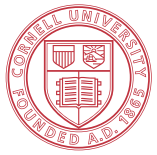
ECE 6775
High-Level Digital Design Automation
Fall 2025

Introduction to Neural Networks

Yixiao Du, Jordan Dotzel, Ritchie Zhao, Zhiru Zhang
School of Electrical and Computer Engineering



Cornell University

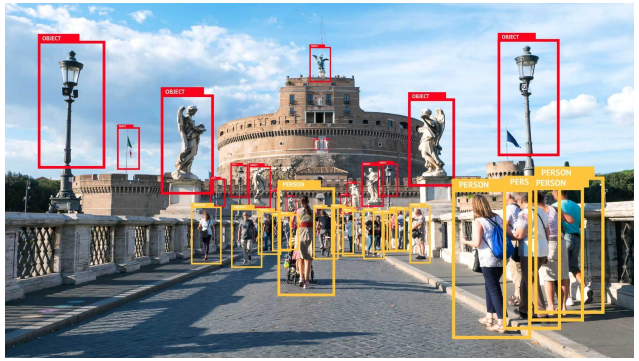
CSL

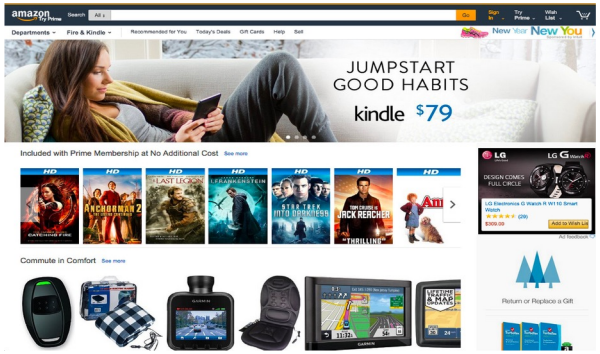
Announcement

- Instructor OH cancelled today (Oct 2)

Rise of Deep Neural Networks (DNNs)

- DNNs have revolutionized information technology
  - computer vision, e-commerce, finance, game AI, healthcare, machine transcription & translation, robots, web search, and many more (to come)



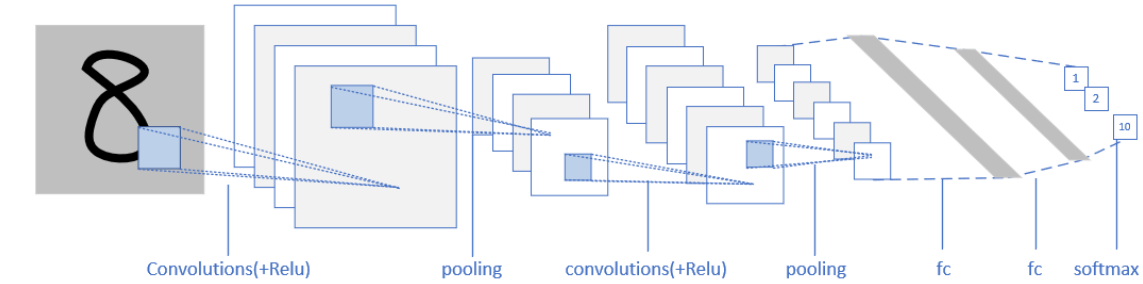


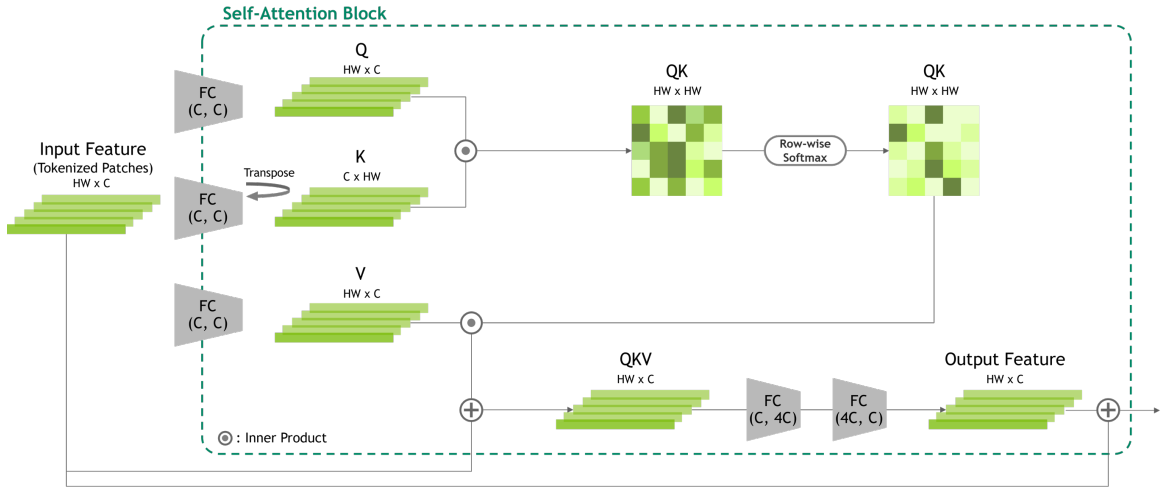
A Brief History

- **1940's – 1950's:** First artificial neural networks proposed based on biological structures in the human visual cortex
- **1980's – 2000's:** Neural networks (NNs) considered inferior to other simpler algorithms (e.g., SVM)
- **Mid 2000's**: NN research considered "dead", machine learning conferences outright reject most NN papers
- **2010 – 2012**: DNNs begin winning large-scale image and document classification contests
- **2012 – Now**: DNNs prove themselves in many industrial applications (web search, translation, image analysis)

Neural Network (NN) in a Nutshell

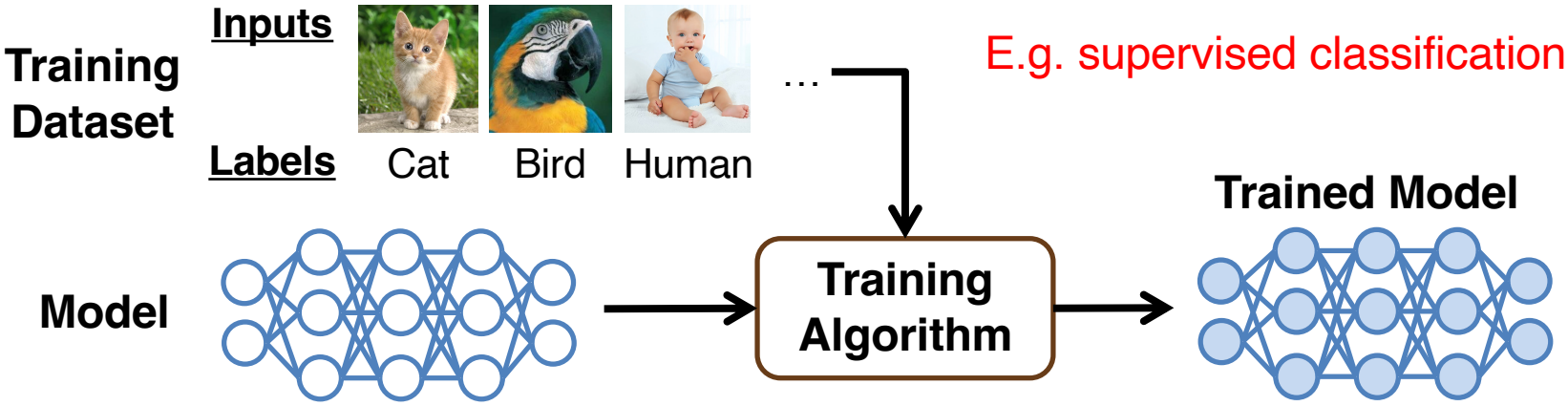
- NN **learns a function** from a **large volume of data**
- The learned function encodes a hierarchy of **features** with a stack of **layers**
  - Fully connected, convolutional, self-attention, …



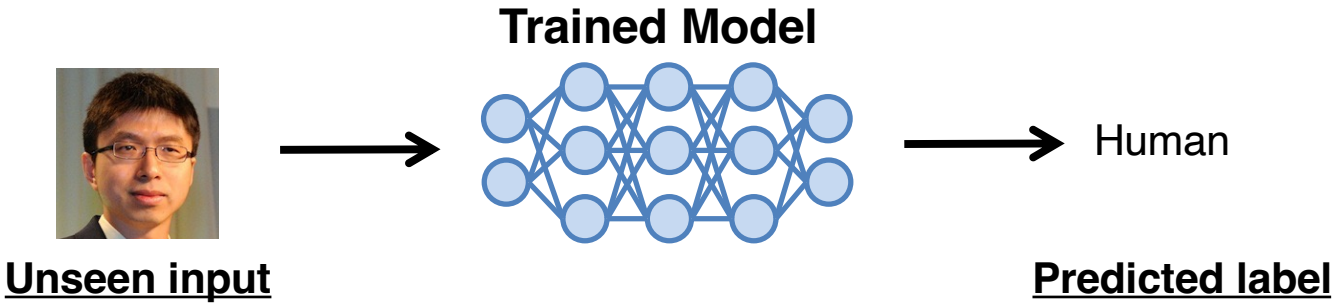


NN Training and Inference

- **Training:** the process an NN model learn to accomplish a **specific task** from a **predetermined dataset**



