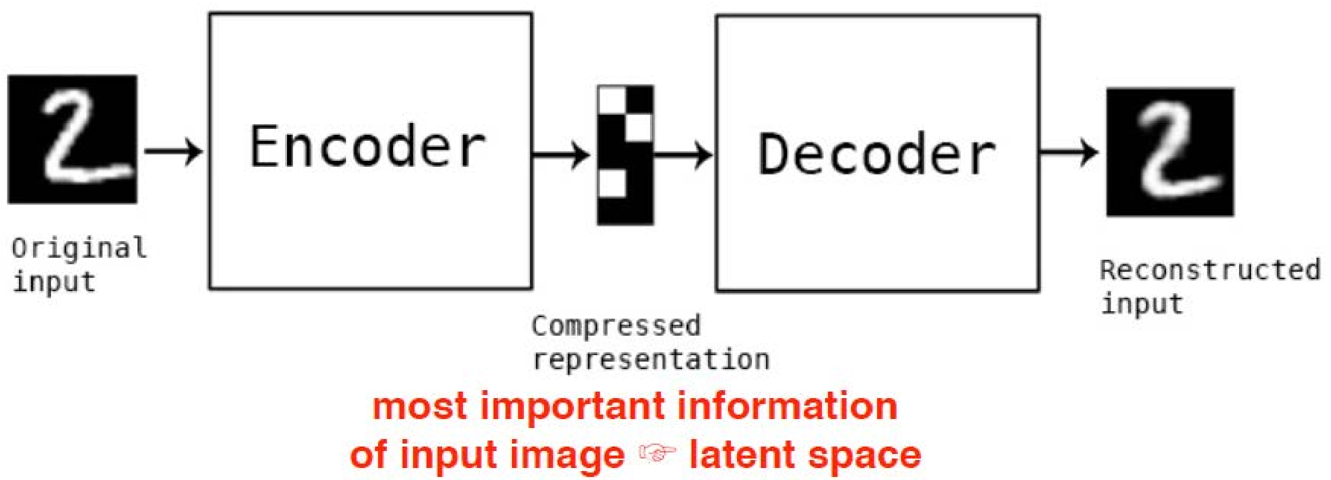
Conditional Generative Model: Learn p(x|y)



Conditional Generative Model: Each possible label induces a competition among all images

GenAI? Let's Start from Autoencoder

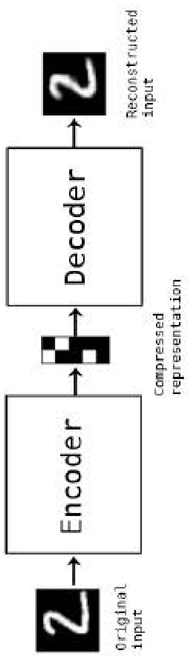
- Autoencoder
  - Autoencoding = encoding itself with recovery purposes
  - In other words, encode/decode data with reconstruction guarantees
  - Latent variables/features as deep representations
  - Example objective/loss function at output:
    - L2 norm between input and output, i.e.,

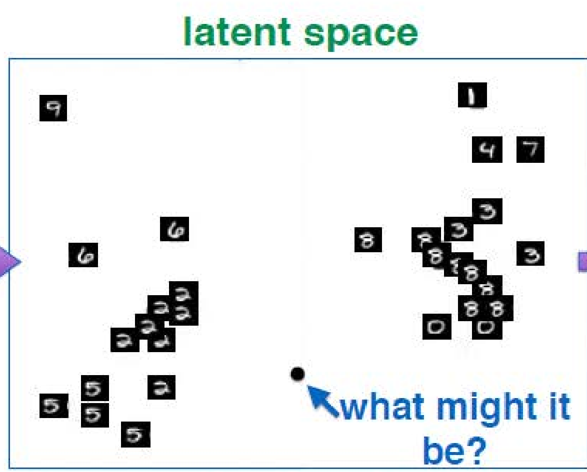


Slide credit: W. Chiu

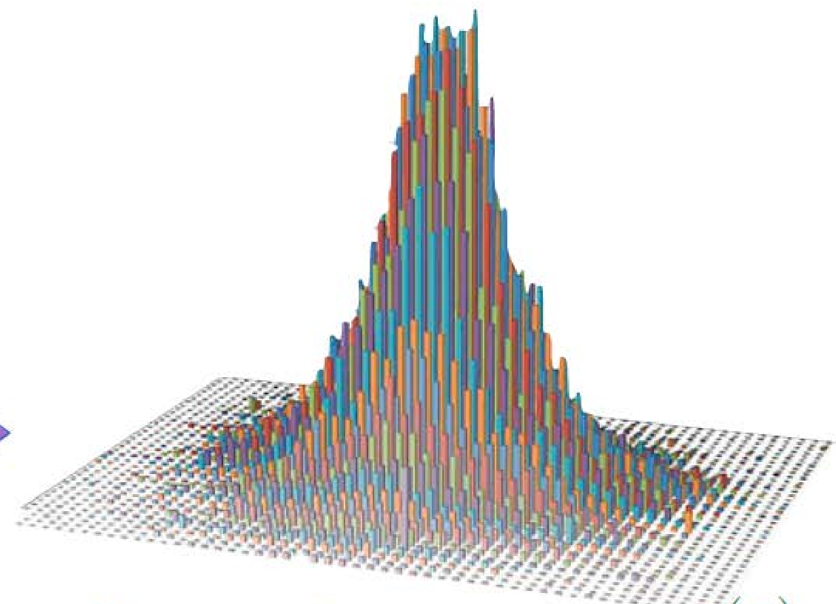
Take a Deep Look to Discover Latent Variables/Representations (cont'd)

- What's the Limitation of Autoencoder?





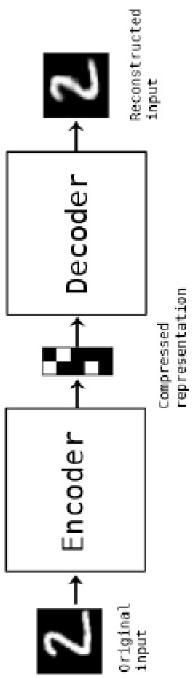
Only samples, how about other regions not covered?



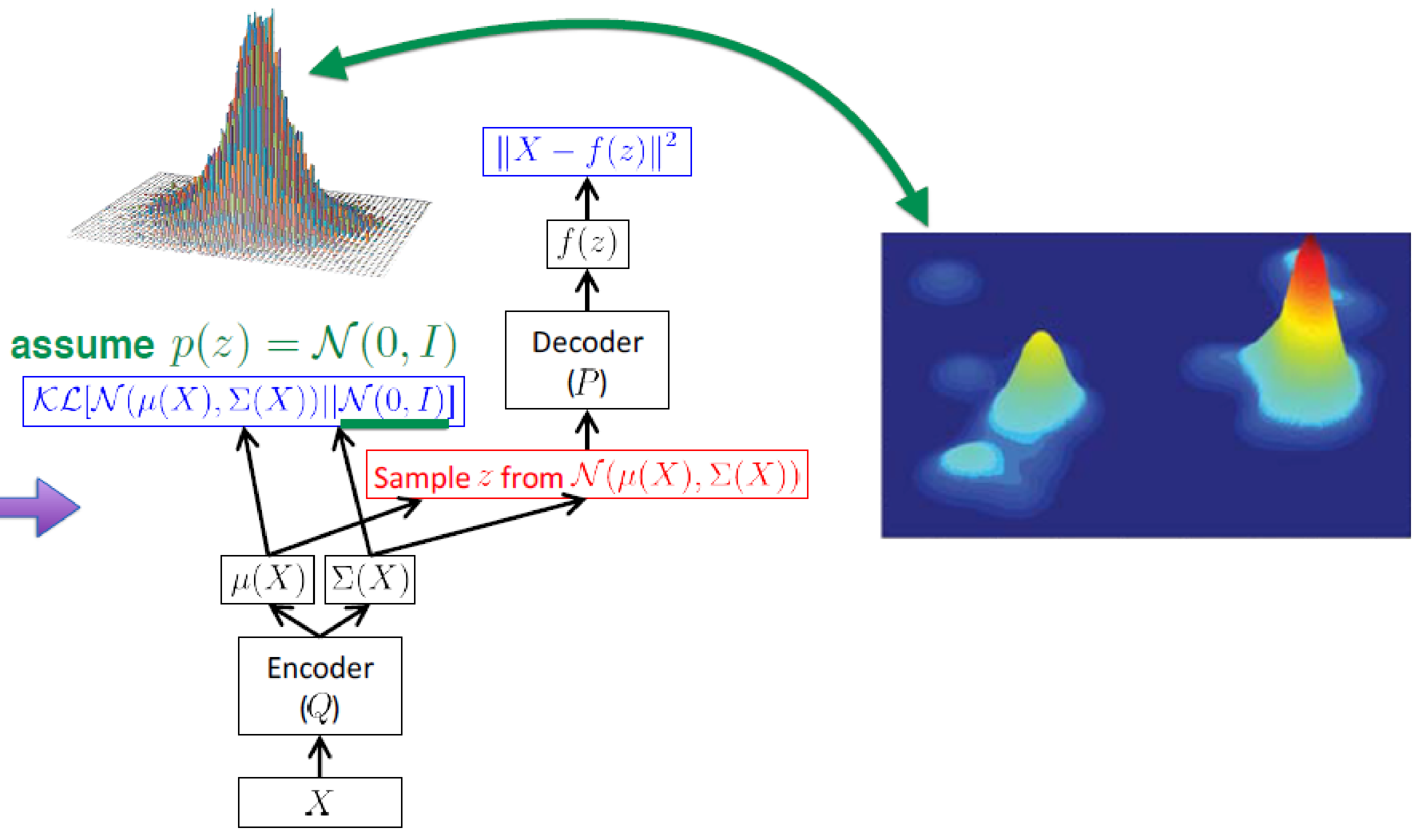
We wanna to have *"distribution"* $p(z)$ where we can sample from any location

Slide credit: W. Chiu

From Autoencoder to Variational Autoencoder



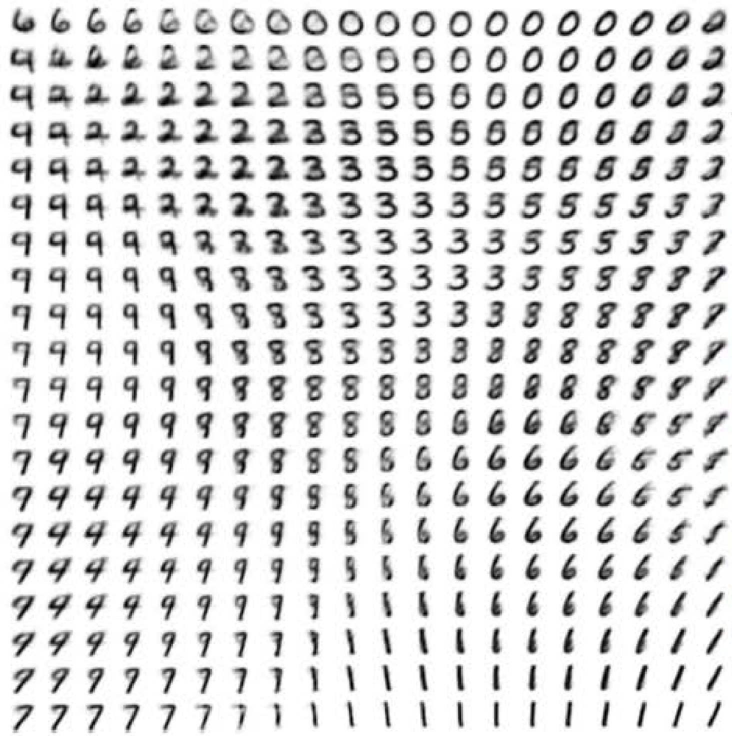
Now is a "*distribution*", we can assume it to be
a distribution easy to sample from, e.g. Gaussian



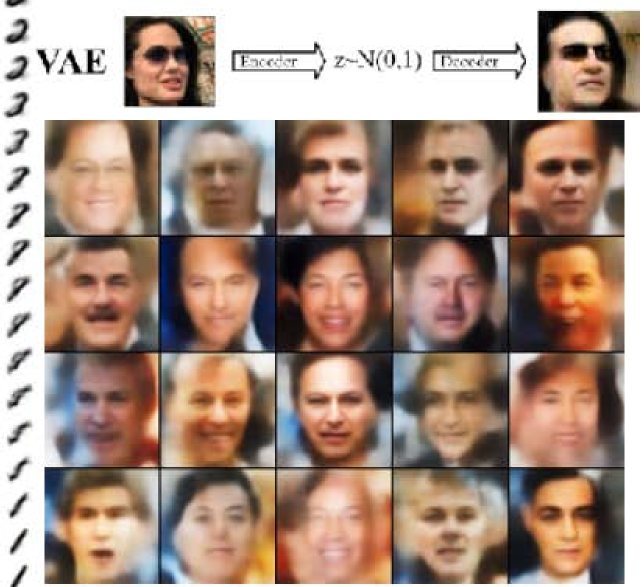
From Autoencoder to Variational Autoencoder (cont'd)

- Example Results

(a) Learned Frey Face manifold



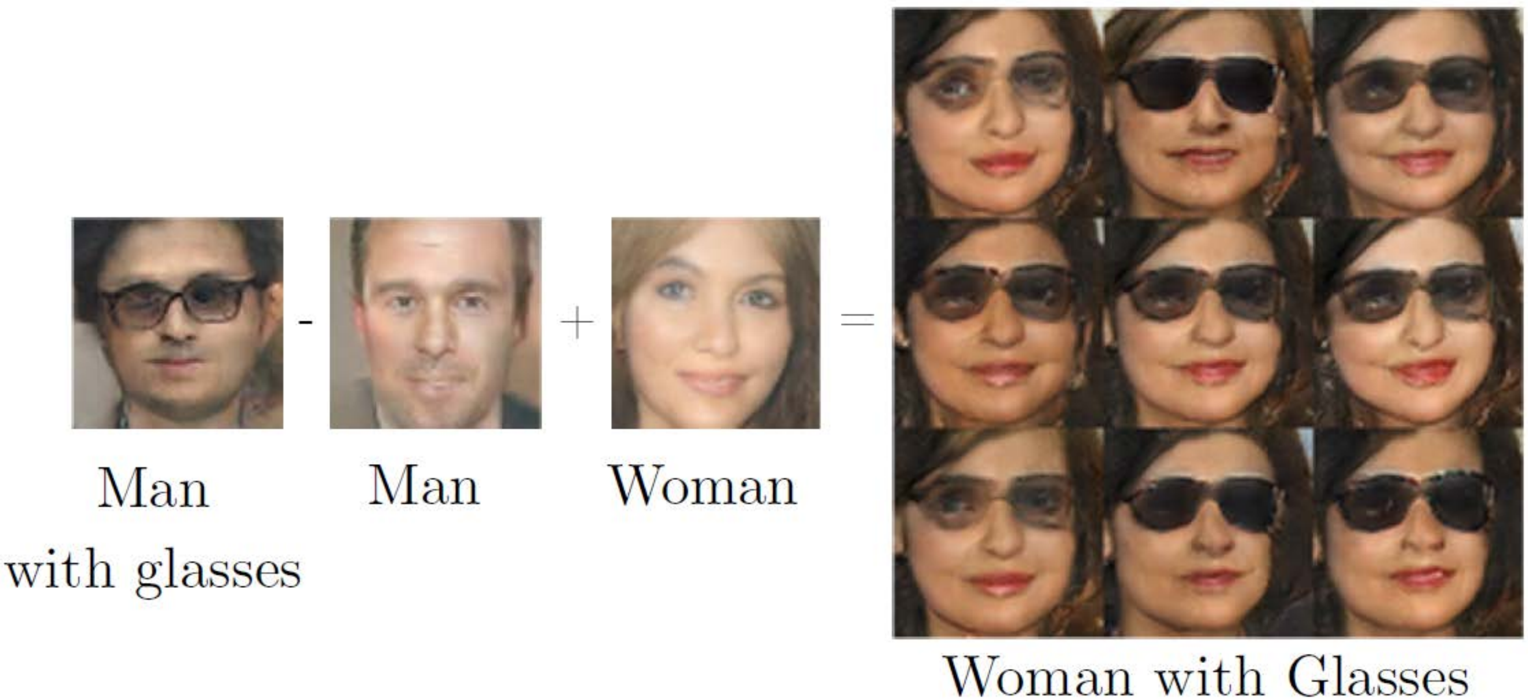
(b) Learned MNIST manifold



Kingma et al., 2013

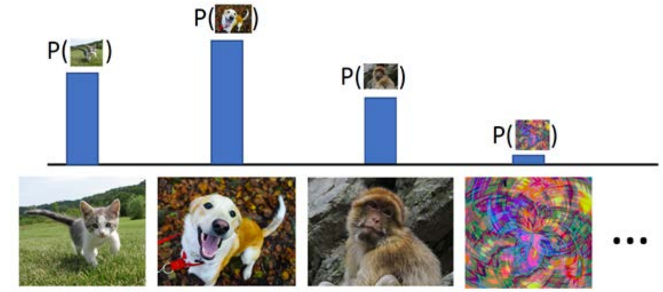
From Autoencoder to Variational Autoencoder (cont'd)

- Example Results
  - A' – A + B = B'

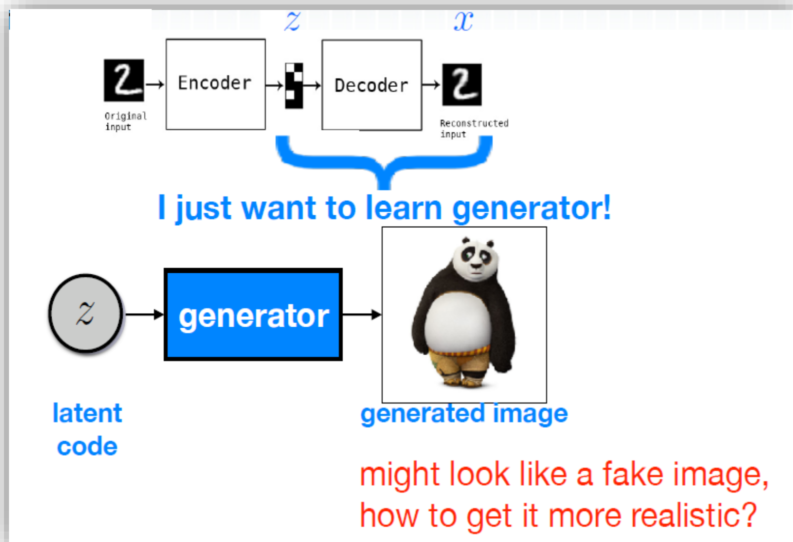


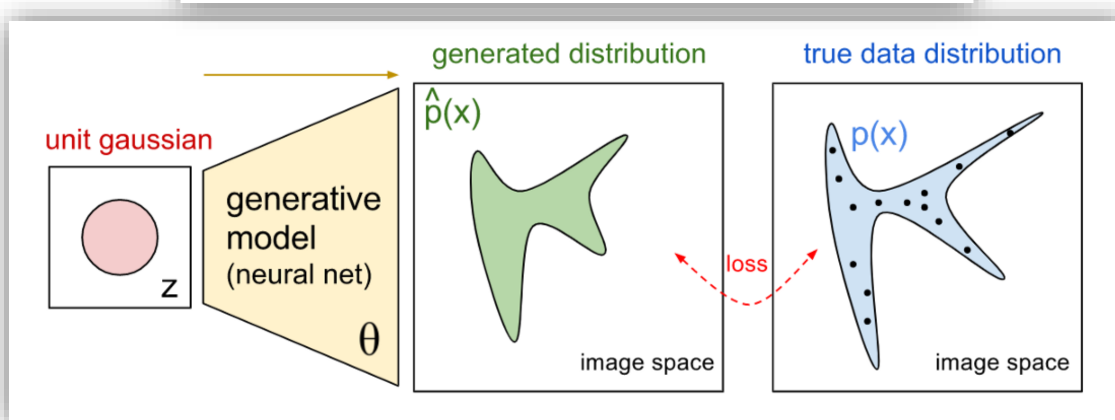
Radford et al., 2015

Limitation of VAE?



- Remarks
  - Why Gaussian distribution is sufficient?
  - What if we only need the decoder/generator in practice?
  - How do we know if the output images are sufficiently good?





Slide credit: W. Chiu

Generative Adversarial Network

- Idea
  - **Generator** to convert a vector z (sampled from P_z) into fake data x (from P_G), while we need $P_G = P_{data}$
  - **Discriminator** classifies data as real or fake (1/0)
  - How? Impose an **adversarial loss** on the observed data distribution!

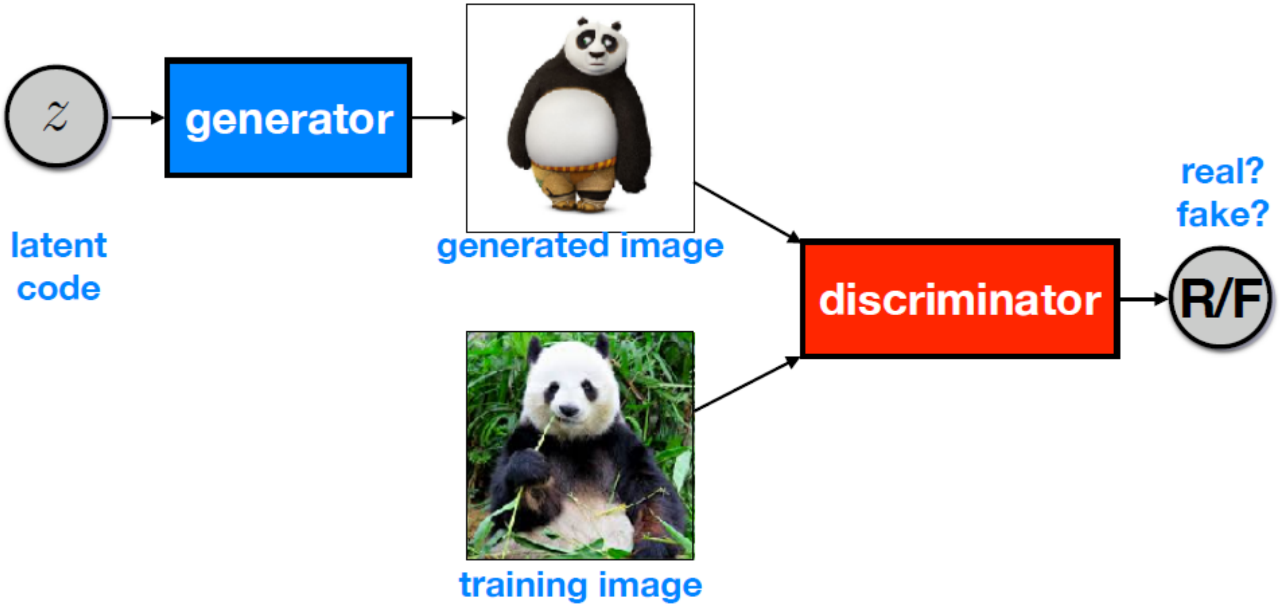
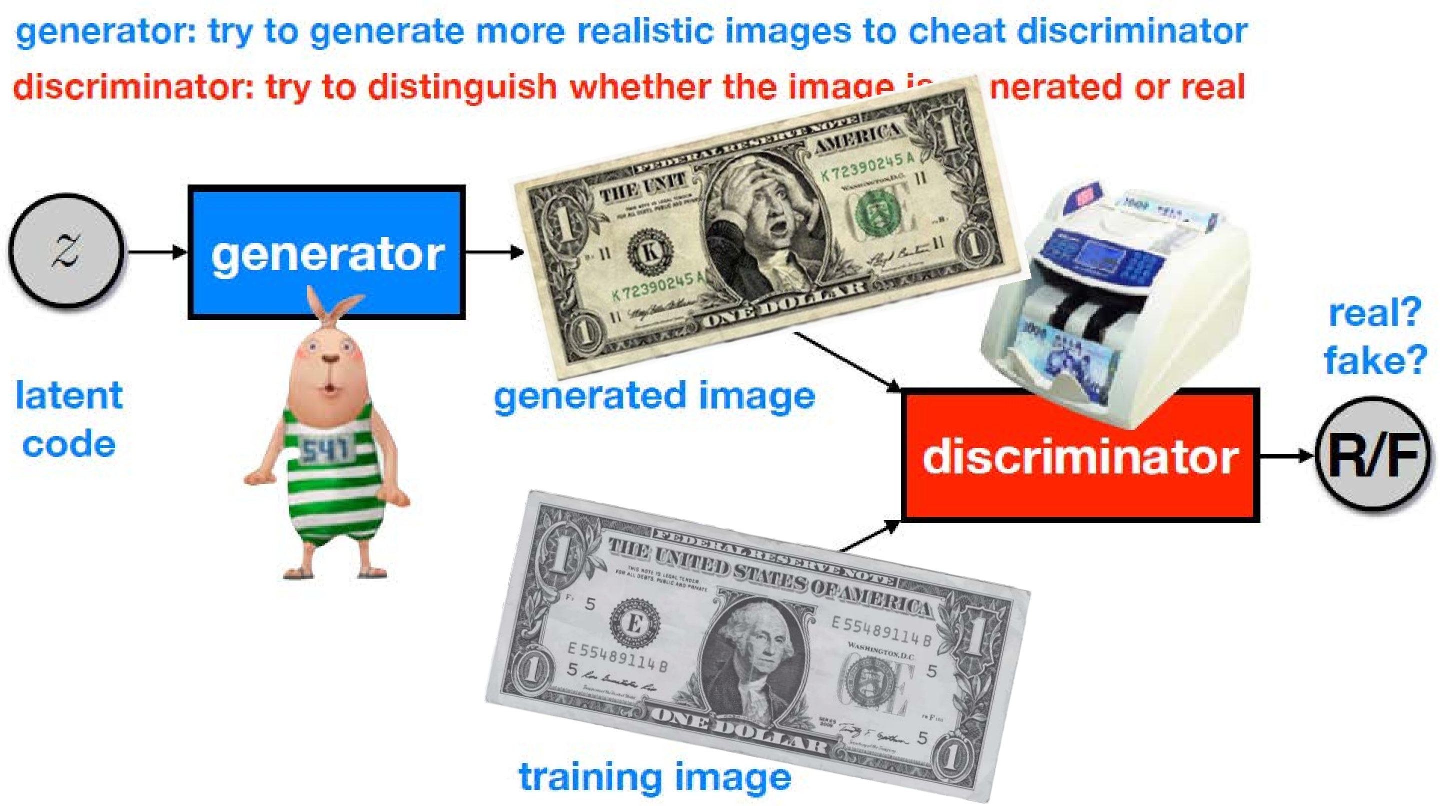


