Convolutional Neural Networks

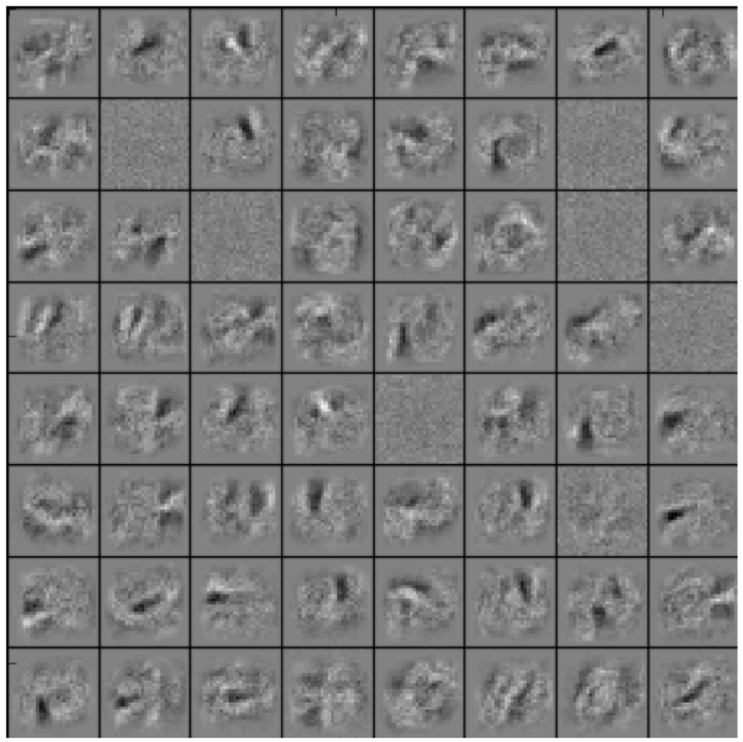
Overview

What makes vision hard?

- Vison needs to be robust to a lot of transformations or distortions:
  - change in pose/viewpoint
  - change in illumination
  - deformation
  - occlusion (some objects are hidden behind others)
- Many object categories can vary wildly in appearance (e.g. chairs)
- Geoff Hinton: "Imaging a medical database in which the age of the patient sometimes hops to the input dimension which normally codes for weight!"

Overview

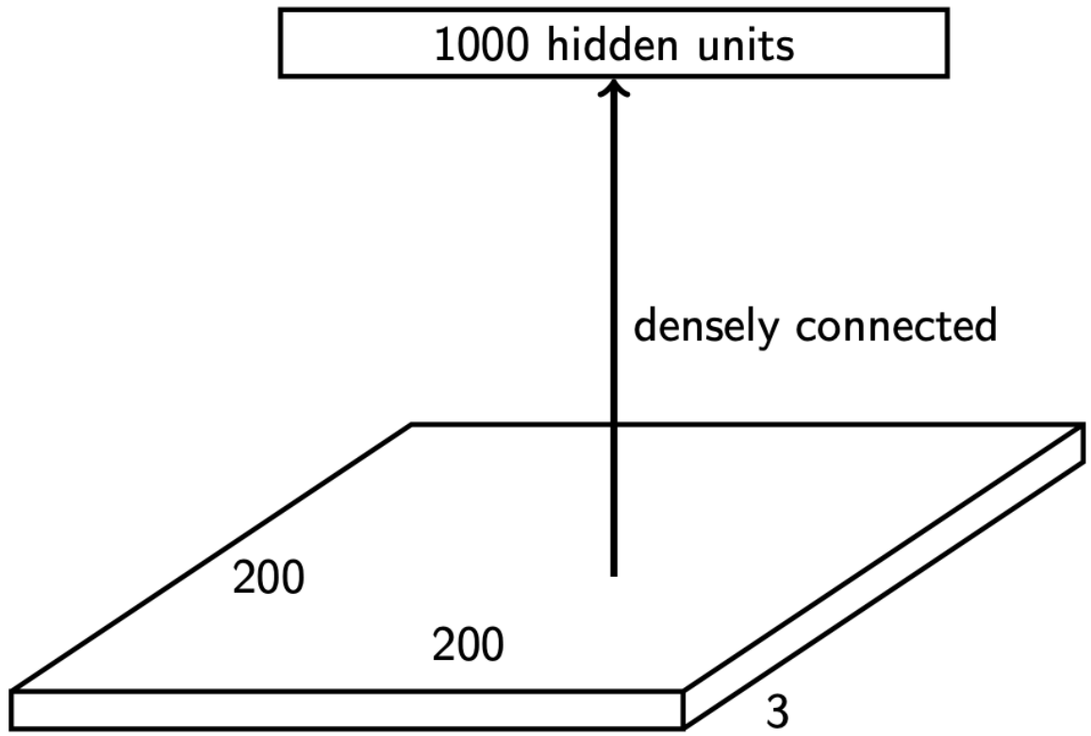
Recall we looked at some hidden layer features for classifying handwritten digits:



This isn't going to scale to full-sized images.

Overview

Suppose we want to train a network that takes a 200×200 RGB image as input.



What is the problem with having this as the first layer?

- Too many parameters! Input size $= 200 \times 200 \times 3 = 120$K. Parameters $= 120\text{K} \times 1000 = 120$ million.
- What happens if the object in the image shifts a little?

Overview

The same sorts of features that are useful in analyzing one part of the image will probably be useful for analyzing other parts as well.

E.g., edges, corners, contours, object parts

We want a neural net architecture that lets us learn a set of feature detectors that are applied at *all* image locations.

Overview

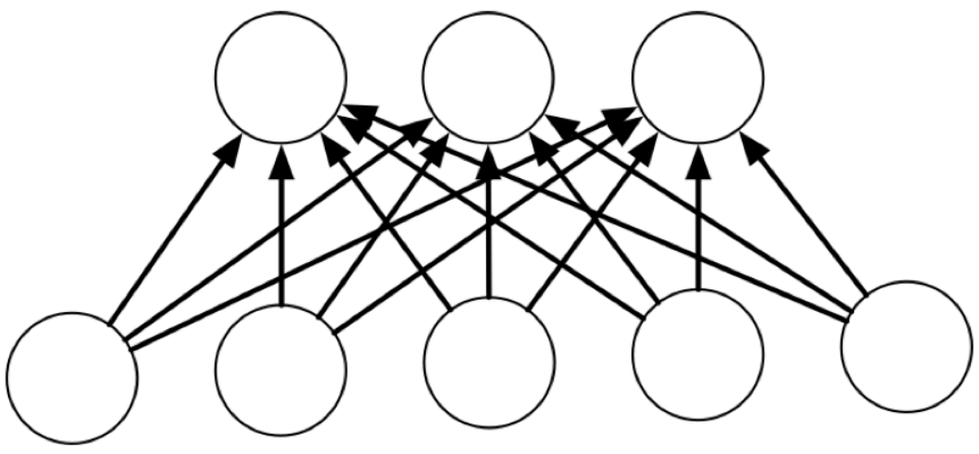
So far, we've seen two types of layers:

