香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Introduction to Neural Network

Ethen Yuen {ethening}

2025-06-29

What to expect from the lecture

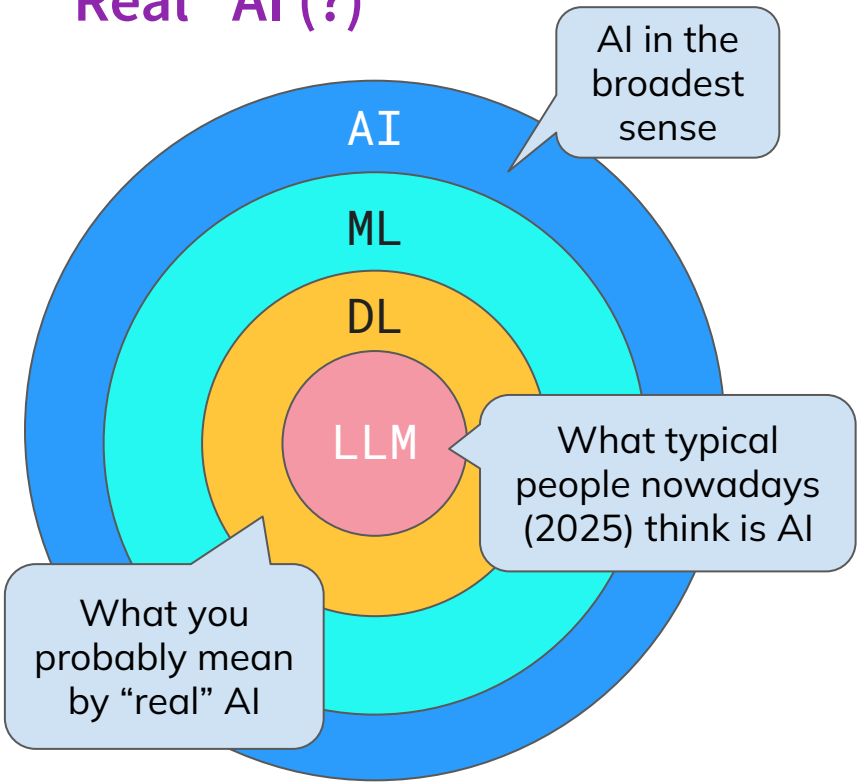
- To learn how neural networks **actually works**, you would need to have sufficient knowledge on the following topics:
  - Probability / Statistics
  - Calculus / Linear Algebra
- It **does not mean** you need to know a lot to start on AI.
  - Doing AI nowadays requires more **engineering effort** than mathematical knowledge.
  - Many people do AI just by tweaking params with their gut feelings (it's like doing output only task!)
- In this lecture, we will go through some high level ideas of AI. (We will go through some maths but they won't be hard.)

https://xkcd.com/1838/

Why Lecture on Neural Network?

- Last HKOI lecture on it is 2017. A lot have changed since then.
- Every year training camp we always get similar feedback on AI Competition: “why don’t we do ‘real’ AI?”.
  - So, what even is “real” AI?
- Useful for people that consider majoring in AI.
  - If you are interested in AI or want to have a job in AI. Taking computer science + spending your effort to do AI projects/join research on your own is a better choice.
- Looking into the future from 2025, it seems that our life will be changed drastically by AI (happening already).

"Real" AI (?)



- **Artificial Intelligence (AI)**
  - To simulate or replicate human intelligence in machines.
- **Machine Learning (ML)**
  - An approach to achieve AI. To enable systems to learn from data without being explicitly programmed for each task.
- **Deep Learning (DL)**
  - A specialized subfield of ML, **our main focus of the lecture**.
- **Large Language Models (LLMs)**
  - A specific type of Deep Learning model. It understand and generate human-like text at very high level of fluency.
  - e.g. ChatGPT, Gemini, Claude, Deepseek

“Real” AI (?)

- It is **technically AI** if “if statement” can produce intelligent results.
  - Although in today’s language context, AI usually refers to Deep Learning model.
- Beware of false advertising: a lot of people use AI as a buzzword only
- Also, not everything have to be “intelligent”, if it **works well**.

What kind of intelligence?

- **What kind of intelligence** AI should exhibit to be considered intelligent?
- We at least know another kind of “intelligent” being, human.
  - AI is considered intelligent if they can **do things that human do**.

What kind of intelligence?

- In reality, the **goalpost of intelligence is always moving.** Many things was once considered difficult AI tasks before, but are no longer considered as intelligent.

Tesler's Theorem (AI effect): "Intelligence is whatever machines haven't done yet."

  - Optical Character Recognition (OCR)
  - Spam Filters
  - Text-to-speech
  - Recommendation System

- It would be fairer to look at before & after, how the technologies have improved.

IOI 2013 - Art Class

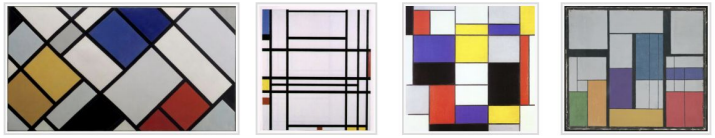
- **Task**: Given an image, classify its style into one of four categories (Neoplasticism, Impressionism, Expressionism, Color Field).
- Implement the function that takes in image dimensions and RGB mats, return the style (90% accuracy = full).

Your Function: `style()`

C/C++

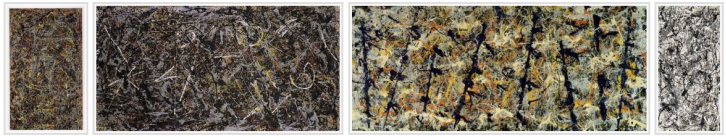
```
int style(int H, int W,
          int R[500][500], int G[500][500], int B[500][500]);
```

Style 1 contains neoplastic modern art. For example:



Style 2 contains impressionist landscapes. For example:

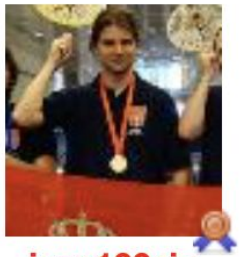
Style 3 contains expressionist action paintings. For example:



Style 4 contains colour field paintings. For example:

IOI 2013 - Art Class

12 years ago, hide # ^ | +6



ivan100sic

It goes something like this:

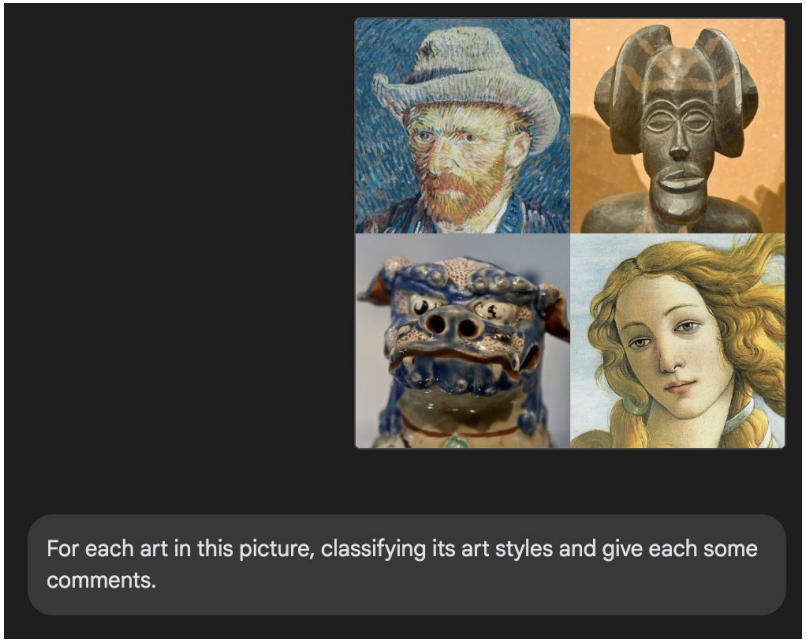
Turn the image to grayscale, calculate the average of absolute differences of values for all adjacent pixel pairs. You can experiment with images given to find appropriate bounds of this value for each class. After you did that, you code them as four if-s. It turns out this works around 91% of the time, of course, scoring 100p. Amazingly simple.

→ Reply

- This was human-driven **feature engineering**. The programmer had to manually devise what visual cues (features) were important and how to extract them.

The Big Leap

- Fast forward from 2013... to **2025**.



https://en.wikipedia.org/wiki/File:Art-portrait-collage_2.jpg

The Big Leap

1. **Top-Left: Self-Portrait with Grey Felt Hat by Vincent van Gogh**
   - **Art Style:** Post-Impressionism
   - **Comments:** This is a classic example of Van Gogh's distinctive style. Notice the thick application of paint (impasto), the short, distinct brushstrokes that create texture and movement, and the expressive, almost emotional use of color. Van Gogh aimed to convey his inner feelings and subjective experience rather than just a visual impression of reality. His self-portraits are particularly powerful explorations of his psyche.
2. **Top-Right: African Tribal Sculpture (Possibly Baule or similar West African style)**
   - **Art Style:** Traditional African Art / Tribal Art
   - **Comments:** This wooden sculpture exhibits common characteristics of many African tribal art forms: stylized human features, often elongated, with an emphasis on symmetry and a serene, powerful presence. The carving is typically geometric and abstract rather than purely naturalistic. Such figures often have spiritual or ceremonial significance, representing ancestors, spirits, or deities, and were integral to community rituals. The specific style can vary greatly by region and tribe.

The Big Leap

3. **Bottom-Left: Chinese Guardian Lion (Foo Dog / Shi Shi)**
   - **Art Style:** Traditional Chinese Decorative Art / Sculpture
   - **Comments:** This is a ceramic Foo Dog (or Shi Shi in Chinese, Komainu in Japanese), a common motif in East Asian art. These guardian lions are traditionally believed to have powerful protective benefits and are often placed in pairs at the entrances of temples, palaces, and homes. They are highly stylized, with fierce expressions, ornate manes, and symbolic attributes. The blue and white glazing is a classic Chinese ceramic technique.
4. **Bottom-Right: Detail of Venus from "The Birth of Venus" by Sandro Botticelli**
   - **Art Style:** Early Renaissance (Italian Renaissance)
   - **Comments:** This is the face of Venus from Botticelli's iconic masterpiece. It epitomizes Early Renaissance ideals: a revival of classical mythology, an emphasis on linear grace and beauty, and a move towards more naturalistic (though still idealized) human forms. Venus's expression is gentle, almost melancholic, and her flowing golden hair is characteristic of Botticelli's elegant style. The painting itself is a landmark of the period, showcasing humanist themes.

The Big Leap

The 2013 solution used simple, human-defined pixel logic. How does today's AI achieve such rich understanding?

- **learning features and reasoning from large data**

Machine Learning Basics

What is Machine Learning?

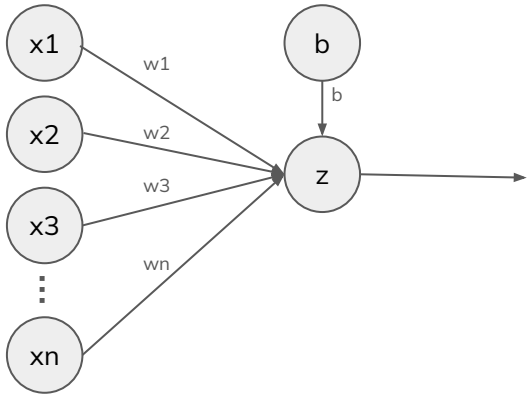
- Definition: Algorithms enabling computers to learn from data without explicit rule-based programming for each task.
- **Supervised Learning**
  - Learning a mapping function **f(X) -> Y** from labeled examples **(X_i, Y_i)**.
  - **X**: Input
  - **Y**: Output labels
- Instead of coming up with the function f(X) ourselves, we make the computer learn what is the best f(X).

What is Machine Learning?

- IOI Art Class
  - A supervised task where
    - X: image pixels
    - Y: art style.
  - In 2013, the function f(X) -> Y, is programmed by programmers after manual feature extraction and adjusting decision boundary until 90% accurate.
  - In **Machine Learning**: the function f(X) -> Y is learned by the computer instead.

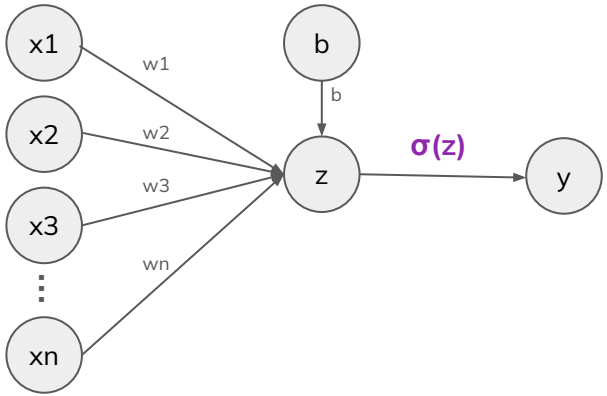
Neuron - Basic Unit

- Analogy: Simplified model of a biological neuron, a basic computation unit.
- Components:
  - Inputs (x_i)
  - Weights (w_i): Importance of each input.
  - Bias (b): An offset.
  - Weighted Sum: $z = \sum(w_i x_i) + b$. This is an **affine transformation** (linear transformation w/ constant).



Neuron - Basic Unit

- Activation Function (σ(z)): The crucial component for **non-linearity**!

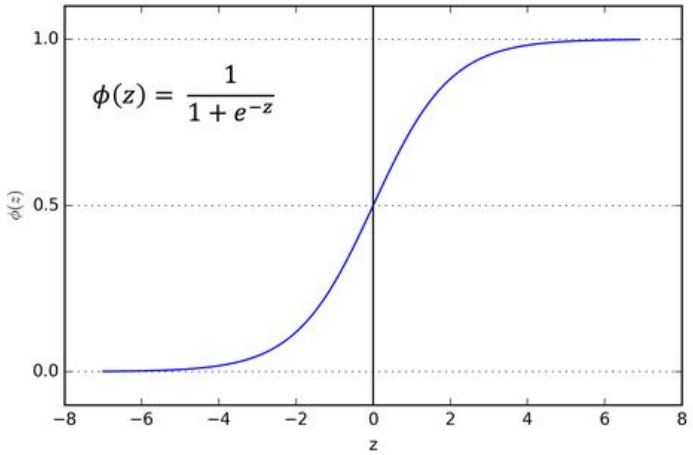


- The following slide show some examples of activation functions.

Activation Functions

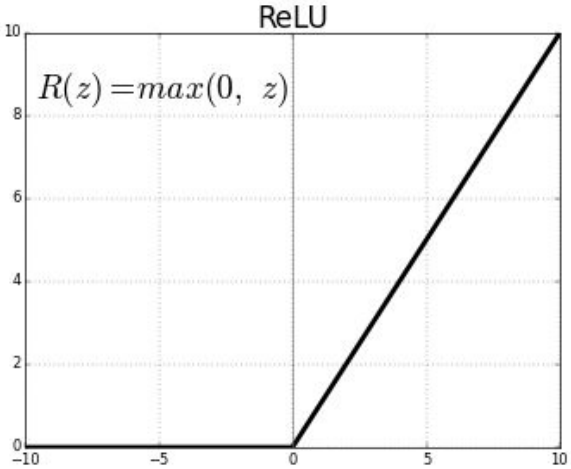
- Sigmoid

Historical. Leads to a problem called “vanishing gradient”



ReLU

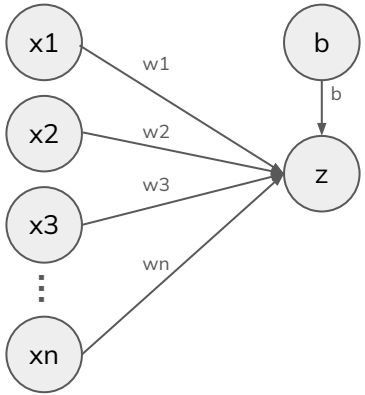
Simple and quick. Widely used nowadays



- The activation functions apply a filter on the neuron’s output, typically meaning the neuron is active or not.

Towards Multilayer Perceptrons (MLP)

- A single neurons (without activation function) is equivalent to a linear function. It is equivalent to a **Linear Regression** problem.



- For example,
  Input: size of a house in a neigbourhood
  Output: selling price
- Usually a linear relation (price per sq. ft)

- The linear model's power is limited, because it assumes that any increase/decrease in input always leads to increase/decrease in output (not always true).

Multilayer Perceptrons (MLP)

- To model more complex function, we need more expressive model. One way to do this is to stack layers of neurons.

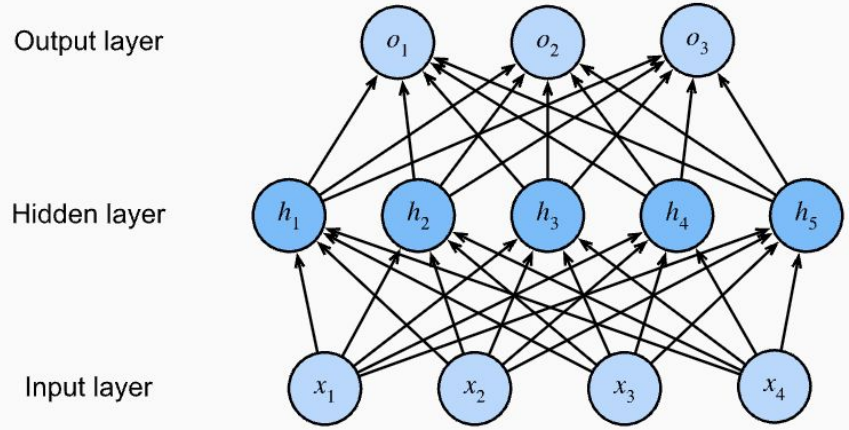
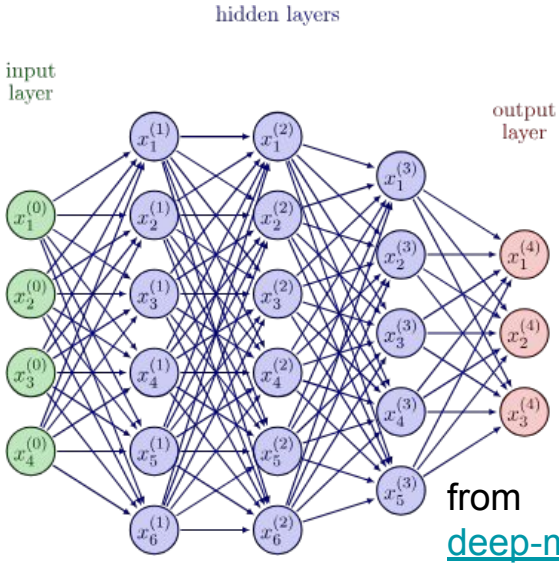


Fig. 5.1.1 An MLP with a hidden layer of five hidden units.
from d2l.ai

Non-linear activation function is crucial. Because linear function on linear function is linear.

