Foundations of Deep Learning

Lecture 09

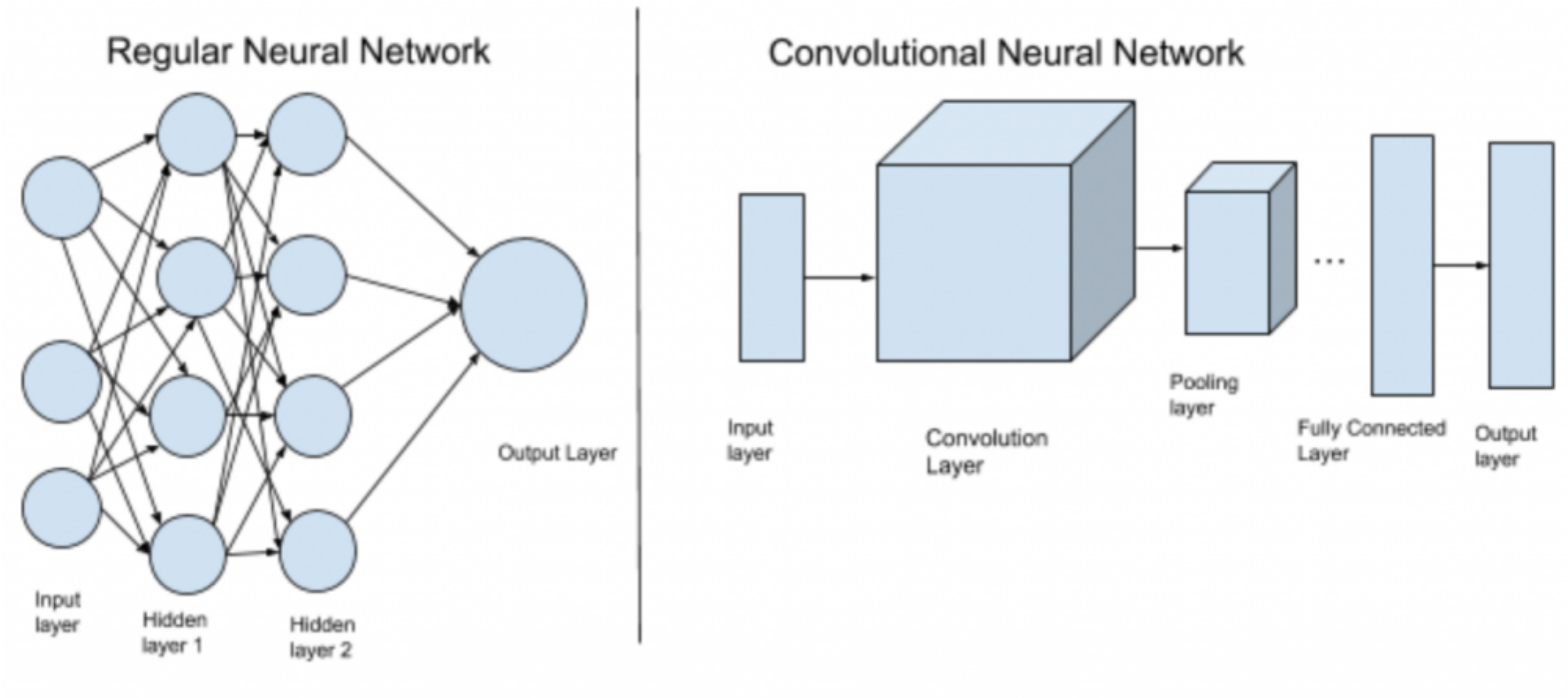
Neural Network Architectures

Aurelien Lucchi

Fall 2024

Constraining the Model

- Unlike MLPs which are characterized by dense layers, convolutional NNs are made up of sequential arrangement of convolutional and pooling layers
- A single convolutional layer in turn comprises multiple filters/feature maps



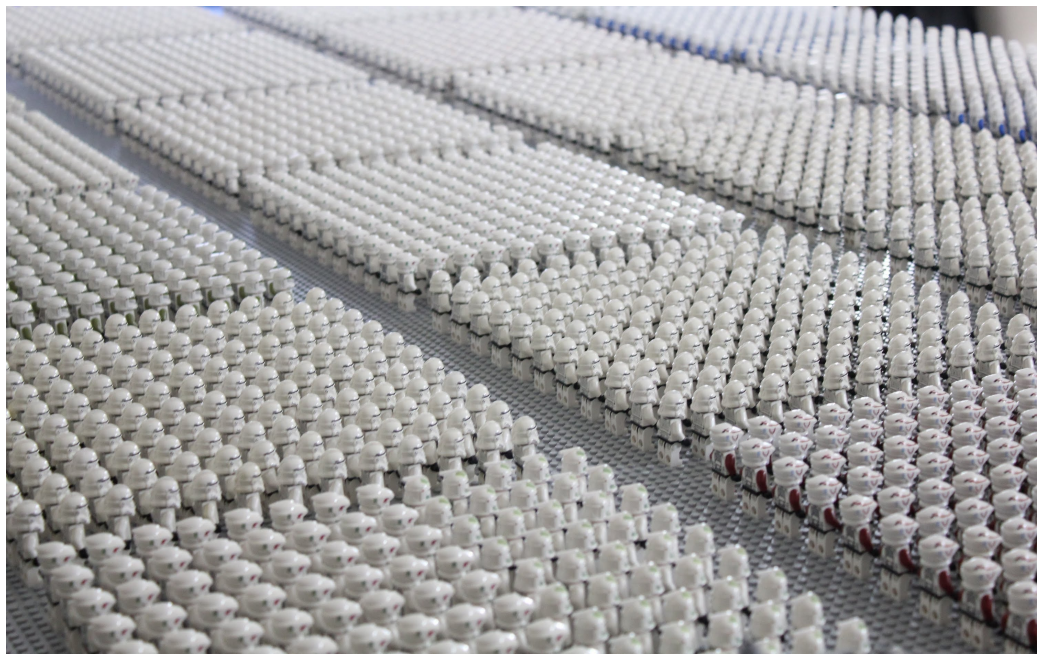
Constraining the Model

- No machine learning model can do well on all problems.
- Need to constrain the function class appropriately.
- Let's focus on the problem of image classification.

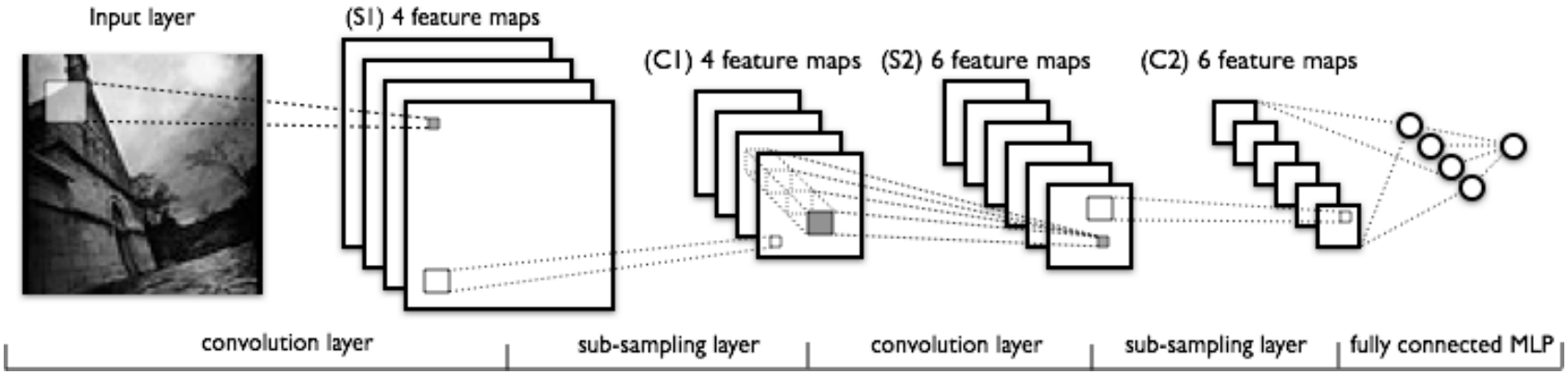
Can you name one property that is essential for this problem?

Example: Translation Invariance

- Translation invariance of images
  - image patches look the same, irrespective of their location
  - idea: extract translation invariant features



CNN: Buildings Blocks



- **Four building blocks:**
  - Convolutional layer
  - Pooling/subsampling layer
  - Fully-connected layer
  - Output: For binary classification, use a sigmoid function:

$$y_1 = P(Y = 1 \,|\, \mathbf{x}) = \frac{1}{1 + \exp\left[-\mathbf{w}^\top \mathbf{x}\right]}$$

Section 1

Convolutional Networks

Subsection 1

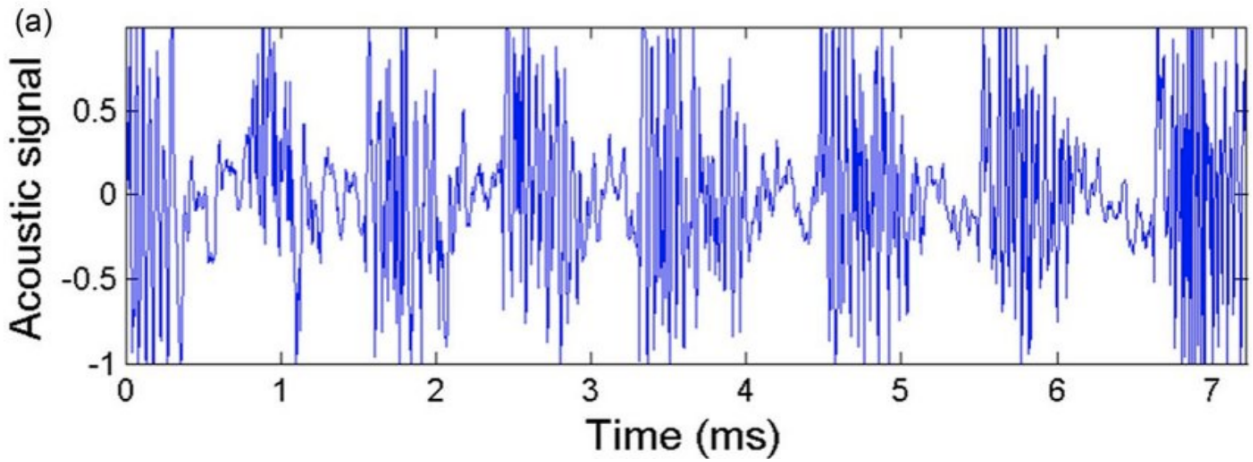
Convolution

Signals

We are interested in processing **signals**.

Signals may be sounds, images, measurement series, etc.

For concreteness: time signal $f : \mathbb{R} \to \mathbb{R}$.



Signal Transformation

Operating on a signal: **operators** or **transformations**

$$T : f \mapsto Tf, \quad Tf : \mathbb{R} \to \mathbb{R}$$

The result is a transformed signal.

Analogy: extract features from raw data representation.

Transformed signal extracts relevant information.

Integral Operators

How can we **represent** an interesting class of such operators?

Via Integrals!

Chose **kernel** $H : \mathbb{R}^2 \to \mathbb{R}$ and interval $-\infty \leq t_1 < t_2 \leq \infty$

This defines an integral operator

$$(Tf)(u) = \int_{t_1}^{t_2} H(u,t) f(t)\, dt$$

- f is the function being transformed
- $H(u,t)$ is the **kernel** of the operator, which determines the nature of the transformation, e.g., smoothing, scaling, or shifting f
- assuming that the integral exists (condition on f)

Fourier Transform

The Fourier transform $\mathcal{F}$ of a real-valued function f is an integral operator defined via

$$t_1 = -\infty, t_2 = \infty$$

$$H(u,t) = \exp(-2\pi i t u)$$

such that

$$(\mathcal{F}f)(u) := \int_{-\infty}^{\infty} e^{-2\pi i\, tu}\, f(t)\, dt$$

Convolution

Given two functions f, h, their convolution is defined as

$$
\begin{aligned}
(f * h)(u) &:= \int_{-\infty}^{\infty} h(u-t) f(t)\, dt \\
&= \int_{-\infty}^{\infty} \underbrace{f(u-t)}_{\substack{\text{Move to desired time} \\ \text{\& flip signal}}} h(t)\, dt \\
&= (h * f)(u).
\end{aligned}
$$

where the second equality is due to a change of variables.

- corresponds to an integral operator with kernel $H(u,t) = h(u-t)$ operating on f
- by symmetry one can also think of $F(u,t) = f(u-t)$ operating on h

Convolution

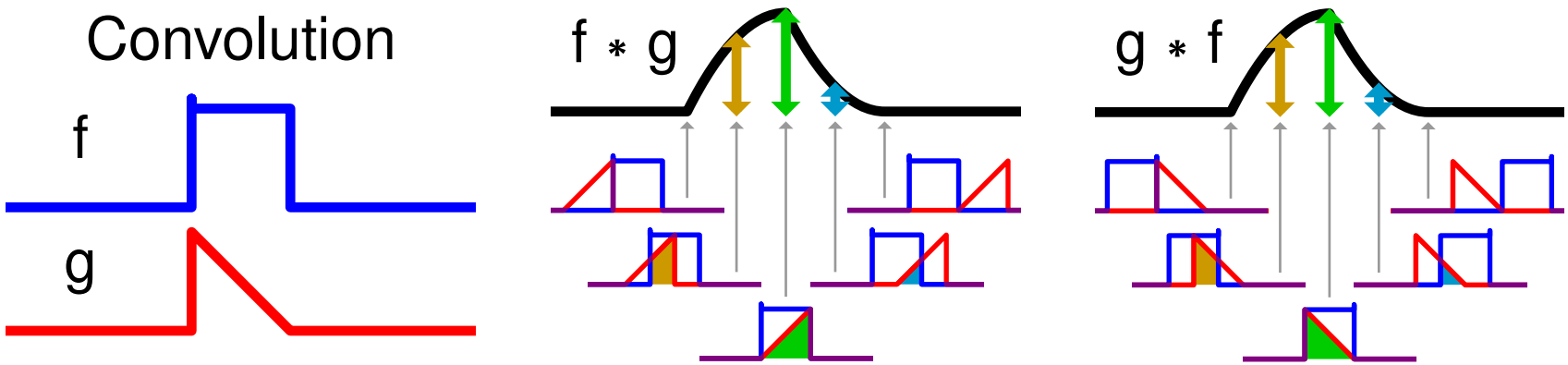


Figure: Source: wikipedia

Shift Equivariance

Shifted function f_Δ

$$f_\Delta(t) := f(t + \Delta)$$

Convolution is shift-equivariant, i.e.

$$\Rightarrow \boxed{f_\Delta * h = (f * h)_\Delta}$$

$$\begin{aligned}(f_\Delta * h)(u) &= \int_{-\infty}^{\infty} h(u - t) f(t + \Delta)\, dt \\ &\overset{t \mapsto t - \Delta}{=} \int_{-\infty}^{\infty} h(u + \Delta - t) f(t)\, dt \\ &= (f * h)(u + \Delta) = (f * h)_\Delta(u)\end{aligned}$$

Comments

1. The convolution operator is commutative, one can exchange the function and the kernels.

2. Its existence depends on integrability properties of f, h.

3. Typical use: $f =$ signal, $h =$ fast decaying kernel function, often of finite support.

Linear Shift-Equivariant Transforms

Instead of defining convolutions as a special case of an integral transform, one can take a principled approach and start with a question ...