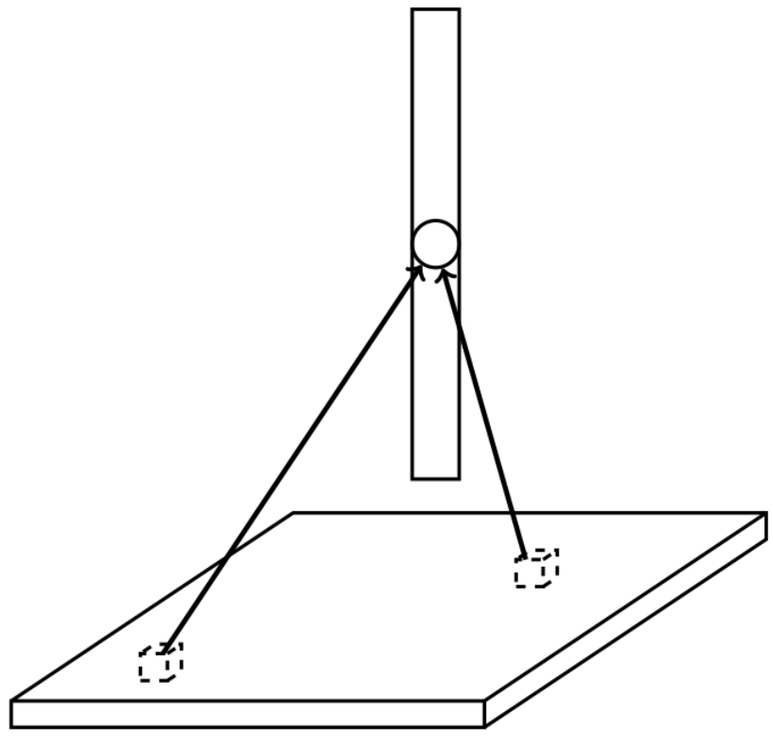
- fully connected layers
- embedding layers (i.e. lookup tables)

Different layers could be stacked together to build powerful models.
Let's add another layer type: the convolution layer.

Convolution Layers

Fully connected layers:

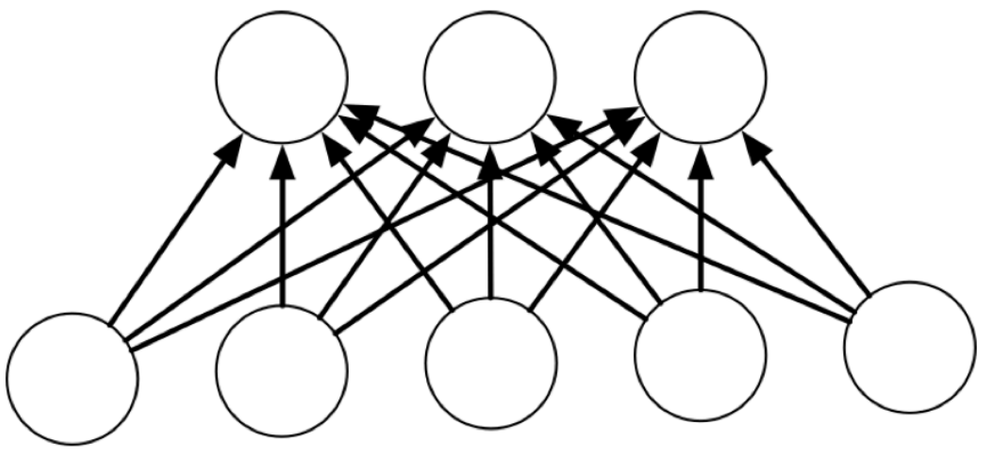


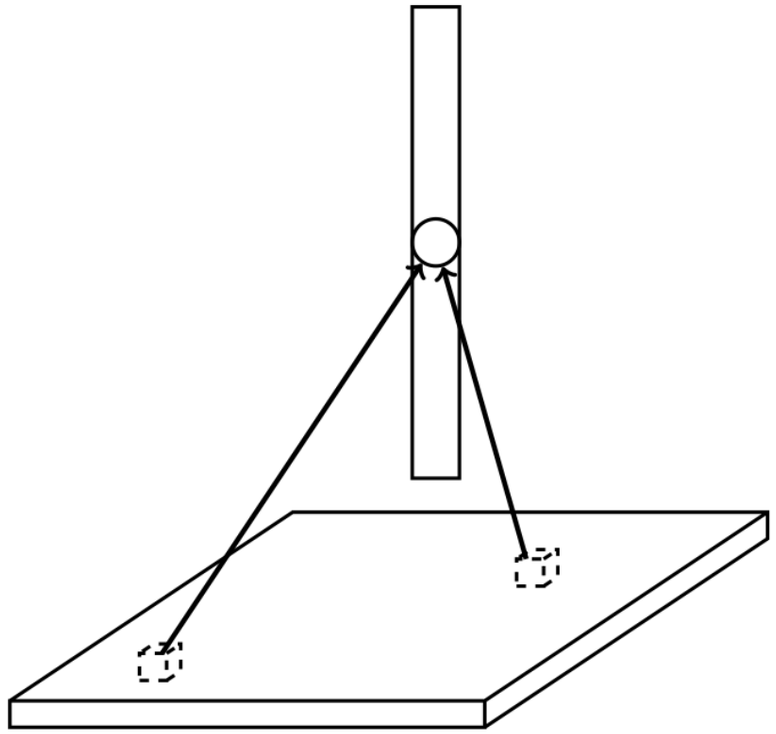


Each hidden unit looks at the entire image.

Convolution Layers

Fully connected layers:

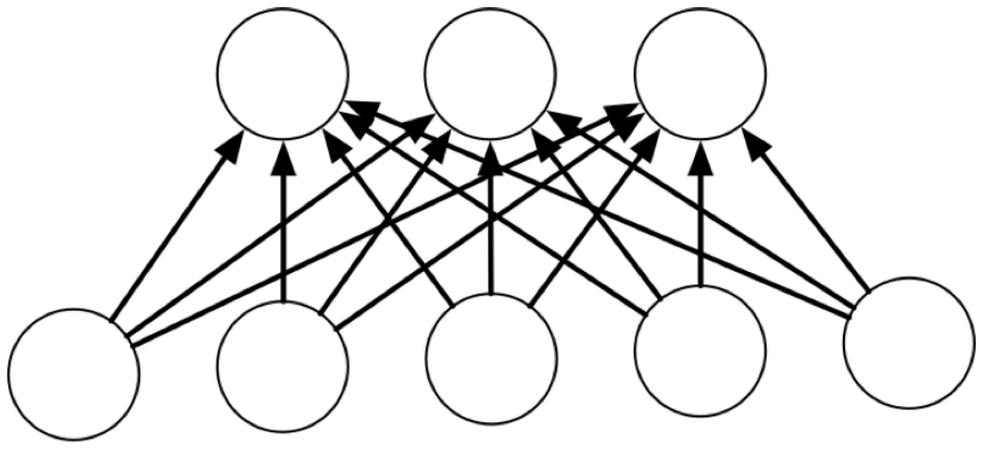


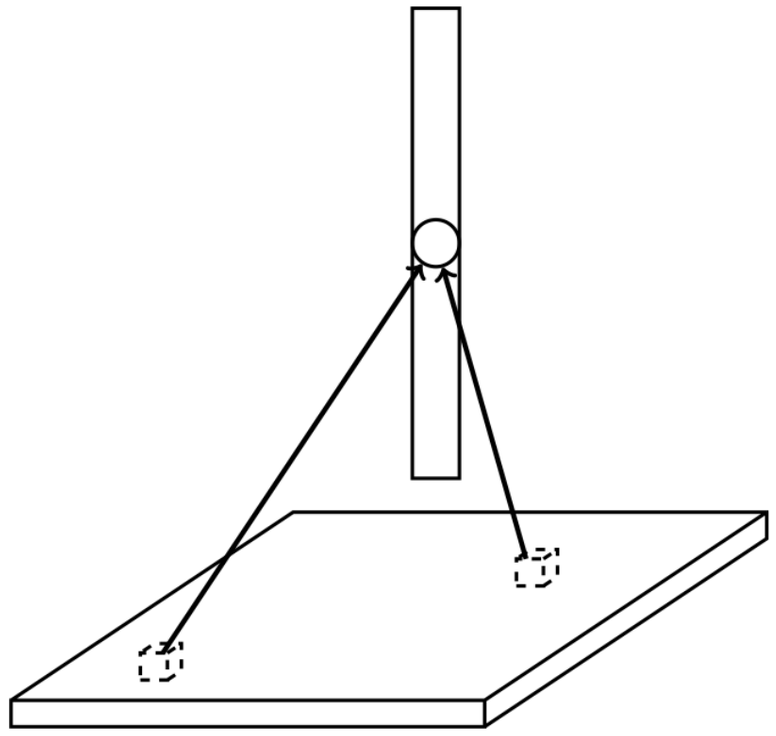


Each hidden unit looks at the entire image.

Convolution Layers

Fully connected layers:

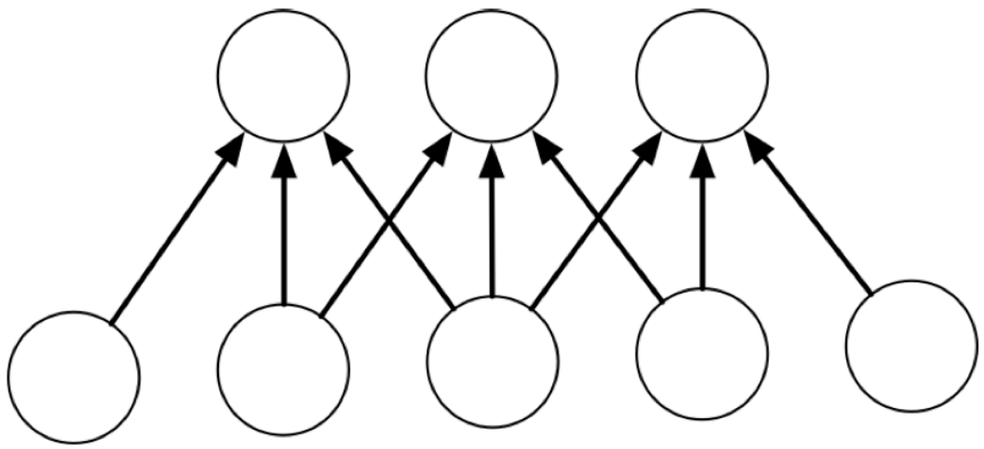


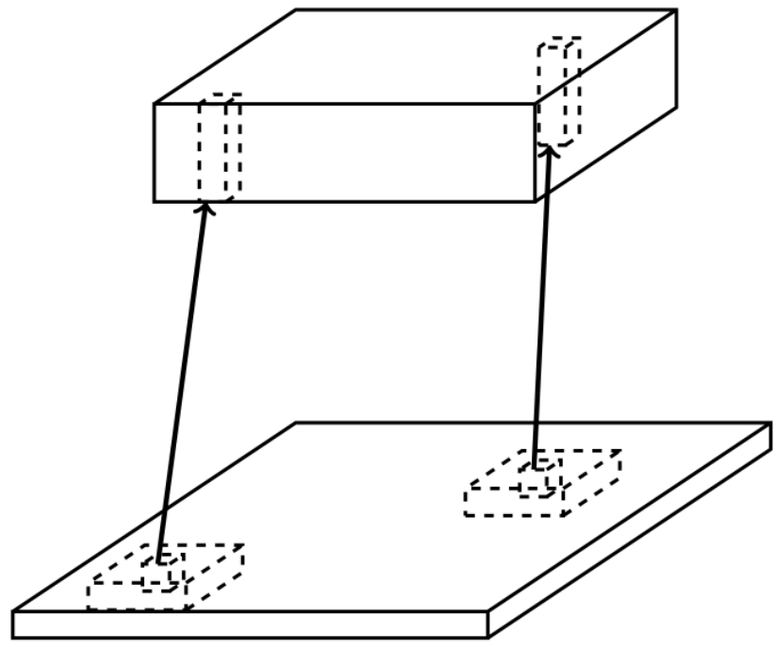


Each hidden unit looks at the entire image.

Convolution Layers

Locally connected layers:

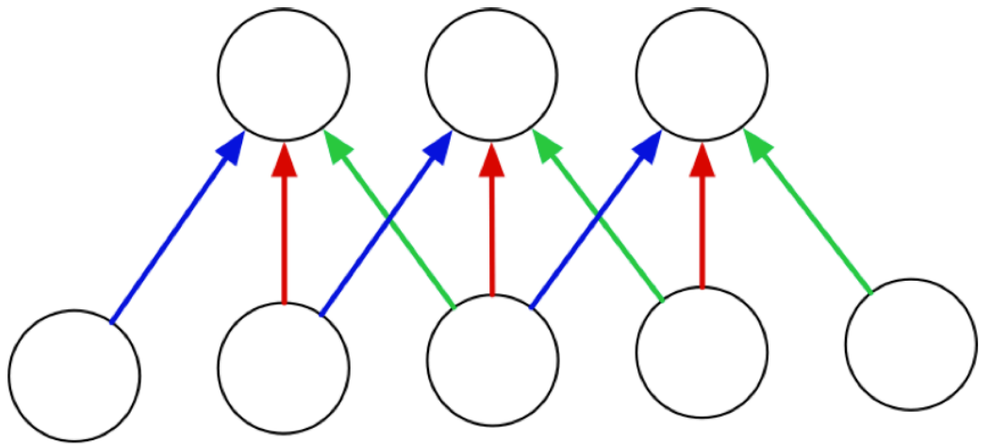


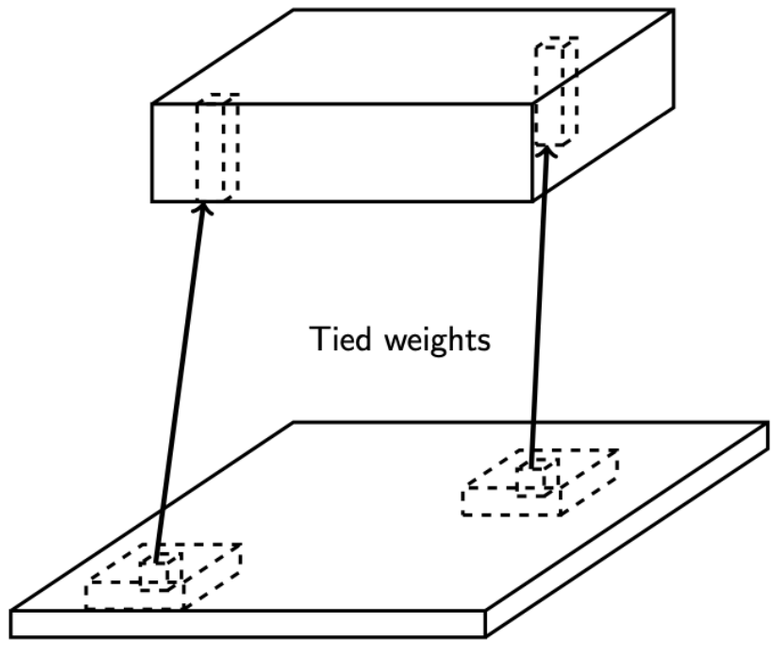


Each column of hidden units looks at a small region of the image.