Image credit: W. Chiu

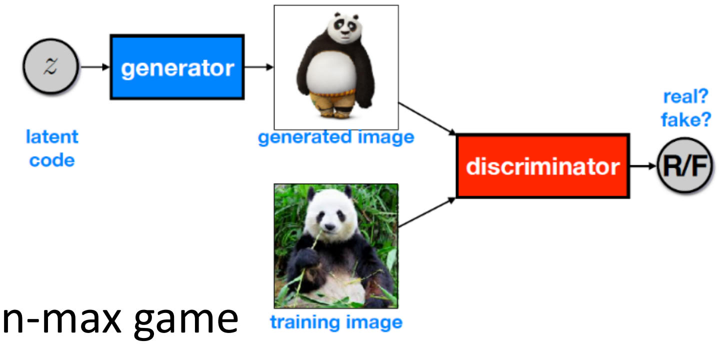
Generative Adversarial Network (cont'd)

- Key idea:
  - Impose *adversarial loss* on data distribution
  - Let's see a practical example…



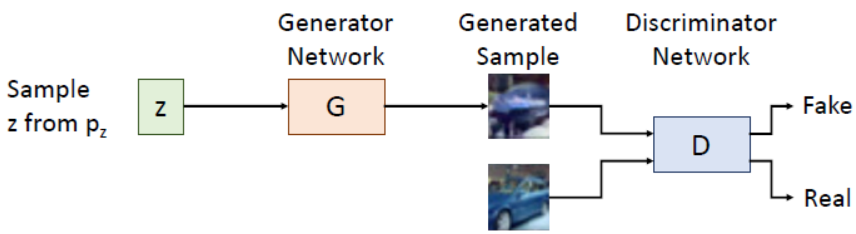
Slide credit: W. Chiu

Training Objective of GAN



- Jointly train generator G and discriminator D with a min-max game

$$\min_{G} \max_{D} \left(\overbrace{E_{x \sim p_{data}} [\log D(x)]}^{\text{Discriminator wants D(x) = 1 for real data}} + E_{z \sim p(z)} \overbrace{\underbrace{\left[\log \left(1 - D(G(z)) \right) \right]}_{\text{Generator wants D(x) = 1 for fake data}}}^{\text{Discriminator wants D(x) = 0 for fake data}} \right)$$



- Train G & D with alternating gradient updates

$$\min_{G} \max_{D} V(G, D)$$

For t in 1, … T:

1. (Update D) $D = D + \alpha_D \frac{\partial V}{\partial D}$
2. (Update G) $G = G - \alpha_G \frac{\partial V}{\partial G}$

Slide credit: I. Goodfellow

Denoising Diffusion Models

- Recently emerging as powerful **visual** generative models
  - Unconditional image synthesis
  - Conditional image synthesis
  - Outperforms GANs

DALL·E 2

"a teddy bear on a skateboard in times square"

Diffusion Models Beat GANs on Image Synthesis, Dhariwai & Nochol, OpenAI, 2021

Imagen

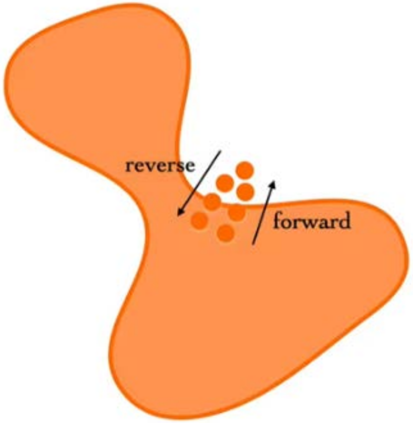
A group of teddy bears in suit in a corporate office celebrating the birthday of their friend. There is a pizza cake on the desk.

Cascaded Diffusion Models for High Fidelity Image Generation, Ho et al., Google, 2021

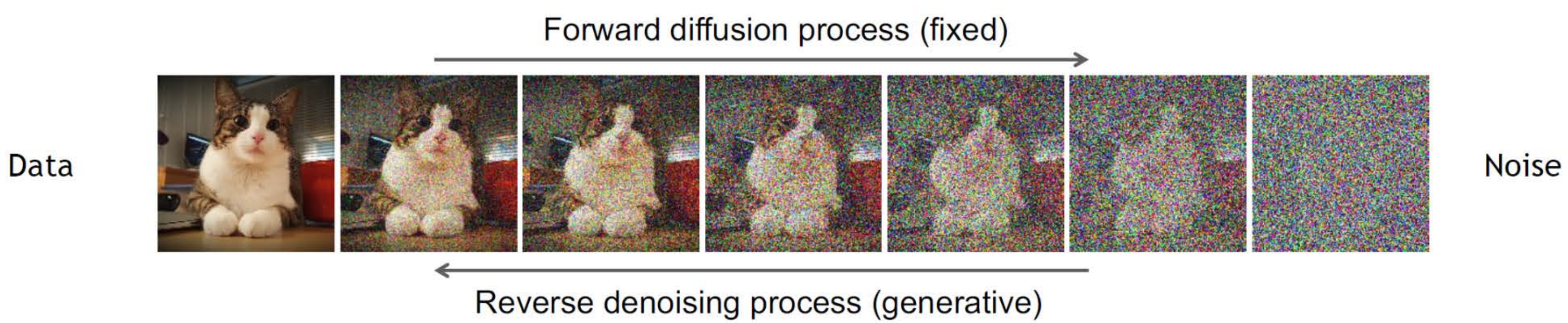
Slide credit: Kreis, Gao, & Vahdat

Denoising Diffusion Probabilistic Models (DDPM)

Learning to generate by denoising



- 2 processes required for training:
  - Forward diffusion process – gradually add noise to input
  - **Reverse diffusion process**
    - learns to generate/restore data by denoising
    - typically implemented via a **conditional U-net**)
  - Comments about noise scheduling (see next slide)



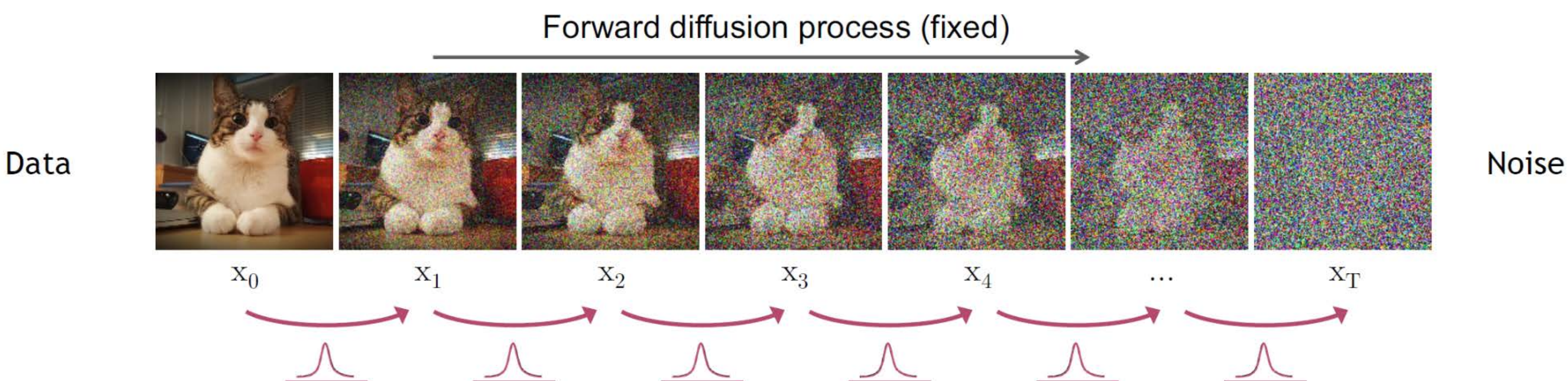
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

Slide credit: Kreis, Gao, & Vahdat

DDPM:

Learning to generate by denoising (cont'd)

- Forward diffusion process
  - Gradually add noise to the input in T steps (e.g., via linear scheduling)
  - Recall that x_0 denotes clean input image, and x_T is the final noisy one.
  - Comments on $q(x_t|x_{t-1})$



Normal distribution — mean — var

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \Rightarrow q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$ (joint)

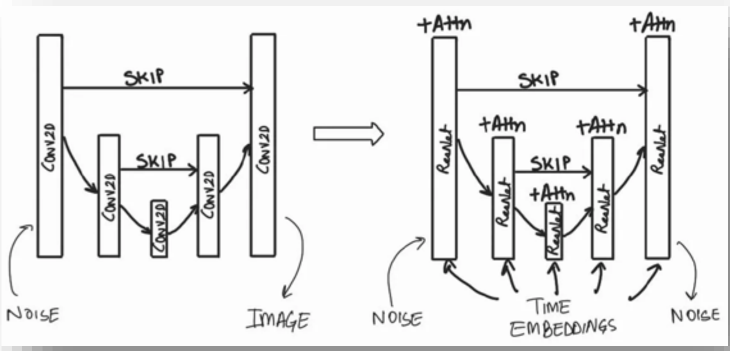
output

variance schedule (hyperparameter)

Slide credit: Kreis, Gao, & Vahdat

Learning of Diffusion Models

- Summary
  - Training and sample generation



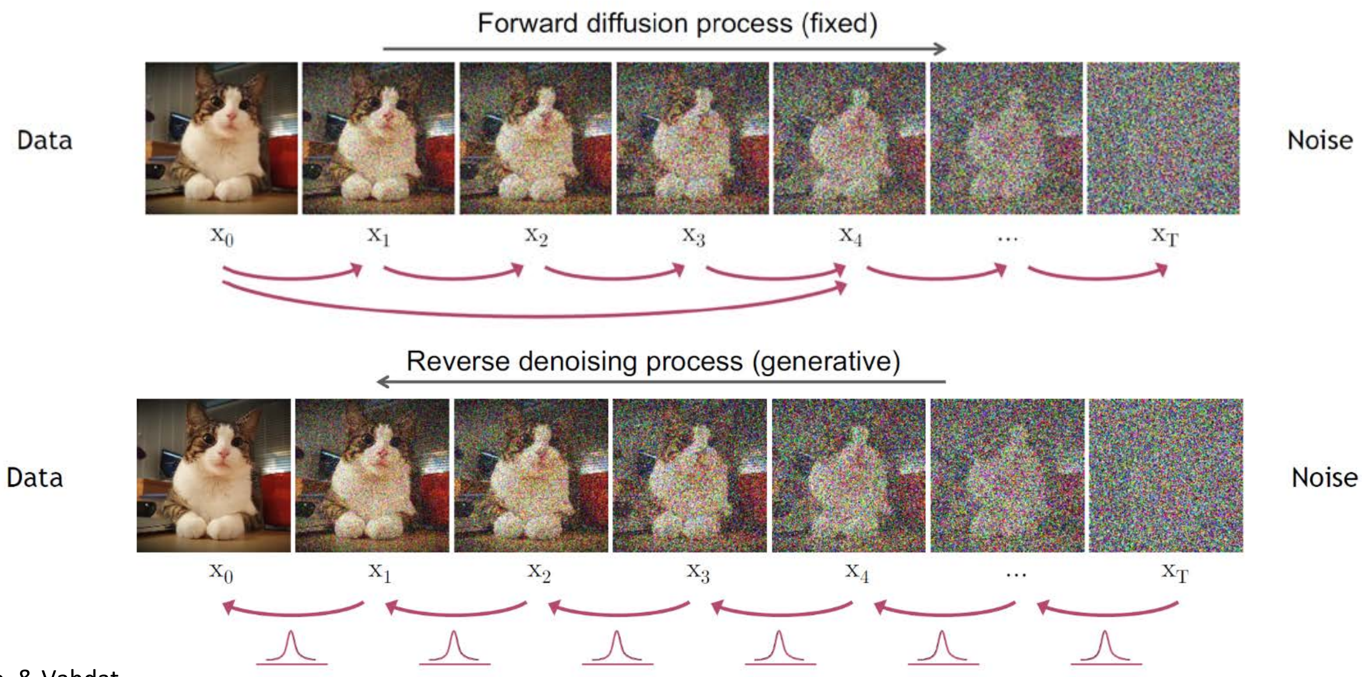
https://medium.com/@vedantjumle/class-conditioned-diffusion-models-using-keras-and-tensorflow-9997fa6d958c

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\boxed{\sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\boldsymbol{\epsilon}}, t\right)\right\|^2$$
6: **until** converged

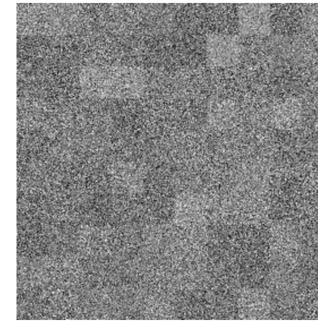
Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
4: $\mathbf{x}_{t-1} = \boxed{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)} + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$



Slide credit: Kreis, Gao, & Vahdat

Learning of Diffusion Models



E.g., MNIST

- Summary
  - Training and sample generation

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boxed{\sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\boldsymbol{\epsilon}}, t) \right\|^2$$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_{t-1} = \boxed{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)} + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

More steps

Slide credit: Kreis, Gao, & Vahdat

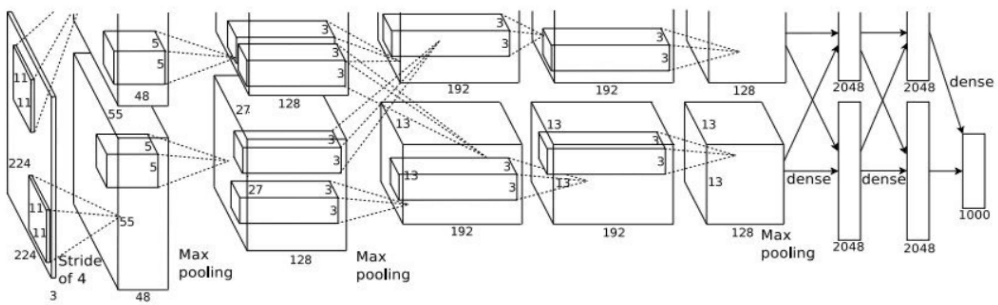
https://medium.com/ai-blog-tw/%E9%82%8A%E5%AF%A6%E4%BD%9C%E9%82%8A%E5%AD%B8%E7%BF%92diffusion-model-%E5%BE%9Eddpm%E7%9A%84%E7%B0%A1%E5%8C%96%E6%A6%82%E5%BF%B5%E7%90%86%E8%A7%A3-4c565a1c09c

What'd Be Covered in This Crash Course…

- **Learning-based Computer Vision**
  - From Linear to Non-Linear Classifiers
- **Start of Deep Learning for Computer Vision**
  - Convolutional Neural Networks
  - Self-Supervised Learning
  - Segmentation & Detection
- **Generative Models**
  - Autoencoder, Variational Autoencoder,
    Generative Adversarial Networks & Diffusion Models
- **Sequence-to-Sequence Learning**
  - Attention is All You Need: Transformer
- **Vision & Language Foundation Models**
  - Image-to-Text vs. Text-to-Image
  - Parameter-Efficient Fine-tuning

What Are The Limitations of CNN?

- Deal with image data
    - Both input and output are images/vectors
- Simply feed-forward processing



More Applications in Vision

Image Captioning

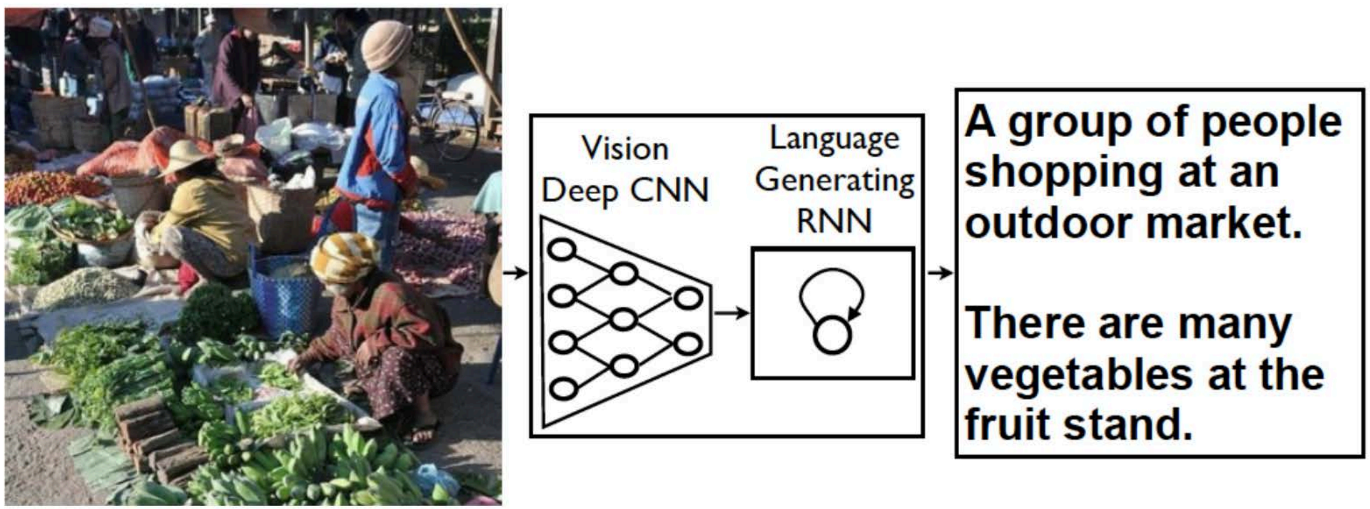
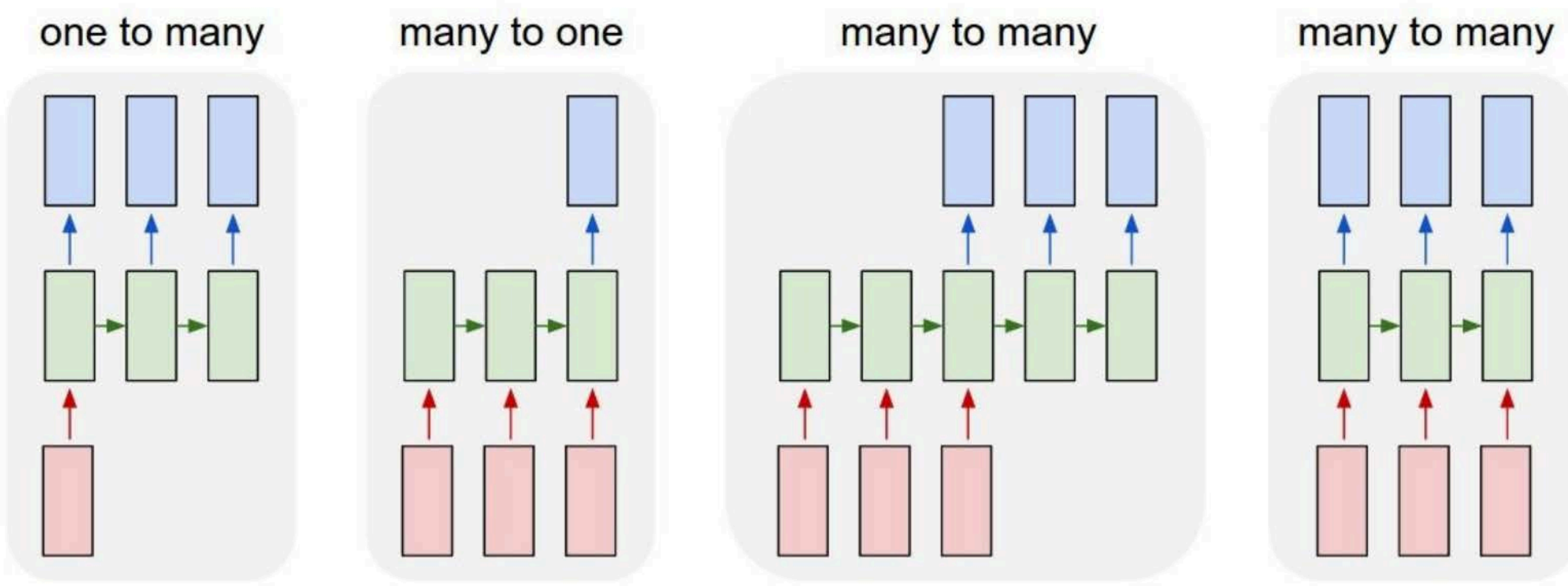


Figure from Vinyals et al, “Show and tell: A neural image caption generator”, CVPR 2015

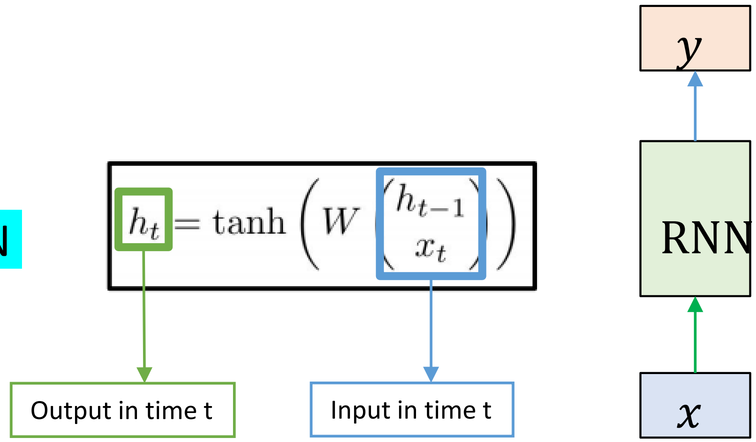
Sequence-to-Sequence Modeling

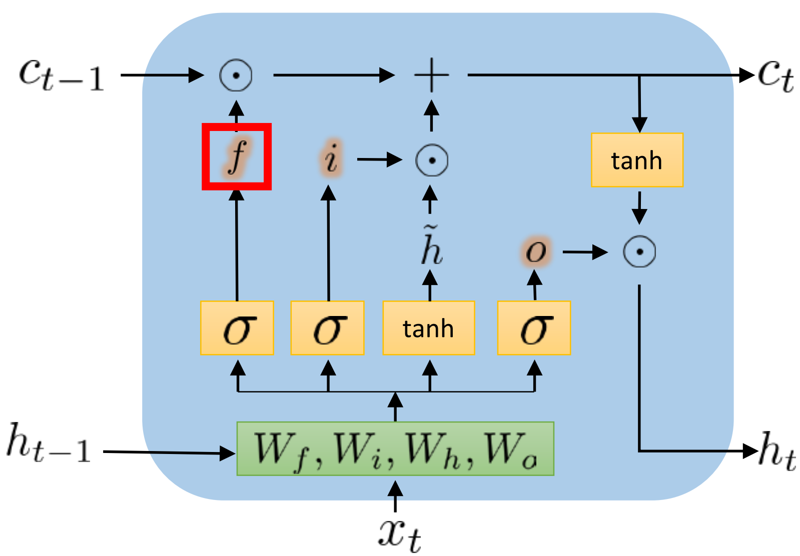


e.g., image caption e.g., action recognition e.g., video prediction e.g., video indexing

