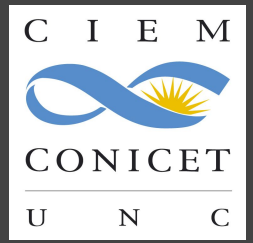
34º Escuela de Ciencias Informáticas
Departamento de Computación, FCEyN, UBA

Aprendizaje con datos escasos

Jorge Sánchez
CIEM-CONICET, UNC
jorge.sanchez@unc.edu.ar



Plan del curso

1. Introducción. Problemas. Representaciones en visión y lenguaje
2. Aprendizaje por transferencia y aumento de datos
3. Aprendizaje basado en prototipos
4. Aprendizaje de métricas
5. Aprendizaje sin ejemplos, auto supervisión, etc

DISCLAIMER

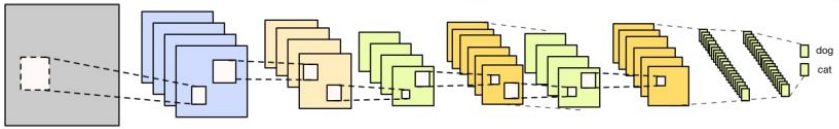
- Slides credit is due to their original authors
- Mostly computer vision problems
- **Small data ≠ small compute**

The deep learning revolution

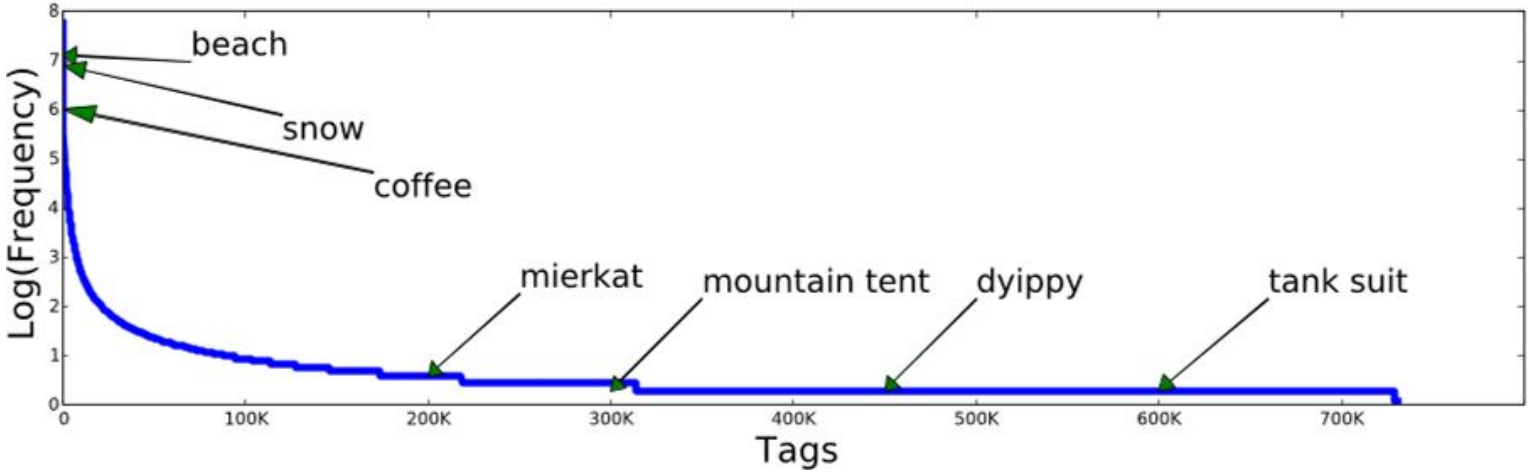
annotation effort

energy

$$\lim_{n,p,q\to\infty} f\left(\texttt{data}(n), \texttt{complexity}(\texttt{p}), \texttt{compute}(q)\right)$$



Long tail: *user tags*



- Generic concepts with many samples
- Specific or fine-grained concepts with few samples
- #unigrams > #bigrams

Kordumova et al. (2016) Exploring the Long-Tail of Social Media Tags

Long tail: *datasets*

Visual Gnome
(>100K images)

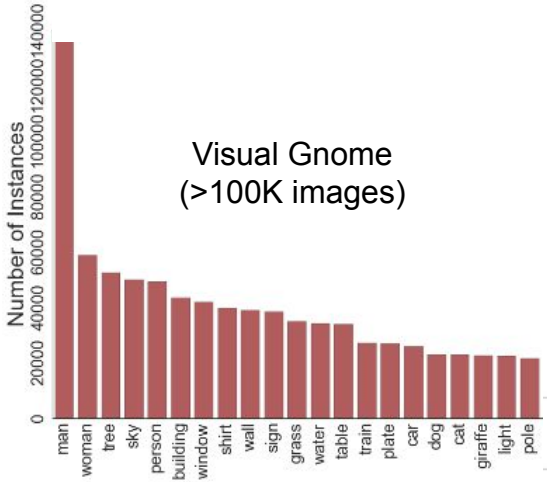
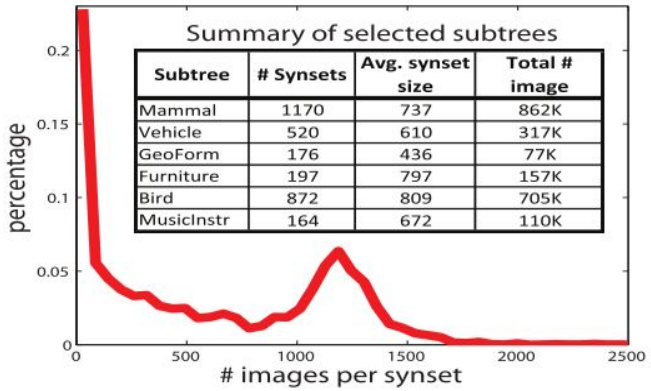


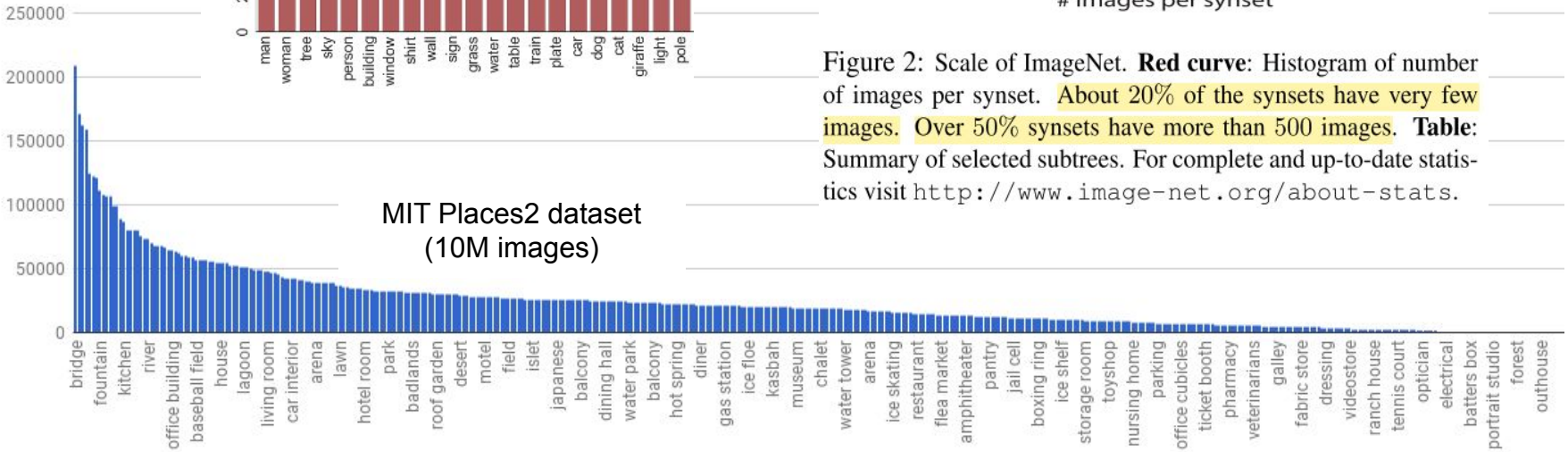
Image-Net (>14M images)



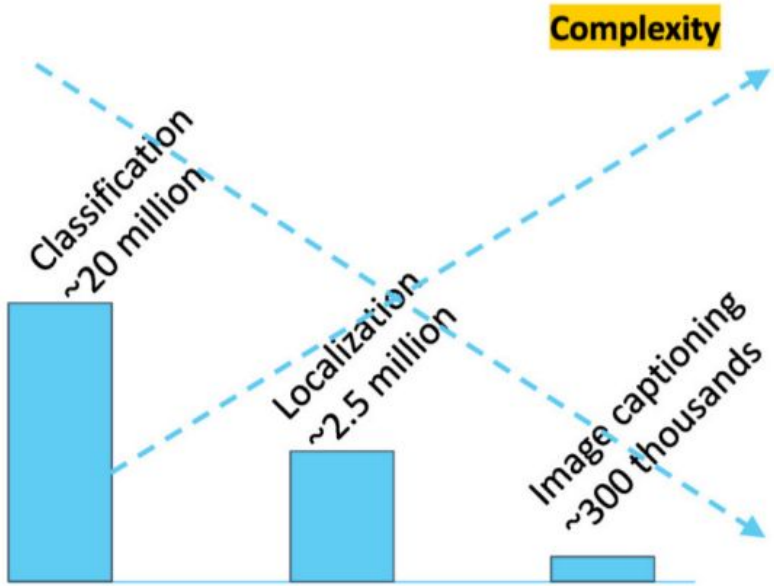
Subtree	# Synsets	Avg. synset size	Total # image
Mammal	1170	737	862K
Vehicle	520	610	317K
GeoForm	176	436	77K
Furniture	197	797	157K
Bird	872	809	705K
MusicInstr	164	672	110K

Figure 2: Scale of ImageNet. **Red curve**: Histogram of number of images per synset. About 20% of the synsets have very few images. Over 50% synsets have more than 500 images. **Table**: Summary of selected subtrees. For complete and up-to-date statistics visit `http://www.image-net.org/about-stats`.

MIT Places2 dataset
(10M images)



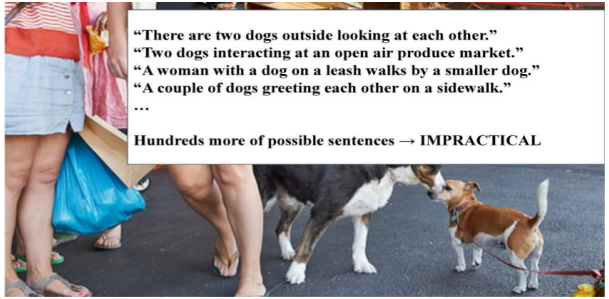
Task complexity vs. annotation effort



classification

segmentation

captioning



cred.: Thomas Mensink

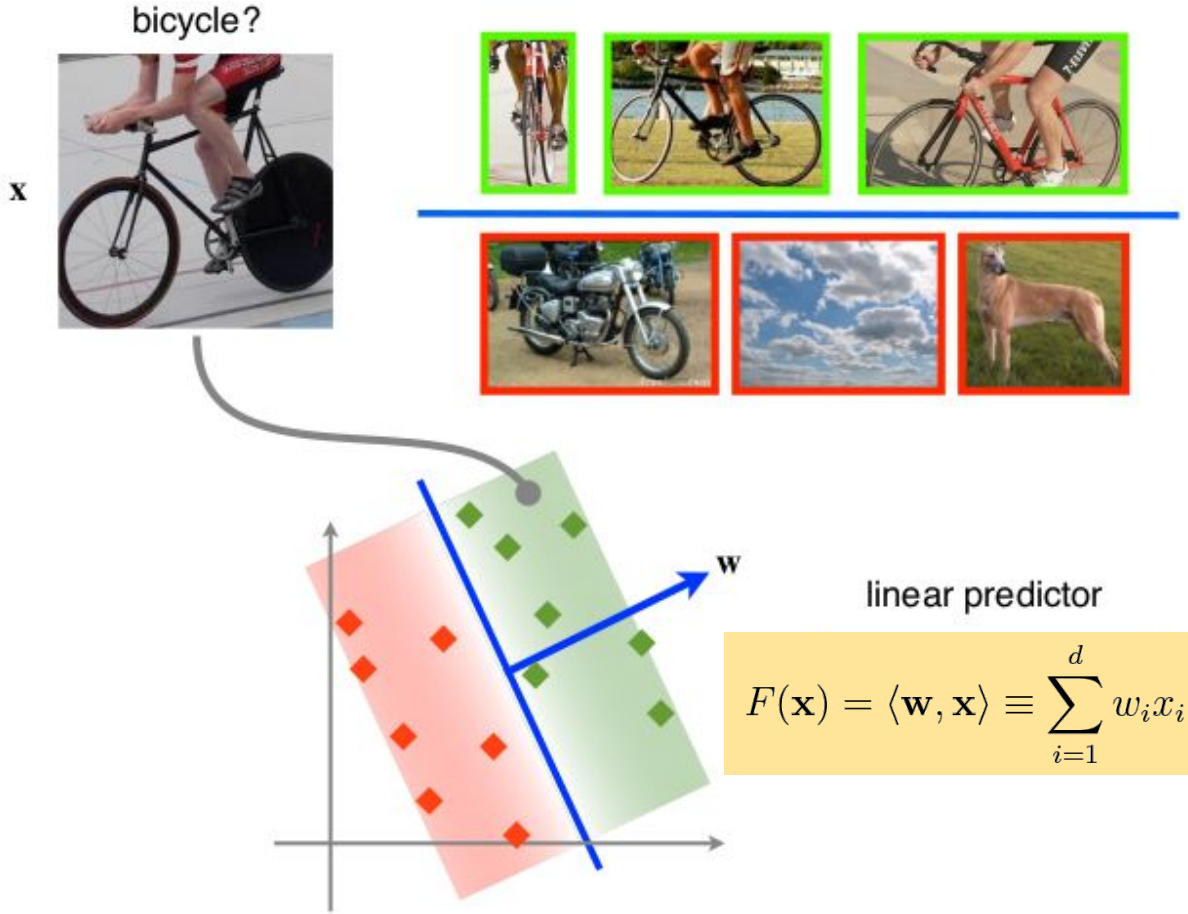
Representations in ML

We would like to build a **predictor** that can tell if an image $\mathbf{x}$ contains a certain object (say a "bicycle").

We **learn** this function from example images that do and do not contain the object.

In the simplest case, the function is a **linear predictor** $F(\mathbf{x})$:

- Images are interpreted as (high-dimensional) vectors.
- $F(\mathbf{x})$ dots $\mathbf{x}$ and a **parameter vector** $\mathbf{w}$ to obtain the **score** for the positive hypothesis (bicycle).
- The **sign** of $F(\mathbf{x})$ is used as prediction.



linear predictor

$$F(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle \equiv \sum_{i=1}^{d} w_i x_i$$

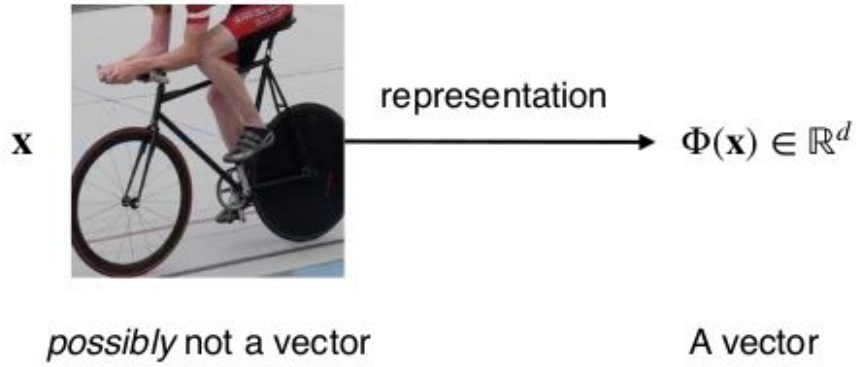
slide: Andrea Vedaldi

Beyond vector data

A linear predictor applies to **vector data**.

However, we want to process images, text, videos, or sounds that are not necessarily vectors.

For this, we use a **representation function Φ**, which maps data to vectors.



Non-linear classification

Representations are used even if the data $\mathbf{x}$ is already a vector.

They result in a non-linear classifier function which can be significantly **more expressive** than a linear one.