Multilayer Perceptrons (MLP)

- You can stack a lot more layers, or add more neurons to increase its expressive power. Each hidden layer learns increasingly abstract and complex representations of the input.



from deep-mind.org/

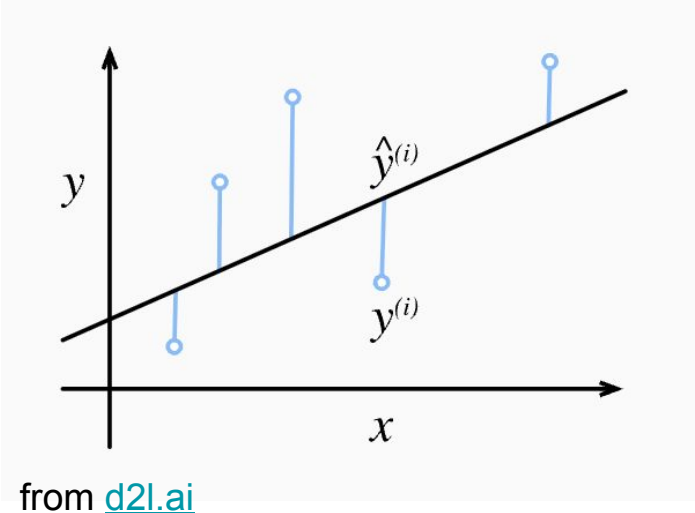
Universal Approximation Theorem: a sufficiently wide MLP can approximate any continuous function

(but finding the weights is the hard part)

How Neural Networks Learn: Minimizing "Loss"

- **Loss Function:** A way to measure how "bad" the network's output predictions are compared to the true labels.

Mean Squared Error (MSE) for regression



from d2l.ai

Cross-Entropy Loss for classification

- True labels (P), one hot vector
  e.g. P = [0, 1, 0] for “cat” out of “dog, cat, bird”
- Model Prediction (Q), probability
  e.g. Q = [0.2, 0.7, 0.1]

- `Loss = -sum( P_i * log(Q_i) )`
  - `-log(0.7) in the example`
- Penalizes confident wrong answers a lot.

How Neural Networks Learn: Minimizing "Loss"

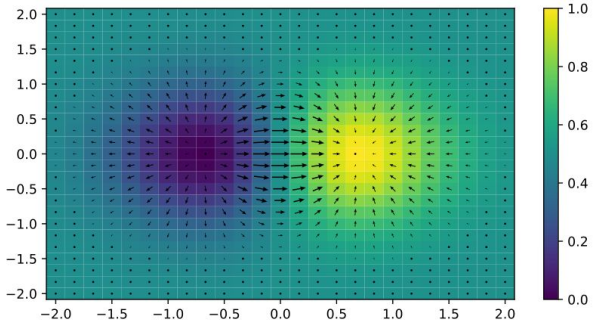
- With loss function, we turn machine learning into an **Optimization problem**. “Find the set of **weights (W)** and **biases (b)** that minimize this loss function over all training data”.

- But, how do we find the W and b?
  - **Random or brute force** like in output-only tasks?
  - There is something more magical than these

Gradient Descent

- **Analogy:** You are standing in a mountain with your eye blindfolded. You can only feel the slope where you are standing but you want to find global min.
- The **gradient (∇L)** of the loss function with respect to the weights tells you the direction of steepest ascent.

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$



- Basically the vector partial derivatives of the output w.r.t. each input.

Gradient Descent

- The **gradient (∇L)** tells you the direction of steepest ascent.
- We take a small step in the opposite direction: **W_new = W_old - η * ∇L**

- **Learning Rate (η)**
  - The size of the step. Critical hyperparameter
  - too large -> overshoot / cannot converge
  - too small -> very slow training

*In practical, the update formula is not as simple, there is something called optimizer that plays a part as well.

Backpropagation

- How do we efficiently calculate ∇L for the final loss of output w.r.t. all weights in (potentially very deep)layered network?
- Let's say Y = f(X), Z = g(Y). With chain rule from calculus, we can get

$$\frac{\partial Z}{\partial X} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X}\right)$$

- We can **work backward from the loss through the layers**. To calculate the gradient of this layer params, we only need the next immediate layer gradient. (and the gradient of next layer's params w.r.t. this layer.

*Early researchers need to implement it from scratch, but modern Deep learning frameworks (PyTorch, TensorFlow) compute this automatically (autograd).

Stochastic Gradient Descent (SGD)

- Calculating gradient on the **entire** dataset (best direction to minimize the total loss of all the data) is too slow.

- **SGD**: Update weights using gradient from just **one** training example (noisy, but fast iterations).
- In practice, we would sample a small random subset of data (**a batch**) each iteration, then compute gradient and update the weight.

Complete Training Flow

- Prepare the dataset first, a bunch of input data + label pair.
- Initialize weights randomly*.
- Repeat for many epochs:
  - Pick a batch of data.
  - **Forward Pass**: Feed data through the network to get predictions.
  - **Compute Loss**: Compare predictions to true labels.
  - **Backward Pass**: Compute gradients of loss with respect to all weights.
  - **Update Weights**: Adjust weights using the optimizer (e.g., SGD with the learning rate).

*Not randomly random, need to be careful or it will cause problem

Generalization: The True Goal

- Analogy: a student can score full marks in all homeworks, but get **0 marks in unseen problems** in exam, we don't actually want that to happen.

- The global minimum of the loss function can be achieved if the model simply **memorize** all the training data and output the label.
  - It achieve a **training loss of 0**, but is it a good model to use in real-life scenario?
  - We want the model to learn the **underlying, abstract rules** instead of hardcoding solution in its brain.
  - **We want it to generalize, not memorize**

Generalization: The True Goal

We don't want the model to just memorize. One useful thing to do is to split all the data you have into 3 different sets.

- **Training set (80%)**
  - Used to update model weights.
- **Validation set (10%)**
  - Used during development to tune hyperparameters (like learning rate, network architecture) and determine when training stop.
- **Test set (10%)**
  - Used only once at the very end to get an unbiased estimate of **performance on unseen data**.

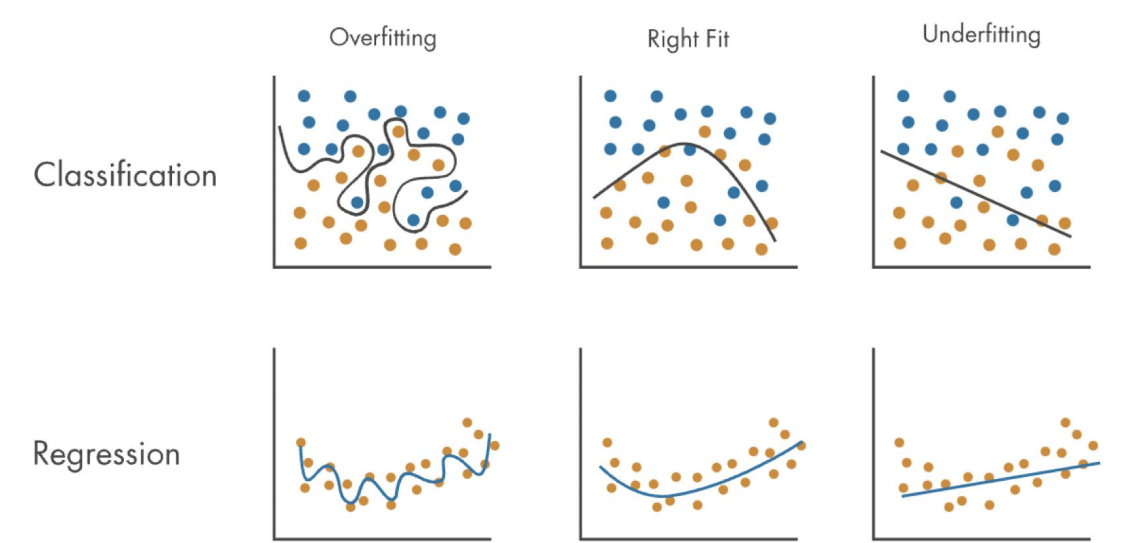
*A simpler setup is to do train (80%) + val (20%)

*This is simplified as well, e.g. K-fold cross validation could be done to get a better estimate.

Generalization: The True Goal

Overfitting: Model learns training data **too well**, including noise. Performs poorly on new, unseen data. (signal: low train loss, high val loss)

Underfitting: Model is too simple, cannot capture patterns even in training data (signal: high train loss, high val loss).

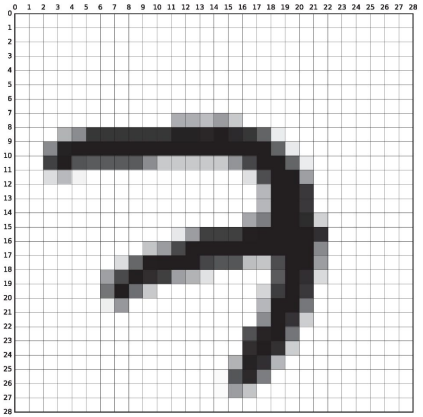


Generalization: The True Goal

- Combat underfitting
  - Add more layers
  - Increase input resolution / features
- Combat overfitting
  - Add more data (diverse data)
  - Use simpler model
  - Use regularization techniques
    - L2 Regularization (Weight Decay): Penalizes large weight values by adding to loss.
    - Dropout: Randomly "deactivates" some neurons during each training step, forcing the network to be more robust.

Learning to See: MNIST

- The “Hello World” of image AI
  - Modified National Institute of Standards and Technology database
  - 60,000 training images and 10,000 testing images (28x28, grayscale)
- A classic classification problem of image as digits.



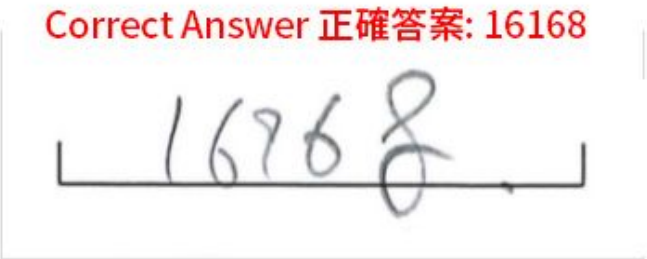
(a) MNIST sample belonging to the digit ‘7’.

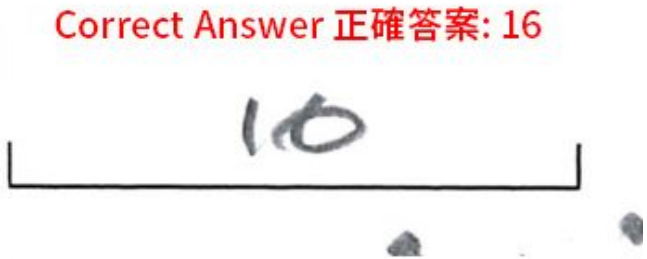
(b) 100 samples from the MNIST training set.

Learning to See: MNIST

- Remarks: MNIST is a rather clean dataset with almost no noise
  - It is often **used as tutorial but is not what you'll deal with in a real-life scenario**
  - Best model as of today can achieve near ~99.9% accuracy.
  - With modern model architecture, it is easy to get 99% accurate.

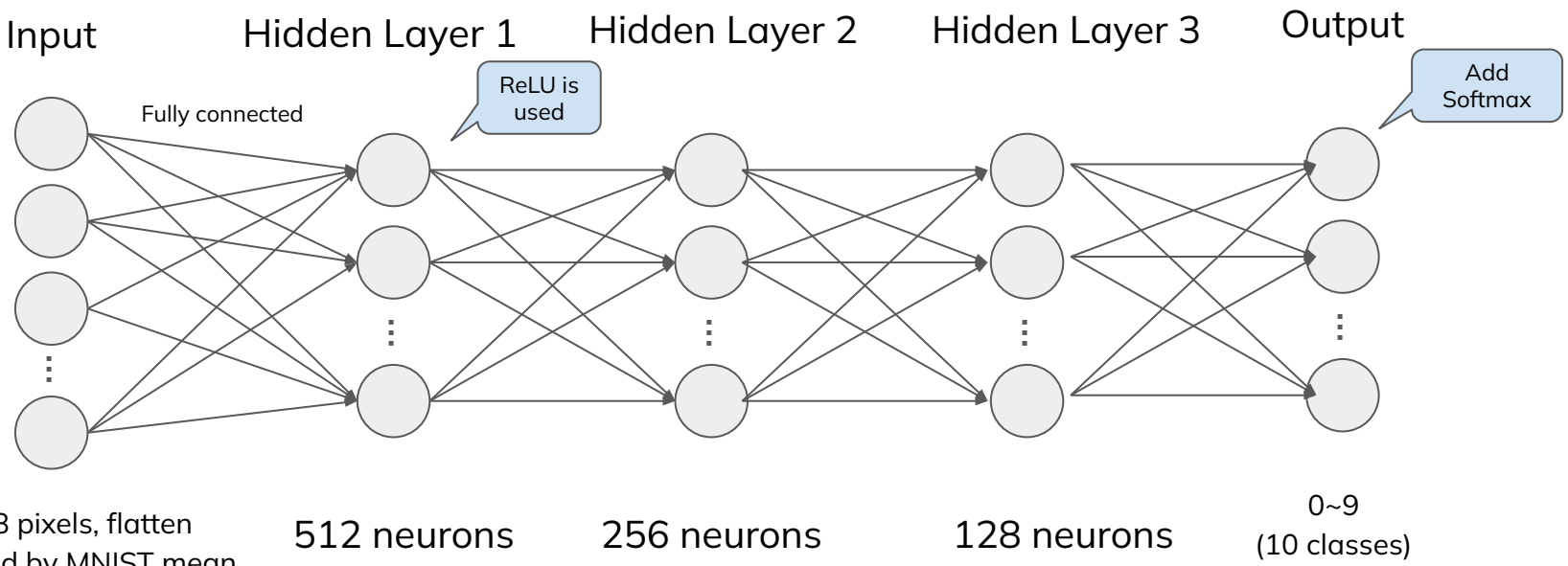
- In real-life, you will see cases like the following (these are from HKOI heat)





Training a MNIST Classifier

- Let's try training a MNIST Classifier



Softmax

- To turn the output results (logits), which are some random values into probability (smaller than 1, add up to 1). We often apply something called softmax.

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

1.3	5.1	2.2	0.7	1.1

↓

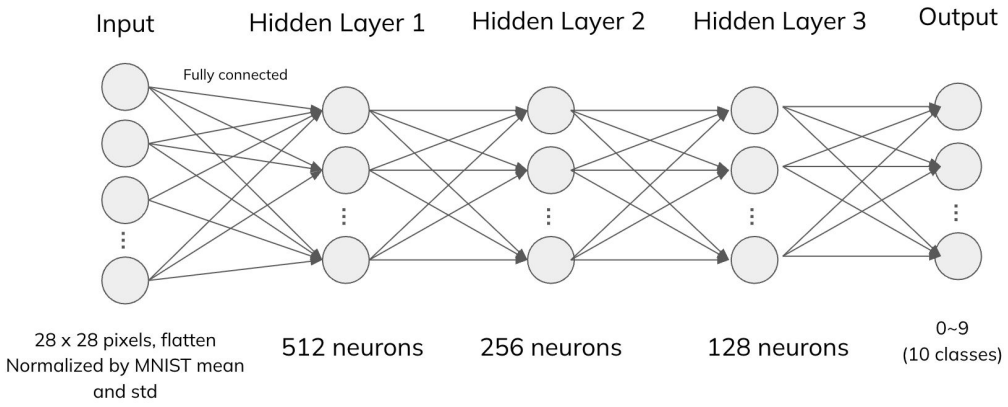
0.02	0.90	0.05	0.01	0.02

- It is typically used with cross-entropy loss.

PyTorch 

- To implement the neural network model, we use a modern python machine learning library named PyTorch.
  - Most commonly used library in research and application nowadays (2025).
- Allow you to quickly compose layers of Neural network, without caring how it is trained under the hood.

PyTorch Implementation

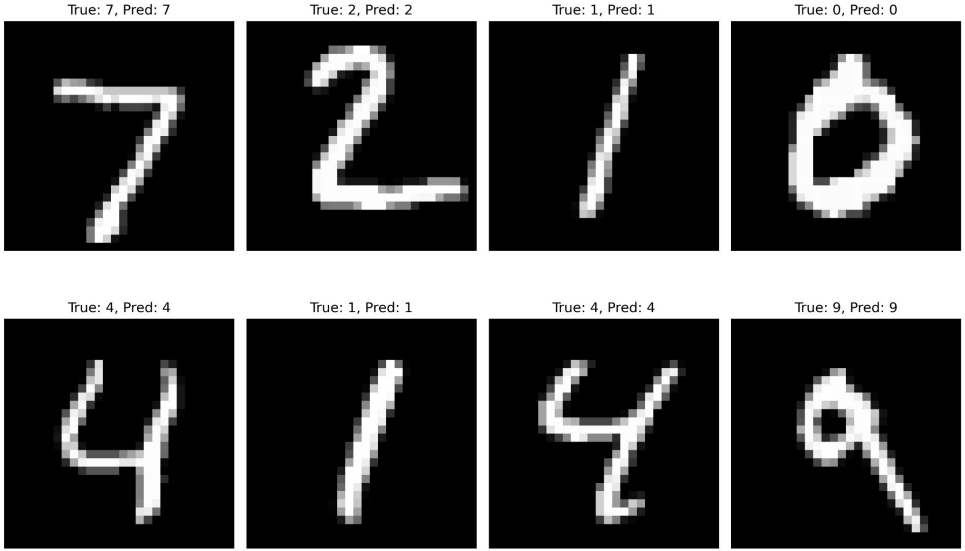



```python
class MNISTClassifier(nn.Module):
    def __init__(self):
        super(MNISTClassifier, self).__init__()
        # Input size: 28*28 = 784 pixels
        self.flatten = nn.Flatten()
        # Fully connected layers
        self.fc1 = nn.Linear(28 * 28, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, NUM_CLASSES)

    def forward(self, x):
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        # Output layer
        x = self.fc4(x)
        return x
```

No need Softmax, because Pytorch Cross-Entropy Loss will handle it

PyTorch Implementation

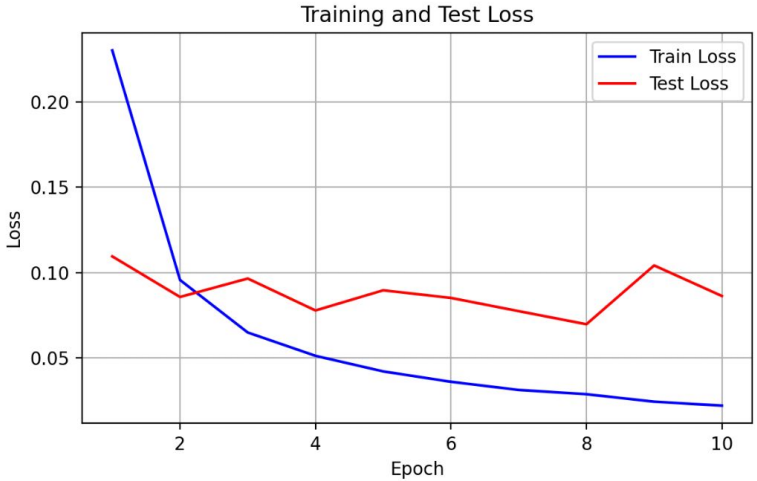
- After the training, we try to run the model on some examples and draw it out.

