Visual Computing Institute

RWTH AACHEN UNIVERSITY

Elements of Machine Learning & Data Science

Winter semester 2025/26

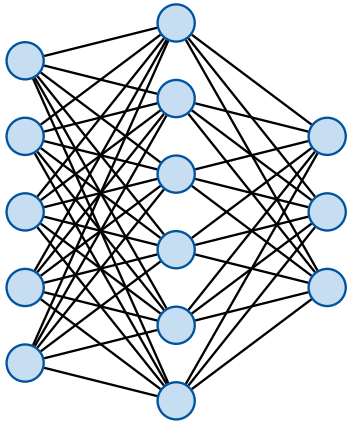
Lecture 17 – Neural Networks Basics II

22.12.2025

Prof. Bastian Leibe

Machine Learning Topics

8. Introduction to ML
9. Probability Density Estimation
10. Linear Discriminants
11. Linear Regression
12. Logistic Regression
13. Support Vector Machines
14. **Neural Network Basics**



Multi-Layer Perceptrons

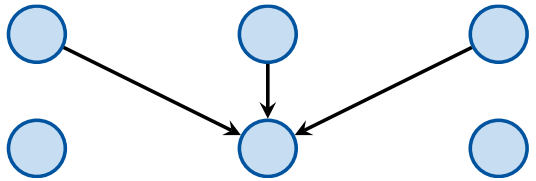
$$\frac{1}{2}(y(\mathbf{x}) - t)^2$$

$$[1 - ty(\mathbf{x})]_+$$

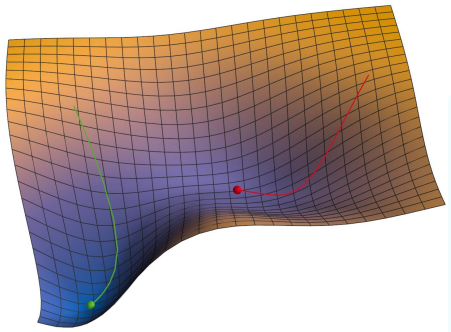
$$-\sum_k \left(\mathbb{I}(t = k) \ln \frac{\exp(y_k(\mathbf{x}))}{\sum_j \exp(y_j(\mathbf{x}))} \right)$$

$$\frac{1}{2}\|\mathbf{w}\|^2$$

Losses & Regularizers



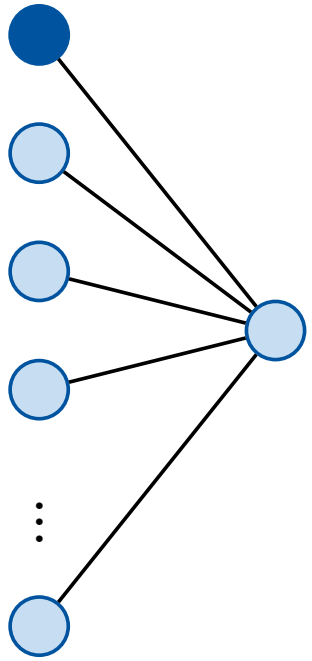
Backpropagation



Stochastic Gradient Descent

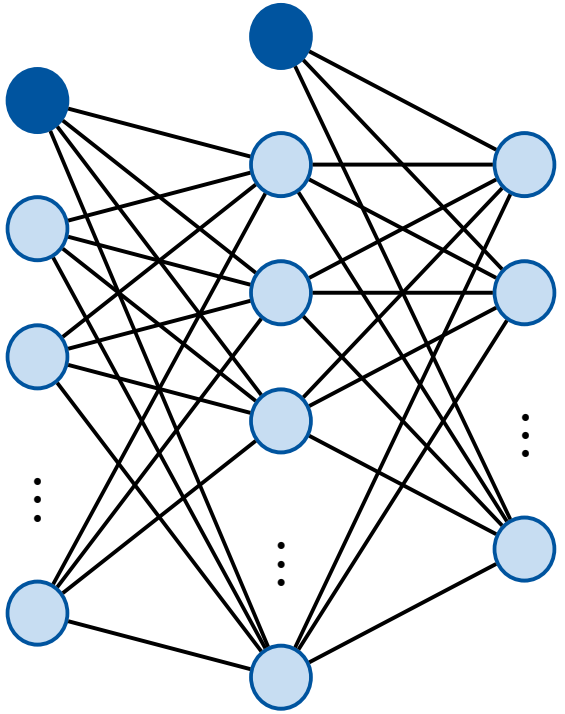
Neural Network Basics

1. **Perceptrons**
2. Multi-Layer Perceptrons
3. Loss Functions
4. Backpropagation
5. Computational Graphs
6. Stochastic Gradient Descent
7. History of Neural Networks
8. Deep Learning's Main Breakthroughs



Neural Network Basics

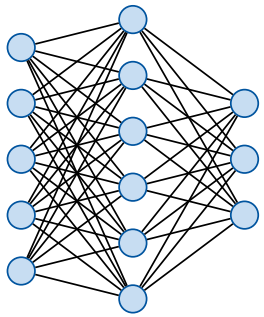
1. Perceptrons
2. **Multi-Layer Perceptrons**
3. Loss Functions
4. Backpropagation
5. Computational Graphs
6. Stochastic Gradient Descent
7. History of Neural Networks
8. Deep Learning's Main Breakthroughs



Recap: Learning with Hidden Units

- We now have a model that contains multiple layers of adaptive non-linear hidden units.
- How can we train such models?
  - Need to train *all* weights, not just last layer.
  - Learning the weights to the hidden units = learning features.
  - We don't know what the hidden units should do.
- Basic Idea: Gradient Descent.

This is the main challenge of deep learning!



$E(\mathbf{w})$

Apply Error Function

$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$

Calculate Gradients

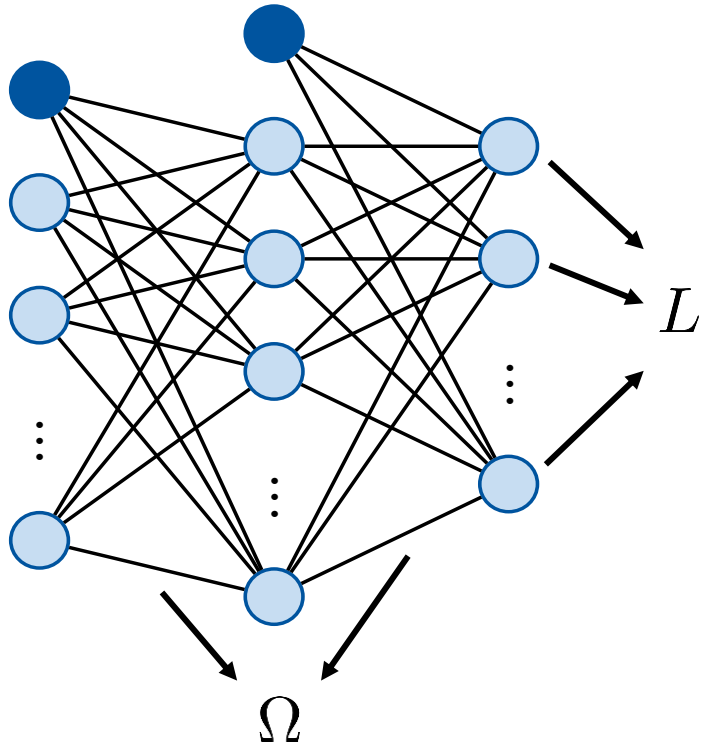
$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$

Update Weights

Repeat until convergence.

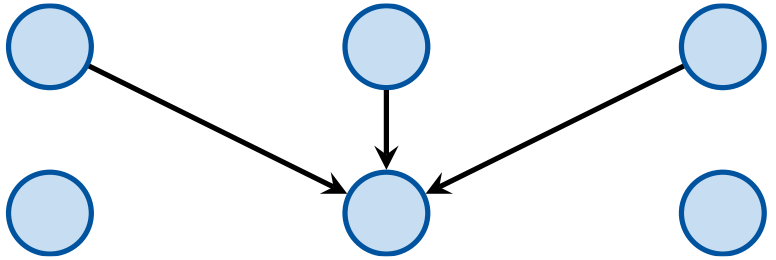
Neural Network Basics

1. Perceptrons
2. Multi-Layer Perceptrons
3. **Loss Functions**
4. Backpropagation
5. Computational Graphs
6. Stochastic Gradient Descent
7. History of Neural Networks
8. Deep Learning's Main Breakthroughs

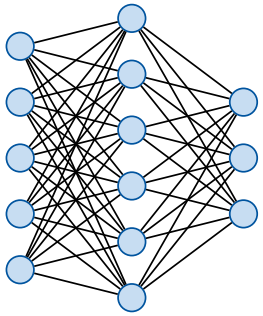


Neural Network Basics

1. Perceptrons
2. Multi-Layer Perceptrons
3. Loss Functions
4. **Backpropagation**
5. Computational Graphs
6. Stochastic Gradient Descent
7. History of Neural Networks
8. Deep Learning's Main Breakthroughs



Backpropagation



$E(\mathbf{w})$

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

Apply model to training data

Evaluate error function

Compute gradients

Update weights

Repeat

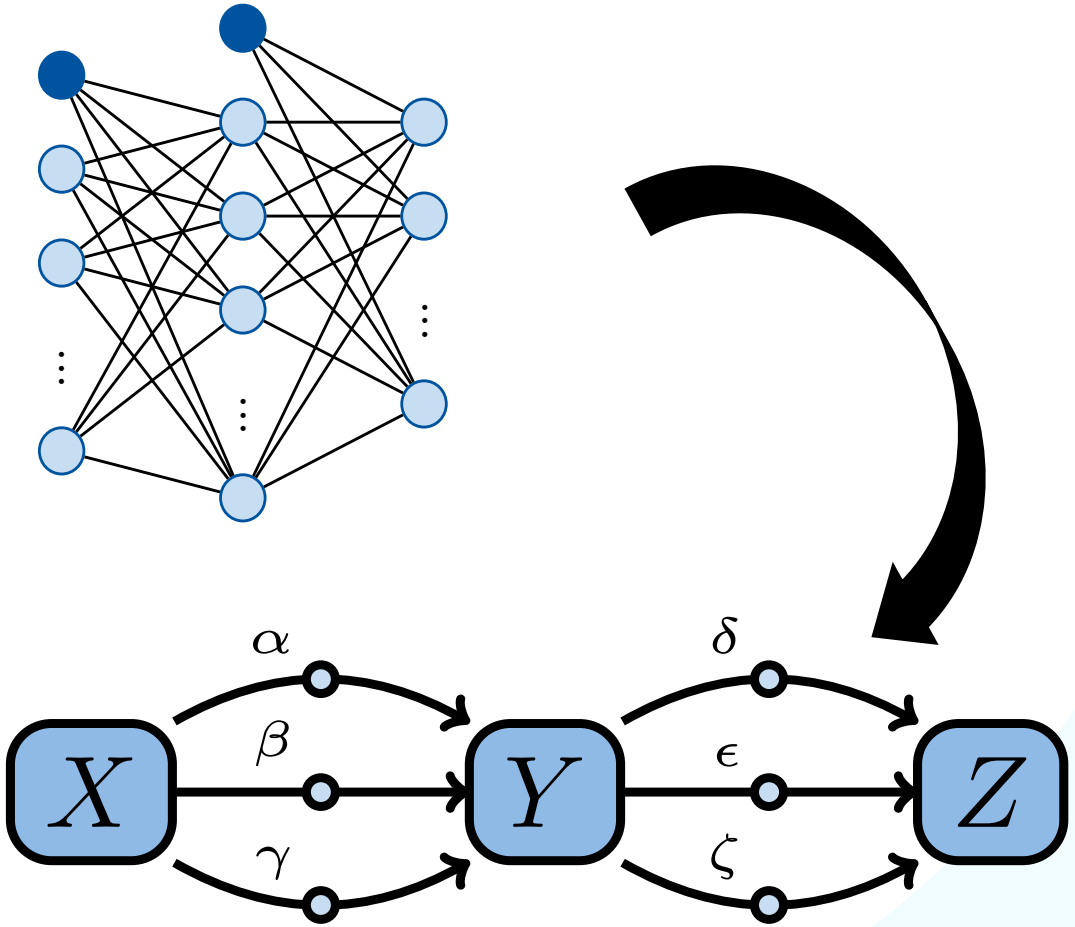
Forward pass

Backward pass

Implemented via
automatic differentiation

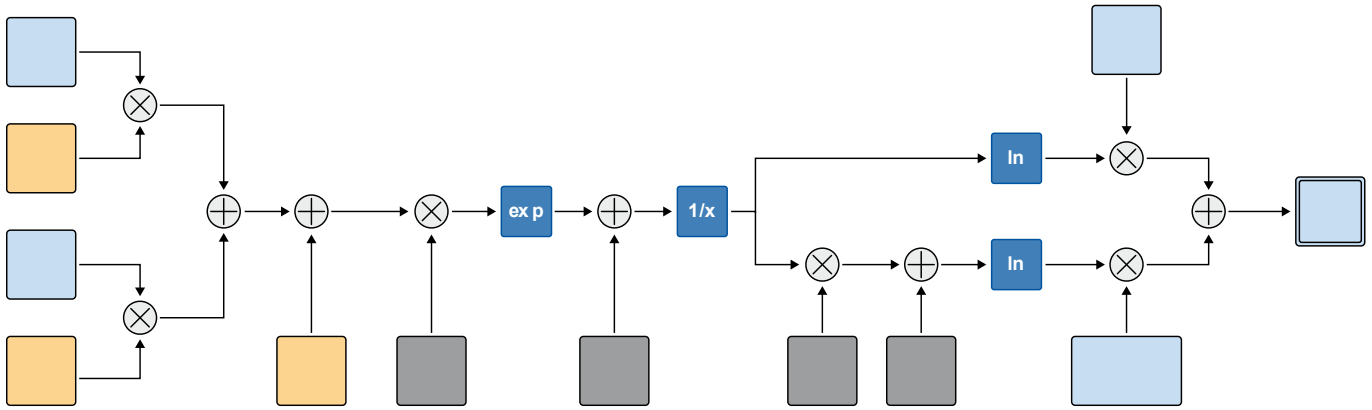
Automatic Differentiation

- Convert the network into a computational graph.
- Each layer/module just needs to specify how it affects the forward and backward passes.
- Apply backpropagation on the computational graph.
- Very general algorithm, used in today's Deep Learning packages.



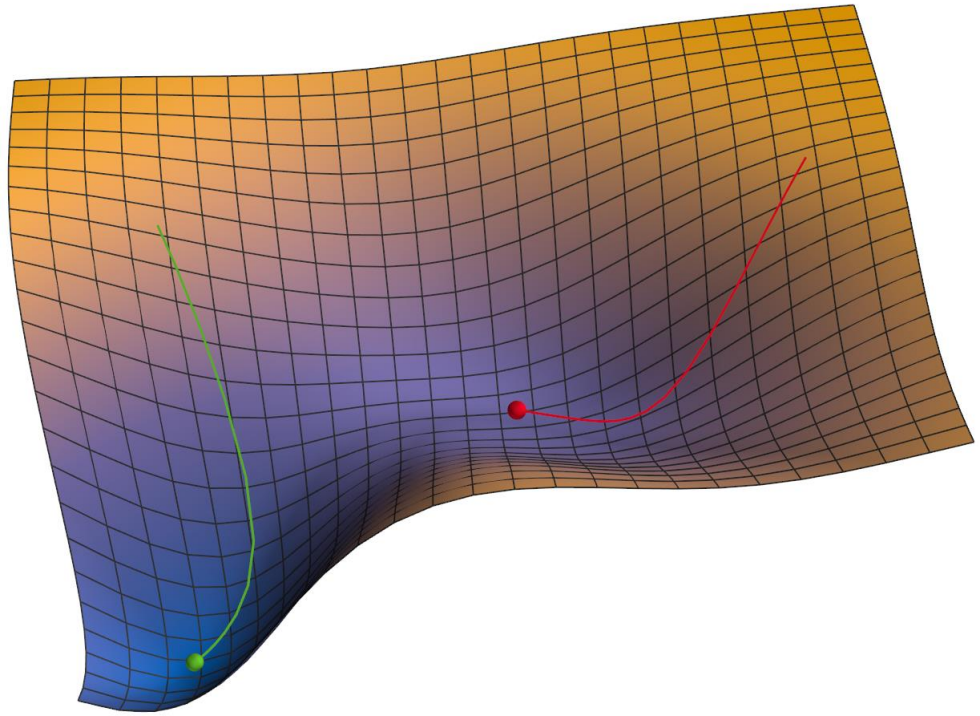
Neural Network Basics

1. Perceptrons
2. Multi-Layer Perceptrons
3. Loss Functions
4. Backpropagation
5. **Computational Graphs**
6. Stochastic Gradient Descent
7. History of Neural Networks
8. Deep Learning's Main Breakthroughs



Neural Network Basics

1. Perceptrons
2. Multi-Layer Perceptrons
3. Loss Functions
4. Backpropagation
5. Computational Graphs
6. **Stochastic Gradient Descent**
7. History of Neural Networks
8. Deep Learning's Main Breakthroughs

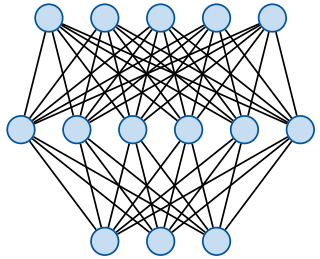


Stochastic Gradient Descent

- Now that we have the gradients, we need to update the weights.
- We already know the basic equation for this
  - (1st-order) Gradient Descent

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

- Remaining Questions:
  - On what data do we want to apply this?
  - How should we choose the learning rate η ?



Apply Error Function

Calculate Gradients

Update Weights

Stochastic vs. Batch Learning

- Batch Learning
  - Process the full dataset in one batch.
  - Compute the gradient based on all training examples.

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

- Stochastic Learning
  - Choose a single example from the training set.
  - Compute the gradient only based on this example.
  - This estimate will generally be noisy, which has some advantages.

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E_n(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

Batch Learning

- Conditions of convergence are well understood.
- Many acceleration techniques only work in batch learning.
- Theoretical analysis of the weight dynamics and convergence rates are simpler.

Stochastic Learning

- Usually much faster than batch learning.
- Often results in better solutions.
- Can be used for tracking changes when the target distribution shifts.