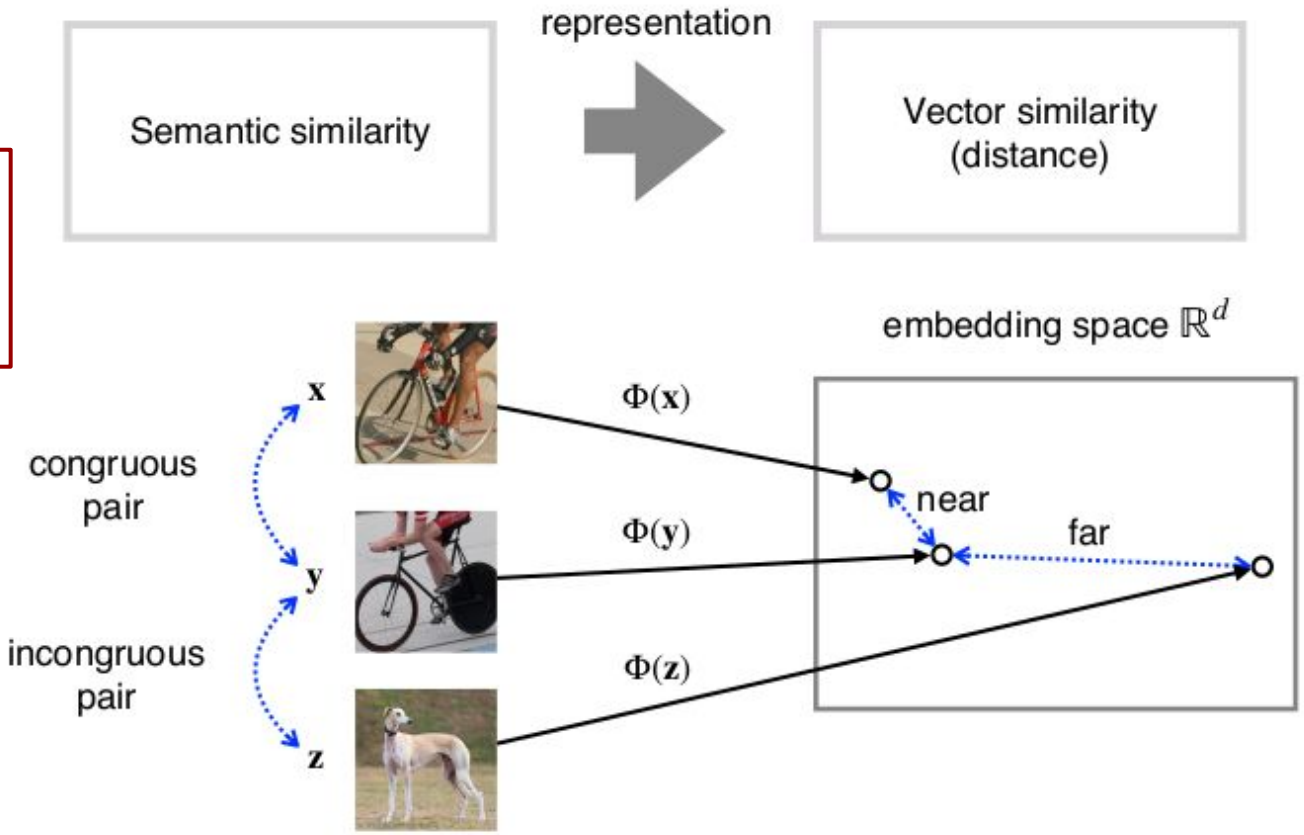
slide: Andrea Vedaldi

A representation should help the linear classifier to perform discrimination.

The goal is to map the **semantic similarity** between data points to a corresponding **vector similarity**.

A good representation is:

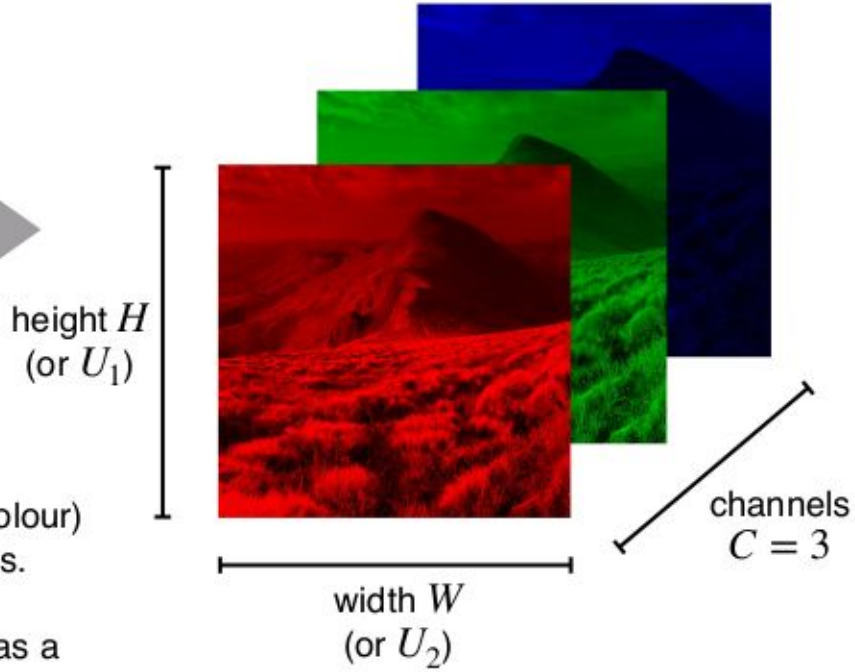
- **invariant** to nuisance factors
- **sensitive** to semantic factors



slide: Andrea Vedaldi

Representations for images

What is an image?



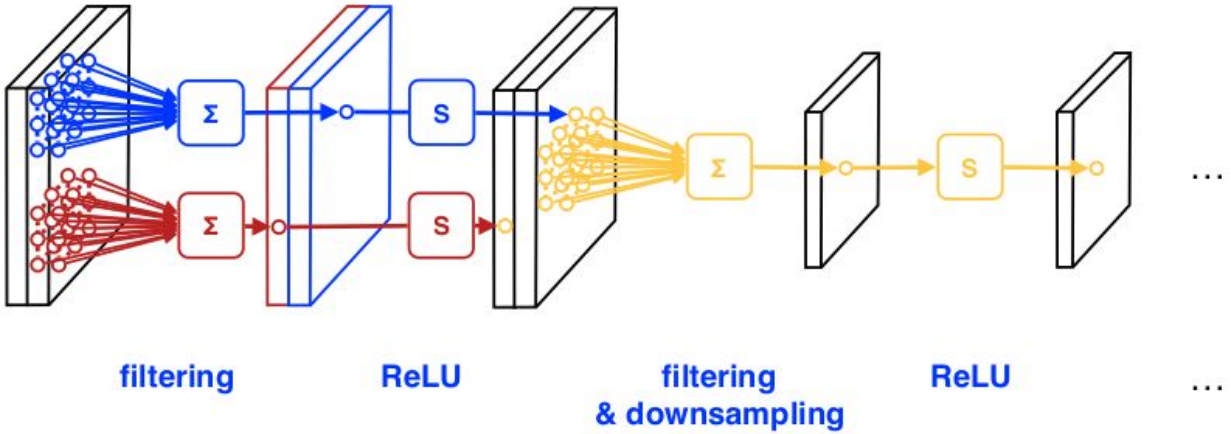
A **color image** can be interpreted as a tensor with $C = 3$ (colour) channels, one for each of the R, G, and B colour components.

More in general, any $C \times H \times W$ tensor can be interpreted as a $H \times W$ **field** of C-dimensional **feature vectors**.

The meaning of the feature channels is often not obvious.

slide: Andrea Vedaldi

Convolutional Neural Networks (CNNs)



- A chain of linear and non-linear operations
- Deep Net = a chain with **many** layers

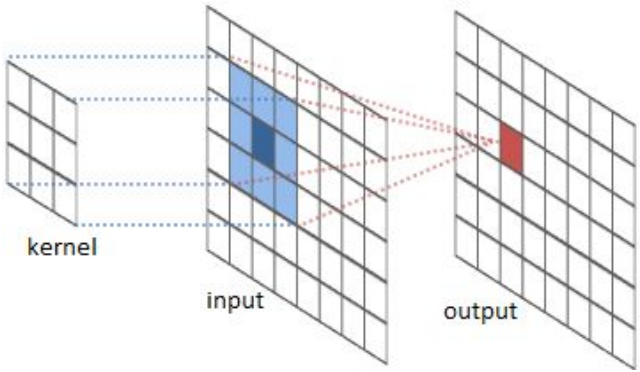
slide: Andrea Vedaldi

Filtering

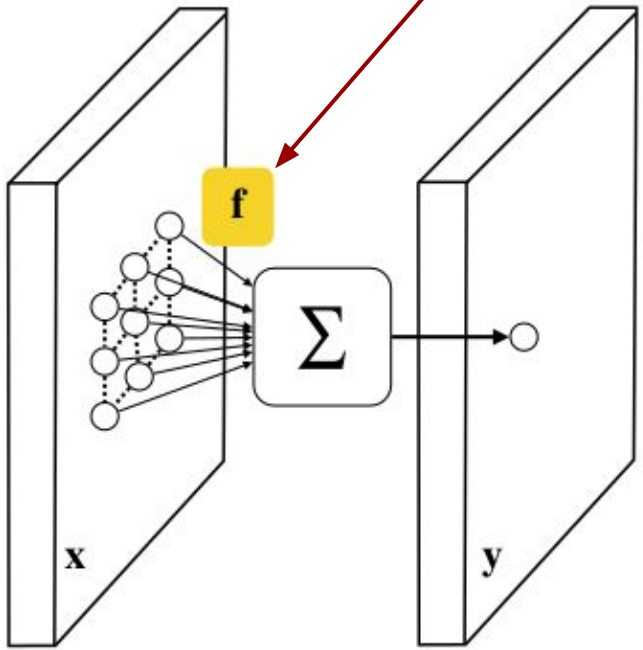
A linear filter $\mathbf{f}$ computes the weighted summation of a window of the input tensor $\mathbf{x}$.

Key properties:

- **Linearity**: the operation is linear in the input and the filter parameters.
- **Locality**: the operator looks at a small window of data.
- **Translation invariance**: all windows are processed using the same filter weights.



The actual value of the coefficients in $\mathbf{f}$ determines the filter response



cred.: Andrea Vedaldi

Filtering

Multiple input and output channels

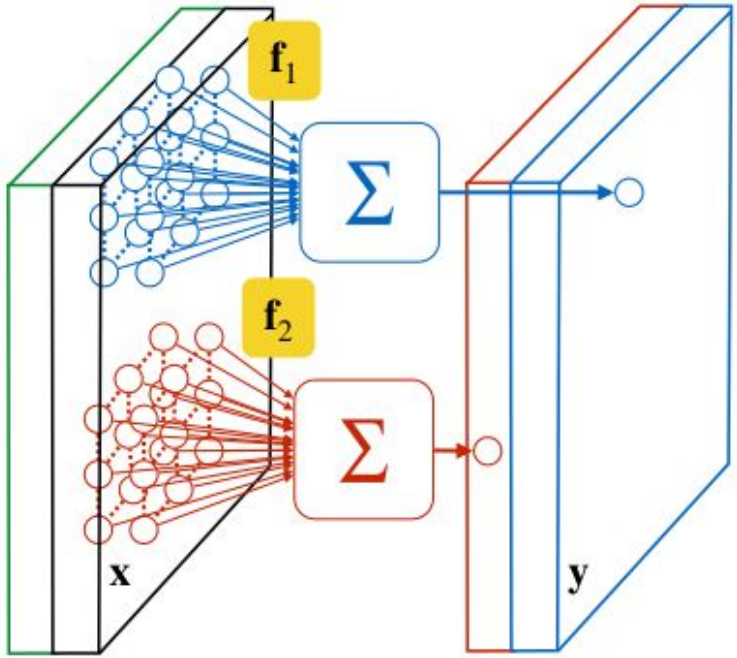
A linear filter $\mathbf{f}$ computes the weighted summation of a window of the input tensor $\mathbf{x}$.

Key properties:

- **Linearity**: the operation is linear in the input and the filter parameters.
- **Locality**: the operator looks at a small window of data.
- **Translation invariance**: all windows are processed using the same filter weights.

The filter has one channel for each input tensor channel.

A **bank of filters** is used to generated multiple output channels, one per filter.

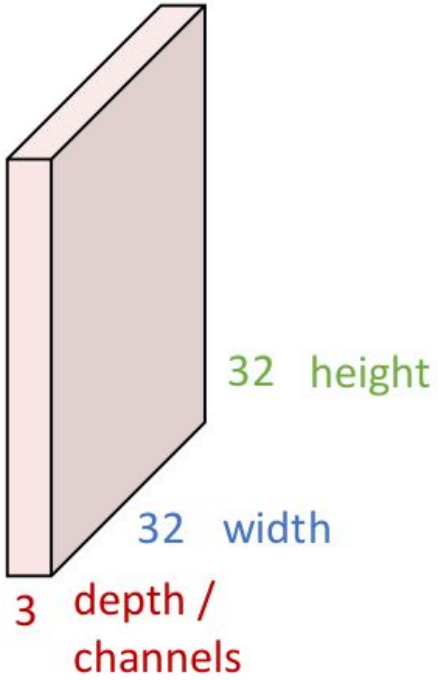


slide: Andrea Vedaldi

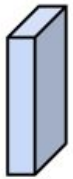
Convolution Layer

Filters always extend the full depth of the input volume

3x32x32 image



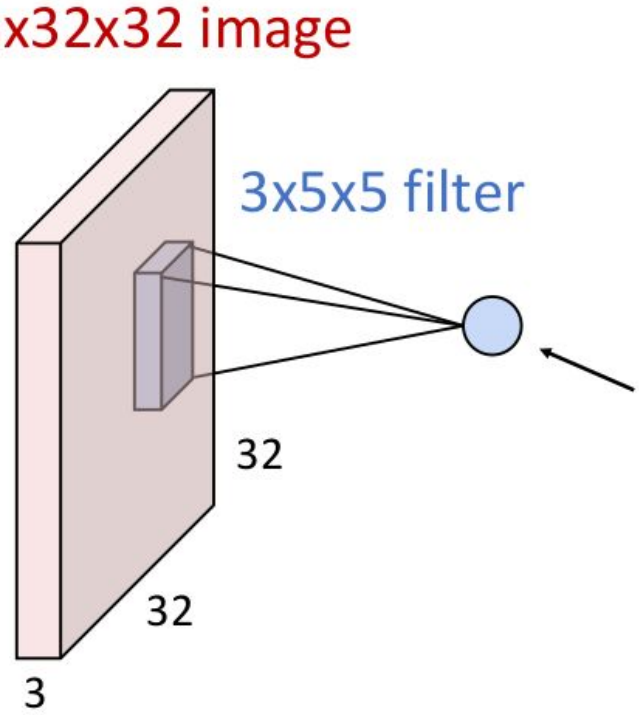
3x5x5 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