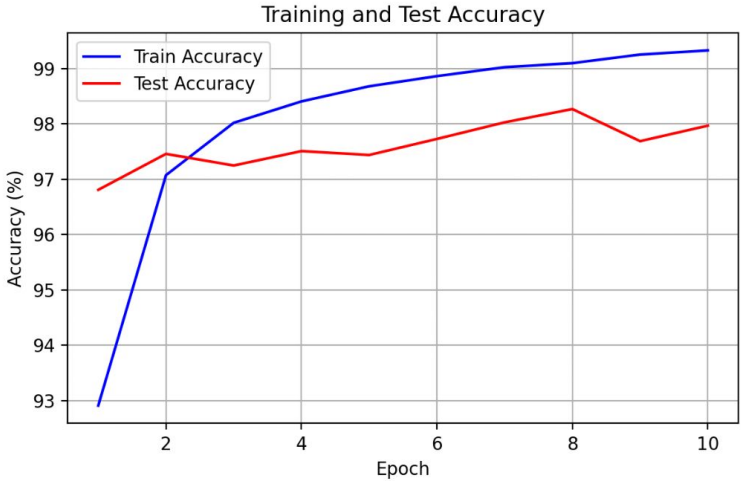


- Looks quite good

PyTorch Implementation

- Visibility into the training process is important: as you can see in the left picture, the test loss is **NOT** really improving.





- This is **overfitting!**

PyTorch Implementation

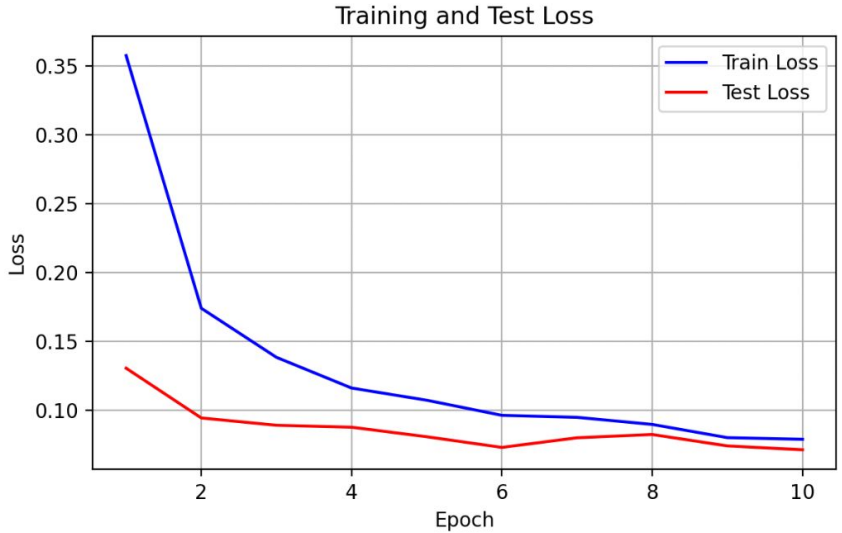
- Let's try to add some dropout!
- **Dropout**: randomly zero some element of the input with probability p.

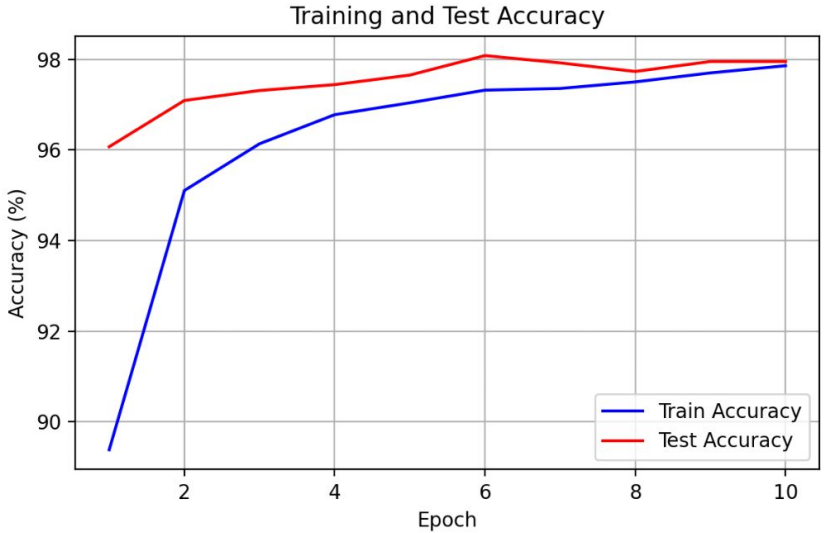
```
class MNISTClassifier(nn.Module):
    def __init__(self):
        ...
        self.dropout1 = nn.Dropout(0.3)
        self.dropout2 = nn.Dropout(0.4)
        self.dropout3 = nn.Dropout(0.5)

    def forward(self, x):
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = F.relu(self.fc2(x))
        x = self.dropout2(x)
        x = F.relu(self.fc3(x))
        x = self.dropout3(x)
        x = self.fc4(x)
        return x
```

Add progressive dropout after each hidden layer

PyTorch Implementation

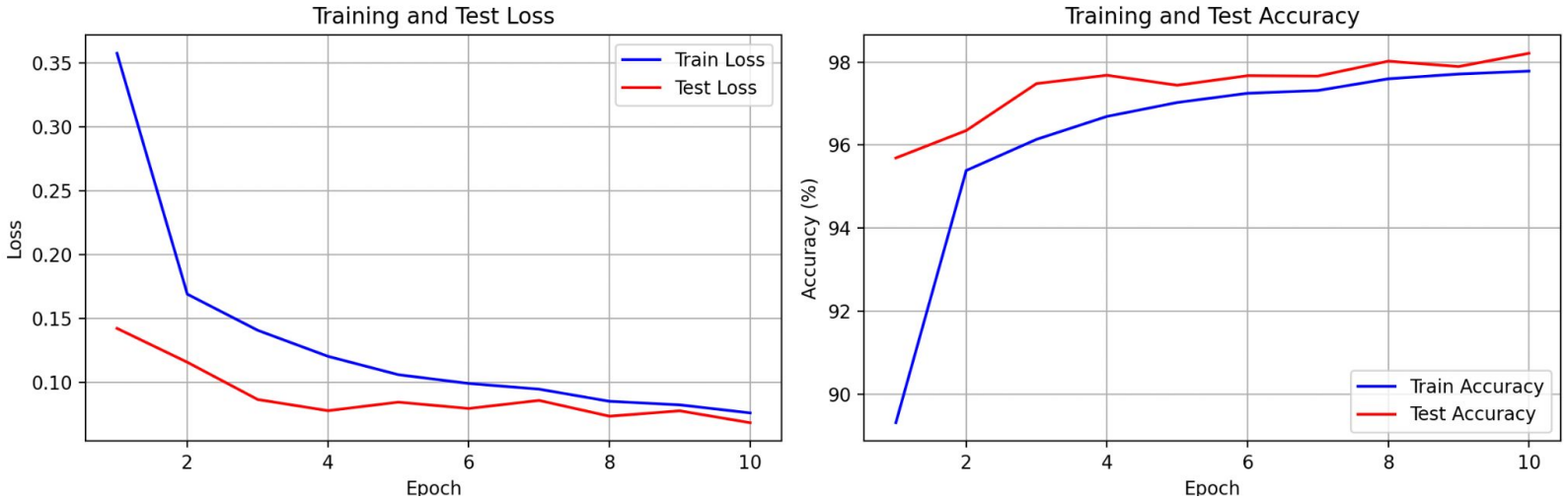
- This time, with dropout, we got a better test loss curve!
- We also got slightly better accuracy.





PyTorch Implementation

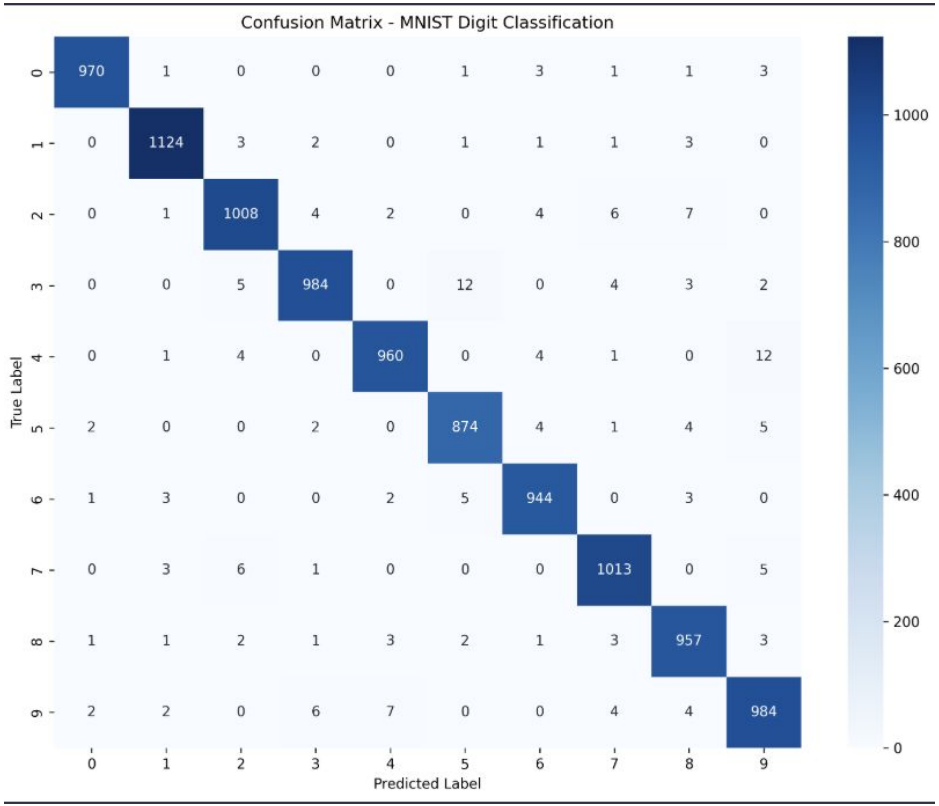
- Adding **L2 Weight Decay** further get our accuracy higher.



- There are a lot of hyper-parameters to be tuned, e.g. learning rate, optimizer, number of nodes per layers, regularization technique strength.

Confusion Matrix

- **Confusion Matrix** helps us understand how well the model is performing and what classes is difficult to it.
- e.g. We can add more training data targeting the issues.



Deep Learning & Convolutional Neural Network

Learning to See, Actually

Despite being successful with MNIST, standard MLPs struggle with Images.

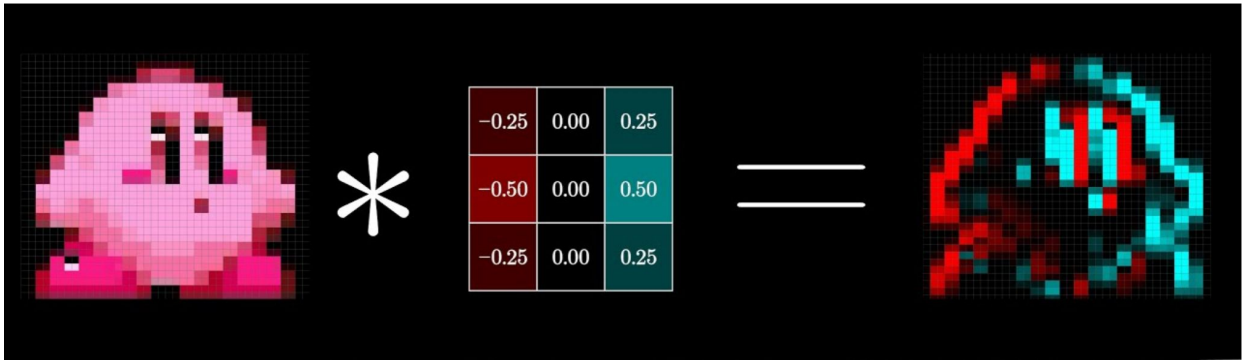
- Flattening images (e.g., 28x28 to a 784-vector) **loses crucial 2D spatial** information.
- **Parameter explosion**: A 1920x1080 image, even if downscale aggressively to a 1000-neuron hidden layer needs 2 billion weights!.
- Lack of **translation invariance**: A cat in the top-left corner is still a cat if it's in the bottom-right. MLPs treat these as completely different inputs.

To handle image, it is more practical to use something called Convolution Neural Network (CNN) instead.

Convolution

Convolution: We slide a small matrix, called a **kernel** or **filter**, over the image.

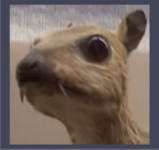
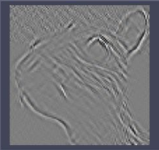
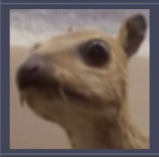
- For a N*N kernel, each step it multiplies the value of a N*N subgrid of image and the kernel to produce a new value.
- We can do some cool things by applying the convolution filter to the image.



From 3b1b - convolution

Convolution

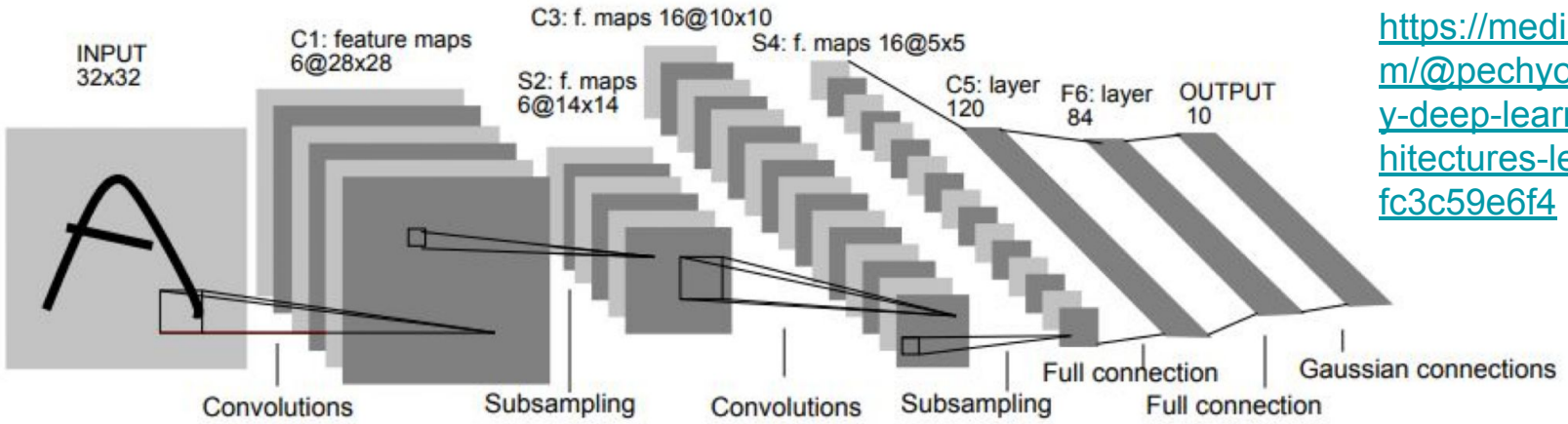
- Despite the small number of parameters of a filter, it can be very expressive when you apply it to the whole input image.
- We can **detect edge**, for example, using a 3x3 kernel.
  - After all, only the local pixels level is important for edge detection.
  - Kernel is a good “Feature Extractor” to a image.

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

From https://en.wikipedia.org/wiki/Kernel_(image_processing)

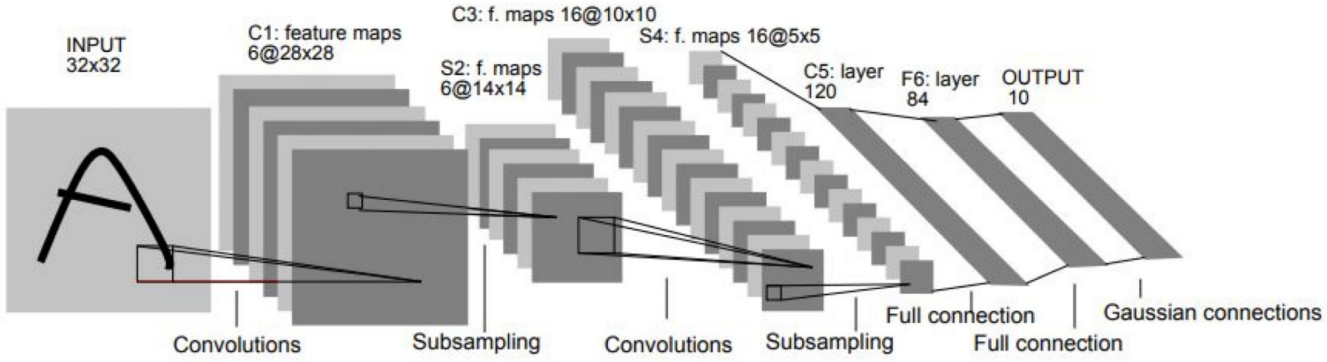
LeNet-5 (Yann LeCun et al 1998)

- A pioneering Convolutional Neural Network (CNN) model design.
- Tackle US Postal Service digit recognition (for MNIST dataset).
  - Applied backpropagation instead of hand designing kernels.