- **Inference:** the use of a trained NN model to perform the **specific task** on **unseen data**



Part 1

FROM THE PERCEPTRON TO DEEP NEURAL NETWORKS

Artificial Neuron

- The simplest possible neural network contains only one "neuron", which is described by the following equation:

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

w_i = weights
b = bias
σ = activation function

- When σ is the unit step (or sign) function, we have a **perceptron** *
  - Invented in 1957 by Frank Rosenblatt

* Perceptron is often used as a synonym for artificial neuron, where σ could be any activation function

Activation Function

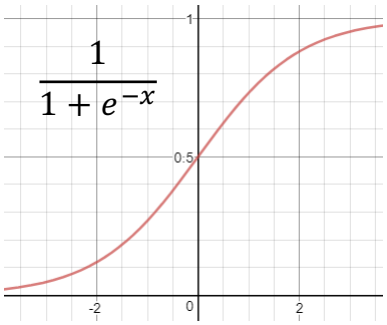
- The activation function σ is non-linear

Unit Step

Hard Yes/No decision

Used in the initial perceptrons

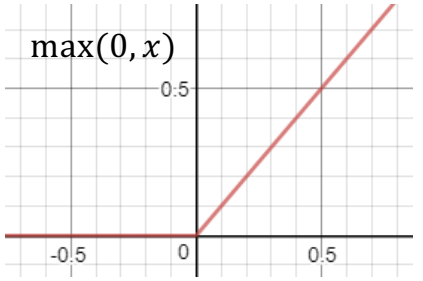
Sigmoid



Soft probability

Used in early neural nets

ReLU

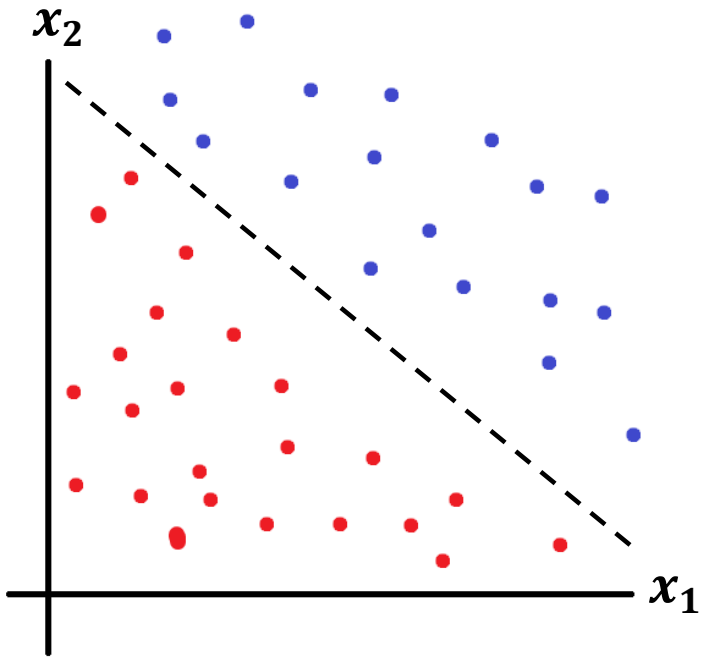


Makes deep networks easier to train

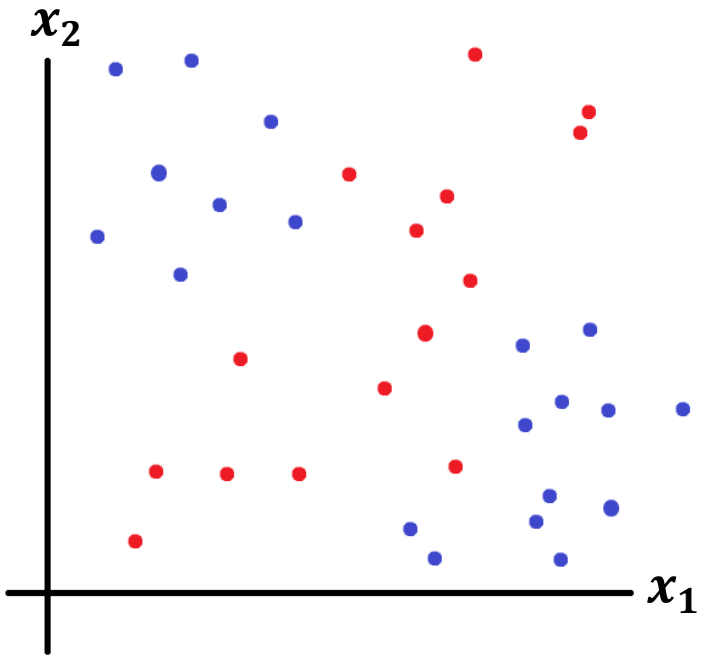
Used in modern deep nets

Decision Boundary

- $y = \sigma(\sum_{i=1}^{n} w_i x_i + b)$ (think of it as y = wx + b) defines a **linear decision boundary**
- Many datasets are not linearly separable!



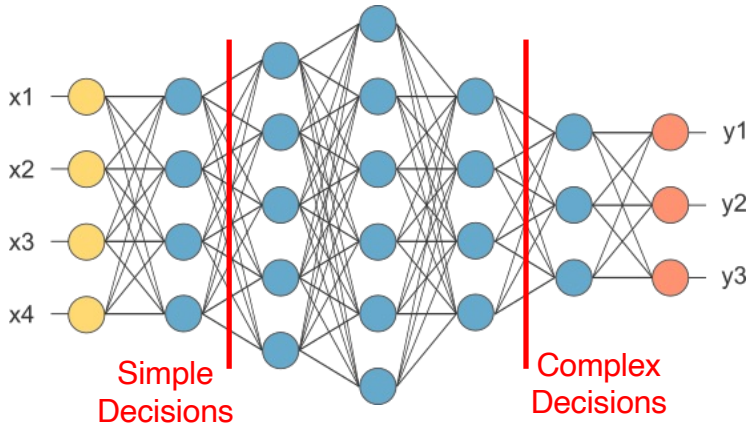
Linearly Separable
Possible to be classified by a single neuron



Non-Linearly Separable
Impossible to be classified by a single neuron

Combining Neurons

- A single neuron can only make a simple decision
- Feeding neurons into each other allows a neural network to learn **complex decision boundaries**

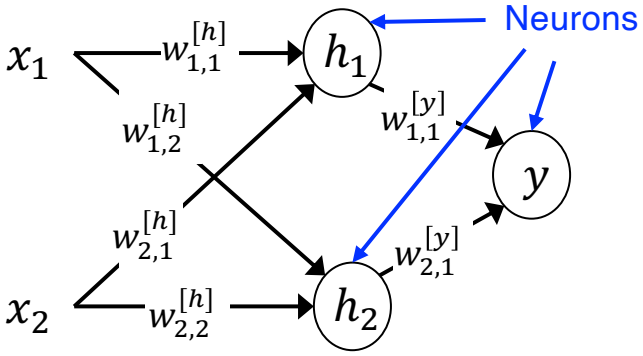


Source: http://www.opennn.net/

Multi-Layer Perceptron (MLP)

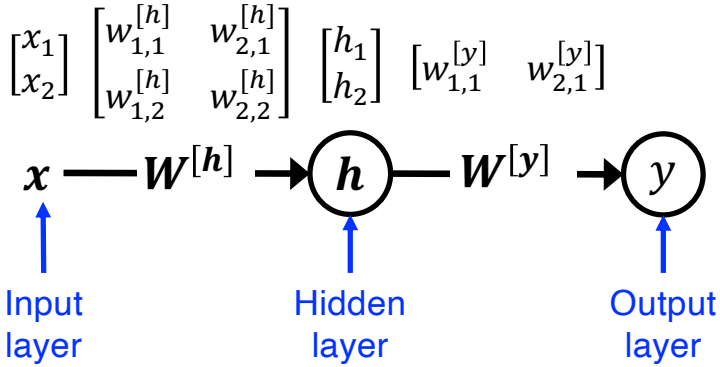
- MLP is a fully connected class of feedforward artificial neural network (ANN)
  - An MLP model consists of multiple **layers** of neurons
  - Often referred to as "vanilla" ANNs, especially with a single hidden layer

MLP



$$h_1 = \sigma\left(\sum_{i=1}^{2} w_{i,1}^{[h]}\, x_i + b_1^{[h]}\right)$$
$$h_2 = \sigma\left(\sum_{i=1}^{2} w_{i,2}^{[h]}\, x_i + b_2^{[h]}\right)$$

A vectorized (tensorized) view of MLP



$$\boldsymbol{h} = \sigma\left(\boldsymbol{W}^{[h]}\boldsymbol{x} + \boldsymbol{b}^{[h]}\right)$$

Deep Neural Network (DNN)

- MLPs are neural networks with at least three layers, while DNNs typically have even more (hidden) layers