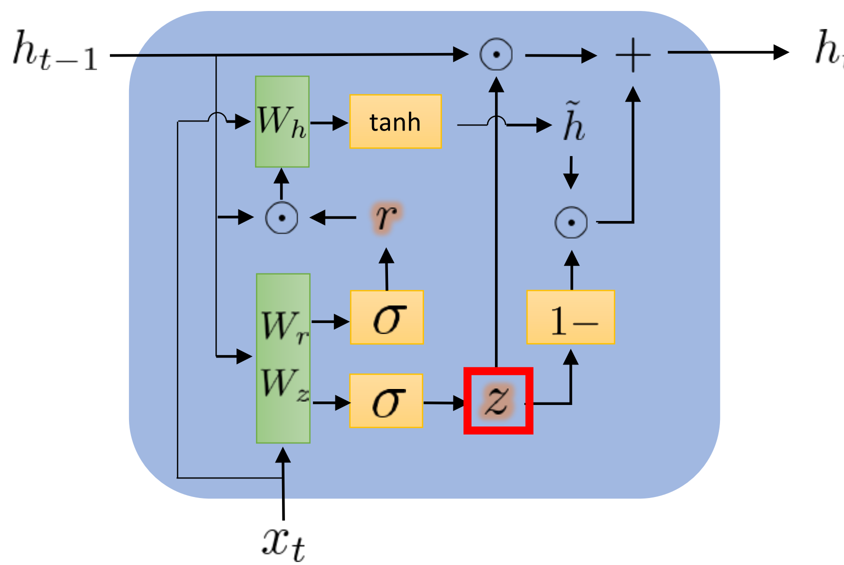
Vanilla RNN, LSTM, & GRU

RNN





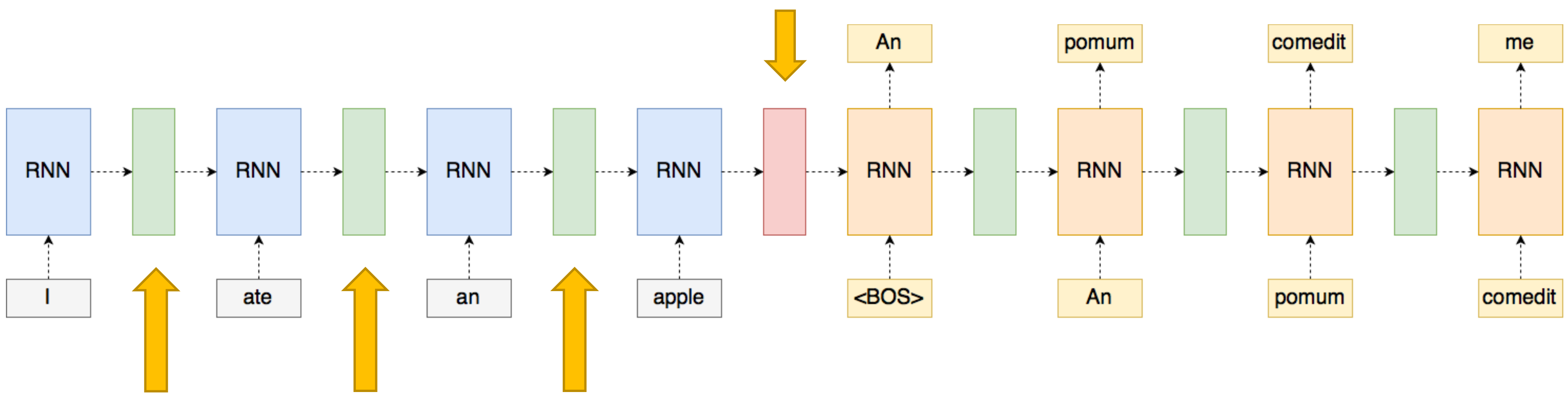
LSTM



GRU

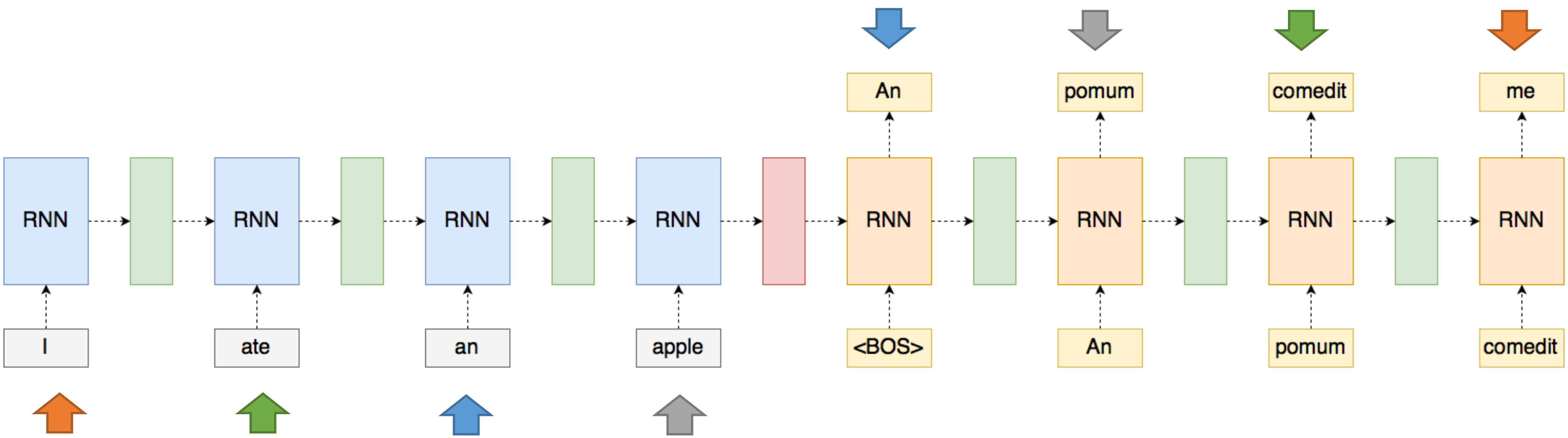
What’s the Potential Problem in RNN?

- Each hidden state vector extracts/carries information across time steps (some might be diluted downstream).
- Information of the entire input sequence is embedded into a single hidden state vector.



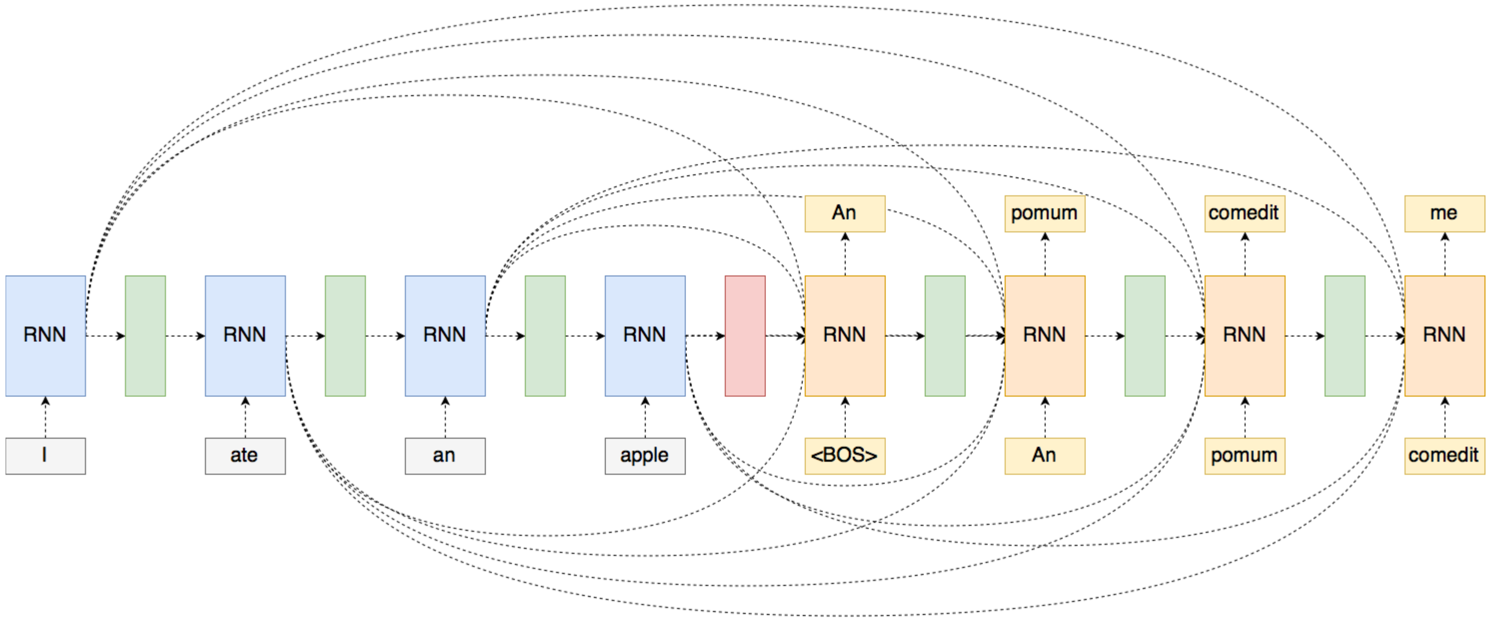
What's the Potential Problem? (cont'd)

- Outputs at different time steps have particular meanings.
- However, synchrony between **input** and **output seqs** is not required**.**



What's the Potential Problem? (cont'd)

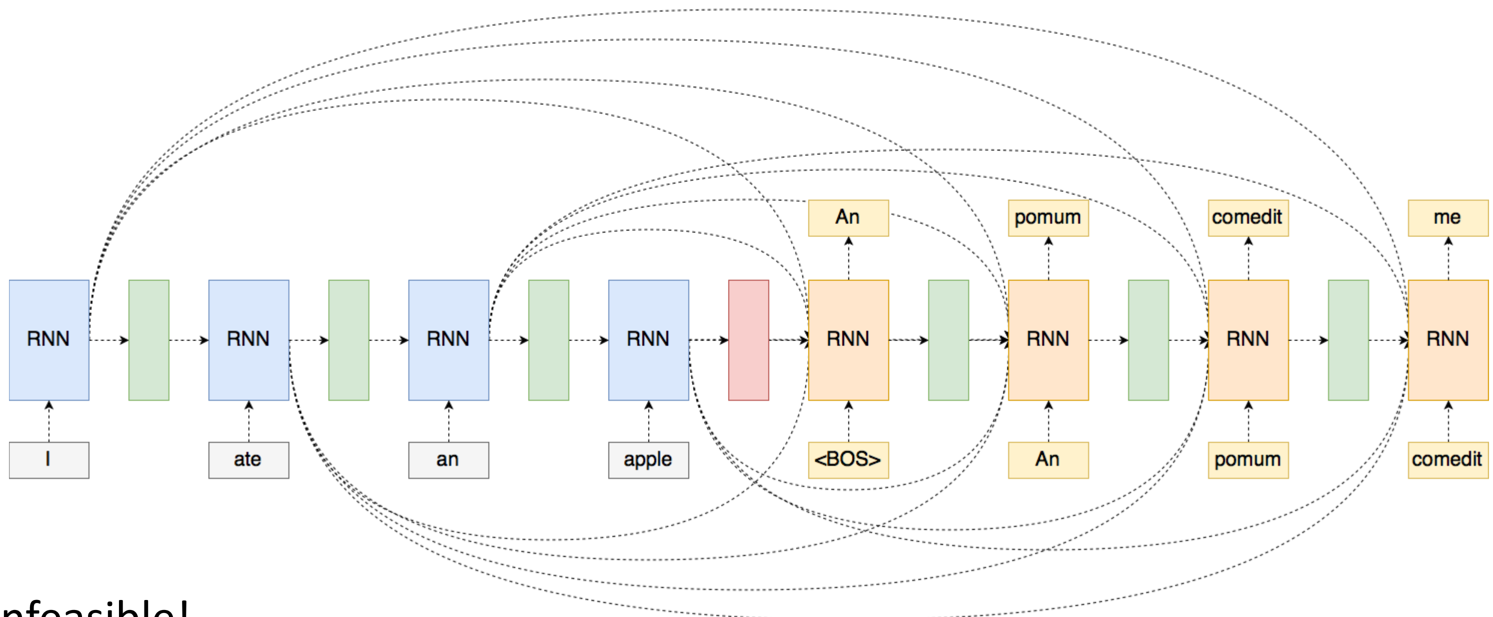
- Connecting every hidden state between encoder and decoder?



- Infeasible!
  - Both inputs and outputs are with varying sizes.
  - Overparameterized
  - Possible solution: **attention**

RNN with Attention is Good, But..

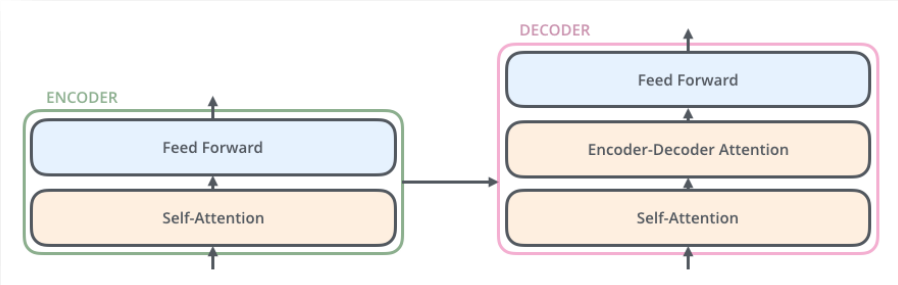
- Attention in a pre-defined sequential order
- Information loss due to long sequences…
- Connecting every hidden state between encoder and decoder?

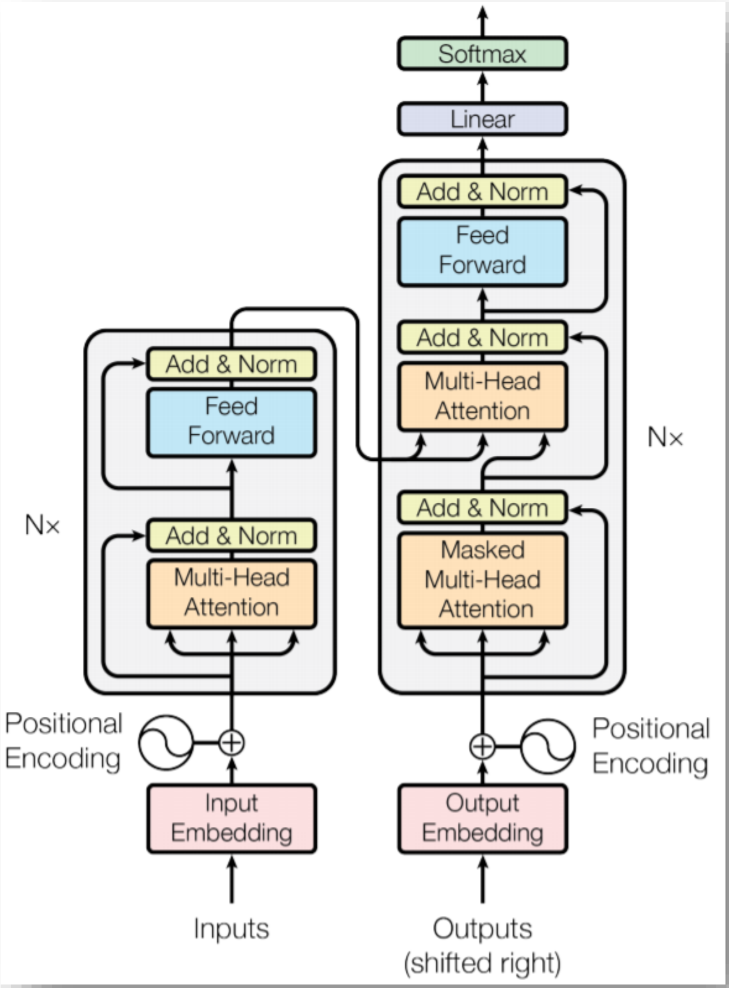


- Infeasible!
    - Both inputs and outputs are with varying sizes.
    - Overparameterized

Transformer

- "Attention is all you need", NIPS/NeurIPS 2017
- Self-attention for text translation
- Say goodbye to CNN & RNN
- More details available at: http://jalammar.github.io/illustrated-transformer/





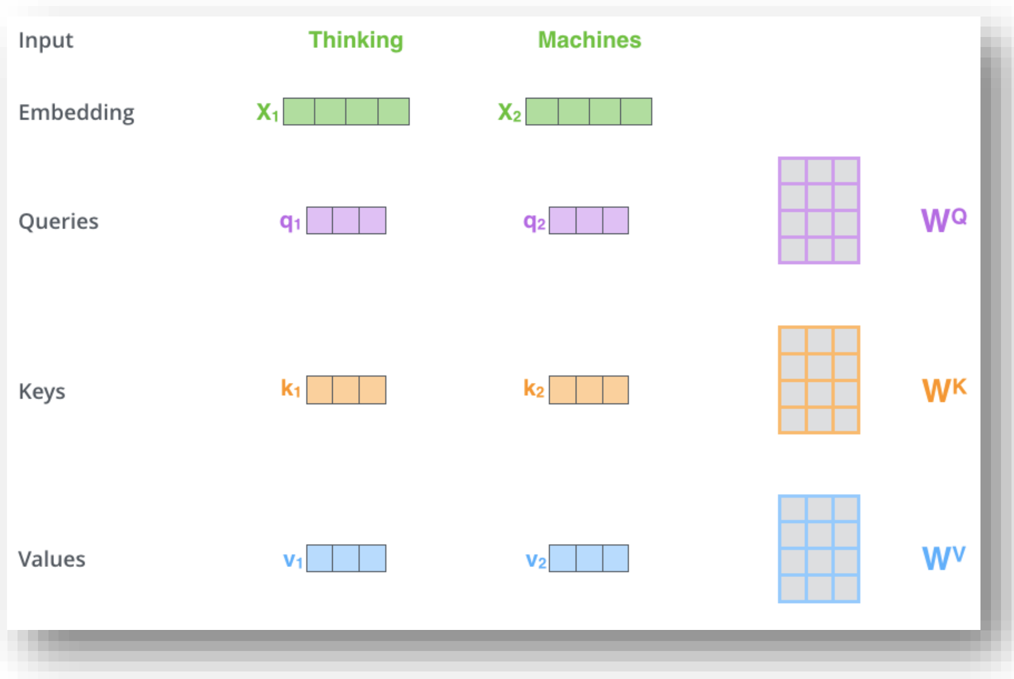
Self-Attention (1/5)

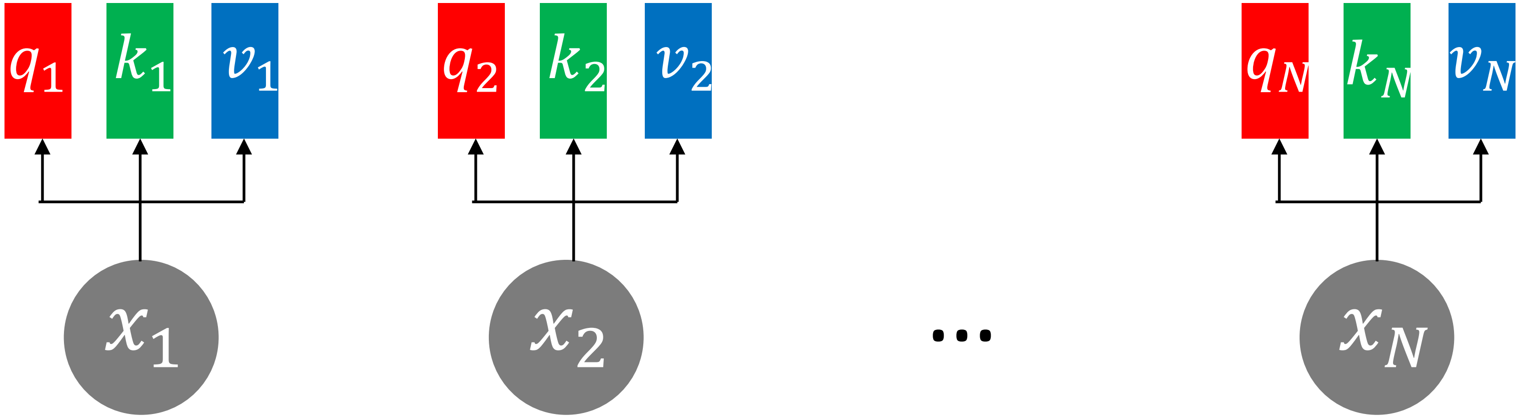
- Query $\boldsymbol{q}$, key $\boldsymbol{k}$, value $\boldsymbol{v}$ vectors are learned from each input $\boldsymbol{x}$

$$q_i = W^Q x_i$$

$$k_i = W^K x_i$$

$$v_i = W^V x_i$$

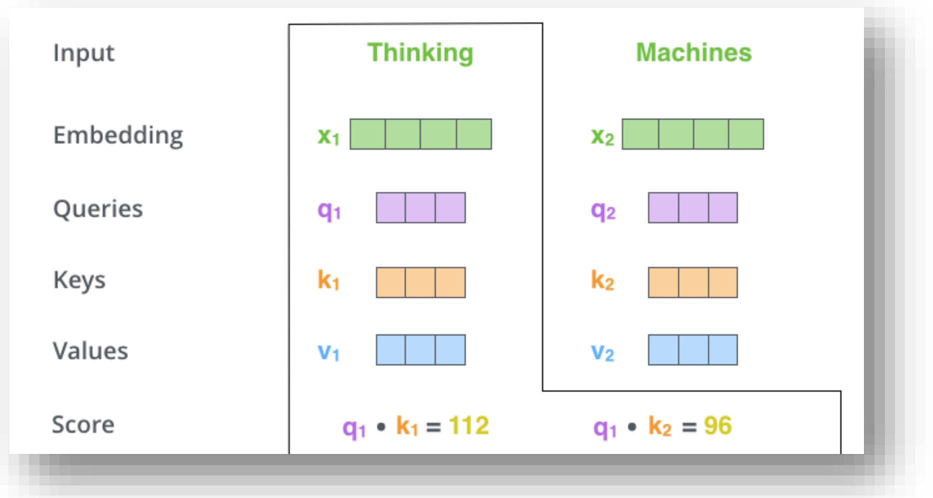


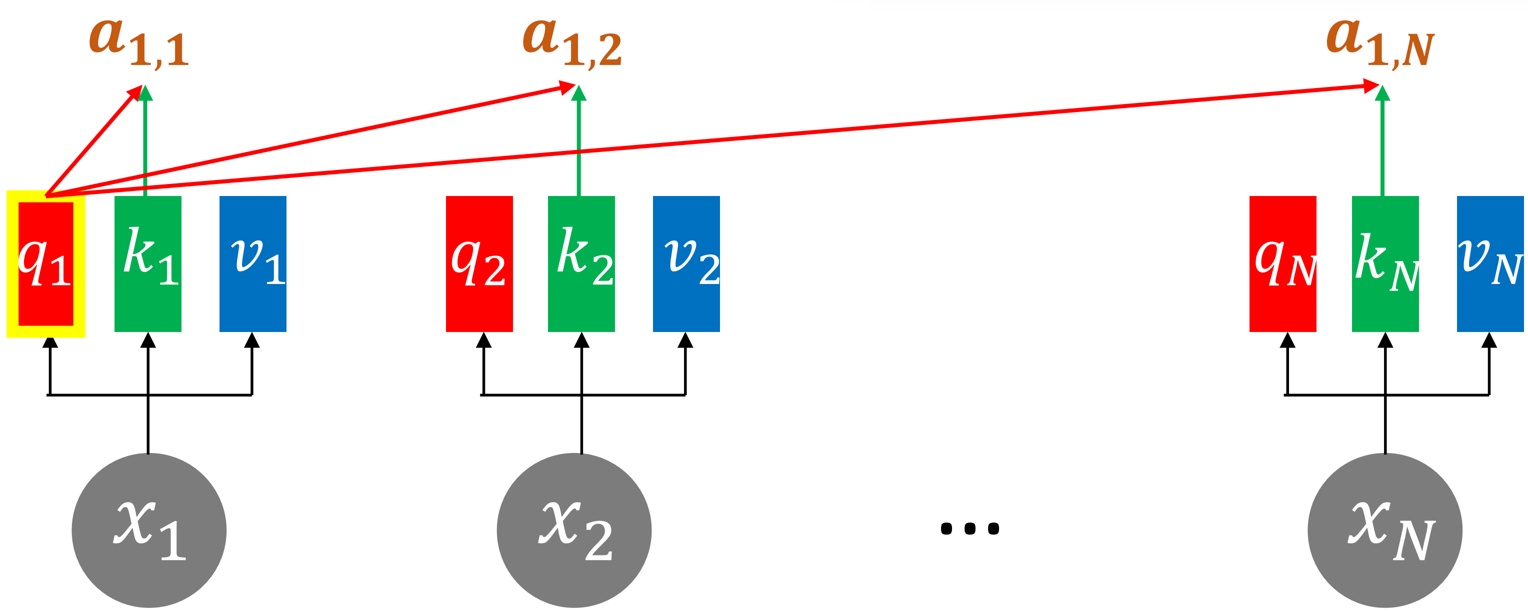


Self-Attention (2/5)