slide: Justin Johnson

Convolution Layer

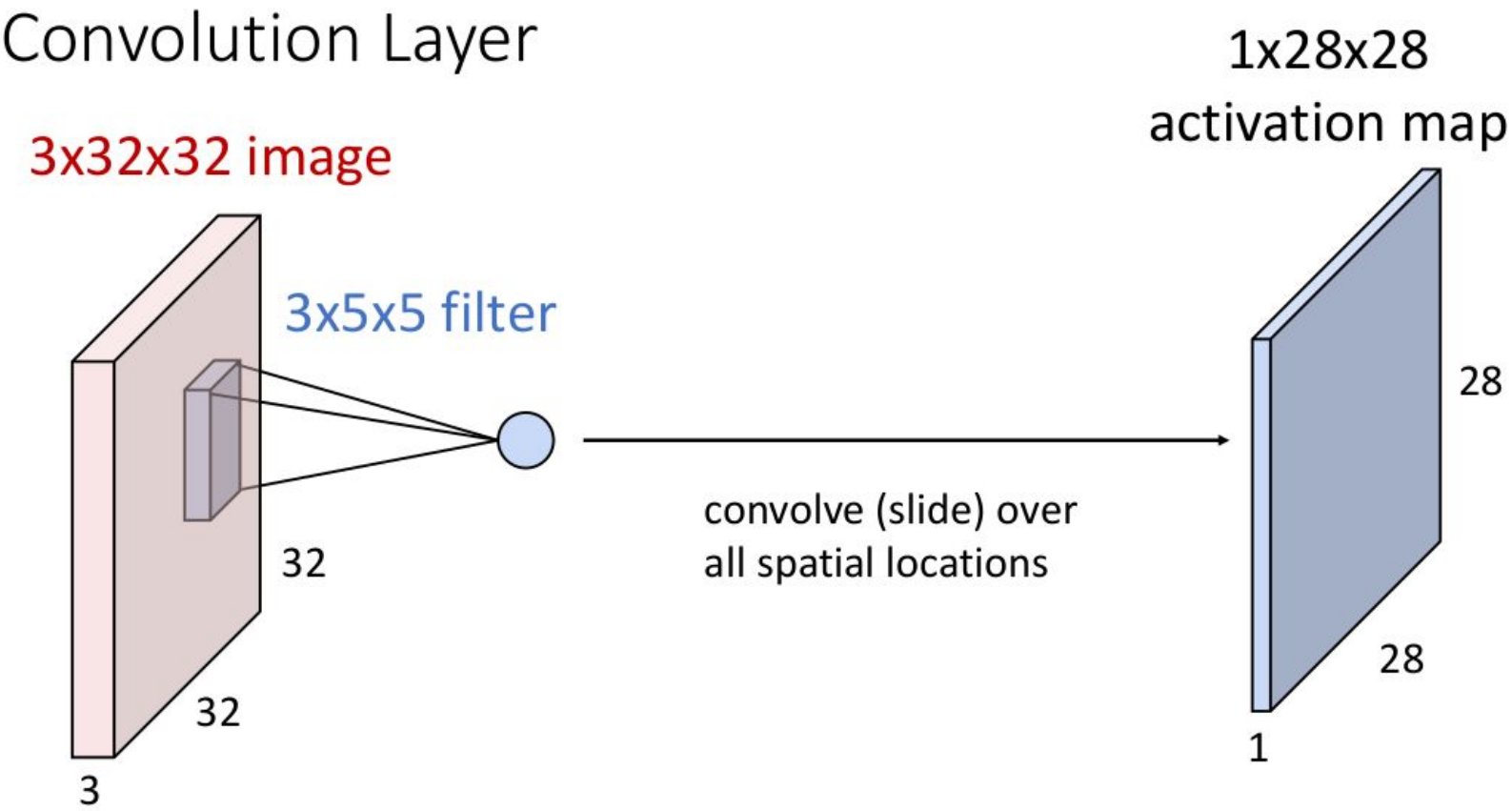


1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

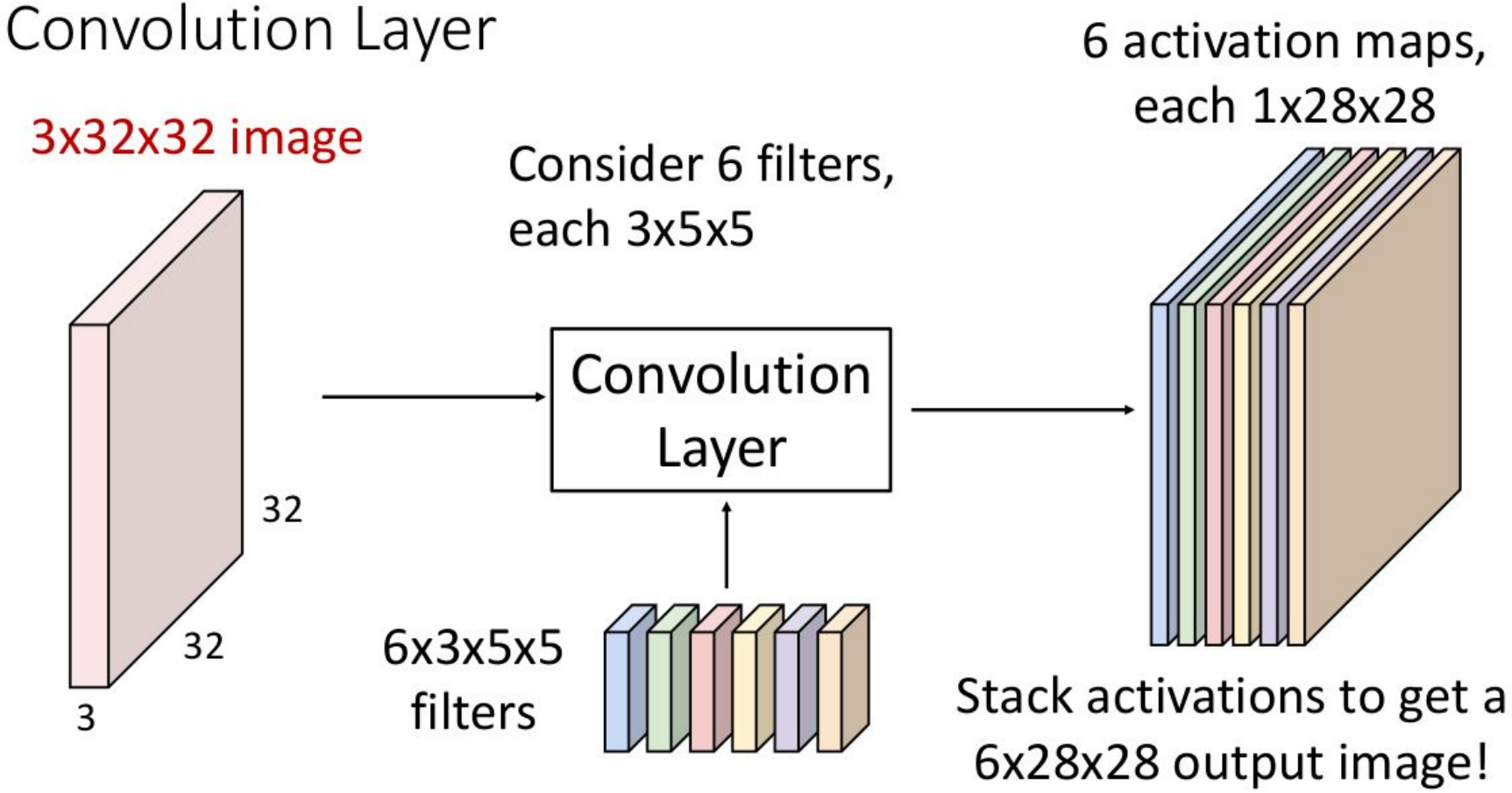
slide: Justin Johnson

Convolution Layer



slide: Justin Johnson

Convolution Layer

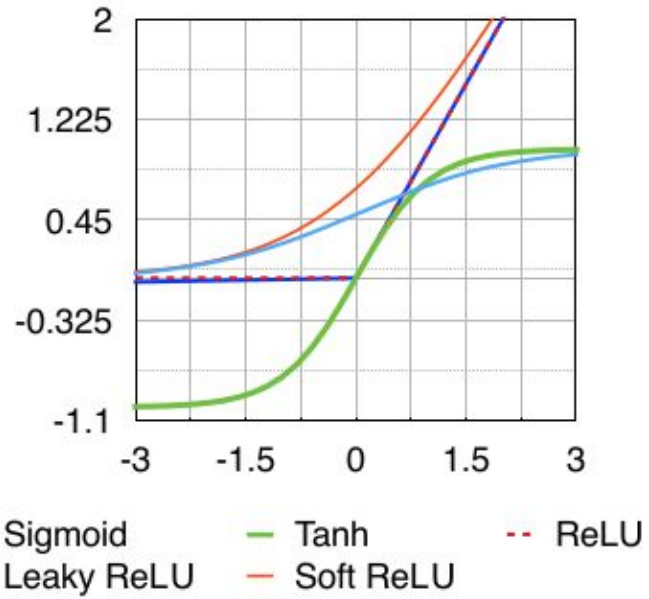


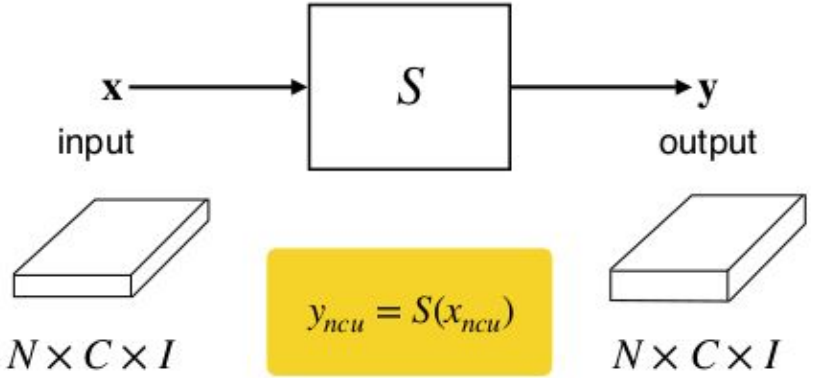
slide: Justin Johnson

Non-linearities

Activation functions

Activation functions are scalar non-linear functions S(z) that are applied element-wise to an input tensor **x** to generate an output tensor **y** (with the same dimensions).





$$\begin{cases} z = \max\{0, z\}, & \text{rectified linear unit (ReLU),} \\ z = \log(1 + e^z), & \text{soft ReLU,} \\ z = \epsilon z + (1 - \epsilon)\max\{0, z\}, & \text{leaky ReLU,} \\ z = (1 + e^{-z})^{-1}, & \text{sigmoid,} \\ z = \tanh(z), & \text{hyperbolic tangent,} \end{cases}$$

slide: Andrea Vedaldi

Non-linearities

Pooling operations

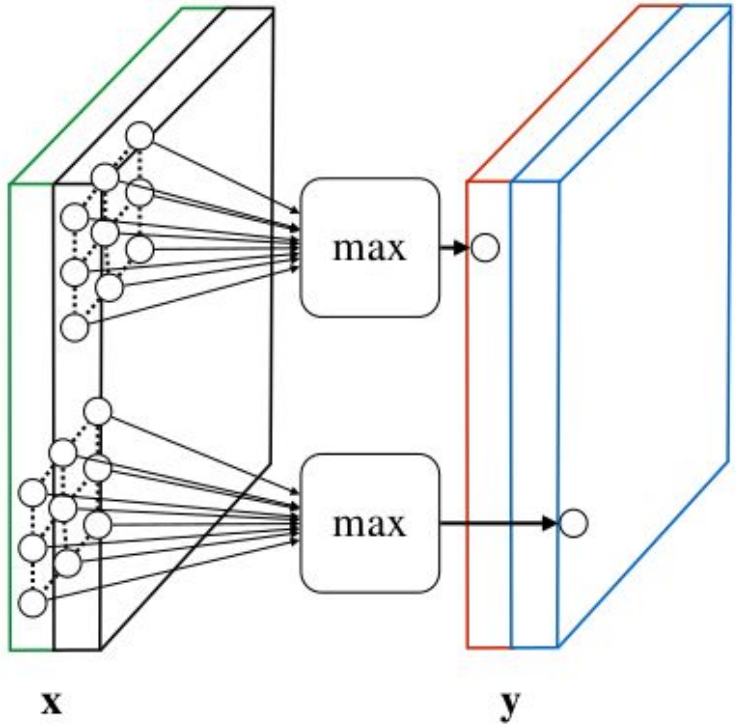
The **max pooling** operator is similar to linear filter, operating transitively on $F = (F_1, F_2)$ sized windows.

The operator extracts the maximum response for each channel and window

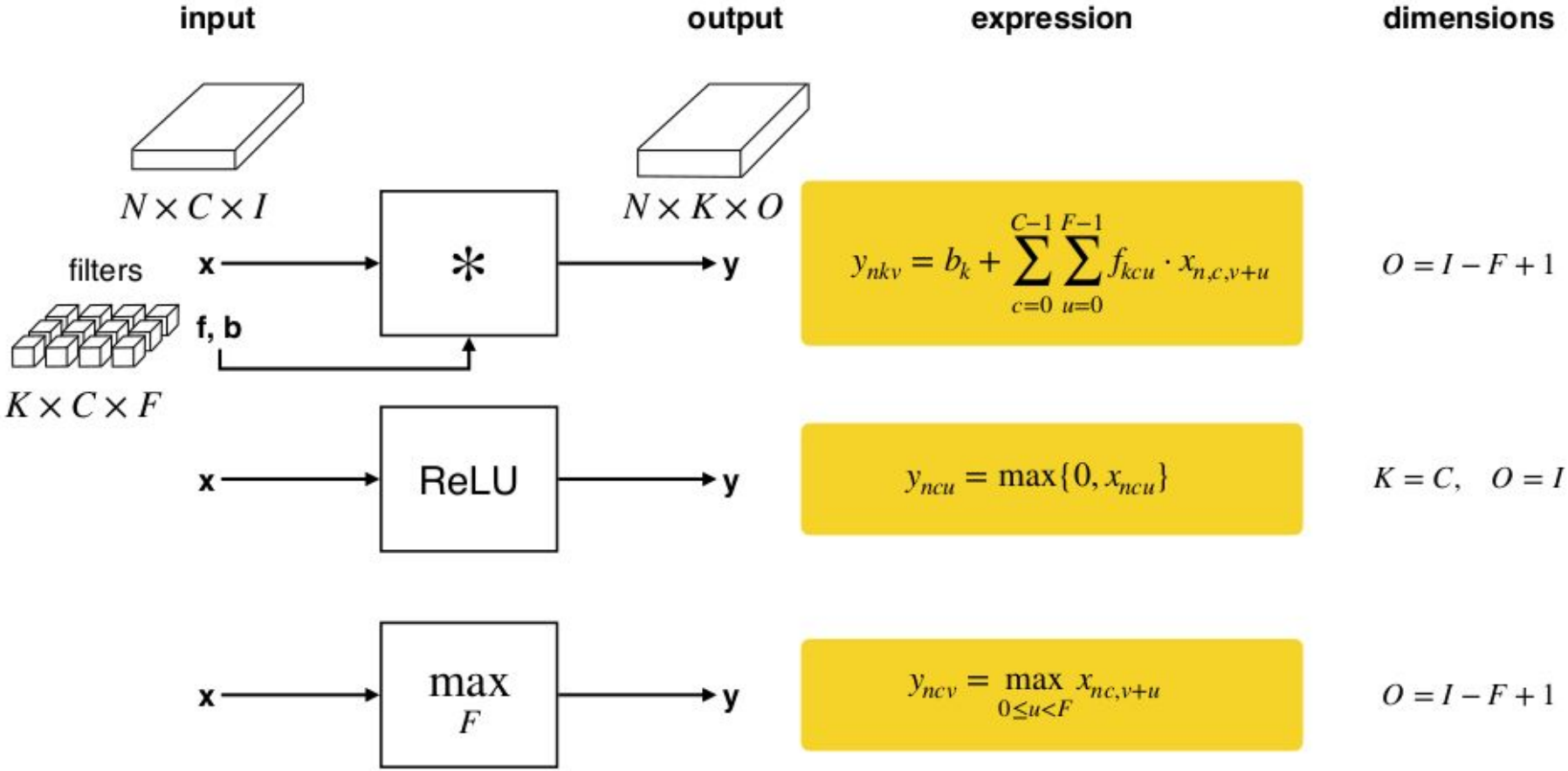
$$y_{ncv} = \max_{0 \le u < F} x_{n,c,v+u}$$

Pooling can use other operators, for example **average**

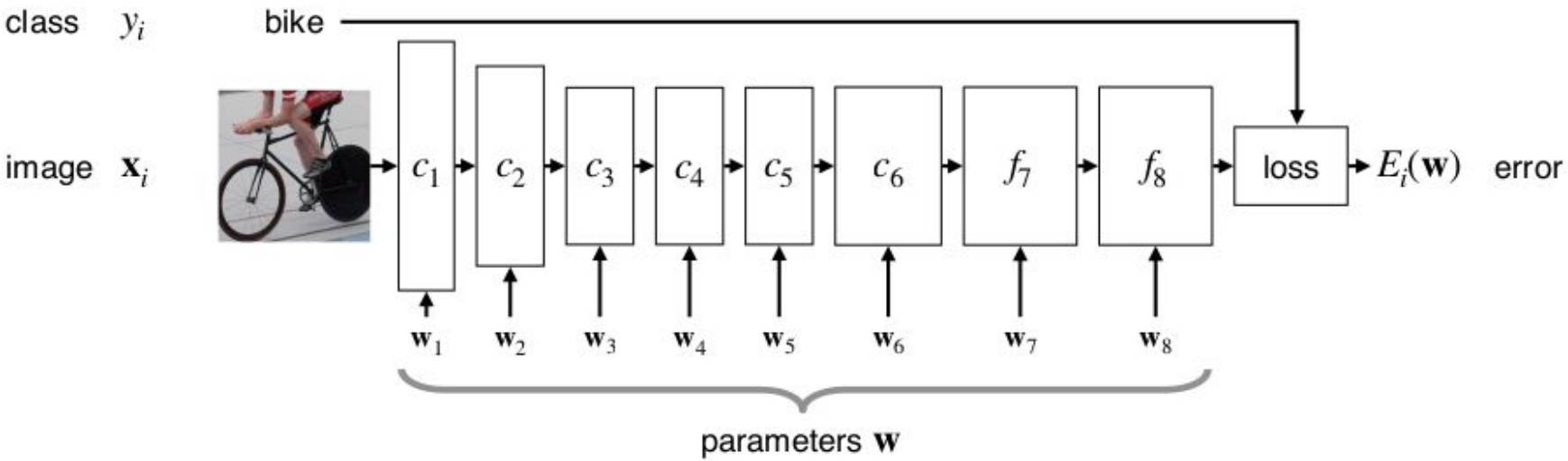
$$y_{ncv} = \frac{1}{F_1 \cdot F_2} \sum_{0 \le u < F} x_{n,c,v+u}$$



slide: Andrea Vedaldi



slide: Andrea Vedaldi



Given a dataset $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots (\mathbf{x}_N, y_N)$ the total error is obtained by averaging the cross-entropy loss.

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} E_i(\mathbf{w}), \qquad E_i(\mathbf{w}) = \ell(y_i, \Phi(\mathbf{x}_i))$$

The goal is to optimize this energy over the model parameters $\mathbf{w}$.