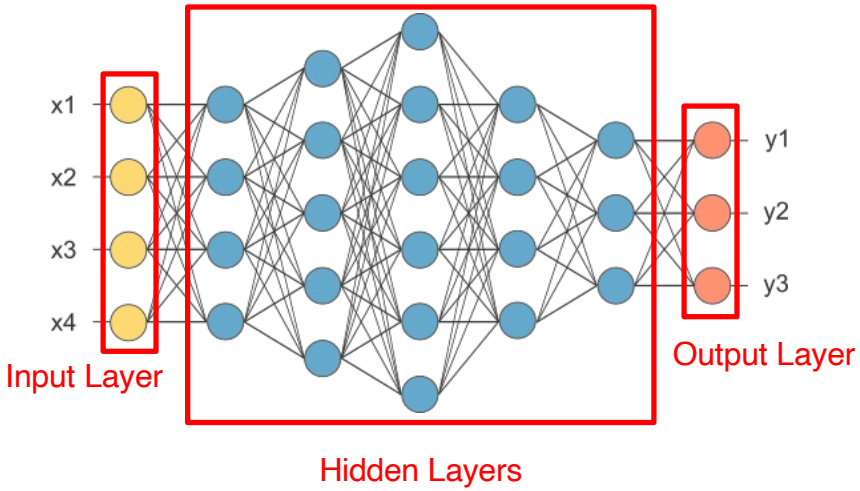
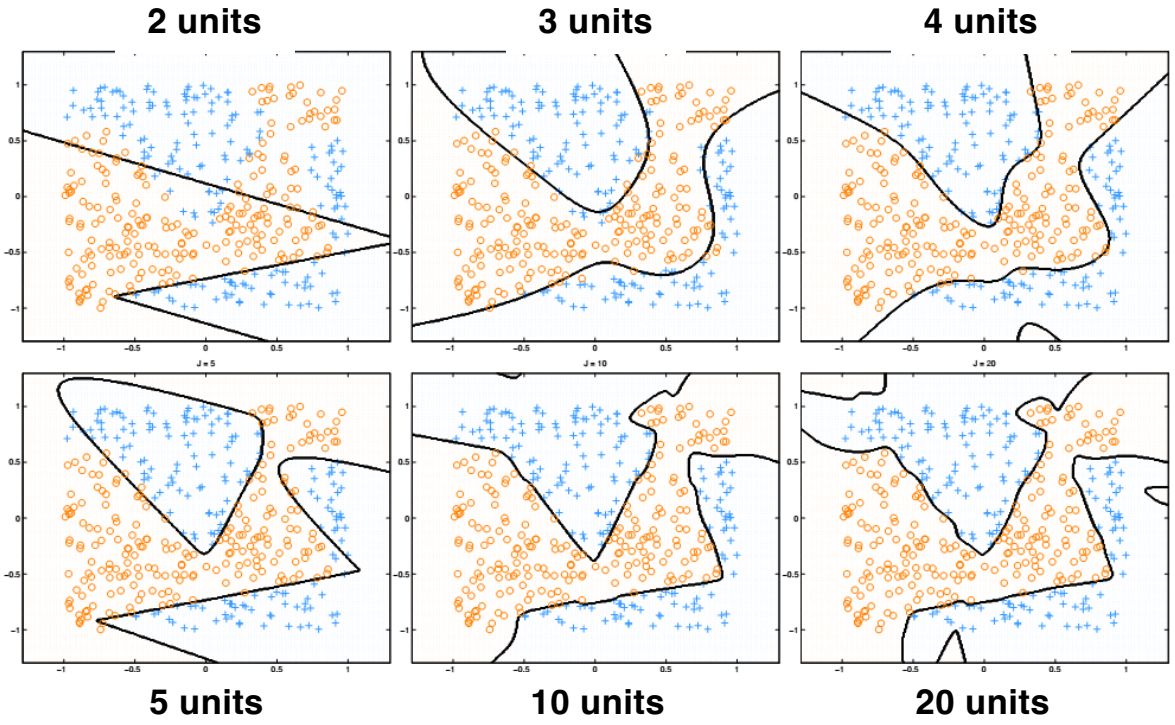


Image credit: http://www.opennn.net/

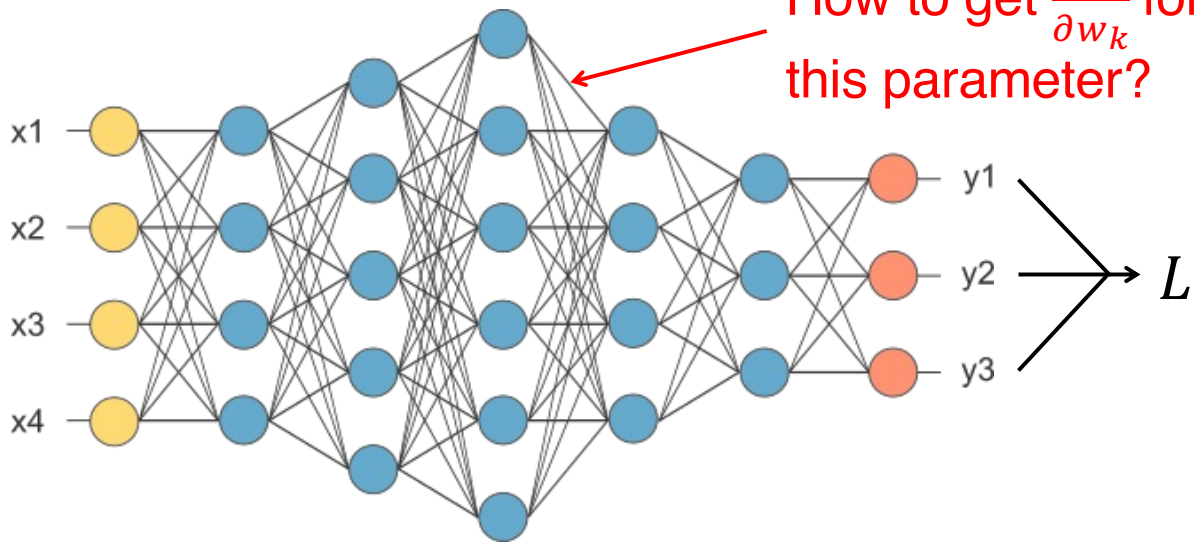
Complex Decision Boundaries



Source: https://www.carl-olsson.com/fall-semester-2013/

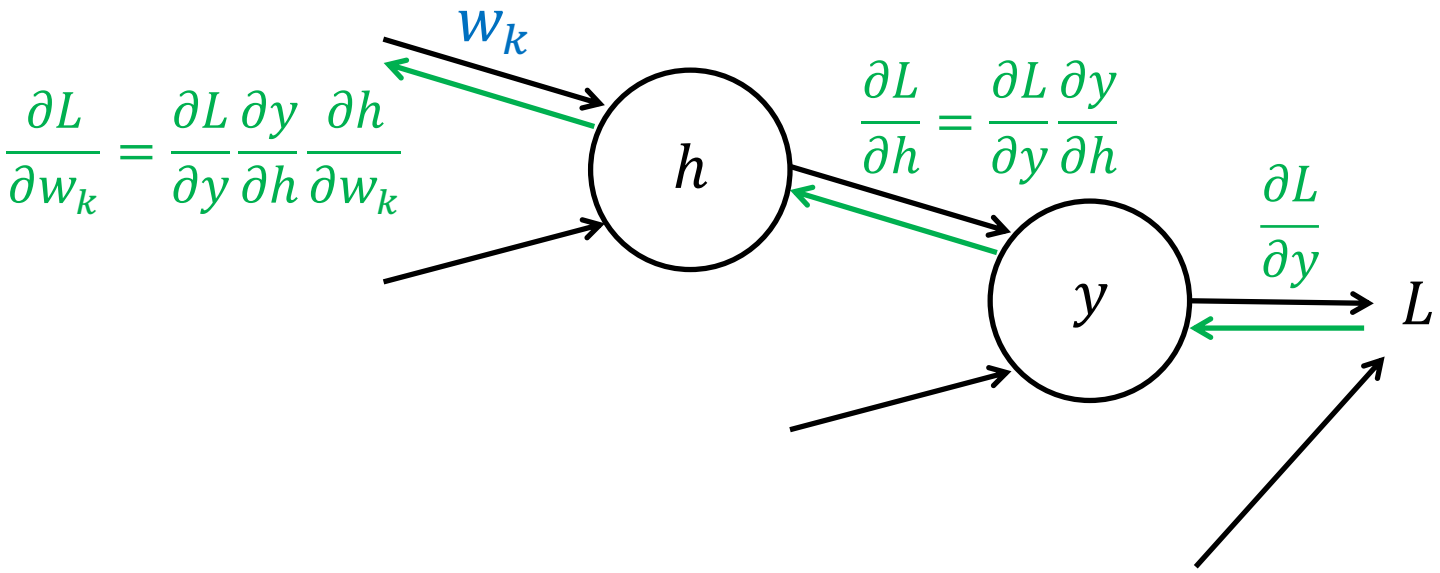
Train a Deep Neural Network

- Define a **loss function** L that increases as the DNN makes more errors
- Run the **gradient descent** method:
  - Feed the training dataset through the network
  - Compute the **gradient** of the loss function w.r.t. parameters $\frac{\partial L}{\partial w_k}$
  - Update each parameter by $w_k \leftarrow w_k - \eta \frac{\partial L}{\partial w_k}$ (learning rate)
  - Repeat until convergence



Backward Propagation

- Use the **chain rule** from calculus to propagate the gradients backwards through the network



Stochastic Gradient Descent

$$w_k \leftarrow w_k - \eta \frac{\partial L}{\partial w_k}$$

- L must be computed over the entire training set, which can be millions of samples!