Convolution Layers

Convolution layers:

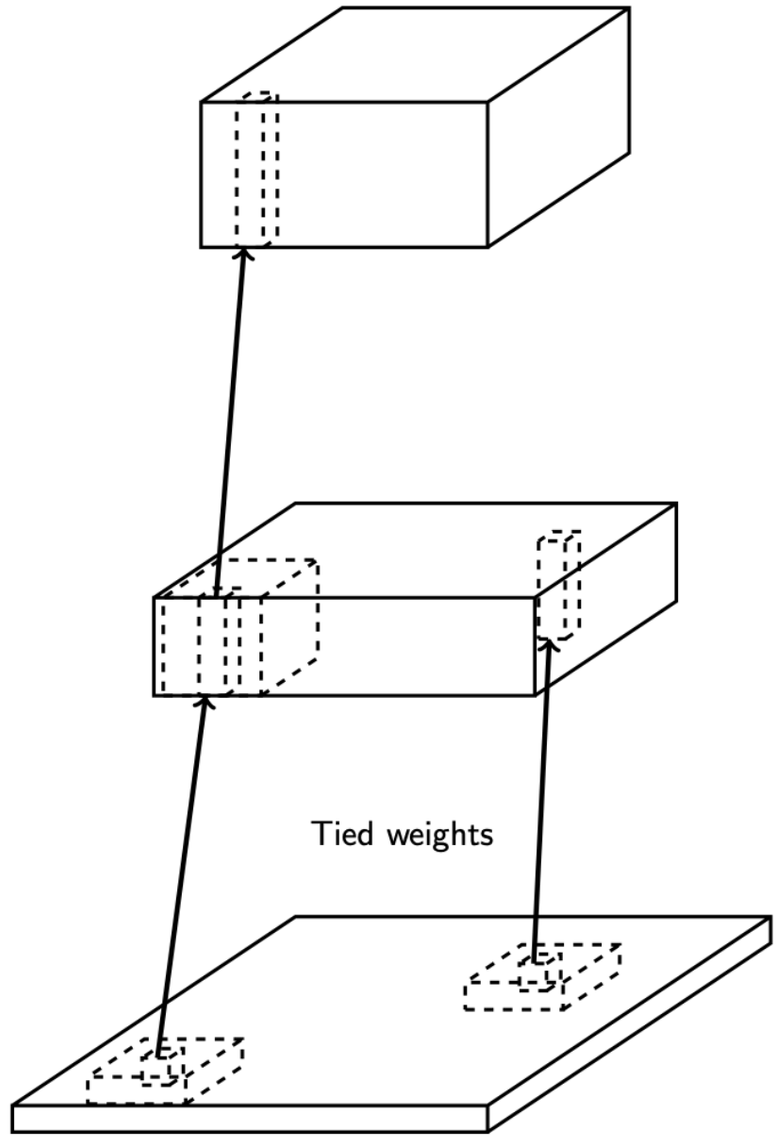




Each column of hidden units looks at a small region of the image, and the weights are shared between all image locations.

Going Deeply Convolutional

Convolution layers can be stacked:



Convolution

We've already been vectorizing our computations by expressing them in terms of matrix and vector operations.

Now we'll introduce a new high-level operation, convolution. Here the motivation isn't computational efficiency — we'll see more efficient ways to do the computations later. Rather, the motivation is to get some understanding of what convolution layers can do.

Convolution

We've already been vectorizing our computations by expressing them in terms of matrix and vector operations.

Now we'll introduce a new high-level operation, convolution. Here the motivation isn't computational efficiency — we'll see more efficient ways to do the computations later. Rather, the motivation is to get some understanding of what convolution layers can do.

Let's look at the 1-D case first. If a and b are two arrays,

$$(a * b)_t = \sum_\tau a_\tau b_{t-\tau}.$$

Note: indexing conventions are inconsistent. We'll explain them in each case.

Convolution

We've already been vectorizing our computations by expressing them in terms of matrix and vector operations.

Now we'll introduce a new high-level operation, convolution. Here the motivation isn't computational efficiency — we'll see more efficient ways to do the computations later. Rather, the motivation is to get some understanding of what convolution layers can do.

Let's look at the 1-D case first. If a and b are two arrays,

$$(a * b)_t = \sum_\tau a_\tau b_{t-\tau}.$$

Note: indexing conventions are inconsistent. We'll explain them in each case.

Convolution

We've already been vectorizing our computations by expressing them in terms of matrix and vector operations.

Now we'll introduce a new high-level operation, convolution. Here the motivation isn't computational efficiency — we'll see more efficient ways to do the computations later. Rather, the motivation is to get some understanding of what convolution layers can do.

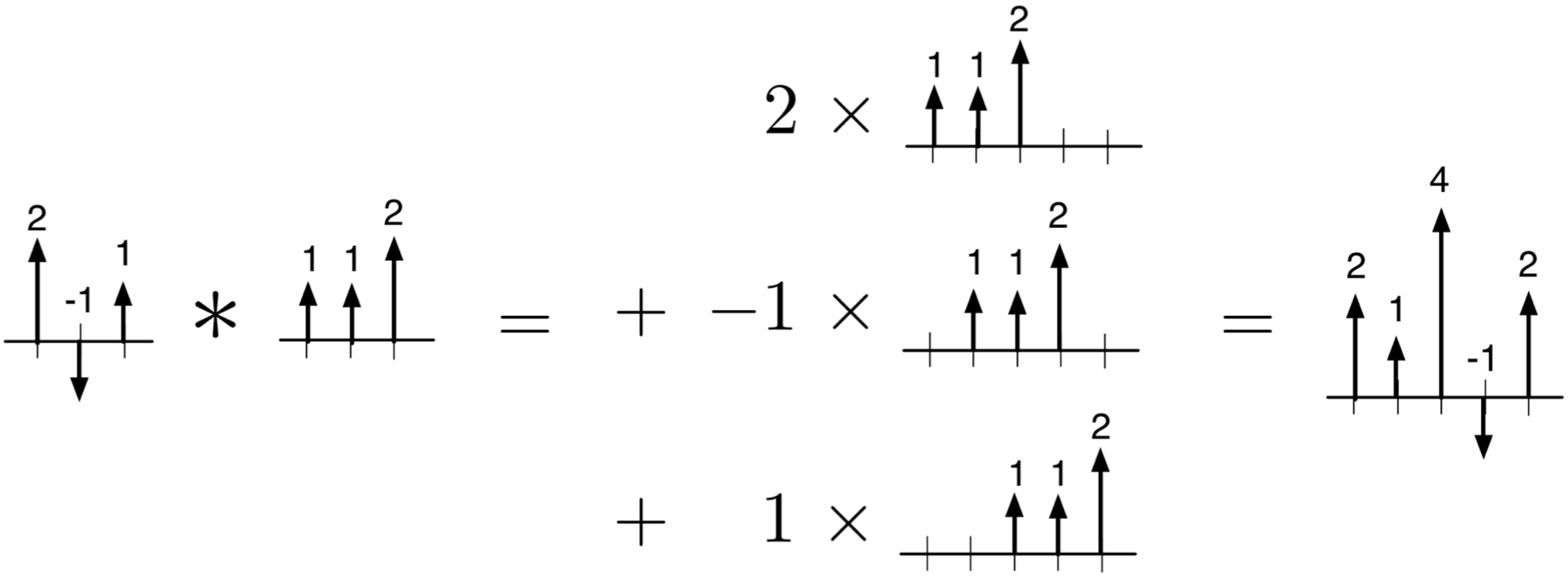
Let's look at the 1-D case first. If a and b are two arrays,