- Relation between each input is modeled by inner-product of query ***q*** and key ***k***.

$$a_{1,i} = \frac{q_1 \cdot k_i}{\sqrt{d}}, \text{ where } a \in R, q, k \in R^d$$

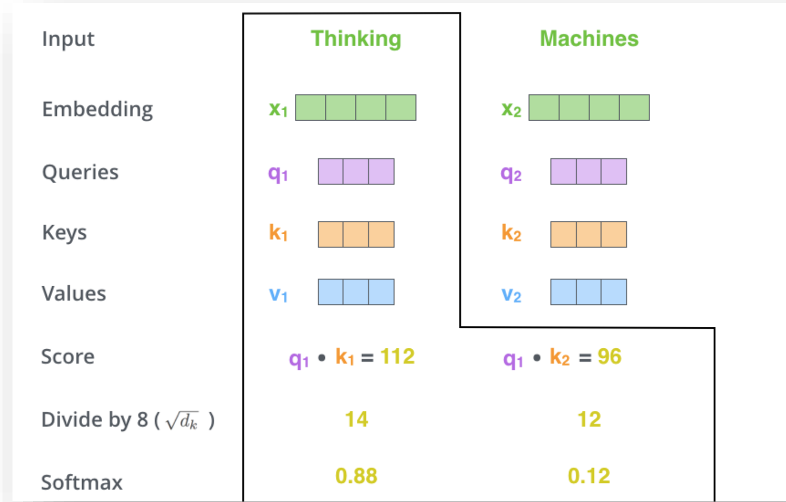


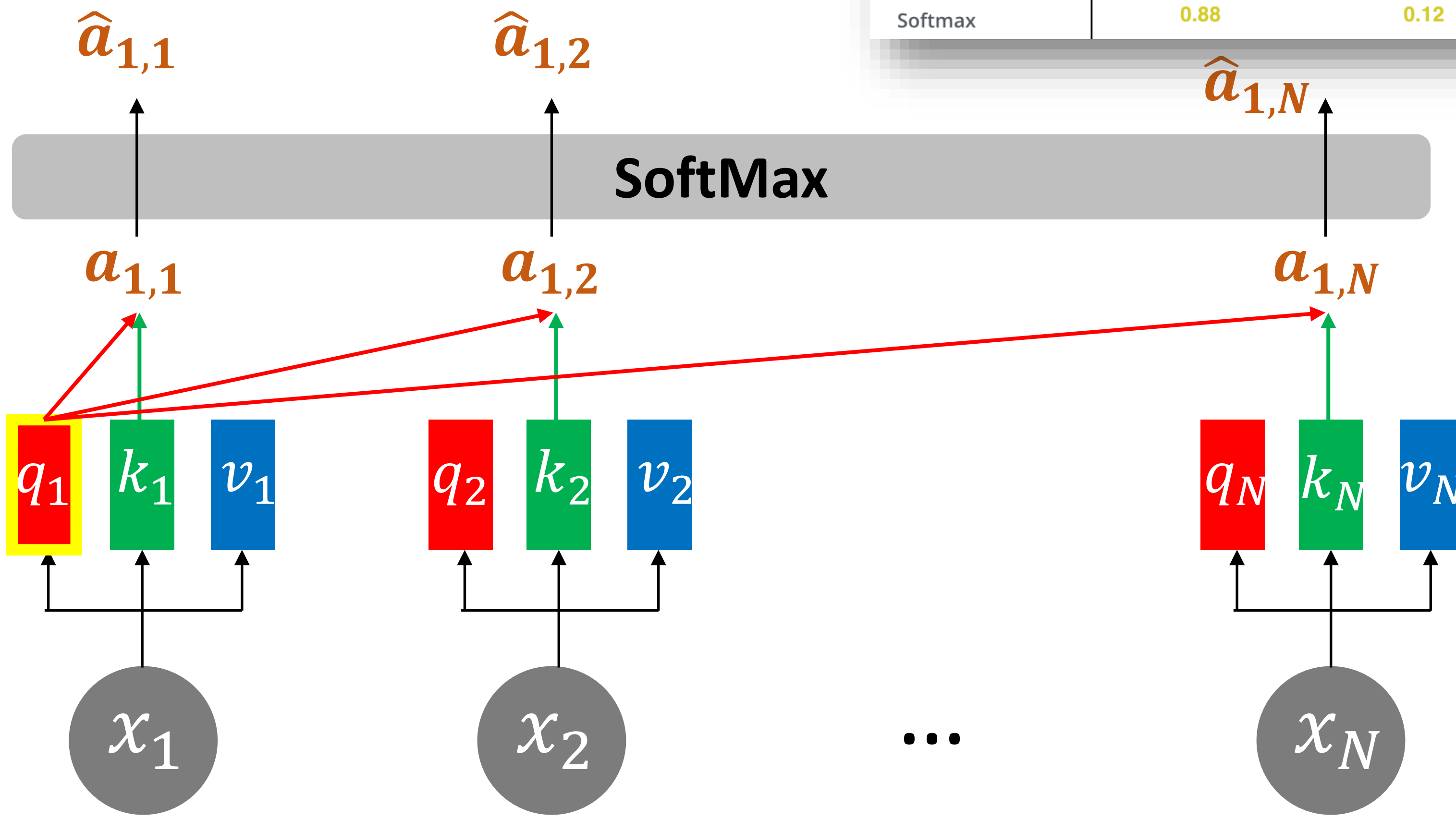


Self-Attention (3/5)

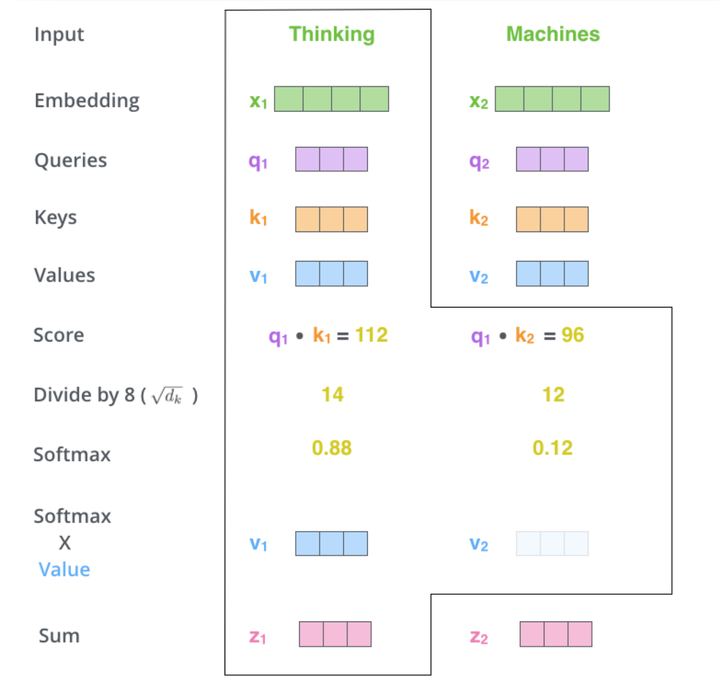
- SoftMax is applied:

  $0 \leq \hat{a}_i = e^{a_i} / \sum_j^{\mathrm{N}} e^{a_j} \leq 1$, for I =1, ..., N

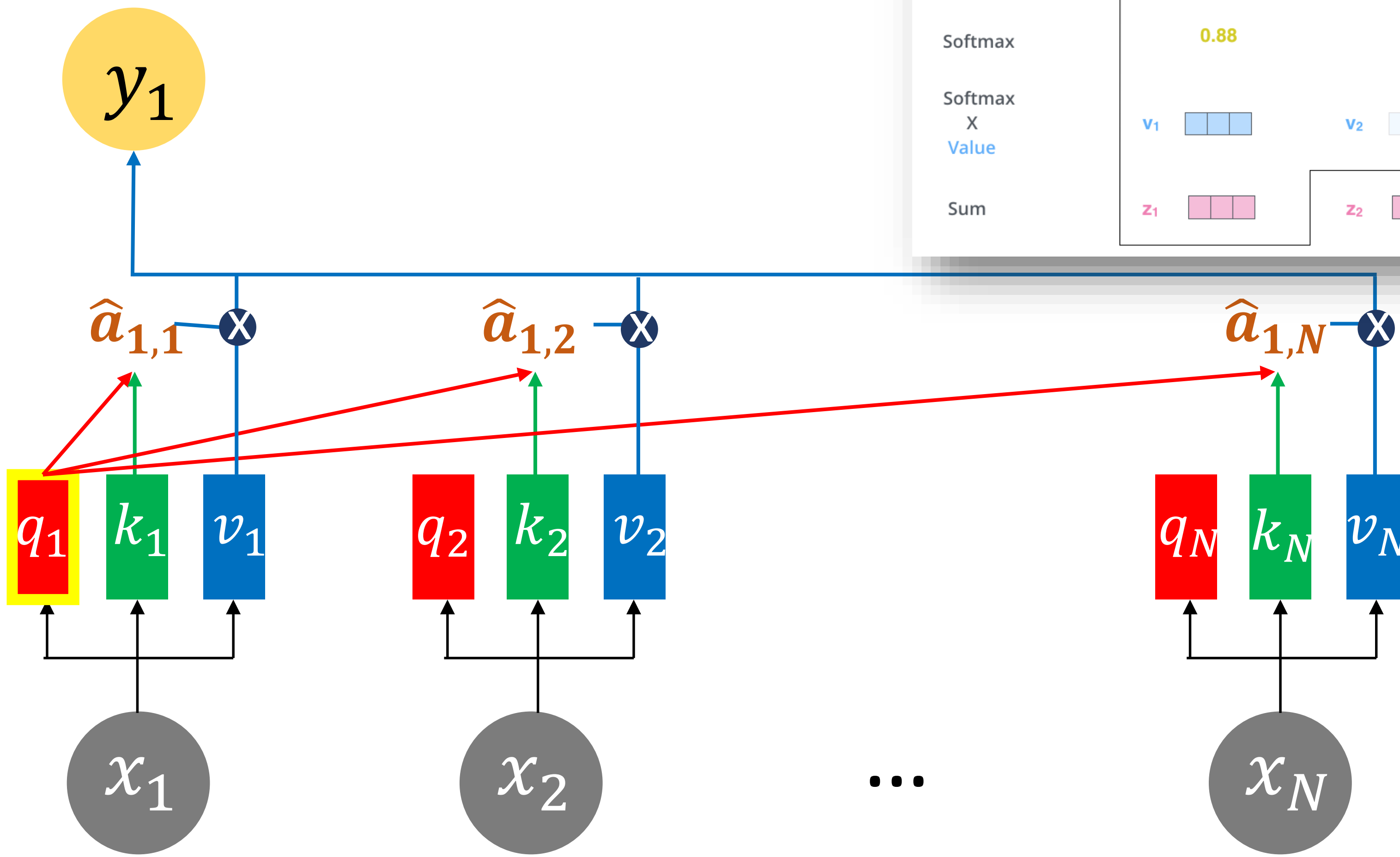




Self-Attention (4/5)

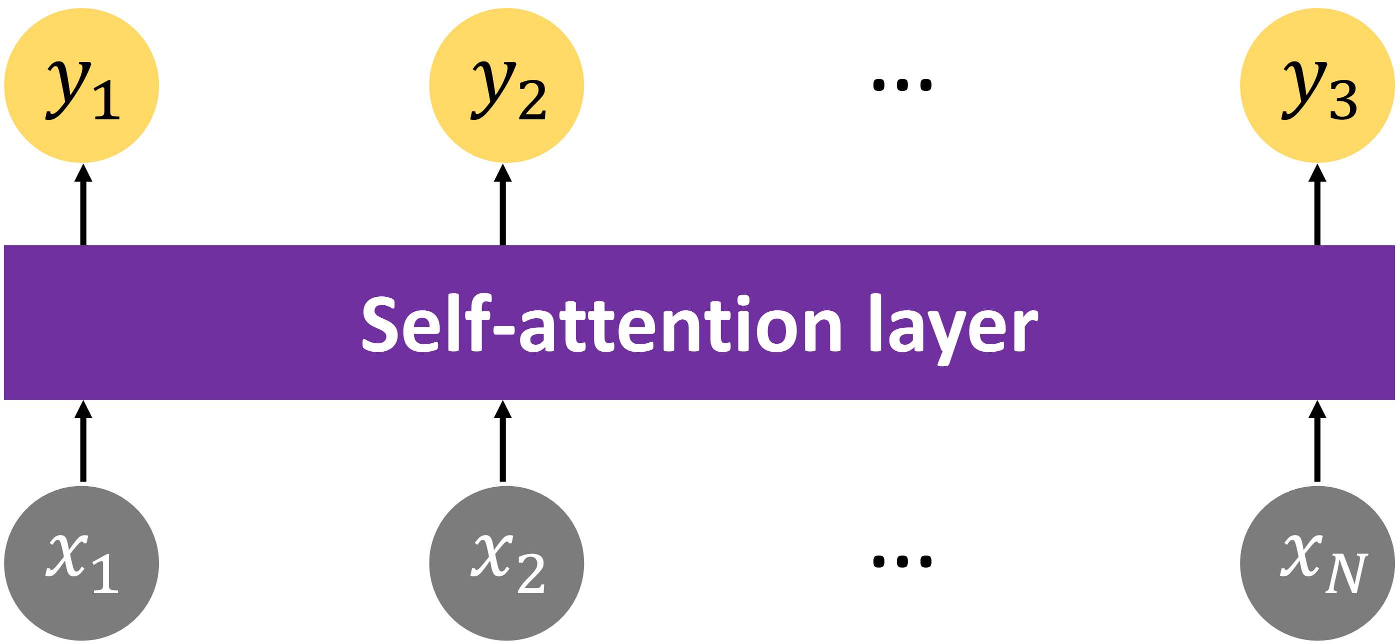


- Value vectors $\boldsymbol{v}$ are aggregated with attention weight $\hat{a}$, i.e., $y_1 = \sum_i^N \hat{a}_i \cdot v_i$



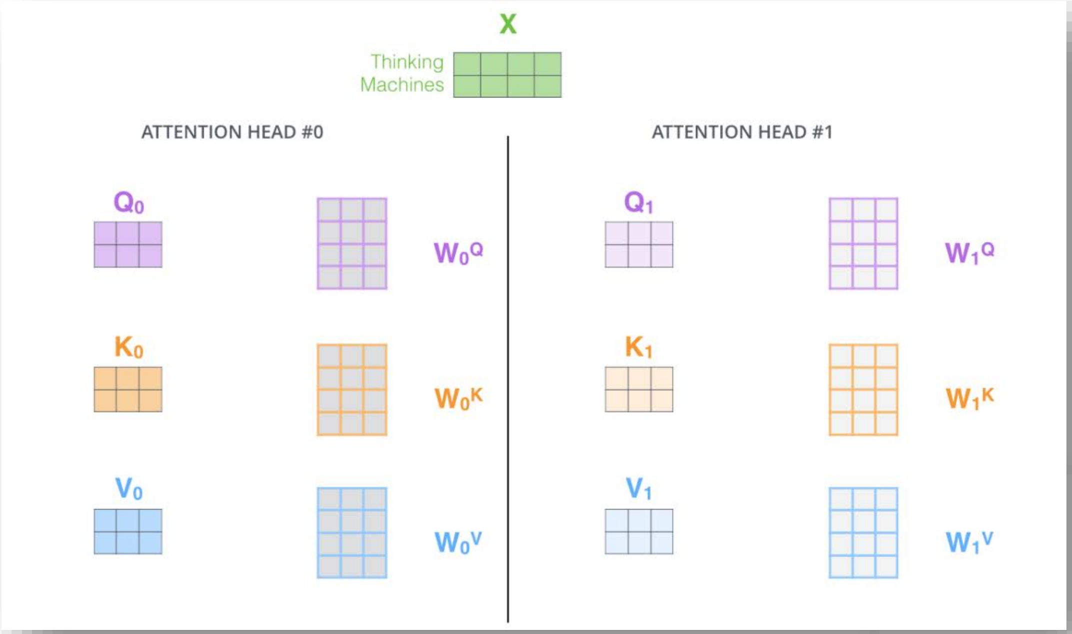
Self-Attention (5/5)

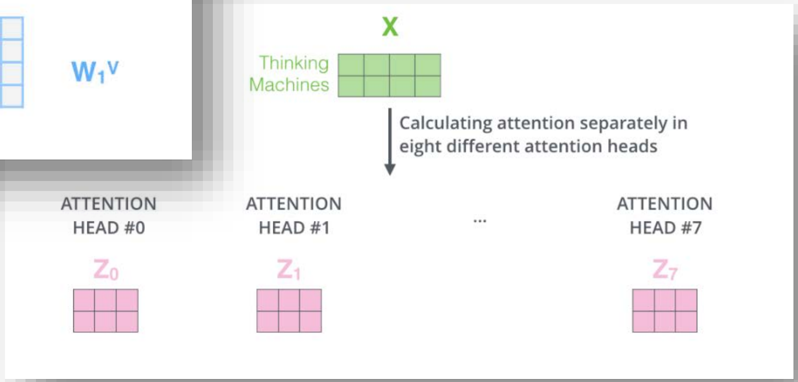
- All y_i can be computed **in parallel**
- Each y_i considers $x_1 \sim x_N$, modeling their **long-distance dependencies**.
- Global feature can be obtained by **average-pooling** over $y_1 \sim y_N$



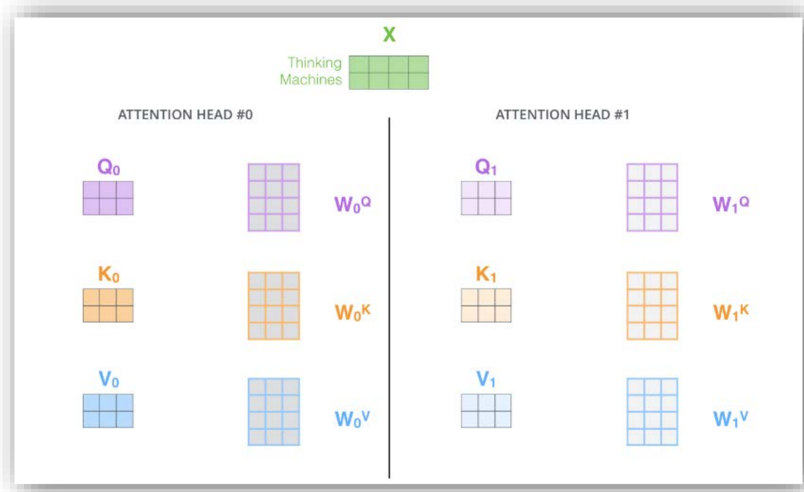
Multi-Head Self-Attention (1/4)

- Perform self-attention at different subspaces, implying performing attention over different input feature types (e.g., representations, modalities, positions, etc.)

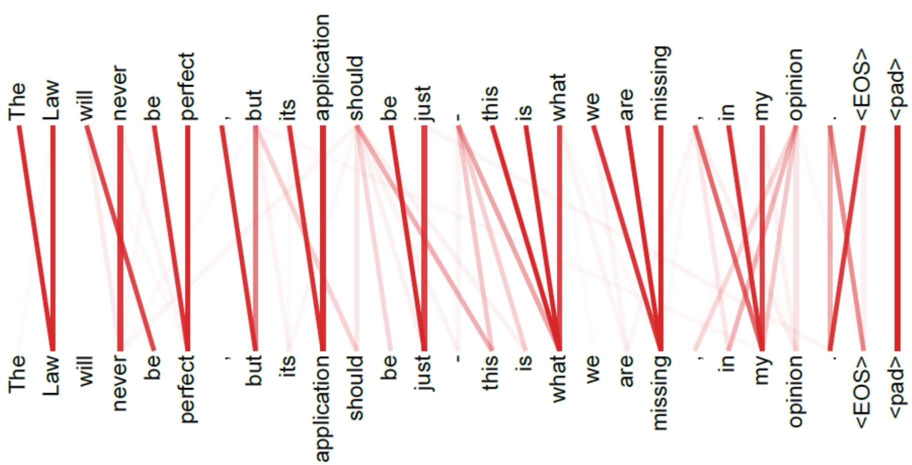




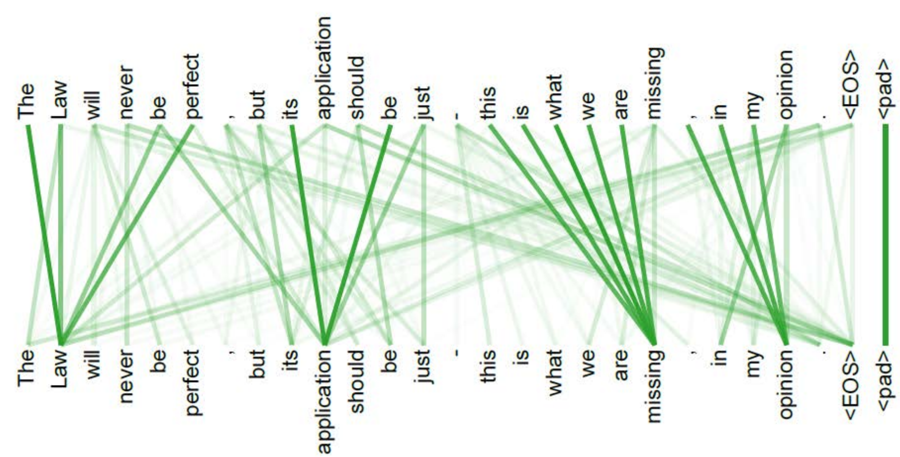
Multi-Head Self-Attention (2/4)



- Perform self-attention at different subspaces, implying performing attention over different input feature types
- See example below



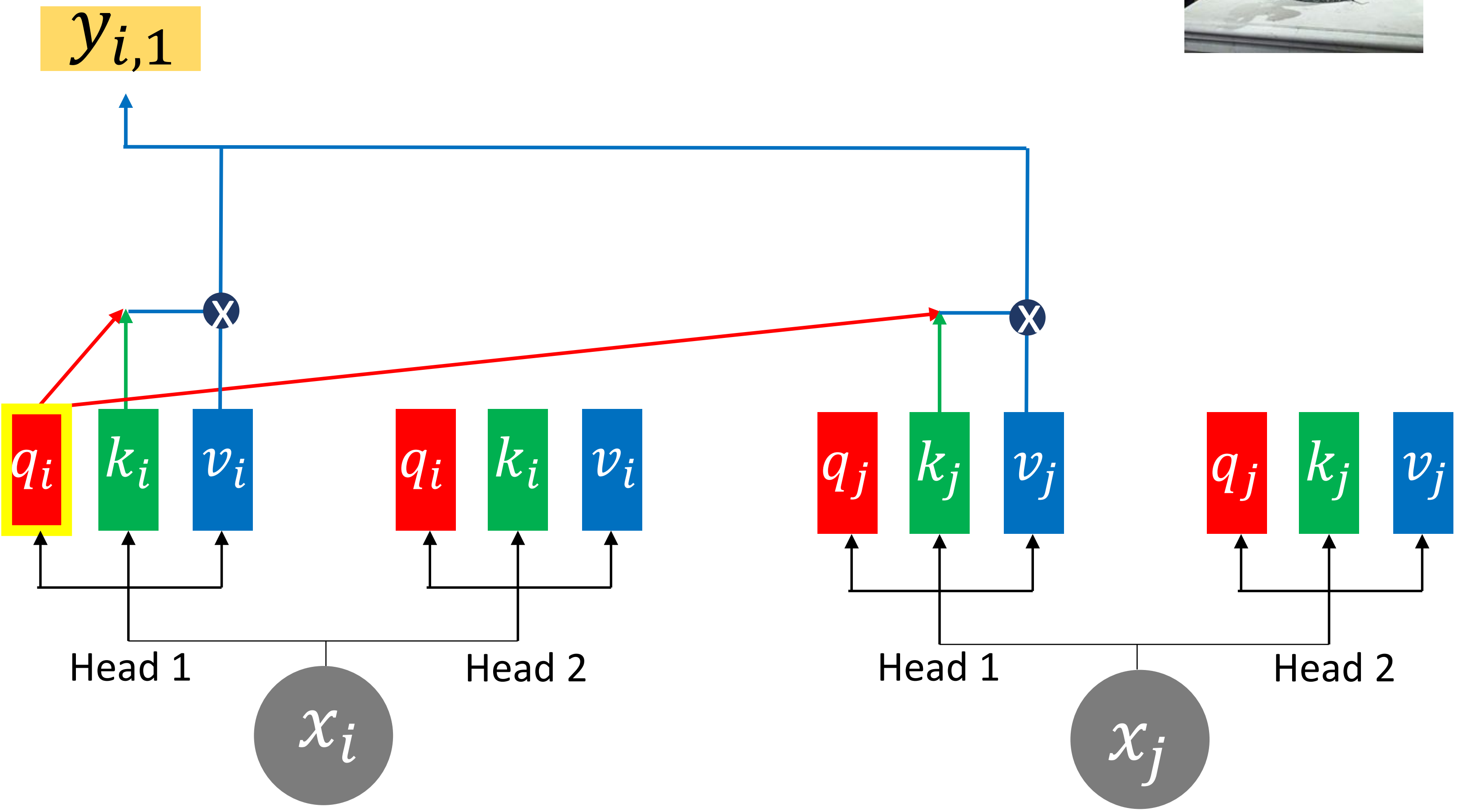
Attention weights of Head 1



Attention weights of Head 2

Multi-Head Self-Attention (3/4)