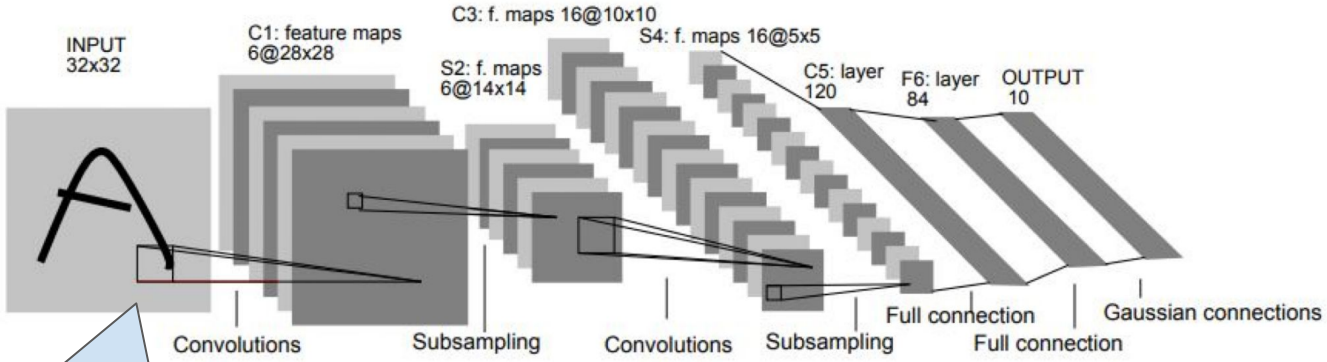
From https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4

LeNet-5 (Yann LeCun et al 1998)



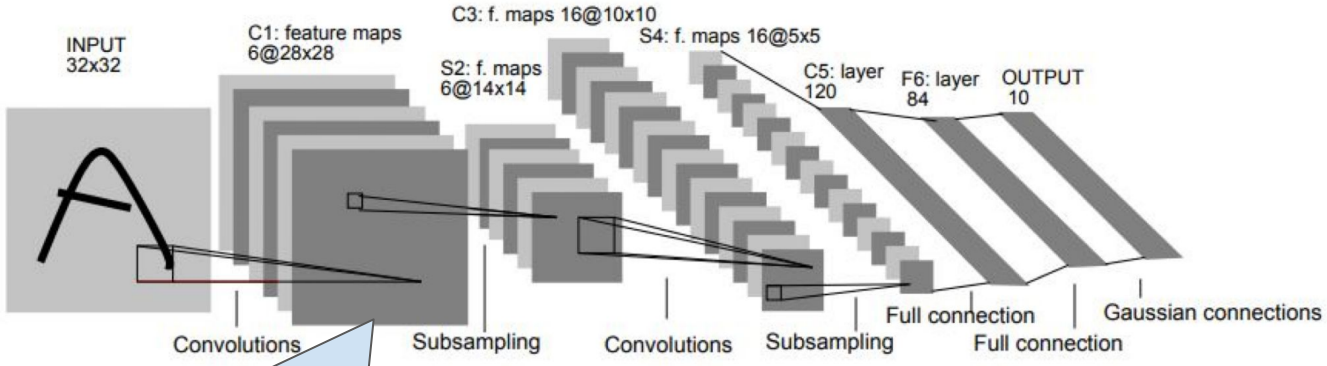
Each kernel learns to detect specific local patterns (edges, corners, textures, simple shapes)

LeNet-5 (Yann LeCun et al 1998)



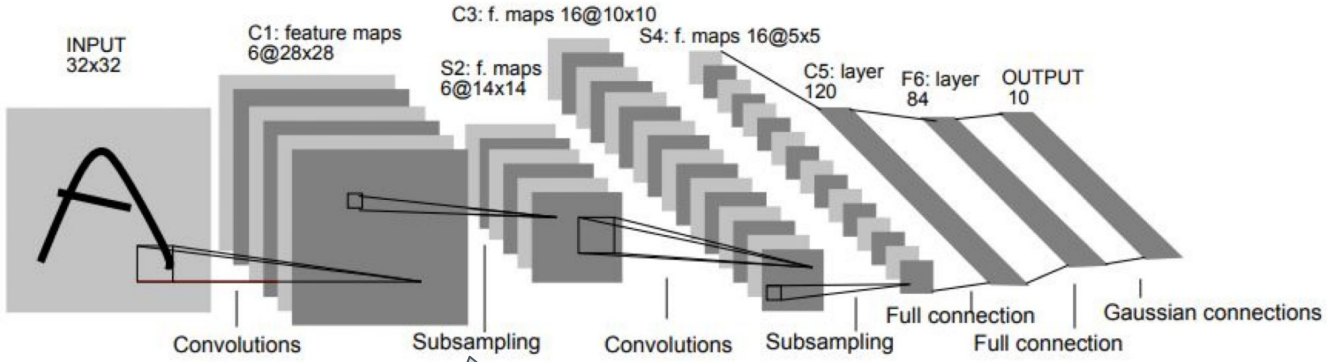
Parameter Sharing: The same kernel is applied across all spatial locations. This is key for efficiency and achieving translation invariance.

LeNet-5 (Yann LeCun et al 1998)



Feature Maps (Activation Maps): The output of a filter applied to an image. Typically use multiple filters per layer to learn diverse features.

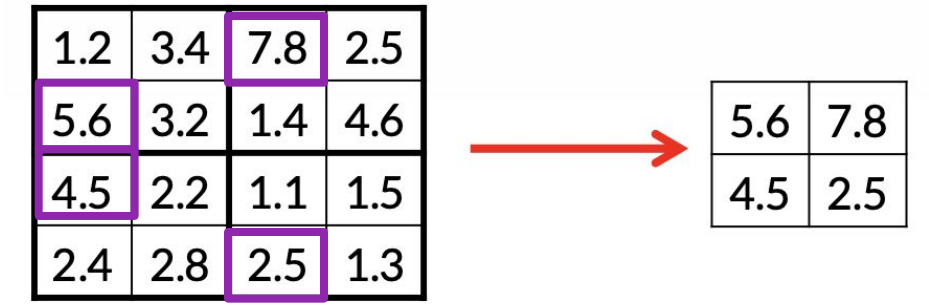
LeNet-5 (Yann LeCun et al 1998)



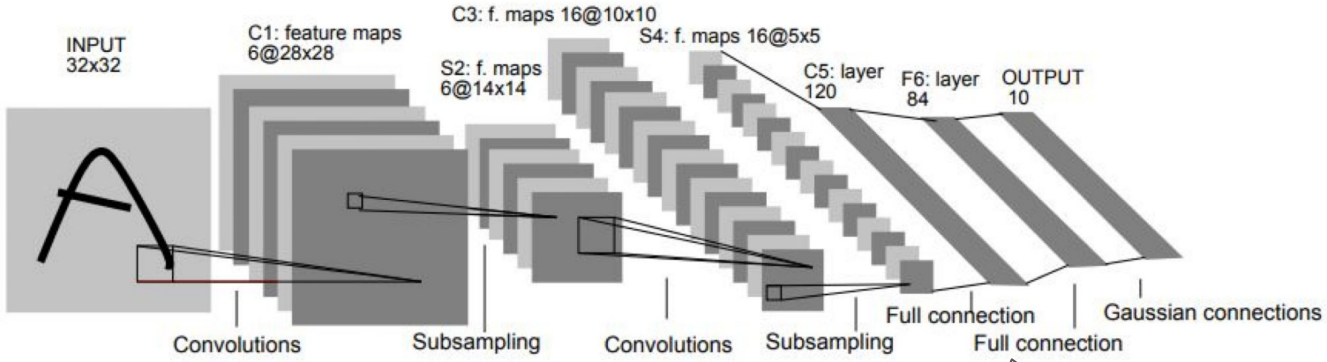
Downsampling: Reduces the spatial dimensions (height, width) of the feature maps.

Downsampling

- **Pooling Layer (e.g., Max Pooling, Average Pooling):**
  - **Downsampling:** Reduces the spatial dimensions (height, width) of the feature maps.
  - Makes the representation more robust to small translations (local translation invariance).
  - Reduces the number of parameters in subsequent layers.



LeNet-5 (Yann LeCun et al 1998)



MLP-like structure at the end: after the early layer that do Feature extraction

Typical CNN Architecture

- **Pattern:** Stack [Convolution -> ReLU -> Pooling] blocks.
- As we go deeper:
  - Spatial dimensions (H, W) typically decrease (due to pooling).
  - Number of channels (features) typically increases (learn more complex features).
- Finally, flatten the high-level feature maps and feed into Fully Connected (Dense) layers for classification.

Before Deep Learning Boom...

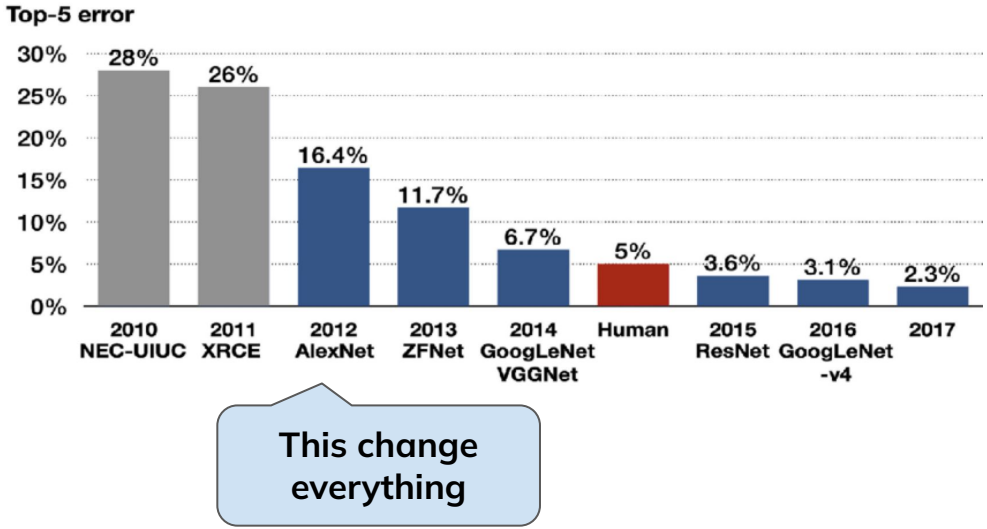
- **Question:** LeNet was created in 1998. Deep learning had its breakthrough in 2012. Why the 14-year gap?
- LeNet proved the concept of CNNs worked, but **the world wasn't ready yet**. Neural networks falls out of favor compared to other classical ML techniques like SVM (Support Vector Machine).
  - **Data**: Real-world problems needed massive datasets, which didn't exist yet.
  - **Compute**: Training on large datasets was computationally impossible on the **CPUs** of the era.
  - **Algorithm**: LeNet used activations (like tanh) that made very deep networks hard to train.

The ImageNet Challenge

- Classify an image into one of 1000 distinct categories (leopard, container ship, cherry, etc.).
- A massive leap in difficulty from MNIST.
- 1281167 training images.
- https://www.image-net.org/

The ImageNet Challenge

- The mainstream solution before deep learning is complicated.
- People basically compete on better hand-crafted feature extractor, probably with domain-specific knowledge.



AlexNet (Alex Krizhevsky et al., 2012)

- Basically, stacking more layers than LeNet
- This is when **Deep Learning** start to become a thing.

- Novelty:
  - Use ReLU instead
  - **Use GPUs** for training!
    - Highly parallelizable for simple calculations

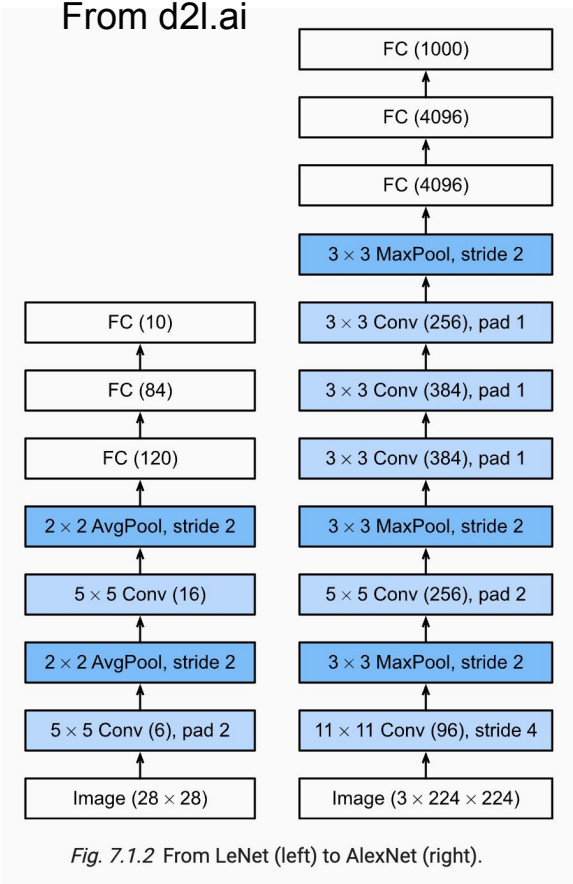


Fig. 7.1.2 From LeNet (left) to AlexNet (right).

AlexNet (Alex Krizhevsky et al., 2012)

- Interesting enough, in the earlier layers of AlexNet, the model actually learn filters like classical feature extractors, without human interference.
  - This kicked off the modern "deep learning" era. It proved that deeper neural network learn to extract feature better than human.
- It was also where we **start to lose understanding** what a model is doing.

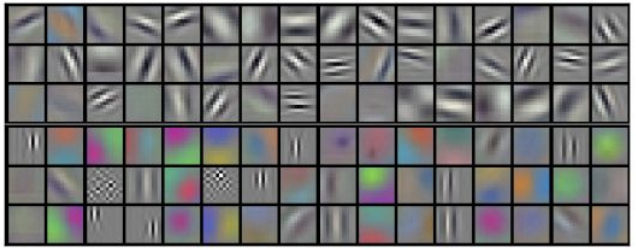


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

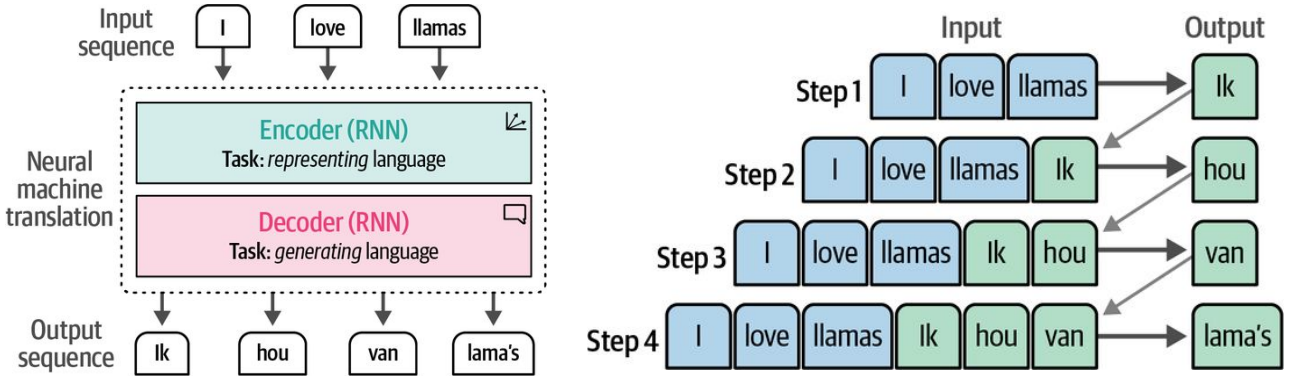
ImageNet Classification with Deep Convolutional Neural Networks, 2012

Towards Large Language Model

NLP (Natural Language Processing)

In NLP tasks, e.g. **Translation**, we often deal with long, variable length text.

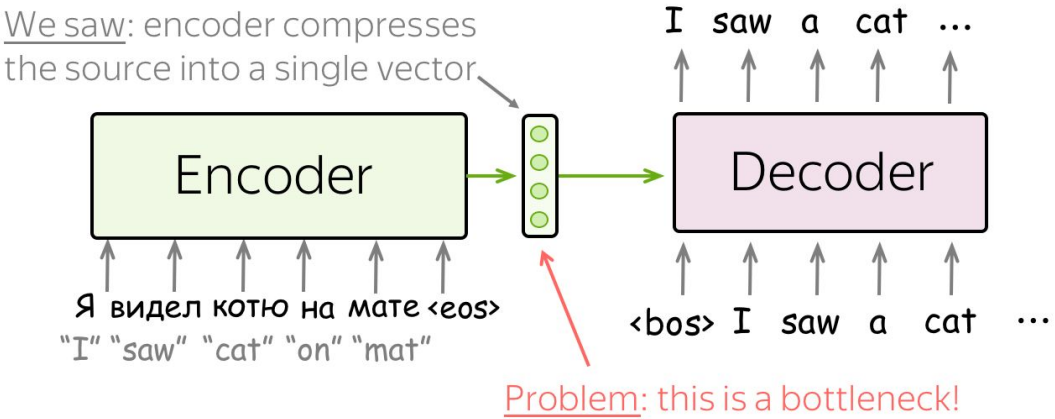
- CNNs: Great for images, but less viable for variable-length text.
- RNNs (LSTMs/GRUs): Process sequences token-by-token, maintaining a "memory" (hidden state). Better for text.



(Hands-On Large Language Models, by Jay Alammar & Maarten Grootendorst)

Challenge with RNNs

- Difficulty capturing **very** long-range dependencies (vanishing/exploding gradients over many time steps).
- Sequential computation limits parallelization.

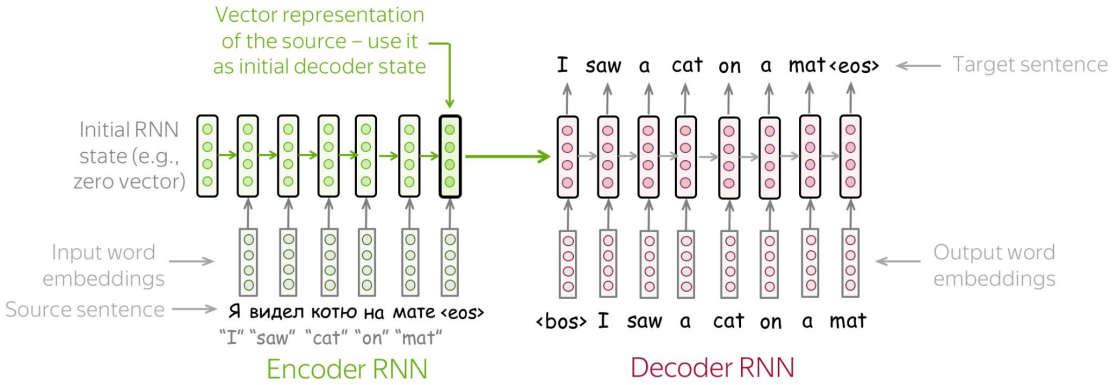


(https://wikidocs.net/178419)

NLP (Natural Language Processing)

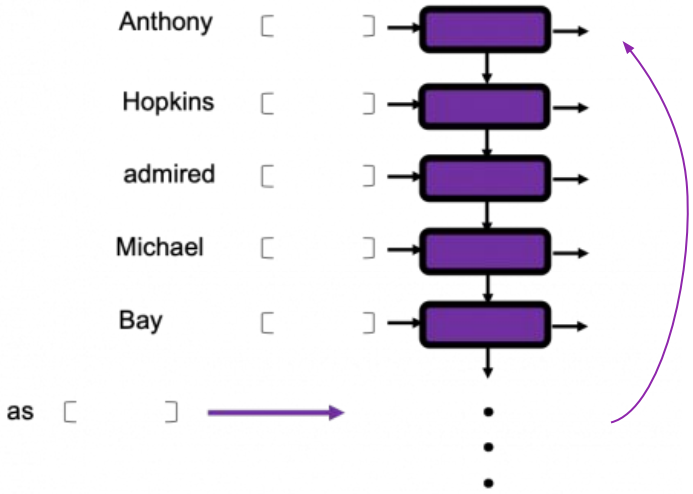
In NLP tasks, e.g. **Translation**, we often deal with long, variable length text.