How can we characterize **linear** shift-equivariant operators?

$$\boxed{T(\alpha f + \beta g) = \alpha T f + \beta T g, \quad \forall f, g; \forall \alpha, \beta \in \mathbb{R}}$$

$$\boxed{(T f_\Delta)(t) = (T f)(t + \Delta)}$$

Linear Shift-Equivariant Transforms

Theorem 1
Any linear, translation-invariant transformation T can be written as a convolution with a suitable h.

Discrete Convolution

In practice: signals (digitally) sampled. Need to consider discrete case.

Let $f, h : \mathbb{Z} \to \mathbb{R}$. Define discrete convolution via

$$\boxed{(f * h)[u] := \sum_{t=-\infty}^{\infty} f[t]\, h[u-t]}$$

- use of rectangular brackets to suggest "arrays"
- typical choice of h: support over finite window, e.g. $h[t] = 0$ for $t \notin [t_{\min}; t_{\max}]$.

Gaussian Kernel

Let us define a small Gaussian kernel with support $[-2:2] \subset \mathbb{Z}$.

$$h[t] = \frac{1}{16} \begin{cases} 6 & t = 0 \\ 4 & |t| = 1 \\ 1 & |t| = 2 \\ 0 & \text{otherwise} \end{cases}$$

Because of finite support, convolution sum can be truncated

$$\begin{aligned} (f * h)[u] &= \sum_{t=u-2}^{u+2} f[t]\, h[u-t] = \sum_{t=-2}^{2} h[t]\, f[u-t] \\ &= \frac{6f[u] + 4f[u-1] + 4f[u+1] + f[u-2] + f[u+2]}{16} \end{aligned}$$

Cross–correlation

Sibling to convolutions: **cross–correlation**.

In the discrete case:

$$\boxed{(h \star f)[u] := \sum_{t=-\infty}^{\infty} h[t]\, f[u+t]}$$

- sliding inner product
- difference to convolution: $u+t$ instead of $u-t$.
- flipped over kernel

$$\Rightarrow \boxed{(h \star f) = (\bar{h} * f), \quad \text{where } \bar{h}[t] := h[-t]}$$

Toeplitz Matrices

In practice: signal f and kernel h finitely supported

Without loss of generality assume $f[t] = 0$ for $t \notin [1:n]$, $h[t] = 0$ for $t \notin [1:m]$, $m \leq n$.

We can then think of f and h as vectors and define

$$(f * h) = \underbrace{\begin{pmatrix} h_1 & 0 & 0 & 0 & \dots & 0 & 0 \\ h_2 & h_1 & 0 & 0 & \dots & 0 & 0 \\ h_3 & h_2 & h_1 & 0 & \dots & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & 0 & 0 & \dots & h_m & h_{m-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & h_m \end{pmatrix}}_{:=\mathbf{H}_n^h \in \mathbb{R}^{(n+m-1)\times n}} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \dots \\ f_n \end{pmatrix}$$

Subsection 2

Convolution Networks

Convolutional Networks: Goal

What is our goal with convolutional networks?

- a compositional (i.e. layered) architecture for signal processing
- exploiting **translation equivariance**
- exploiting locality and scale (e.g. temporal, spatial)
- learning (parameterized) kernel functions or **filters**
- increased efficiency relative to fully-connected DNNs

Convolutions in Higher Dimensions

Generalize concept of convolution to ...

- $2D$: e.g. images, spectograms
- $3D$: e.g. color or multi-spectral images, voxel images, video
- or even higher dimensions

Replace vectors by

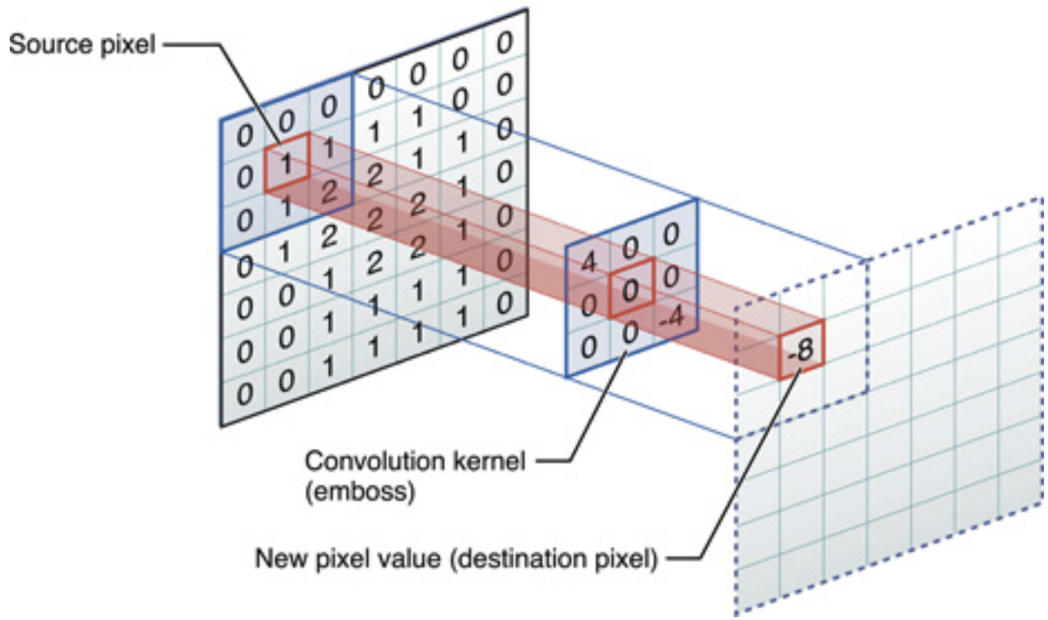
- matrices or fields (e.g. in discrete case)

$$(F * G)[i, j] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} F[i-k, j-l] \cdot G[k, l]$$

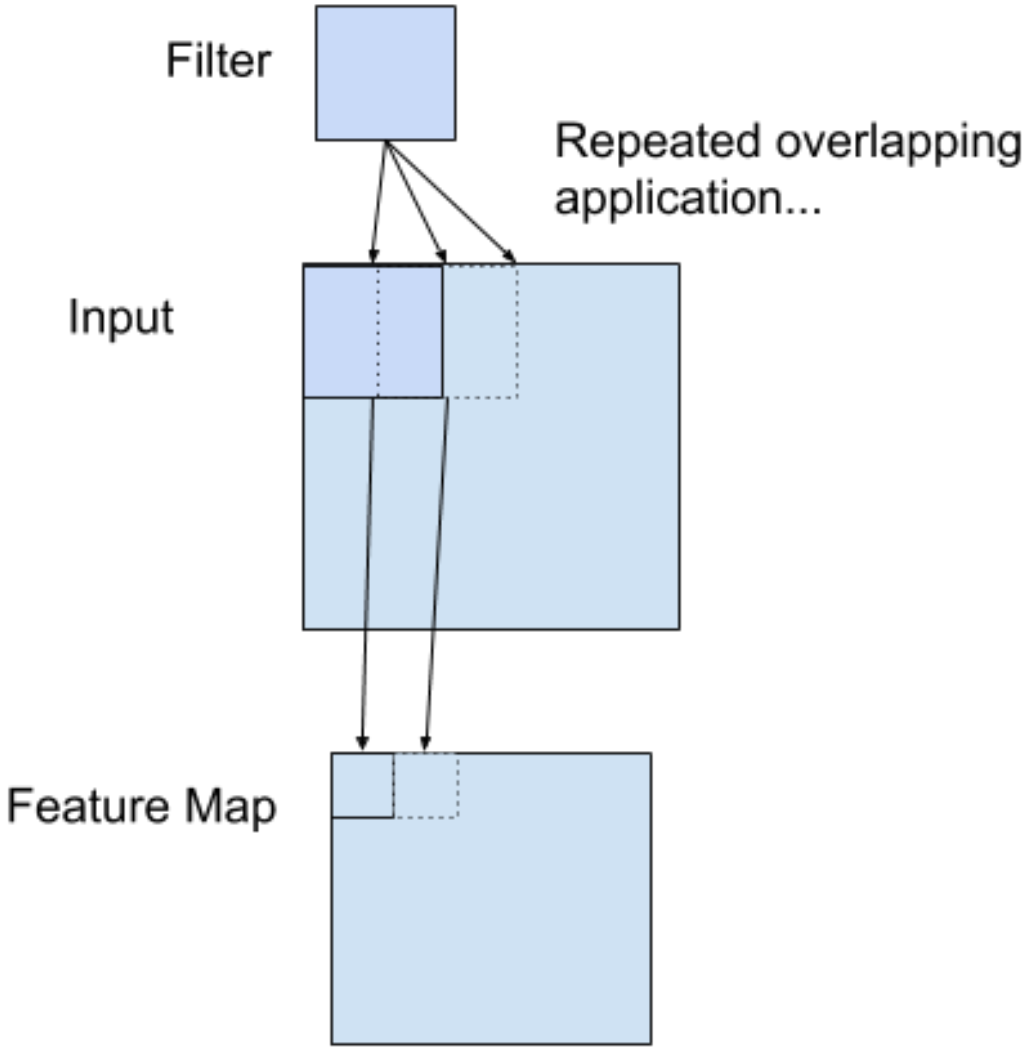
- tensors: for $3D$ and higher

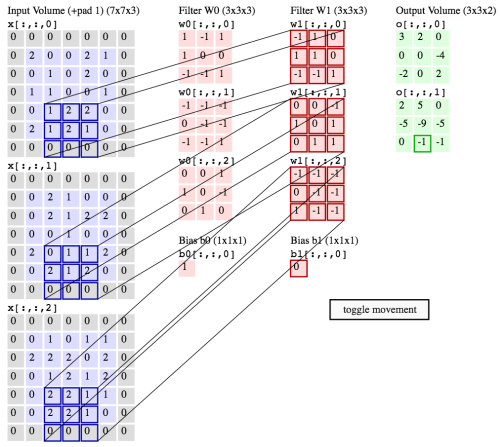
Convolutional Layers: Layout

- Convolved signal **inherits** topology of original signal.
- Hence: units in a convolutional layer are typically arranged on the same grid (1D, 2D, 3D, ...)



Convolutional Layers: Animation





cs231n.github.io/assets/conv-demo/index.html

Convolutional Layers: Border Handling

Different options for border handling

1. padding with zeros
2. only retain values from windows fully contained in support of signal f

Convolutional Layers

- **Convolution:**
  - Mathematical operation on two functions (f and g)
  - It produces a third function that is typically viewed as a modified version of one of the original function
  - This operation can be used to detect edges in an image

-1	0	1
-2	0	2
-1	0	1

Horizontal

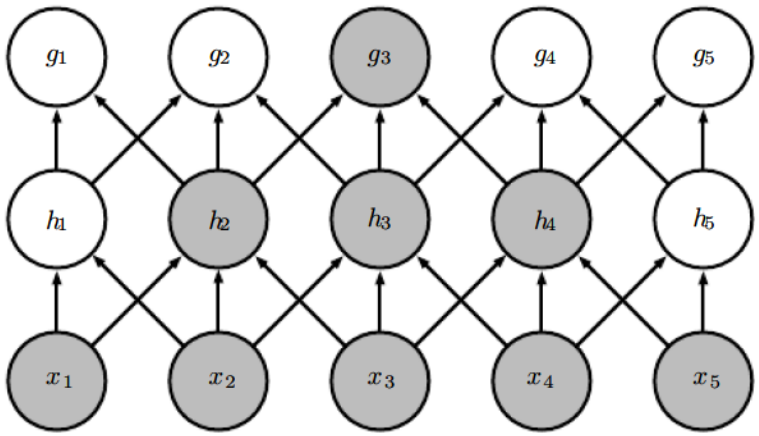
-1	-2	-1
0	0	0
-1	-2	-1

Vertical

Important: In a CNN, the filters are learned!

Nested convolutions & Receptive field

Schematic view of the notion of a **receptive field** in a deep network (input $x \rightarrow$ layer $h \rightarrow$ layer g)



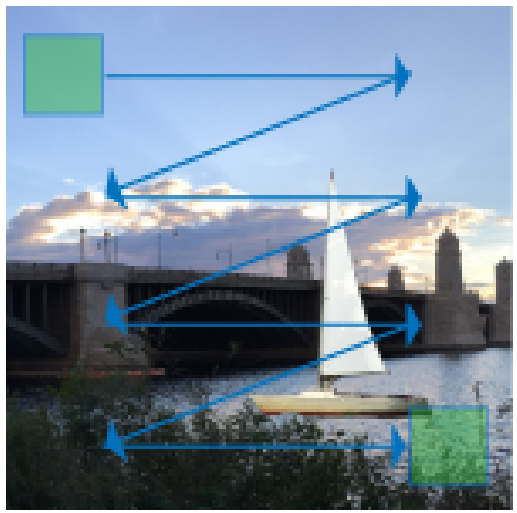
- nesting of convolutions: receptive fields grow
- encourage network to first extract **localized** features
- subsequent layers: less and less localized features

Weight Sharing

- **MLP: No weight sharing**
  - each neuron is parametrized with weights $\mathbf{w} \in \mathbb{R}^n$
- **Weight Sharing**
  - neurons share the same weights = compute same function
  - differ in location of their receptive field = **different input**
  - multiple channels = multiple filters

-1	0	1
-2	0	2
-1	0	1

Filter =
set of weights



Efficient Computations of Convolutional Activities

FFT (**Fast Fourier Transform**): compute convolutions fast(er).