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} E(\mathbf{w})$$

slide: Andrea Vedaldi

The objective function is an average over $N = 1.2\text{M}$ data points, and so is the gradient. The cost of a single gradient descent update is way too large to be practical.

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} E_i(\mathbf{w}) \quad \Rightarrow \quad \nabla E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \nabla E_i(\mathbf{w})$$

Stochastic gradient

Approximate the gradient **by sampling a single data point** (or a small batch of size N' << N). Perform the gradient update using the approximation.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla E_i(\mathbf{w}_t), \qquad i \sim U(\{1,2,\ldots,N\})$$

uniform distribution

Momentum

SGD can be accelerated by denoising the gradient estimate using a moving average. This average is called **momentum**.

$$\mathbf{m}_{t+1} = 0.9\,\mathbf{m}_t + \eta_t \nabla E_i(\mathbf{w}_t), \quad \mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{m}_{t+1}$$

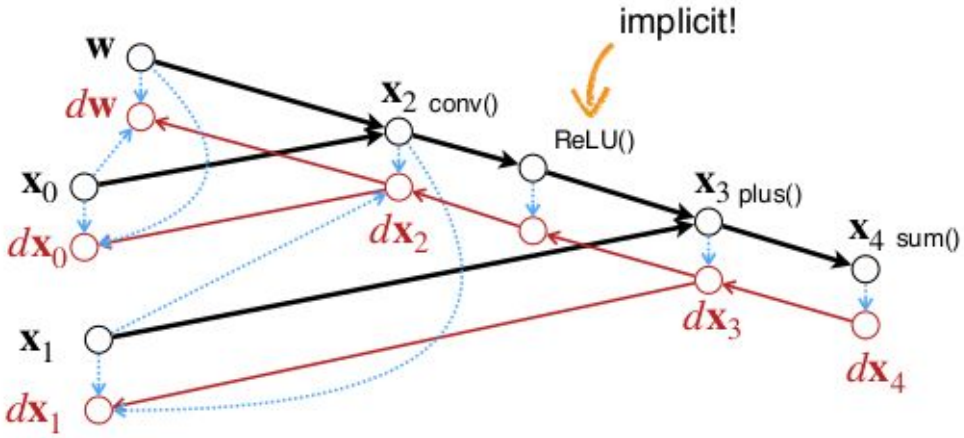
slide: Andrea Vedaldi

Modern machine learning toolboxes provide **AutoDiff**.

This means that calculations can be performed as normal in a programming language.

Underneath, the toolbox builds a compute graph.

Eventually, gradients can be requested.




```python
import torch

# Define two random inputs, both requiring grads
x0 = torch.randn(1,3,20,20, requires_grad=True)
x1 = torch.randn(1,10,18,18, requires_grad=True)

# Get a convolutional layer. It contains
# a parameter tensor conv.weight with requires_grad=True
conv = torch.nn.Conv2d(3,10,3)

# Intermediate calculations
x2 = conv(x0)
x3 = torch.nn.ReLU()(x2) + x1
x4 = x3.sum() # Scalar!

# Invoke AutoGrad to compute the gradients
x4.backward()

# Print the gradient shapes
print(x0.grad.shape)
print(x1.grad.shape)
print(conv.weight.grad.shape)
```

slide: Andrea Vedaldi

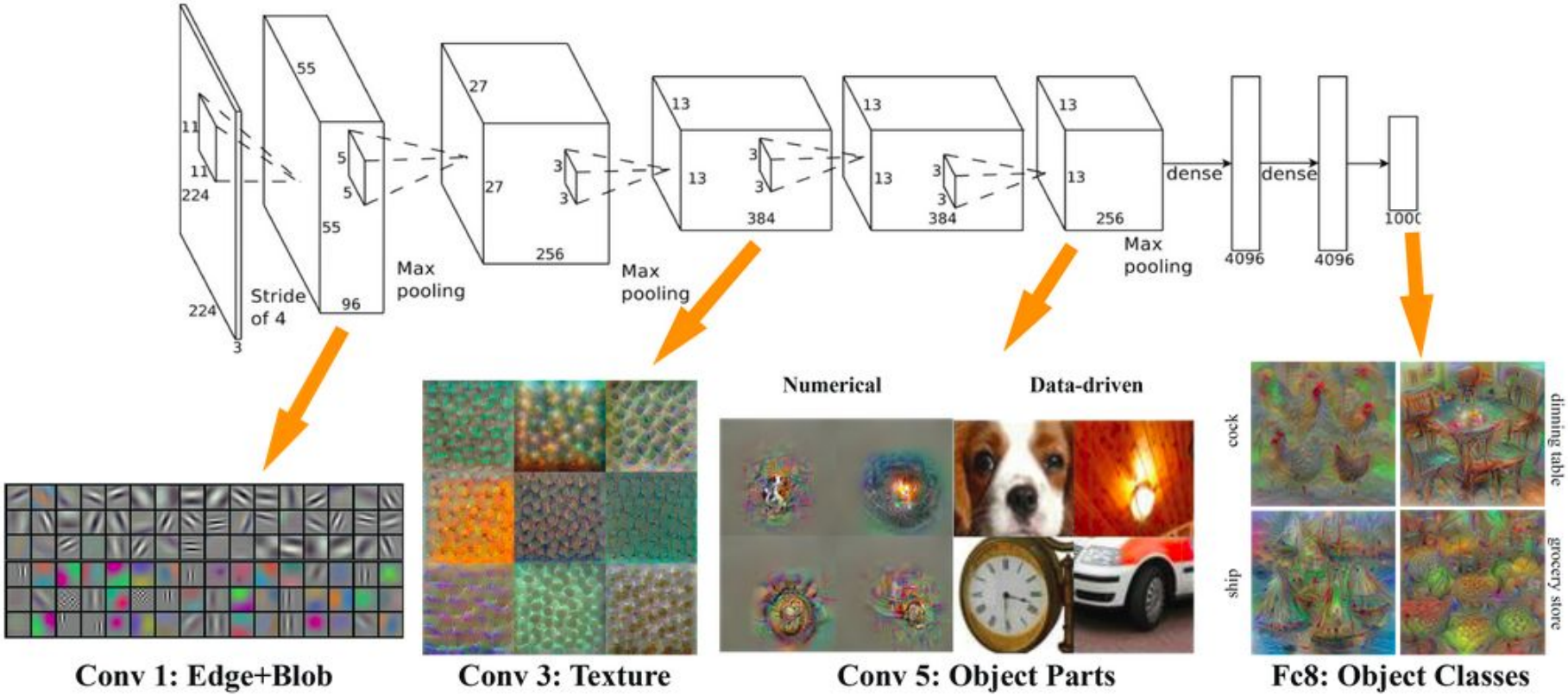


fig. http://vision03.csail.mit.edu/cnn_art/

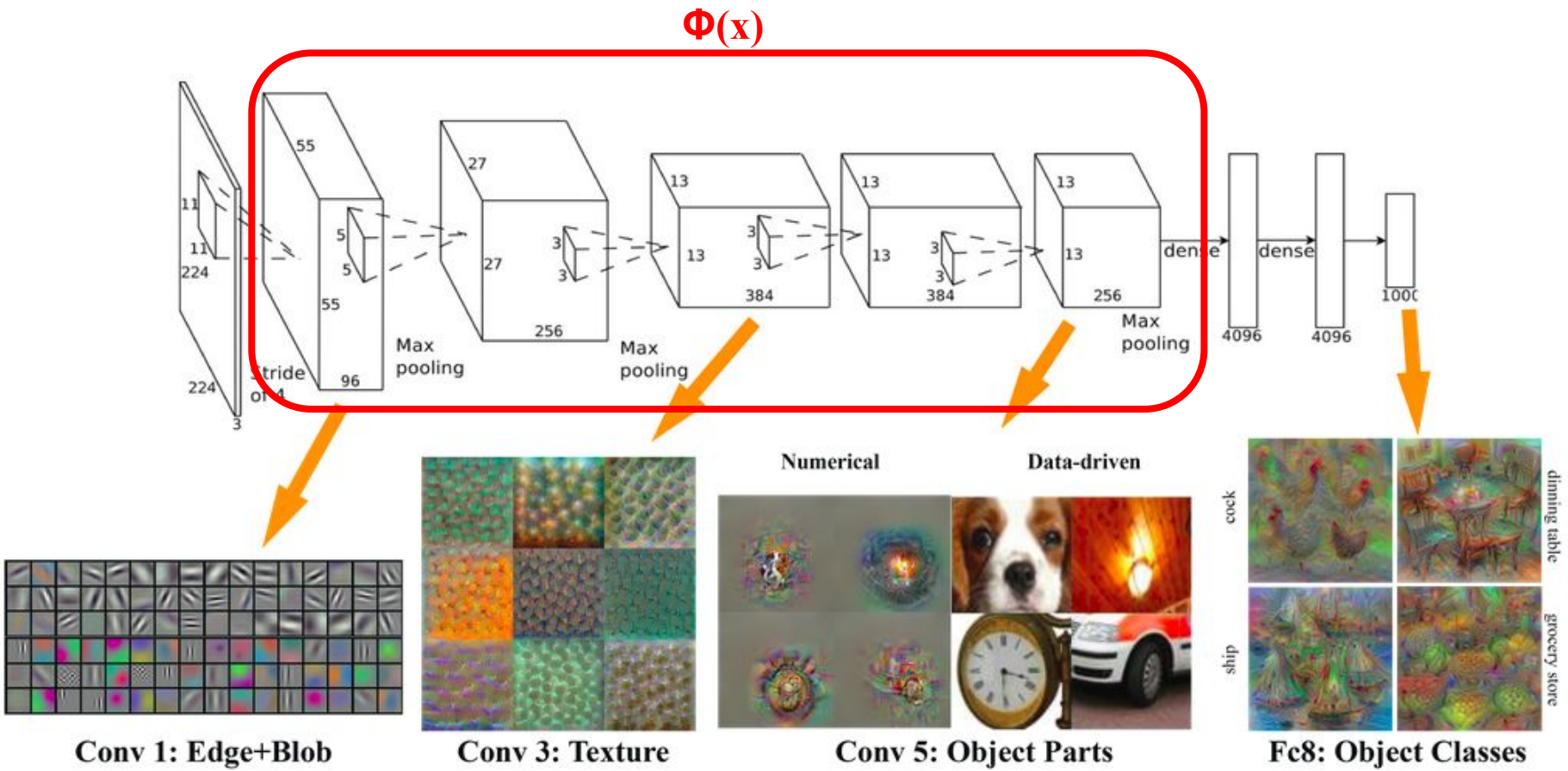
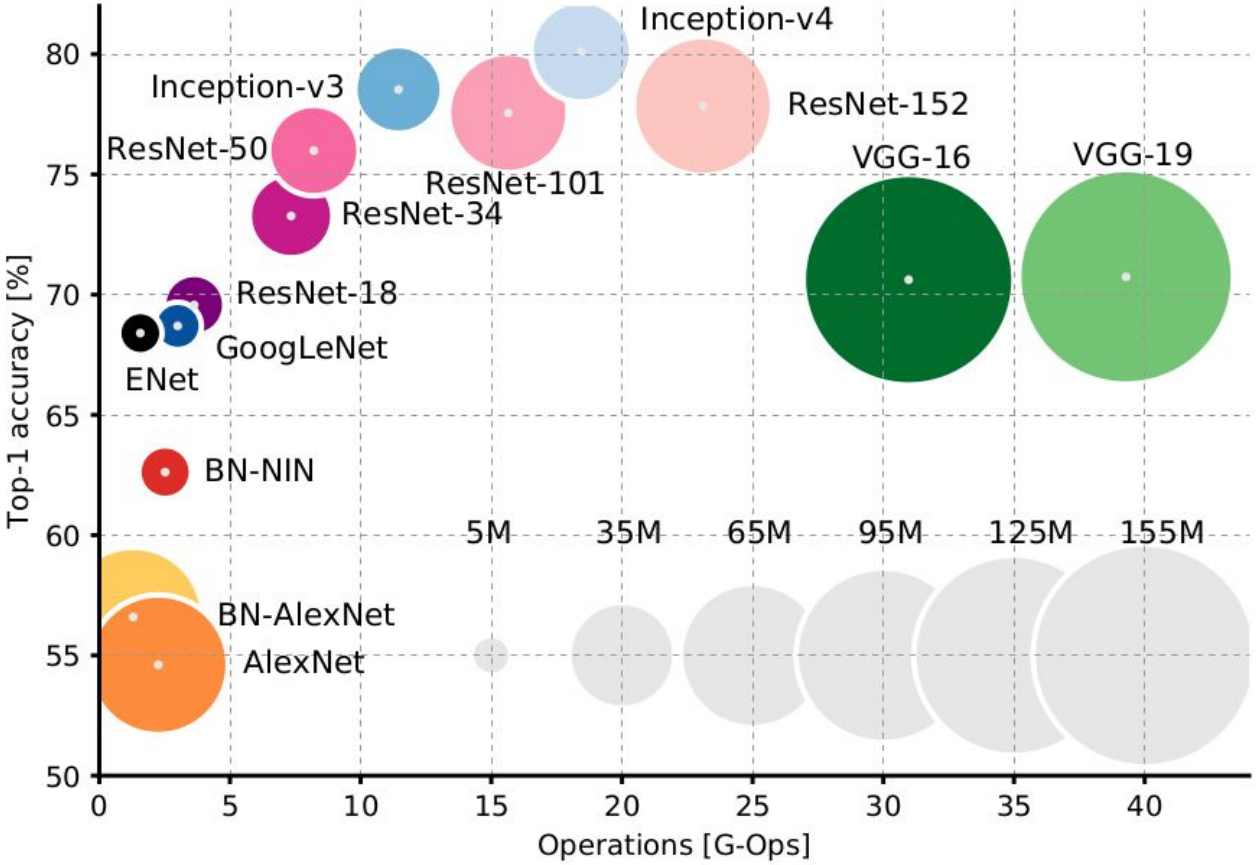
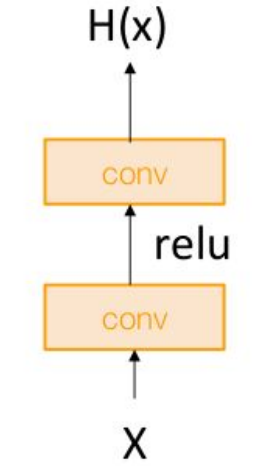


fig. http://vision03.csail.mit.edu/cnn_art/

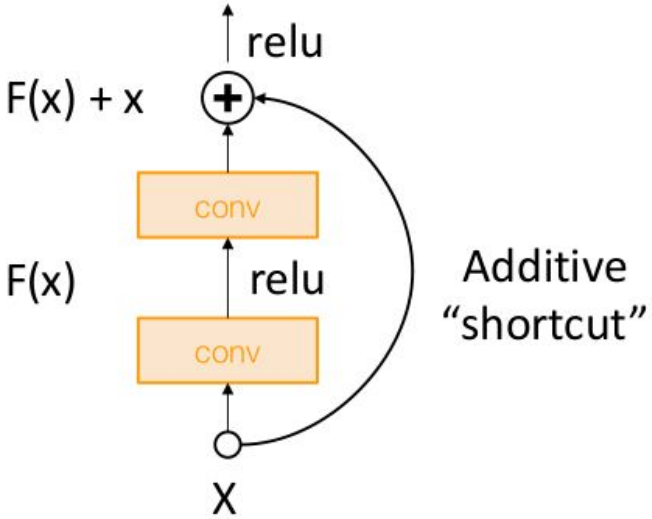


Canziani et al. *An analysis of deep neural network models for practical applications* (2016)

Residual Networks



"Plain" block



Residual Block

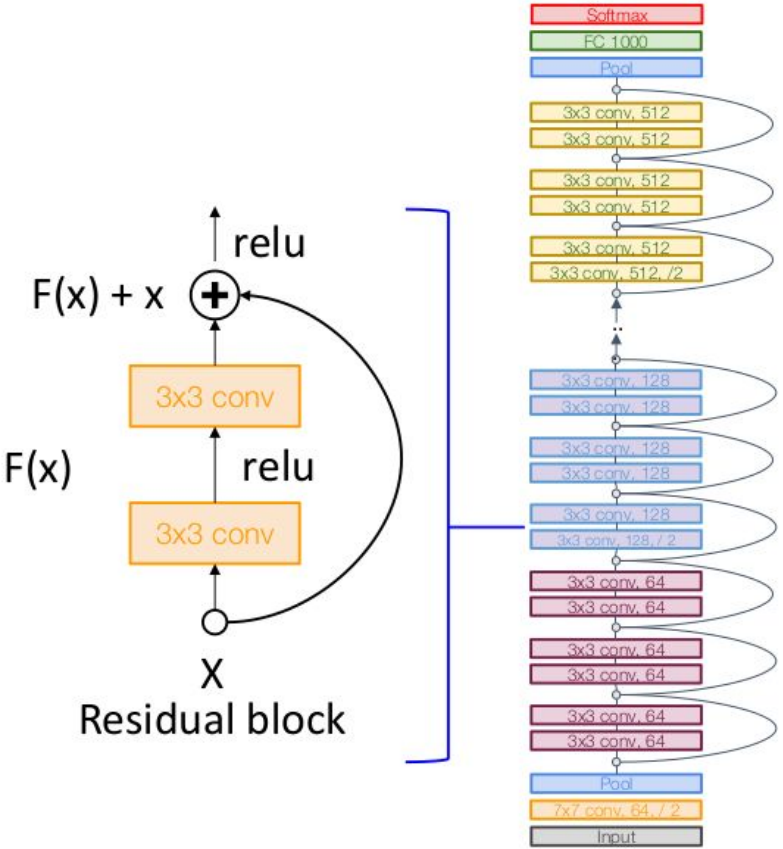
He et al. (2015) Deep Residual Learning for Image Recognition