- CNNs: Great for images, but less viable for variable-length text.
- RNNs (LSTMs/GRUs): Process sequences token-by-token, maintaining a "memory" (hidden state). Better for text.

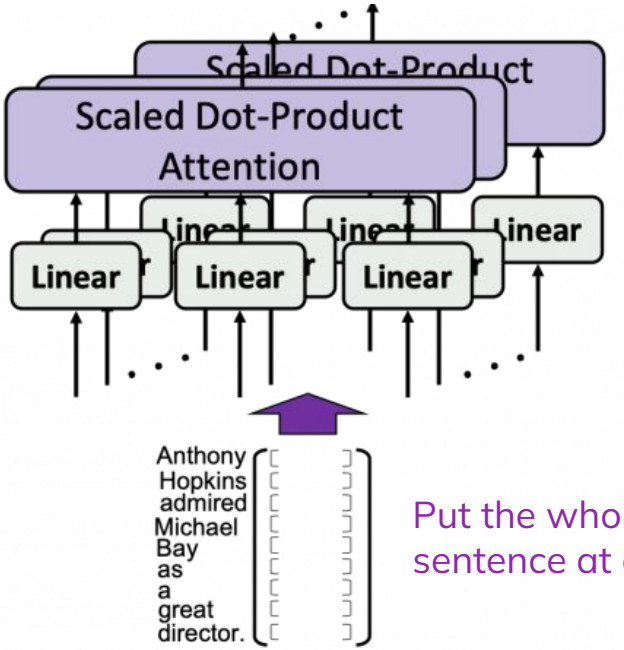


(https://wikidocs.net/178419)

RNN vs. Self-Attention



Tend to "forget"

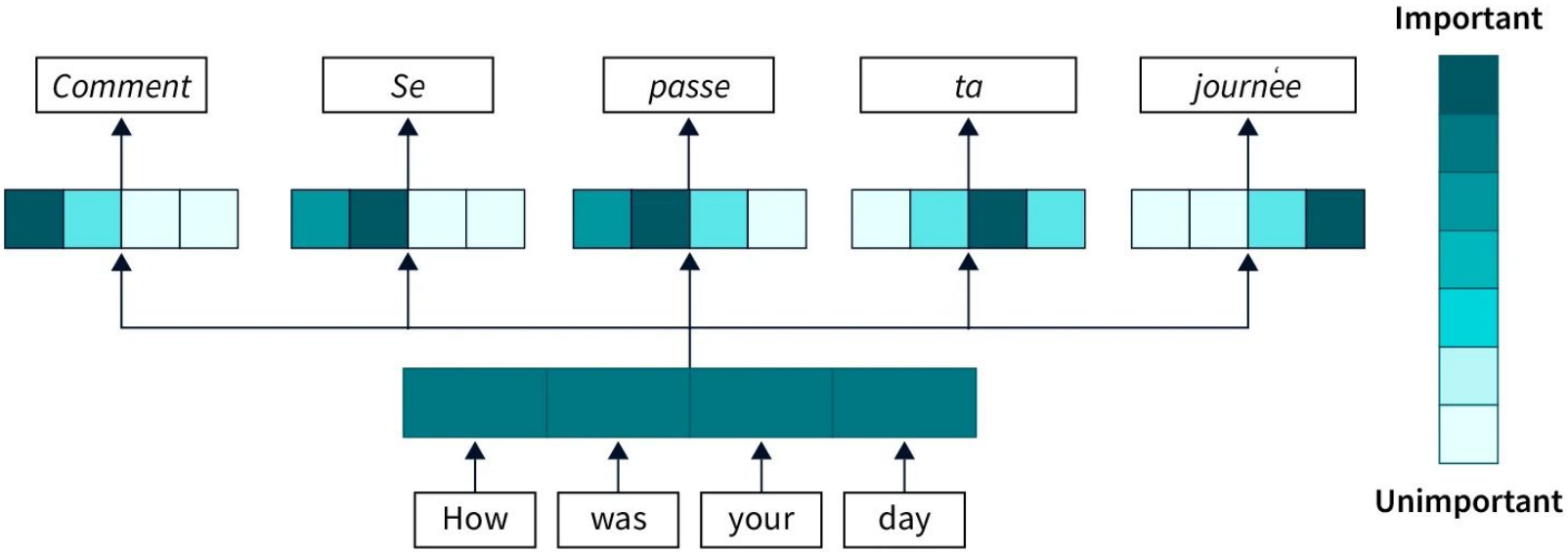


Put the whole sentence at once

(https://wikidocs.net/167212)

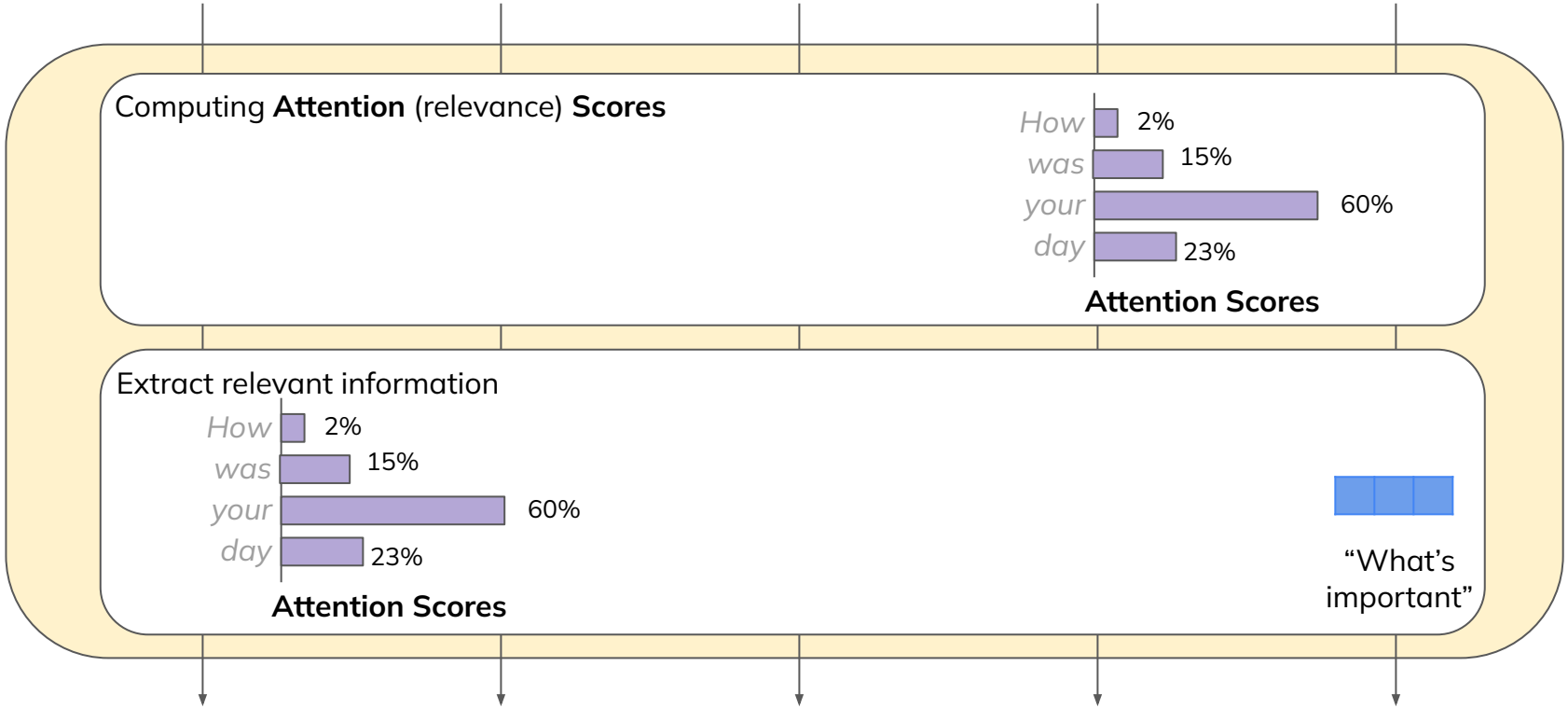
Attention Mechanism: Focusing on What Matters

High level: At different timestamp, allow the model "focus" on different parts of the input.

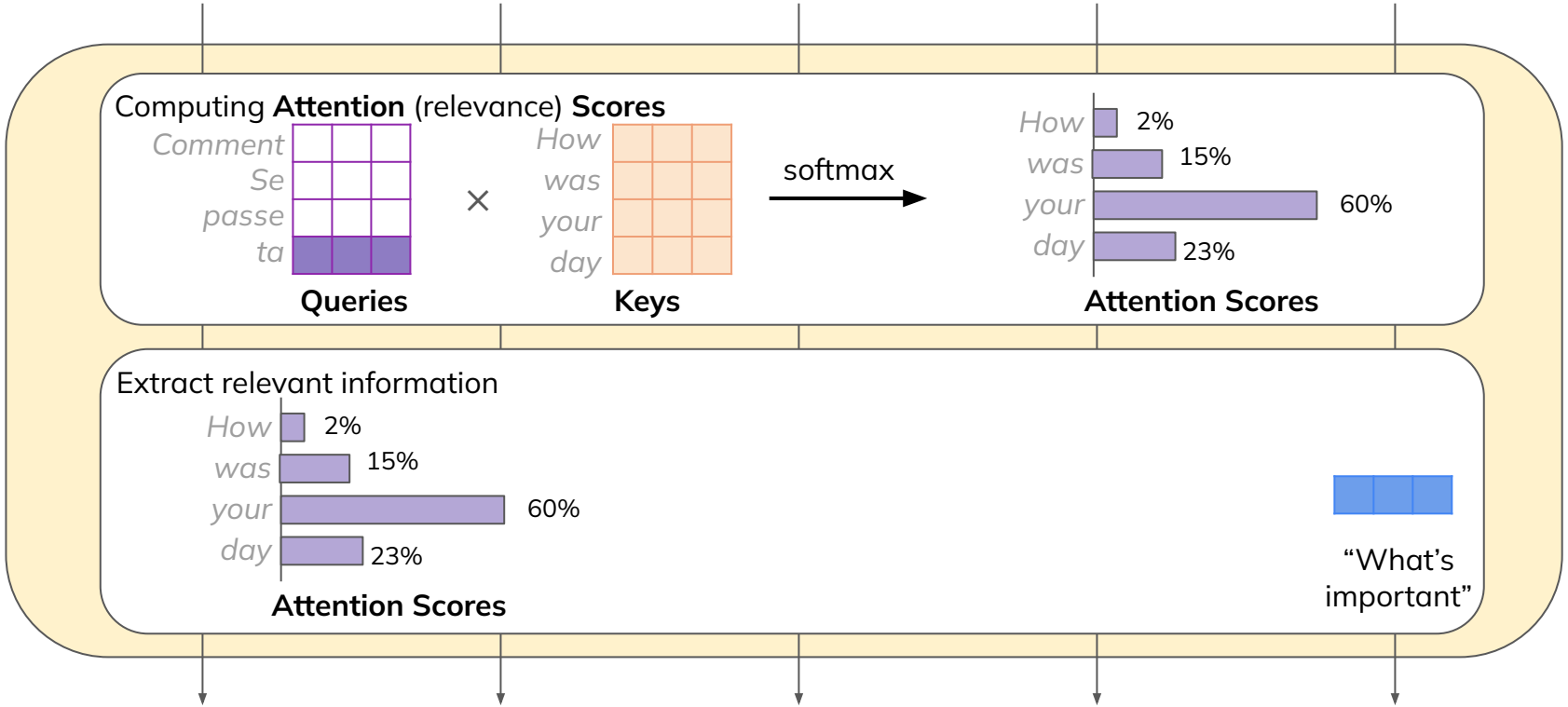


(https://www.scaler.com/topics/deep-learning/attention-mechanism-deep-learning/)

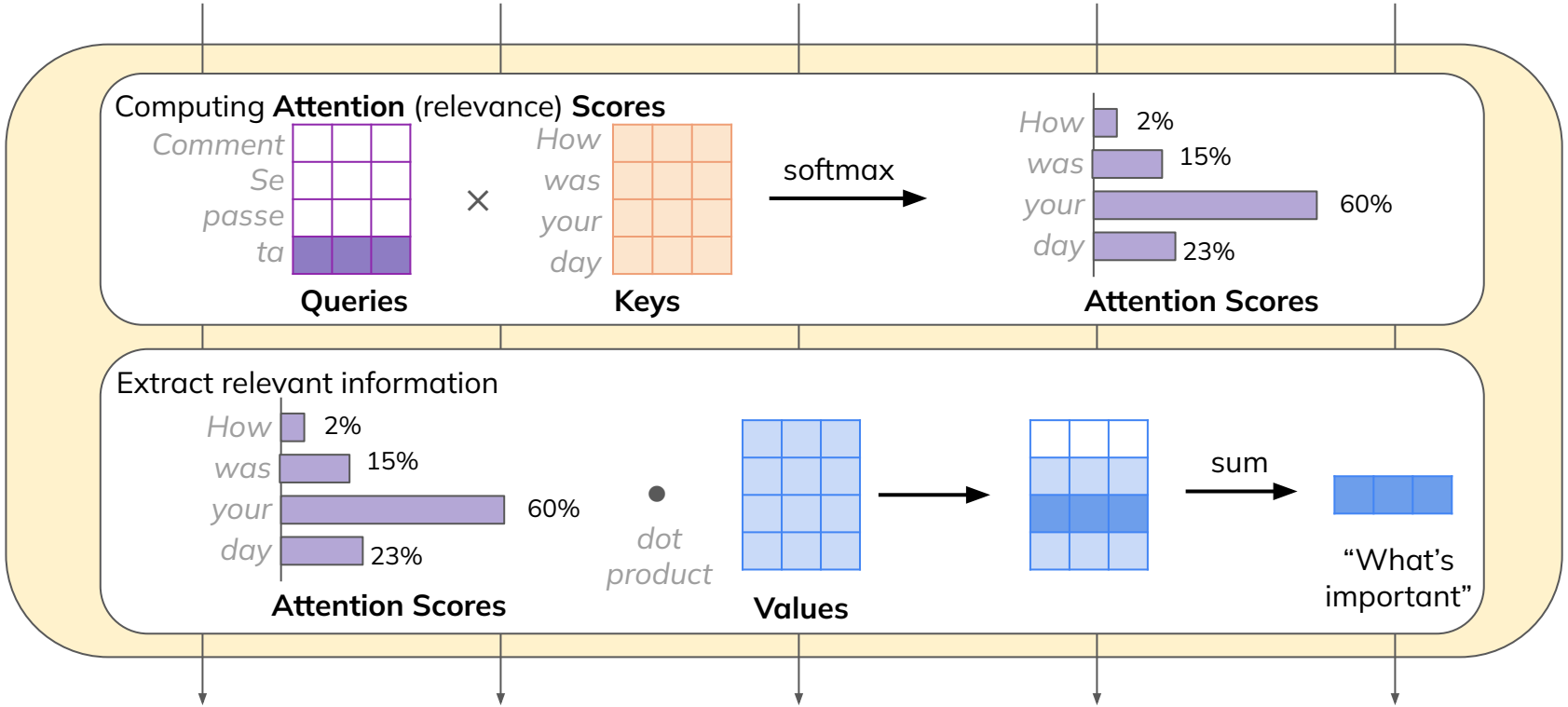
The Attention Layer



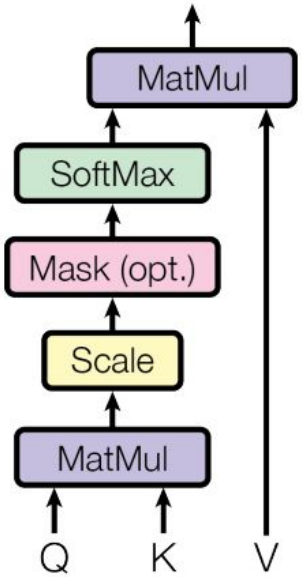
The Attention Layer



The Attention Layer



The Attention Layer

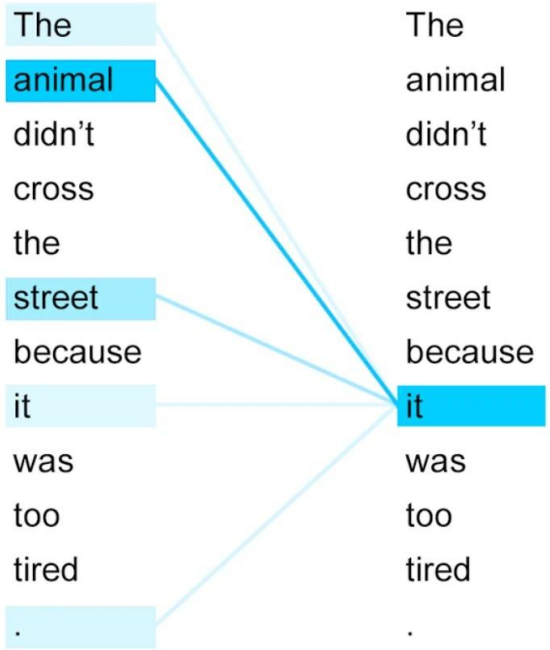


In one formula:

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

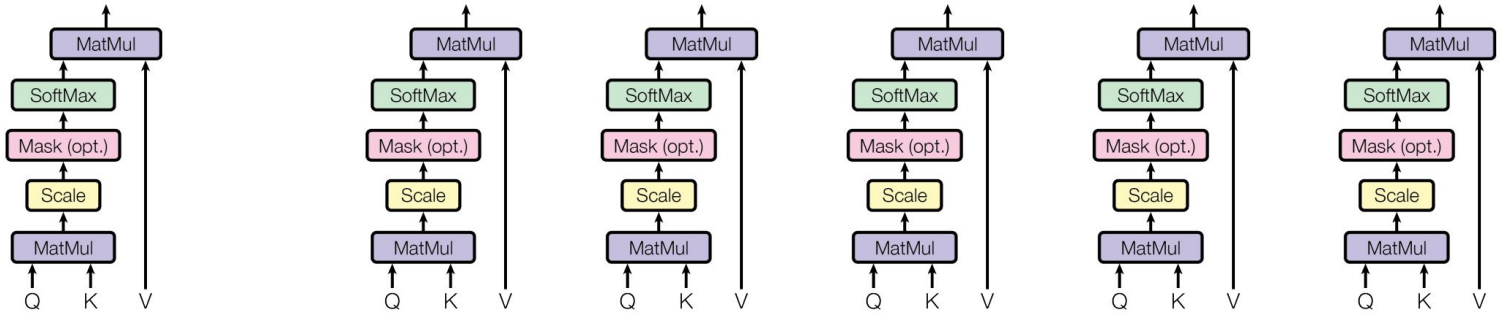
Cross-Attention vs. Self-Attention

- What we saw just now is cross attention (the attention is computed between the **prompt** (values) and the **output text** (keys).
- A **self-attention** layer uses the same tokens for both the keys and values. (how related the two tokens is)



Multi-Head Attention

Split

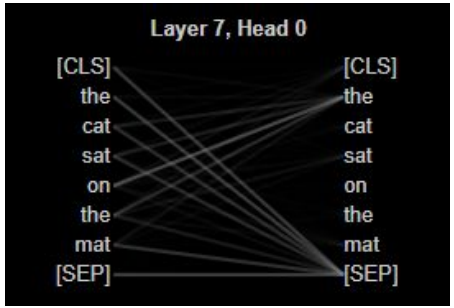


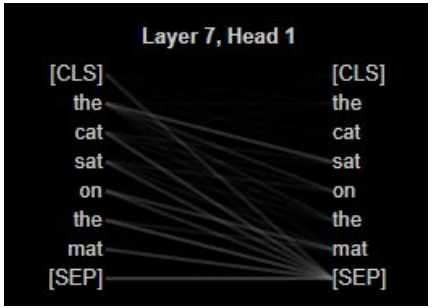
“One attention head”

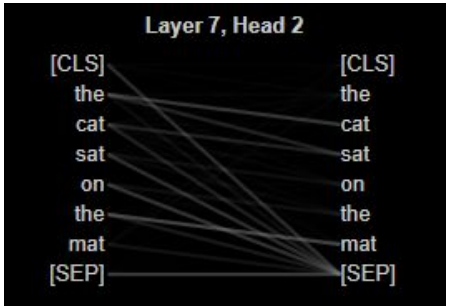
Concatenate

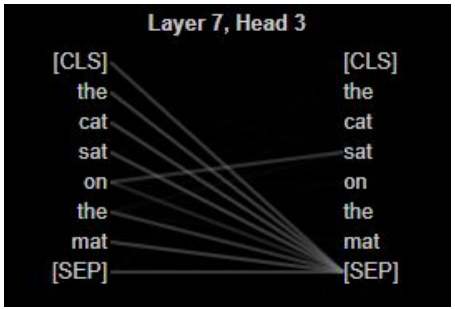
Multi-Head Attention

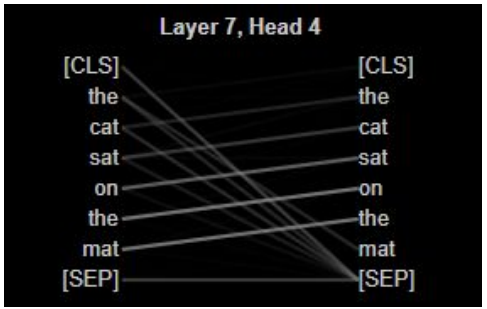
- Different attention heads focus on different things!