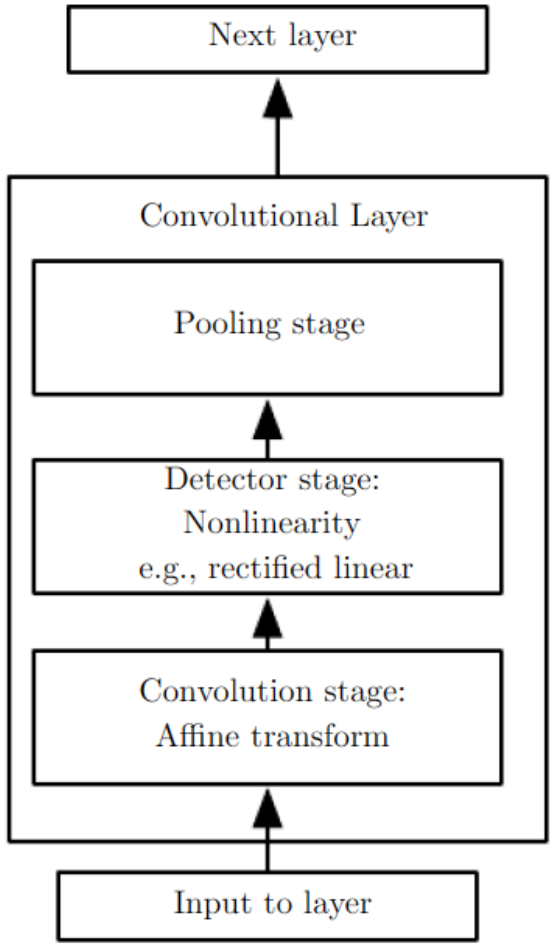
1. Fourier transform of signal $f \to (\mathcal{F}f)$ and kernel $h \to (\mathcal{F}h)$
2. pointwise multiplication and inverse Fourier transform

$$(f * h) = \mathcal{F}^{-1}\left((\mathcal{F}f) \cdot (\mathcal{F}h)\right)$$

3. FFT: signal of length n, can be done in $\mathcal{O}(n \log n)$
4. pays off, if many channels (amortizes computation of $\mathcal{F}f$)
5. small kernels ($m < \log n$): favor time/space domain

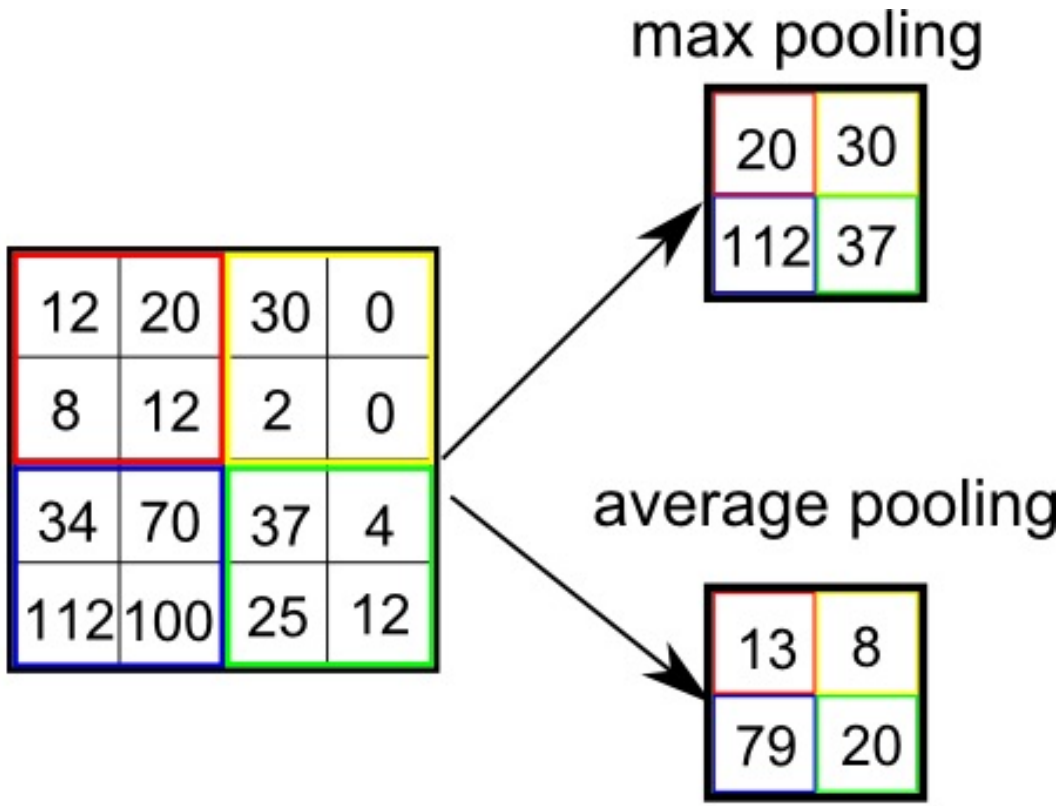
Convolutional Layers: Stages

1. **Non-linearities**: detector stage. As always: scalar non-linearities (activation function)
2. **Pooling** stage: locally combine activities



Pooling/subsampling layer

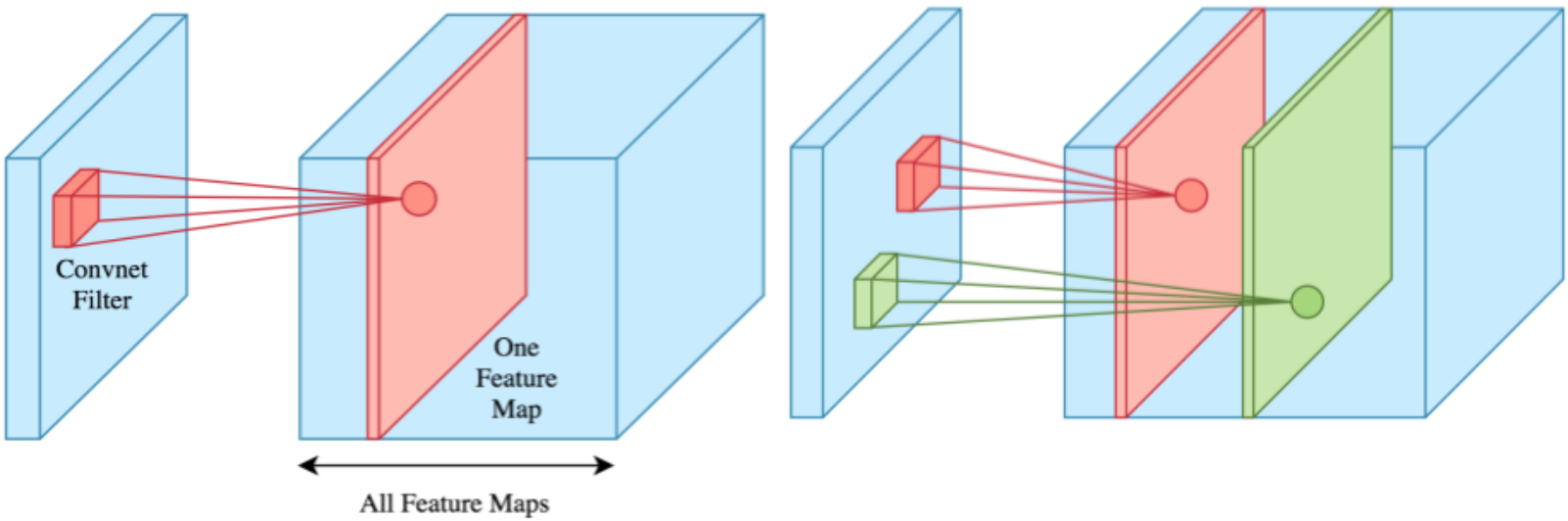
- Reduce size of convolutional layers by down-sampling
- Take average over window (e.g. 2x2)
- Common practice: max pooling = take maximum in window



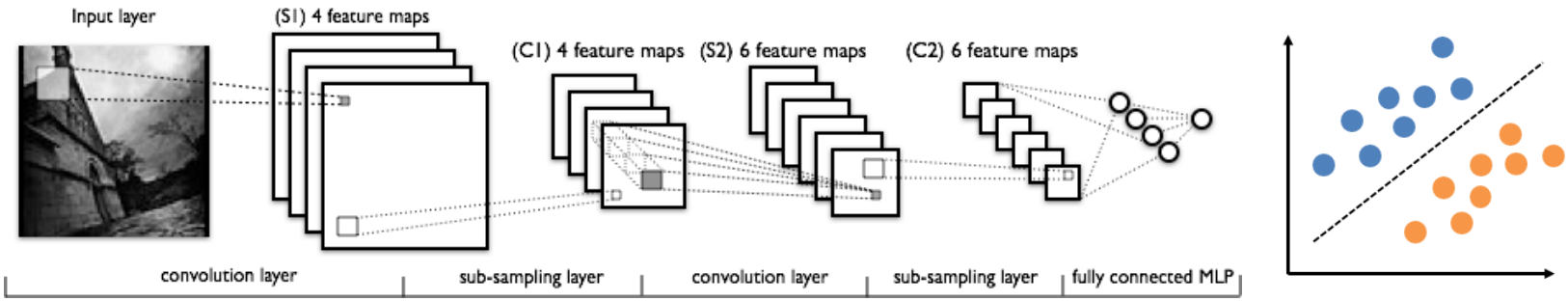
Channels

Learn multiple convolution kernels (or filters) = multiple **channels**

- typically: all channels use same window size
- channels form additional dimension for next layers (e.g. 2D signal $\times$ channels = 3D tensor)
- number of channels: design parameter



Fully-connected layer



- High-level reasoning
- Connects all neurons in the previous layer to every single neuron in the fully-connected layer
- Can be computed with a matrix multiplication

Subsection 3

Convolutional Computer Vision

Visual Object Recognition

Important challenge for computer vision systems:

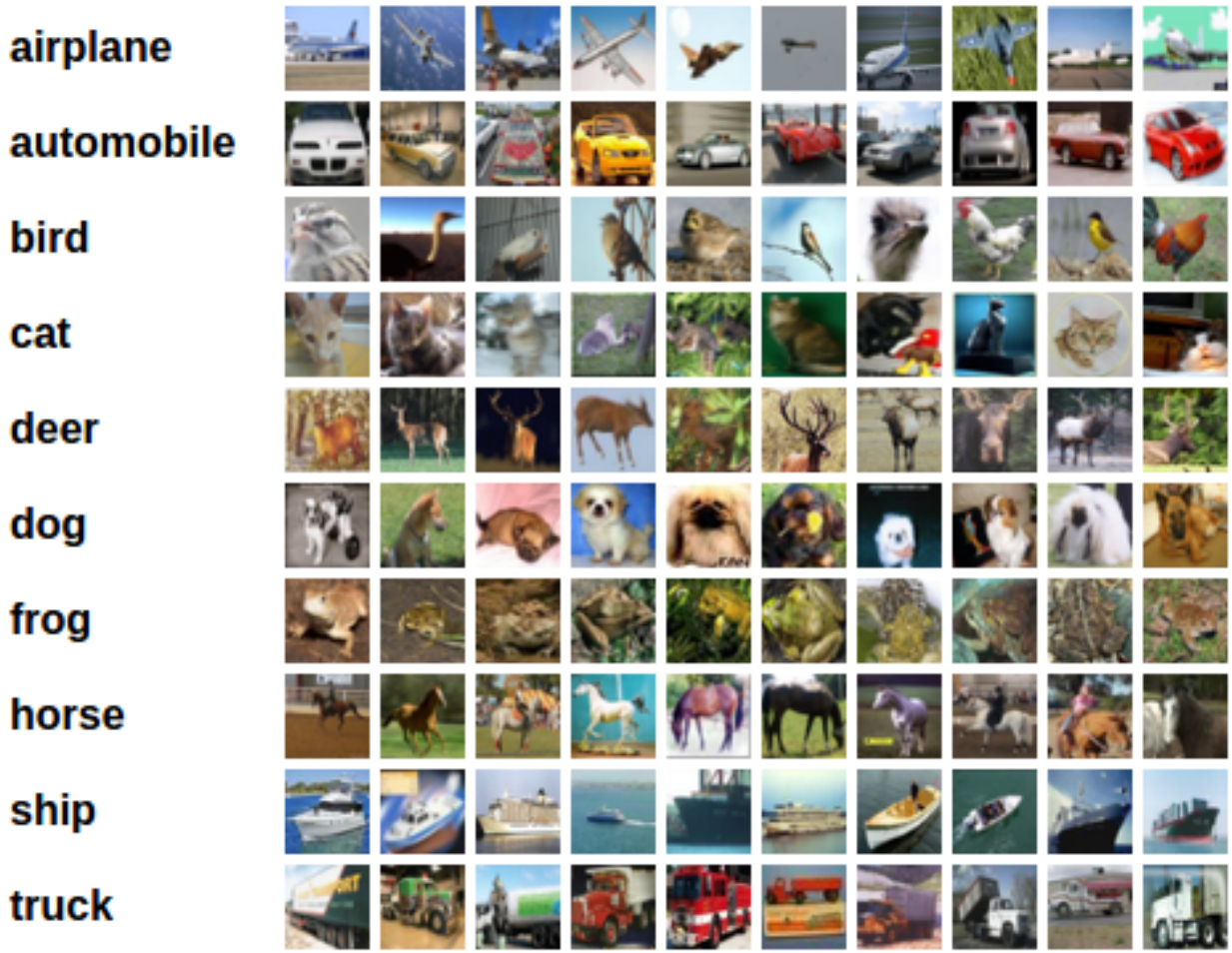
Visual **object recognition**: paradigmatic problem

Convolutional networks have been used with remarkable success (**ImageNet** challenge)

- Pushing recognition to human-level accuracy
- Debatable of whether there is real understanding (adversarial examples, correlated features) of how the models work (lack of interpretability)

Visual Object Recognition: Examples

Examples from CIFAR dataset



ImageNet

Data set: ImageNet (categories derived from WordNet)

... more than 20'000 categories

... more than 10 million hand-labeled images

Convolutions for Images

Color images

- pixel-based image representation: 3d.
- regular two dimensional, spatial sampling grid
- 3 color channels (color space such as RGB).

Convolutional kernels are usually operating on 3-*tensors*.

- channels: permutation-invariant, numbering arbitrary.
- typically fully connected across channels
- combined kernel over $r \times r$ window and m channels.