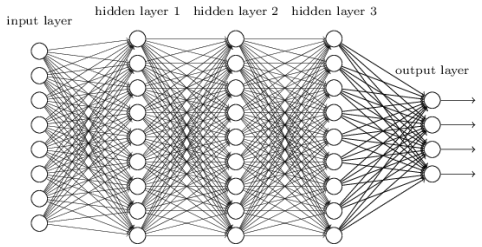
- **Stochastic Gradient Descent:**
  - At each step, only compute L for a **minibatch** (a few samples randomly taken from the training set)
  - SGD is faster and **more accurate** than GD for DNNs!

Part 2

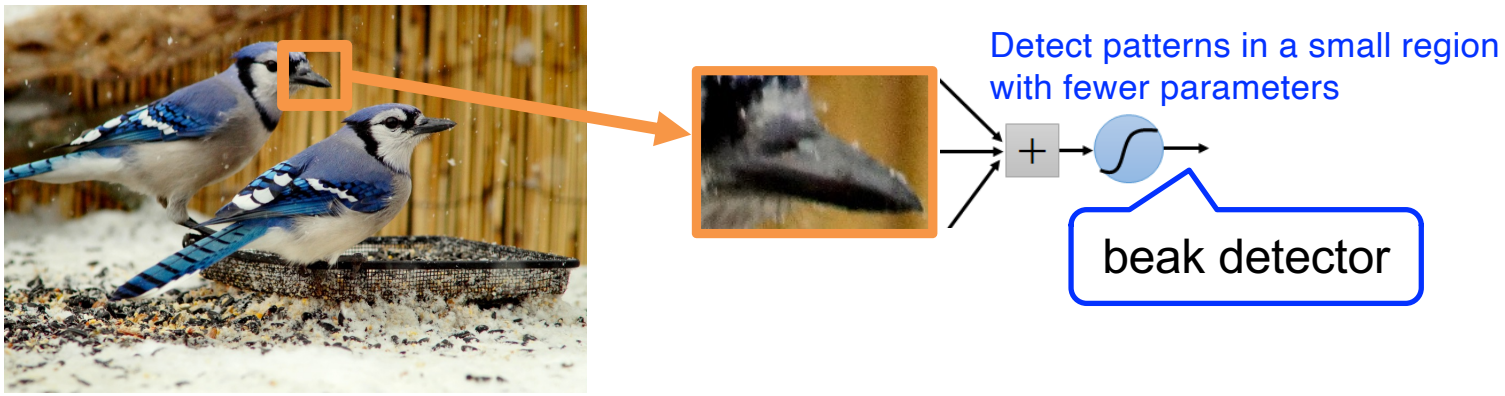
CONVOLUTIONAL NEURAL NETWORKS

Neural Networks for Images

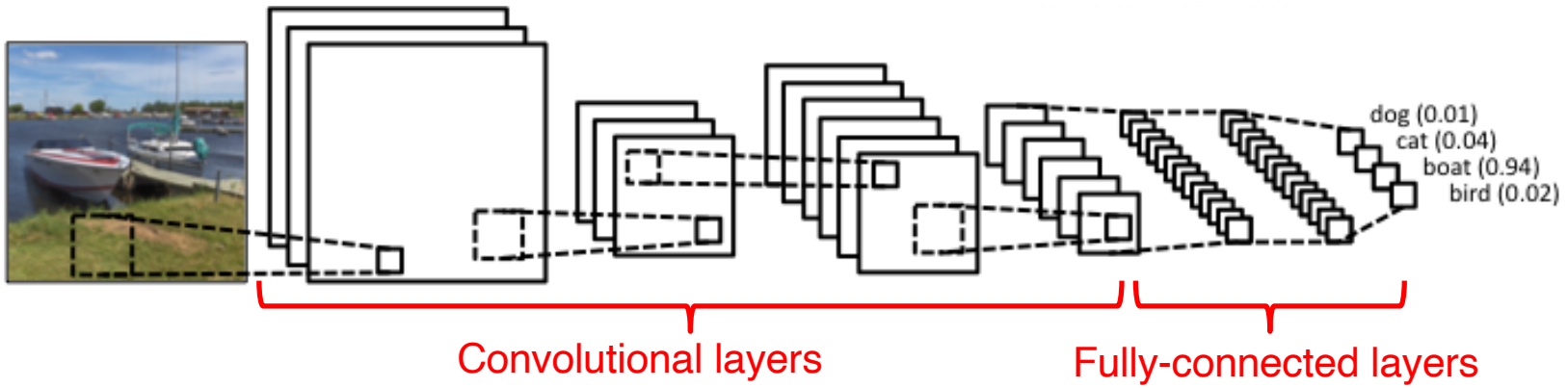
- So far, we've seen neural networks built from **fully-connected layers**



- Do we really need all the edges for learning an image?
  - Important patterns are typically much smaller than the whole image
  - Images are also shift-invariant (e.g., a bird is a bird even when shifted)



Convolutional Neural Network (CNN)



- **Front**: convolutional layers learn visual features
- Feature maps get down-sampled through the network
  - "Zoomed out" to perceive larger areas of the image
- **Back**: fully-connected layers perform classification using the visual features

Image credit: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Learning Complex Features with CNNs

- Deep CNNs combine simple features into complex patterns
  - Early conv layers = edges, textures, ridges
  - Later conv layers = eyes, noses, mouths

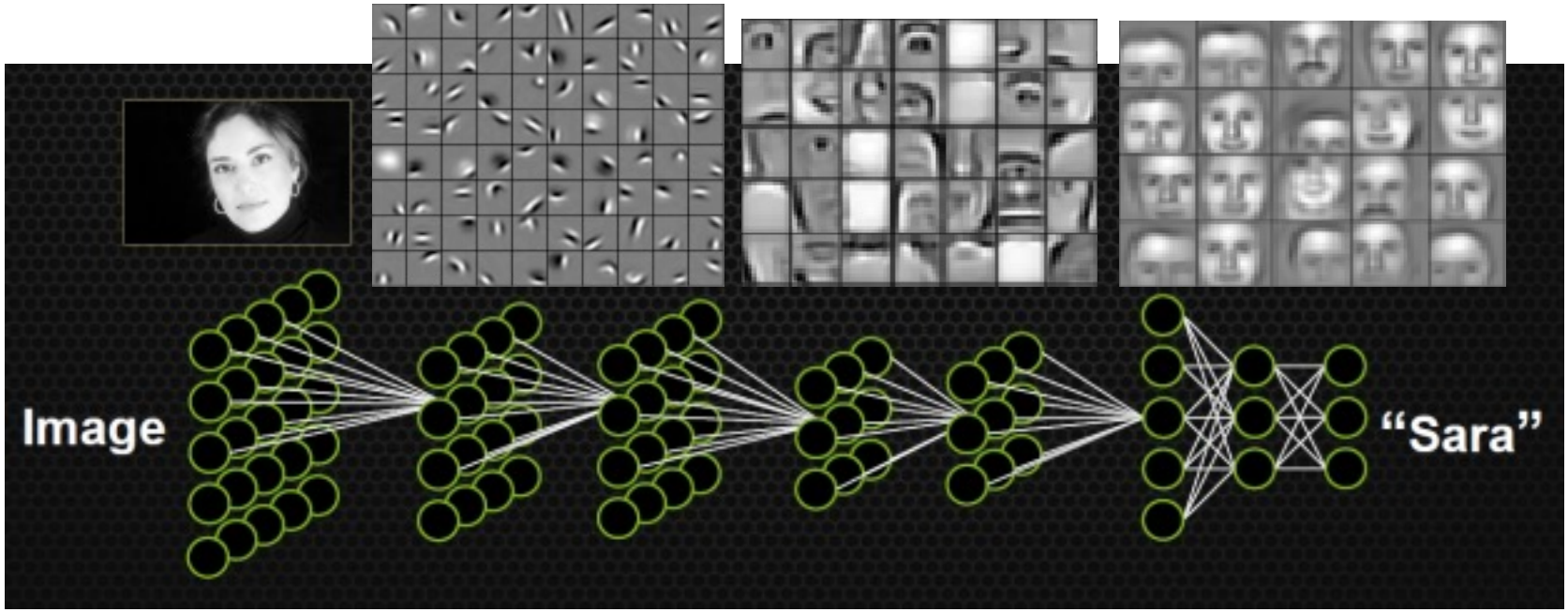
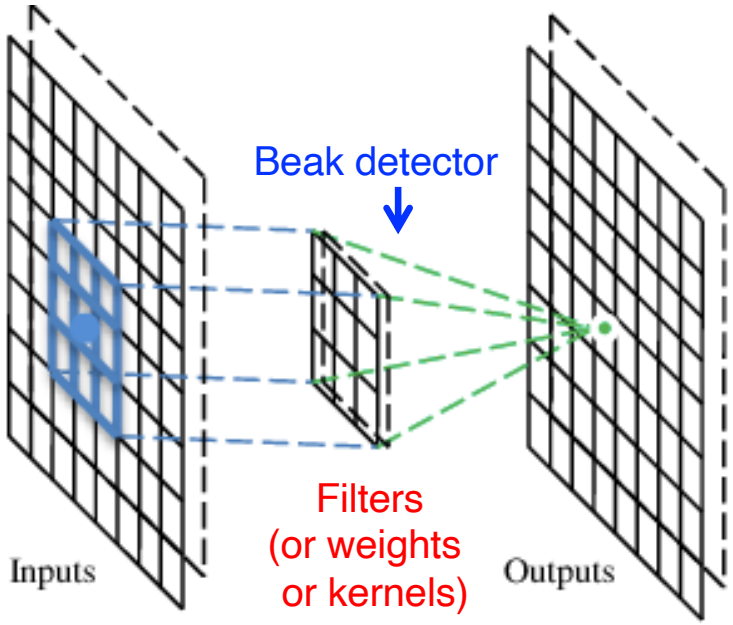