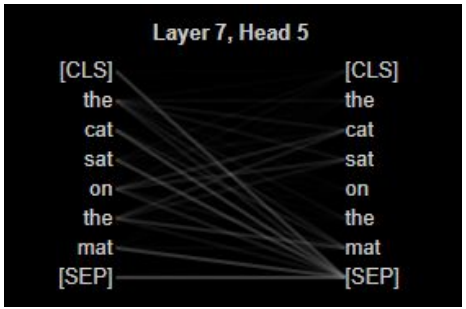












Transformers: "Attention is All You Need" (Vaswani et al. 2017)

- A crucial paper in 2017
- Architecture built entirely around self-attention and simple feed-forward networks.
  - No recurrence

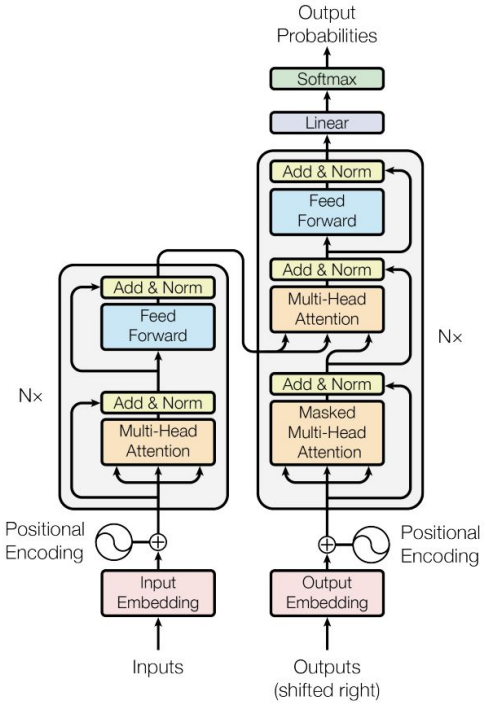
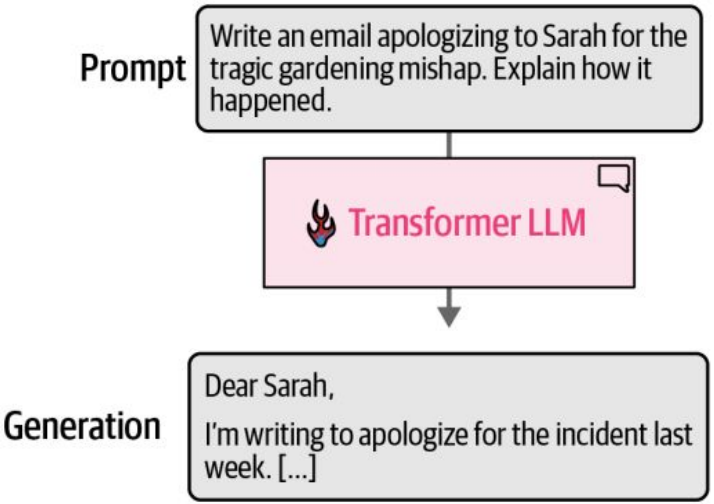


Figure 1: The Transformer - model architecture.

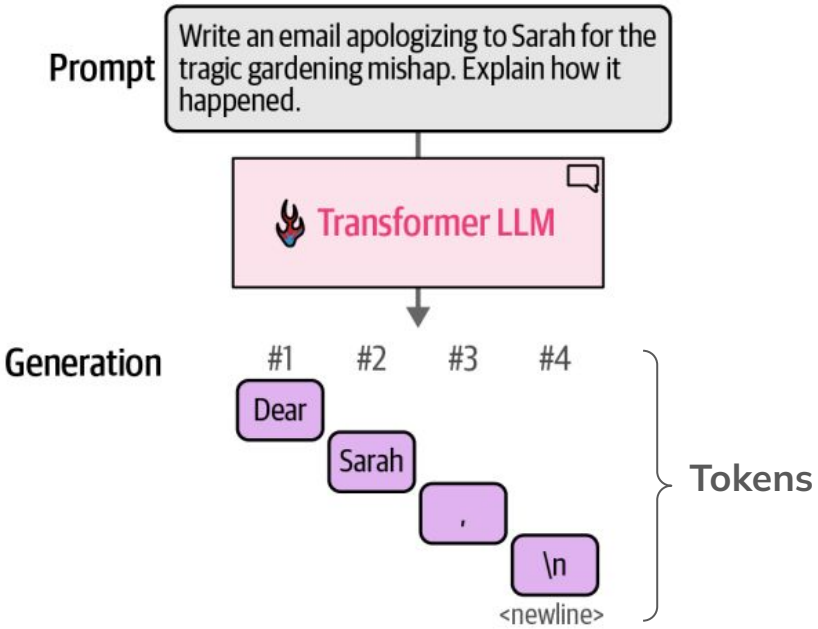
LLM from 30,000 feet



← Is this generated at once?

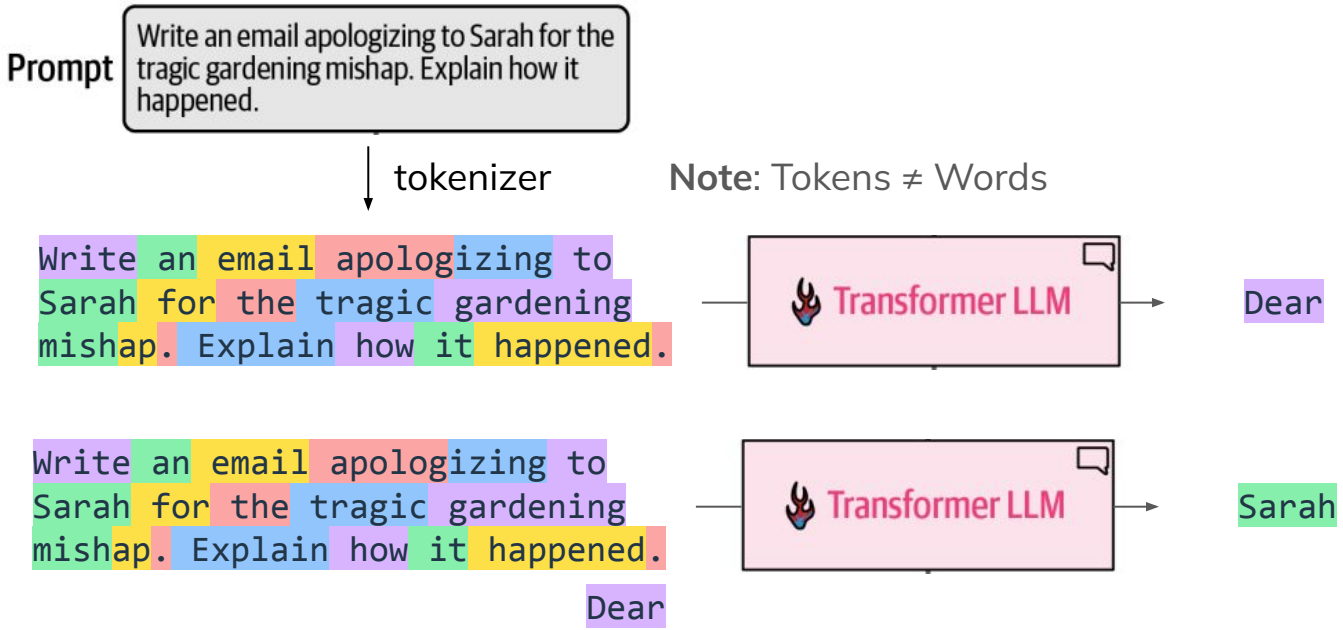
(Hands-On Large Language Models, by Jay Alammar & Maarten Grootendorst)

LLM from 25,000 feet



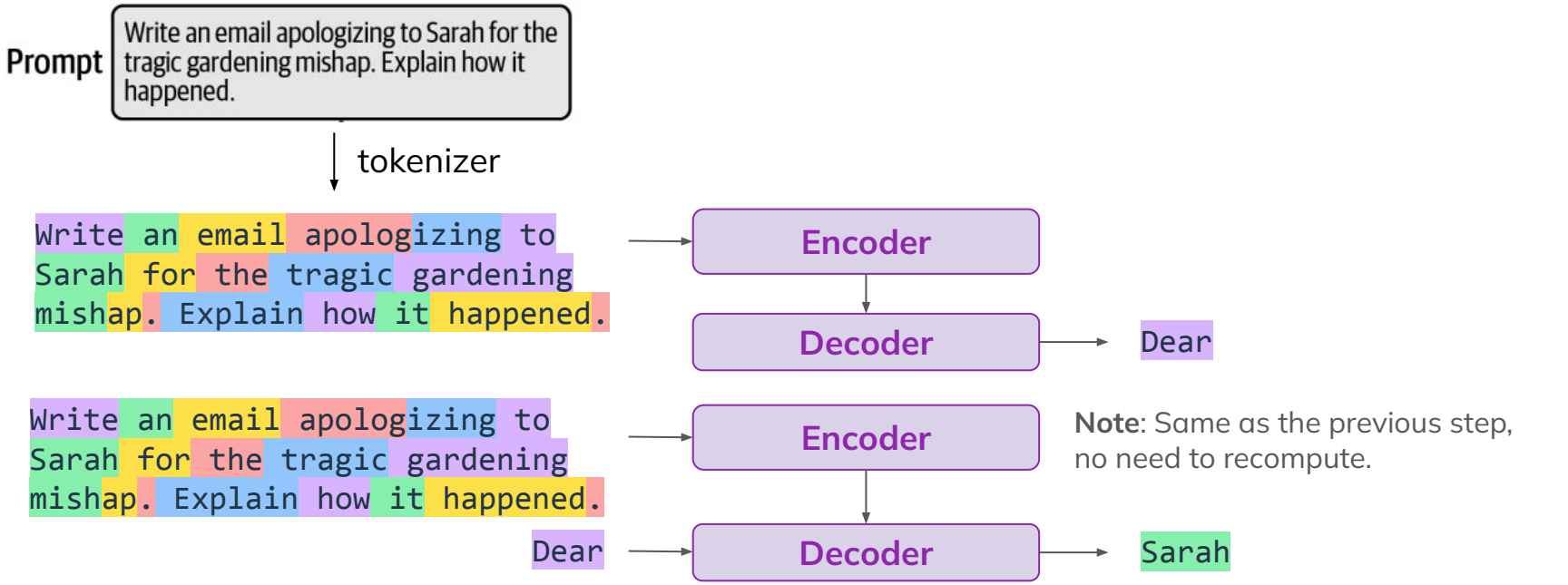
(Hands-On Large Language Models, by Jay Alammar & Maarten Grootendorst)

LLM from 20,000 feet



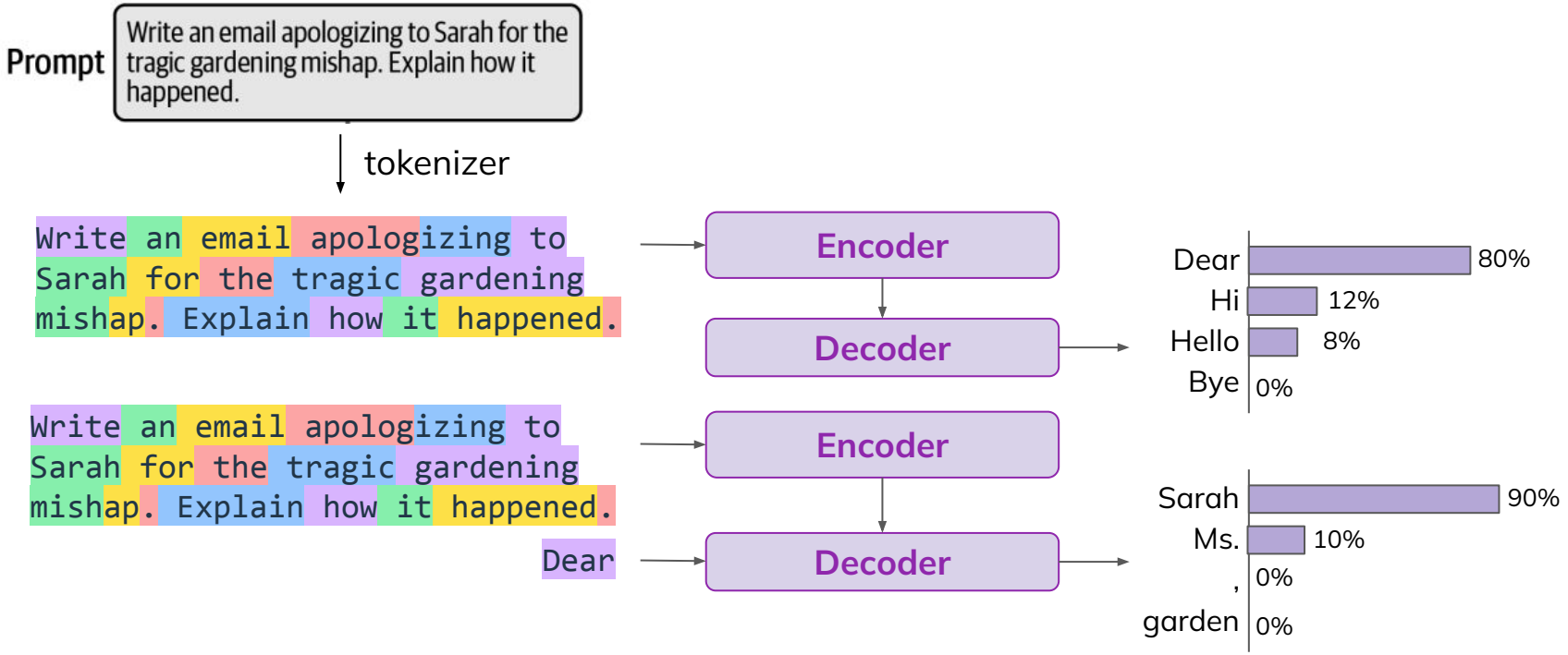
(Hands-On Large Language Models, by Jay Alammar & Maarten Grootendorst)

LLM from 15,000 feet



(Hands-On Large Language Models, by Jay Alammar & Maarten Grootendorst)

LLM from 10,000 feet



LLM from 1,000 feet

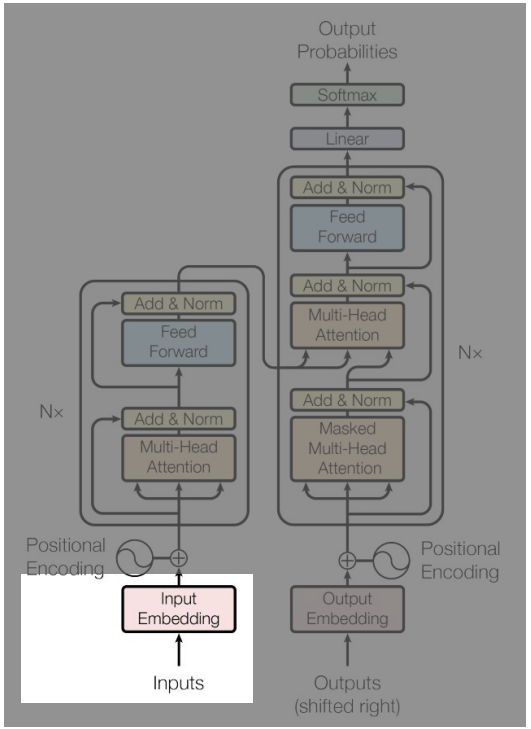
Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.