$$(a * b)_t = \sum_\tau a_\tau b_{t-\tau}.$$

Note: indexing conventions are inconsistent. We'll explain them in each case.

Convolution

Method 1: translate-and-scale



Convolution

Convolution can also be viewed as matrix multiplication:

$$(2,-1,1) * (1,1,2) = \begin{pmatrix} 1 & & \\ 1 & 1 & \\ 2 & 1 & 1 \\ & 2 & 1 \\ & & 2 \end{pmatrix} \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}$$

Aside: This is how convolution is typically implemented. (More efficient than the fast Fourier transform (FFT) for modern conv nets on GPUs!)

Convolution

Some properties of convolution:

- Commutativity

$$a * b = b * a$$

- Linearity

$$a * (\lambda_1 b + \lambda_2 c) = \lambda_1 a * b + \lambda_2 a * c$$

2-D Convolution

2-D convolution is defined analogously to 1-D convolution.

If A and B are two 2-D arrays, then:

$$(A * B)_{ij} = \sum_s \sum_t A_{st} B_{i-s,j-t}.$$

2-D Convolution

Method 1: Translate-and-Scale

$$1 \times \begin{array}{|c|c|c|c|}\hline 1 & 3 & 1 & \\ \hline 0 & -1 & 1 & \\ \hline 2 & 2 & -1 & \\ \hline & & & \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|}\hline 1 & 3 & 1 \\ \hline 0 & -1 & 1 \\ \hline 2 & 2 & -1 \\ \hline \end{array} * \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 0 & -1 \\ \hline \end{array} = + 2 \times \begin{array}{|c|c|c|c|}\hline & 1 & 3 & 1 \\ \hline & 0 & -1 & 1 \\ \hline & 2 & 2 & -1 \\ \hline & & & \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 1 & 5 & 7 & 2 \\ \hline 0 & -2 & -4 & 1 \\ \hline 2 & 6 & 4 & -3 \\ \hline 0 & -2 & -2 & 1 \\ \hline \end{array}$$

$$+ -1 \times \begin{array}{|c|c|c|c|}\hline & & & \\ \hline & 1 & 3 & 1 \\ \hline & 0 & -1 & 1 \\ \hline & 2 & 2 & -1 \\ \hline \end{array}$$

2-D Convolution

The thing we convolve by is called a kernel, or filter.

What does this convolution kernel do?

$*$

0	1	0
1	4	1
0	1	0

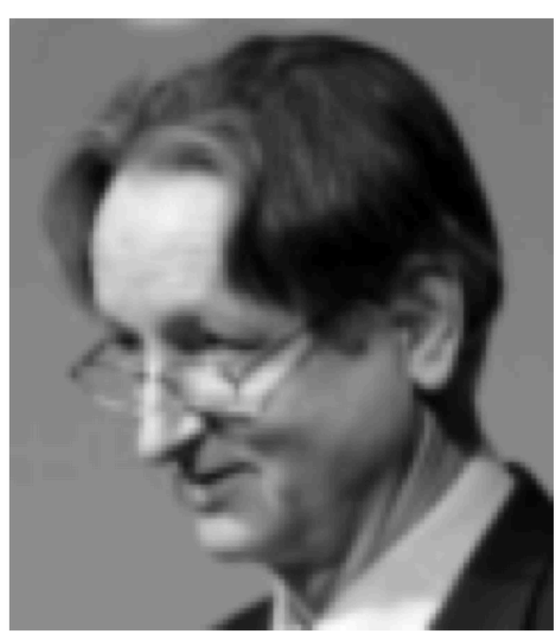
2-D Convolution

The thing we convolve by is called a kernel, or filter.

What does this convolution kernel do?

$*$

0	1	0
1	4	1
0	1	0



2-D Convolution

What does this convolution kernel do?

$*$

0	-1	0
-1	8	-1
0	-1	0

2-D Convolution

What does this convolution kernel do?

$*$

0	-1	0
-1	4	-1
0	-1	0

2-D Convolution

What does this convolution kernel do?

$*$

0	-1	0
-1	4	-1
0	-1	0

2-D Convolution

What does this convolution kernel do?

$*$

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

2-D Convolution

What does this convolution kernel do?

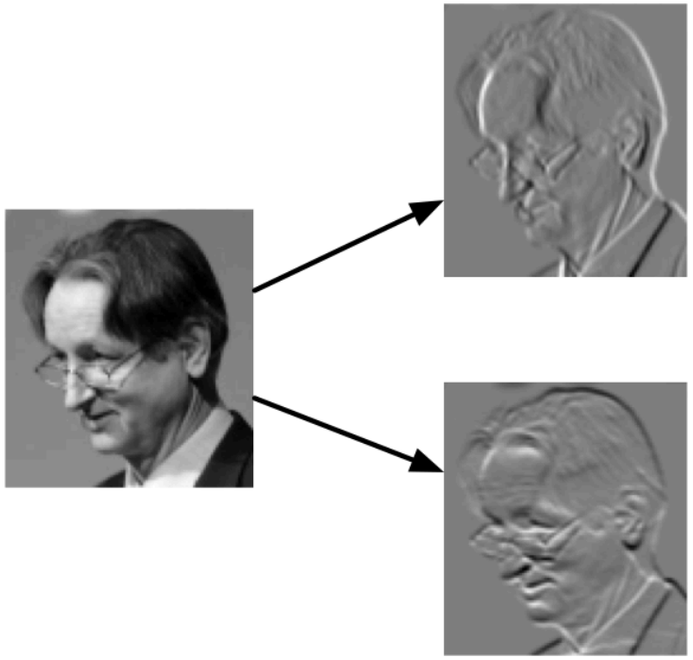
$*$

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

Convolutional networks

Let's finally turn to convolutional networks. These have two kinds of layers: detection layers (or convolution layers), and pooling layers.

The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.