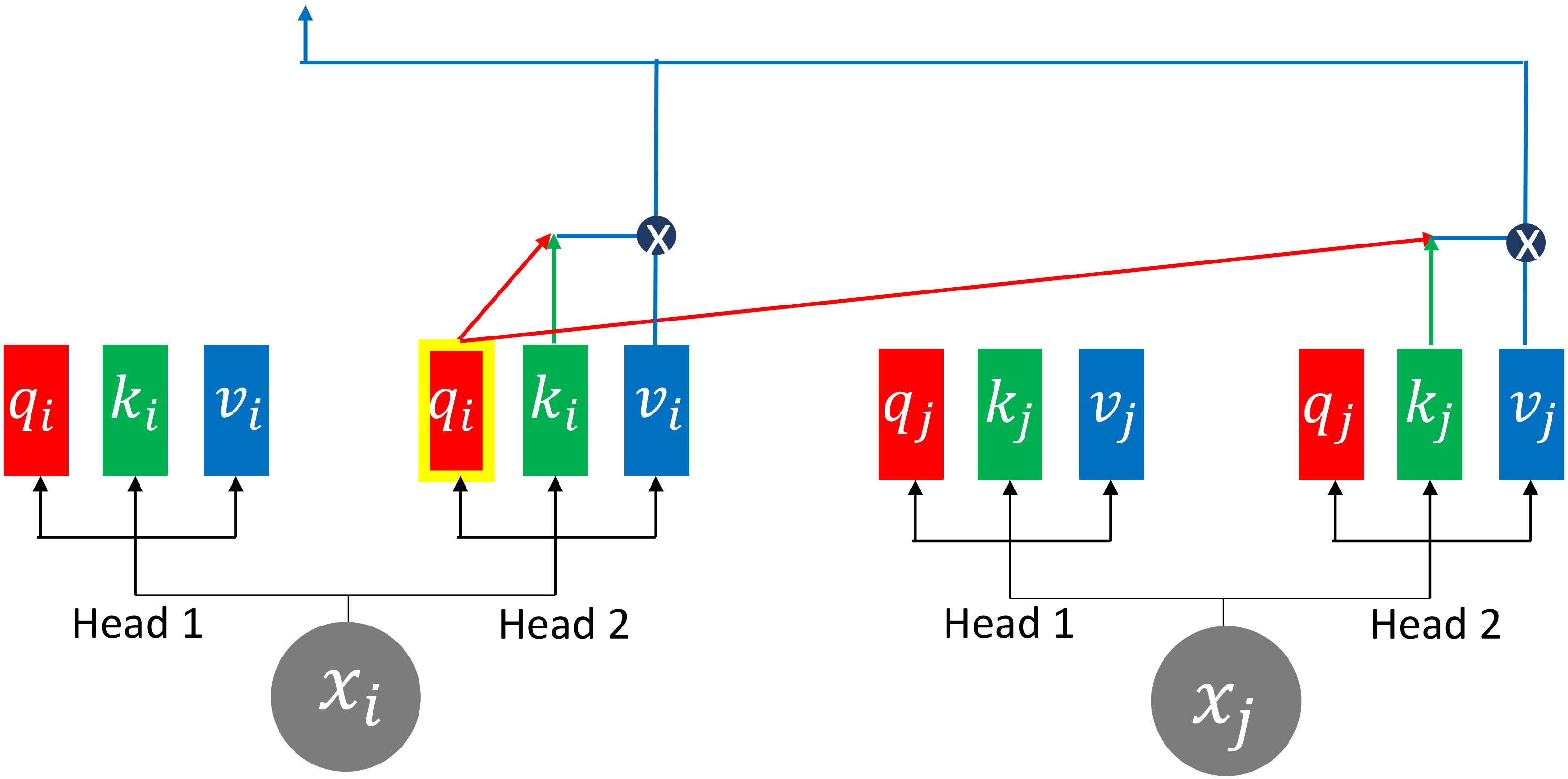
- A 2-head example, output of two heads are concatenated.



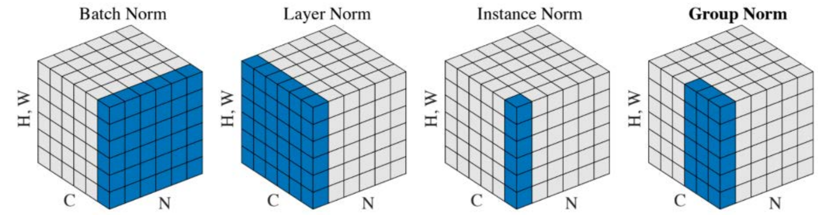
Multi-Head Self-Attention (4/4)

- A 2-head example, output of two heads are concatenated.

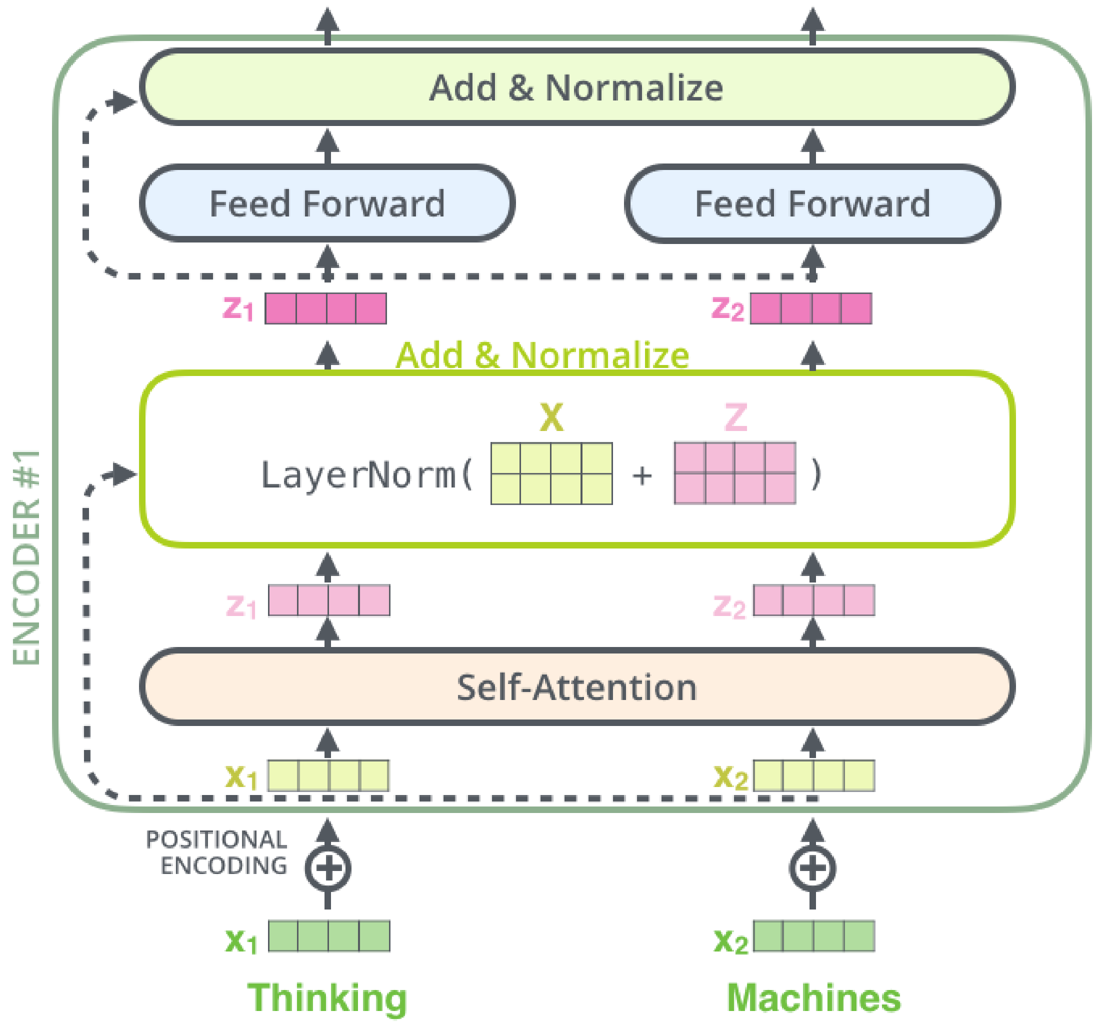
$$y_{i,1} \; y_{i,2} = y_i$$



The Residuals

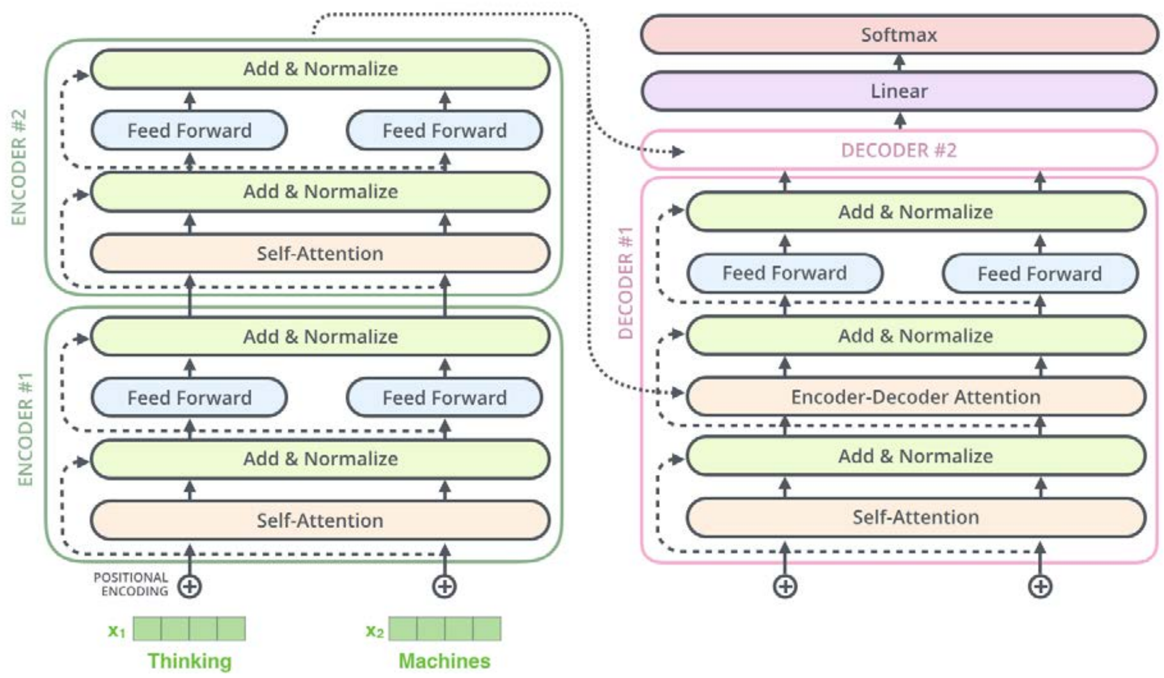


- A **residual connection** followed by layer normalization



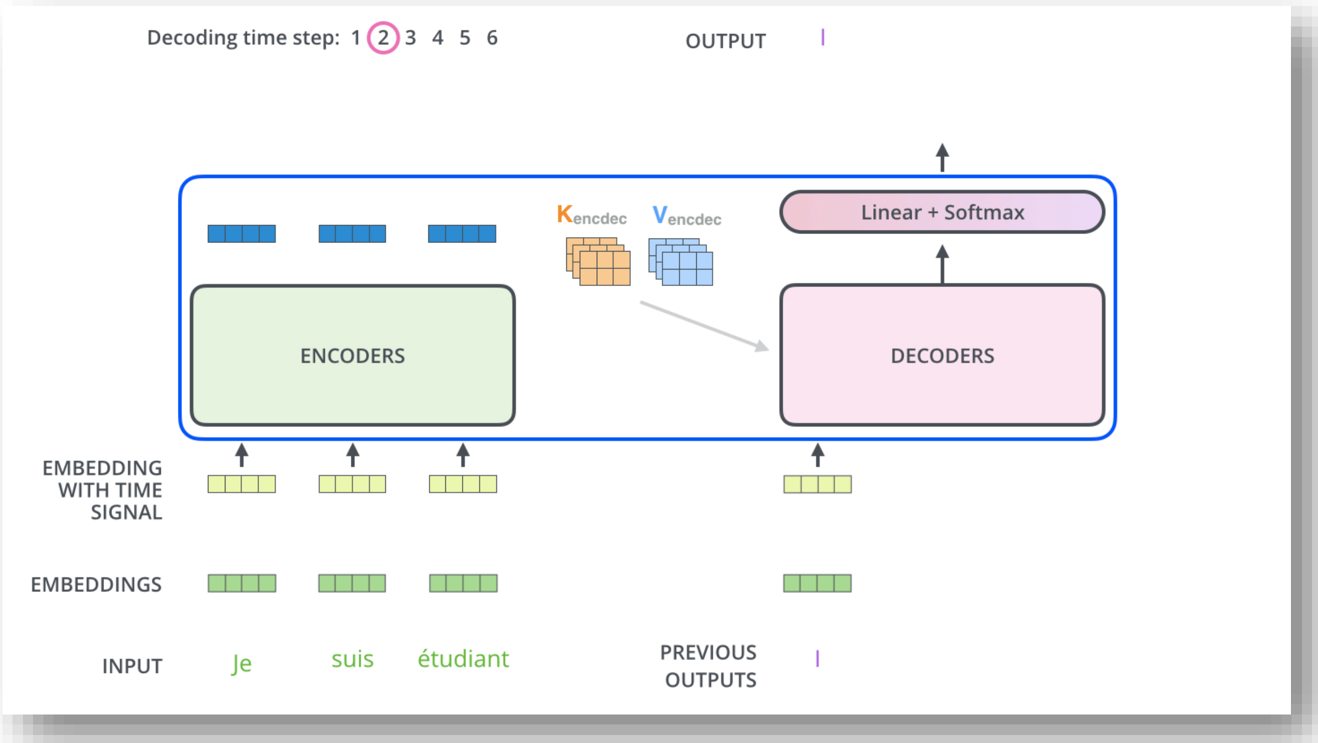
The Decoder in Transformer

- Encoder-decoder attention
  - Q from self-attn in decoder, K & V from encoder outputs
- Masked multi-head attention
  - Design similar to that of encoder, except for decoder #1 which takes additional inputs (of GT/predicted word embeddings).
  - Mask unpredicted tokens during softmax: why?



The Decoder in Transformer (cont'd)

- Encoder-decoder attention
  - Q from self-attn in decoder, K & V from encoder outputs
- Masked multi-head attention
  - Design similar to that of encoder, except for decoder #1 which takes additional inputs (of GT/predicted word embeddings).
  - Mask unpredicted tokens during softmax: why?



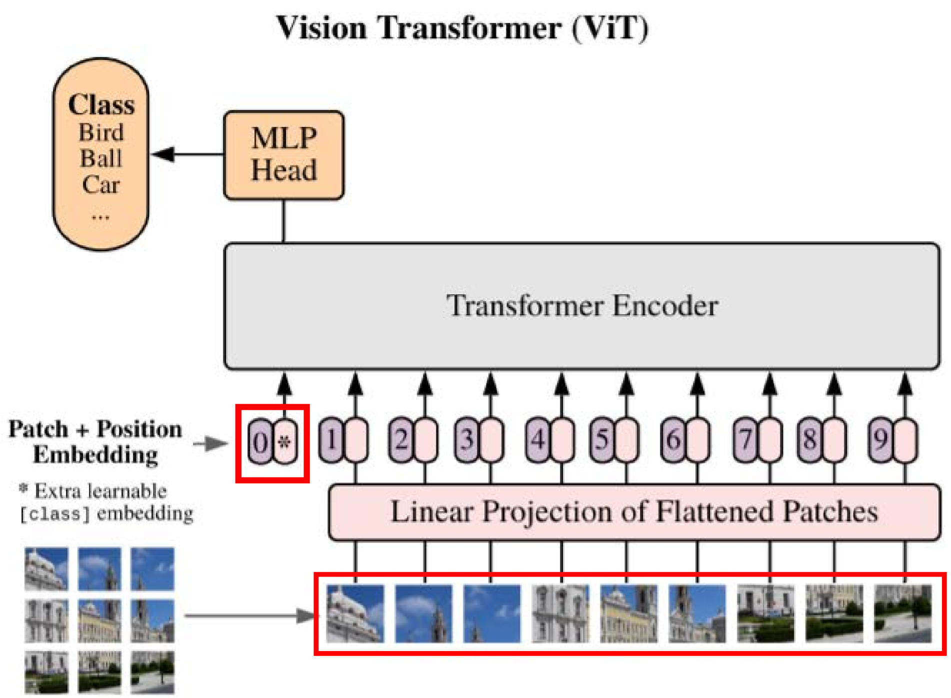
Overview of Decoding in Transformer

- Encoder/Decoder Cross-Attention + Decoder self-attention

https://medium.com/ml-note/autoencoder-%E4%B8%89-self-attention-transformer-c37f719d222

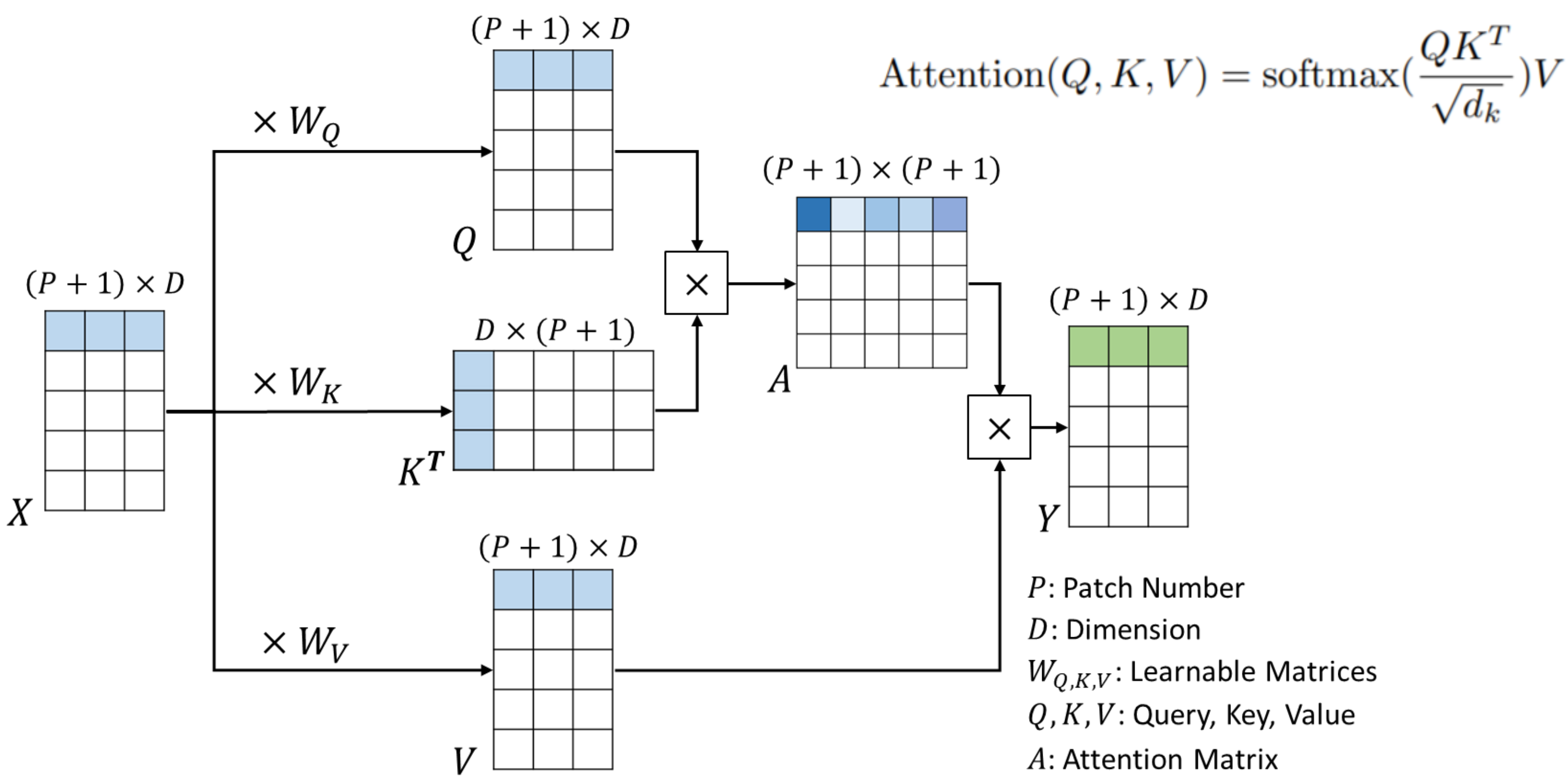
Vision Transformer

- "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR, 2021. (Google Research)
- Partition the input image into a **patch sequence**
- An additional **token** (*) is appended to perform attention on patches
- Both the "*" token and positional embeddings (denoted by 0, 1, 2 …) are **trainable vectors**.



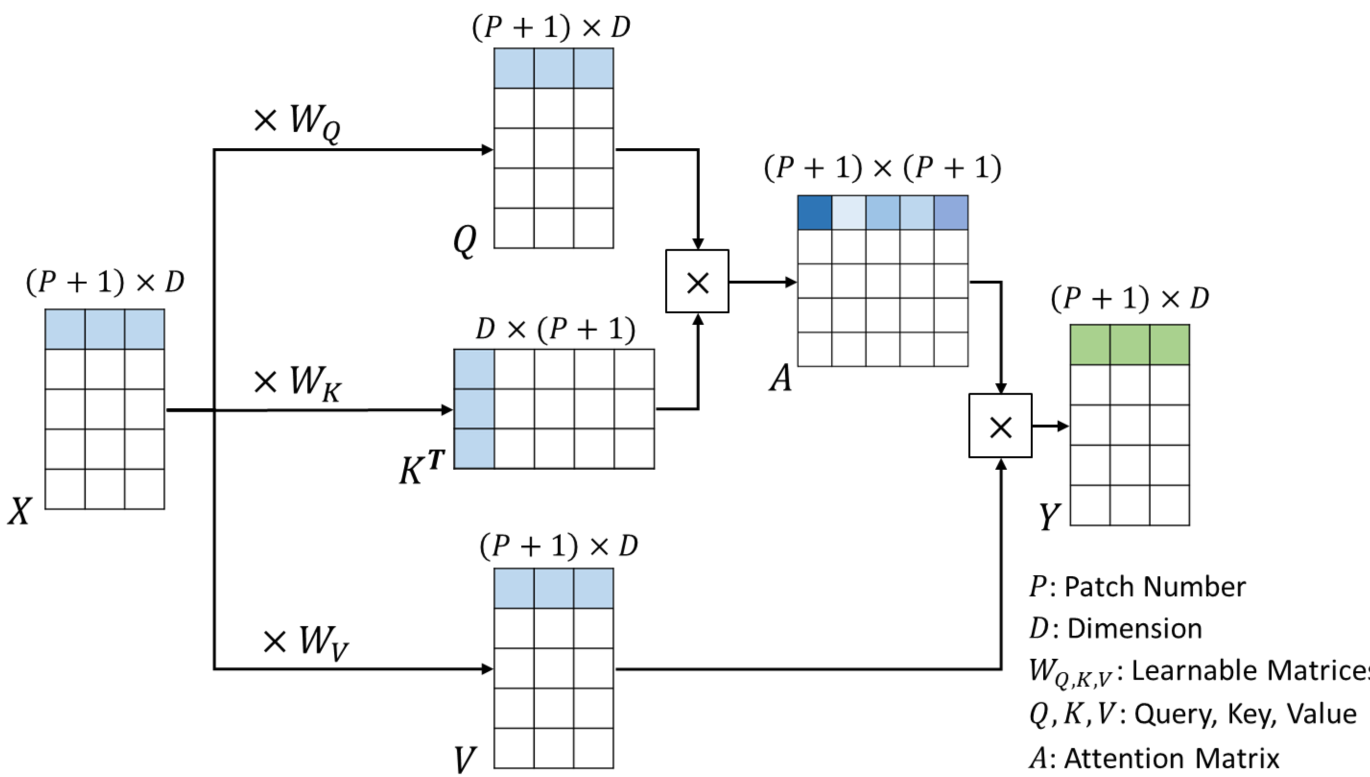
Query-Key-Value Attention in ViT

- E.g., An input image is partitioned into 4 patches, with feature dimension = 3 (i.e., P=4 and D=3).
- Note that there are (P+1) rows since we have an additional token of *.