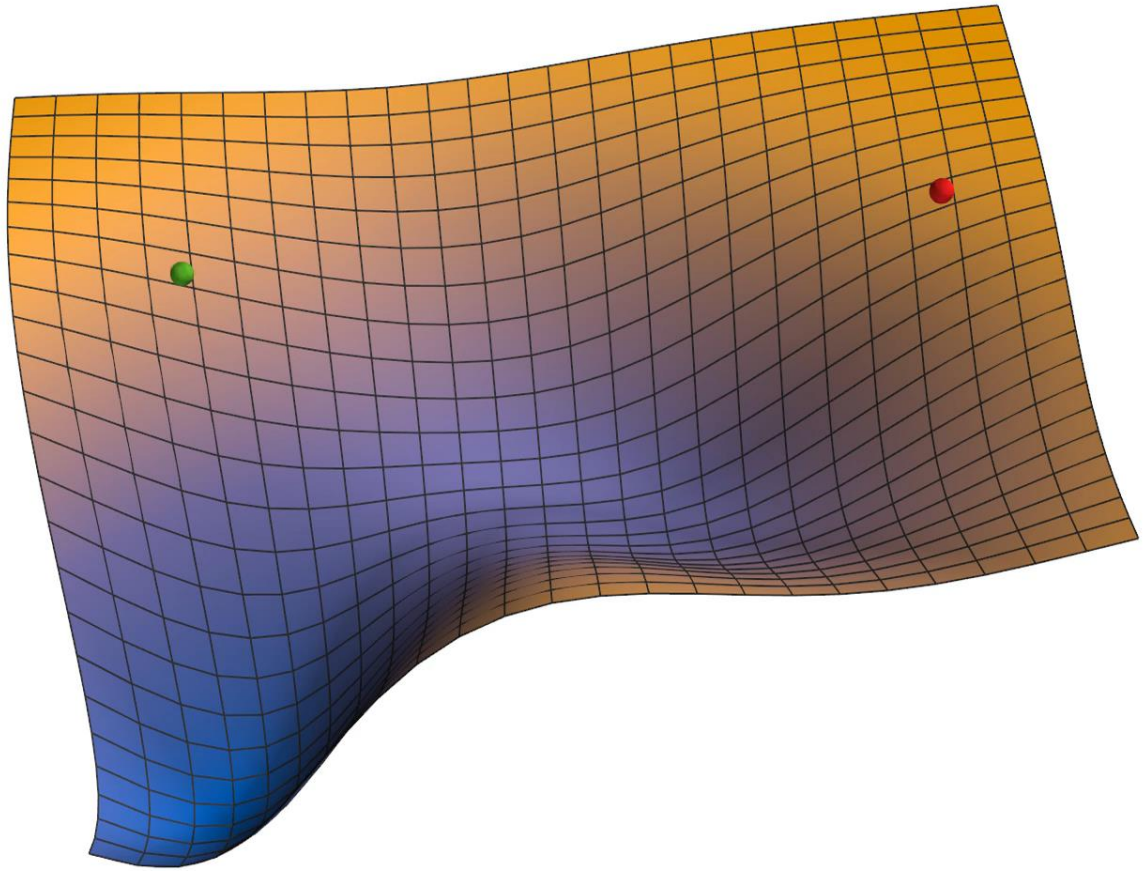
Middle ground: Minibatches

Minibatches

- Idea
  - Process only a small batch of training examples together. $\mathcal{B} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_B, t_B)\} \subset \mathcal{D}$
  - Start with a small batch size & increase it as training proceeds.
- Advantages
  - Gradients will be more stable than for stochastic gradient descent, but still faster to compute than with batch learning.
  - Take advantage of redundancies in the training set.
  - Matrix operations are more efficient than vector operations.
- Caveat
  - Need to normalize error function by the minibatch size to use the same learning rate between minibatches

$$E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} L(t_n, y(\mathbf{x}_n; \mathbf{w})) + \frac{\lambda}{N} \Omega(\mathbf{w})$$

Example

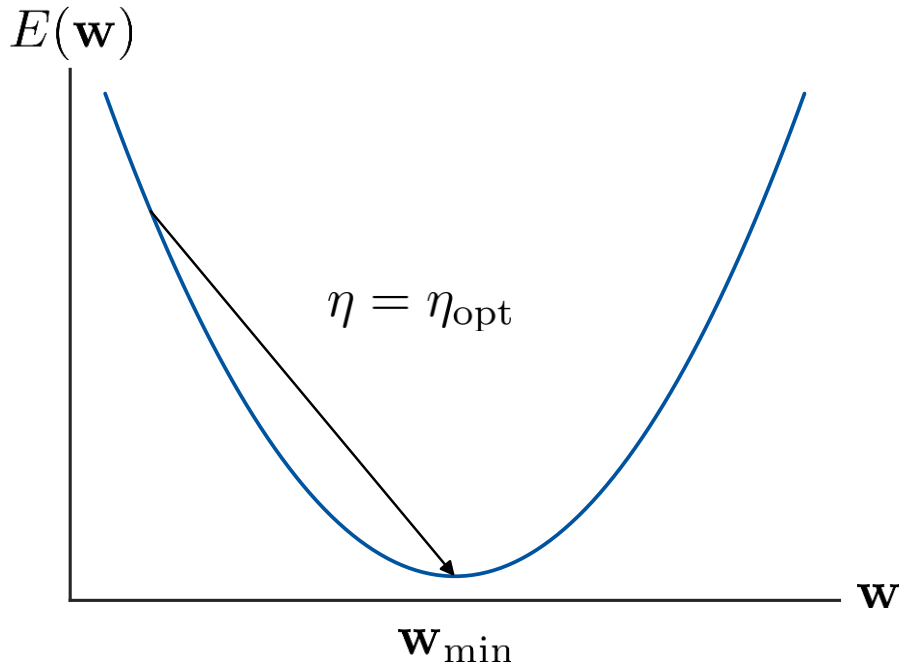


Choosing the Right Learning Rate

- Consider a simple 1D example:

$$w^{(\tau+1)} = w^{(\tau)} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

- What is the optimal learning rate η_{opt}?

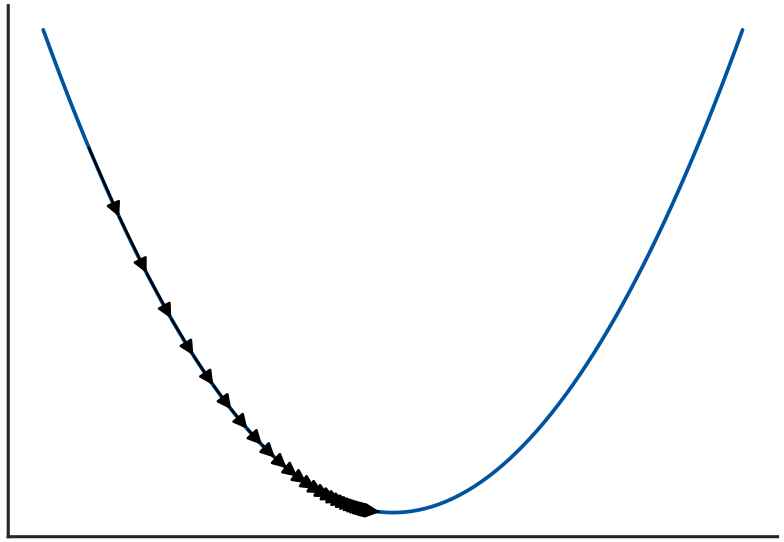


- If E is quadratic, the optimal learning rate is given by the inverse of the Hessian:

$$\eta_{\text{opt}} = \left(\frac{\partial^2 E(\mathbf{w}^{(\tau)})}{\partial \mathbf{w}^2} \right)^{-1}$$

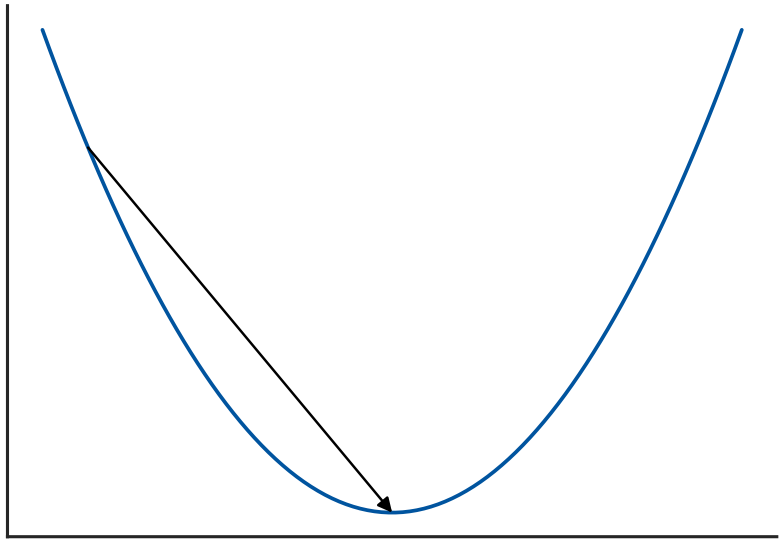
- *For neural networks, the Hessian is usually infeasible to compute.*

η too small



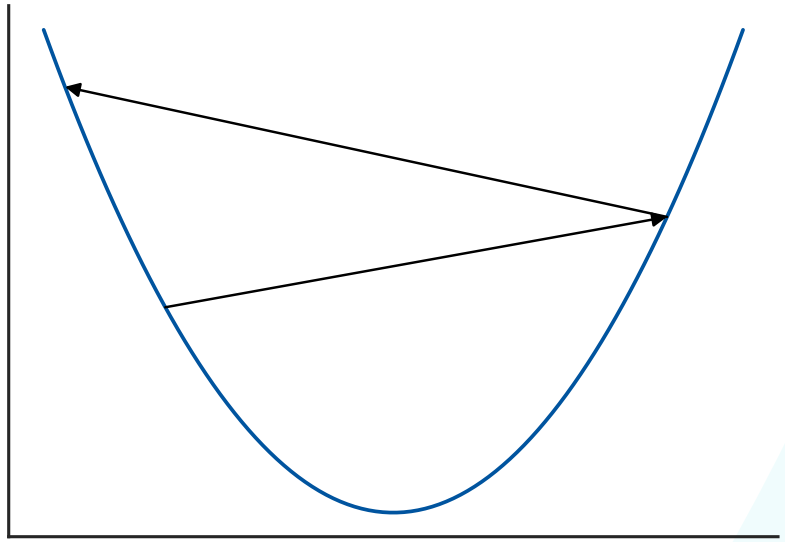
Convergence is slow

η_{opt}



Converges ideally in a single step

η too large



Might not converge

Discussion: Stochastic Gradient Descent

Advantages

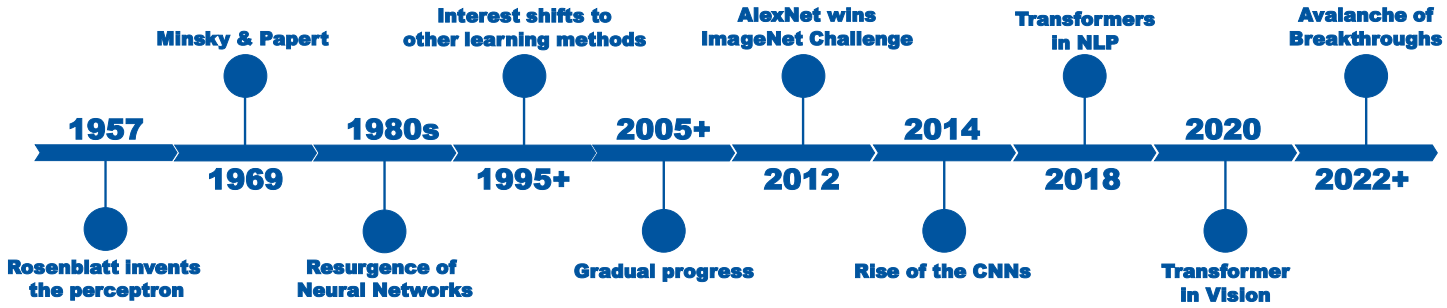
- Very simple, but still quite robust method.
- Minibatches offer a compromise between stability and faster computation.
- Stochasticity in minibatches is often beneficial for learning

Limitations

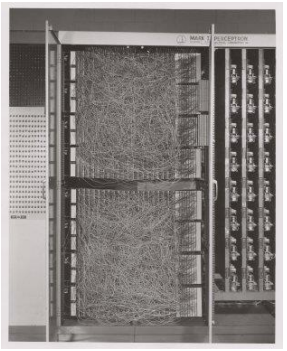
- Finding a good setting for the learning rate is very important for fast convergence.
  - Choosing the right learning rate is a challenge and requires experience.
  - A different learning rate may be optimal for different parts of the network
- Following the direction of steepest descent is not always the fastest way to the optimum
  - E.g., in highly correlated data

Neural Network Basics

1. Perceptrons
2. Multi-Layer Perceptrons
3. Loss Functions
4. Backpropagation
5. Computational Graphs
6. Stochastic Gradient Descent
7. **History of Neural Networks**
8. Deep Learning‘s Main Breakthroughs

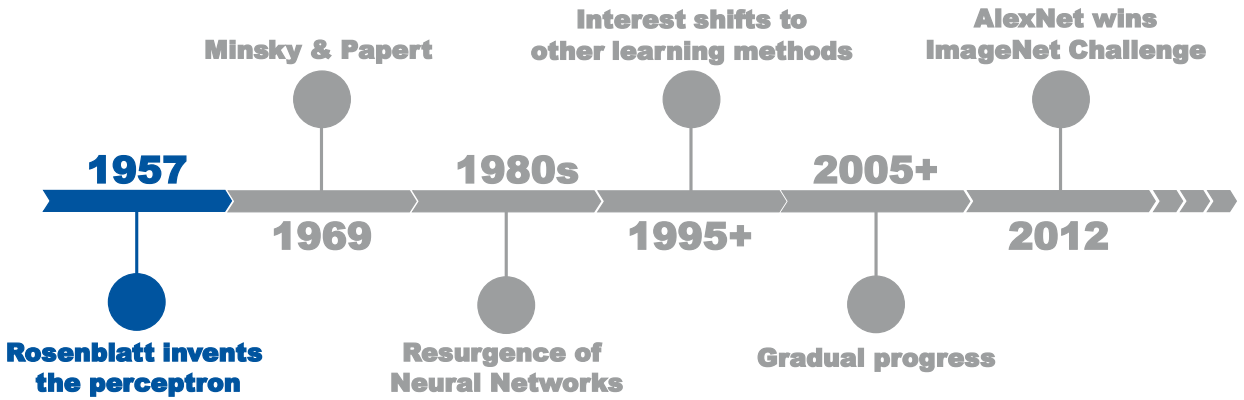


A Brief History of Neural Networks



The Beginnings

- Rosenblatt invents the Perceptron and a cool learning algorithm: “Perceptron Learning”
- Hardware implementation “Mark I Perceptron” for 20x20 pixel image analysis



The New York Times

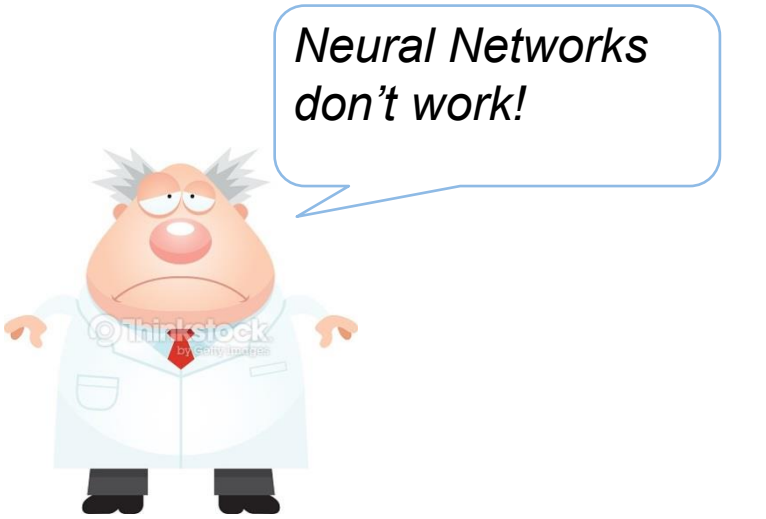
“The embryo of an electronic computer that [...] will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”

Image source: Wikipedia, clipartpanda.com

A Brief History of Neural Networks

Criticism of Perceptrons

- Minsky & Papert show that (single-layer) Perceptrons cannot solve all problems
- This is misunderstood by many that they are worthless



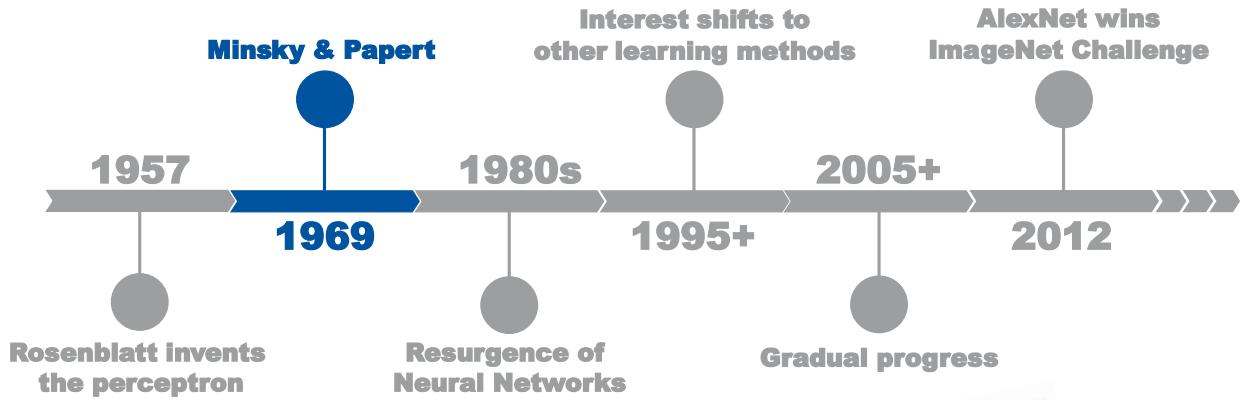
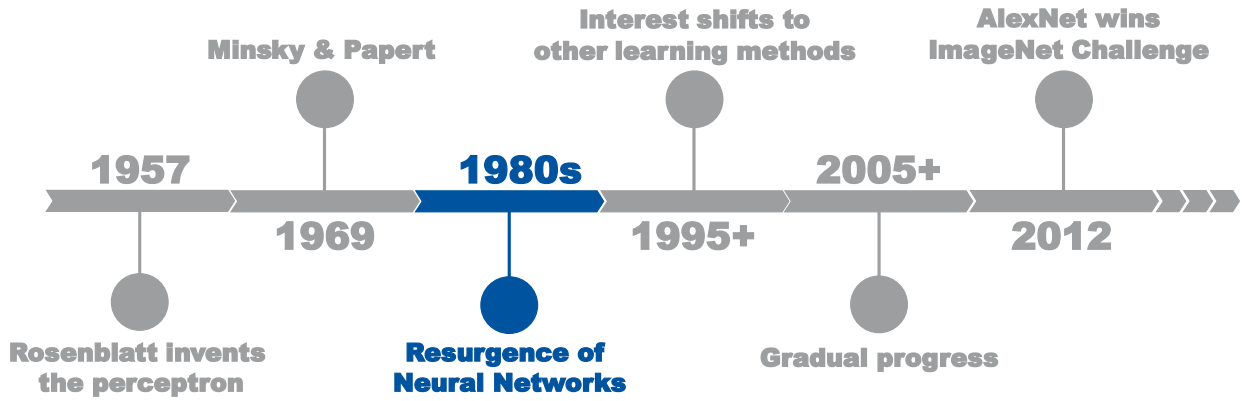


Image source: colourbox.de, thinkstock

A Brief History of Neural Networks

Resurgence of Neural Networks

- Some notable successes with multi-layer perceptrons
- Backpropagation learning algorithm



Oh no! Killer robots will achieve world domination!

OMG! They work like the human brain!