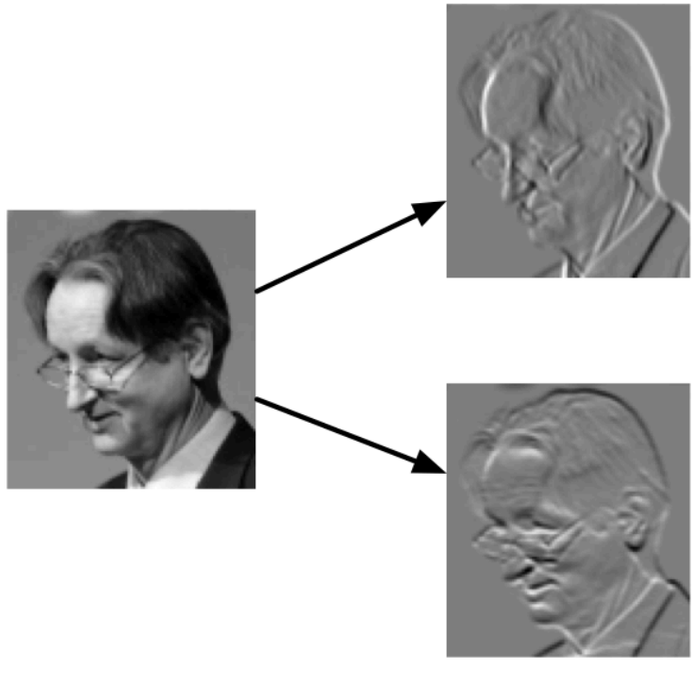


convolution

Convolutional networks

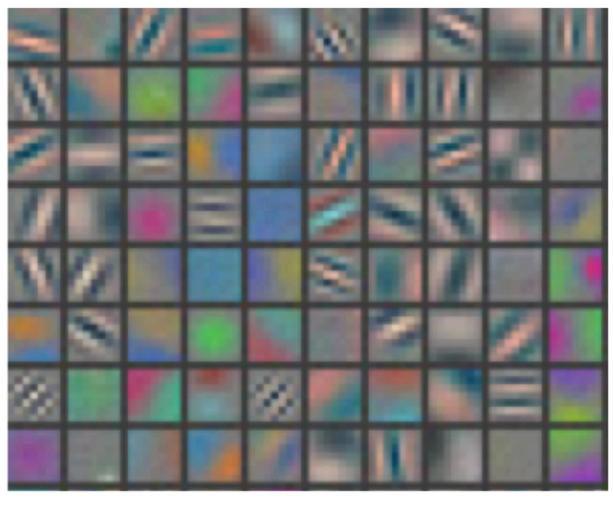
Let's finally turn to convolutional networks. These have two kinds of layers: detection layers (or convolution layers), and pooling layers.

The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.



convolution

Example first-layer filters

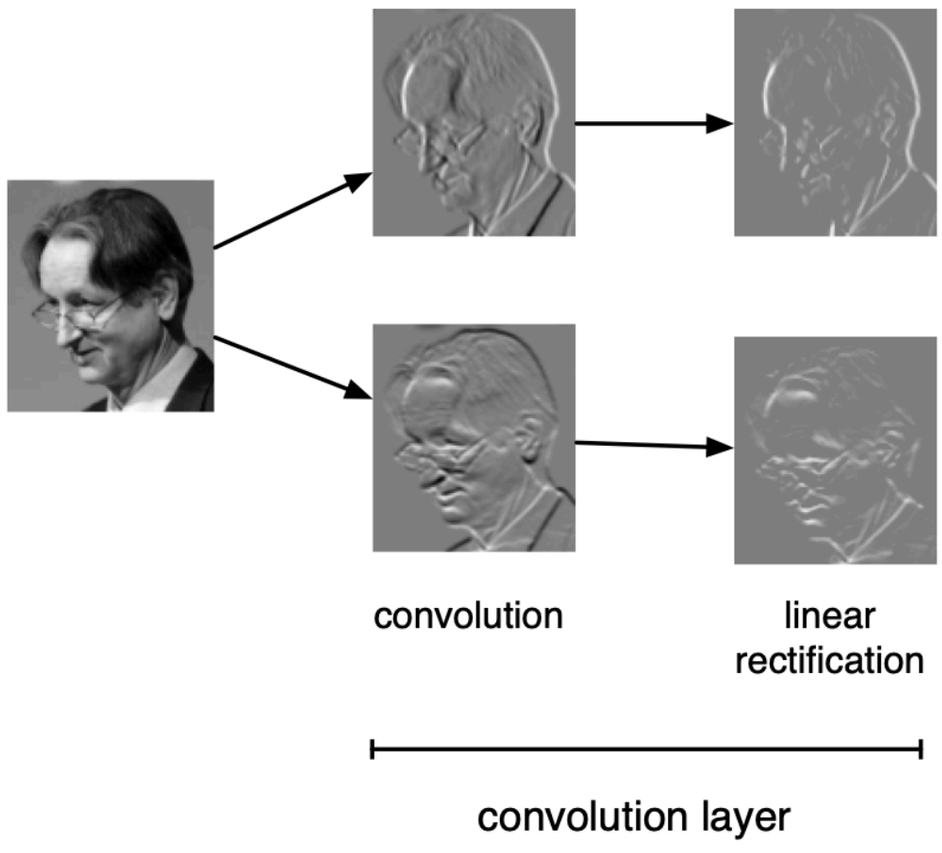


(Zeiler and Fergus, 2013, Visualizing and understanding convolutional networks)

Convolutional networks

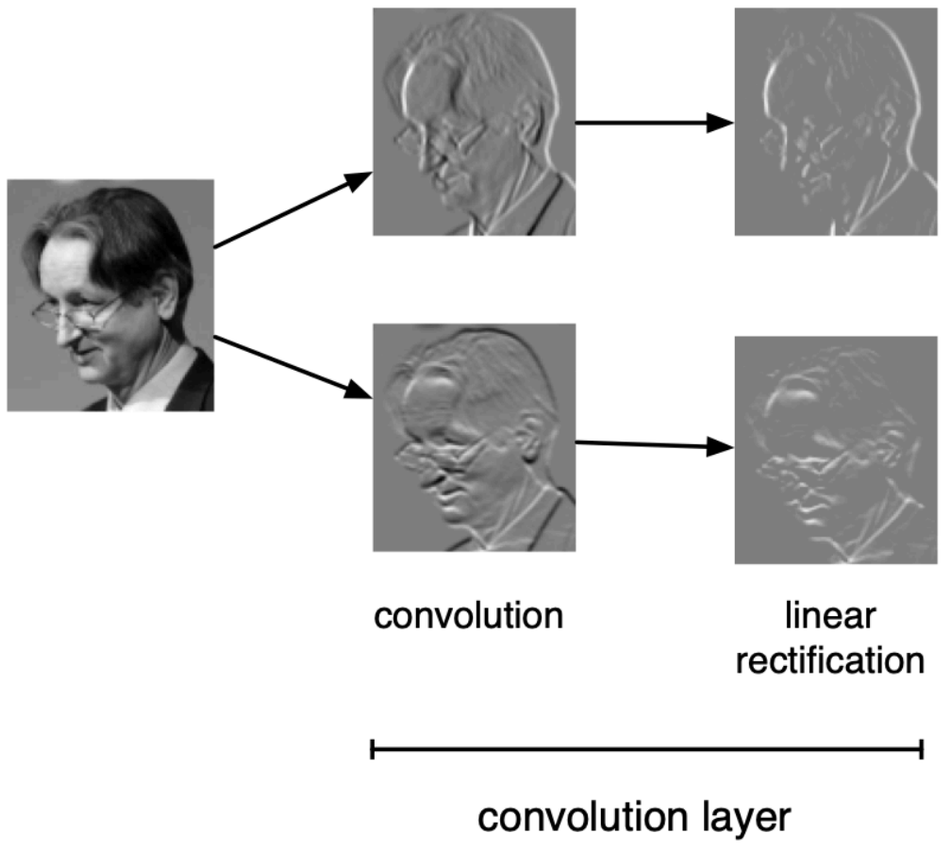
It's common to apply a linear rectification nonlinearity: $y_i = \max(z_i, 0)$

Why might we do this?