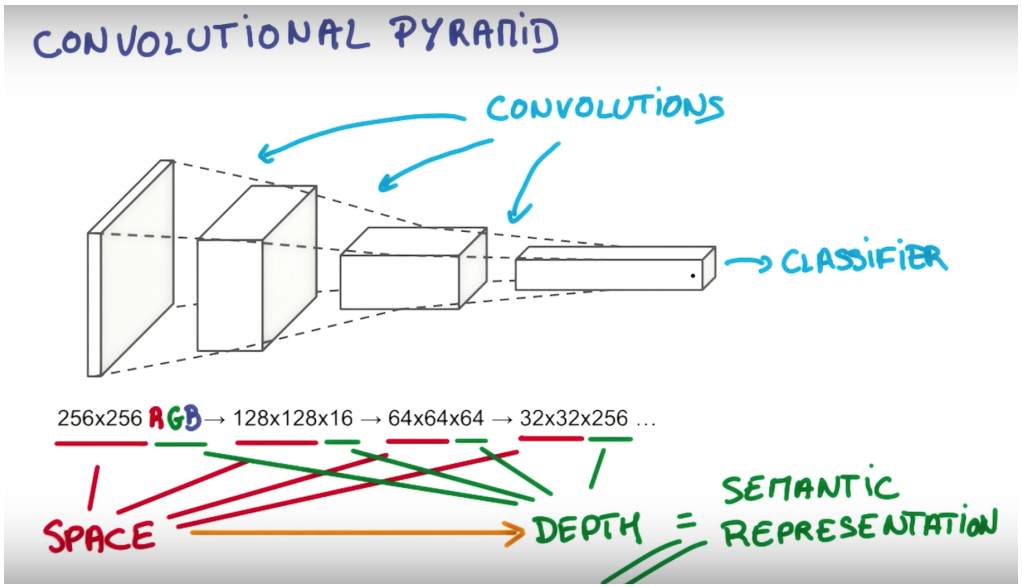
Pyramidal Architecture

Very common high-level model structure: **pyramid**

... stack convolutions, reducing spatial resolution of feature maps

... increasing the number of channels

... in the end: densely connected layer



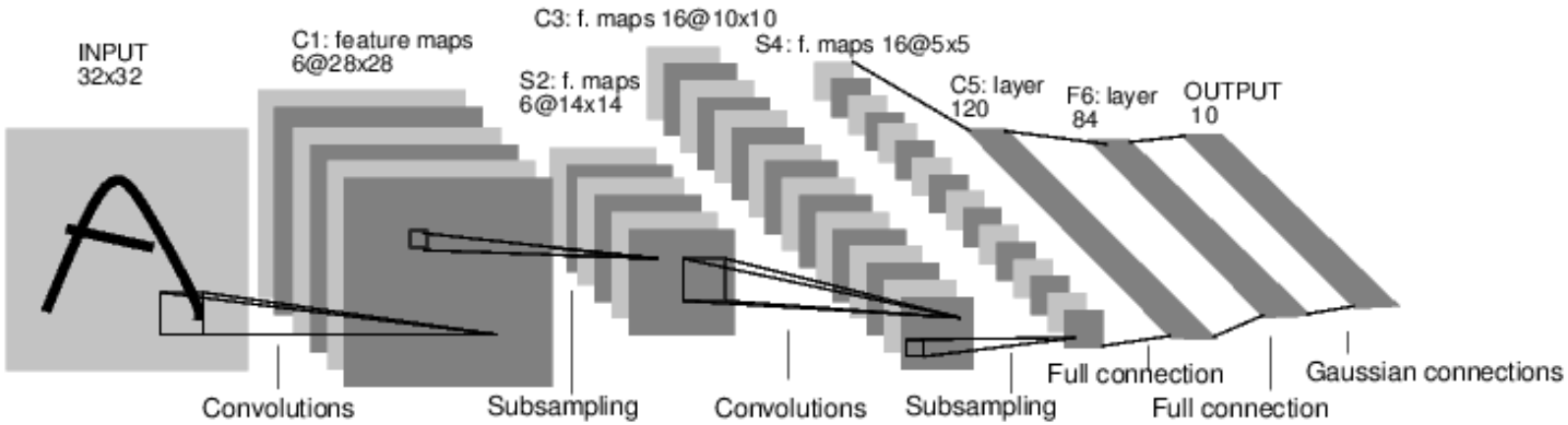
LeNet5

Classical model: LeNet5 (LeCun at al 1989, 1998)

C1/S2: 6 channels, 5×5 kernels, 2×2 subsampling (4704 units)

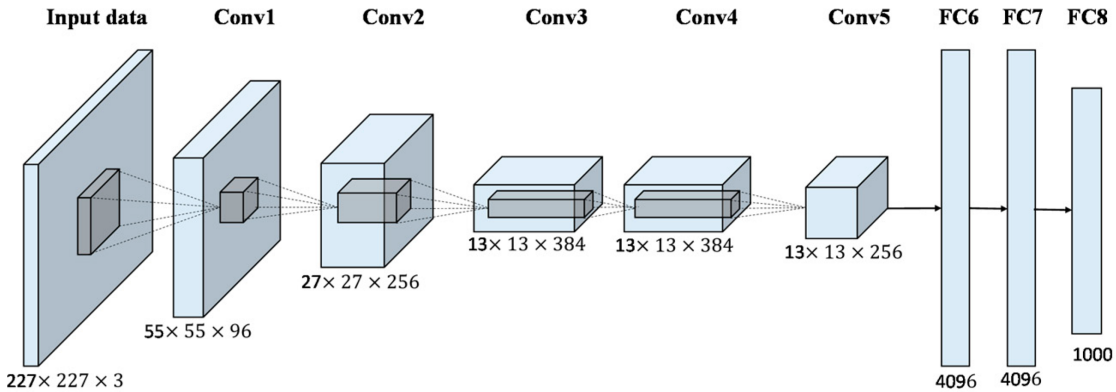
C3/S4: 16 channels, 6×6 kernels, 2×2 subsampling (1600 units)

C5: 120 channels; F6: fully-connected



AlexNet

More modern version: AlexNet (2012, 70k+ citations)



Parameter count:

384 to 384 channels with 3×3 window $\mapsto 1.3M+$ parameters

Layer FC6: $13 \times 13 \times 256$ tensor to $4096 \mapsto 177M+$ parameters.

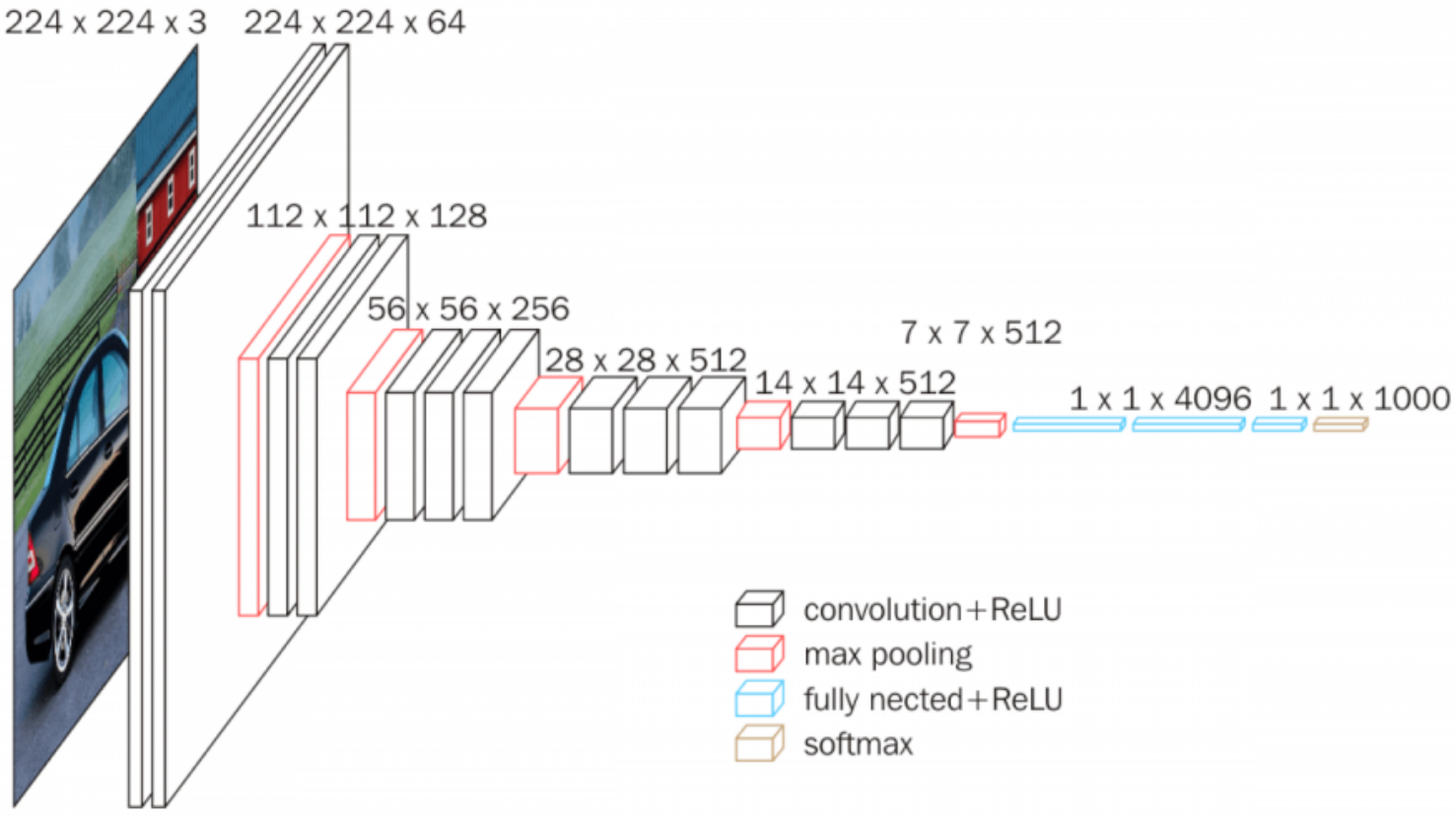
Architectural Considerations

So what matters with such vision architectures?

- control performance (many real-world applications require high throughput) $\Longrightarrow$ avoid blow-up in model size
- make models **deep** (compositionality matters)
- make layers **wide** (allow for a sufficient number of channels)

Have to make trade-offs!

Very Deep Convolutional Networks: VGG



VGG

Traits of the VGG architecture

- very small convolution windows: 3×3
- avoids down-sampling and pooling
- significant increase in depth (relative to AlexNet, LeNet5)
- 3 layer blocks: $3 \cdot (3 \times 3) = 27 < 49 = (7 \times 7)$

Inception

Another popular model w/ deep stacking:

Inception Network (Google)

(1) Uses many channels for high accuracy

(2) Dimension reduction in between convolutions

(3) $1 \times 1 \times m$ convolution $k \leq m$:

$$\mathbf{x}_{ij}^{+} = \sigma\left(\mathbf{W}\mathbf{x}_{ij}\right), \quad \mathbf{W} \in \mathbb{R}^{k \times m}$$

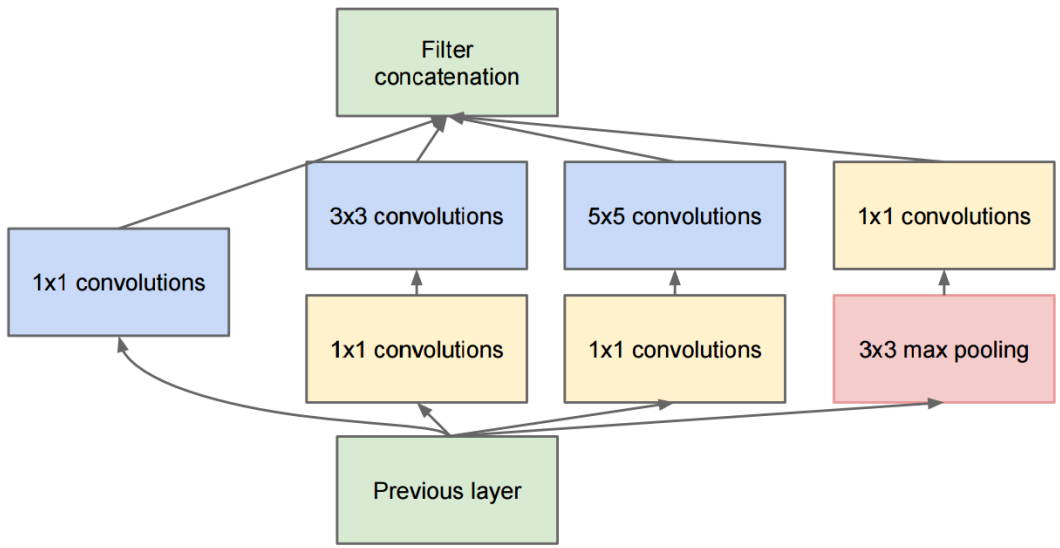
- 1×1: i.e. no spatial convolution
- shared one-layer network transforms feature vectors across channels at each position
- instance of the network within a network design pattern

Inception: Block

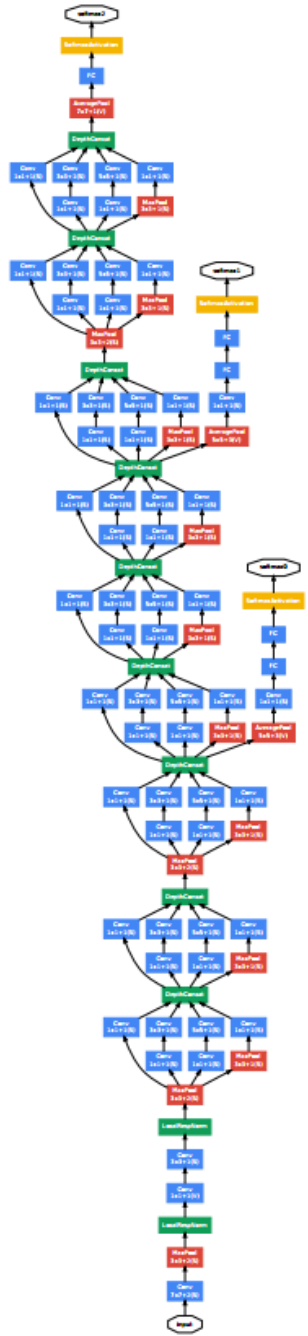
Why chose, if one can have it all?

Multiple processing paths: trade-off of between channel dimensionality vs. window size solved as part of the learning

Create block structures, combined with dimension reduction.