Image sources: clipartpanda.com, cliparts.co

A Brief History of Neural Networks

Resurgence of Neural Networks

- Some notable successes with multi-layer perceptrons
- Backpropagation learning algorithm
- But they are hard to train, tend to overfit, and have unintuitive parameters
- So, the excitement fades again…

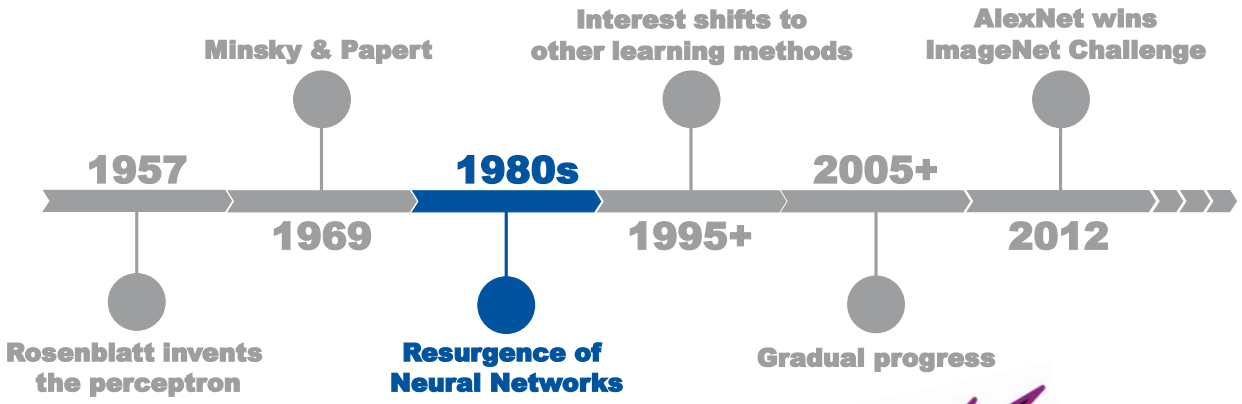
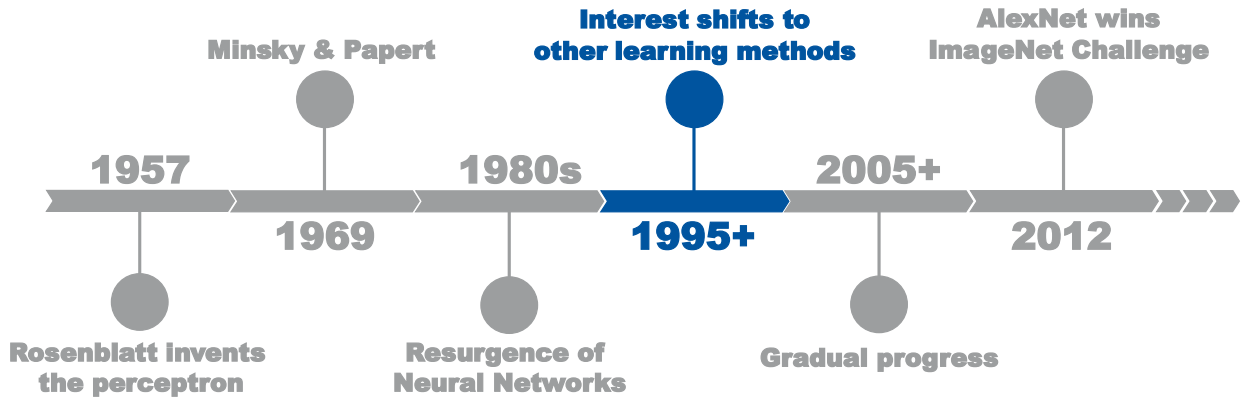


Image source: clipartof.com, colourbox.de

A Brief History of Neural Networks

Interest shifts

- Interest shifts to other learning methods, notably Support Vector Machines
- Machine Learning becomes a discipline of its own



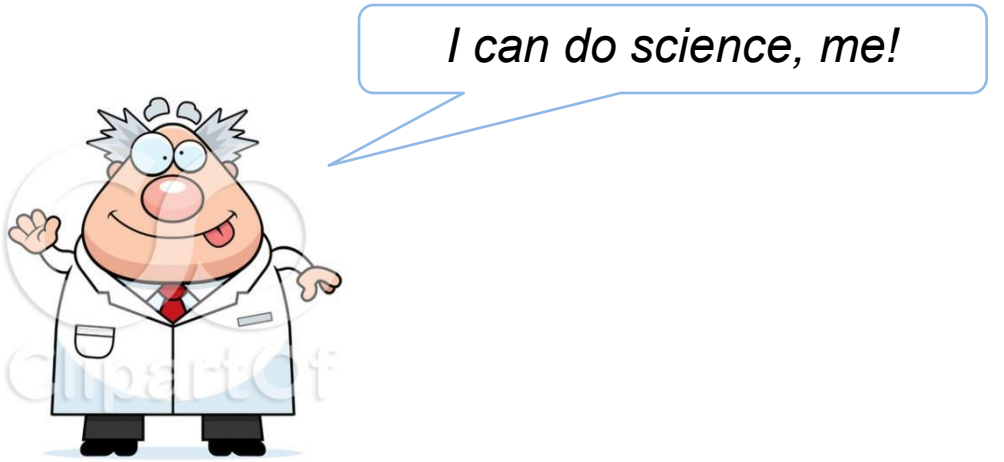
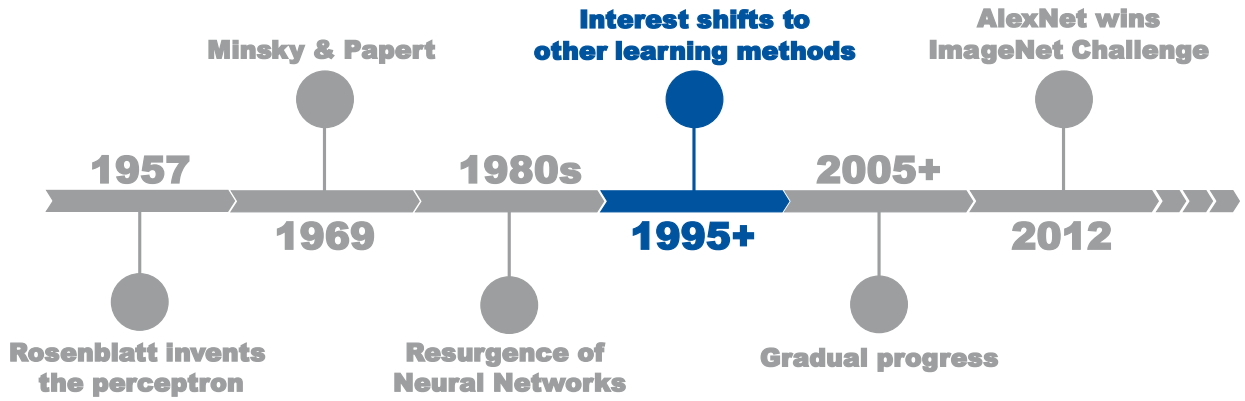


Image source: clipartof.com

A Brief History of Neural Networks

Interest shifts

- Interest shifts to other learning methods, notably Support Vector Machines
- Machine Learning becomes a discipline of its own
- The general public and the press still love Neural Networks



I'm doing Machine Learning.

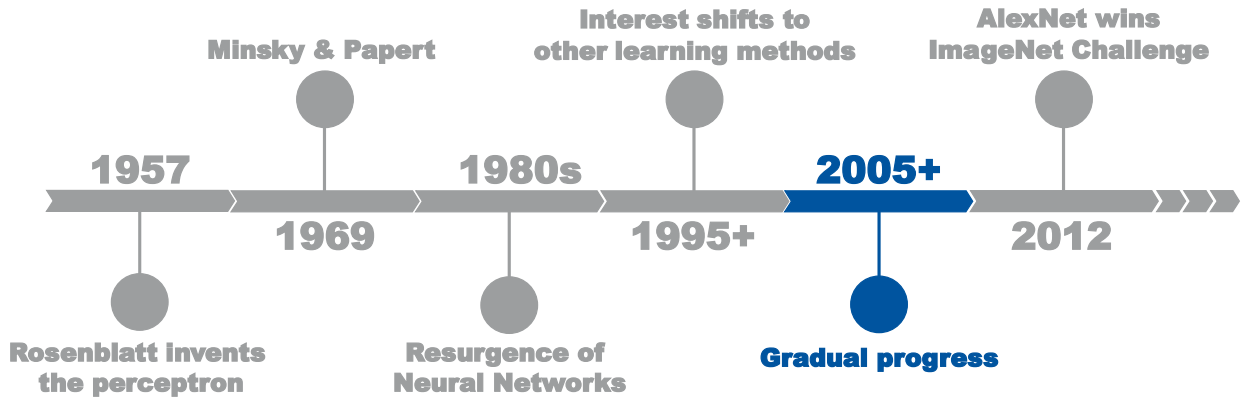
So, you're using Neural Networks?

Actually...

A Brief History of Neural Networks

Gradual Progress

- Better understanding how to successfully train deep networks
- Availability of large datasets and powerful GPUs
- Still largely under the radar for many disciplines applying ML



Are you using Neural Networks?

Come on. Get real!

A Brief History of Neural Networks

The ImageNet Moment

- ImageNet Large Scale Visual Recognition Challenge
- Breakthrough success:
  A ConvNet halves the error rate of dedicated vision approaches
- Deep Learning is widely adopted
- Exponential increase in number of researchers working in Deep Learning over the following years

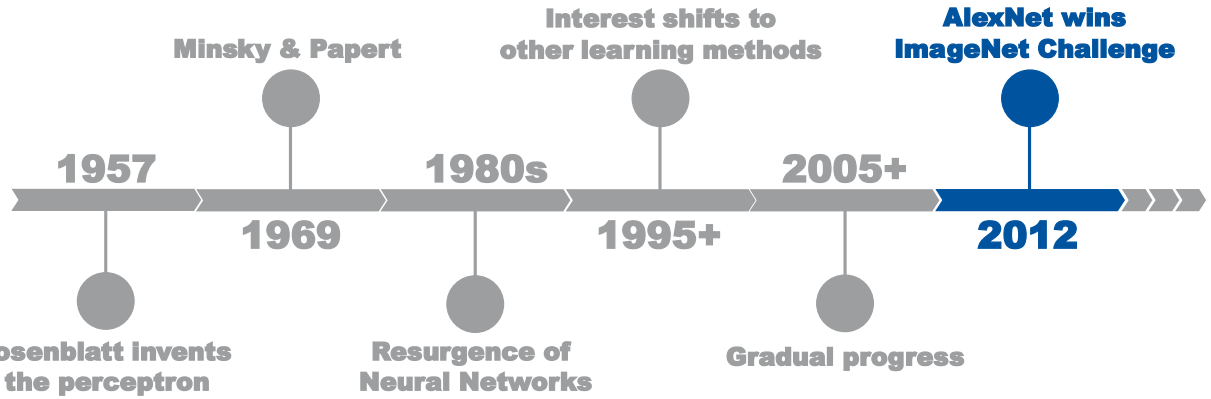
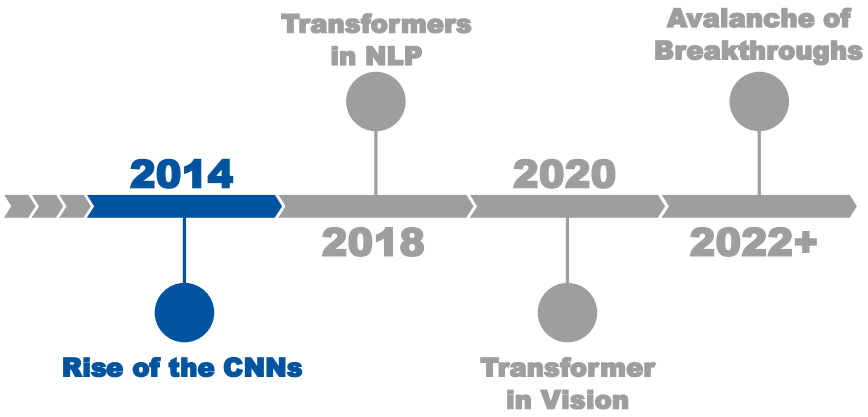


Image source: clipartpanda.com, clipartof.com

A Brief History of Neural Networks

Rise of the CNNs

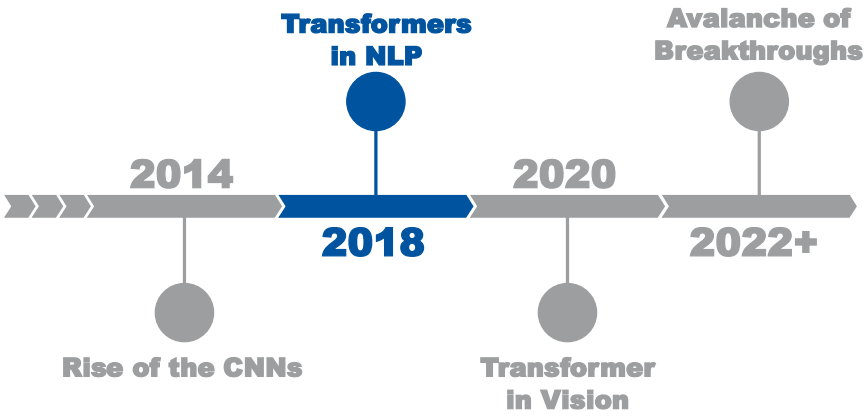
- Convolutional Networks become the de-facto standard for computer vision tasks
- CNN architectures become well understood
- Many discoveries on the way



A Brief History of Neural Networks

Transformers in NLP

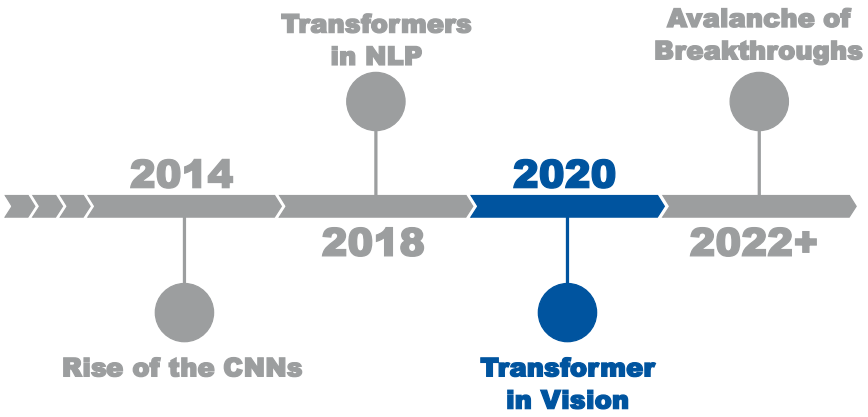
- Transformers show impressive results for NLP tasks
- Very general architecture, much easier to train at large scale than RNNs, the previous default for sequence prediction tasks



A Brief History of Neural Networks

Transformers in Vision

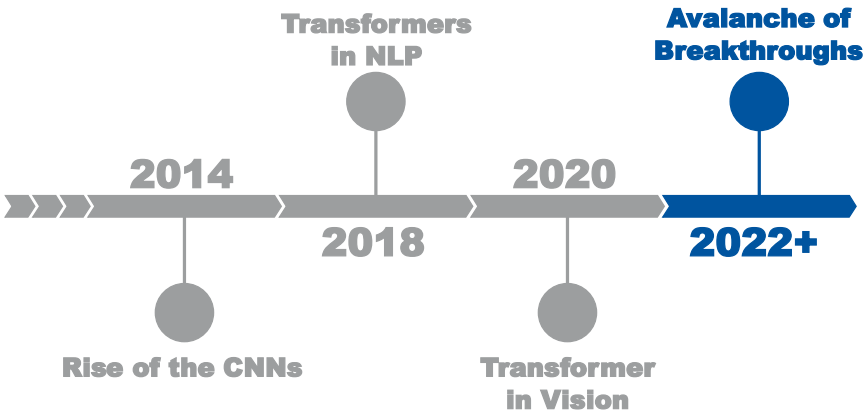
- Transformers are also adopted for vision tasks
- Show better scaling behavior than CNNs
- Similarity of architectures enables powerful synergies with NLP



A Brief History of Neural Networks

Avalanche of Breakthroughs

- Breakthrough results are achieved at a staggering cadence
  - Large Language Models (LLMs)
  - Language-Vision Models (LVMs)
  - Image Generation Models
  - ChatGPT
- The results become accessible to the general public (and change the way how many tasks are performed).
- Generative AI becomes a mainstream research field.



Hello Grandma! You have WHAT? Talked to ChatGPT???

Neural Network Basics

1. Perceptrons
2. Multi-Layer Perceptrons
3. Loss Functions
4. Backpropagation
5. Computational Graphs
6. Stochastic Gradient Descent
7. History of Neural Networks
8. **Deep Learning‘s Main Breakthroughs**

Convolutional Neural Networks (CNNs)

- Image Classification:
  - Input an image, output a label.
  - A core task in Computer Vision.

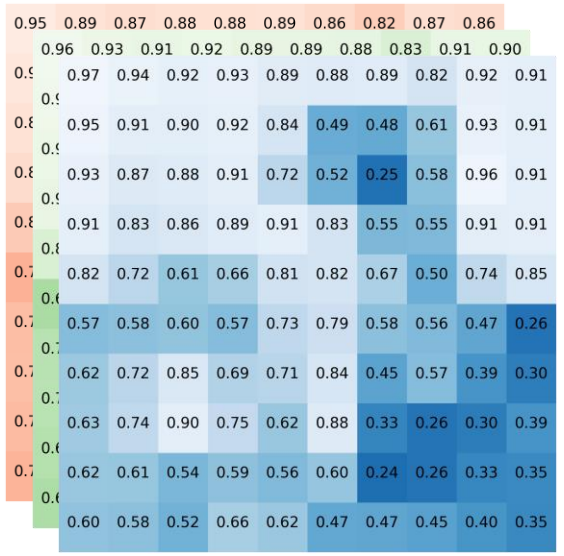
- Vision is difficult!
  - What computers see is…

Mouse
Rabbit
Cat
Dog
Elephant

Convolutional Neural Networks (CNNs)

- Image Classification:
  - Input an image, output a label.
  - A core task in Computer Vision.

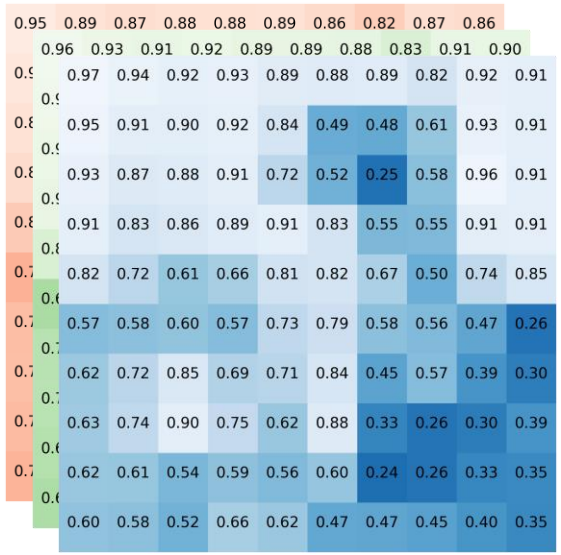
- Vision is difficult!
  - What computers see is…

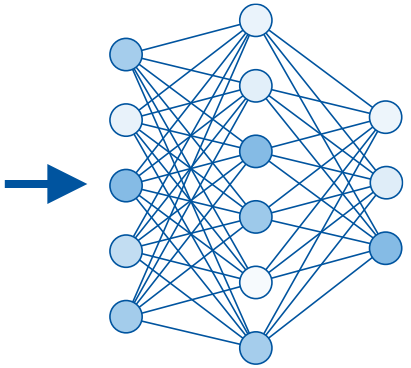


Mouse
Rabbit
Cat
Dog
Elephant

Convolutional Neural Networks (CNNs)

- Image Classification:
  - Input an image, output a label.
  - A core task in Computer Vision.
- Vision is difficult!
  - What computers see is…
- How can we tackle this with neural networks?