Residual Networks

A residual network is a stack of many residual blocks

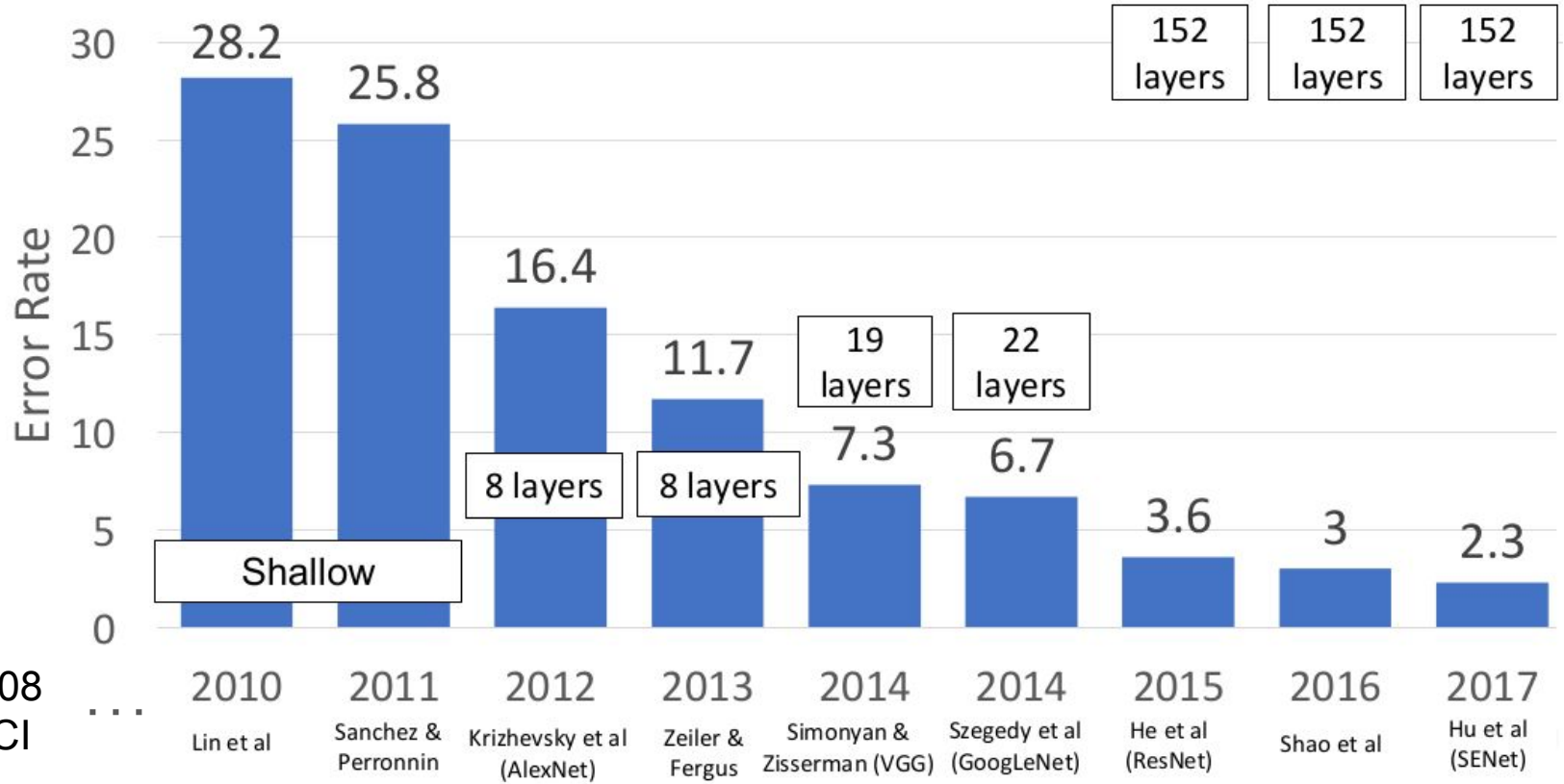
Regular design, like VGG: each residual block has two 3x3 conv

Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels



slide: Justin Johnson

ImageNet Classification Challenge



slide: Justin Johnson

Representations for language

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000)

slide: C. Manning

Representing words as discrete symbols

Example: in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

slide: C. Manning

Representing words as vectors

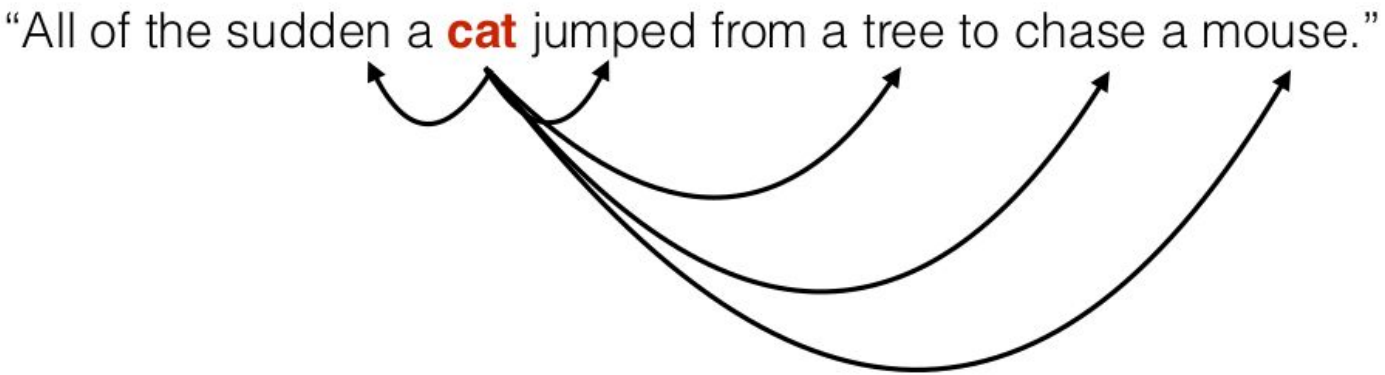
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus ("body") of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

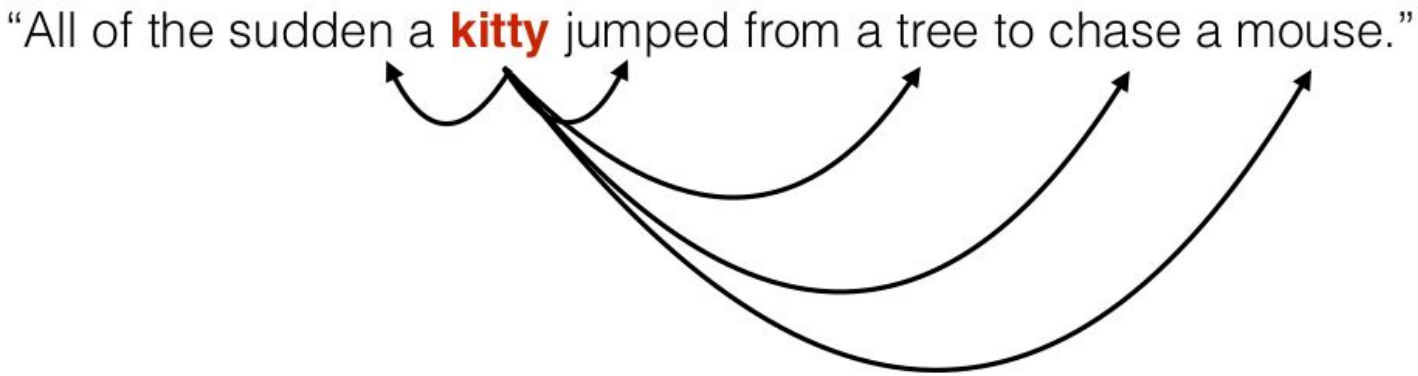
Mikolov et al. (2013) Efficient estimation of word representations in vector space

Representing words as vectors



The meaning of a word is determined by its context.

Representing words as vectors



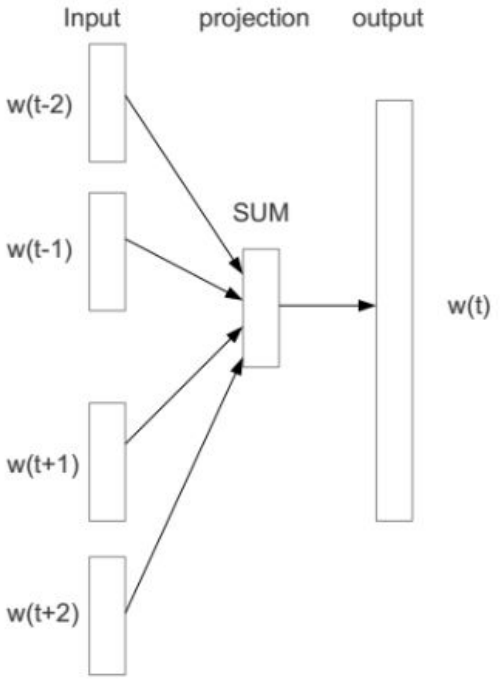
The meaning of a word is determined by its context.

Two words mean similar things if they have similar context.

(distributional hypothesis)

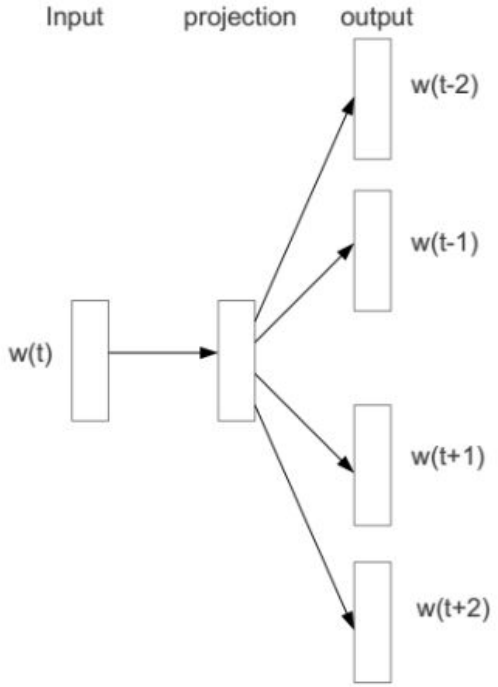
Word vectors (I)

- The ‘continuous bag-of-words model’ (CBOW) adds inputs from words within short window to predict the current word
- The weights for different positions are shared
- Computationally much more efficient than n-gram NNLM of (Bengio, 2003)
- The hidden layer is just linear



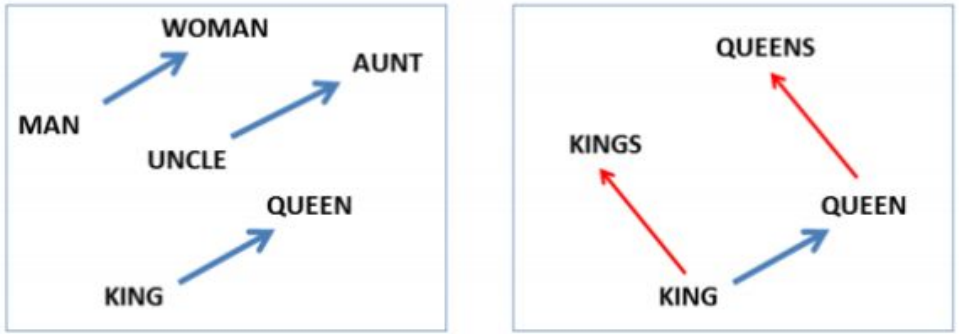
Word vectors (II)

- Predict surrounding words using the current word
- This architectures is called ‘skip-gram NNLM’
- If both are trained for sufficient number of epochs, their performance is similar



Word vectors similarity

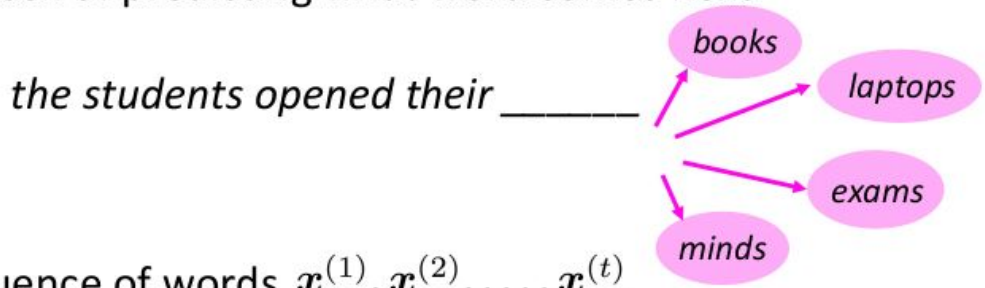
- Recently, it was shown that word vectors capture many linguistic properties (gender, tense, plurality, even semantic concepts like “capital city of”)
- We can do nearest neighbor search around result of vector operation “king – man + woman” and obtain “queen”



Mikolov et al. (2013) Linguistic regularities in continuous space word representations

Sequence modeling
Language models

- **Language Modeling** is the task of predicting what word comes next



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**

slide: C. Manning

Sequence modeling
Language models

- You can also think of a Language Model as a system that assigns probability to a piece of text

- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

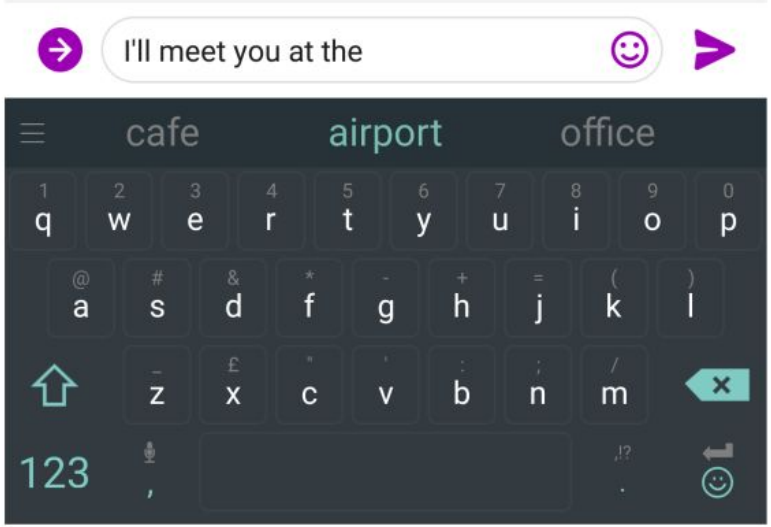
$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \, \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \, \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