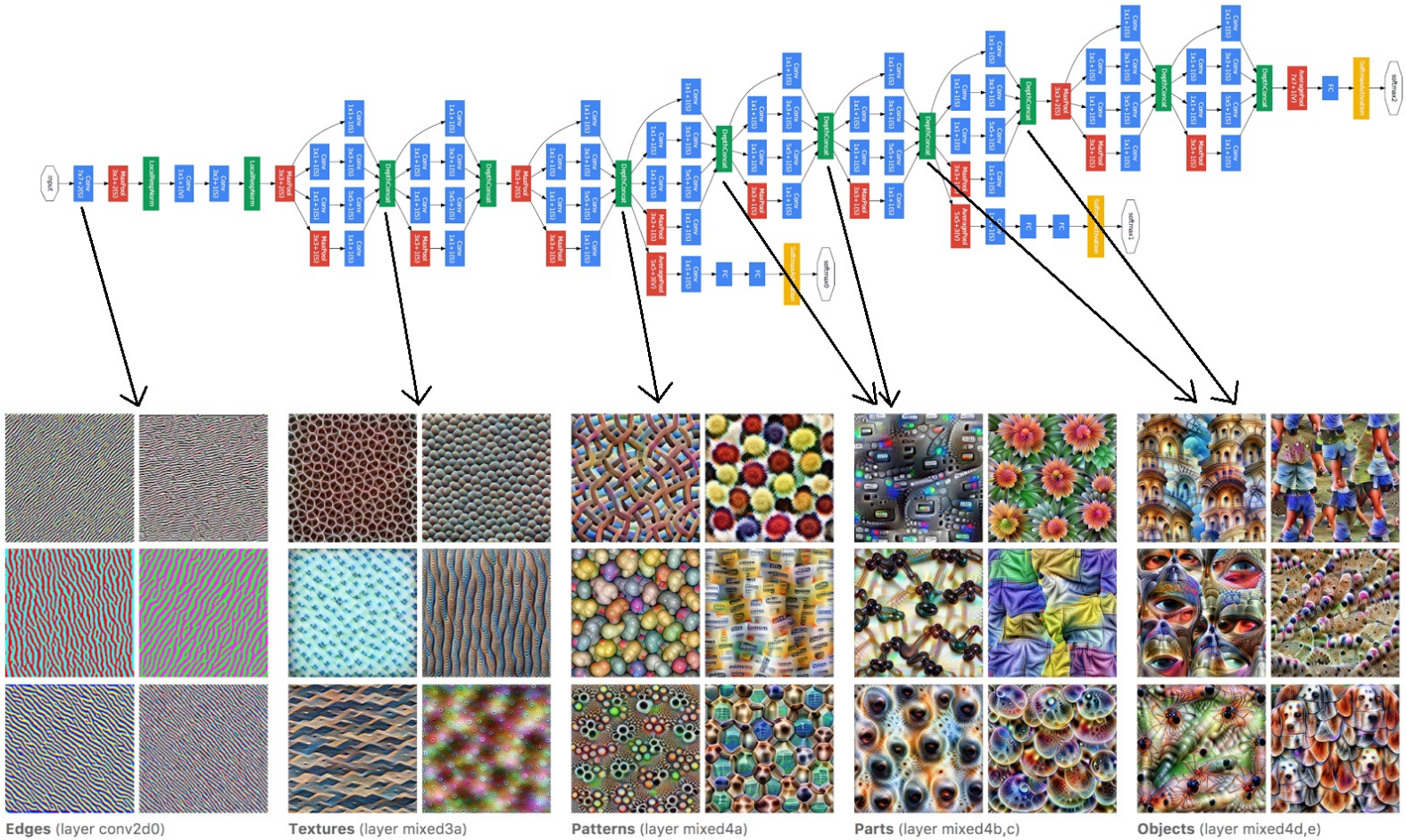


Inception: Model Architecture



Inception: Model Architecture

Inception: Interpretation



Edges (layer conv2d0) **Textures** (layer mixed3a) **Patterns** (layer mixed4a) **Parts** (layer mixed4b,c) **Objects** (layer mixed4d,e)

Subsection 4

Convolutional Models for Natural Language

Embeddings

CNNs: successfully applied in natural language processing (NLP)

Typically: start with a sequence of **symbols** (characters, subword units, words) $\omega \in \Omega$

Map these symbols to vector space: **embedding**

$$\Omega \ni \omega \mapsto \mathbf{x}_\omega \in \mathbb{R}^n$$

$\Longrightarrow$ nothing but a (possibly large) lookup table: entries are learnable parameters

word2vec

Word2vec: Embedding-based model of word representations

Input and output embedding:

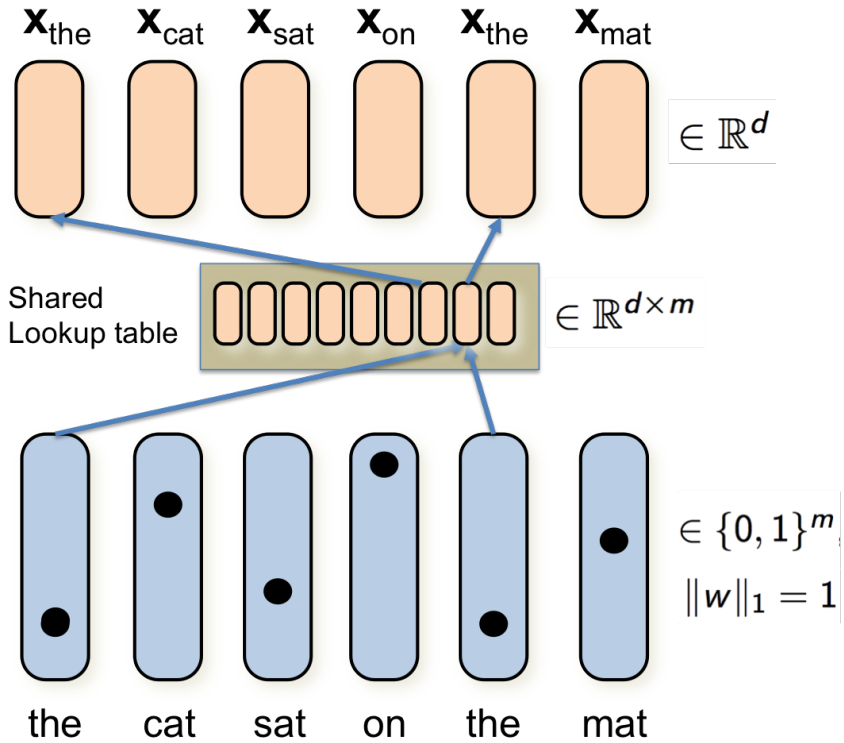
$$\Omega_{\text{in}} \ni \omega \mapsto \mathbf{x}_\omega \in \mathbb{R}^d \qquad \Omega_{\text{out}} \ni \nu \mapsto \mathbf{y}_\nu \in \mathbb{R}^d$$

Conditional **log-bilinear** model

$$\mathbb{P}(\nu \,|\, \omega) = \frac{\exp[\mathbf{x}_\omega \cdot \mathbf{y}_\nu]}{\sum_\mu \exp[\mathbf{x}_\omega \cdot \mathbf{y}_\mu]}$$

$\Longrightarrow$ predicts word ν in the neighborhood of word ω.

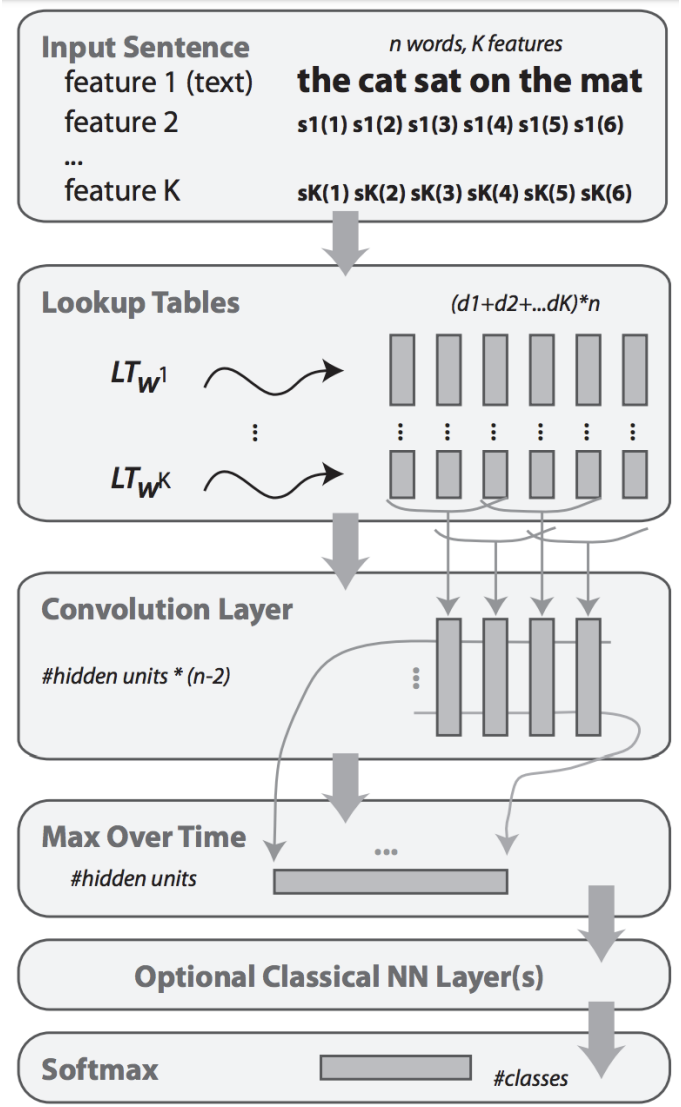
Variable Length Token Embeddings



$\Longrightarrow$ conversion of one-hot vectors (symbol index) into embeddings via a shared lookup table

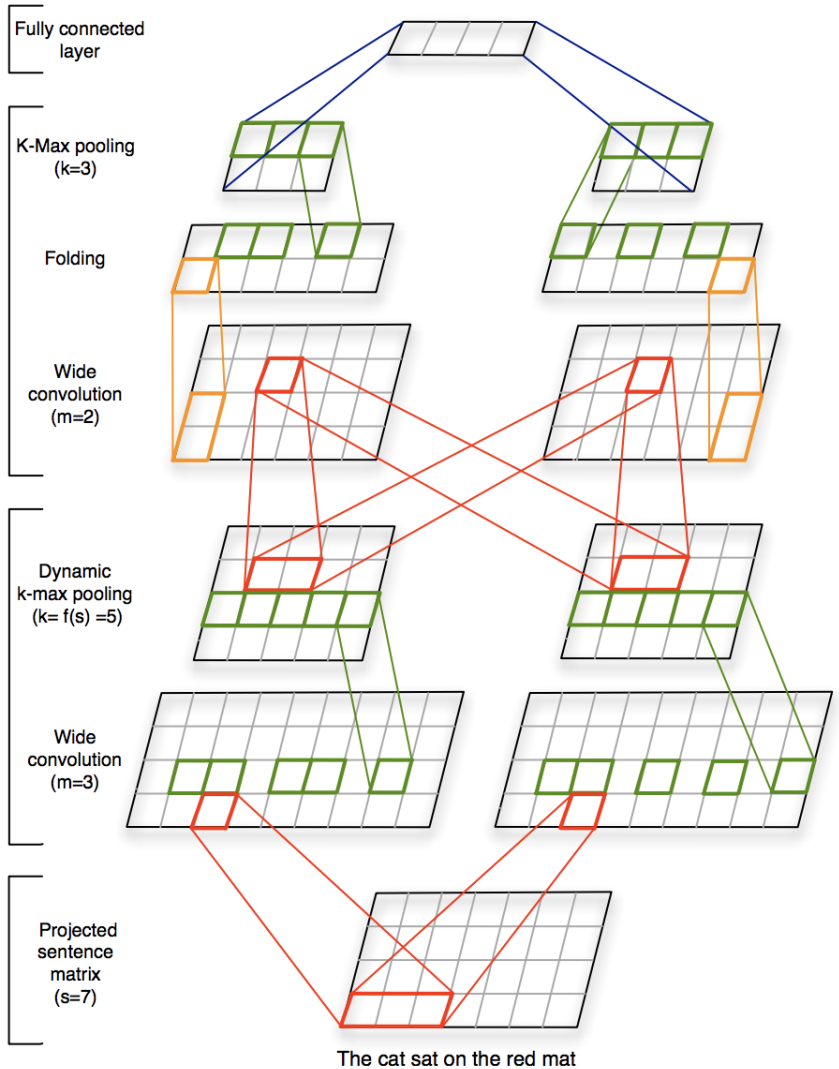
CNN Architectures for NLP

Classical model due to Collobert et al. 2008



CNN Architectures for NLP

Single-channel cross-pooling CNNs of Kalchbrenner et al. 2014



CNN Architectures: Discussion

Pros: Easy to design and diagnose. High degree of parallelism.

Cons: Limited dependency range. Fixed-length requires pooling.

Recently: Combinations of CNN with attention and memory units

...

Alternatives: recurrent neural networks, transformers

Section 2

Connectivity

Subsection 1

Short Connectivity

Residual Layers

Assume that we have a layer transfer function

$$g : \mathbb{R}^m \to \mathbb{R}^m.$$

We can parameterize it as

$$g(\mathbf{x}) = \mathbf{x} + f(\mathbf{x}; \boldsymbol{\theta}).$$

Identity function as the default; network can additively modify.

Initialize $f \approx 0$, Jacobian $\mathbf{J}_g = \mathbf{I} + \mathbf{J}_f \approx \mathbf{I}$.

Ideal for gradient propagation!