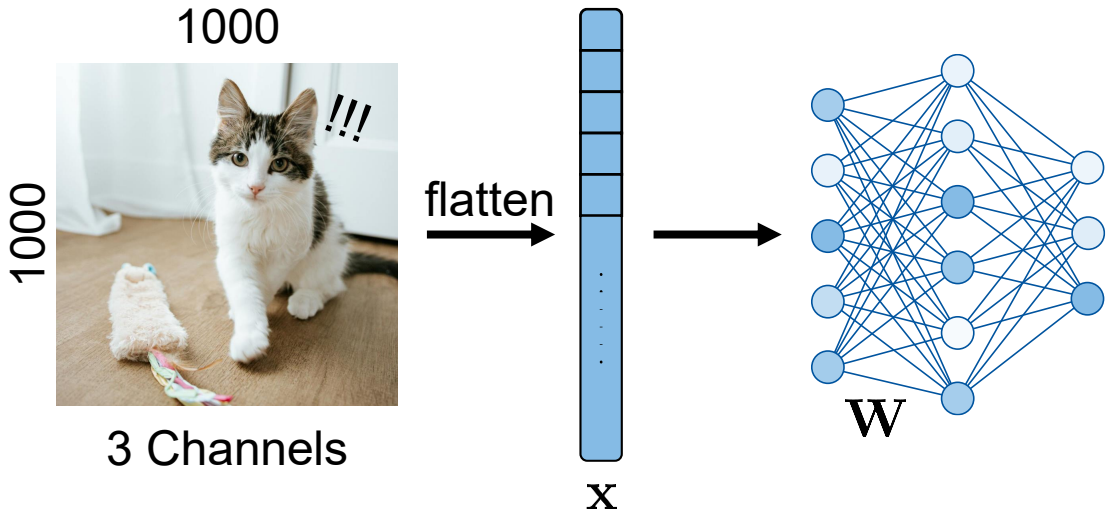




Mouse
Rabbit
Cat
Dog
Elephant

Image Processing

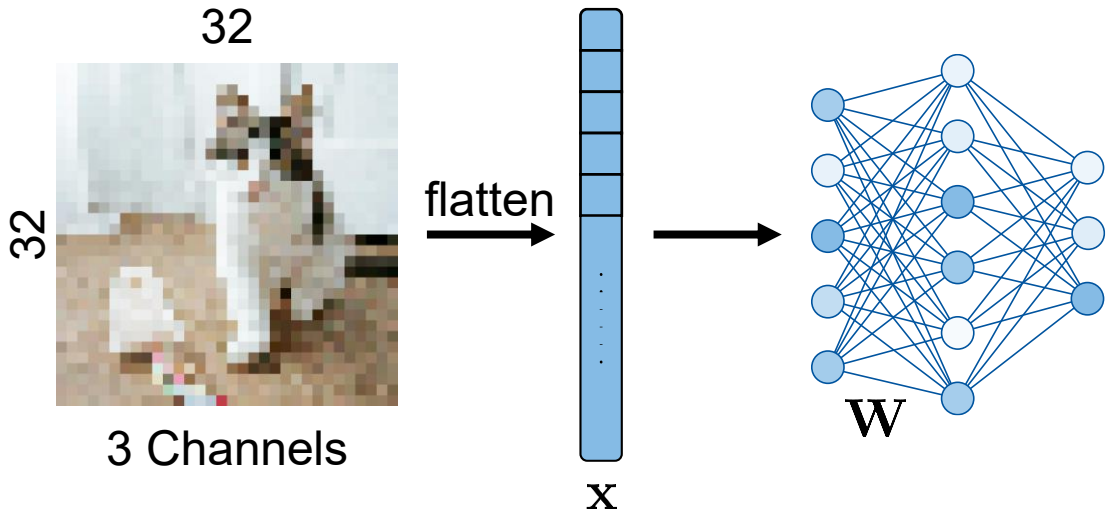
- First idea: use a fully connected MLP.
- Let's count the parameters needed:
  - Use pixels as input, e.g. 1000×1000×3 (RGB)
  - That is 3M inputs…
  - A single hidden layer with 1000 neurons will contain 3B parameters!
- We can lower the resolution…



$$\dim(\mathbf{x}) = 3 \cdot 10^6$$
$$\dim(\mathbf{W}) = 3 \cdot 10^9$$

Image Processing

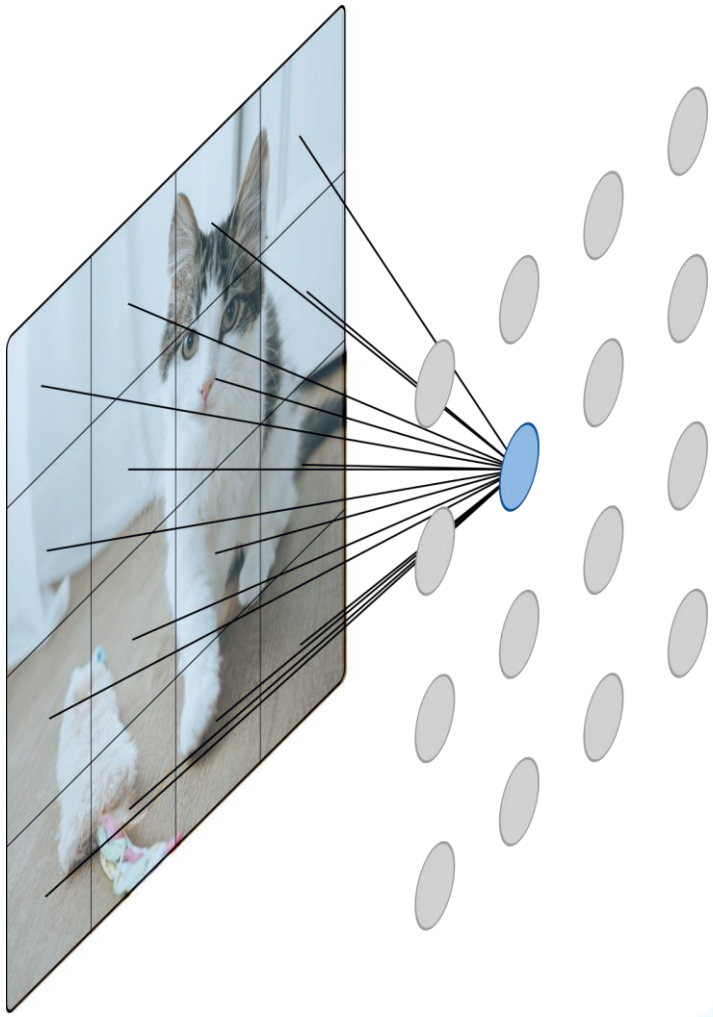
- First idea: use a fully connected MLP.
- Let's count the parameters needed:
  - Use pixels as input, e.g. 1000×1000×3 (RGB)
  - That is 3M inputs…
  - A single hidden layer with 1000 neurons will contain 3B parameters!
- We can lower the resolution…
  - …but this loses a lot of information.
  - Fully-connected layers still have millions of parameters each.
  - And we would like to have many layers…



$$\dim(\mathbf{x}) \approx 10^3$$
$$\dim(\mathbf{W}) \approx 10^6$$

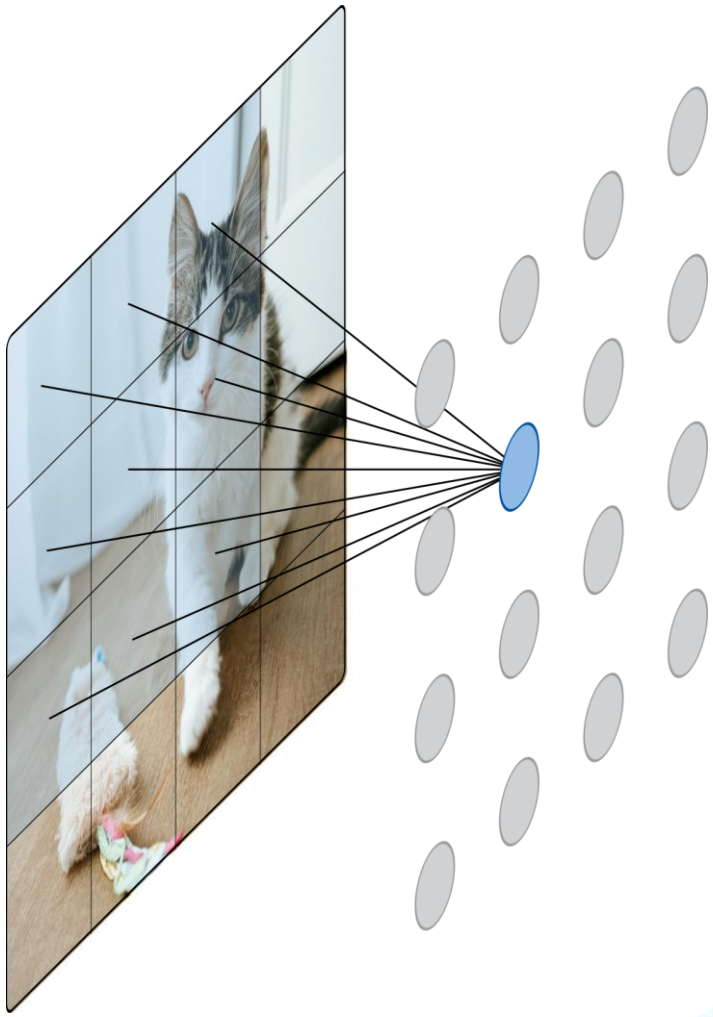
Designing a learnable image operator

- We want to keep the high-resolution grid structure.



Designing a learnable image operator

- We want to keep the high-resolution grid structure.
- Idea 1: Local connectivity only.
  - Not all image content matters for local features.
  - Massively reduces parameters!
- Idea 2: Translational invariance.
  - It should not matter where a feature is located.
  - Reuse the same parameters for every location.

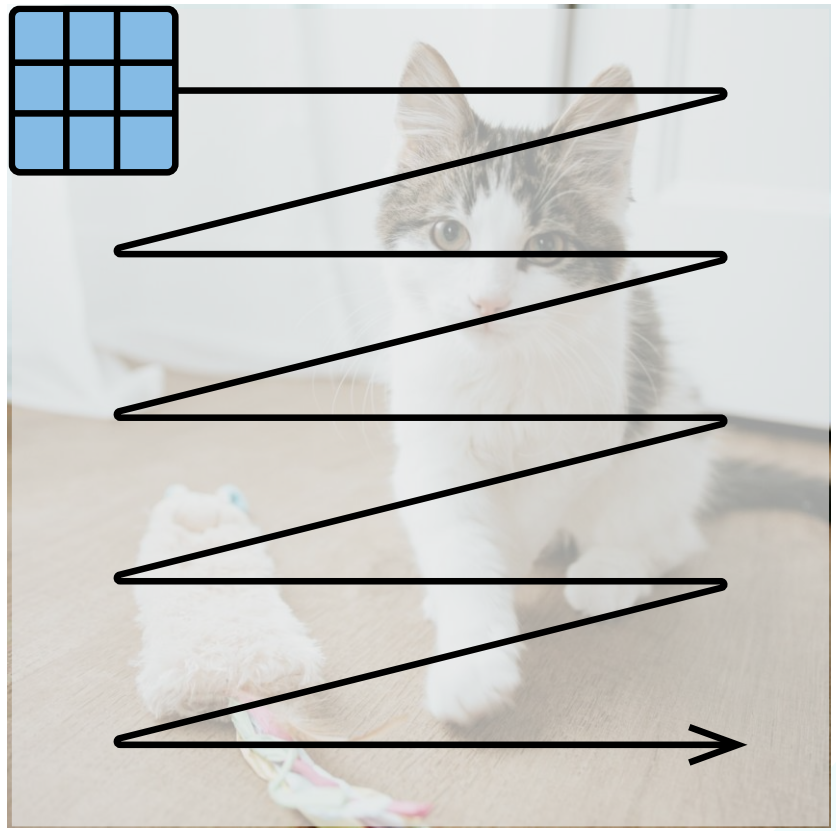


Convolutions

- Resulting operator: slide a small window over the input image, and only process this area.

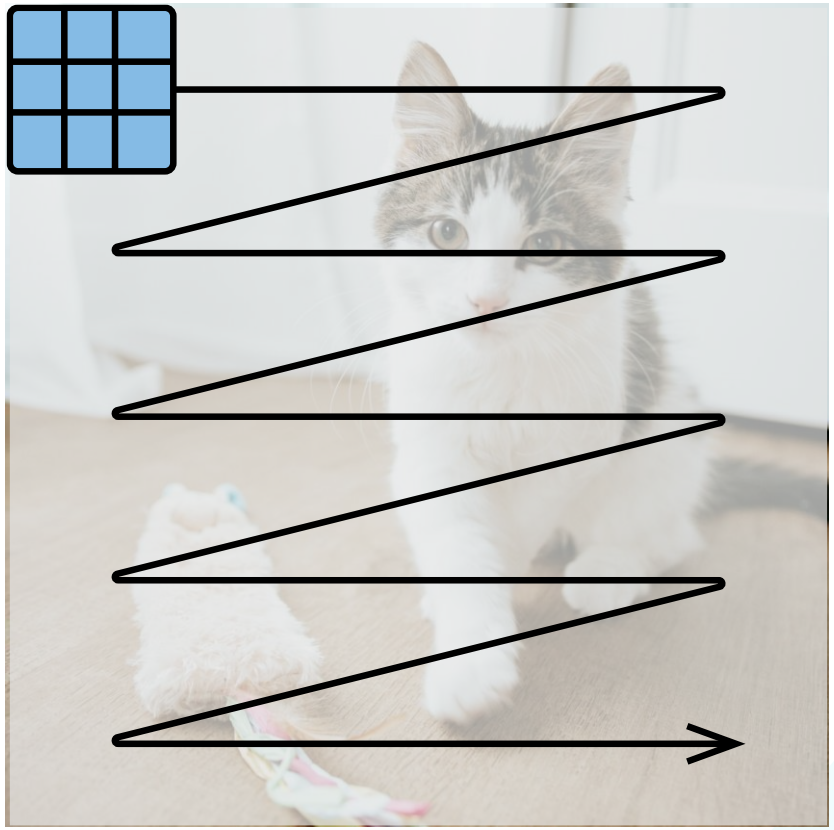
- The weights are shared between local windows.
  - This is called a filter (or a kernel).
  - Only $3 \cdot 3 \cdot 3 = 27$ parameters for a 3x3 filter on RGB data – *independent of input size*.

- The output is a called a feature map.



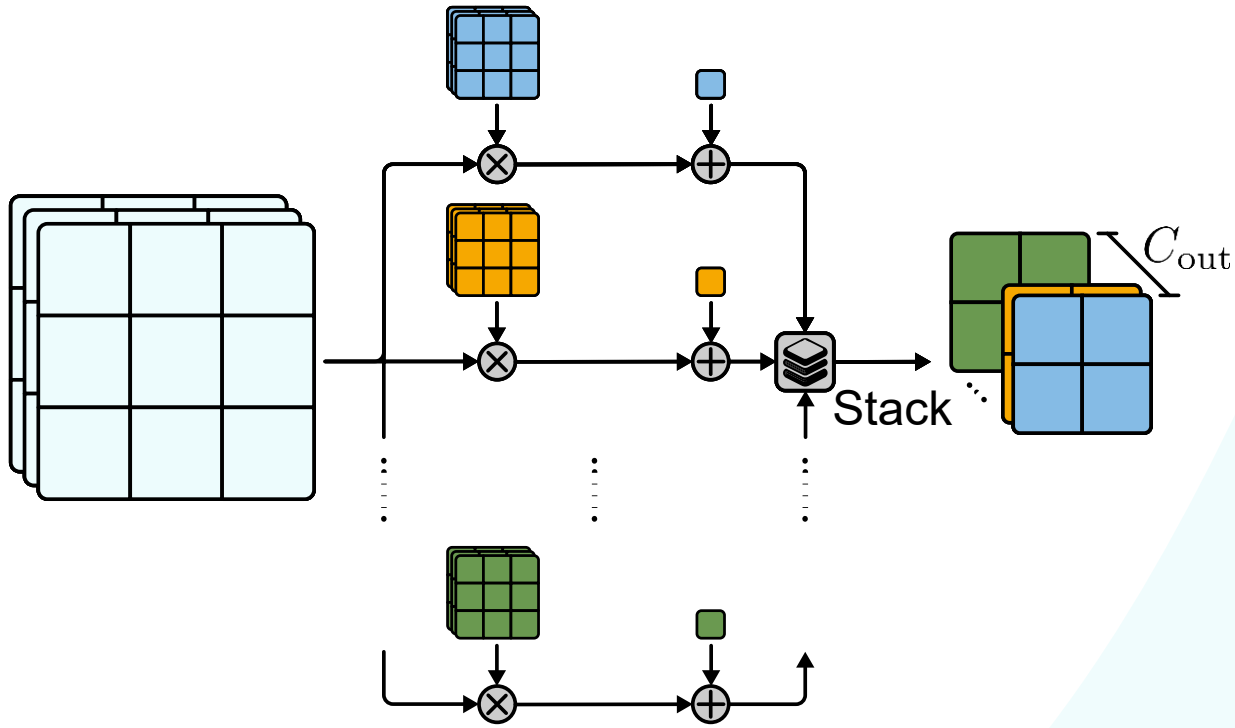
Convolutions: Advantages

- **Parameter reduction**:
  - Due to local connectivity and weight sharing.
- **Local feature detection**:
  - Long history of using local features in Computer Vision.
  - Now, we use learnable filters that can be updated with backpropagated gradients.
- **Translation Invariance**:
  - Once a filter has learned to recognize a feature, it can recognize it anywhere in the image.



Convolutional Layers

- A convolutional layer can have more than one filter, resulting in multiple output channels.
- C_{out} = Number of output channels
  = Number of filters
- By default, filters are applied to all input channels.
- This allows the network to learn different features in the same layer.
  - E.g., one filter for edge detection, another filter for corner detection.