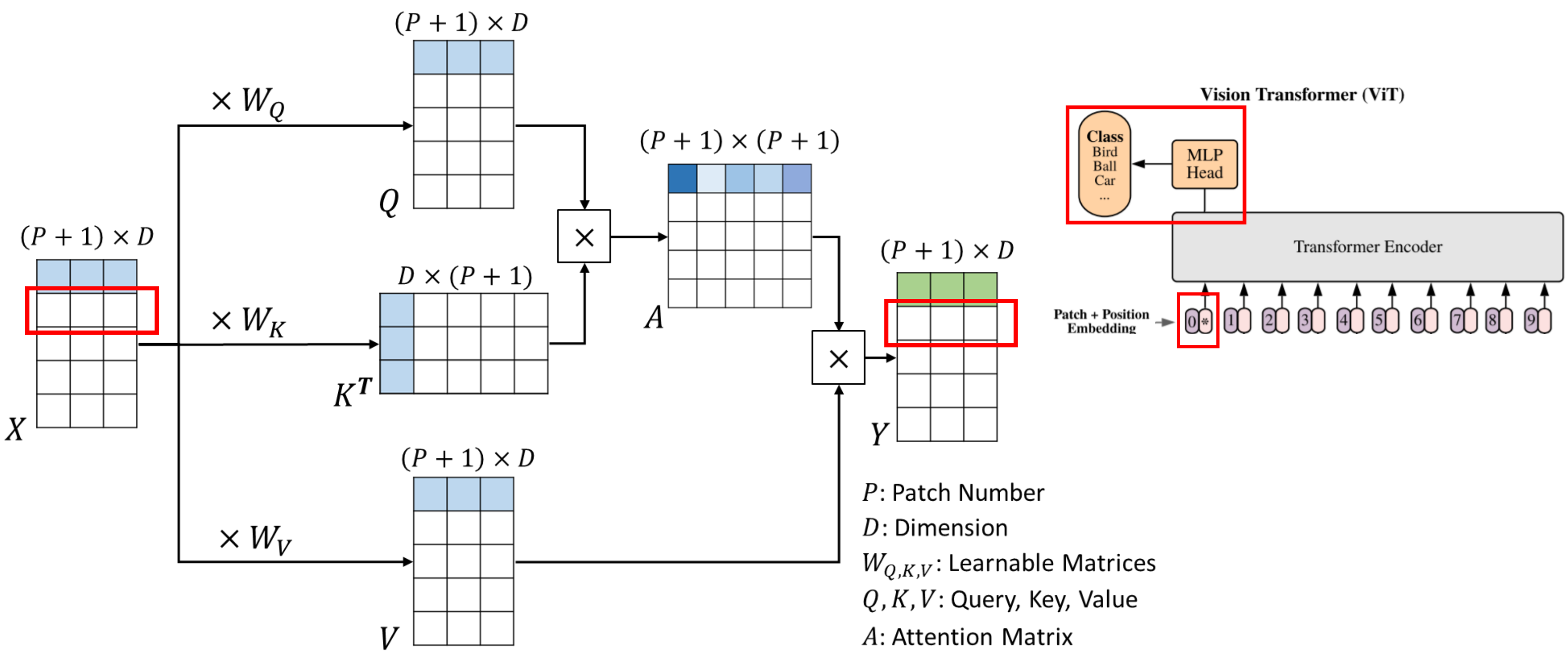
Query-Key-Value Attention in ViT (cont'd)

- By performing attention, the input sequence X (of length P+1) is "transformed" into another sequence Y with the same length
- Again, that's why it is called "**Transformer**" and as a **seq2seq** model.



Query-Key-Value Attention in ViT (cont'd)

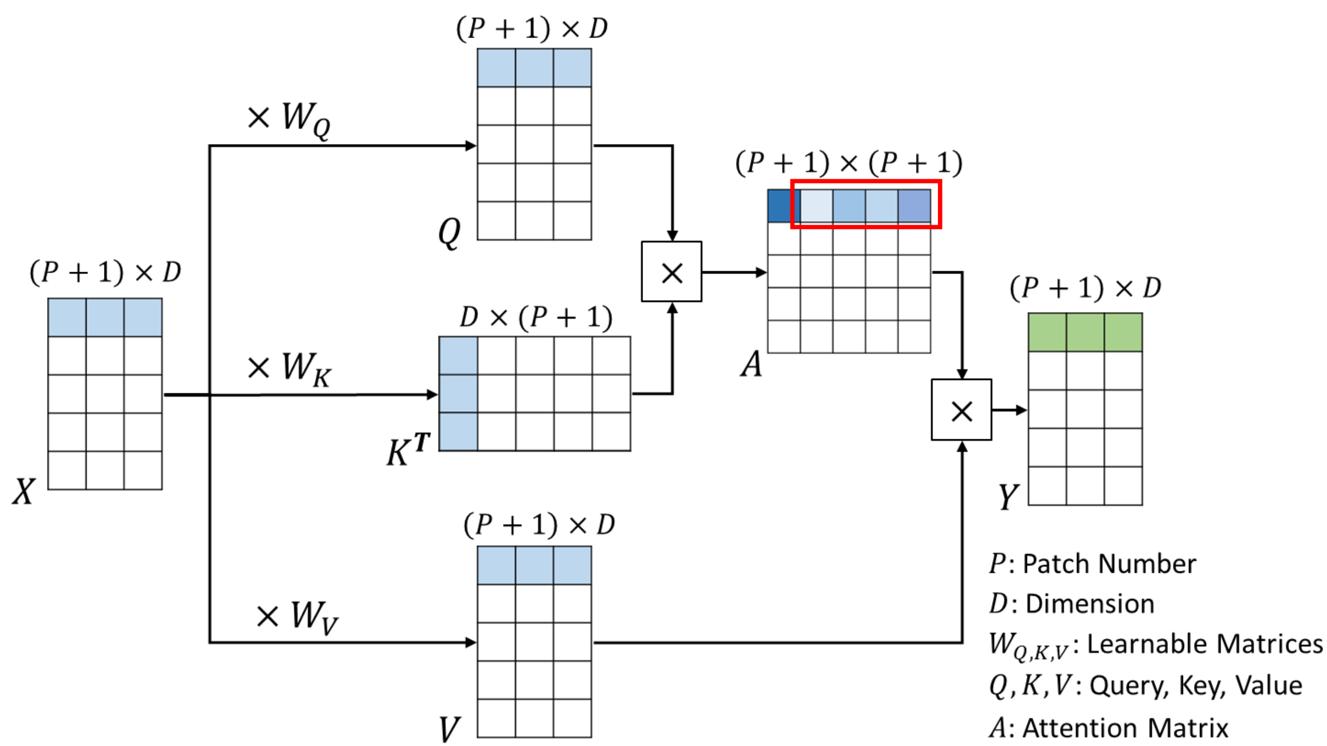
- In standard vision transformer, we only take the **first output token** of the output sequence (the **first row** of Y) for classification purposes
- This corresponds to the output when **token "0"** serves as query



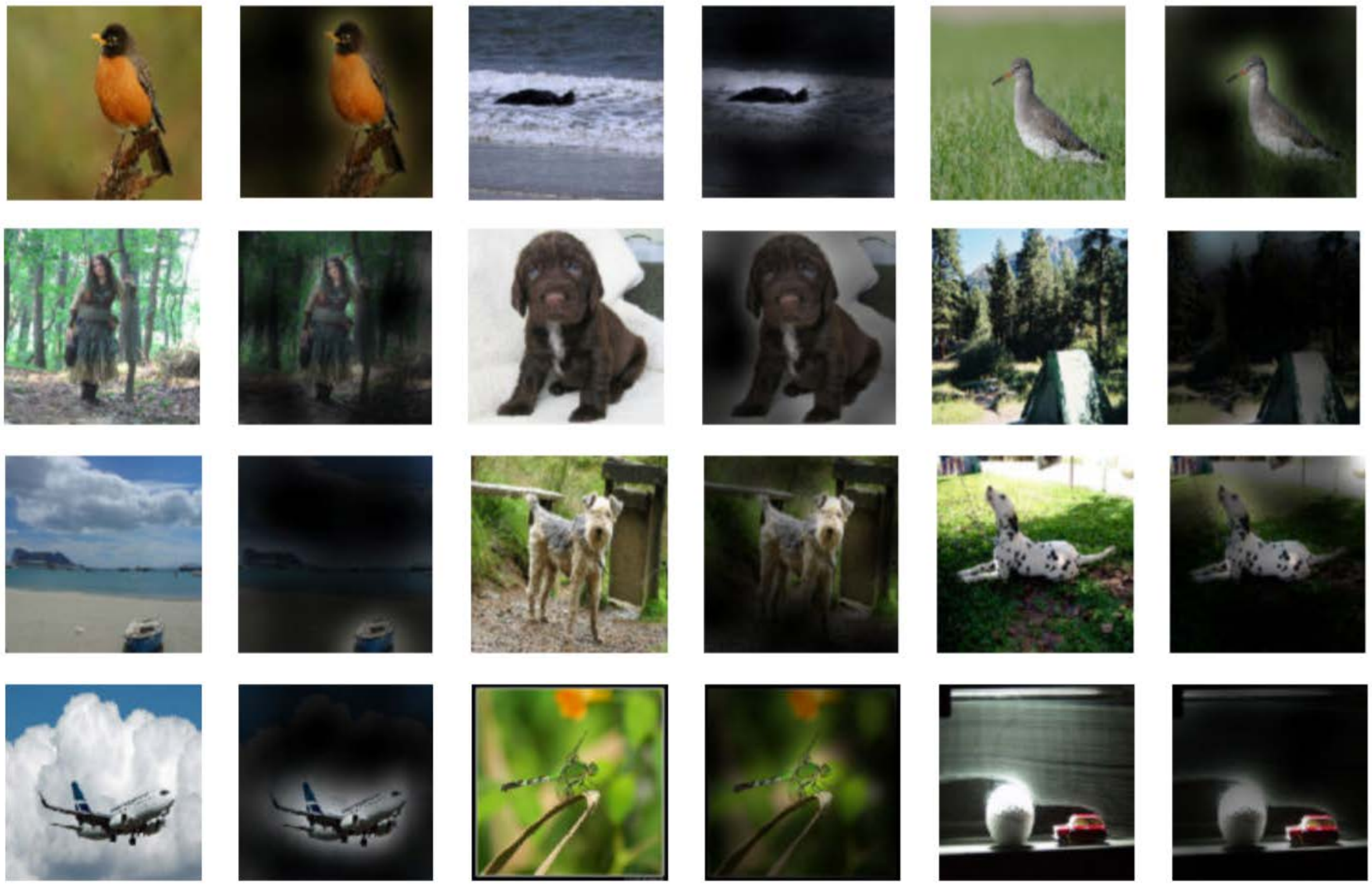
P: Patch Number
D: Dimension
$W_{Q,K,V}$: Learnable Matrices
Q, K, V: Query, Key, Value
A: Attention Matrix

Visualization of ViT

- To visualize the attention maps, we take the attention scores from the **first row** of A (when token "0" serves as query)
- Note the first element is excluded, and thus there are **P scores** corresponding to the P image patches



Example Visualization for Image Classification

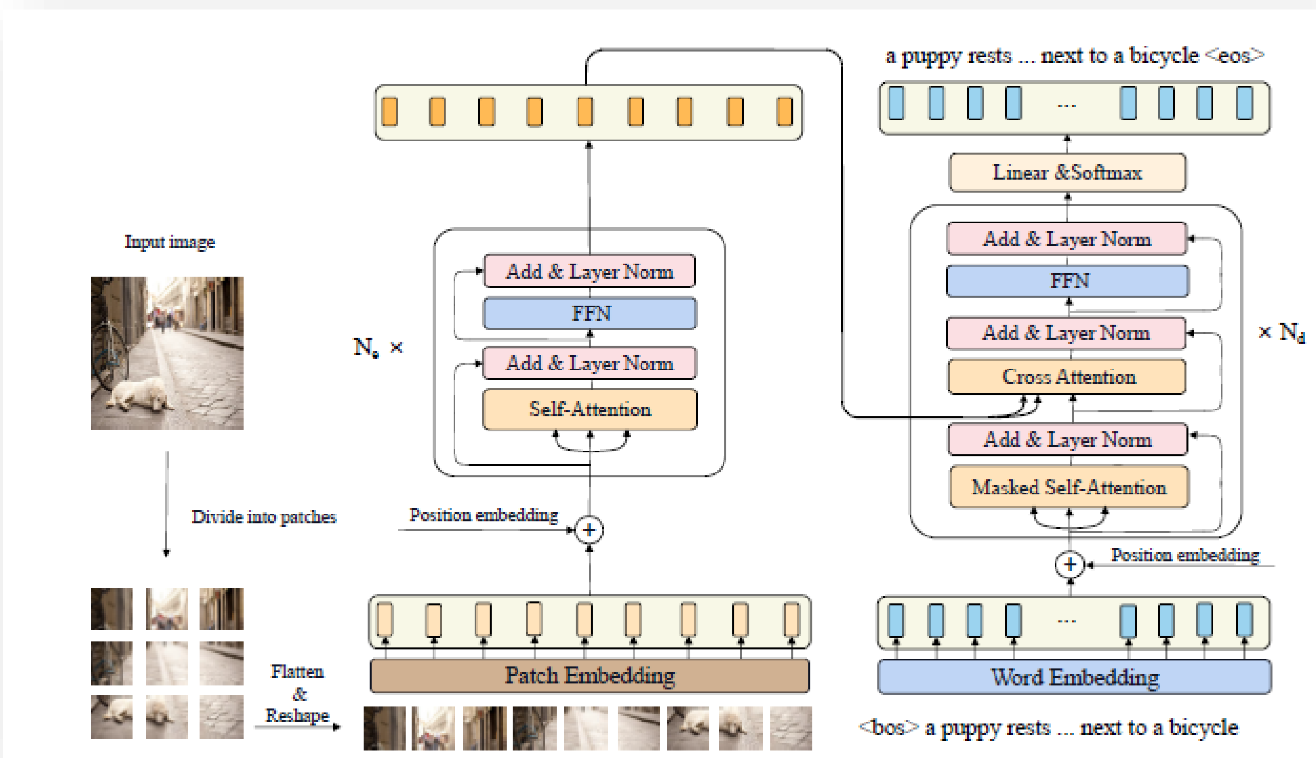


What'd Be Covered in This Crash Course…

- **Learning-based Computer Vision**
  - From Linear to Non-Linear Classifiers
- **Start of Deep Learning for Computer Vision**
  - Convolutional Neural Networks
  - Self-Supervised Learning
  - Segmentation & Detection
- **Generative Models**
  - Autoencoder, Variational Autoencoder,
    Generative Adversarial Networks & Diffusion Models
- **Sequence-to-Sequence Learning**
  - Attention is All You Need: Transformer
- **Vision & Language Foundation Models**
  - Image-to-Text vs. Text-to-Image
  - Parameter-Efficient Fine-tuning

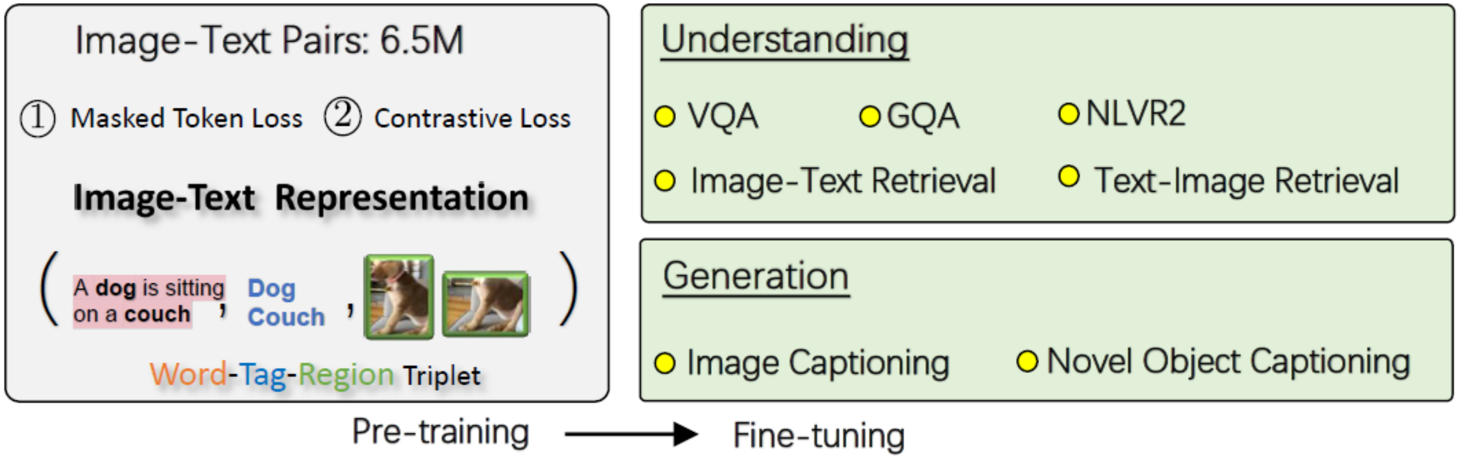
Image Captioning

- **Cap**tion **T**ransformer (CPTR) -
  CPTR: Full Transformer Network for Image Captioning, arxiv 2021
- Motivation: patch translation for image captioning



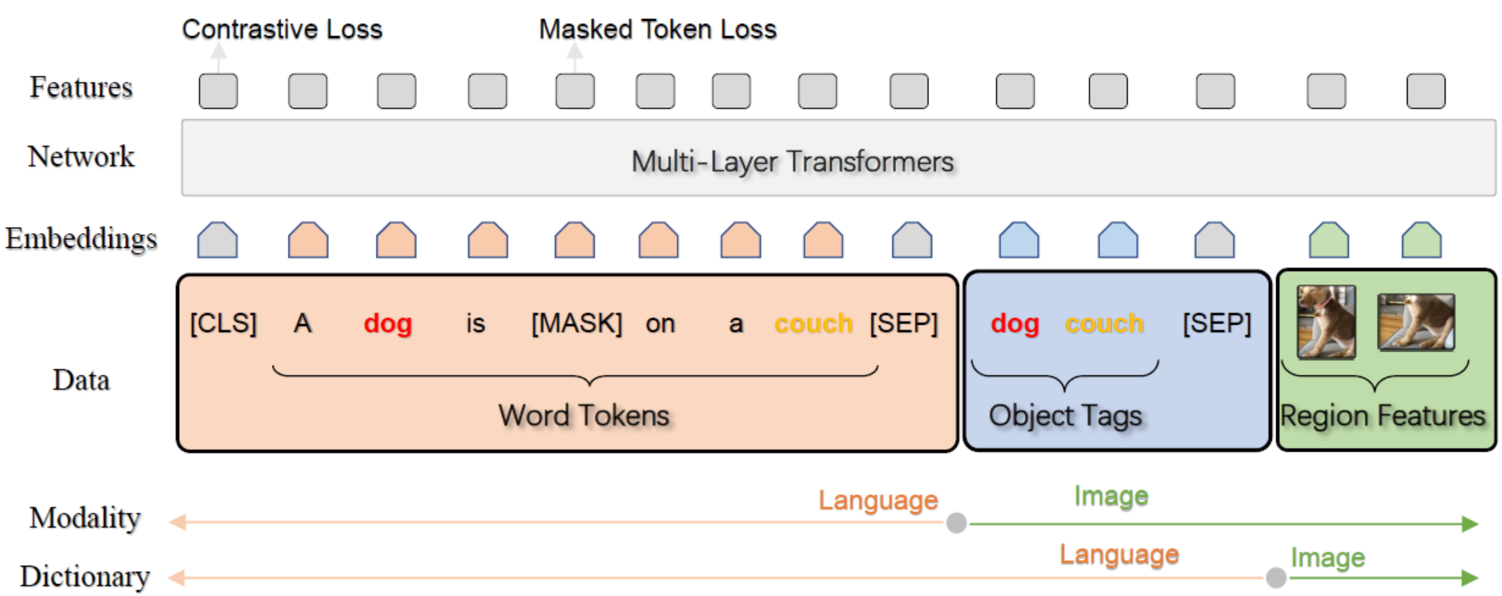
Beyond Image Captioning: Unified Vision & Language Model

- **Oscar**: **O**bject-**S**emanti**c**s **A**ligned P**r**e-training for Vision-Language Tasks (ECCV'20)
  - Training data:
    triplets of caption-tag-region
  - Objectives:
    1. Masked token loss for words & tags
    2. Contrastive loss tags and others
  - Fine-tuning:
    5 vision & language tasks (VQA, image-text retrieval, image captioning, NOC, etc.)



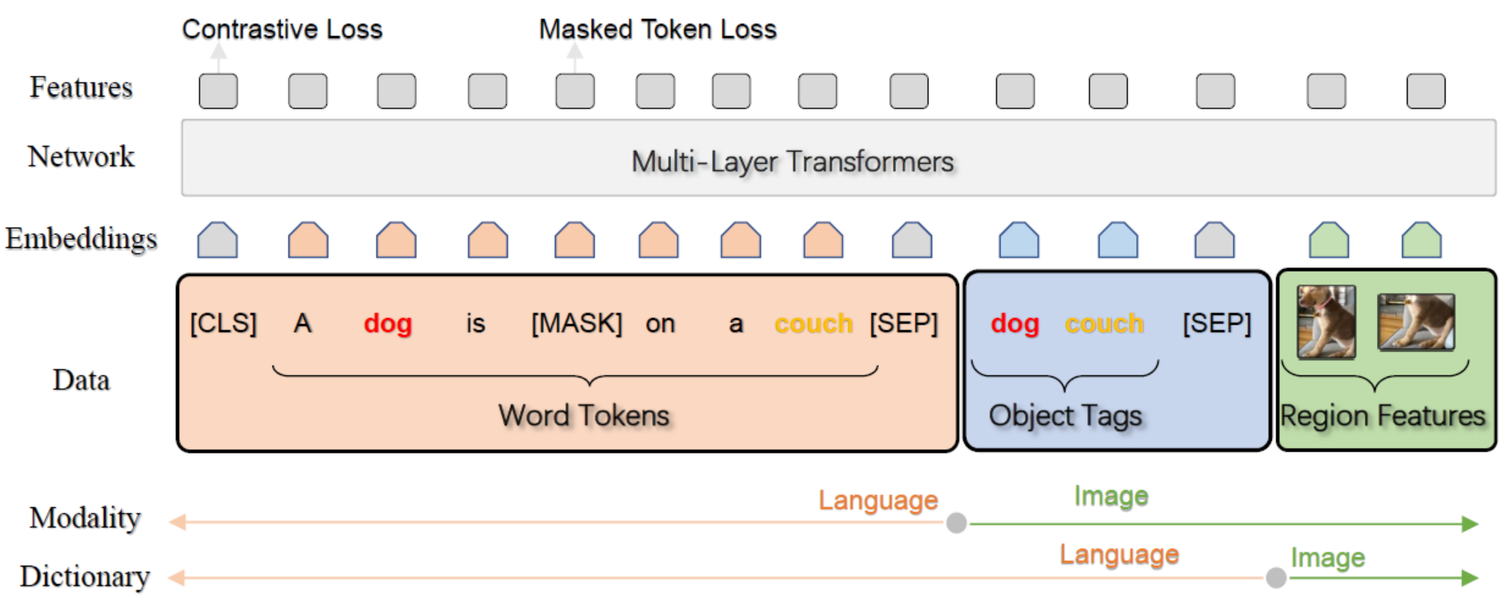
Semantics-Aligned Pre-training for V+L Tasks

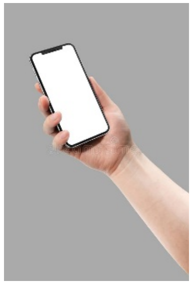
- **Oscar**: **O**bject-**S**emanti**c**s **A**ligned P**r**e-training for Vision-Language Tasks (ECCV'20)
  - Training:
    - Inputs: triplets of caption-tag-region
    - Objectives: Masked token loss for words & tags + Contrastive loss tags and others
  - Fine-tuning:
    5 vision & language tasks (image captioning, NOC, VQA, image-text retrieval, etc.)