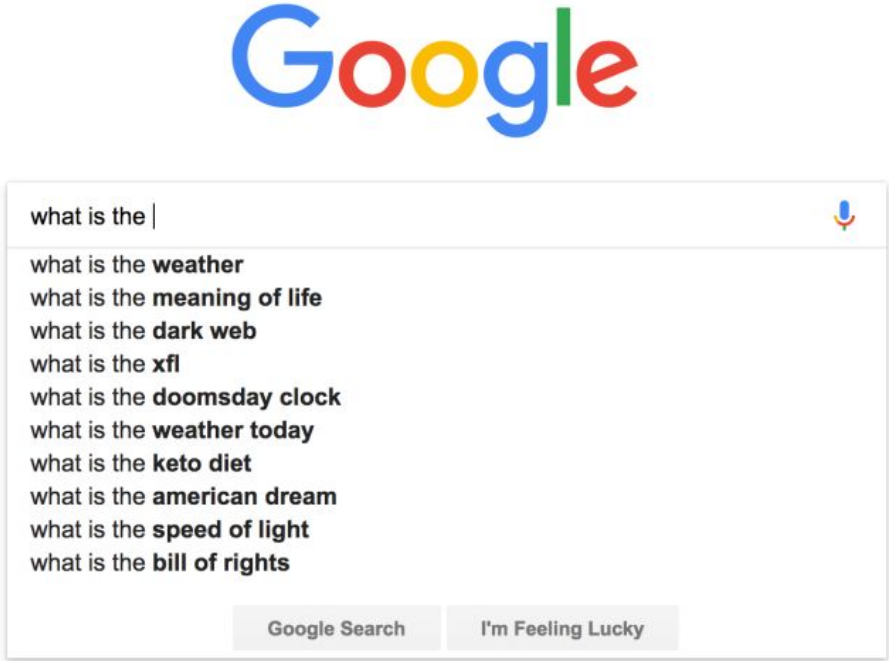
$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

slide: C. Manning

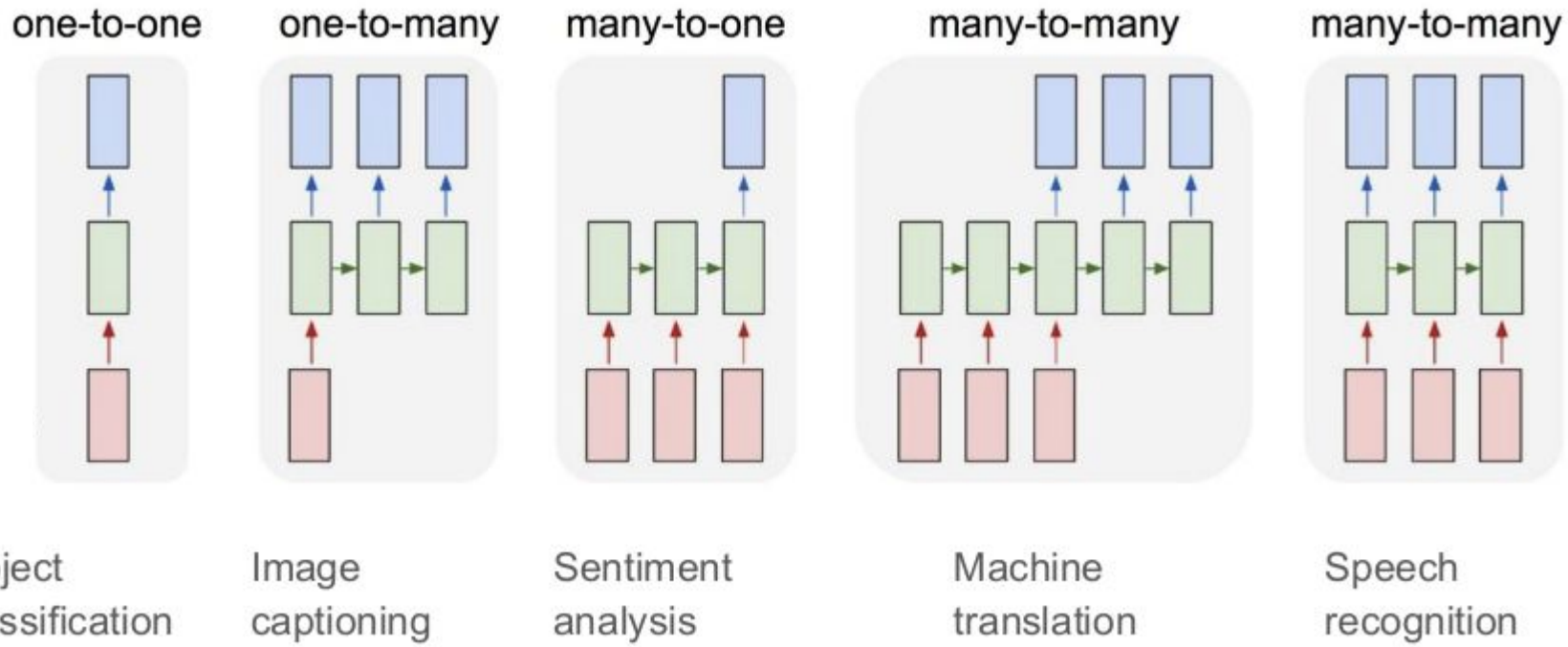
Sequence modeling
Language models





slide: C. Manning

Sequence modeling
Architectures

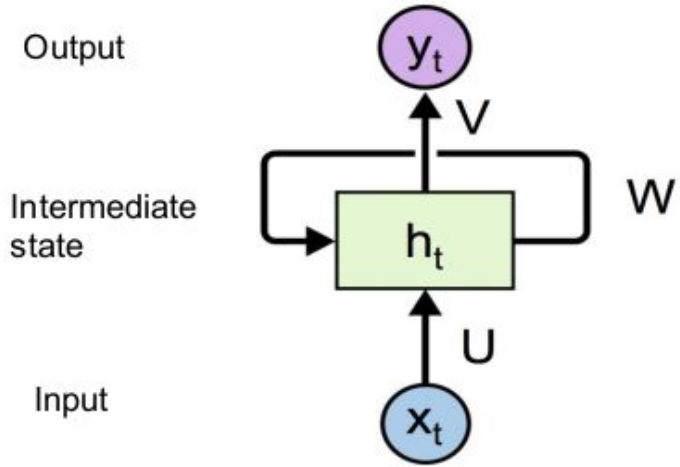


Recurrent Neural Networks

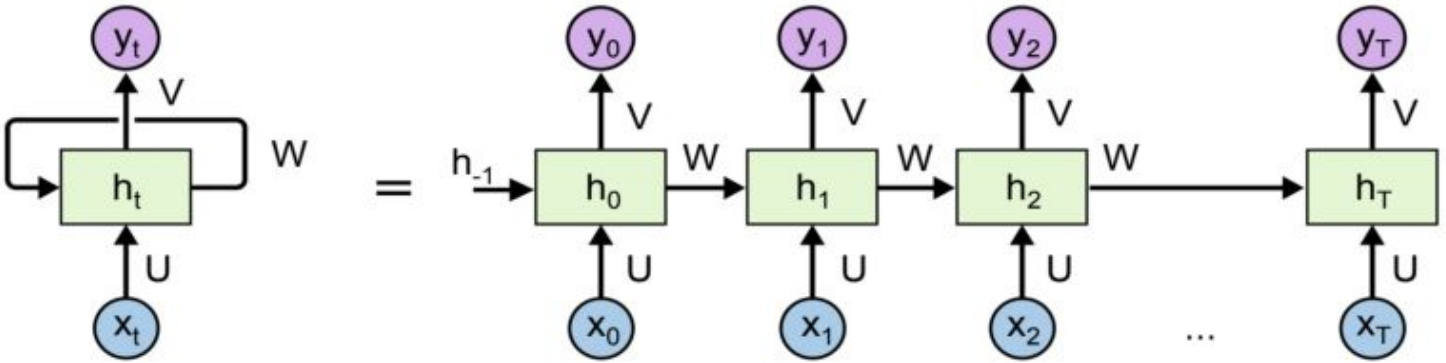
- Input: $x_0, x_1, \ldots, x_T$
- Output: $y_0, y_1, \ldots, y_T$
- Intermediate states: $h_0, h_1, \ldots, h_T$
- Simple RNN:

$$h_t = \tanh(Ux_t + Wh_{t-1})$$
$$y_t = f(Vh_t)$$

- Parameters shared across time

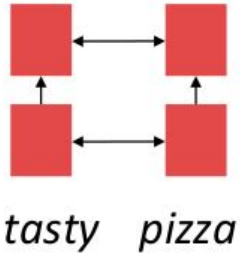


Recurrent Neural Networks

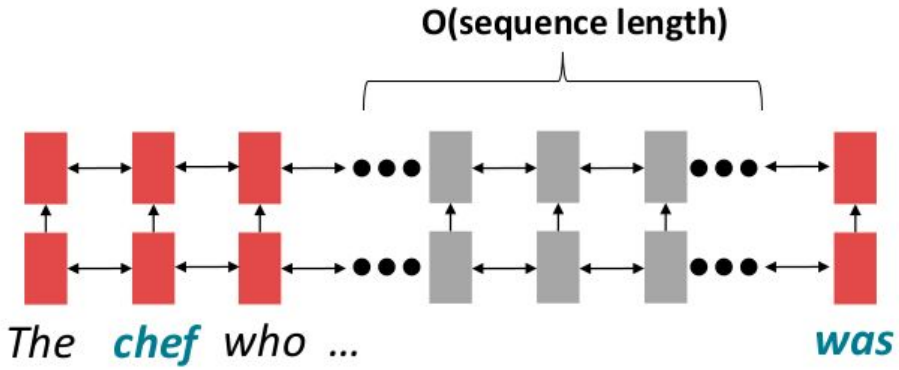


Issues with recurrent models: *linear interaction distance*

- RNNs are unrolled "left-to-right".
- This encodes linear locality: a useful heuristic!
  - Nearby words often affect each other's meanings



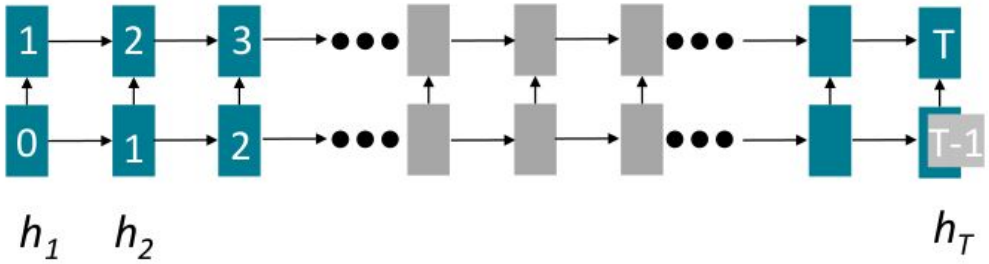
- **Problem:** RNNs take **O(sequence length)** steps for distant word pairs to interact.



slide: C. Manning

Issues with recurrent models: *lack of parallelizability*

- Forward and backward passes have **O(sequence length)** unparallelizable operations
  - GPUs can perform a bunch of independent computations at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
  - Inhibits training on very large datasets!

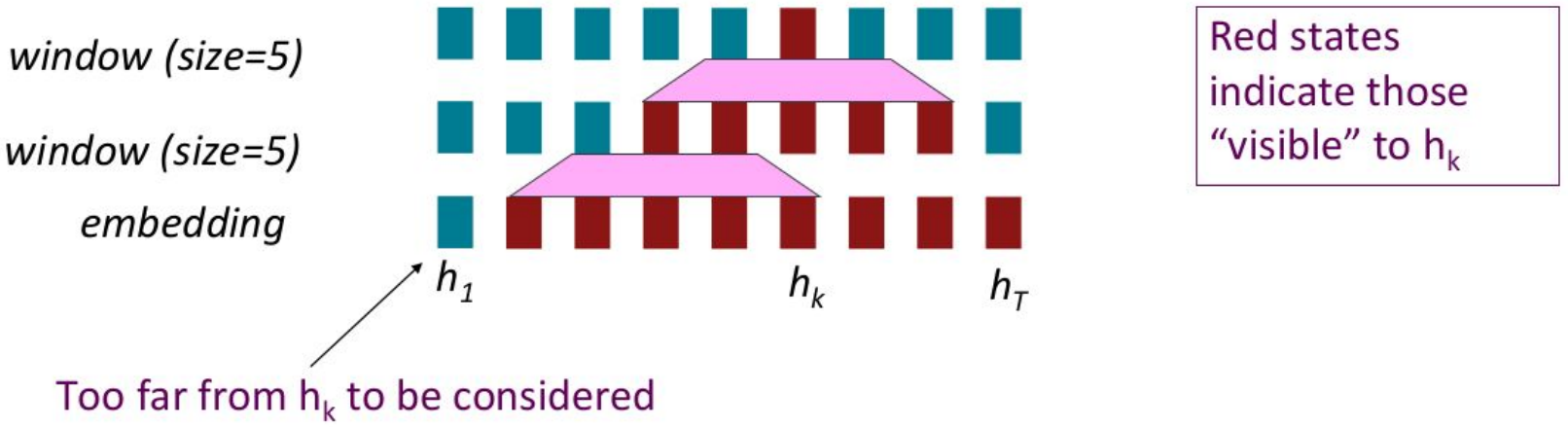


Numbers indicate min # of steps before a state can be computed

slide: C. Manning

If not recurrence, how about word windows?

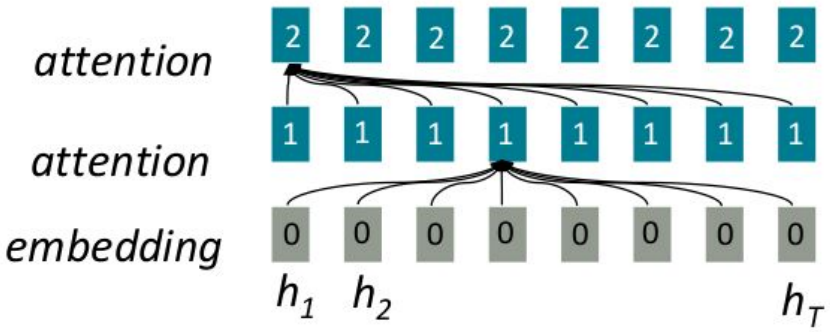
- **Word window models aggregate local contexts**
- What about long-distance dependencies?
  - Stacking word window layers allows interaction between farther words
- Maximum Interaction distance = **sequence length / window size**
  - (But if your sequences are too long, you'll just ignore long-distance context)



slide: C. Manning

If not recurrence, how about attention?

- **Attention** treats each word's representation as a **query** to access and incorporate information from **a set of values.**
- Number of unparallelizable operations does not increase sequence length.
- Maximum interaction distance: O(1), since all words interact at every layer!

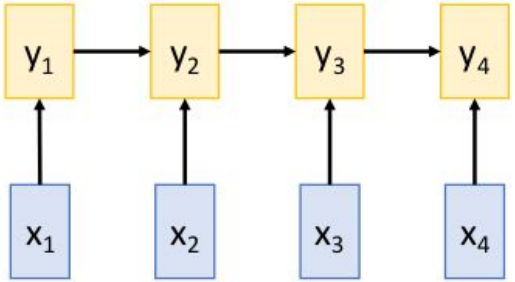


All words attend to all words in previous layer; most arrows here are omitted

slide: C. Manning

Sequence modelling

Recurrent Neural Network

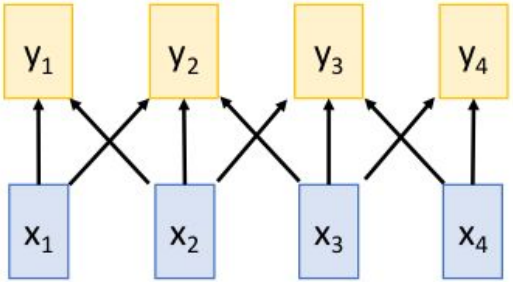


Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

1D Convolution

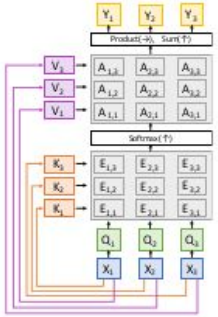


Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Self-Attention



Works on **Sets of Vectors**

(-) Good at long sequences: after one self-attention layer, each output "sees" all inputs!