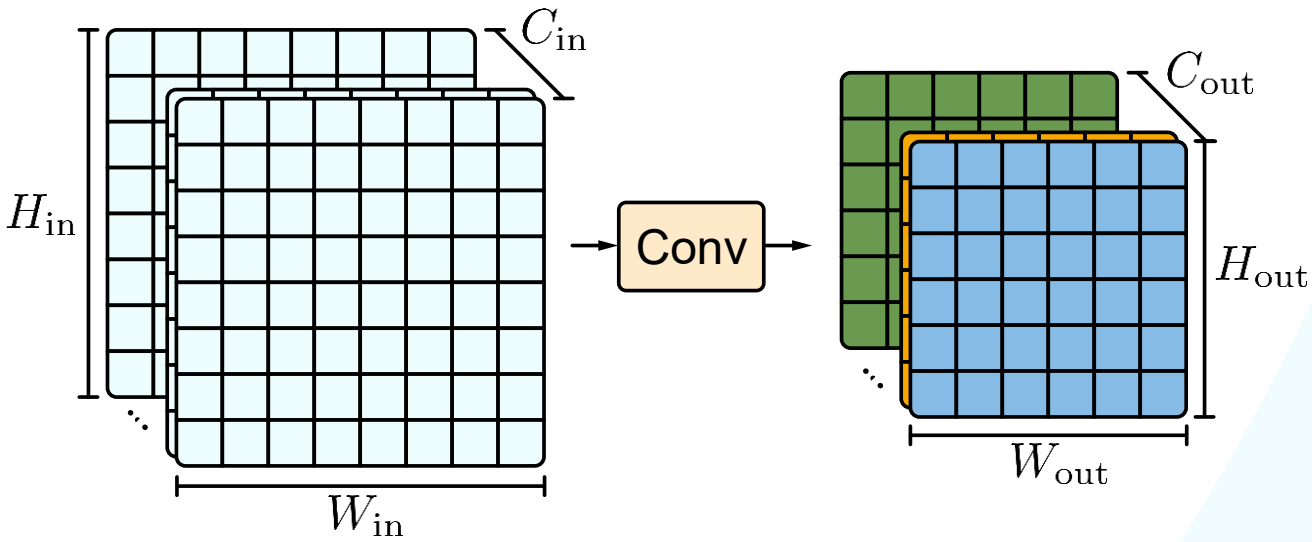
Convolutional Layers

- To summarize, a convolutional layer transforms an input grid such that

$$(C_{\text{in}}, H_{\text{in}}, W_{\text{in}}) \to (C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$$

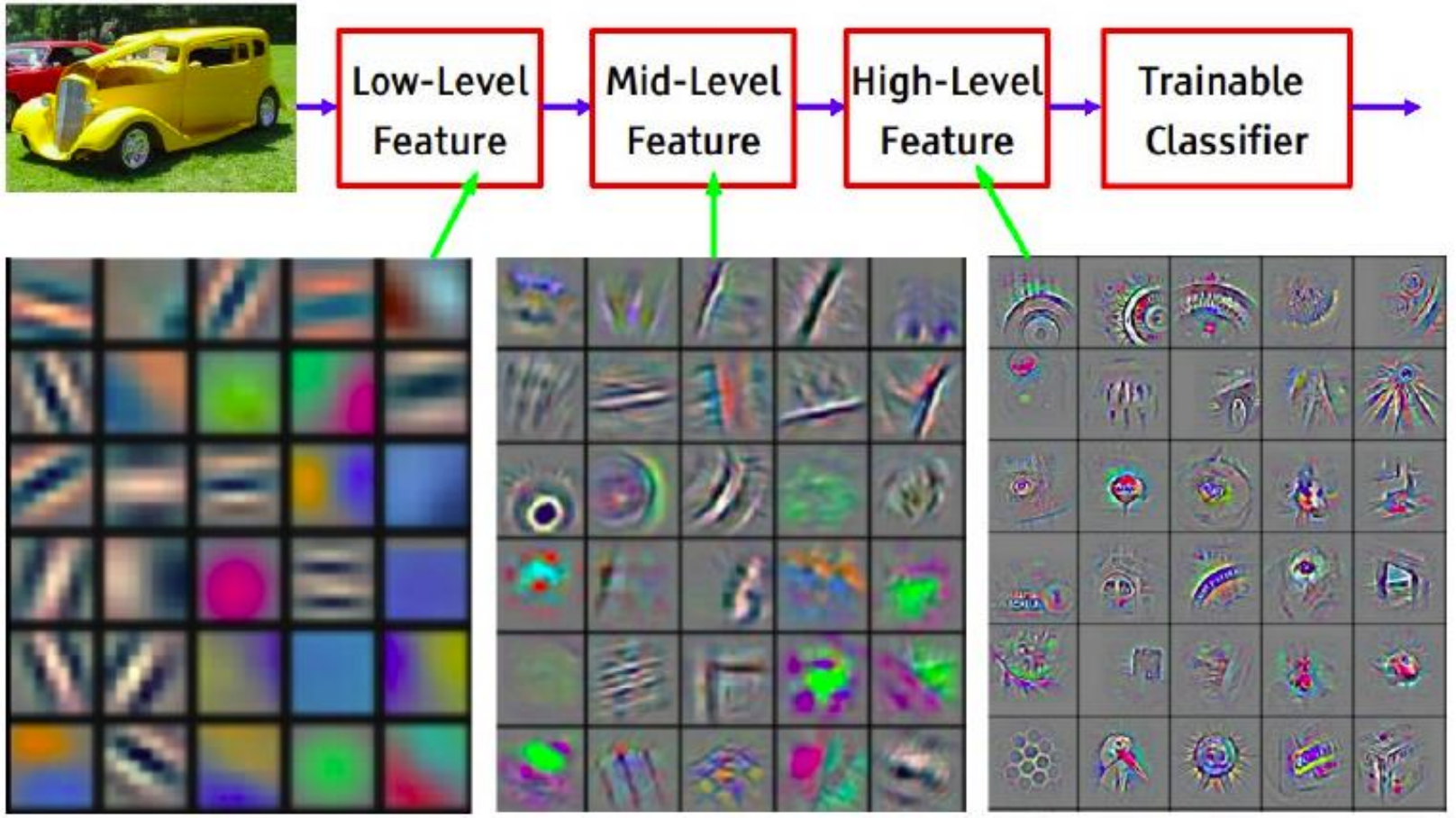
  using a set of filters with shape $F_H \times F_W$.

- This is still a linear operation!
  - Need to combine with non-linear activation functions as before.



- Fundamental paradigm shift in how we think of neural networks
  - MLPs: *Neural units performing computation*
  - CNNs: *Transforming an input volume into an output volume of activations*

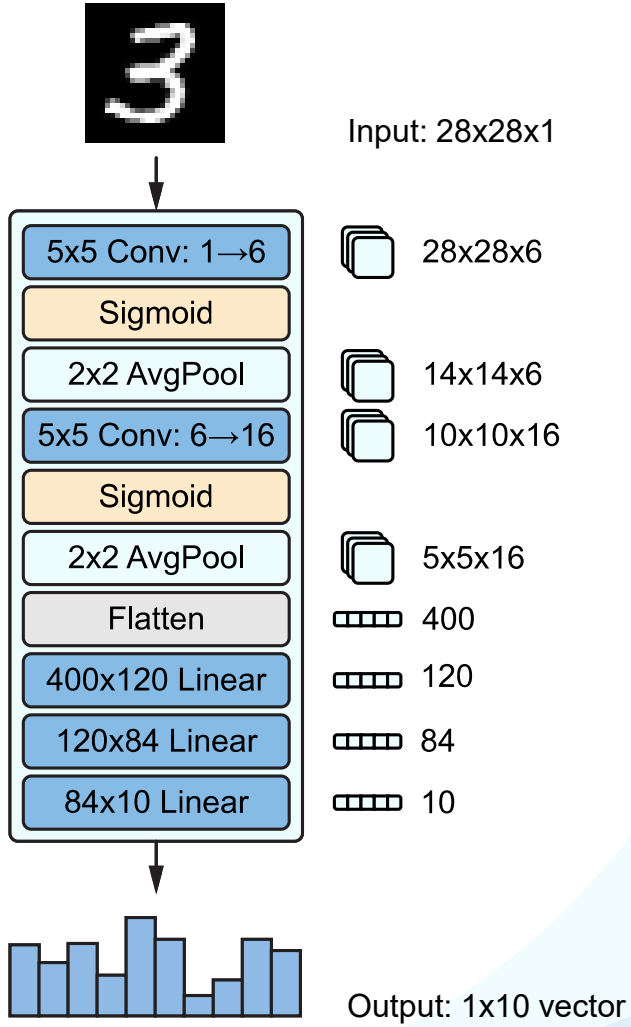
Effect of Multiple Convolution Layers



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

LeNet (1998)

- One of the first widely successful convolutional architectures.
  - E.g., handwritten digit recognition for postal zip codes.
- 2 convolutional layers, 2 pooling layers.
- Fully-connected NN layers for classification.
- Trained on several thousand images (low-resolution, black-and-white).



Y. LeCun et al., “Gradient-based learning applied to document recognition”, Proceedings of the IEEE 86(11): 2278–2324, 1998.

Breakthrough Moment: ImageNet Challenge 2012

- ImageNet
  - Community effort to collect ~14M labeled internet images
  - 20k classes
  - Human labels via Amazon Mechanical Turk

- ImageNet Challenge (ILSVRC)
  - 1.2 million training images
  - 1000 classes
  - Goal: Predict ground-truth class within top-5 responses
  - One of the top benchmarks in Computer Vision

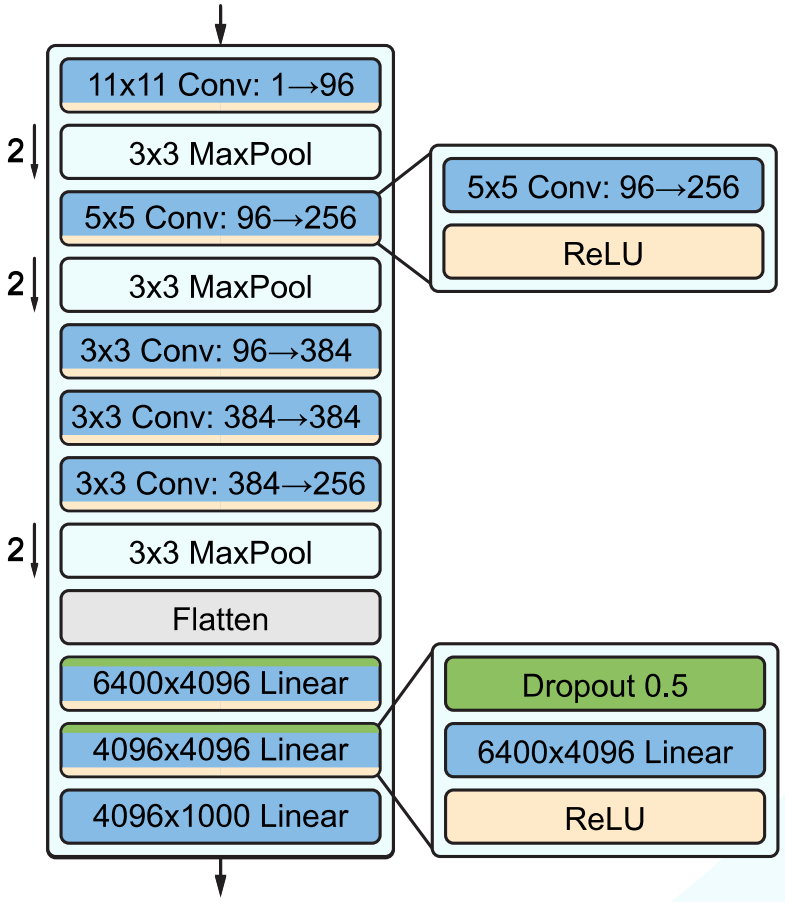
Image from [J. Deng et al., CVPR'09]

J. Deng et al., "ImageNet: A Large-Scale Hierarchical Image Database", CVPR 2009.

O. Russakovsky, J. Deng et al., "ImageNet Large Scale Visual Recognition Challenge", International Journal of Computer Vision, 115(3):211-252, 2015.

AlexNet (2012)

- Won the ImageNet Challenge 2012 (ILSVRC).
  - Outputs a prediction over 1000 classes!
- Bigger model than LeNet
  - 7 hidden layers, 60M parameters.
  - Used effective training strategies (Dropout, ReLU)
- Trained on about a million images.
  - Possible because of efficient GPU implementation.
- Achieved an ILSVRC top-5 error rate of 16.4%
  - Huge improvement over traditional Computer Vision methods at the time
  - *This kicked off a revolution in Computer Vision*



A. Krizhevsky et al., “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.

ILSVRC Winners Over the Years

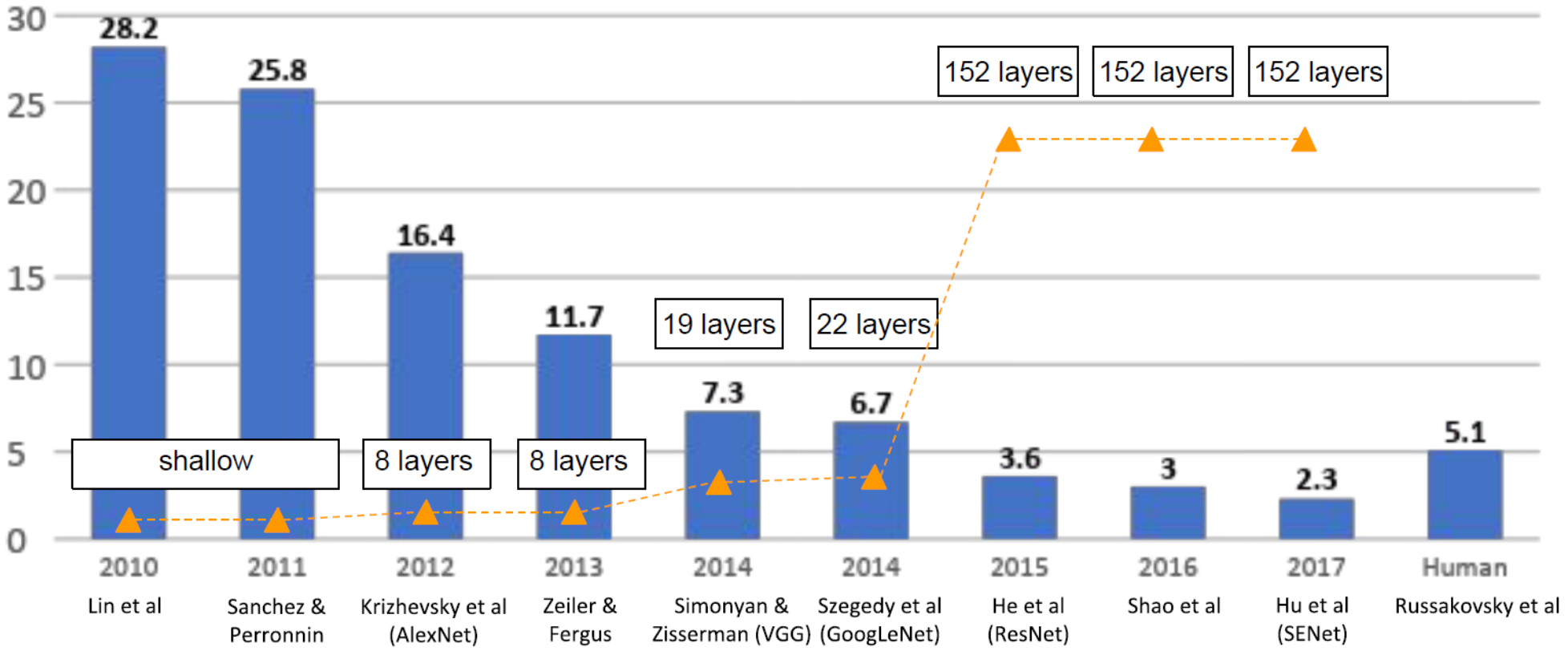
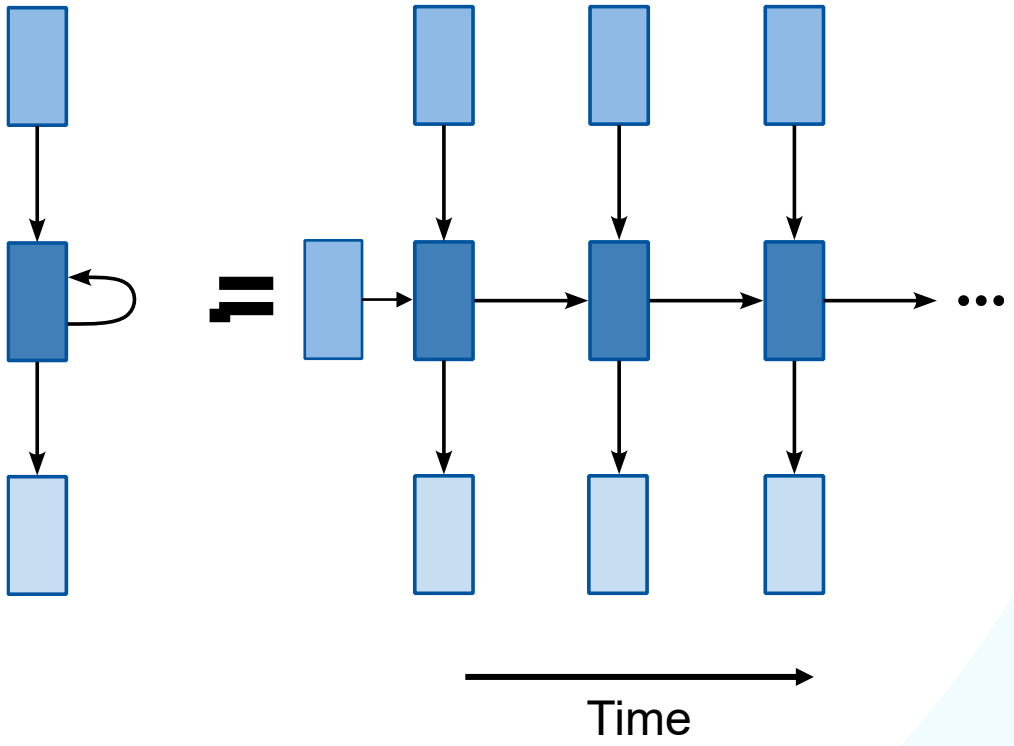


Image from Li Fei-Fei

Recurrent Neural Networks (RNNs)

- RNNs are neural networks with an additional forward connection over time.
  - You can unroll them to create a network that extends over multiple time steps.
  - Essentially a network with variable input length and shared weights.
- Combines a distributed hidden state with non-linear dynamics.



Sequence Prediction Tasks

- Often, we want to produce sequences that relate to some other information.
  - E.g., textual descriptions.
- We can use an Encoder-Decoder structure with an RNN decoder for this kind of task.
  - Encoder compresses input into context vector.
  - Decoder uses context to generate transformed output.