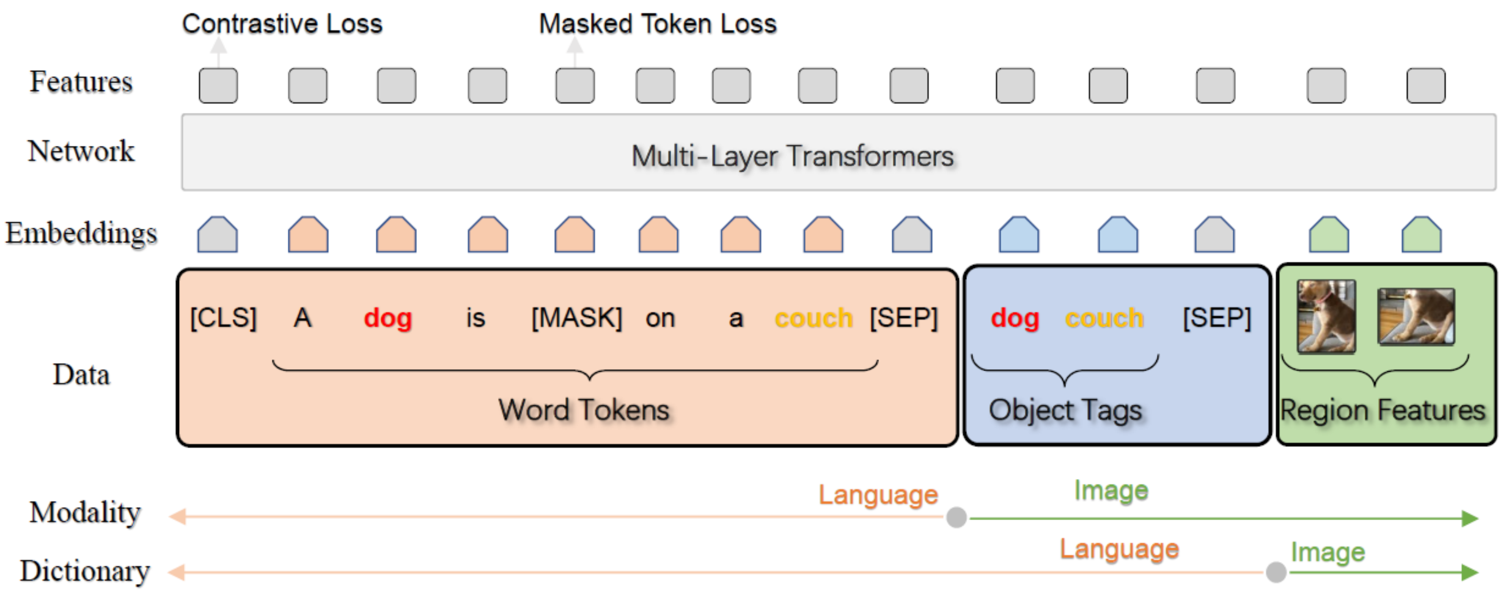
Semantics-Aligned Pre-training for V+L Tasks (cont'd)

- **Oscar**: **O**bject-**S**emanti**c**s **A**ligned P**r**e-training for Vision-Language Tasks (ECCV'20)
  - Training:
    - Inputs: triplets of word-tag-region
    - Objectives: Masked token loss for words & tags + Contrastive loss tags and others
  - Fine-tuning:
    - 5 vision & language tasks (image captioning, NOC, VQA, image-text retrieval, etc.)



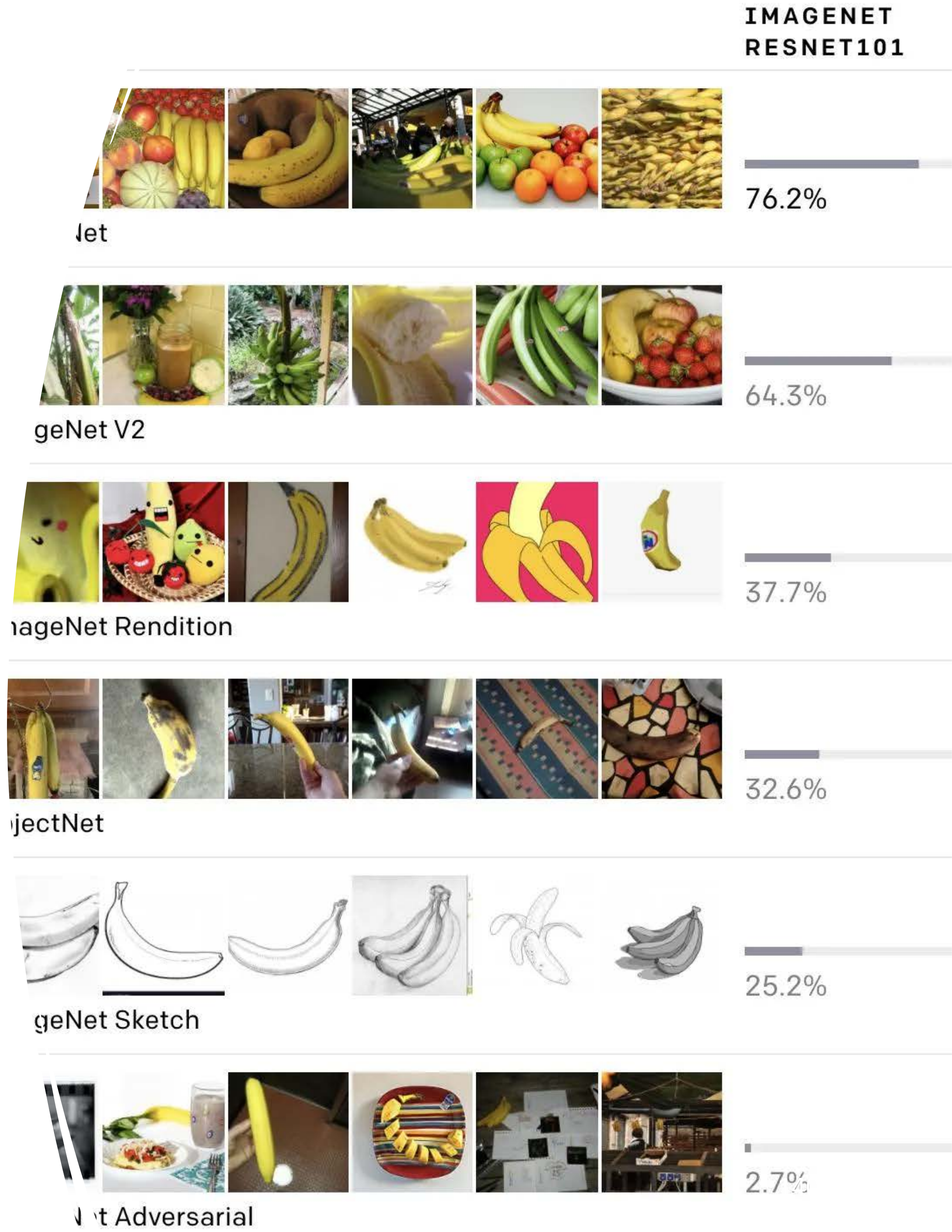
Holding an apple  or 

- **Oscar** (cont'd)
  - Fine-tuning:
    5 vision & language tasks (image captioning, NOC, VQA, image-text retrieval, etc.)
  - Take **image-text retrieval** as an example
    - Training: aligned/mis-aligned image-text pairs as positive/negative input pairs, with **[CLS]** for binary classification (1/0)
    - Inference: for either image or text retrieval, calculate classification score of **[CLS]** for the input query



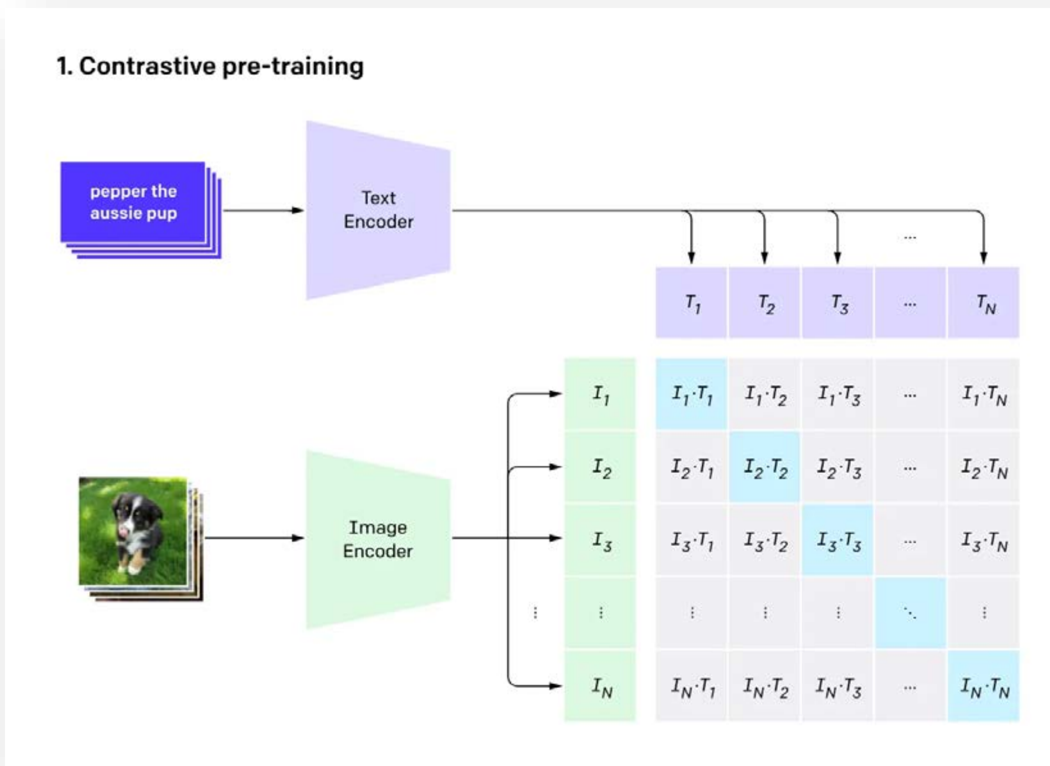
CLIP: Contrastive Language-Image Pretraining

- OpenAI, *Learning Transferable Visual Models From Natural Language Supervision*, NeurIPS WS 2021 (w/ 9000+ citations)
- Why DL/CNN not good enough?
  - Require annotated data for training image classification
  - Domain gap between closed-world and open-world domain data
  - Lack of ability for zero-shot classification



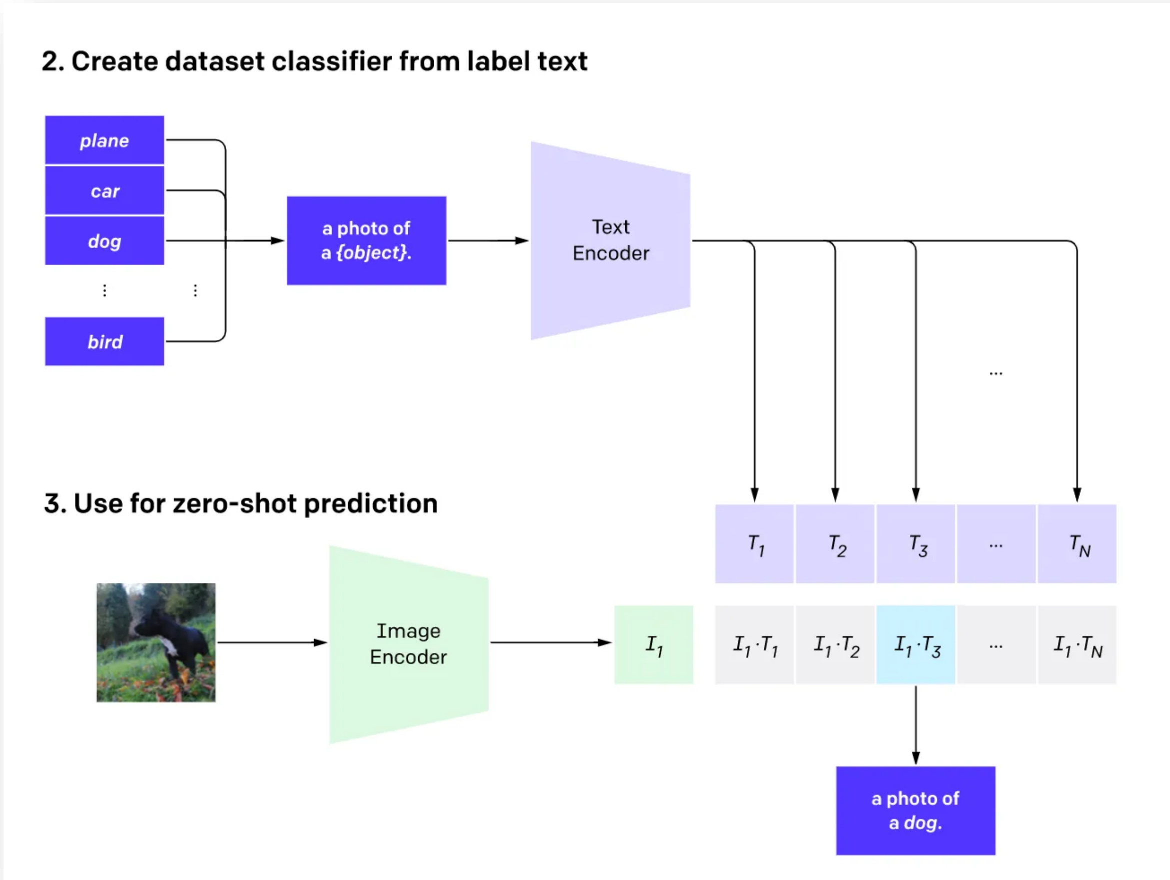
CLIP (cont'd)

- Why DL/CNN not good enough?
  - Require annotated data for training image classification
  - Domain gap between closed-world and open-world domain data
  - Lack of ability for zero-shot classification
- Motivation/Objectives
  - Cross-domain contrastive learning from large-scale image-language data



CLIP (cont'd)

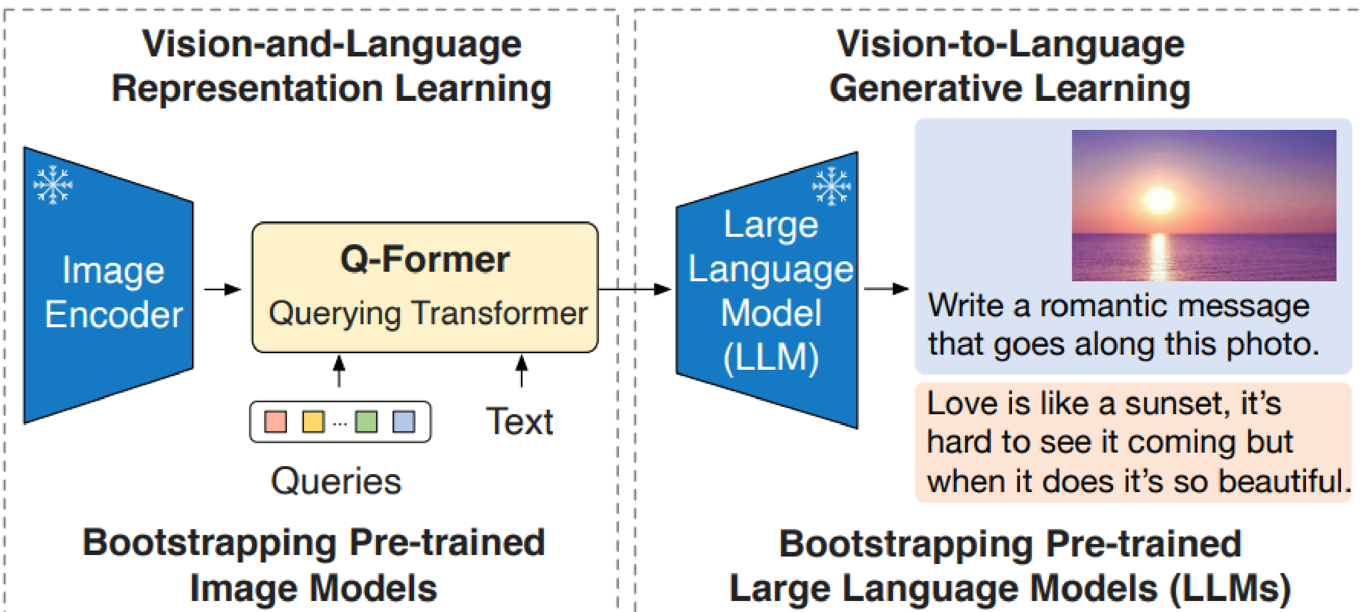
- (Zero-shot) Inference:



- Potential concerns/disadvantages?

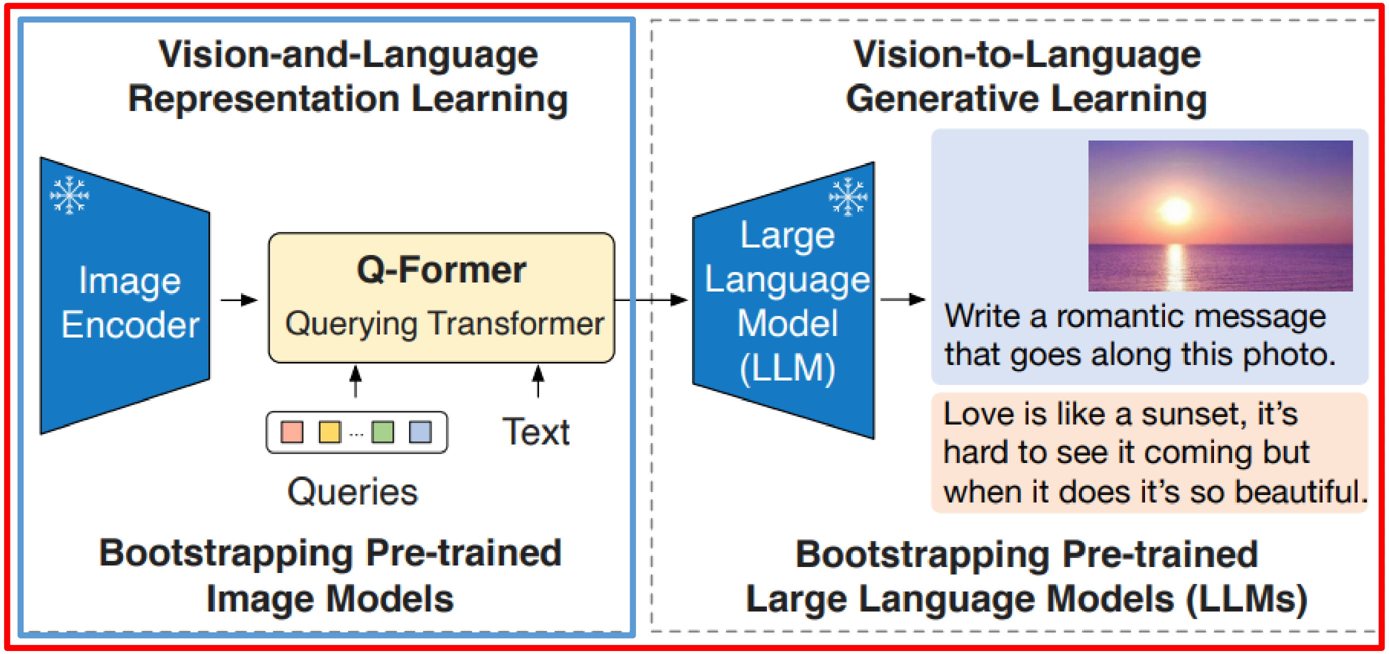
BLIP-2 (ICML'23)

- **BLIP:**
  Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation, NeurIPS 2021
- **Goal:**
  Bridge the modality gap between off-the-shelf **frozen pre-trained image encoders** and **frozen large language models** with a lightweight Querying Transformer (Q-Former).
- **Result:**
  1. SOTA performance on various downstream vision-language tasks.
  2. Zero-shot image-to-text generation that can follow natural language instructions.



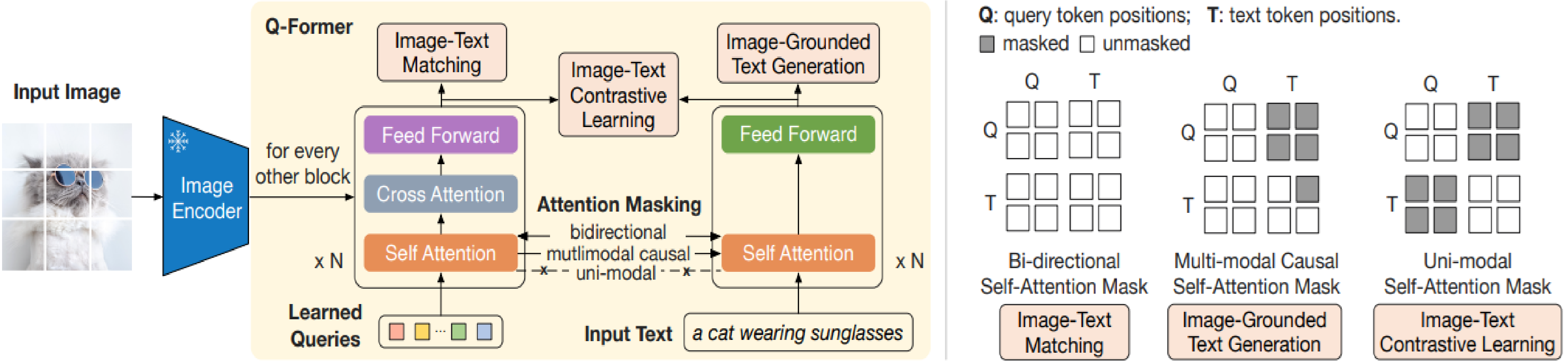
Pre-training

- **Two-stage** Pre-training
  - Stage 1:
    VL representation learning which enforces the **Q-Former** to learn **visual representation** that is most relevant to the text.
  - Stage 2:
    VL generative learning makes the output representation of **Q-Former** to be understood by **LLMs**.