Convolutional networks

It's common to apply a linear rectification nonlinearity: $y_i = \max(z_i, 0)$

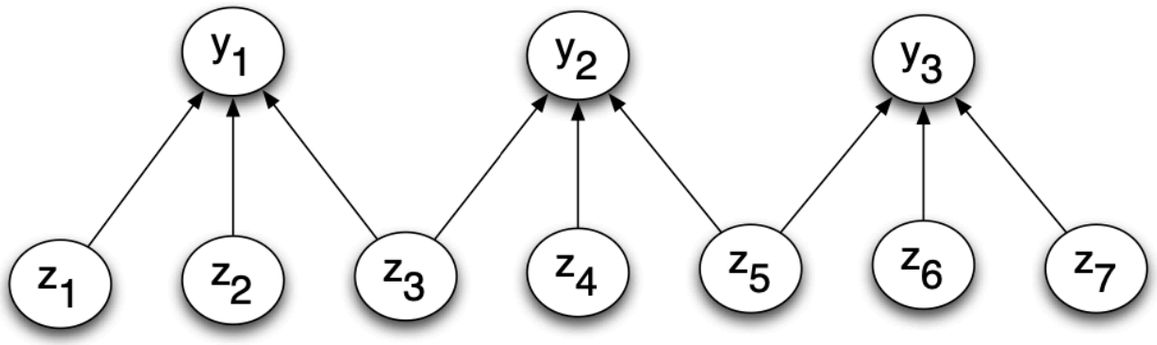


Why might we do this?

- Convolution is a linear operation. Therefore, we need a nonlinearity, otherwise 2 convolution layers would be no more powerful than 1.
- Two edges in opposite directions shouldn't cancel
- Makes the gradients sparse, which helps optimization (recall the backprop exercise from Lecture 6)

Pooling layers

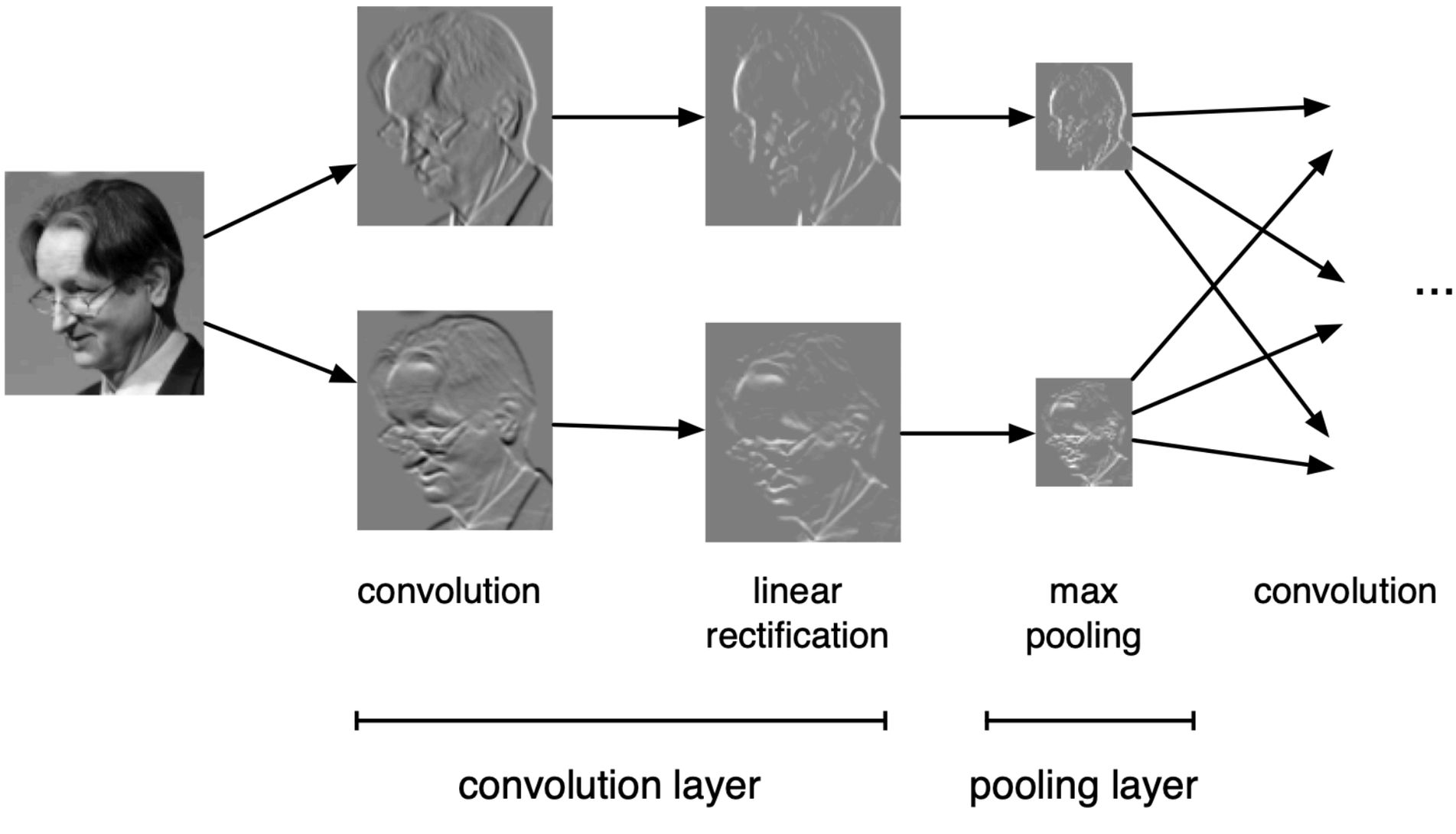
The other type of layer in a pooling layer. These layers reduce the size of the representation and build in invariance to small transformations.