Linear Residuals

If the dimensions do not match, can use a linear network as default.

$$g(\mathbf{x}) = \mathbf{W}\mathbf{x} + f(\mathbf{x}; \boldsymbol{\theta}).$$

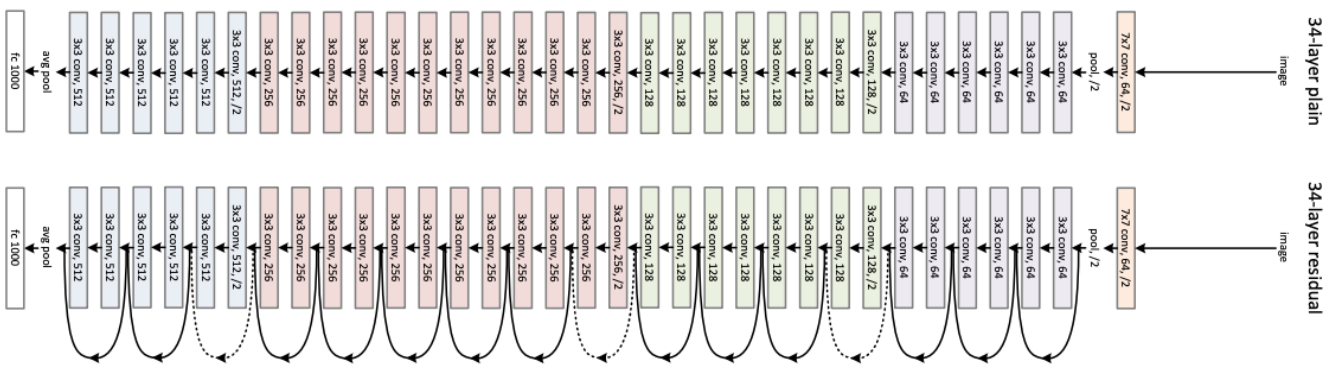
Initialize $f \approx 0$, Jacobian $\mathbf{J}_g = \mathbf{W} + \mathbf{J}_f \approx \mathbf{W}$.

Again: easy to initialize and well-behaved gradients.

Networks with Residual Layers

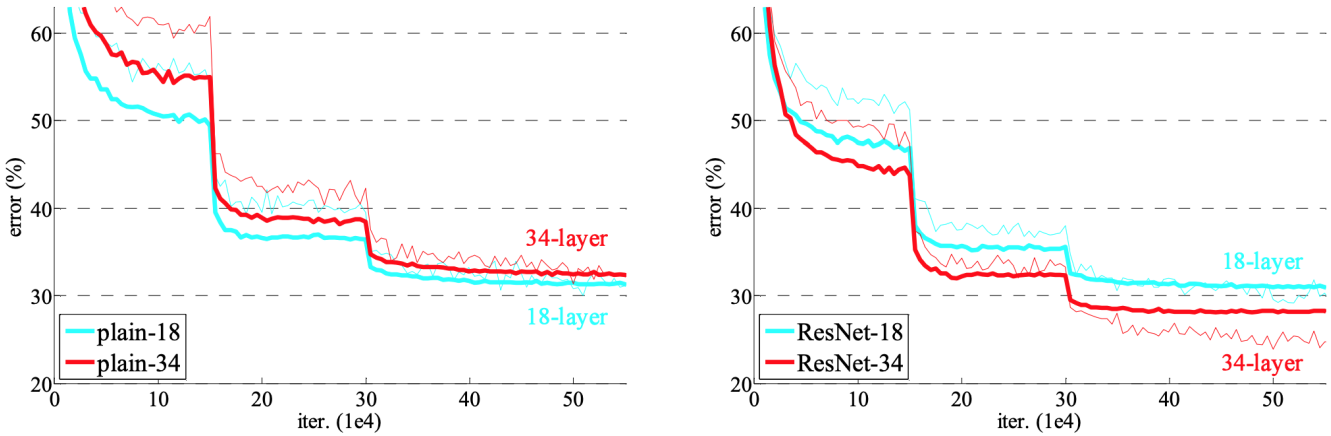
Use any standard DNN and add connectivity (branch and add)

Here: deep convolutional (VGG-style) network, augmented by residual connections (cf. arrows in diagram)



ResNets: Imagenet Results

Experimental comparison on ImageNet. Benefits of depth?



w/o residual connections: deeper networks is poorer

w/ residual connections: **advantage of depth**

ResNets

Basic building block of residual network (ResNets, He et al. 2016)

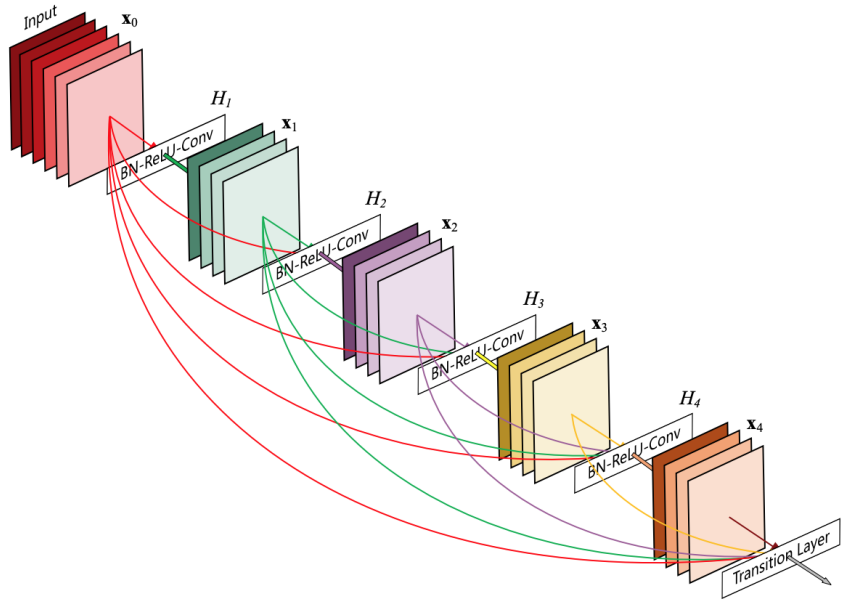
Depth of $100+$ layers trainable and beneficial.

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC'14)	-	8.43†
GoogLeNet [43] (ILSVRC'14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Dense Connectivity

Dense blocks (Huan et al. 2017):

Connect layer output to *all* downstream layers



Concatenate activations, instead of adding (residual layers).

Section 3

Normalization

Normalization Techniques

Observation: poorly calibrated dynamic range of activations

$\Longrightarrow$ poor backpropagation of errors (**vanishing gradients**)

How to measure dynamical range / variability?

1. Across many data points (e.g. mini batch)
   $\rightsquigarrow$ **batch normalization**
2. Across units in the same layer
   $\rightsquigarrow$ **layer normalization**

Batch Normalization: Motivation

Strong dependencies between weights in layers exist.

$\rightsquigarrow$ Hard to find suitable learning rate.

- toy example: deep network with one unit per layer

$$y = w_1 \cdot \dots \cdot w_L x \Longrightarrow y^{new} = \prod_{l=1}^{L} \left(w_l - \eta \frac{\partial \ell}{\partial w_l} \right) x$$

multiplying out leads to terms of up to order L.

- Gradients are products of $\mathcal{O}(L)$ terms:
  $\rightsquigarrow$ If these gradients are small (less than 1), the product tends to zero as L increases, leading to vanishing gradients
  $\rightsquigarrow$ If these gradients are large (greater than 1), the product grows exponentially, leading to exploding gradients.

Batch Normalization

[Ioffe & Szegedy, 2015]

Key idea: normalization is performed using statistics calculated from the **current mini-batch**

- fix layer l, fix set of examples $I \subseteq [1:s]$
- mean activity, vector of standard deviation

$$\boldsymbol{\mu}^l := \frac{1}{|I|} \sum_{j \in I} \mathbf{z}^l[t], \quad \mathbf{z}^l[j] := (F^l \circ \cdots \circ F^1)(\mathbf{x}[j])$$

$$\sigma_i^l := \sqrt{\delta + \frac{1}{|I|} \sum_{j \in I} \left(z_i^l[j] - \mu_i^l\right)^2}, \quad \delta > 0$$

- $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are functions of the weights: can be differentiated

Batch Normalization

[Ioffe & Szegedy, 2015]

- Normalized activities (cf. z-score in statistics)

$$\tilde{z}_i^l := \frac{z_i^l - \mu_i^l}{\sigma_i^l}, \quad \mathbb{E}[\tilde{z}_i^l] = \mathbf{0}, \quad \mathbb{E}[(\tilde{z}_i^l)^2] = 1$$

- Regain representational power

$$\hat{z}_i^l = \alpha_i^l \tilde{z}_i^l + \beta_i^l$$

  - in principle: can exactly undo the batch normalization
  - however: different learning dynamics, mean activation (and scale) can now be directly controlled

Layer Normalization

Same idea as batch normalization.

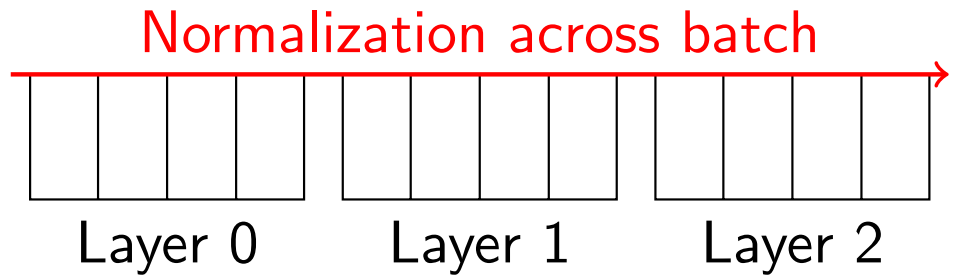
But: normalization is performed using statistics calculated for each input instance **across its features**.

$$\mu^l[t] := \frac{1}{m^l} \sum_{i=1}^{m^l} z_i^l[t]$$

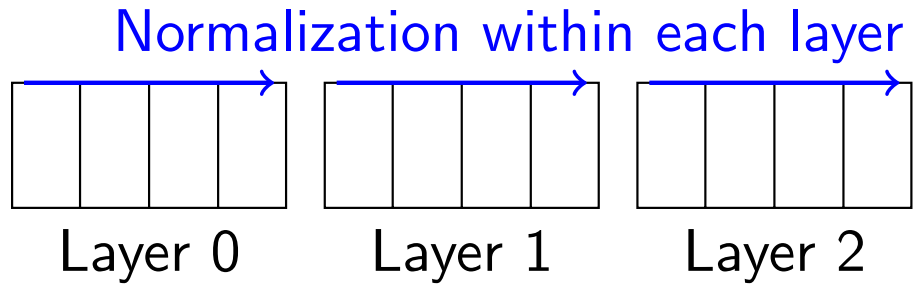
$$\sigma^l[t] := \sqrt{\delta + \frac{1}{m^l} \sum_{i=1}^{m^l} \left(z_i^l[t] - \mu^l[t]\right)^2}, \quad \delta > 0$$

Batch vs. Layer Normalization: illustration

Batch Normalization



Layer Normalization



Batch vs. Layer Normalization

Batch normalization:

(+) very effective, broadly used in vision

(–) dependence on batch size, not suitable for all architectures

Layer normalization:

(+) very effective, used in natural language (transformers, RNNs)

(–) no consistent benefits

Section 4

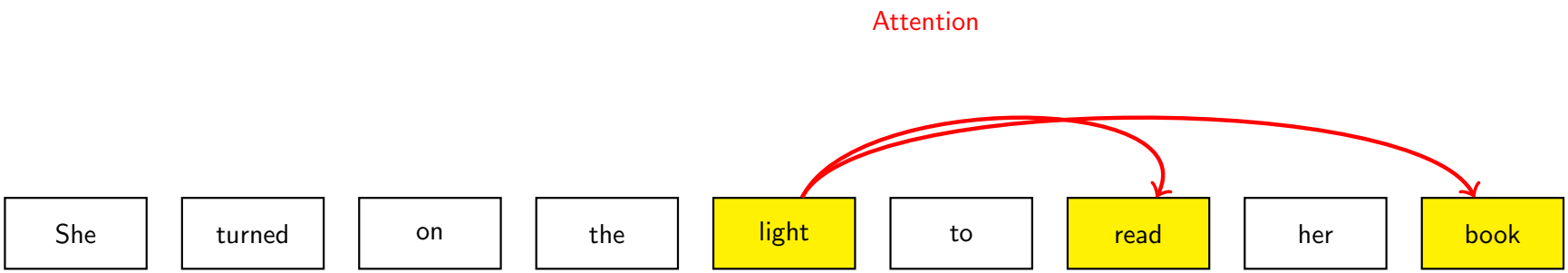
Attention Models

Transformers in Deep Learning

- Transformers are a type of deep learning architecture introduced in the paper "Attention is All You Need" by Vaswani et al. (2017).
- They have become the state-of-the-art approach in many natural language processing (NLP) tasks, such as machine translation and language understanding.

Transformer Model: Attention Mechanism

- Unlike traditional recurrent neural networks (RNNs) or convolutional neural networks (CNNs), Transformers rely solely on **self-attention mechanisms** to process input data.
- Self-attention allows Transformers to capture **dependencies** between different words in a sentence, making them more effective at modeling long-range dependencies compared to RNNs or CNNs.



Transformer Model: Input data

Input data to a transformer is a set of vectors $\{\mathbf{x}_n\}$ of dimensionality d, where $n = 1, \ldots, N$. We refer to these data vectors as **tokens**, where a token might, e.g., correspond to:

- A word within a sentence
- A patch within an image
- An amino acid within a protein

The elements x_{ni} of the tokens are called **features**.

Attention Mechanism

- **Goal:** map input tokens $\mathbf{x}_1, \ldots, \mathbf{x}_N$ to another set $\mathbf{y}_1, \ldots, \mathbf{y}_N$ that captures a richer semantic structure.
- With attention, this dependence should be stronger for those inputs $\mathbf{x}_m$ that are particularly important for determining the modified representation of $\mathbf{y}_n$.

Mathematically:

$$\mathbf{y}_n = \sum_{m=1}^{N} a_{nm}\mathbf{x}_m, \qquad a_{nm} = \frac{\exp(\mathbf{x}_n^\top \mathbf{x}_m)}{\sum_{k=1}^{N} \exp(\mathbf{x}_n^\top \mathbf{x}_k)}$$

where a_{nm} are the attention coefficients.

$$\rightsquigarrow \text{Matrix form: } \mathbf{Y} = \text{Softmax}(\mathbf{X}\mathbf{X}^\top)\mathbf{X}$$

Softmax(A) applies the exponential function to each element of a matrix A and then normalizes each row so that the sum of the elements in each row equals one.

Attention Mechanism: Adjustable Parameters

We redefine the modified feature vectors as the result of applying a linear transformation to the original vectors, expressed as $\mathbf{XU}$, where $\mathbf{U}$ is a $d \times d$ matrix containing learnable weight parameters.