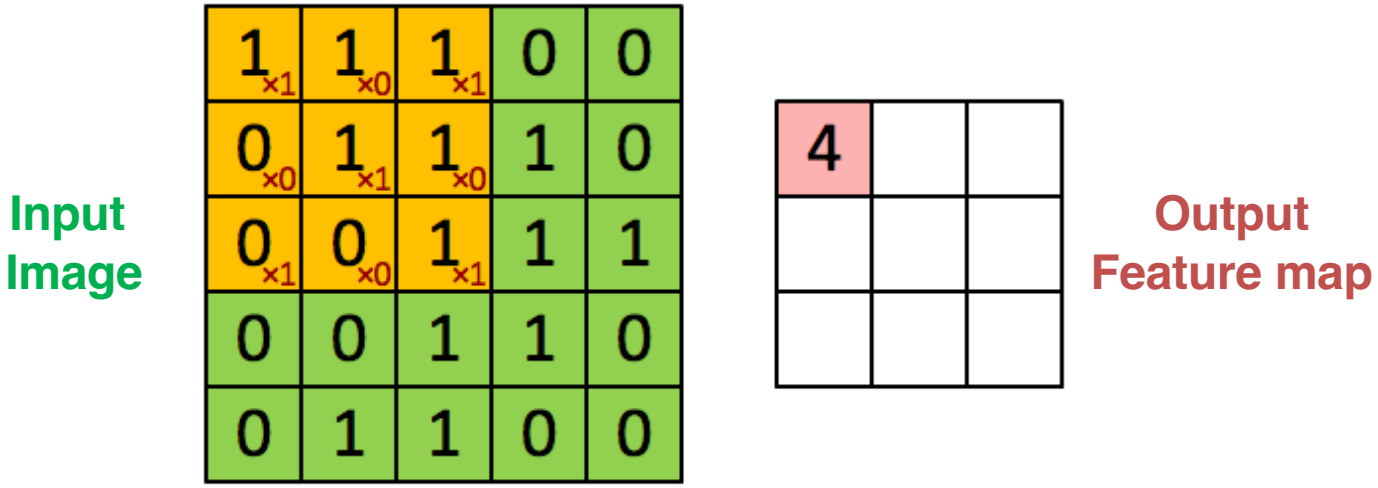
Image credit: https://devblogs.nvidia.com/parallelforall/accelerate-machine-learning-cudnn-deep-neural-network-library/; H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks", CACM Oct 2011

A Convolutional (conv) Layer

- A CNN stacks multiple conv layers (and some other layers)
- A conv layer has a set of learnable filters that perform convolution operation



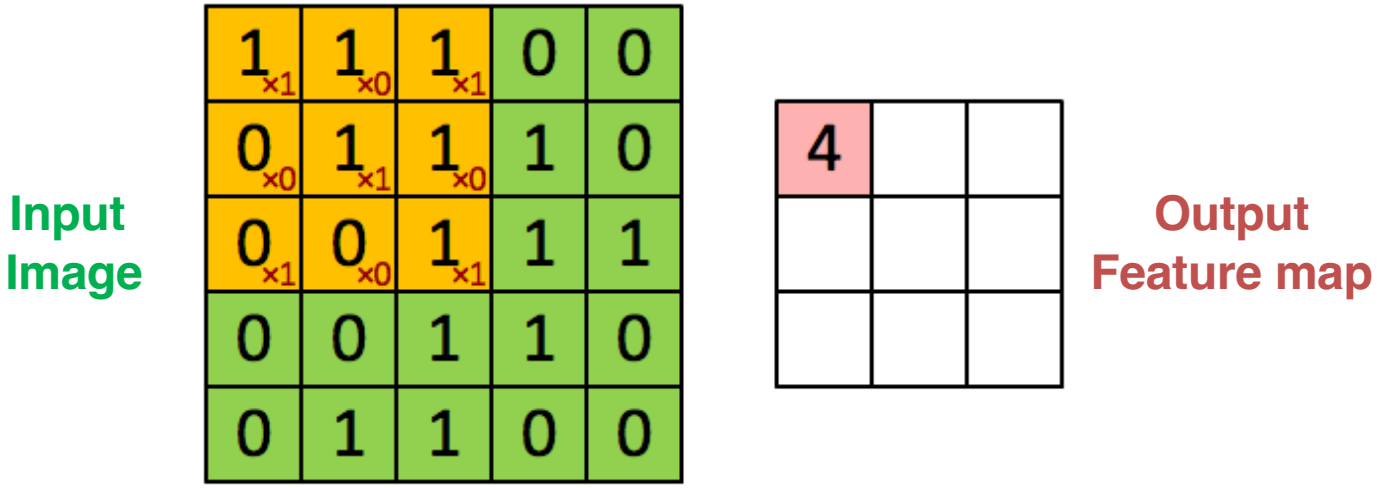
The Convolutional Filter



- Each neuron learns a **weight filter** and **convolves** the filter over the image
- Each neuron outputs a 2D **feature map** (basically an image of features)

Image credit: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

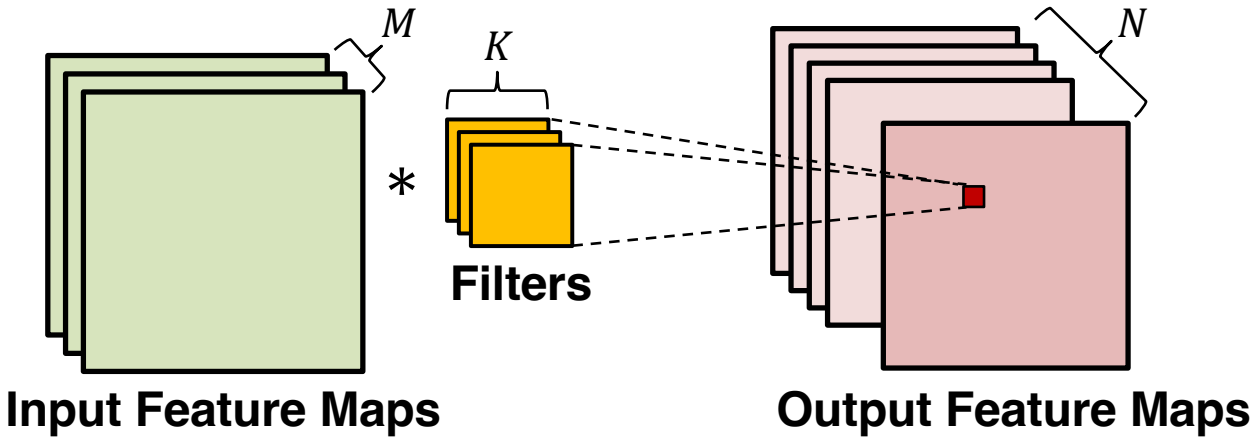
The Convolutional Filter



- Each point in the feature map encodes both a decision and its spatial location
- **Detects the pattern anywhere in the image!**

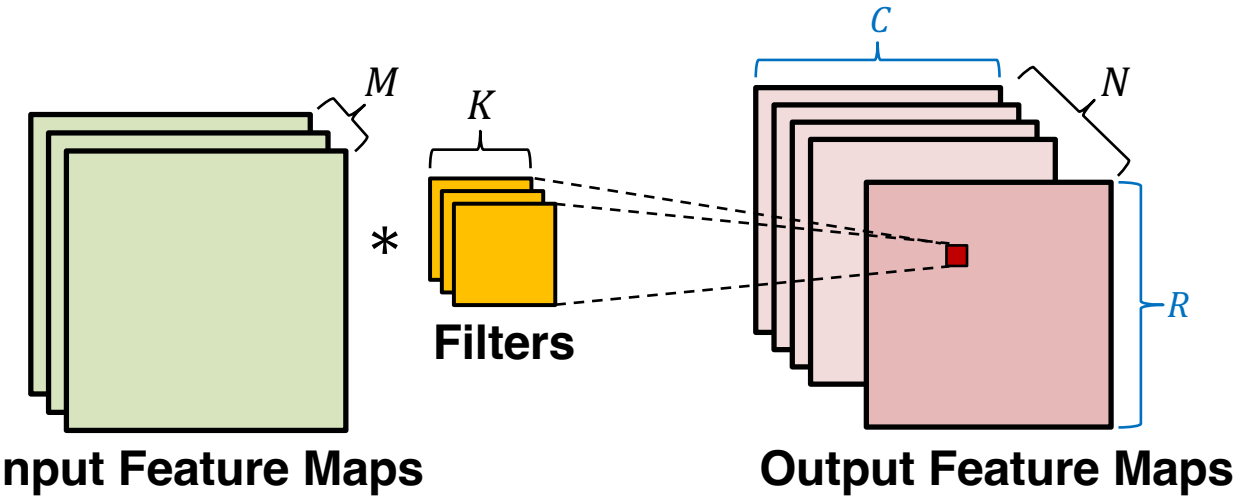
Image credit: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