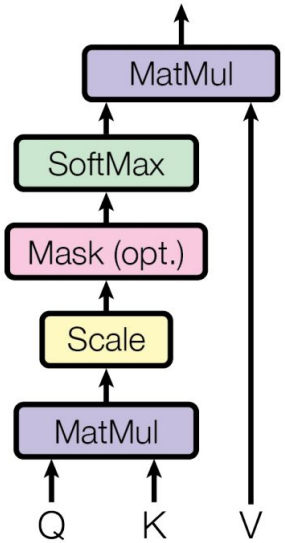
(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

slide: Justin Johnson

Self-attention

- Recall: Attention operates on **queries**, **keys**, and **values.**
  - We have some **queries** $q_1, q_2, \ldots, q_T$. Each query is $q_i \in \mathbb{R}^d$
  - We have some **keys** $k_1, k_2, \ldots, k_T$. Each key is $k_i \in \mathbb{R}^d$
  - We have some **values** $v_1, v_2, \ldots, v_T$. Each value is $v_i \in \mathbb{R}^d$
- In **self-attention**, the queries, keys, and values are drawn from the same source.
  - For example, if the output of the previous layer is $x_1, \ldots, x_T$, (one vec per word) we could let $v_i = k_i = q_i = x_i$ (that is, use the same vectors for all of them!)
- The (dot product) self-attention operation is as follows:



$$e_{ij} = \frac{q_i^\top k_j}{\sqrt{d}}$$

Compute **key-query** affinities

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

Compute attention weights from affinities (softmax)

$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

Compute outputs as weighted sum of **values**

slide: C. Manning

Dot-Product Attention

- Inputs: a query q and a set of key-value (k-v) pairs to an output
- Query, keys, values, and output are all vectors

- Output is weighted sum of values, where
- Weight of each value is computed by an inner product of query and corresponding key
- Queries and keys have same dimensionality d_k value have d_v

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

slide: R. Socher

Dot-Product Attention

- When we have multiple queries q, we stack them in a matrix Q:

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

- Becomes: $A(Q, K, V) = softmax(QK^T)V$

$[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]$

softmax row-wise

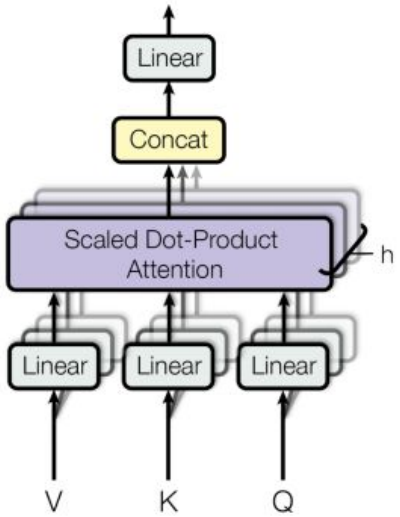
$= [|Q| \times d_v]$

slide: R. Socher

Self-attention and Multi-head attention

- The input word vectors could be the queries, keys and values
- In other words: the word vectors themselves select each other
- Word vector stack = Q = K = V
- Problem: Only one way for words to interact with one-another
- Solution: Multi-head attention
- First map Q, K, V into h many lower dimensional spaces via W matrices
- Then apply attention, then concatenate outputs and pipe through linear layer

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

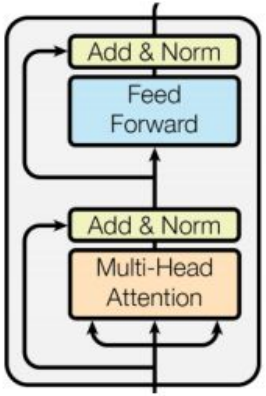


slide: R. Socher

Transformer block

- Each block has two “sublayers”

1. Multihead attention
2. 2 layer feed-forward Nnet (with relu)



Each of these two steps also has:

Residual (short-circuit) connection and LayerNorm:

LayerNorm(x + Sublayer(x))

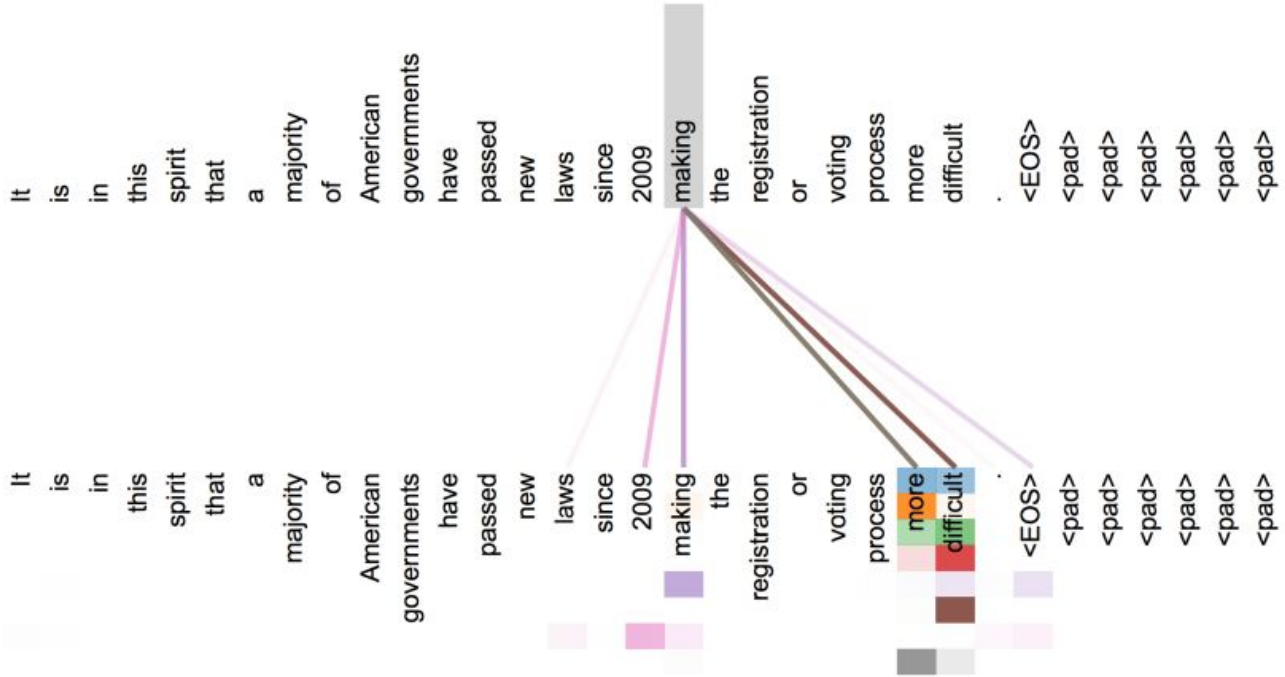
Layernorm changes input to have mean 0 and variance 1,
per layer and per training point (and adds two more parameters)

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \qquad h_i = f(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i)$$

slide: R. Socher

Attention visualization

- Words start to pay attention to other words in sensible ways



slide: R. Socher

Barriers and solutions for self-attention as a building block

Barriers	Solutions
• Doesn't have an inherent notion of order!	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	• Easy fix: apply the same feedforward network to each self-attention output.
• Need to ensure we don't "look at the future" when predicting a sequence	• Mask out the future by artificially setting attention weights to 0!

slide: C. Manning

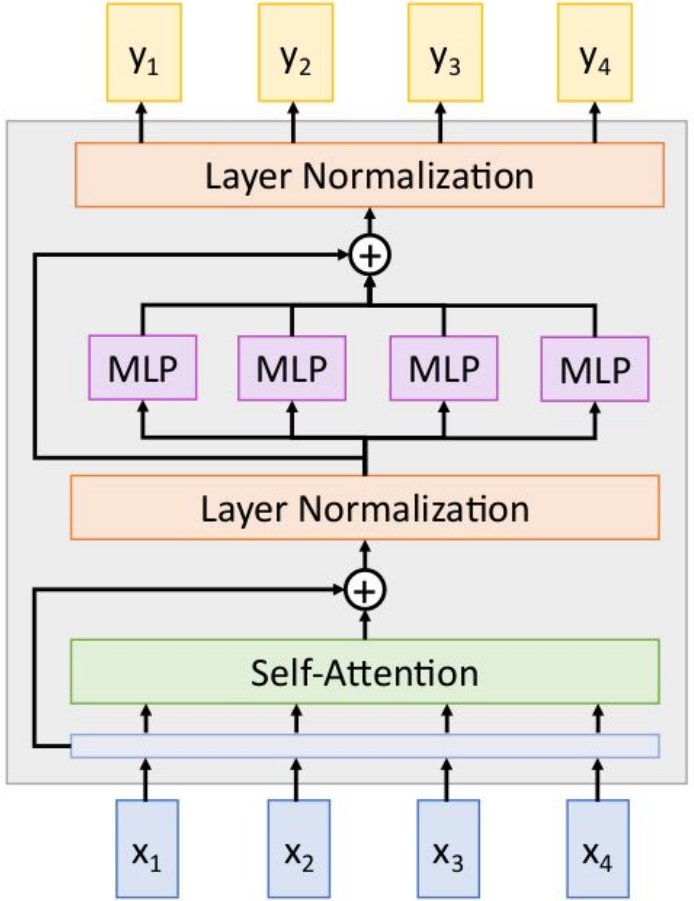
Transformers

Transformer Block:
Input: Set of vectors x
Output: Set of vectors y

Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable



slide: Justin Johnson

Transformers

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

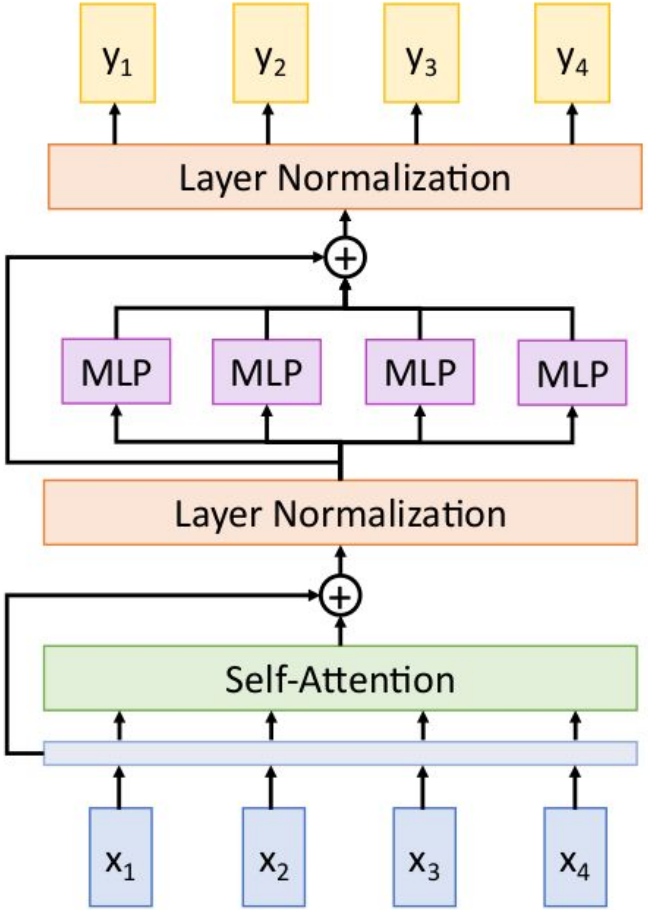
Ba et al, 2016

Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other



slide: Justin Johnson

Vaswani et al. (2017) Attention is all you need

Transformers

Transformer Block:
Input: Set of vectors x
Output: Set of vectors y

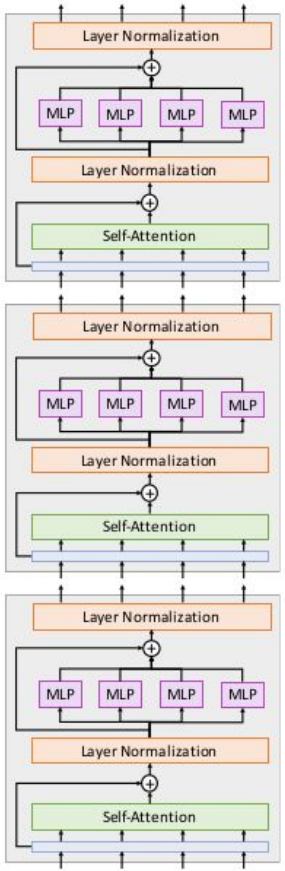
Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

A **Transformer** is a sequence of transformer blocks

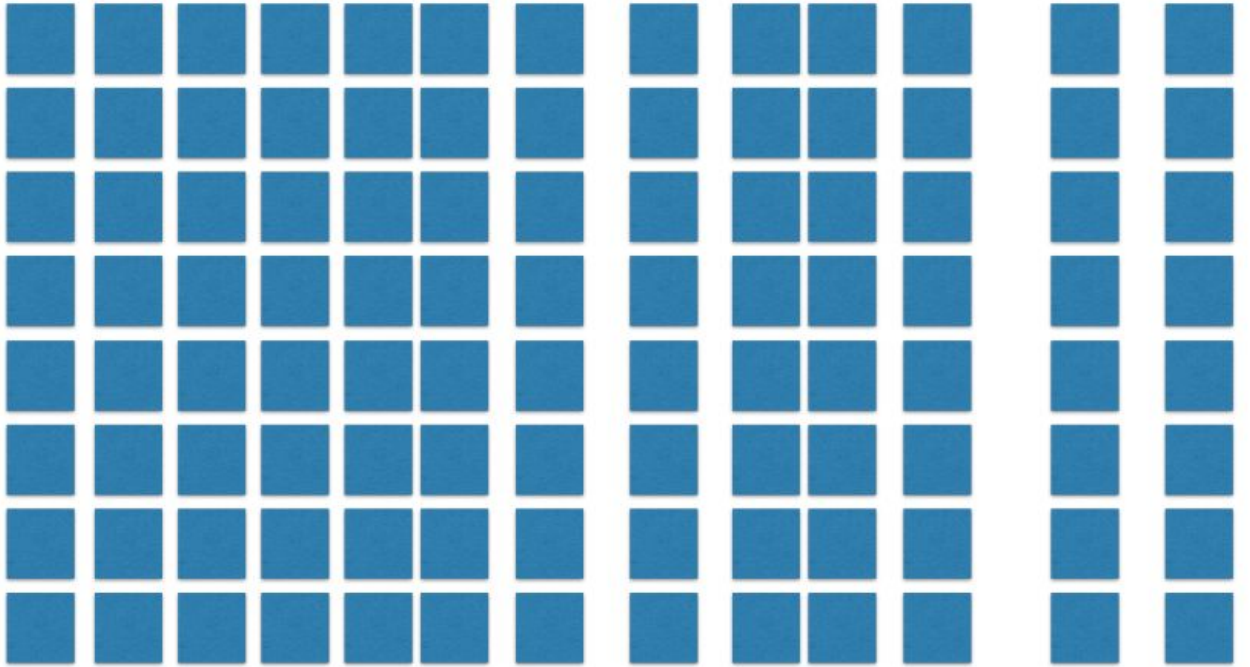
Vaswani et al:
12 blocks, D_Q=512, 6 heads



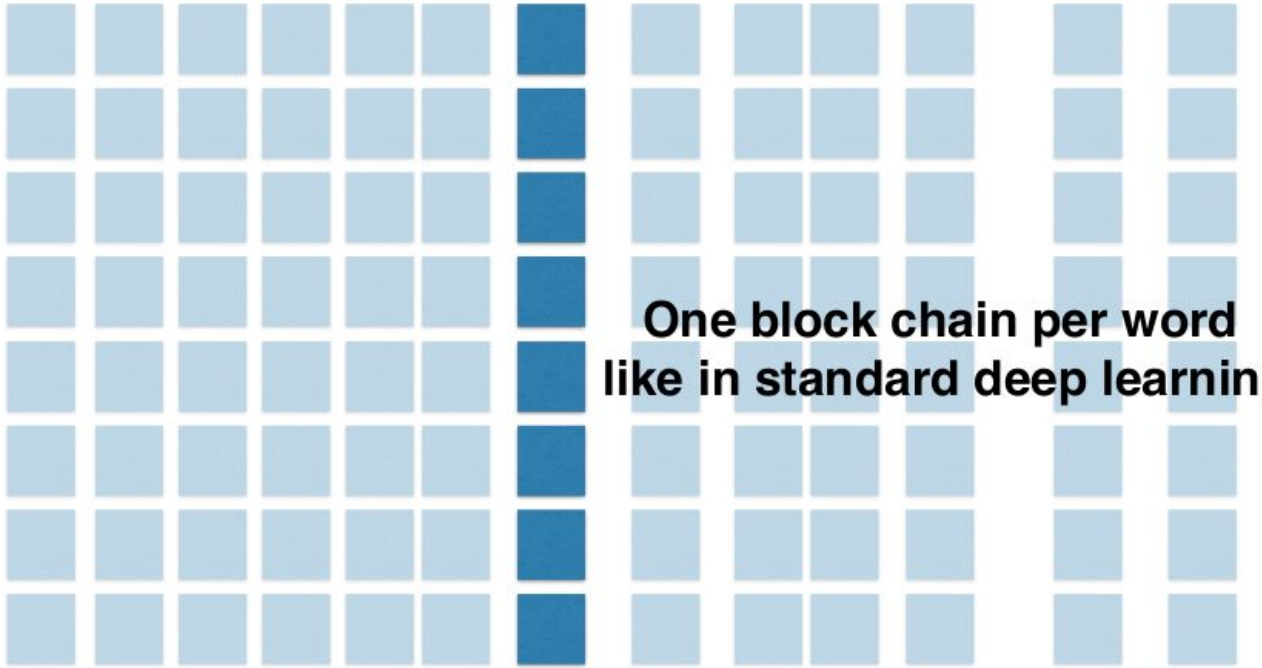
slide: Justin Johnson

Vaswani et al. (2017) Attention is all you need

BERT



J. Devlin et al. “BERT: Pre-training of deep bidirectional transformers for language understanding”, 2018

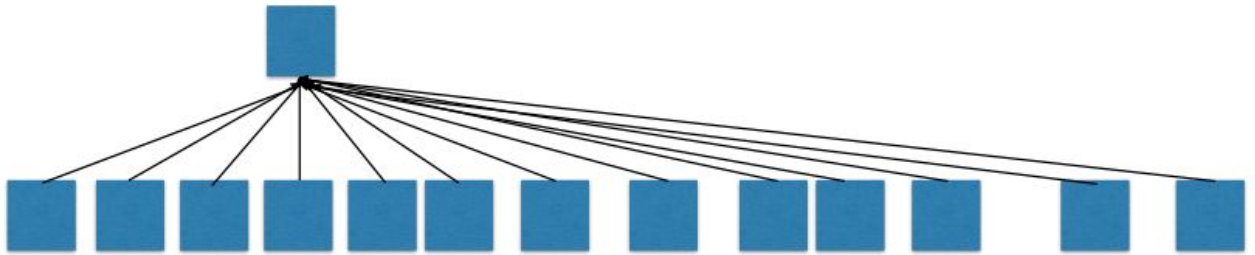


One block chain per word
like in standard deep learning

<s> The cat sat on the mat <sep> It fell asleep soon after

BERT

Each block receives input from all the blocks below.
Mapping must handle variable length sequences…