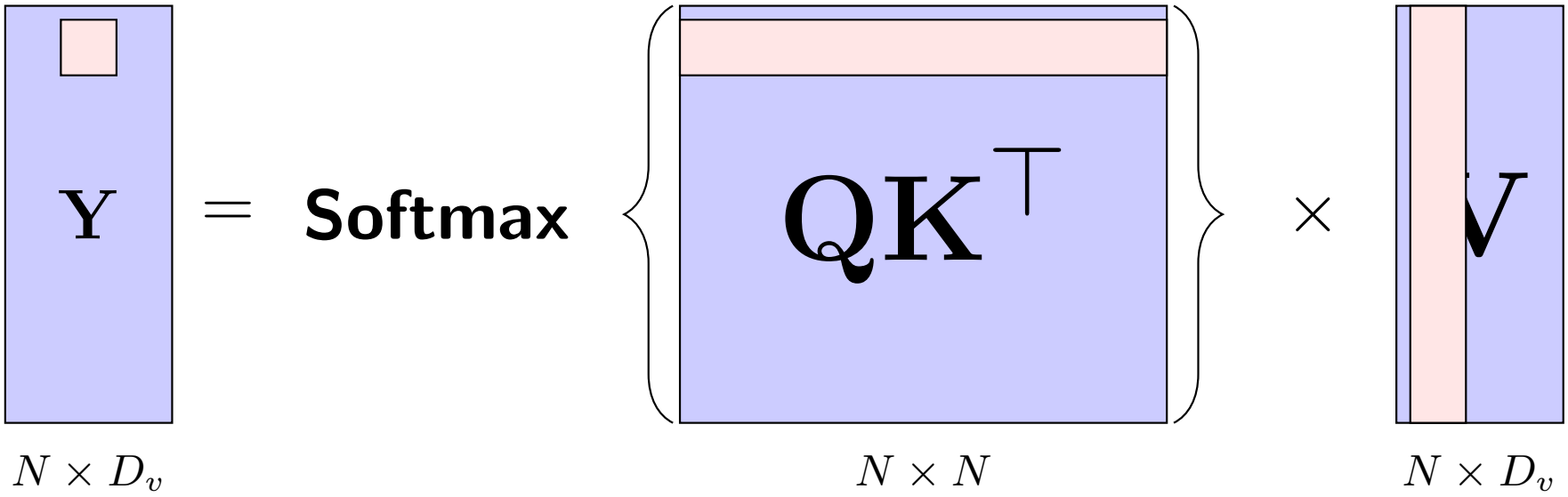
$$\rightsquigarrow \text{Matrix form: } \mathbf{Y} = \text{Softmax}(\mathbf{XUU}^\top\mathbf{X}^\top)\mathbf{XU}$$

Add more flexibility by defining 3 matrices (query, key, value):

$$\mathbf{Q} = \mathbf{XW}^q \quad \mathbf{K} = \mathbf{XW}^k \quad \mathbf{V} = \mathbf{XW}^v. \tag{1}$$

$$\rightsquigarrow \text{Matrix form: } \mathbf{Y} = \text{Softmax}(\mathbf{QK}^\top)\mathbf{V}$$

Attention Mechanism cs CNN



- In contrast to a conventional neural network, where activations are multiplied by fixed weights, the signal paths in this context involve multiplicative interactions between activation values.
- Instead of static weights, the activations here are dynamically multiplied by data-dependent attention coefficients.

Attention Mechanism: Multihead attention

Head: Attention layer described so far enables the output vectors to focus on input-specific patterns and is referred to as an **attention head**

$\rightsquigarrow$ there might be multiple patterns of attention that are relevant at the same time.

Concatenation: The outputs from multiple attention heads can be combined by concatenating them and applying a linear transformation to generate the final output.

Let head_i denote the output of the i-th head. Then, the multi-head attention output is:

$$\text{MultiHead}(\mathbf{X}) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)\mathbf{W}^O$$

Here, $\mathbf{W}^O \in \mathbb{R}^{hd_k \times d_{\text{model}}}$ is the output projection matrix.

Transformer Model

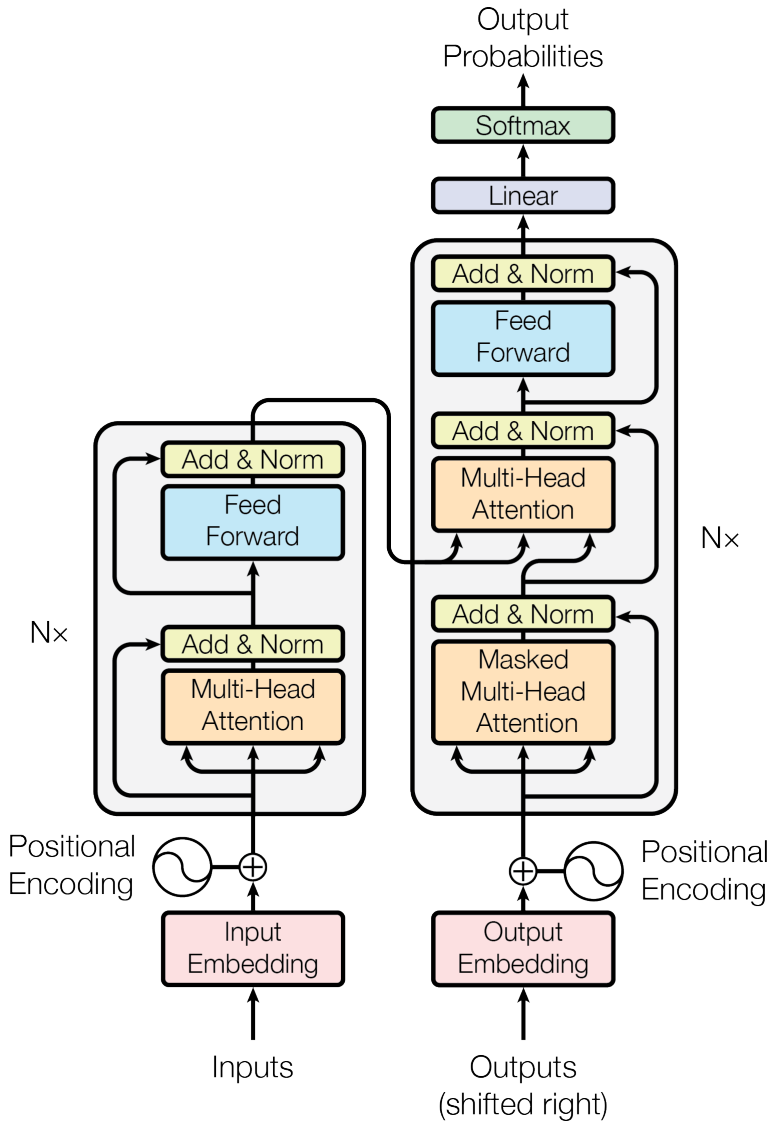


Figure: A single Transformer block. Source: Attention Is All You Need, Vaswani et al.

Transformer Model (simplified)

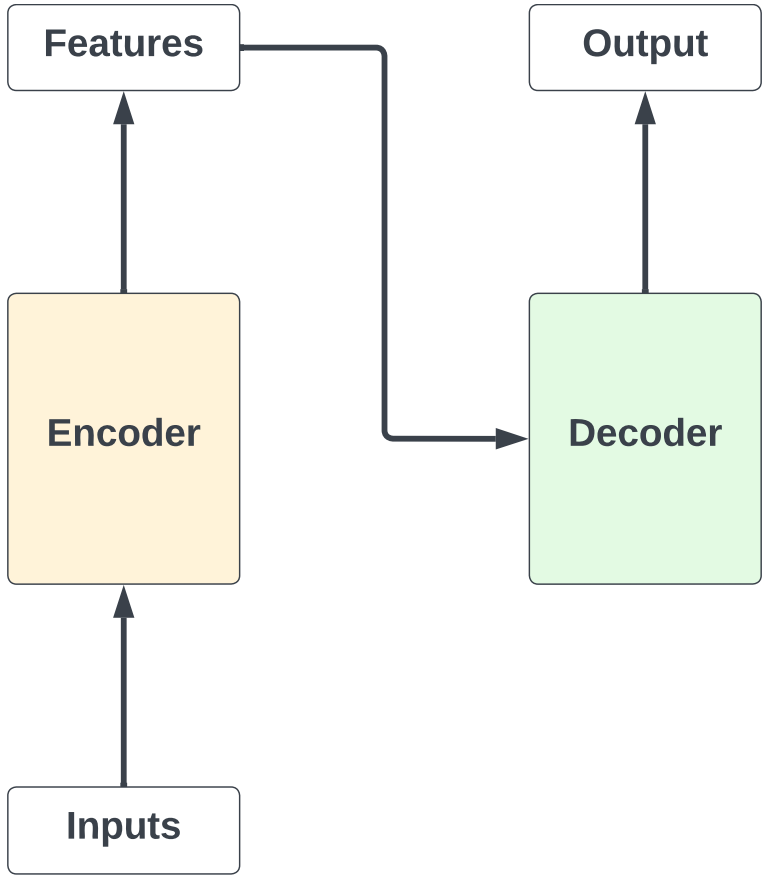
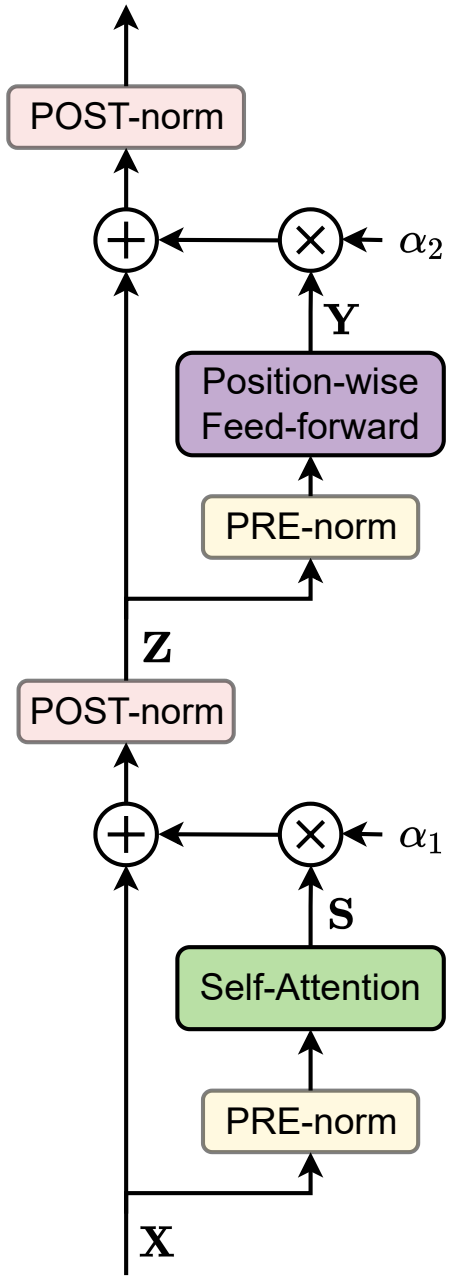


Figure: A single Transformer block: The encoder extracts features from an input sentence, and the decoder uses the features to produce an output sentence (e.g. for a translation task). The encoder is not necessary for all tasks.

Block of an Encoder



Self-attention

Given an input sequence $\mathbf{X} \in \mathbb{R}^{n \times d_v}$, with n tokens of dimension d_v, the (single-head) self-attention is defined as:

$$\mathbf{S}^\ell := \mathbf{A}^\ell \mathbf{X}^\ell \mathbf{W}^V, \text{ where } \mathbf{A}^\ell = \text{softmax}\left(\frac{1}{\sqrt{d_k}} \mathbf{X}^\ell \mathbf{W}^Q \left(\mathbf{X}^\ell \mathbf{W}^K\right)^\top\right),$$

where the softmax function is applied independently across each row, and the superscript ℓ indexes the ℓ-th layer.

- Matrices $\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{d_v \times d_k}$ and $\mathbf{W}^V \in \mathbb{R}^{d_v \times d_v}$ are learnable parameters, and each layer is initialized with an independent set of weights
- Matrices $\mathbf{X}^\ell \mathbf{W}^Q, \mathbf{X}^\ell \mathbf{W}^K, \mathbf{X}^\ell \mathbf{W}^V$ are referred to as queries, keys and values, respectively.

Note on nomenclature

The names queries, keys and values come from the field of information retrieval systems:

- The query is the question or request you're asking.
- The keys are indexed labels or features that allow the system to understand where the relevant information is.
- The values are the actual content or information retrieved once a match is made.

Block of a Encoder (2)

The complete encoder block can be written recursively as:

$$\begin{aligned}\mathbf{Z}^{\ell} &= \alpha_1 \mathbf{S}^{\ell} + \mathbf{X}^{\ell} \\ \mathbf{Y}^{\ell} &= \sigma(\mathbf{Z}^{\ell}\mathbf{W}^{F_1})\mathbf{W}^{F_2} \\ \mathbf{X}^{\ell+1} &= \alpha_2 \mathbf{Y}^{\ell} + \mathbf{Z}^{\ell},\end{aligned}$$

where the introduced α_1, α_2 parameters indicate the strength of the residual block, $\mathbf{W}^{F_1}$, $\mathbf{W}^{F_2} \in \mathbb{R}^{d_v \times d_v}$ are matrices of learnable parameters; we set $\mathbf{X}^0 := \mathbf{X}$, and $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function.

Block of a Decoder

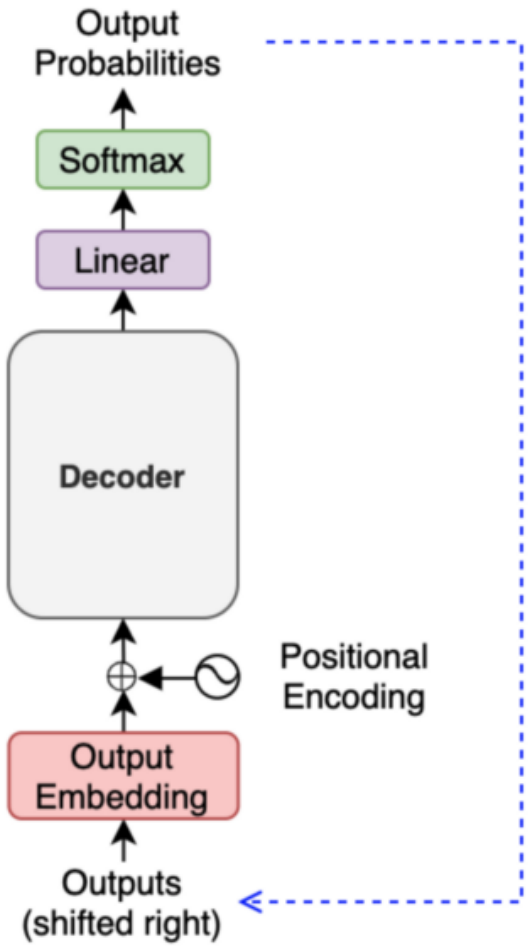


Figure: The decoder generates tokens sequentially, where each token it produces becomes the input for generating the next token. Put differently, the previous decoder output serves as the final segment of the subsequent decoder input. Source: `https://kikaben.com/transformers-encoder-decoder/`

Applications of Transformer Architectures

- Transformers have found success in a wide range of applications beyond NLP, including:
  - Machine Translation
  - Image Generation (e.g., DALL-E)
  - Speech Recognition
  - Protein Folding
- They excel in tasks requiring sequence-to-sequence modeling.