→

```
[8144, 459, 2613, 21050,
4954, 311, 21077, 369,
279, 35279, 60299, 64496,
391, 13, 83017, 1268,
433, 7077, 13]
```

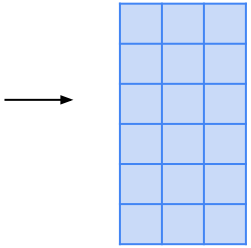
Token vocabulary

Token ID	Token
0	!
1	"
⋮	⋮
8144	Write
⋮	⋮
21050	apolog



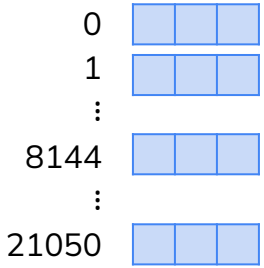
LLM from 1,000 feet

```
[8144, 459, 2613, 21050,
4954, 311, 21077, 369,
279, 35279, 60299, 64496,
391, 13, 83017, 1268,
433, 7077, 13]
```

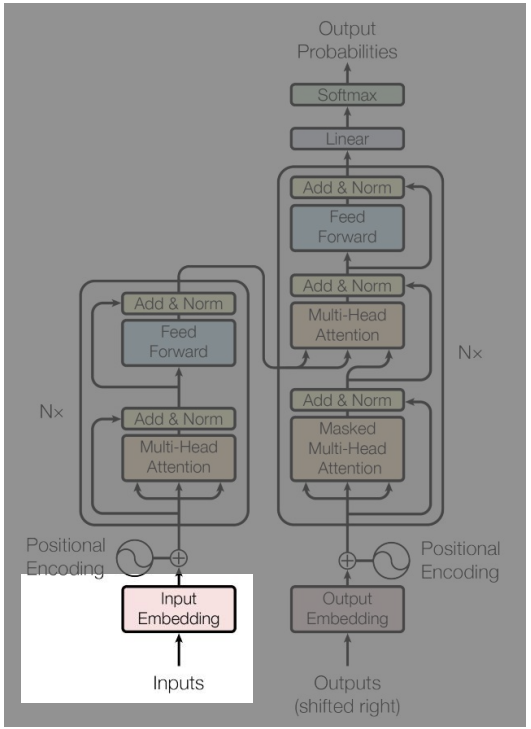


Token

Token embeddings

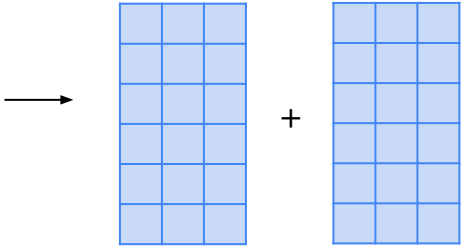


"**Hidden state**" in the Neural Network

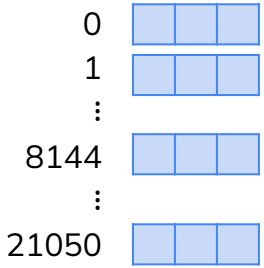


LLM from 1,000 feet

```
[8144, 459, 2613, 21050,
4954, 311, 21077, 369,
279, 35279, 60299, 64496,
391, 13, 83017, 1268,
433, 7077, 13]
```

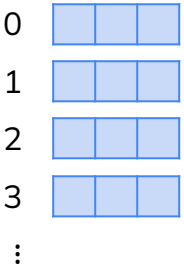


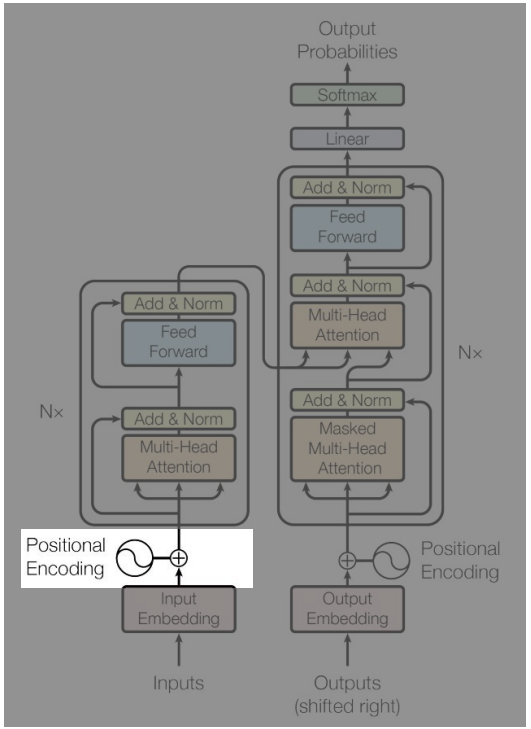
Token embeddings



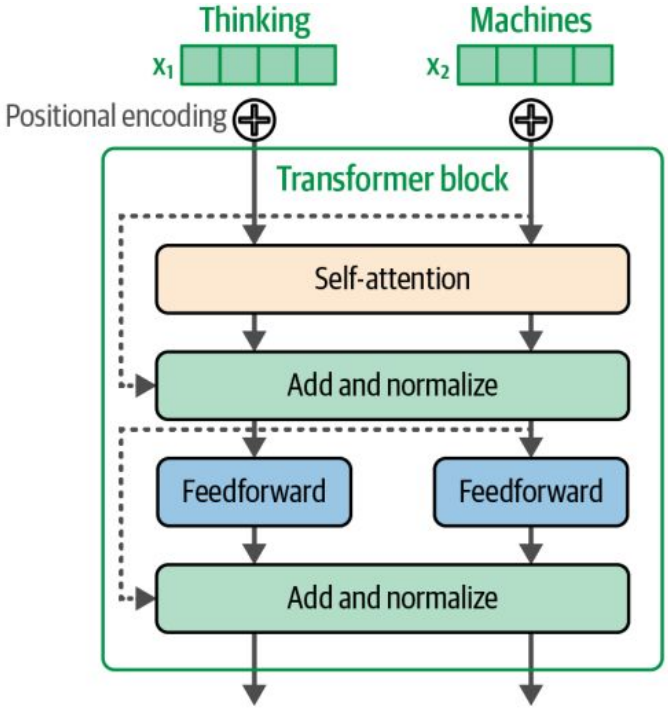
Issue: Does not capture **positional information** of the tokens.

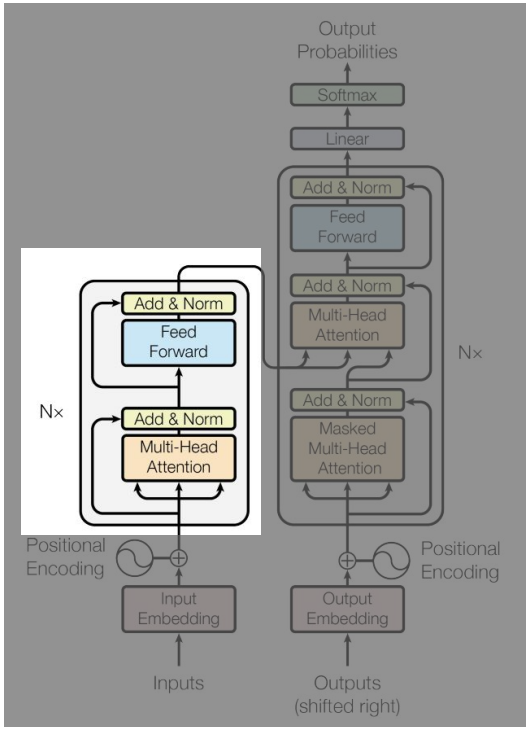
Positional encodings





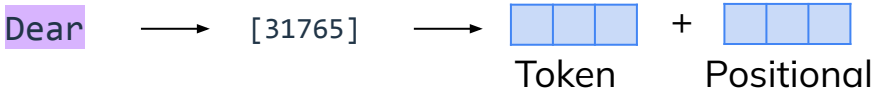
LLM from 1,000 feet





(Hands-On Large Language Models, by Jay Alammar & Maarten Grootendorst)

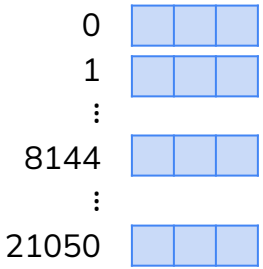
LLM from 1,000 feet



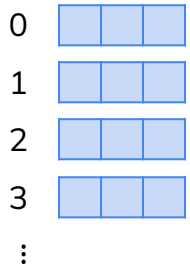
Token vocabulary

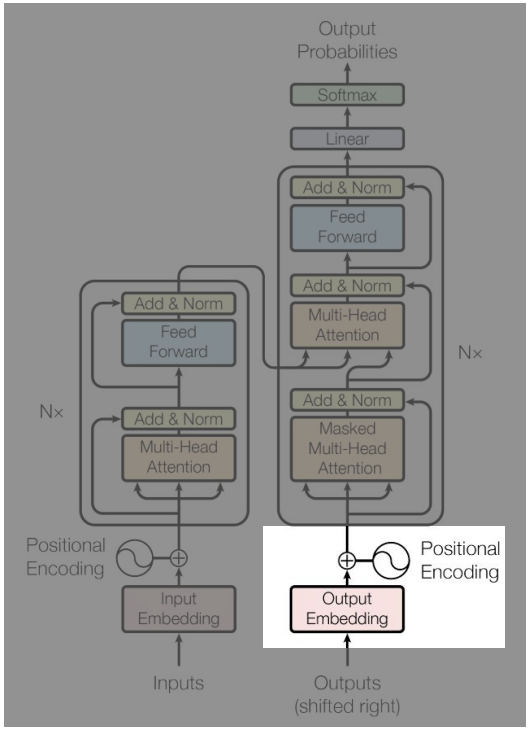
Token ID	Token
0	!
1	"
⋮	⋮
8144	Write
⋮	⋮
21050	apolog

Token embeddings



Positional encodings

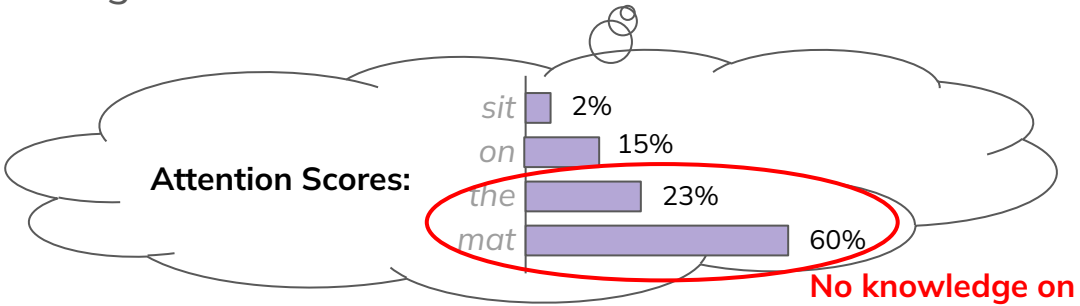




LLM from 1,000 feet

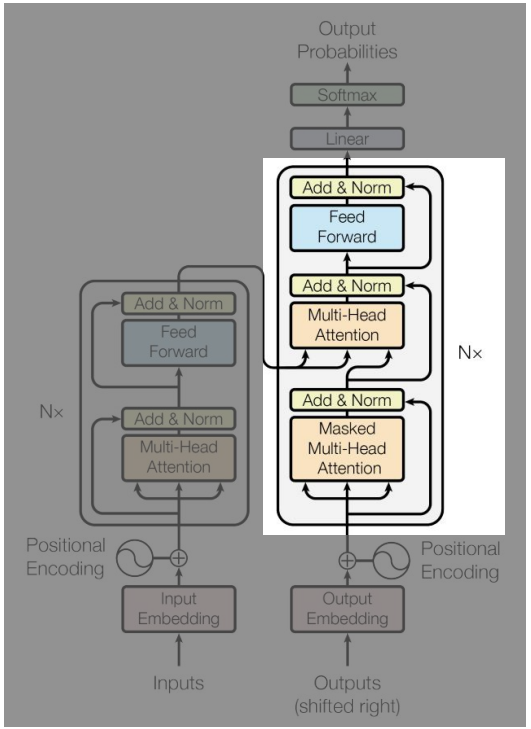
What is ***Masked*** Attention?

Training Data: The cat sit on the mat.



No knowledge on future tokens!

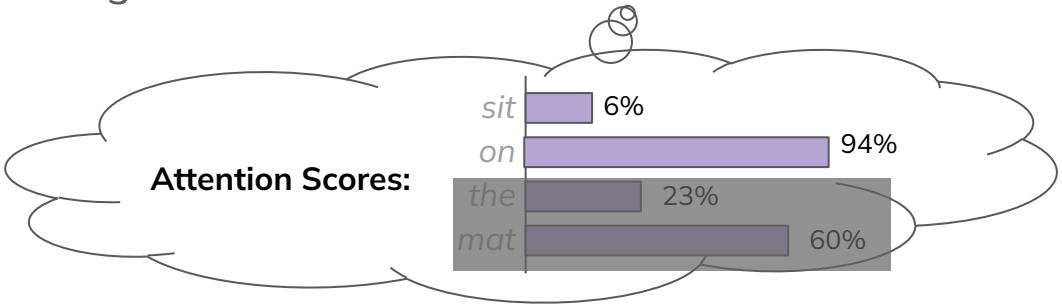
Reality: The cat sit on [generating...]



LLM from 1,000 feet

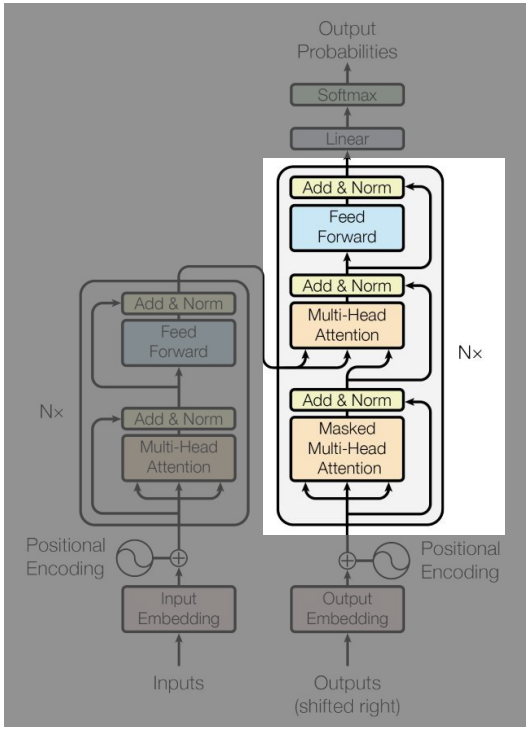
What is ***Masked*** Attention?

Training Data: `The cat sit on the mat.`

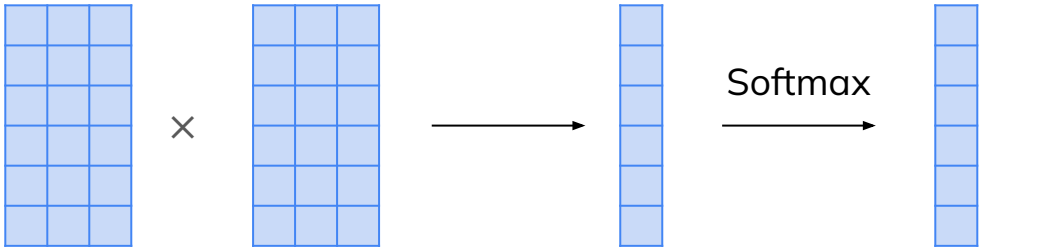


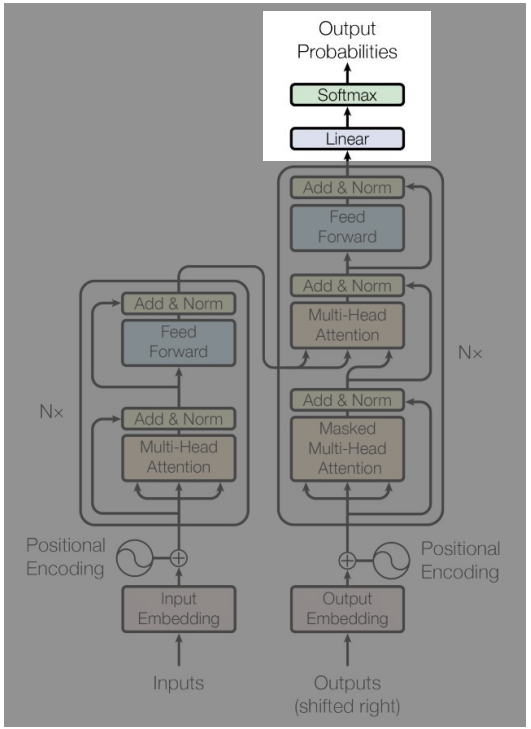
Reality: `The cat sit on` [generating...]

Still works!



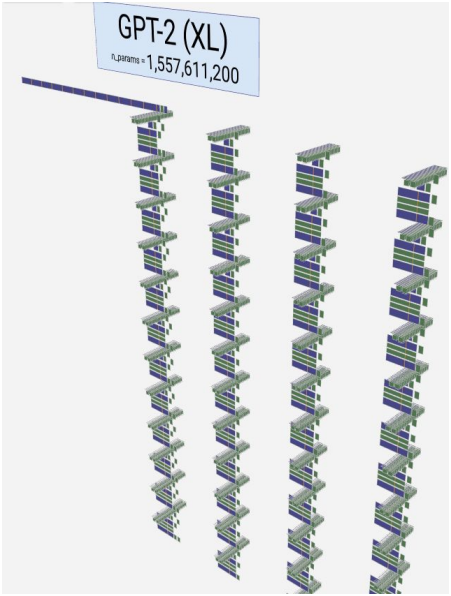
LLM from 1,000 feet



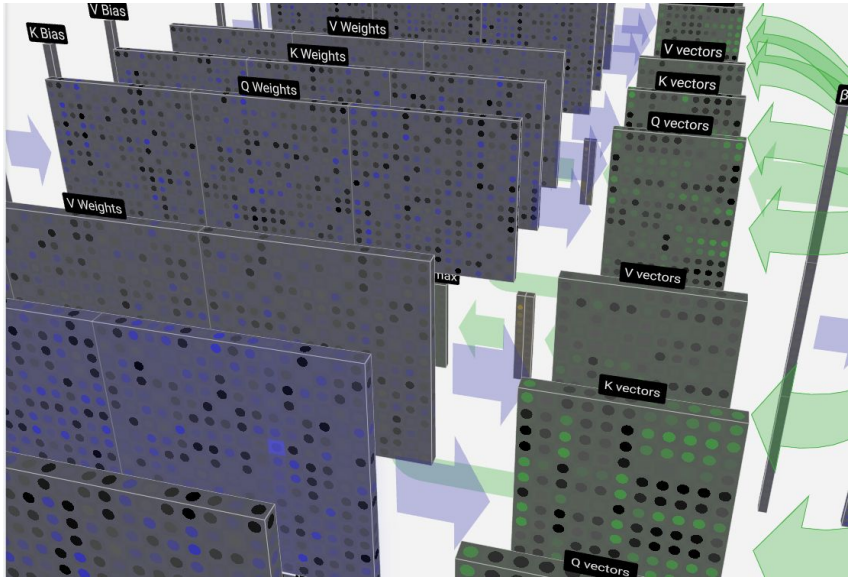


LLM in Action – GPT (Decoder-only Model)

- Visualize it at https://bbycroft.net/llm!