Machine Translation

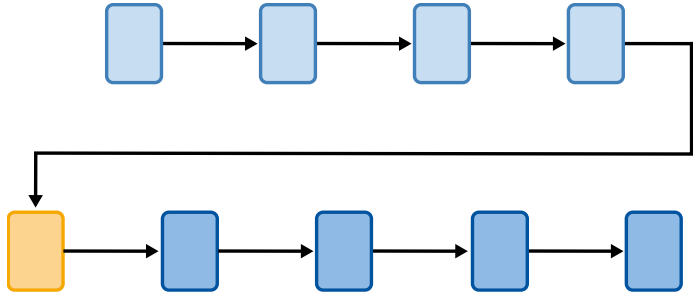
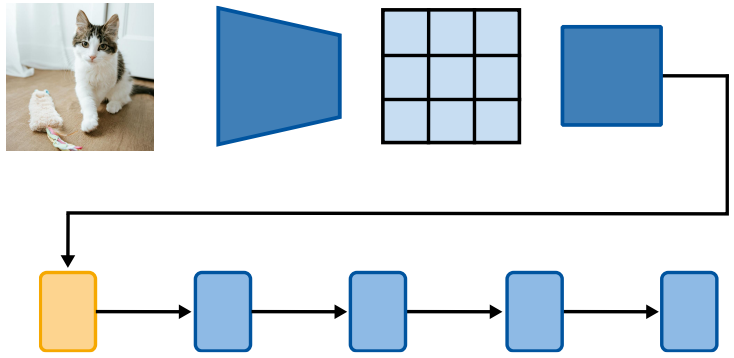


Image Captioning



Language Translation

- Input: Sequence $\mathbf{x} = x_1, x_2, \ldots, x_T$
- Output: Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$
- Encoder $h_t = \mathrm{RNN}_\Theta(x_t, h_{t-1})$

  $s_0,\ c = \mathrm{MLP}(\mathbf{h})$
- Decoder $y_t,\ s_t = \mathrm{RNN}_\Phi(y_{t-1}, s_{t-1}, c)$

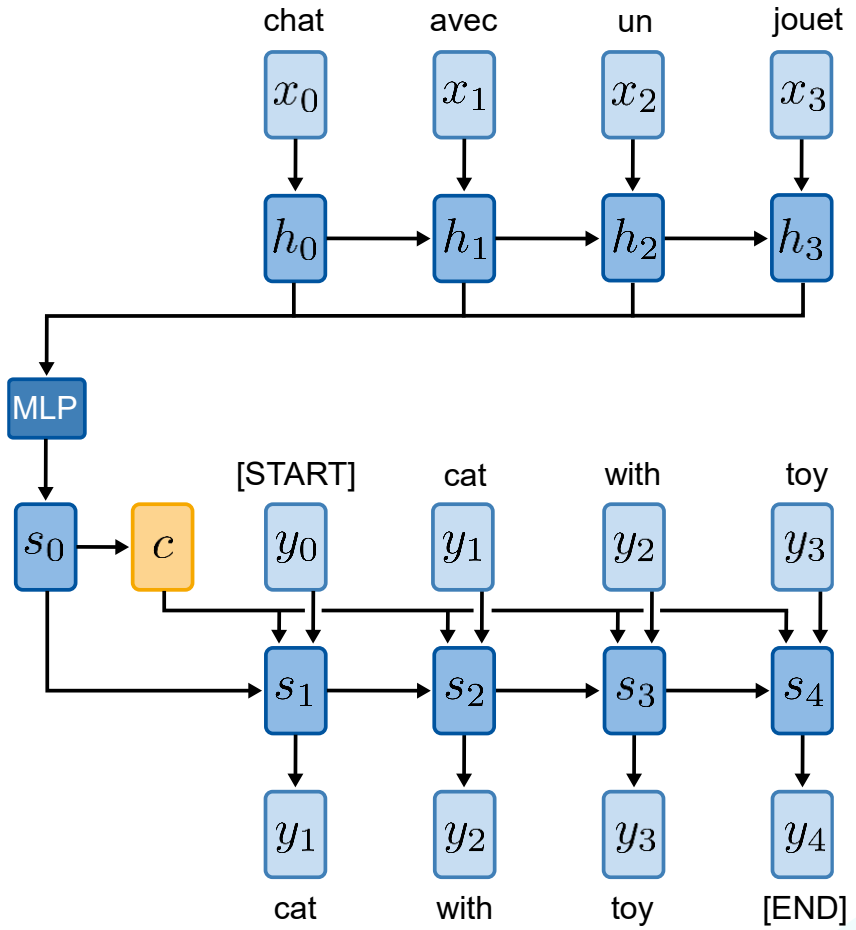


Image Captioning

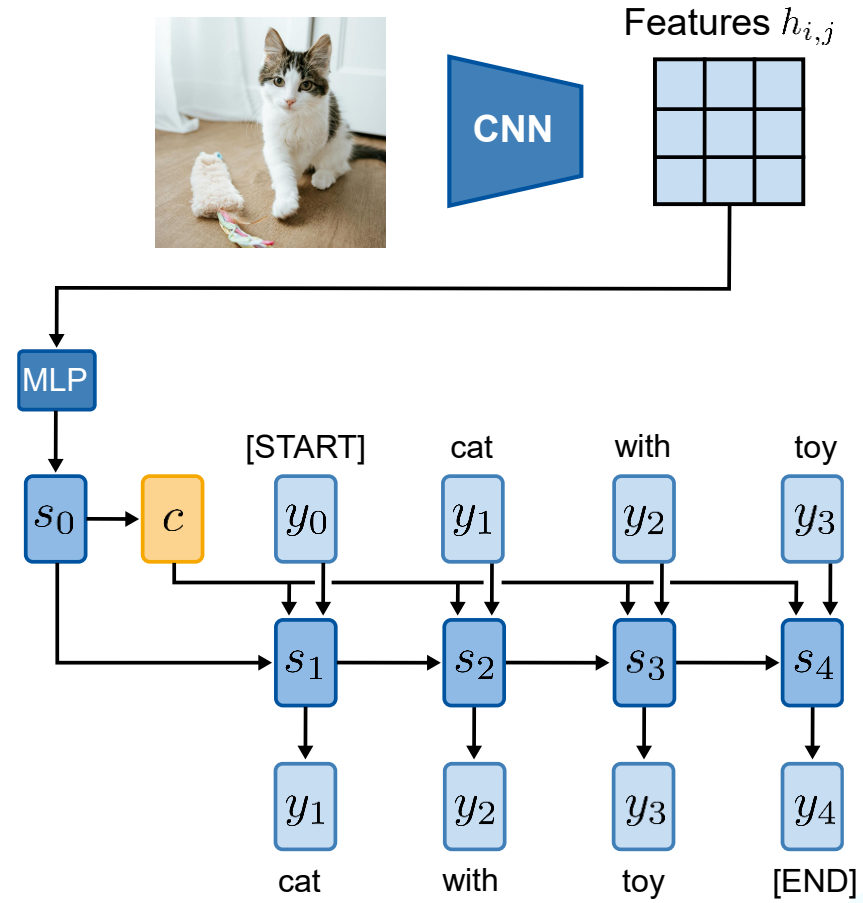
- Input: Image $\mathbf{I}$
- Output: Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$
- Encoder $\mathbf{h} = \mathrm{CNN}(\mathbf{I})$

$$s_0,\ c = \mathrm{MLP}(\mathbf{h})$$

- Decoder $y_t,\ s_t = \mathrm{RNN}_\Phi(y_{t-1}, s_{t-1}, c)$

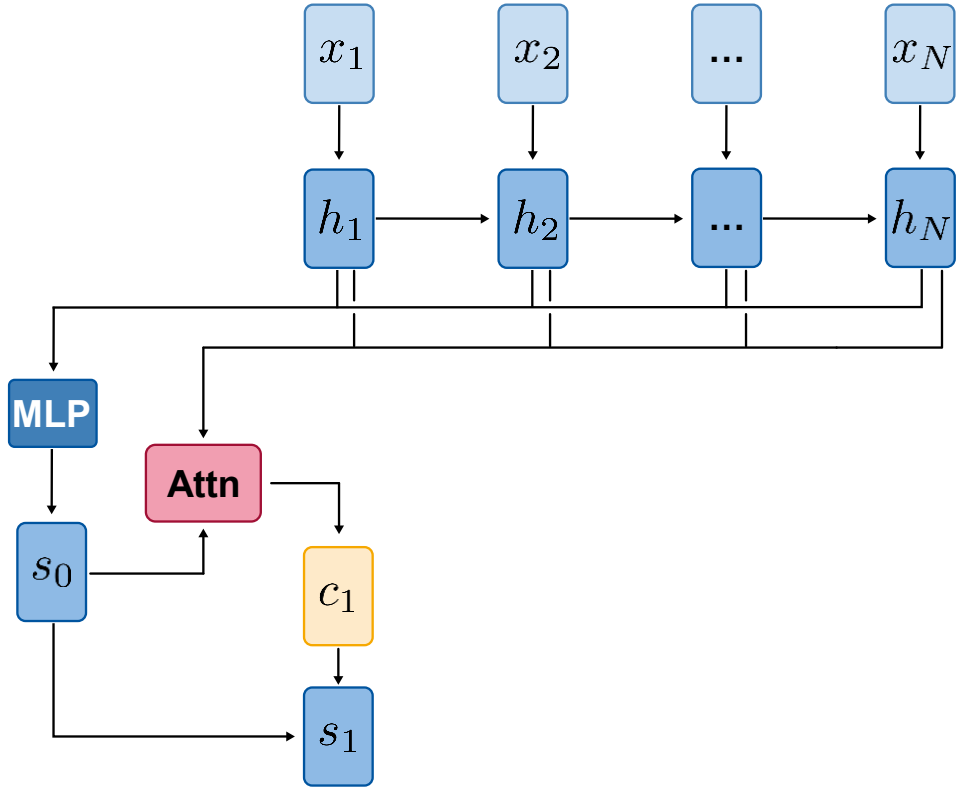
In both cases, the context vector c is an information bottleneck.

It limits the expressive power of the decoder network.



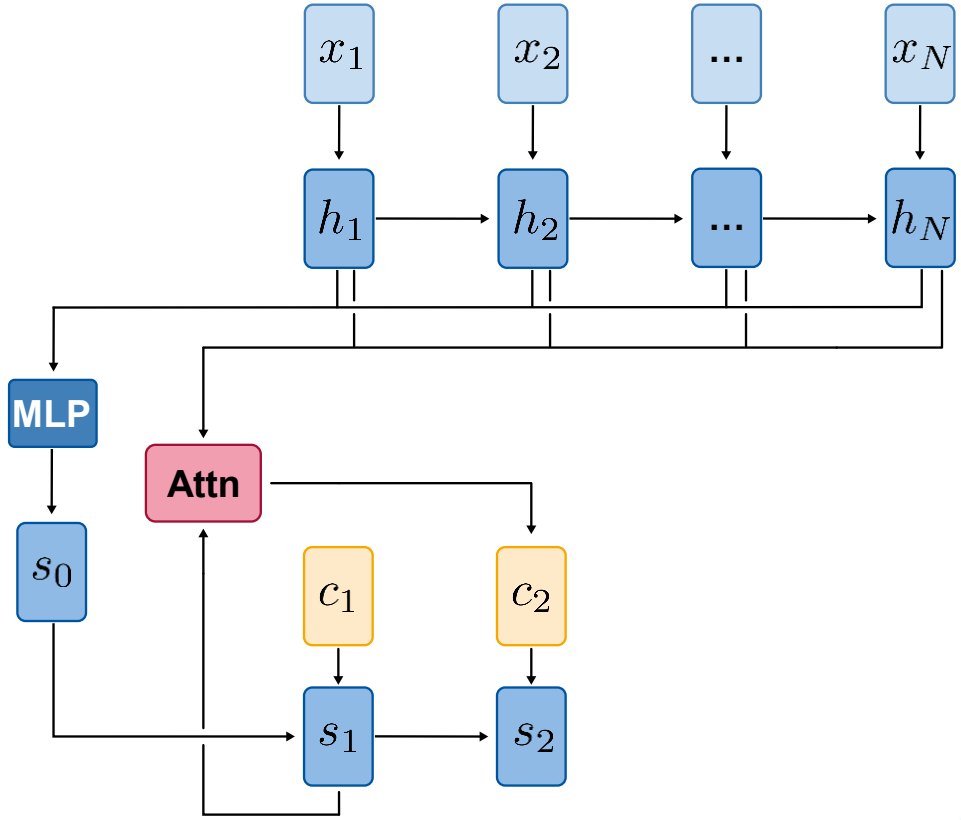
Machine Translation with Attention

- Compute Alignment scores (scalars):
  - $e_{t,i} = f_{\text{Attn}}(s_{t-1}, h_i)$
  - where $f_{\text{Attn}}(\cdot)$ is an MLP
- Normalize to get attention weights:
  - $a_{t,:} = \text{softmax}(e_{t,:})$
  - $0 < a_{t,i} < 1$
  - Note: attention values sum to 1
- Compute context vector:
  - $c_t = \sum_i a_{t,i} h_i$



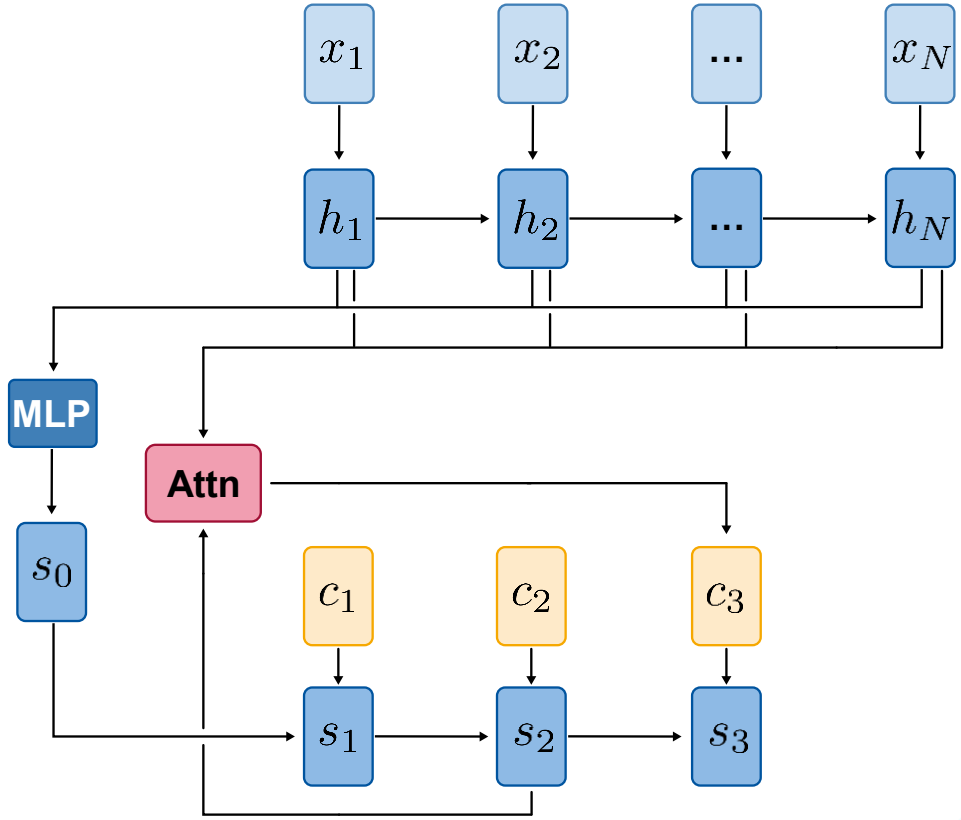
Machine Translation with Attention

- Compute Alignment scores (scalars):
  - $e_{t,i} = f_{\text{Attn}}(s_{t-1}, h_i)$
  - where $f_{\text{Attn}}(\cdot)$ is an MLP
- Normalize to get attention weights:
  - $a_{t,:} = \text{softmax}(e_{t,:})$
  - $0 < a_{t,i} < 1$
  - Note: attention values sum to 1
- Compute context vector:
  - $c_t = \sum_i a_{t,i} h_i$



Machine Translation with Attention

- Compute Alignment scores (scalars):
  - $e_{t,i} = f_{\text{Attn}}(s_{t-1}, h_i)$
  - where $f_{\text{Attn}}(\cdot)$ is an MLP
- Normalize to get attention weights:
  - $a_{t,:} = \text{softmax}(e_{t,:})$
  - $0 < a_{t,i} < 1$
  - Note: attention values sum to 1
- Compute context vector:
  - $c_t = \sum_i a_{t,i} h_i$



Machine Translation with Attention

- Compute Alignment scores (scalars):
  - $e_{t,i} = f_{\text{Attn}}(s_{t-1}, h_i)$
  - where $f_{\text{Attn}}(\cdot)$ is an MLP
- Normalize to get attention weights:
  - $a_{t,:} = \text{softmax}(e_{t,:})$
  - $0 < a_{t,i} < 1$
  - Note: attention values sum to 1
- Compute context vector:
  - $c_t = \sum_i a_{t,i} h_i$

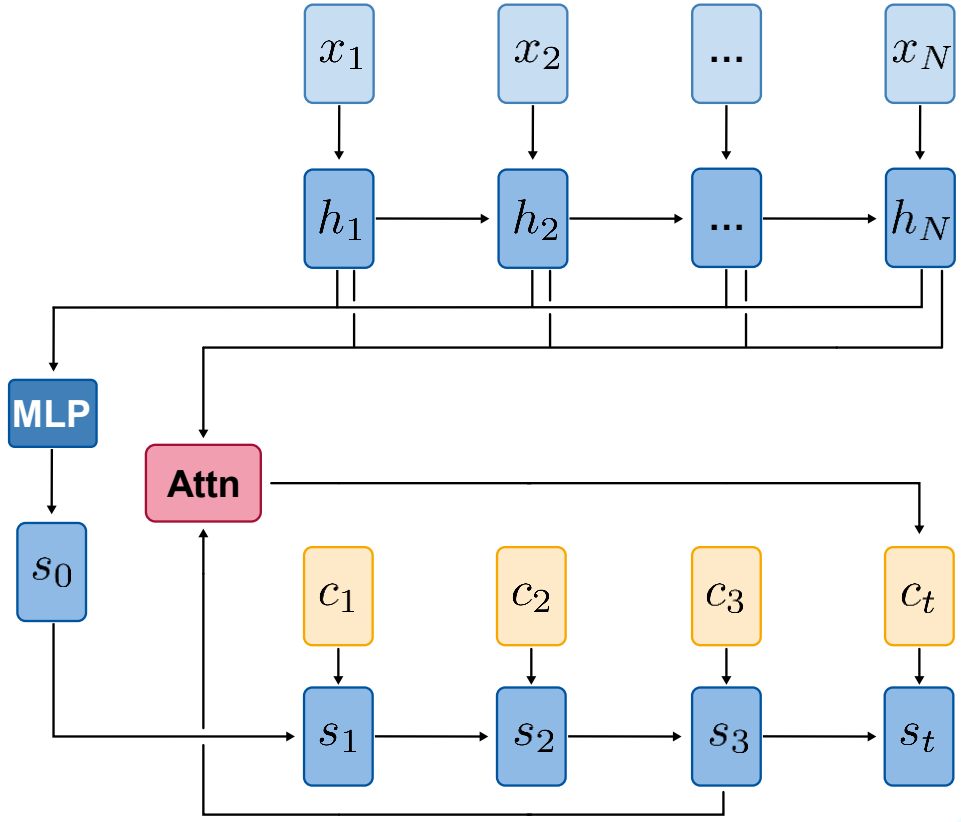
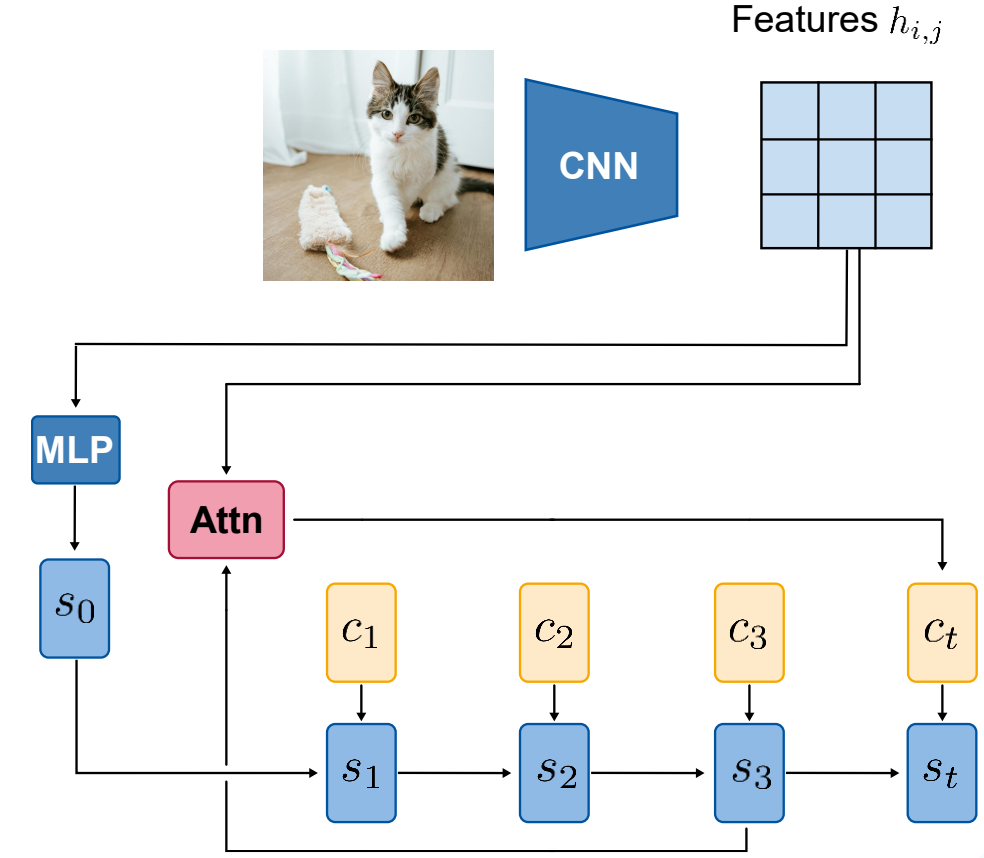