Applications of Transformer Architectures

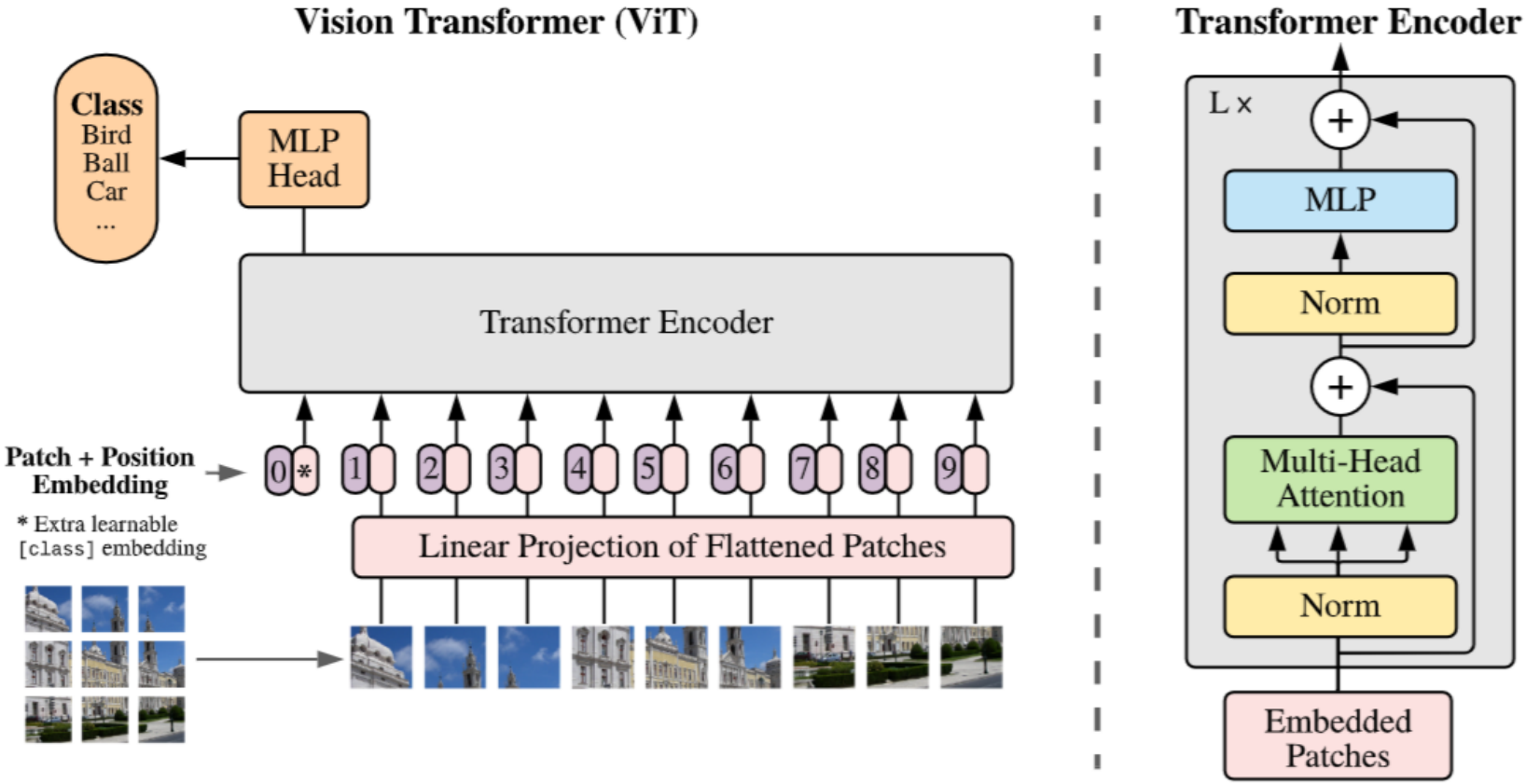


Figure: Vision Transformers (Dosovitskiy et al. 2021)

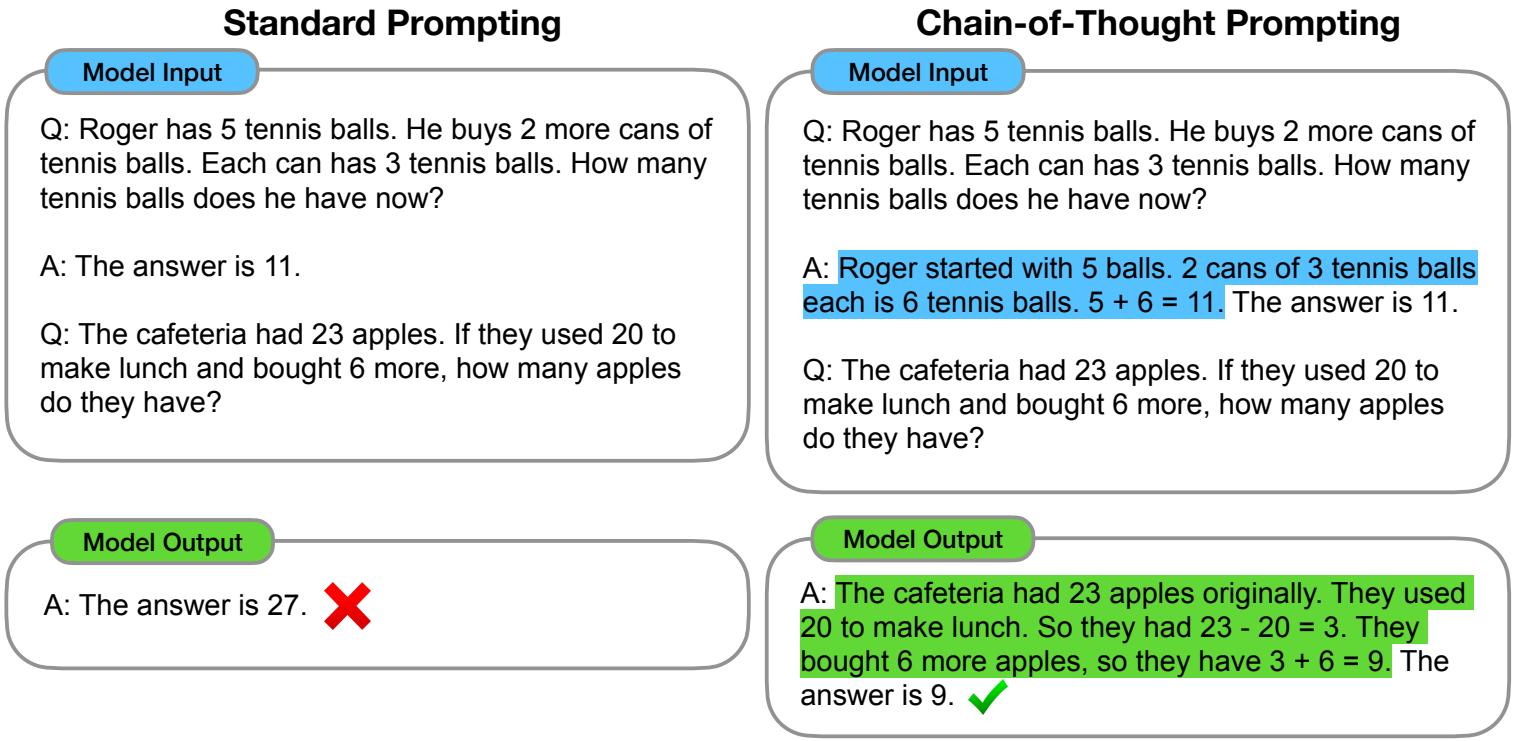
Shortcomings of Transformer Architectures

When it comes to **reasoning tasks**, such as logical reasoning, mathematical problem solving, commonsense reasoning, and structured thinking, Transformers still face some significant challenges, such as:

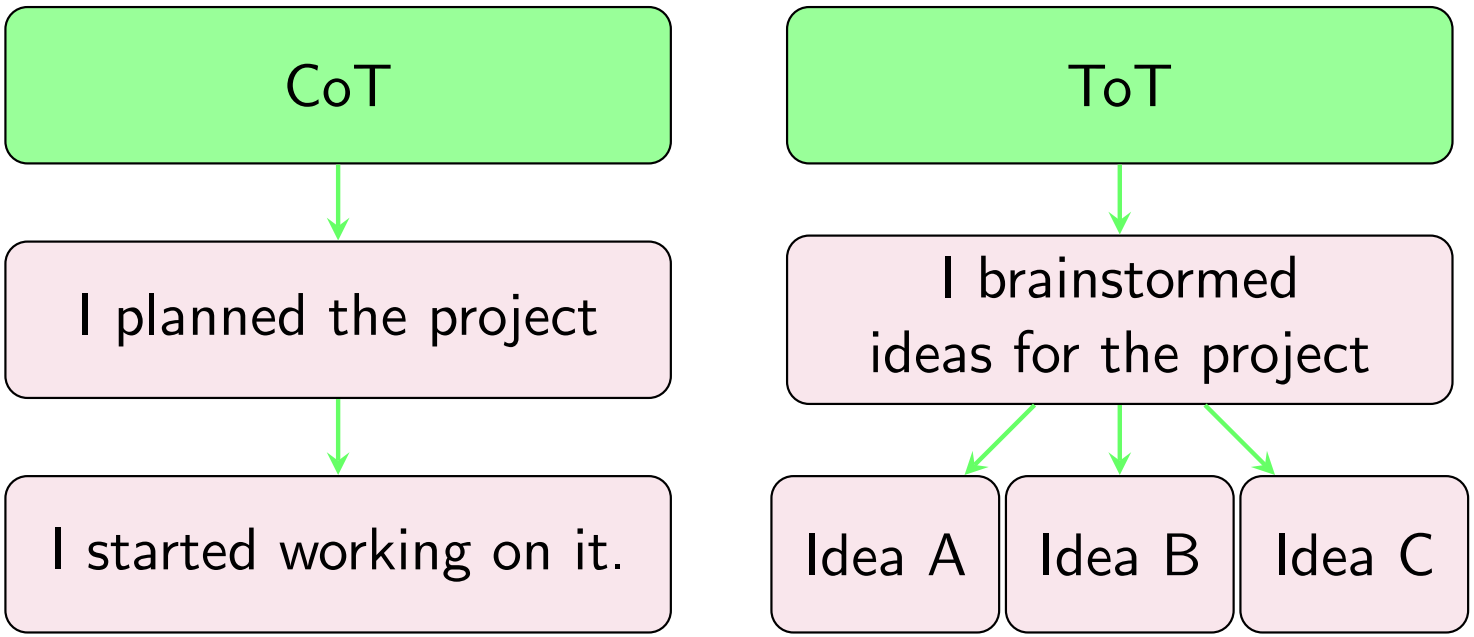
- Poor handling of multi-step reasoning
- Difficulty with symbolic and mathematical reasoning
- Lack of explicit reasoning mechanisms
- Challenges in commonsense and world knowledge reasoning
- Difficulty with logical consistency
- Inability to perform causal and counterfactual reasoning

Chain-of-thought Wei et al. (2022)

Introduced in Wei et al. (2022), chain-of-thought (CoT) prompting enables complex reasoning capabilities by generating intermediate reasoning steps.



Tree-of-thought Yao et al. (2023)



Tree-of-thought Yao et al. (2023)

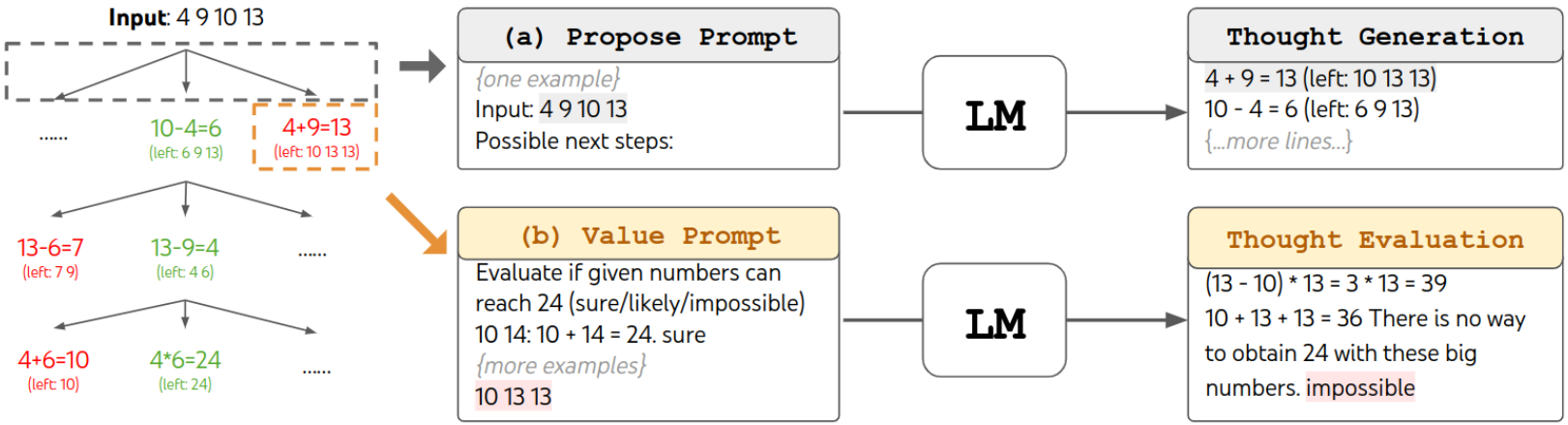


Figure: ToT in a game of 24. The LM is prompted for (a) thought generation and (b) valuation.

Try it yourself: `https://www.4nums.com/`.

Acknowledgment

The part of this lecture about Transformers was made following the excellent book "Deep Learning, Foundations and Concepts" by Christopher M. Bishop and Hugh Bishop.