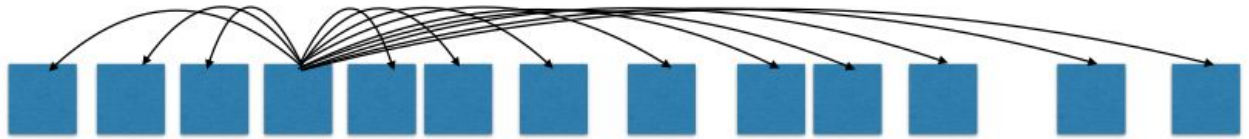


BERT

This accomplished by using attention (each block is a Transformer)

For each layer and for each block in a layer do (simplified version):

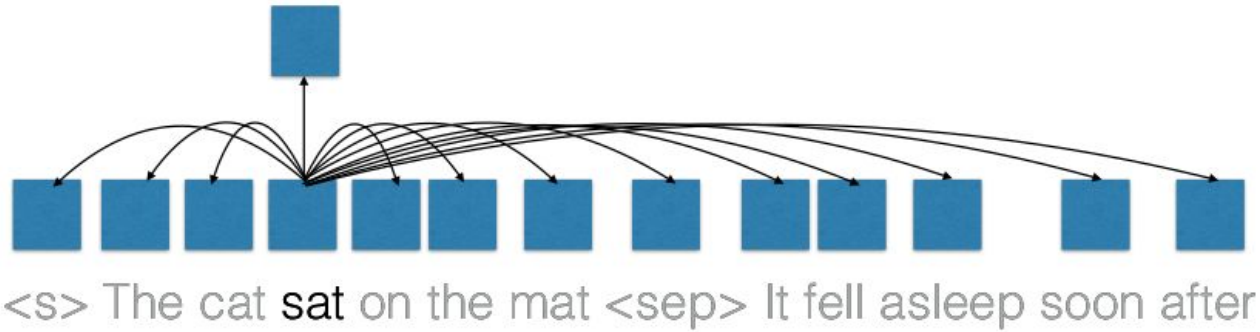
1) let each current block representation at this layer be: h_j
2) compute dot products: $h_i \cdot h_j$
3) normalize scores: $\alpha_i = \frac{\exp(h_i \cdot h_j)}{\sum_k \exp(h_k \cdot h_j)}$
4) compute new block representation as in: $h_j \leftarrow \sum_k \alpha_k h_k$



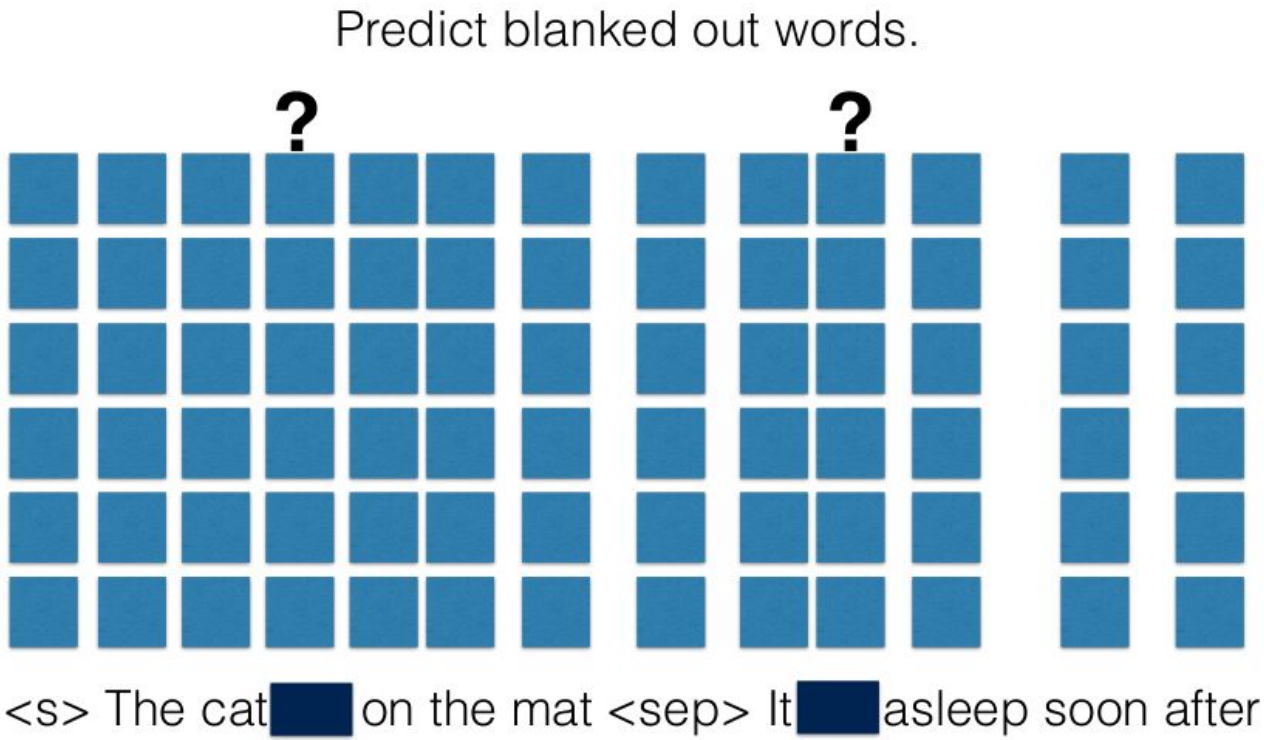
<s> The cat sat on the mat <sep> It fell asleep soon after

BERT

The representation of each word at each layer depends on all the words in the context. And there are lots of such layers…

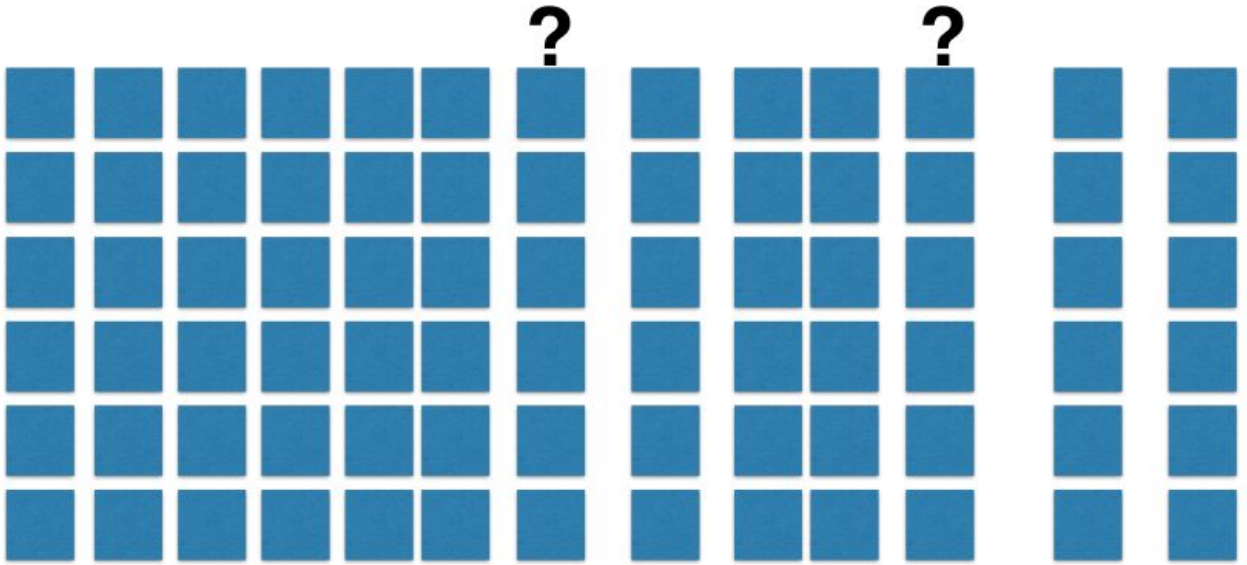


BERT Training



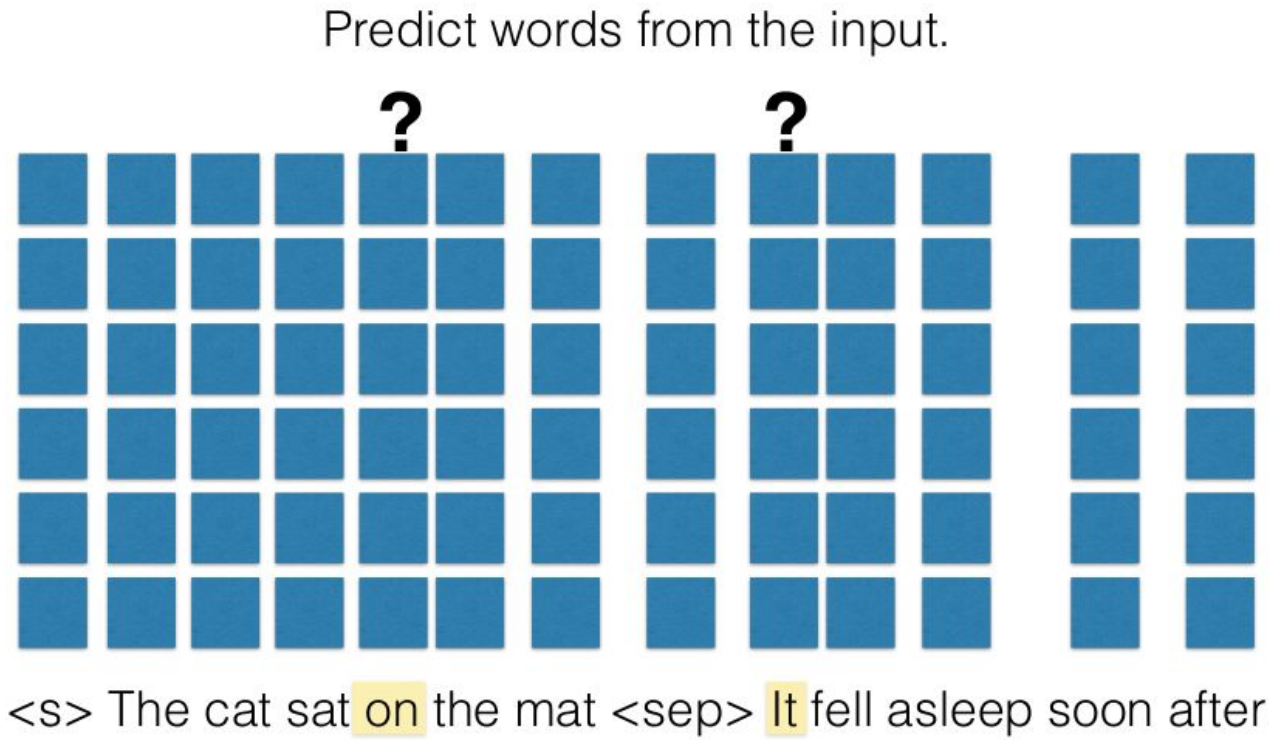
BERT Training

Predict words which were replaced with random words.



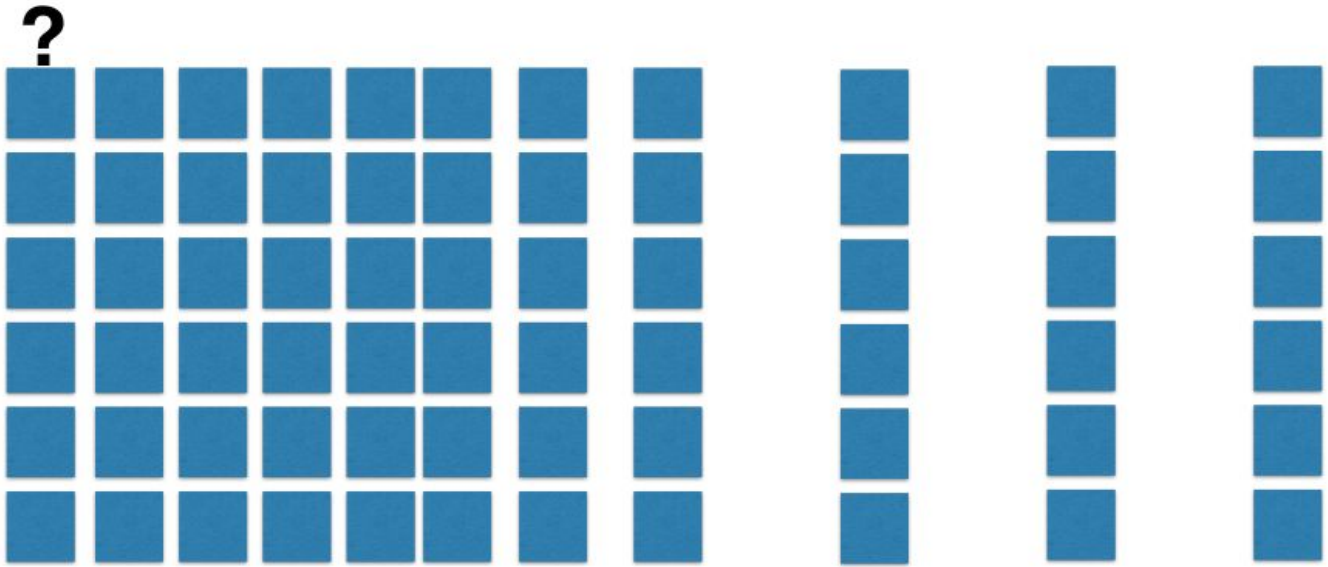
<s> The cat sat on the wine <sep> It fell scooter soon after

BERT Training

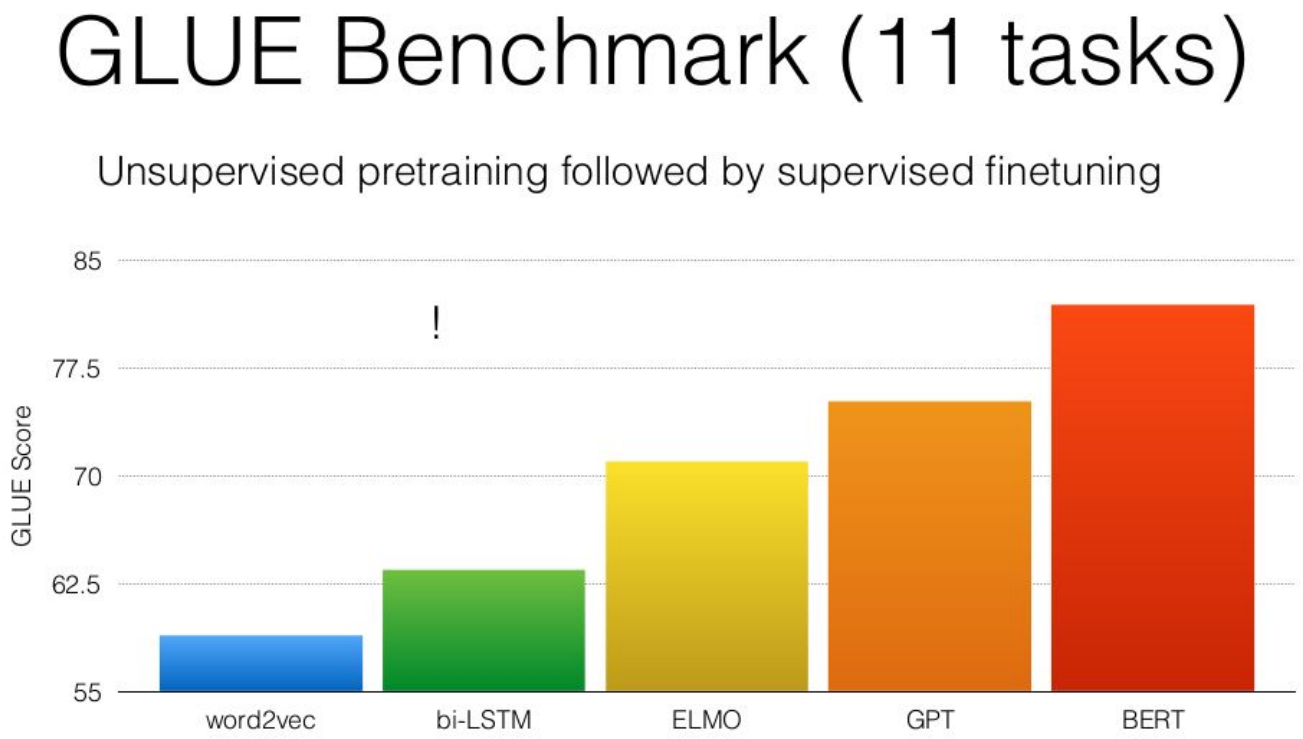


BERT Training

Predict whether the next sentence is taken at random.



BERT


GLUE Benchmark (11 tasks)
Unsupervised pretraining followed by supervised finetuning
GLUE Score
85
77.5
70
62.5
55
!
word2vec
bi-LSTM
ELMO
GPT
BERT


ECI
34
VIRTUAL