Image Captioning with Attention

- Compute Alignment scores (scalars)
  - $e_{t,i,j} = f_{\text{Attn}}(s_{t-1}, h_{i,j})$
  - where $f_{\text{Attn}}(\cdot)$ is an MLP

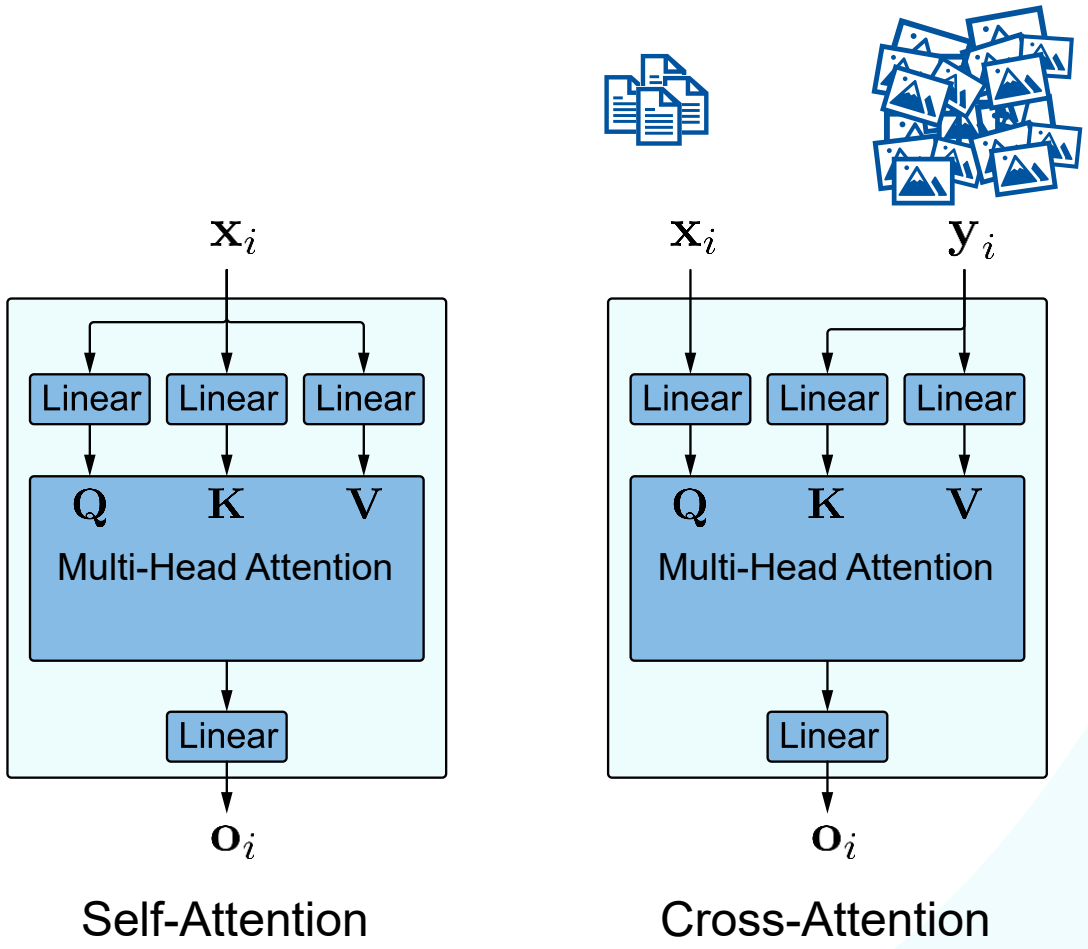
- Normalize to get attention weights
  - $a_{t,:,:} = \text{softmax}(e_{t,:,:})$
  - $0 < a_{t,i} < 1$
  - Note: attention values sum to 1

- Compute context vector
  - $c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$



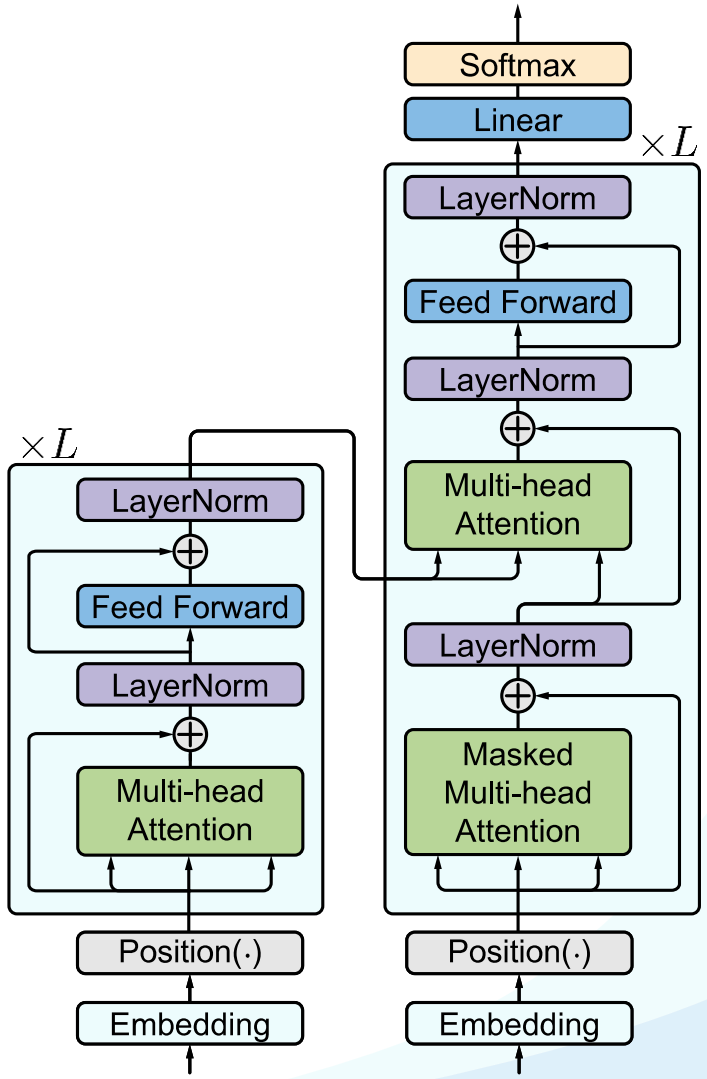
Self- and Cross-Attention

- Self-Attention distributes information between tokens of the input.
  - Keys, queries, and values are computed by linear projections from the same data $\mathbf{x}_i$
- Cross-Attention allows to incorporate information from other sources.
  - Other modalities, models, feature extractors, …
  - Inputs might have different number of tokens.



Transformer Architecture

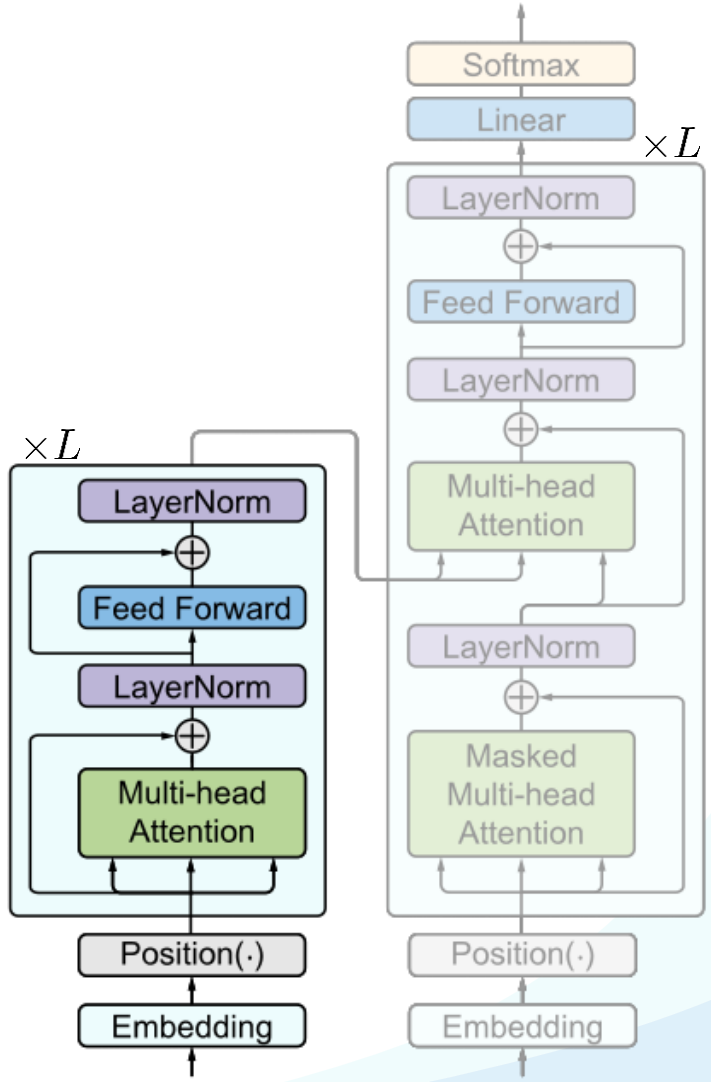
- Hugely successful architecture, based on attention, MLPs, and normalization layers.
- The original transformer design had an encoder-decoder structure.
  - Designed for NLP sequence transduction tasks, e.g., machine translation.



A. Vaswani et al., “Attention Is All You Need”, Neurips 2017

Transformer Architecture

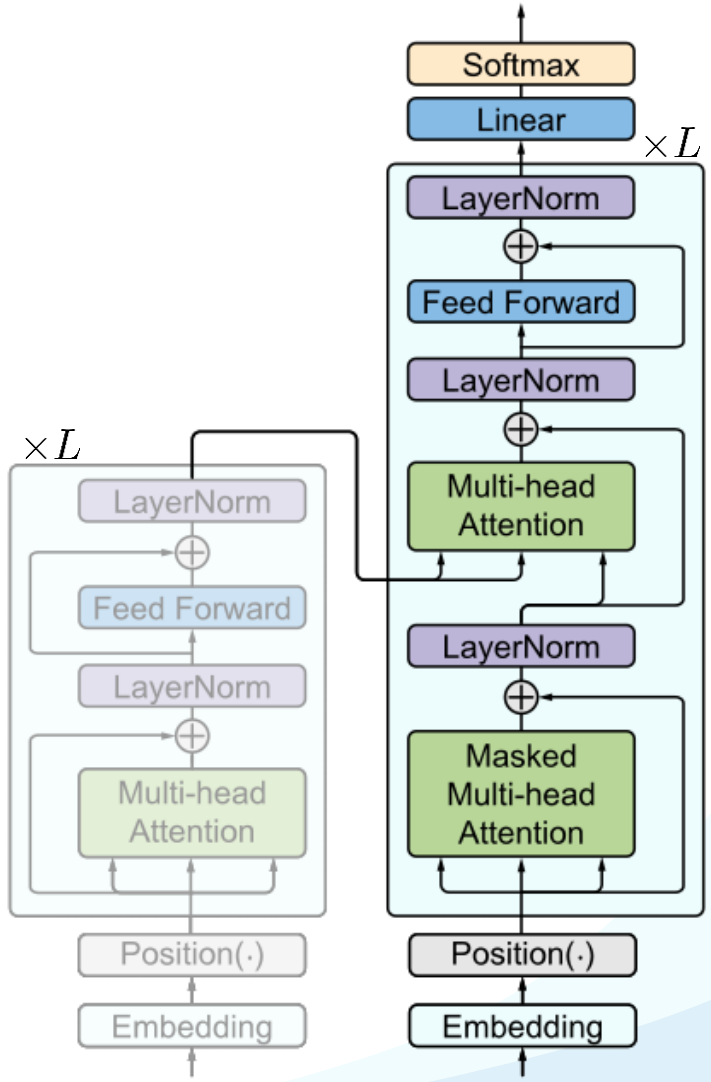
- Hugely successful architecture, based on attention, MLPs, and normalization layers.
- The original transformer design had an encoder-decoder structure.
  - Designed for NLP sequence transduction tasks, e.g., machine translation.
  - Encoder computes features from inputs to provide context.



A. Vaswani et al., “Attention Is All You Need”, Neurips 2017

Transformer Architecture

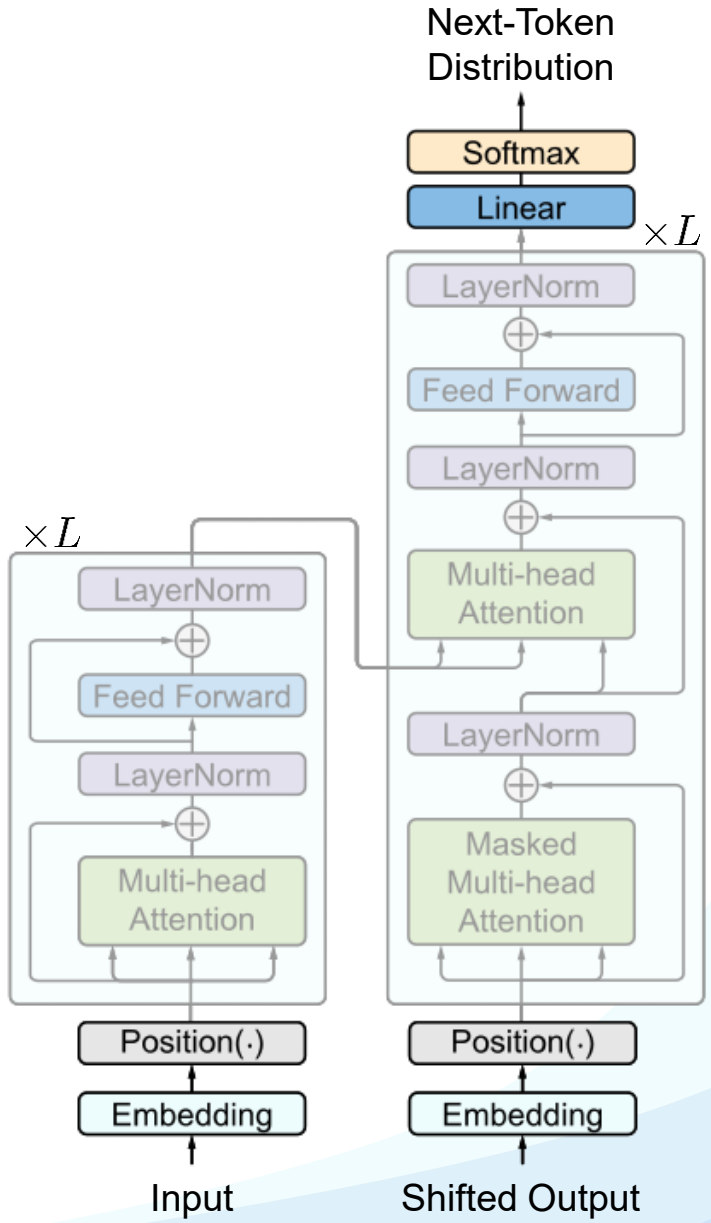
- Hugely successful architecture, based on attention, MLPs, and normalization layers.
- The original transformer design had an encoder-decoder structure.
  - Designed for NLP sequence transduction tasks, e.g., machine translation.
  - Encoder computes features from inputs to provide context.
  - Decoder allows efficient training and auto-regressive decoding during inference.



A. Vaswani et al., “Attention Is All You Need”, Neurips 2017

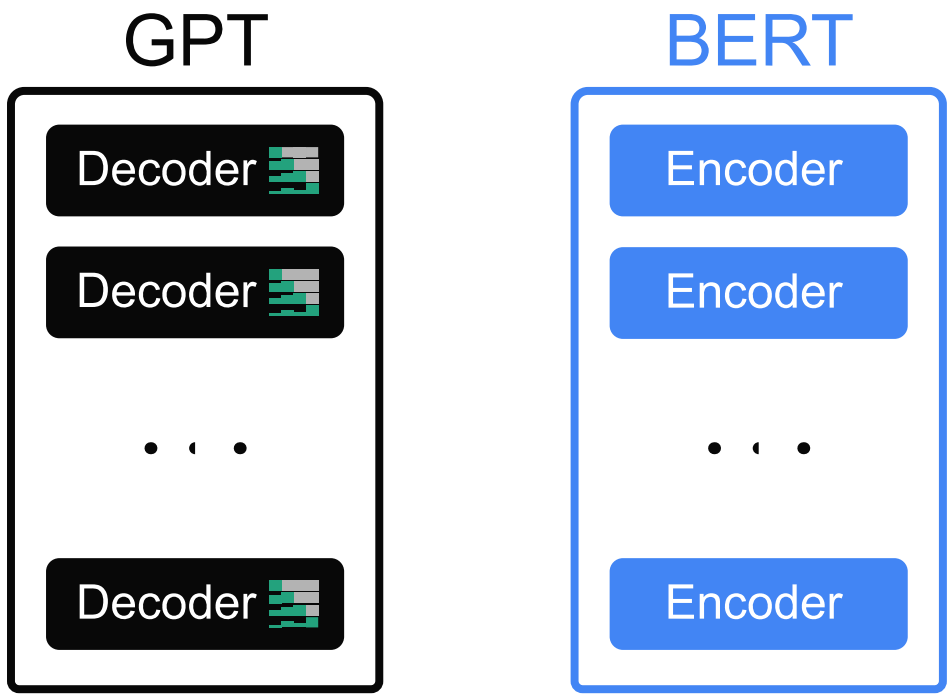
Inputs and Outputs

- Trained on input-output pairs of sequences.
  - E.g., English-German sentence pairs.
- Text is split into word-like units from the vocabulary (tokenization) and linearly embedded into feature space.
  - Input embedding and last linear layer share weights.
- Add positional encoding to both encoder and decoder.
- The output sequence is shifted one position to the right to model conditional probability of next token.