The Convolutional Layer: A Closer Look



- M input and N output feature maps
  - Usually referred to as “channels”
- Each output map uses M filters, 1 per input map
- $M \times N$ total filters

The Convolutional Layer: A Closer Look

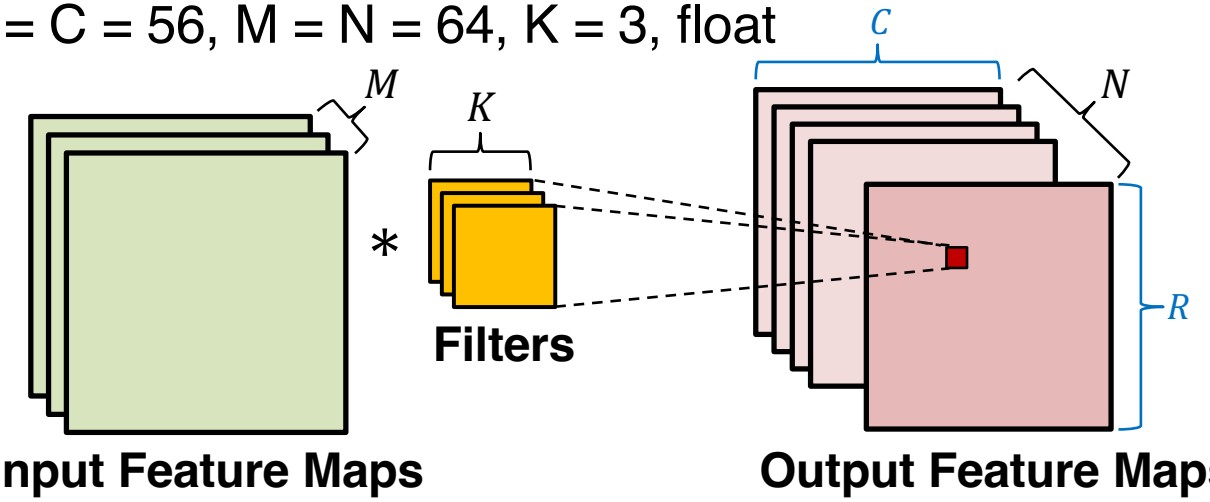



```
1 for(row=0; row<R; row++) {
2    for(col=0; col<C; col++) {
3       for(to=0; to<N; to++) {
4          for(ti=0; ti<M; ti++) {
5             for(i=0; i<K; i++) {
6                for(j=0; j<K; j++) {
                    output_fm[to][row][col] +=
                     weights[to][ti][i][j]*input_fm[ti][row+i][col+j];
}}}}}}
```

Huge amount of parallelism!

OI Analysis of The Convolutional Layer

- E.g., one conv layer from ResNet50:
  - R = C = 56, M = N = 64, K = 3, float




```
1 for(row=0; row<R; row++) {
2    for(col=0; col<C; col++) {
3       for(to=0; to<N; to++) {
4          for(ti=0; ti<M; ti++) {
5             for(i=0; i<K; i++) {
6                for(j=0; j<K; j++) {
                    output_fm[to][row][col] +=
                     weights[to][ti][i][j]*input_fm[ti][row+i][col+j];
}}}}}}
```

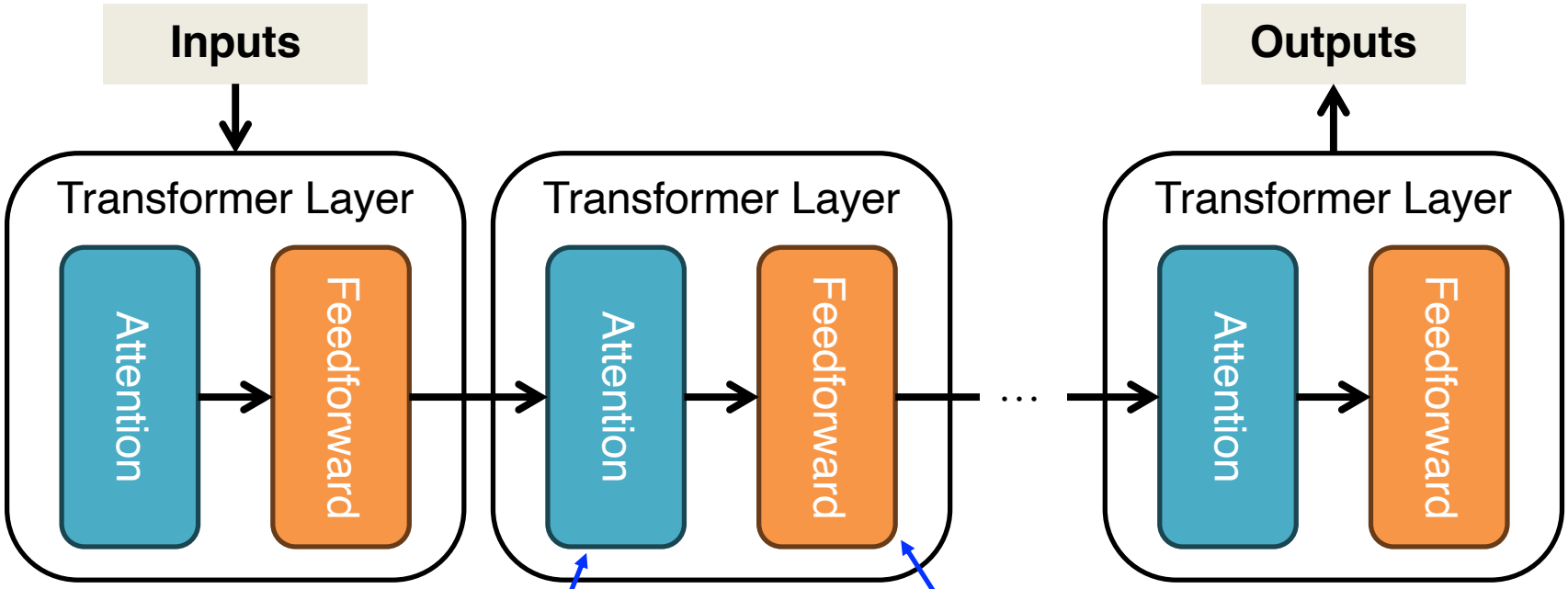
Compute: 231,211,008 FLOPs
Memory: 1,753,088 Bytes
Operational intensity: 131.89