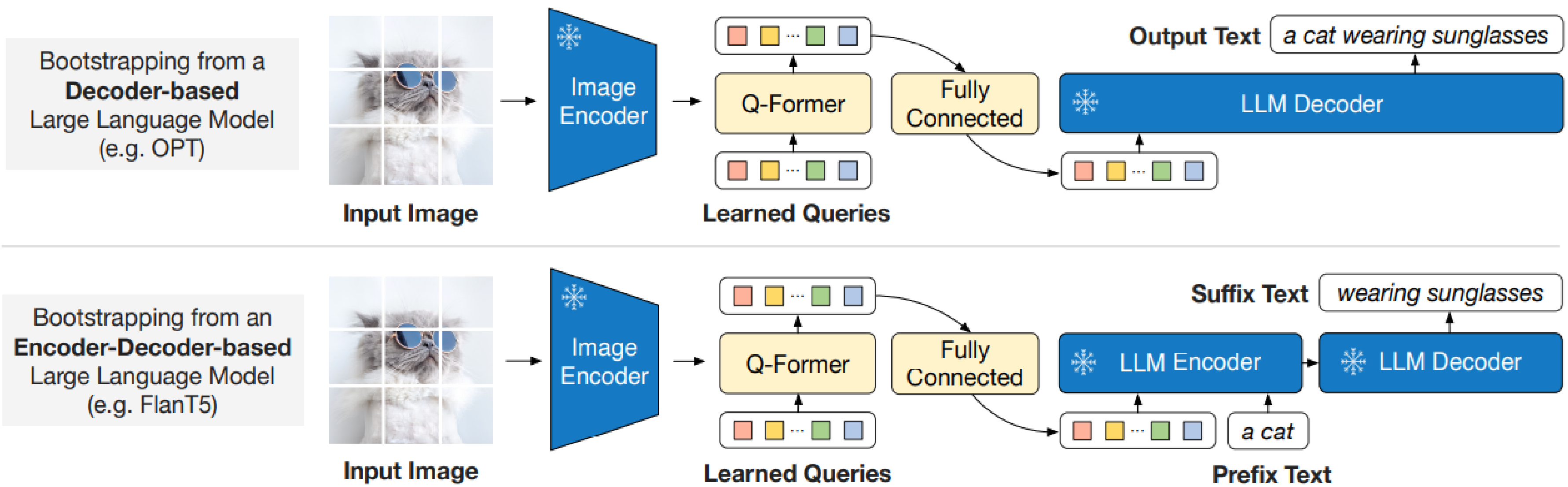
Pre-training Stage 1 - VL Representation Learning

- **Goal:**
  enforce the **Q-Former** to extract visual representation relevant to the text.
- **Method:** three pre-training tasks
  - **Image-Text Matching (ITM)**:
    for each learnable query -> linear classifier for binary decision
  - **Image-grounded Text Generation (ITG)**:
    self-attn in Q for encoder training; T->Q for image-to-text generation
  - **Image-Text Contrastive Learning (ITC)**:
    self-attn in Q/T, followed by max (sim(Q, T))



Pre-training Stage 2 - VL Generative Learning

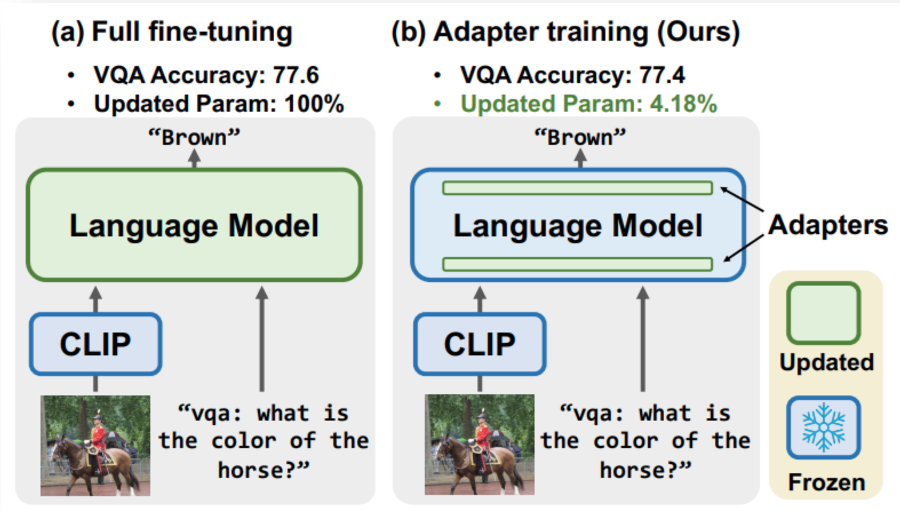
- **Goal:**
  Learning with LLM guidance
  i.e., make the output representation of **Q-Former** to be understood by **LLMs**.
- **Method:**
  pre-training with Image-grounded Text Generation (ITG)



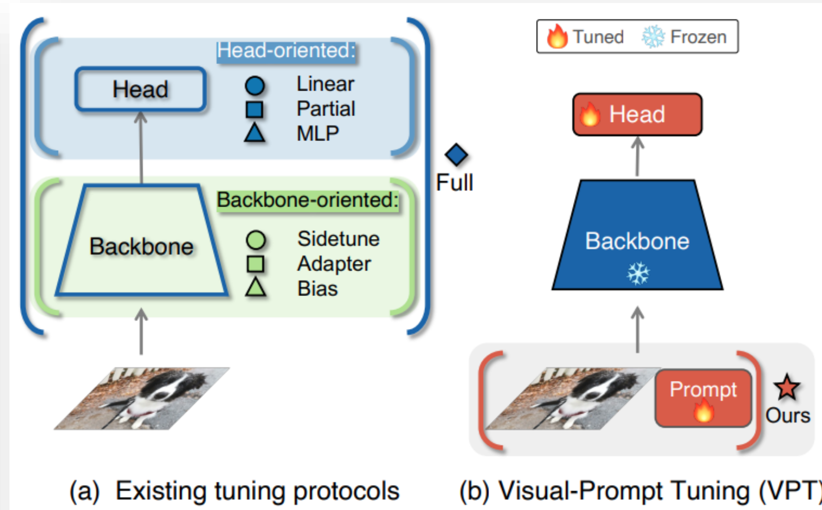
Parameter-Efficient Fine-Tuning

- **Adapter**
  - VL-ADAPTER: Parameter-Efficient Transfer Learning for Vision-and-Language Tasks (CVPR, 2022)
- **Visual Prompt Tuning**
  - Visual Prompt Tuning (ECCV, 2022)
- **LoRA**
  - LoRA: Low-Rank Adaptation of Large Language Models (ICLR, 2022)

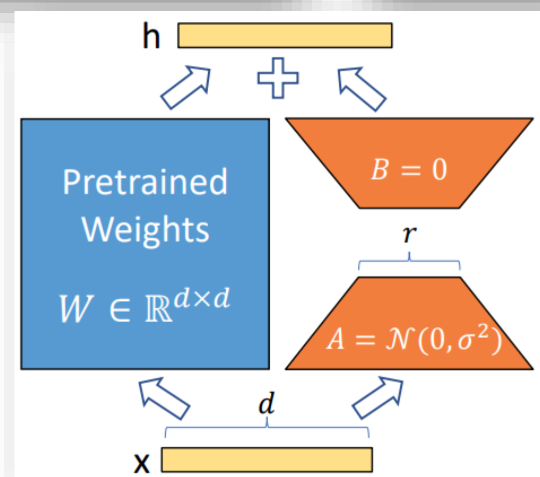
Parameter Efficient Fine Tuning



Adapter

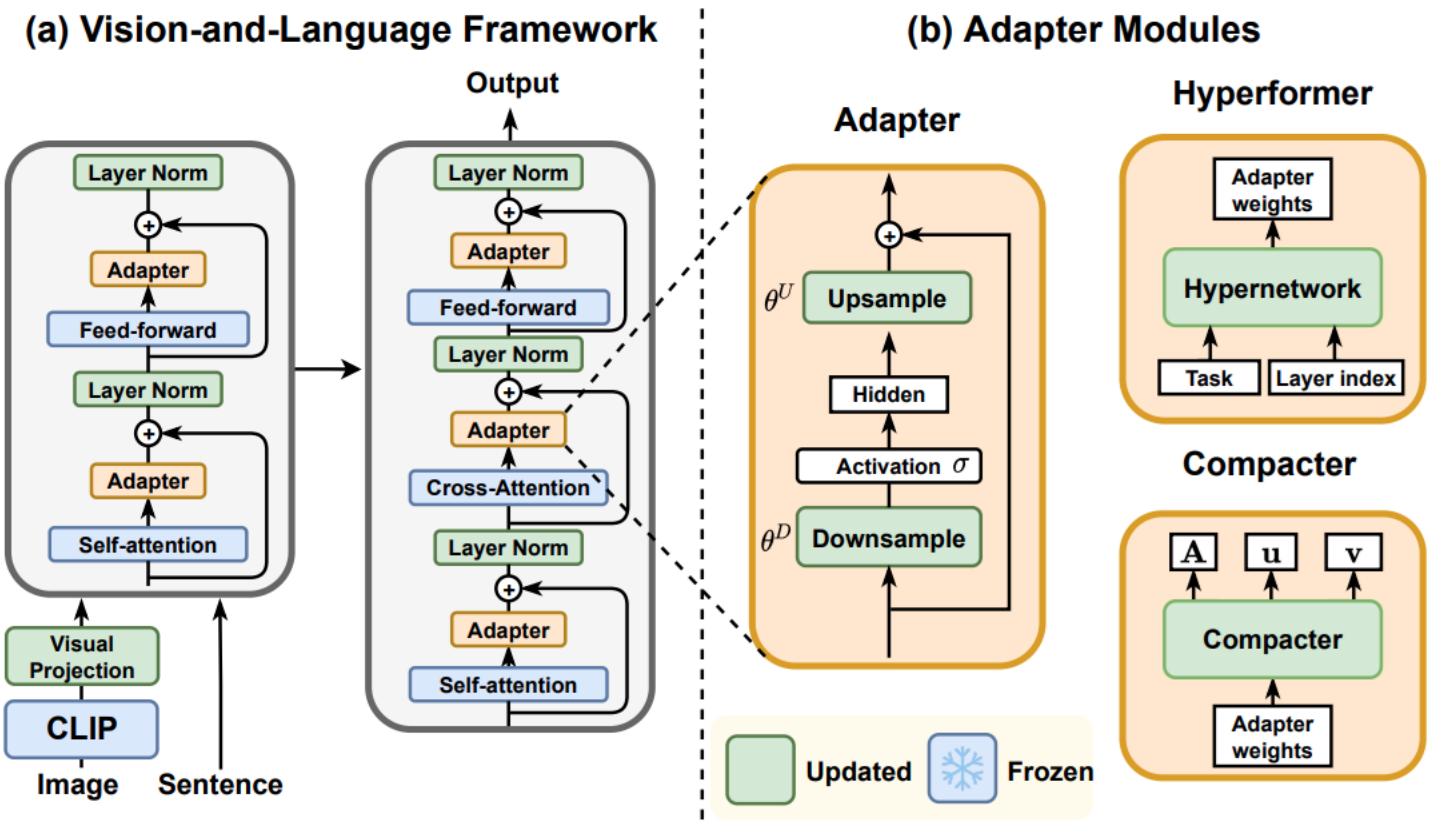


Prompt Tuning



LoRA

VL-ADAPTER: Parameter-Efficient Transfer Learning for Vision-and-Language Tasks



https://arxiv.org/abs/2112.06825

Visual Prompt Tuning

- Shallow:

$$[\mathbf{x}_1, \mathbf{Z}_1, \mathbf{E}_1] = L_1([\mathbf{x}_0, \mathbf{P}, \mathbf{E}_0]) \tag{4}$$

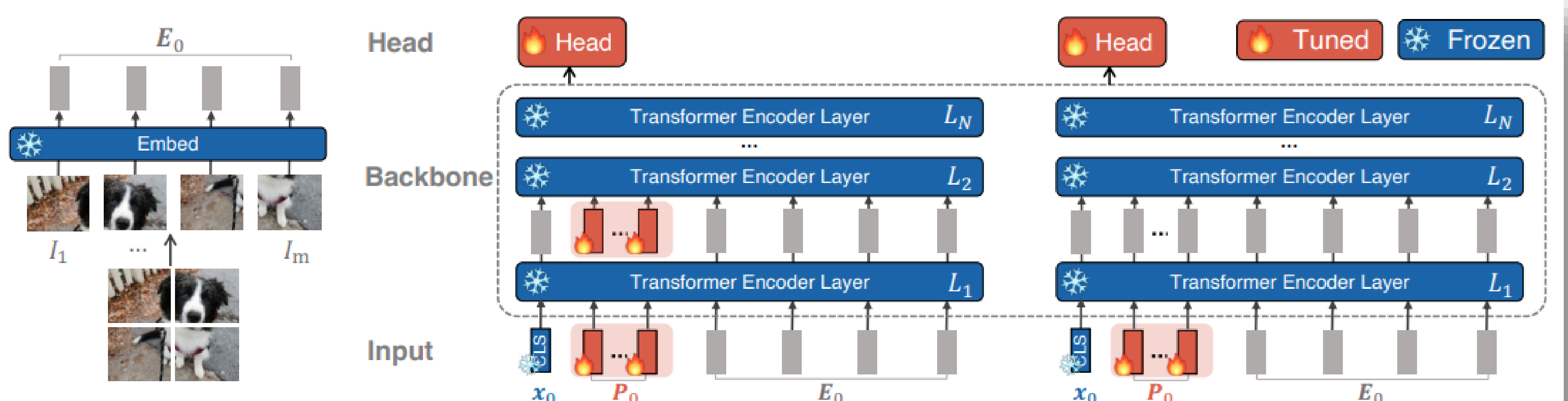
$$[\mathbf{x}_i, \mathbf{Z}_i, \mathbf{E}_i] = L_i([\mathbf{x}_{i-1}, \mathbf{Z}_{i-1}, \mathbf{E}_{i-1}]) \qquad i = 2, 3, \ldots, N \tag{5}$$

$$\mathbf{y} = \texttt{Head}(\mathbf{x}_N) \ , \tag{6}$$

- Deep:

$$[\mathbf{x}_i, \_\_, \mathbf{E}_i] = L_i([\mathbf{x}_{i-1}, \mathbf{P}_{i-1}, \mathbf{E}_{i-1}]) \qquad i = 1, 2, \ldots, N \tag{7}$$

$$\mathbf{y} = \texttt{Head}(\mathbf{x}_N) \ . \tag{8}$$



(a) Visual-Prompt Tuning: Deep (b) Visual-Prompt Tuning: Shallow

https://arxiv.org/abs/2203.12119

LoRA: Low-Rank Adaptation of Large Language Models

- Previous problems
  - Adapter Layers Introduce Inference Latency
  - Directly Optimizing the Prompt is Hard

- LoRA

$$W_0 \in \mathbb{R}^{d \times k}$$
$$W_0 + \Delta W = W_0 + BA$$
$$B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}$$
$$\text{rank } r \ll \min(d, k)$$

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

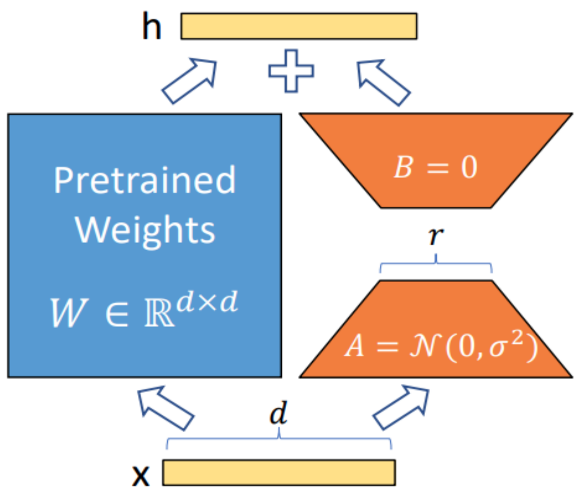


Figure 1: Our reparametrization. We only train A and B.

https://arxiv.org/abs/2106.09685

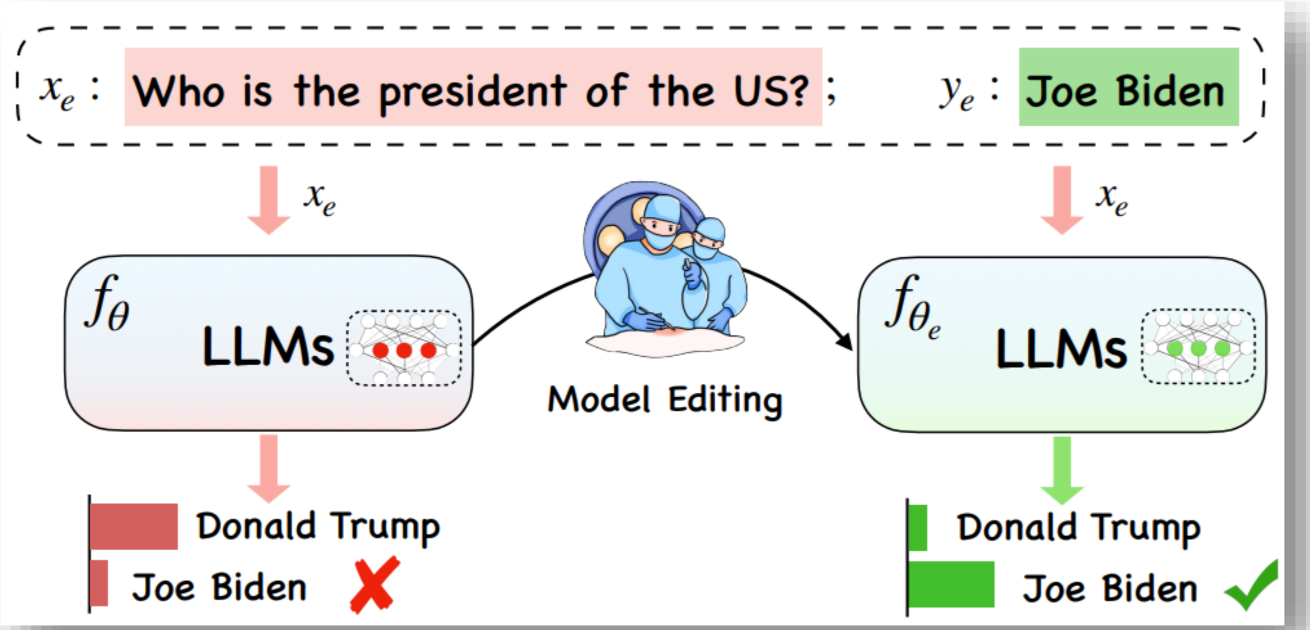
LoRA: Low-Rank Adaptation of LLMs (cont'd)

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	87.1$_{\pm.0}$	94.2$_{\pm.1}$	88.5$_{\pm1.1}$	60.8$_{\pm.4}$	93.1$_{\pm.1}$	90.2$_{\pm.0}$	71.5$_{\pm2.7}$	89.7$_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	87.3$_{\pm.1}$	94.7$_{\pm.3}$	88.4$_{\pm.1}$	62.6$_{\pm.9}$	93.0$_{\pm.2}$	90.6$_{\pm.0}$	75.9$_{\pm2.2}$	90.3$_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	87.5$_{\pm.3}$	**95.1**$_{\pm.2}$	89.7$_{\pm.7}$	63.4$_{\pm1.2}$	**93.3**$_{\pm.3}$	90.8$_{\pm.1}$	**86.6**$_{\pm.7}$	**91.5**$_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.9**$_{\pm1.2}$	**68.2**$_{\pm1.9}$	**94.9**$_{\pm.3}$	91.6$_{\pm.1}$	**87.4**$_{\pm2.5}$	**92.6**$_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	90.2$_{\pm.3}$	96.1$_{\pm.3}$	90.2$_{\pm.7}$	**68.3**$_{\pm1.0}$	**94.8**$_{\pm.2}$	**91.9**$_{\pm.1}$	83.8$_{\pm2.9}$	92.1$_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	**90.5**$_{\pm.3}$	**96.6**$_{\pm.2}$	89.7$_{\pm1.2}$	67.8$_{\pm2.5}$	**94.8**$_{\pm.3}$	91.7$_{\pm.2}$	80.1$_{\pm2.9}$	91.9$_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	89.9$_{\pm.5}$	96.2$_{\pm.3}$	88.7$_{\pm2.9}$	66.5$_{\pm4.4}$	94.7$_{\pm.2}$	92.1$_{\pm.1}$	83.4$_{\pm1.1}$	91.0$_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	90.3$_{\pm.3}$	96.3$_{\pm.5}$	87.7$_{\pm1.7}$	66.3$_{\pm2.0}$	94.7$_{\pm.2}$	91.5$_{\pm.1}$	72.9$_{\pm2.9}$	91.5$_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.2**$_{\pm1.0}$	68.2$_{\pm1.9}$	**94.8**$_{\pm.3}$	91.6$_{\pm.2}$	**85.2**$_{\pm1.1}$	**92.3**$_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	**91.9**$_{\pm.2}$	96.9$_{\pm.2}$	**92.6**$_{\pm.6}$	**72.4**$_{\pm1.1}$	**96.0**$_{\pm.1}$	**92.9**$_{\pm.1}$	**94.9**$_{\pm.4}$	**93.0**$_{\pm.2}$	**91.3**

Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($\text{Adapter}^{\text{H}}$)	11.09M	67.3$_{\pm.6}$	8.50$_{\pm.07}$	46.0$_{\pm.2}$	70.7$_{\pm.2}$	2.44$_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	**70.4**$_{\pm.1}$	**8.85**$_{\pm.02}$	**46.8**$_{\pm.2}$	**71.8**$_{\pm.1}$	**2.53**$_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($\text{Adapter}^{\text{L}}$)	0.88M	69.1$_{\pm.1}$	8.68$_{\pm.03}$	46.3$_{\pm.0}$	71.4$_{\pm.2}$	**2.49**$_{\pm.0}$
GPT-2 L ($\text{Adapter}^{\text{L}}$)	23.00M	68.9$_{\pm.3}$	8.70$_{\pm.04}$	46.1$_{\pm.1}$	71.3$_{\pm.2}$	2.45$_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	**70.4**$_{\pm.1}$	**8.89**$_{\pm.02}$	**46.8**$_{\pm.2}$	**72.0**$_{\pm.2}$	2.47$_{\pm.02}$

Adv. Topic: Knowledge Editing

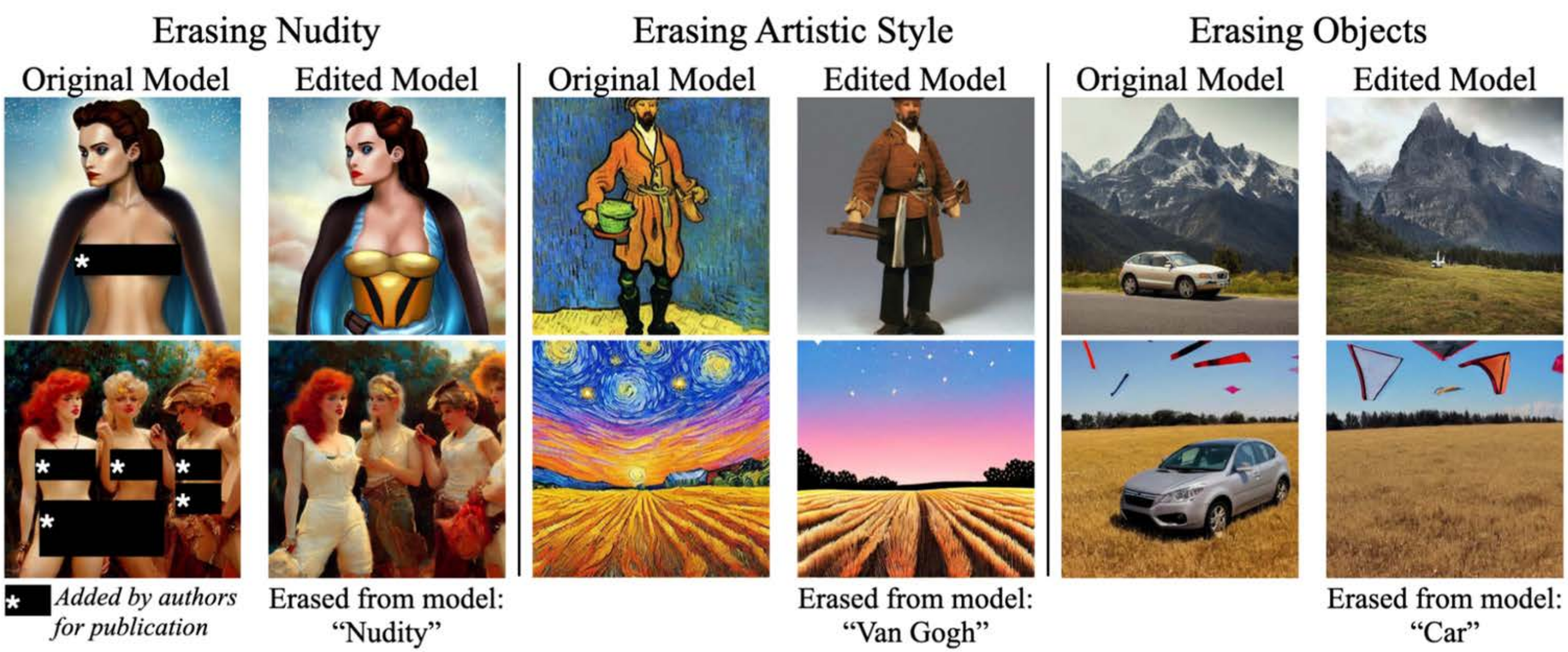
- **Motivation:**
  Knowledge updates everyday while retraining LLMs is expensive.
- **Goal**:
  Propose more efficient & effective solutions to update knowledge in LLM



Editing Large Language Models: Problems, Methods, and Opportunities (EMNLP’23)

Adv. Topic: Unlearning

- **Goal:**
  Erase the undesirable visual concepts from Diffusion Models.
  Concepts can be abstractive concept, artistic style, object, or personality.
- **Method:**
  Diffusion Model Fine-tuning.



What We (Try to) Cover Today…

- **Learning-based Computer Vision**
  - From Linear to Non-Linear Classifiers
- **Start of Deep Learning for Computer Vision**
  - Convolutional Neural Networks
  - SSL, Segmentation & Detection
- **Generative Models**
  - AE, VAE, GAN, & Diffusion Models
- **Sequence-to-Sequence Learning**
  - Attention is All You Need: Transformer
- **Vision & Language Foundation Models**
  - Image-to-Text vs. Text-to-Image
  - Parameter-Efficient Fine-tuning
- **Lots of research topics we haven't covered…**
  - 3D vision
  - Video-based synthesis & analysis, etc.

Feel free to reach me at ycwang@ntu.edu.tw