Transformer Architectures: Bert and GPT

- Both models are transformer-based language models.
  - Main operation is attention.
  - Pretrained on large corpora of text using self-supervised objectives.
  - Finetuned for specific tasks and data.
  - Comparable size of models and datasets.
  - But quite different characteristics.
- Both models have seen lots of improvements over time!
  - The general characterization still applies.



J. Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, NAACL 2019

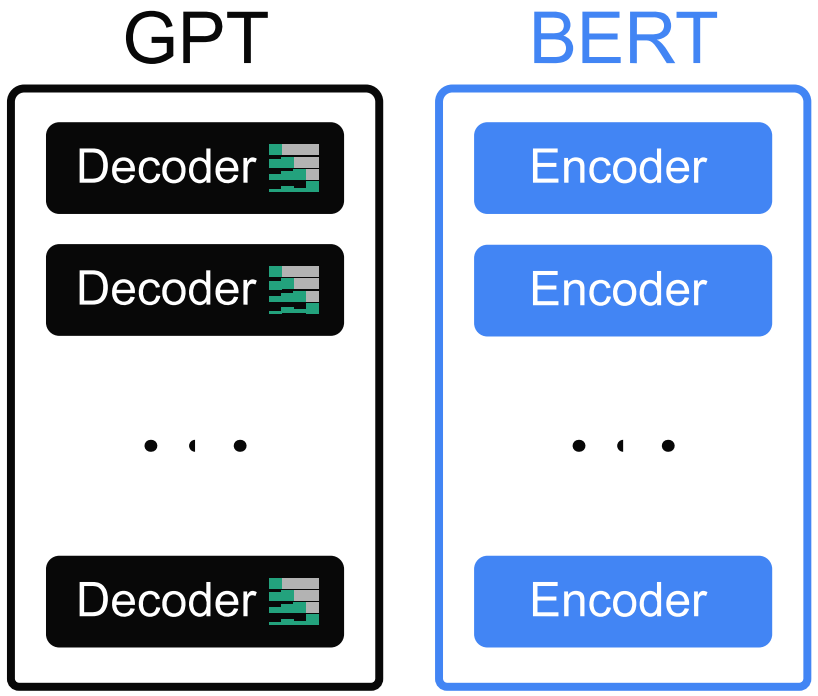
A. Radford et al., “Improving Language Understanding by Generative Pre-Training”, openai.com 2018

Architecture

Transformer Decoder.

Masked Self-Attention (Unidirectional)

Tokens may only attend to preceding tokens.



Transformer Encoder.

Full Self-Attention (Bidirectional)

Every token may attend to every other token.

Note: no Cross-Attention like in the original transformer decoder.

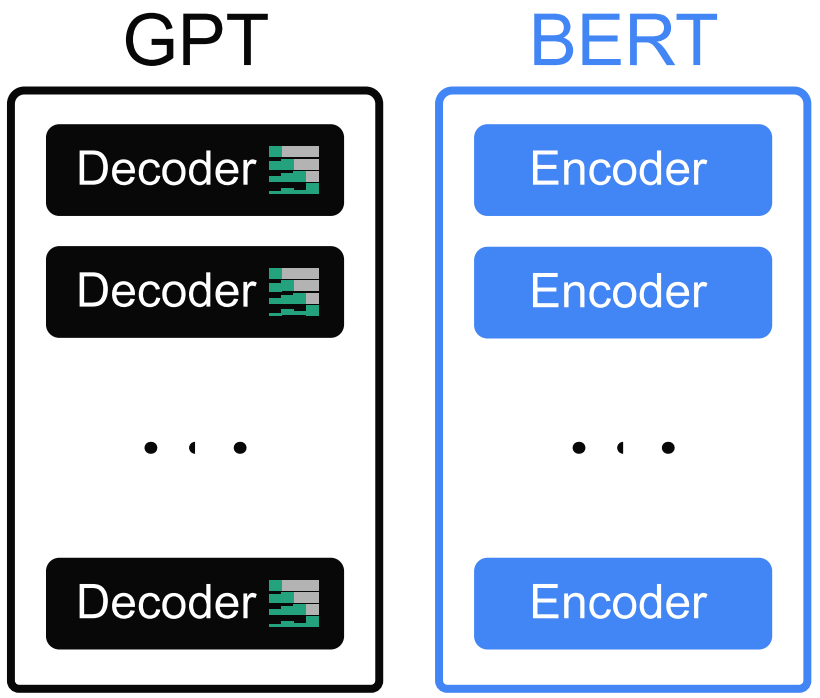
Pre-training

Next-word prediction:
predict next token in a corpus.

<bos> Deep Learning is Fun!

↓

Deep Learning is Fun! <eos>



Masked Language Modeling:
reconstruct randomly masked out words in a sentence.

[Mask] Learning is [Mask]

↓

Deep Learning is Fun!

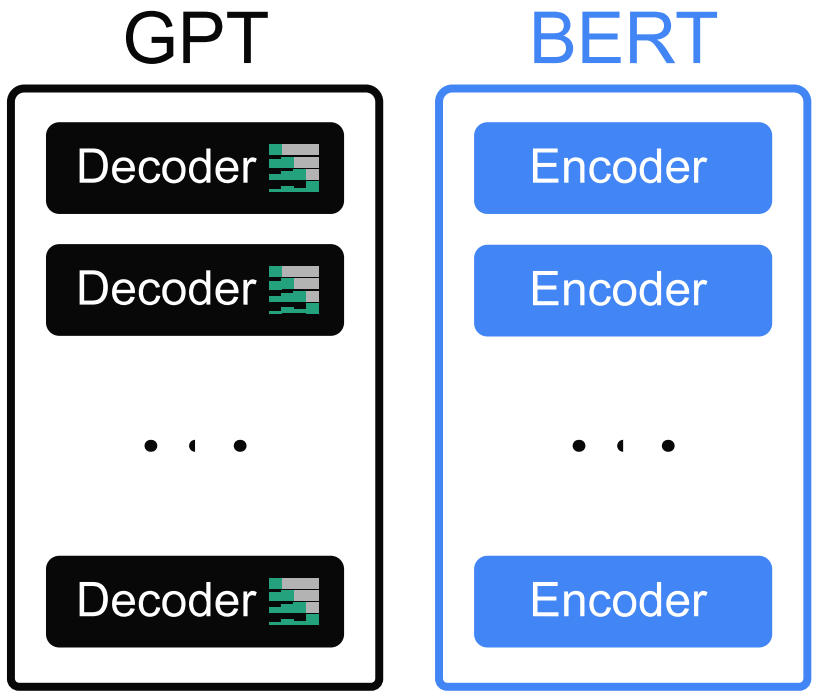
Next Sentence Prediction:
decide whether two sentences follow each other.

Deep Learning is Fun!
The cat is cute. → ✓/✗

Finetuning

Finetuned for language generation tasks.

- Chat
- Translation
- Question Answering



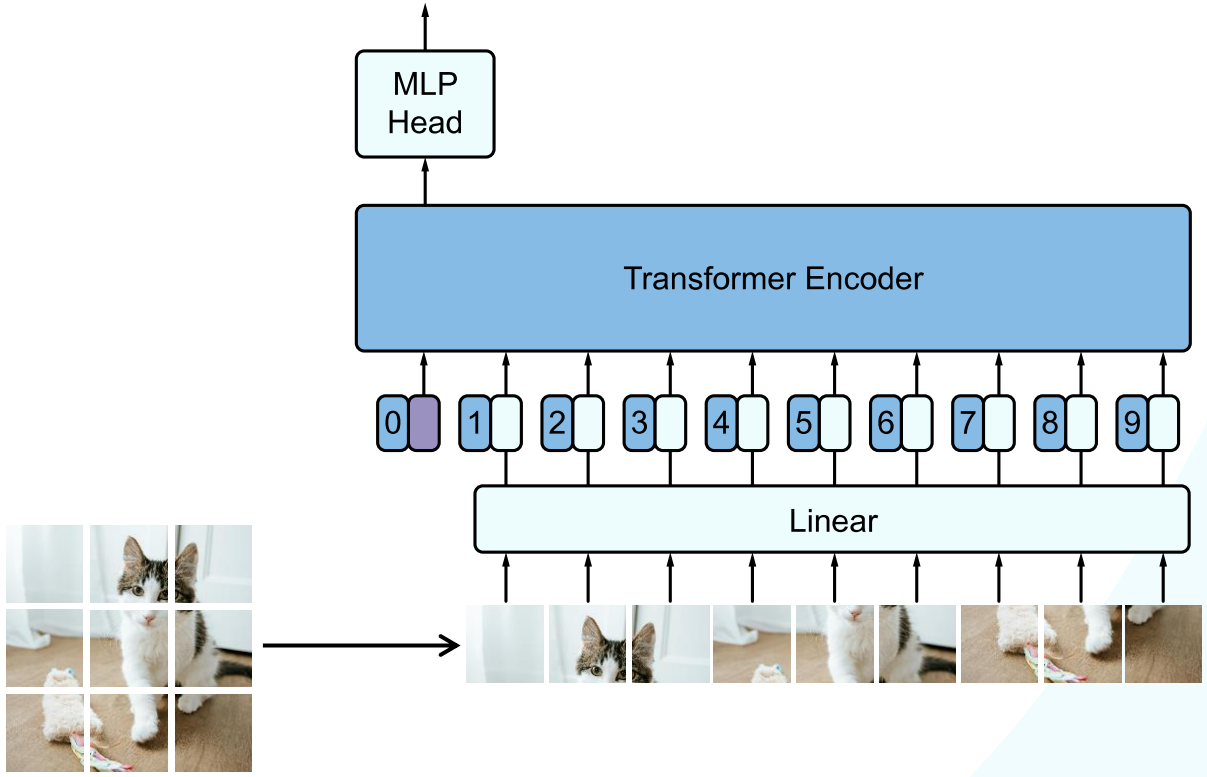
Finetuned for language interpretation tasks.

- Sentiment Analysis
- Text Embeddings
- Text Comprehension

Vision Transformer (ViT)

- The first fully transformer-based architecture for vision tasks.
  - Does not use convolutions (directly).
  - Based on a transformer encoder.

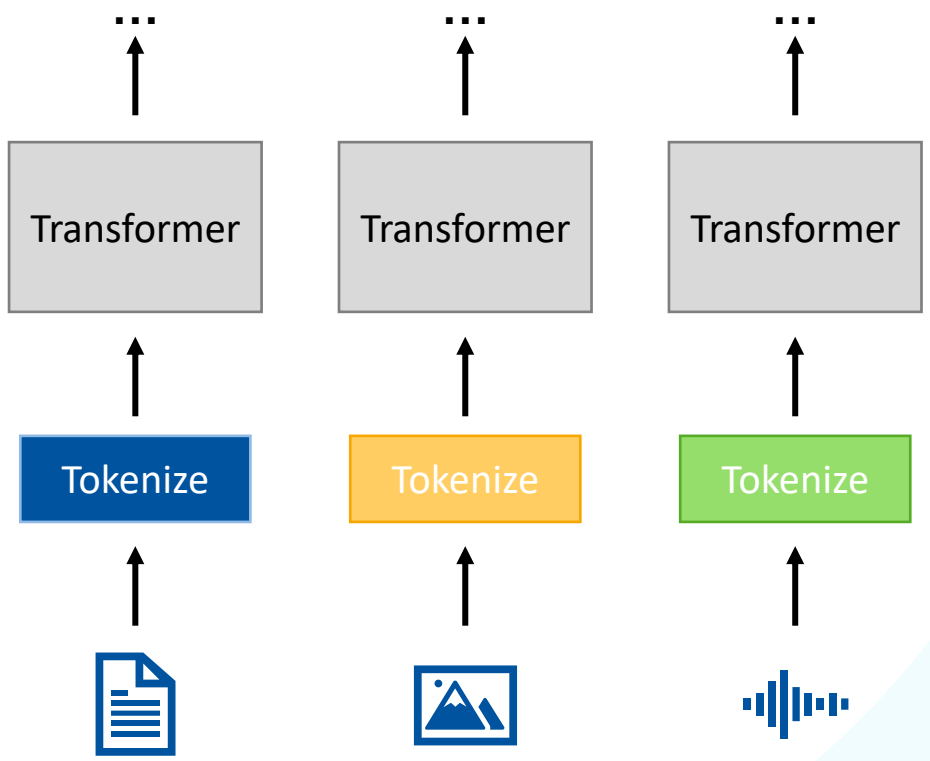
- Common architecture simplifies inter-connection of vision and language models.
  - NLP: tokens are sentence pieces.
  - Vision: tokens are image pieces.
  - Transformer implicitly learns to relate different modalities to each other.



A. Dosovitskiy et al., “An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

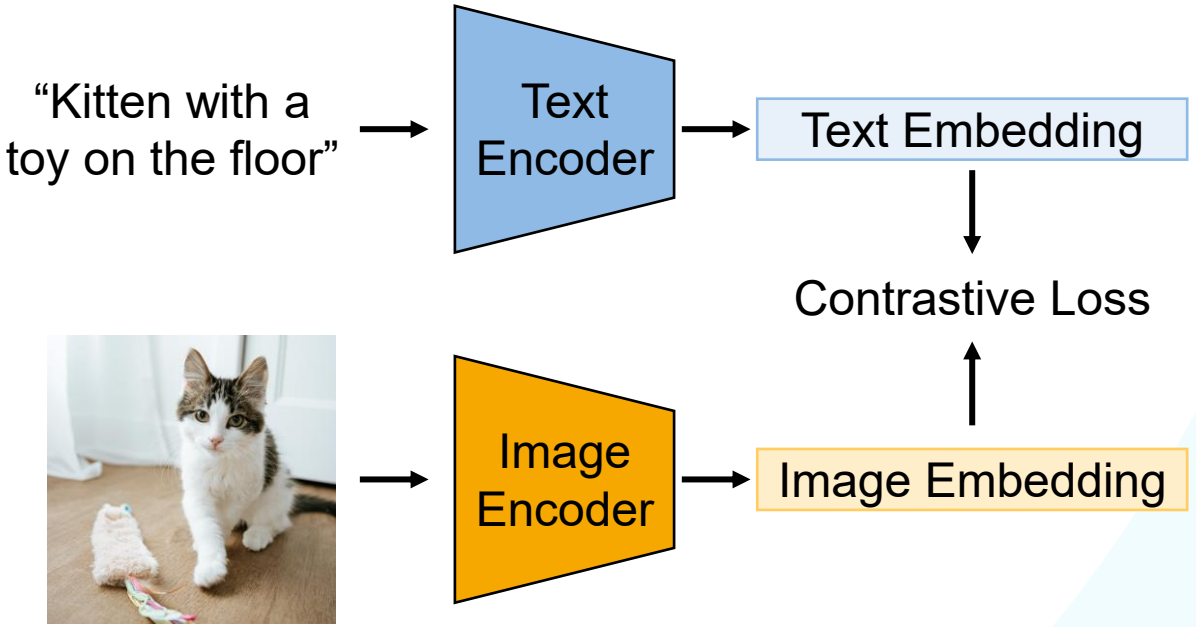
Multimodal Token Streams

- The transformer architecture has been shown to work well with many modalities.
  - Just interpret data as a set of tokens.
- With similar architectures and data representations, we can easily mix-and-match to combine models and modalities.
  - E.g., train shared embedding space for multiple modalities.
  - Learn single model to handle multi-modal inputs and outputs.



Shared Embedding Spaces

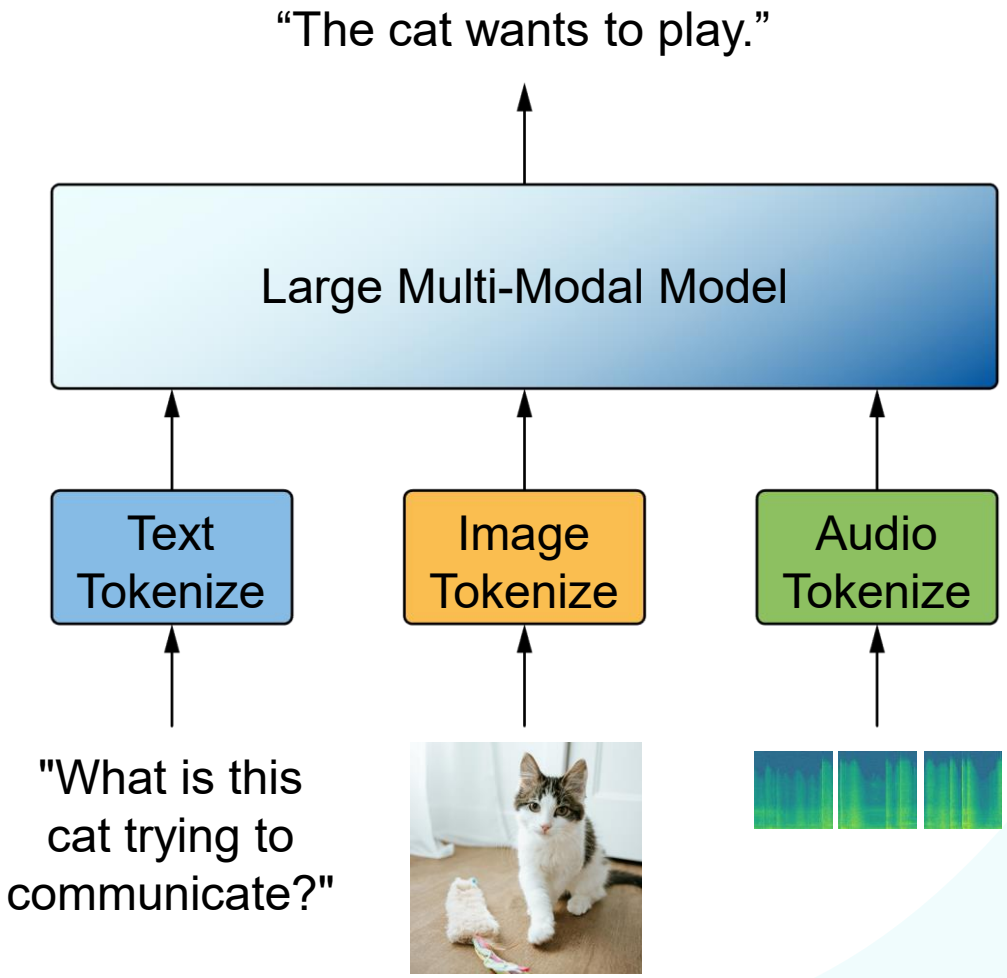
- Relate matching samples from different modalities with contrastive loss.
  - Requires pairs of samples, e.g., image and matching text.
  - Embed both samples into feature space.
  - Apply contrastive loss, pulling matching pairs together while repelling non-matching concepts.
  - Prominent example: CLIP



A. Radford et al., “Learning Transferable Visual Models From Natural Language Supervision”, ICML 2021

Multi-Modal Transformer Models

- Large transformer model with modality specific heads.
  - May use pretrained LLM as core model.
  - Initialize modality-specific heads from pretrained shared-embedding models.
- May also output additional modalities, e.g., image tokens to generate images.



H. Liu et al., "Visual Instruction Tuning", Neurips 2023

Chameleon Team (FAIR), "Chameleon: Mixed-Modal Early-Fusion Foundation Models", Arxiv 2024

References and Further Reading

- More information about Neural Networks and Deep Learning is available in the following book.

I. Goodfellow, Y. Bengio, A. Courville
Deep Learning
MIT Press, 2016

https://www.deeplearningbook.org/