Part 3

TRANSFORMER-BASED LANGUAGE MODELS

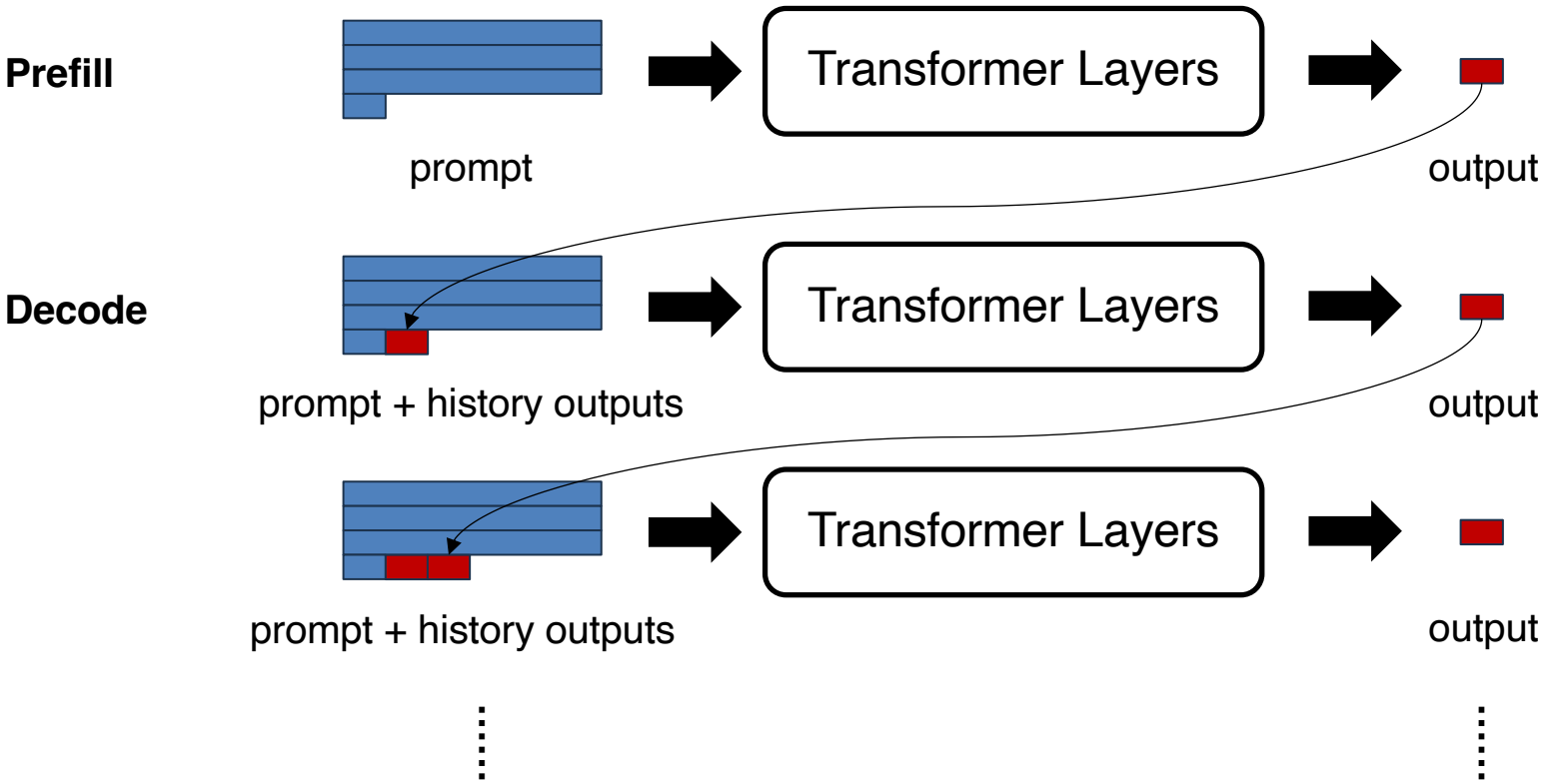
Transformers Overview

- Basic capabilities:
  - Encode the meaning of words and their relationships
  - Process the encoded information
  - Output the encoded information back to words
- This is done by a stack of **transformer layers**



Transformers Overview

- Most language models are **autoregressive**
  - **Prefill**: parse the input prompt and generate the first output
  - **Decode**: append previous output to the input and generate the next output (until the model decides to stop)



Activity: Fill in the Next Word

- What should follow these words:

I	ate	a	sandwich	for	_____

a) **lunch**

b) **HLS**

- Which word provides the critical clue?

I	ate	a	**sandwich**	for	**lunch**

→ it's talking about food

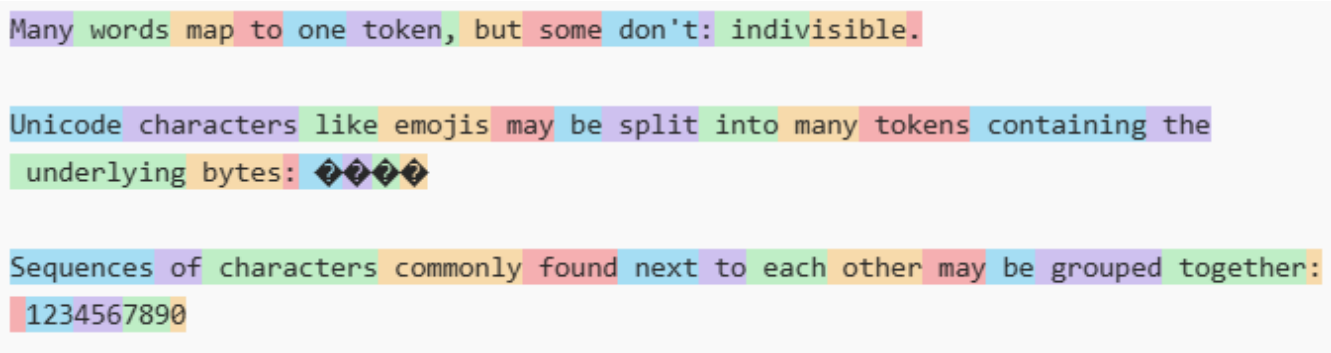
- How do we relate it to food?

I	ate	a	**sandwich**	for	**lunch**

"a" suggests it's a noun
"ate" suggests it's edible

Attention Mechanism

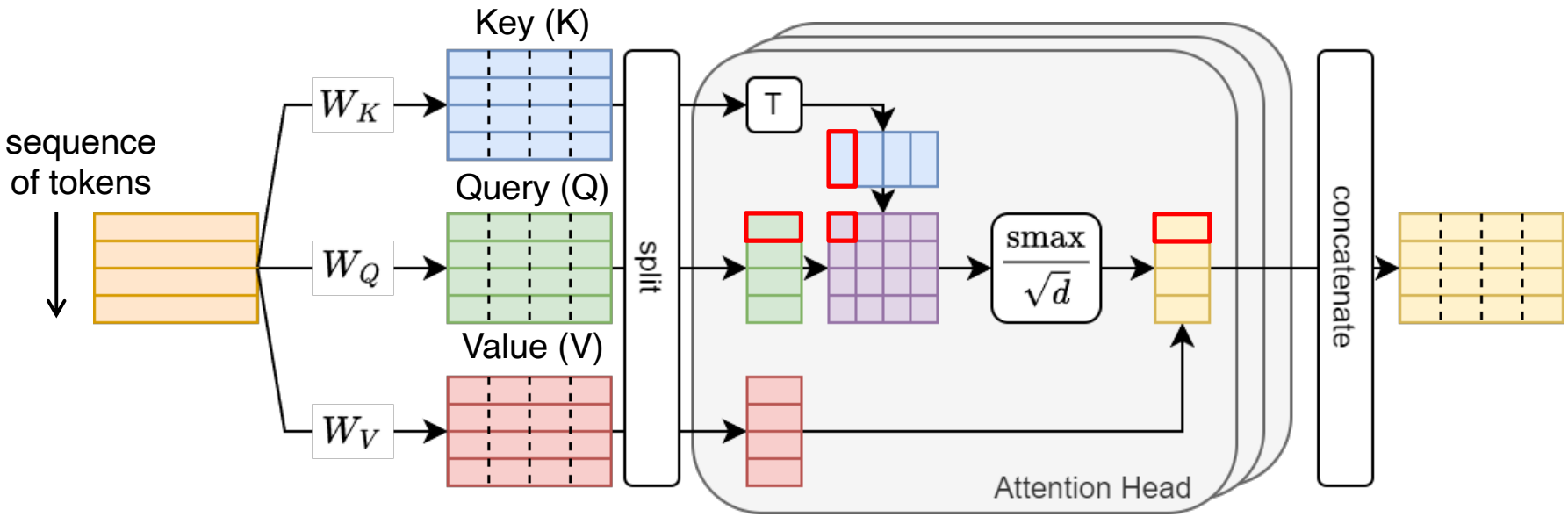
- Attention mechanism helps the model focus on key parts of the input
- It does so by evaluating the relationships among a **sequence of tokens**
  - How strongly are tokens related
  - What is the meaning of each relationship
- Token: meaningful parts of text