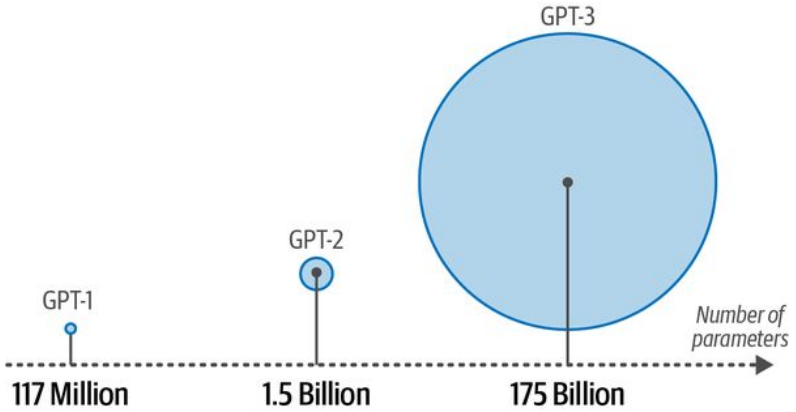
The entire GPT-2 (XL)



Self Attention Layer

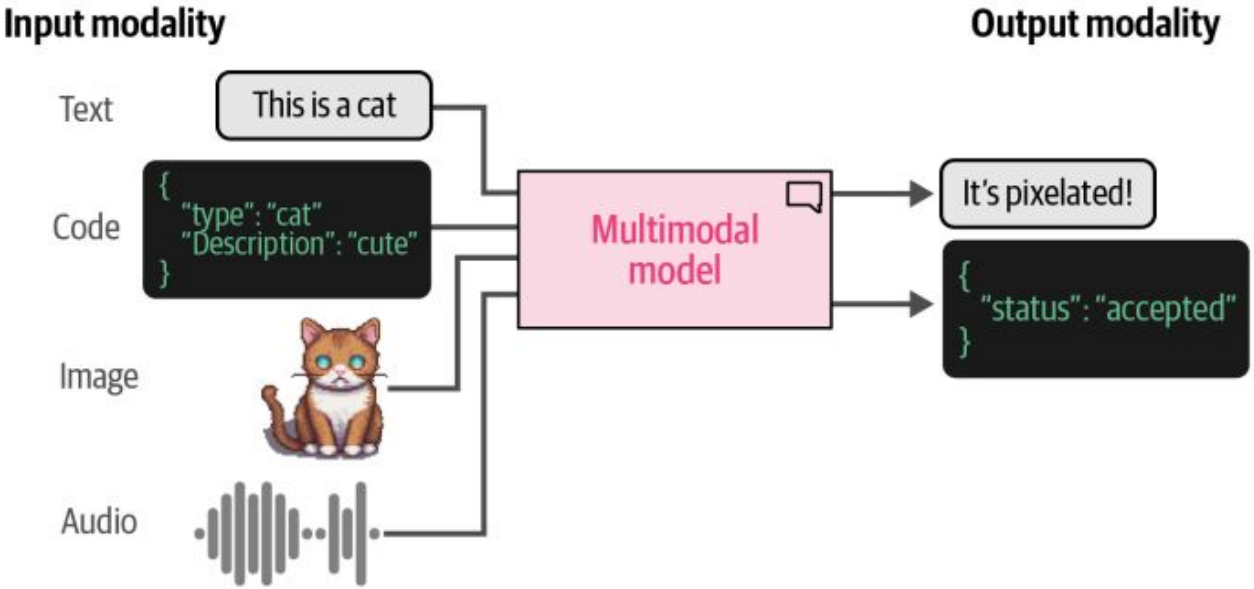
The era of Large Language Model (LLM)

- GPT-2 still output gibberish mostly, it captures human language and way of thinking **vaguely** and cannot be used directly.
- GPT-3 is a success. One major thing it is different than GPT-2 is it scale up the model params size / training data size significantly.



(Hands-On Large Language Models, by Jay Alammar & Maarten Grootendorst)

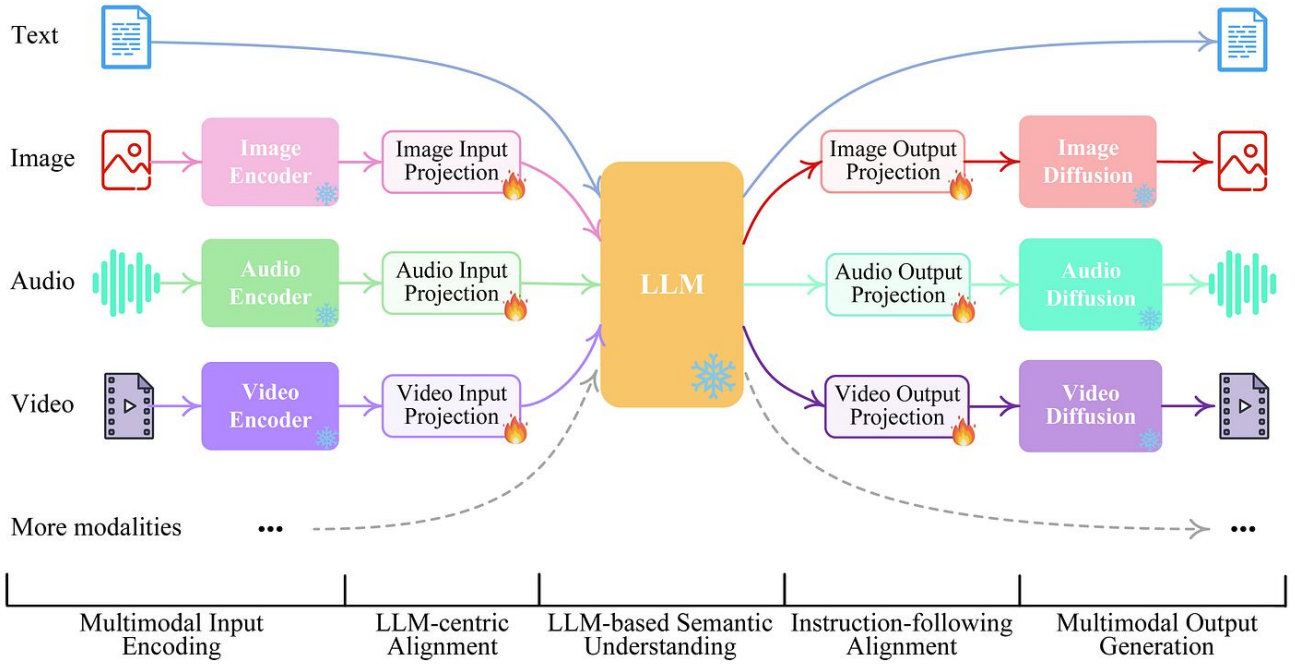
Beyond Text – Multimodal Models



(Hands-On Large Language Models, by Jay Alammar & Maarten Grootendorst)

Beyond Text – Multimodal Models

Idea: Multiple encoders



(https://medium.com/@cout.shubham/exploring-multimodal-large-language-models-a-step-forward-in-ai-626918c6a3ec)

Pretraining

Core Idea: Train a very large Transformer model on an enormous and diverse corpus of text (e.g., much of the public internet, all of Wikipedia, vast collections of books)**.**

Expected: The model learns complex representations of human language. It captures grammar, syntax, semantics, factual knowledge, and even some reasoning patterns.

Pretraining

Self-Supervised Learning: The magic is that no human-provided labels are needed for this initial, pretraining phase. The labels are derived from the data itself.

- Masked Language Modeling (MLM) (e.g. BERT-style)

  1. Take a sentence.

  2. Randomly hide (mask) about 15% of its words.

  3. Train the Transformer encoder to predict the original masked words, using all **other** words in the sentence (both left and right context, hence "bidirectional").

Pretraining

Self-Supervised Learning: The magic is that no human-provided labels are needed for this initial, pretraining phase. The labels are derived from the data itself.

- Masked Language Modeling (MLM) (e.g. BERT-style)

  1. Take a sentence.

  2. Randomly hide (mask) about 15% of its words.

  3. Train the Transformer encoder to predict the original masked words, using all **other** words in the sentence (both left and right context, hence "bidirectional").

Pretraining

Self-Supervised Learning: The magic is that no human-provided labels are needed for this initial, pretraining phase. The labels are derived from the data itself.

- Next Token Prediction (Causal Language Modeling) (e.g. GPT-style)

  1. Take a sequence of words.

  2. Train the Transformer decoder to predict the **very next word** given all the preceding words. (This is "causal" because future tokens cannot influence past predictions).

Alignment: Making LLMs Helpful

- **The Problem:** LLMs pretrained on raw internet text can learn biases, generate false information ("hallucinations"), or produce harmful/toxic content because they're just predicting statistically likely sequences.
- **The Goal (Alignment):** Make the LLM's behavior to be more aligned with human values and intentions.

Alignment Techniques

- **Instruction Tuning / Supervised Fine-Tuning (SFT):** Fine-tune the pretrained LLM on a dataset of high-quality instruction-response pairs curated by humans.
- **Reinforcement Learning from Human Feedback (RLHF):**
  - Train a "reward model": Humans rank different outputs generated by the LLM for the same prompt. A separate model is trained to predict these human preference scores.
  - Fine-tune the LLM using RL algorithms (like PPO) to maximize this reward from reward model, effectively teaching it to produce outputs that humans prefer.

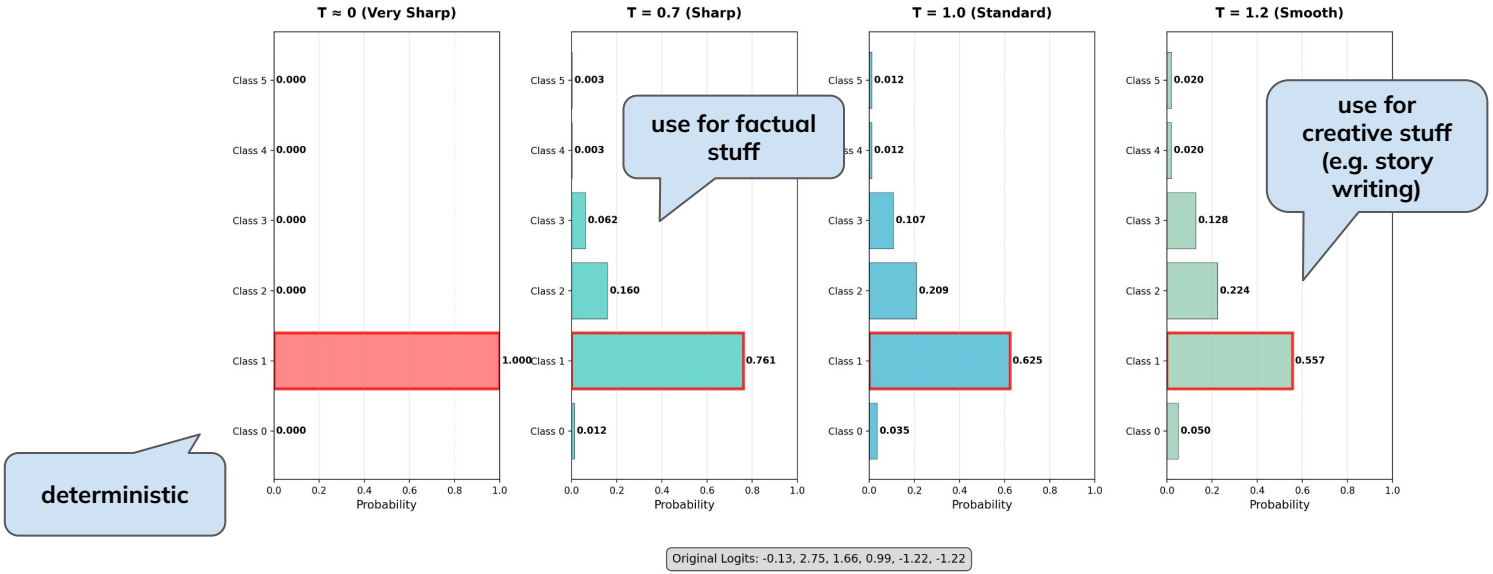
In-Context Learning

(Technically not learning in the gradient-update sense, but conditioning on the prompt to adapt behavior.)

- For today's very large LLMs (like GPT-3/4, Gemini), a new paradigm emerged. You can often get them to perform new tasks without any weight updates at all.
  - You provide a **prompt** which includes:
    - A description of the task.
    - (Optionally) A few examples of input-output pairs (this is **few-shot** learning).
    - (Optionally) Just one example (**one-shot**).
    - (Optionally) No examples, just the task description (**zero-shot**).
  - The LLM then "completes" the prompt by generating the desired output. It "learns" the task from the context provided in the prompt.

Controlling Text Generation

- **Temperature:** A parameter that scales the logits (raw scores before softmax) of the output distribution.



Controlling Text Generation

Top-p Sampling:

- Instead of sampling from the entire vocabulary, sample only from the smallest set of tokens whose cumulative probability exceeds a threshold p (e.g., p=0.9).
- This prunes the long tail of unlikely, nonsensical words, improving coherence, especially with higher temperatures.

Conclusion & The Future

- **Key Advancement:**
  - The power of **learned representations** over hand-crafted features.
  - The impact of **scale** (data, model size, compute).
- **What was state-of-the-art 5 years ago (or even 1 year ago) might be old news now.**
  - The field is fast-moving and you have to continuously learn
  - **The general public is still lagging behind in adoption the latest tech**
  - Potentially lucrative career path

References

- Deep Neural Network (2017)
  https://assets.hkoi.org/training2017/nn.pdf
- 3Blue1Brown Neural Networks Lesson
  https://www.3blue1brown.com/topics/neural-networks
  Good for building intuition and concepts understanding
- The Illustrated Transformer
  https://jalammar.github.io/illustrated-transformer
- Dive into Deep Learning
  https://d2l.ai/
  Very comprehensive book for learning from the ground up

Workshop

Workshop: Train your own AI face recognizer

- In 2017 NN lecture, we trained a classifier to recognize faces of GFRIEND member.

- Training from scratch actually requires quite a lot data.
- Let's demonstrate some modern (2025) training techniques by training a AI model that **recognize your face**!

Workshop: Train your own AI face recognizer

- In practice, we could use less data on our own if we train our AI model by **fine-tuning** some powerful pretrained image model.
- It takes much shorter time to finish training as well.

Workshop: Train your own AI face recognizer

- For example, a MobileNetV2 Model (a lightweight model by Google for running on edge devices) trained on ImageNet classification.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

https://arxiv.org/pdf/1801.04381

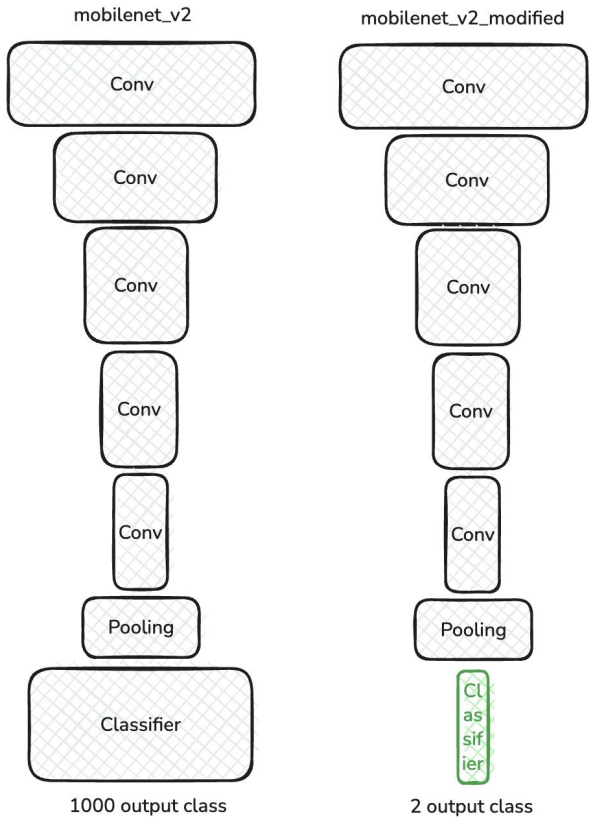
Workshop: Train your own AI face recognizer

- For example, a MobileNetV2 Model (a lightweight model by Google for running on edge devices) trained on ImageNet classification.
- We can, quite easily, use the model implemented by the PyTorch library, and load in pretrained weight that other have published:

```
model = models.mobilenet_v2(weights=models.MobileNet_V2_Weights.IMAGENET1K_V1)
```

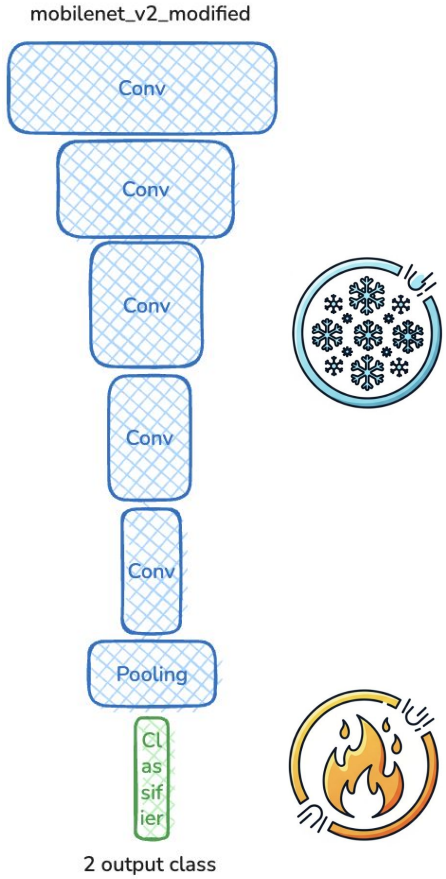
Workshop: Train your own AI face recognizer

- We can slightly edit the model to replace the 1000-class by a 2-class classifier (my_face vs not_my_face)
- This is quite easy to do (just replace classifier head):
  - `num_ftrs = model.classifier[1].in_features`
  - `model.classifier[1] = nn.Linear(num_ftrs, 2)`



Workshop: Train your own AI face recognizer

- We can freeze the original mobilenetv2 backbone weights (not updating it by gradient).
- Then we can just train the model but only update the **classifier head's weights**.



Workshop: Train your own AI face recognizer

- The main philosophy here is called **Transfer Learning**
- Knowledge gained by classifying images can be applied when recognizing face.

Where's the knowledge gained is stored?

- Earlier layers: simple image feature
- End of backbone: high level abstracted understanding
- Head: use the feature to perform tasks

Workshop: Train your own AI face recognizer

- Let's open Google COLAB: https://colab.research.google.com/
  - A way to run training on cloud and use google drive as storage
  - Easier alternative than setting up a training environment locally
  - Can use free GPU to train
- Find the training notebook here: https://github.com/hkoi/neural-network-2025

- You would need to prepare your own dataset. Follow the instructions in the notebook.

Workshop: Train your own AI face recognizer

Key concepts to learn

- Dataset curation
- Dataset bias
- Data Augmentation
- Understanding graphs