Source: https://platform.openai.com/tokenizer

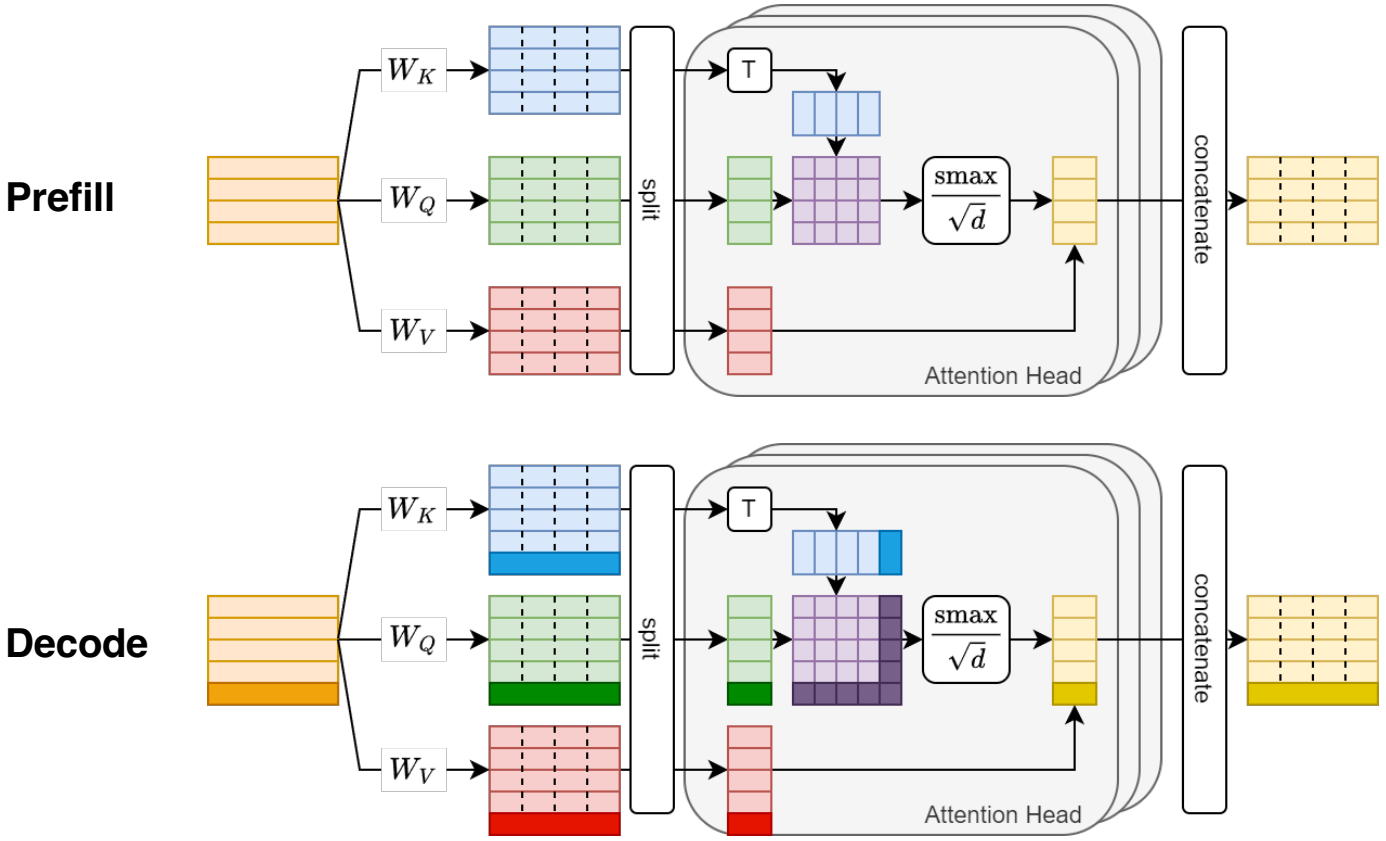
Attention Mechanism Implementation

- Tokens are turned to **key**, **value** and **query** matrices
- $\boldsymbol{k}$, $\boldsymbol{v}$, $\boldsymbol{q}$ of each token are split into **heads**
  - Each head "thinks" about a specific aspect (e.g., grammar)
- $\boldsymbol{q}\boldsymbol{k}^{\mathrm{T}}\boldsymbol{v}$ evaluates the relationship
- Full tensorized form: $\mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d}}\right)\mathbf{V}$



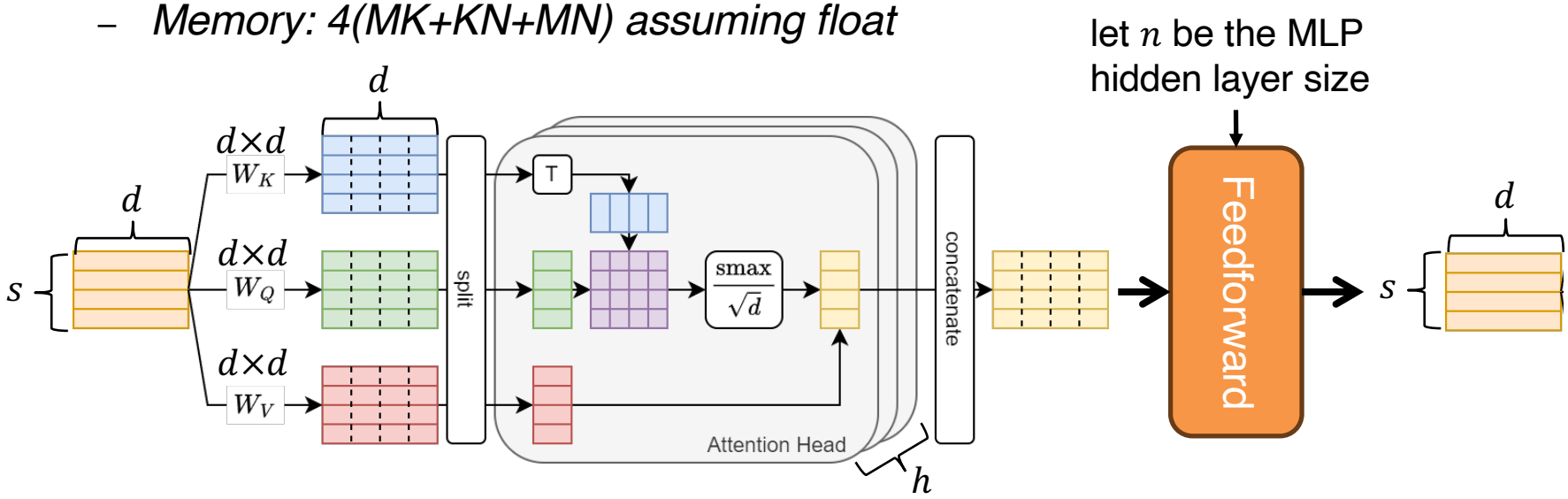
Computation Need for Transformers

- Prefill: matrix-matrix multiplies
- Decode: compute **only one extra entry** – matrix-vector multiplies



OI Analysis for Transformer Layer

- Recap: OI for matrix-matrix multiply $\mathbf{A}^{M\times K}\mathbf{B}^{K\times N} = \mathbf{C}^{M\times N}$
  - Compute: *2MKN*
  - *Memory: 4(MK+KN+MN) assuming float*



Prefill:

compute: $6sd^2 + 2s^2d + 4sdn$

memory: $4(12sd + 3d^2 + 2hs^2 + 2sn + 2dn)$

OI (GPT2-small): 85.33

Cost increases **quadratically** with sequence length (prompt length)!

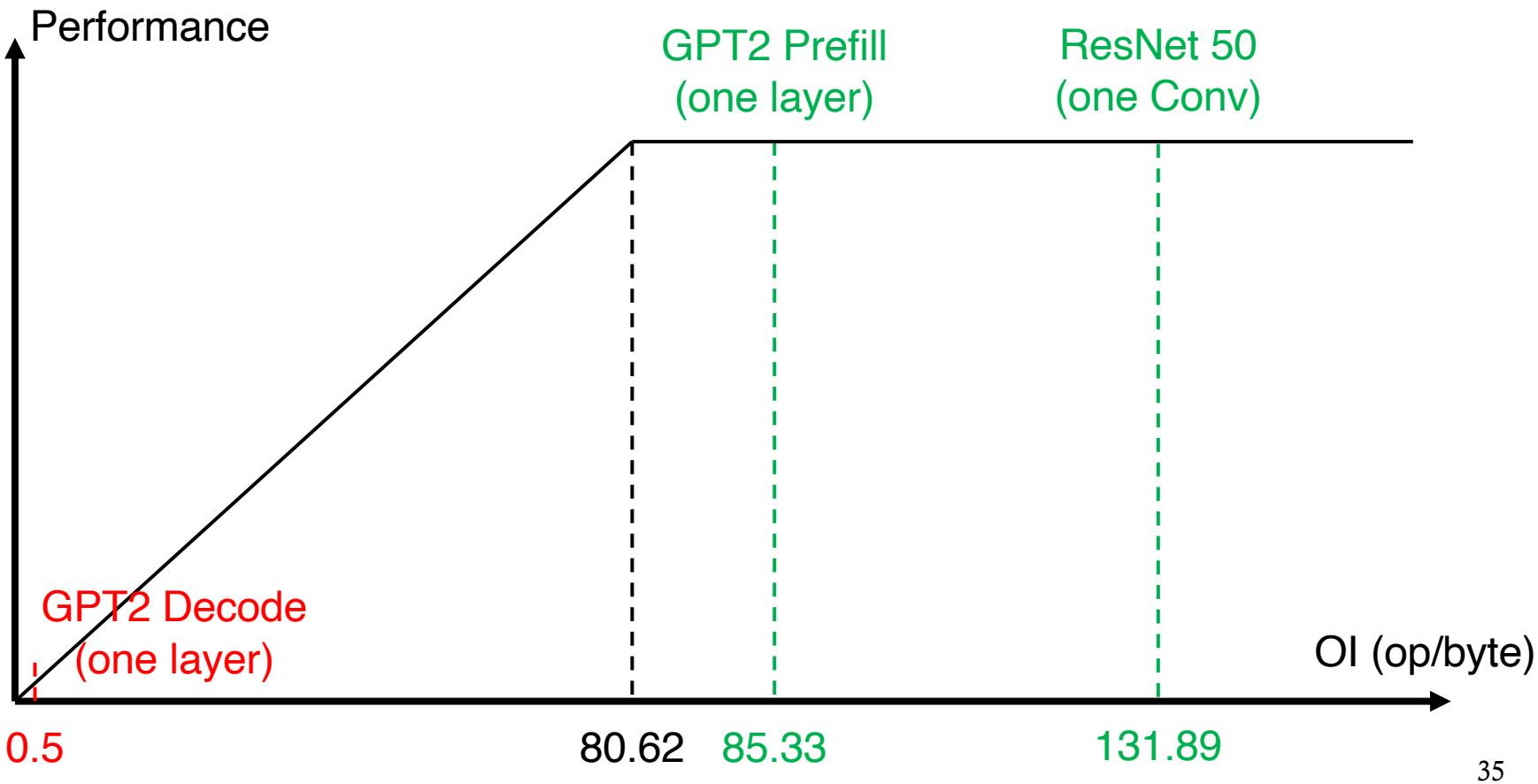
Decode:

OI (GPT2-small): 0.50

Decode stage has a very low OI!

Transformers vs. CNNs

- Assume we use NVIDIA A100
  - Maximum device throughput: 156 TFLOPS
  - Memory bandwidth: 1935 GB/s



Acknowledgements

- The lecture slides contain/adapt materials from
  - Dr. Ritchie Zhao (NVIDIA)
  - ECE 5545 by Prof. Mohamed Abdelfattah (Cornell Tech)
  - CS898 by Prof. Ming Li (University of Waterloo)
  - System for AI Education Resource by Microsoft Research