Most commonly, we use max-pooling, which computes the maximum value of the units in a pooling group:

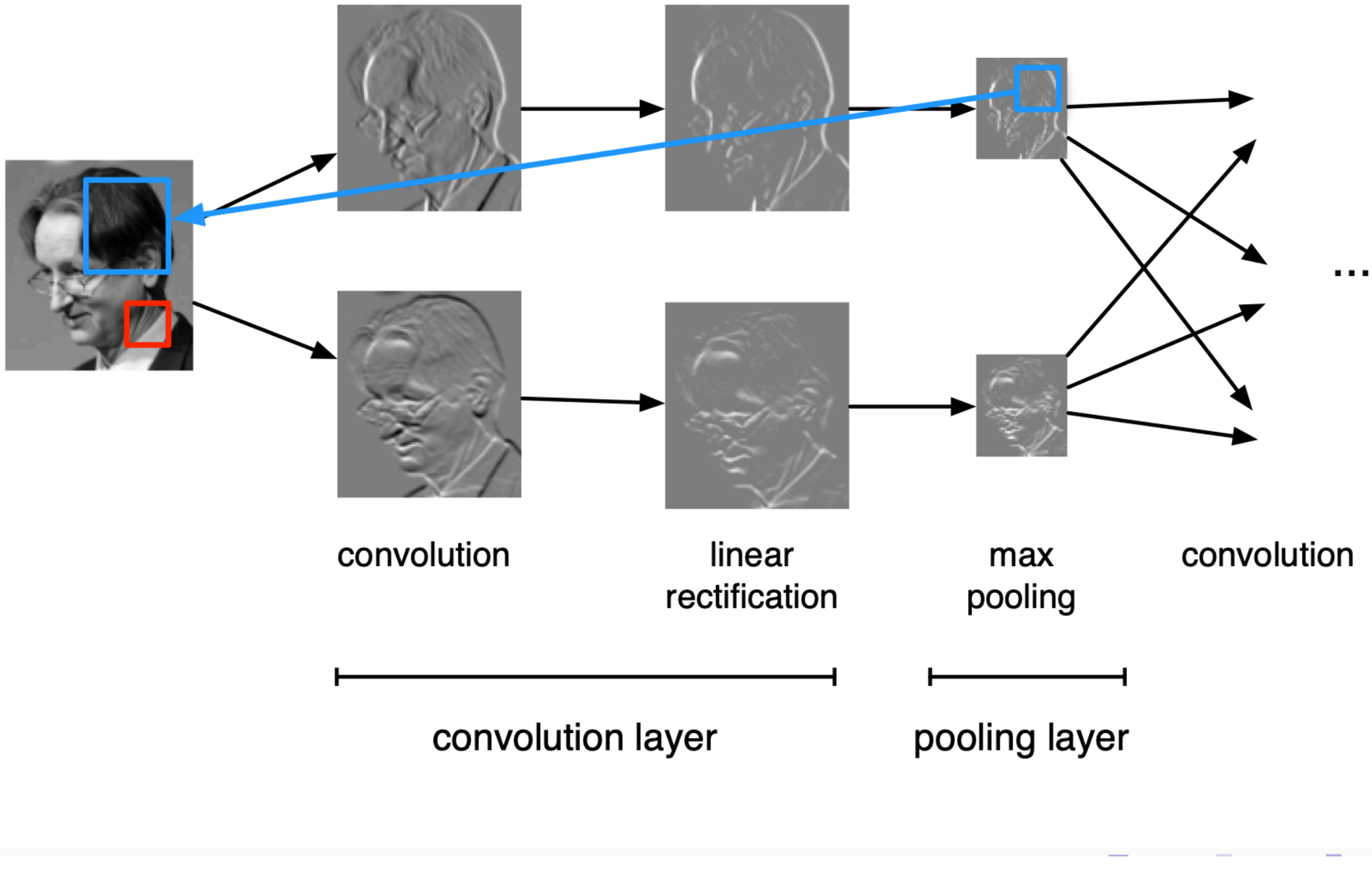
$$y_i = \max_{j \text{ in pooling group}} z_j$$

Convolutional networks



Convolutional networks

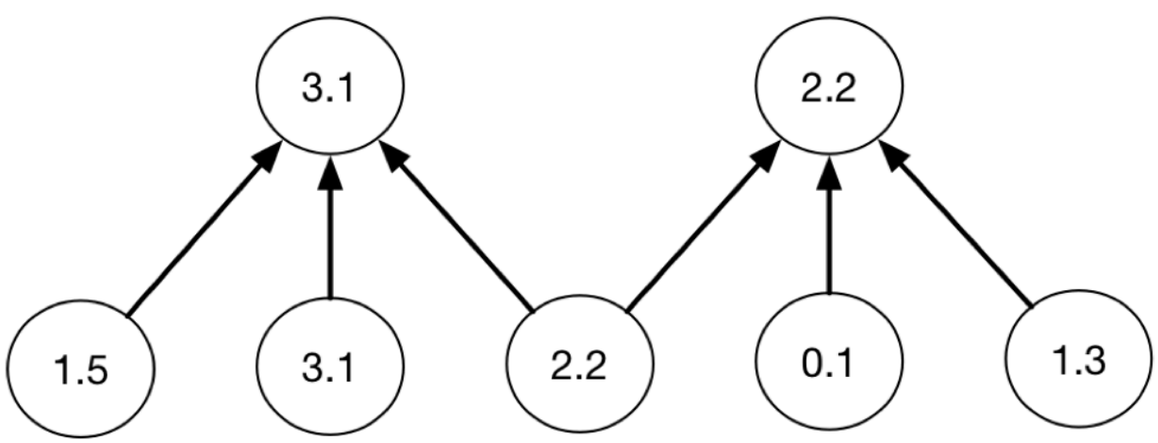
Because of pooling, higher-layer filters can cover a larger region of the input than equal-sized filters in the lower layers.

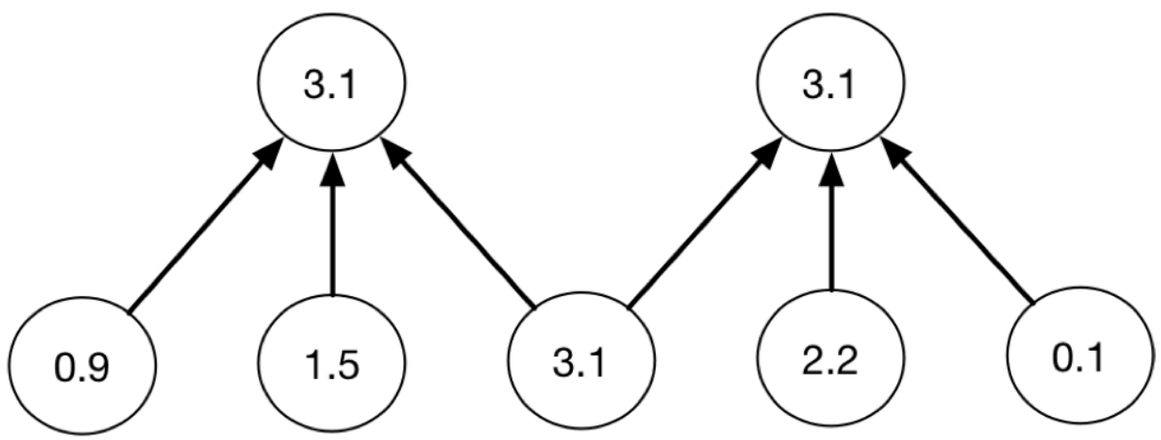


Equivariance and Invariance

We said the network's responses should be robust to translations of the input. But this can mean two different things.

- Convolution layers are equivariant: if you translate the inputs, the outputs are translated by the same amount.
- We'd like the network's predictions to be invariant: if you translate the inputs, the prediction should not change.
- Pooling layers provide invariance to small translations.

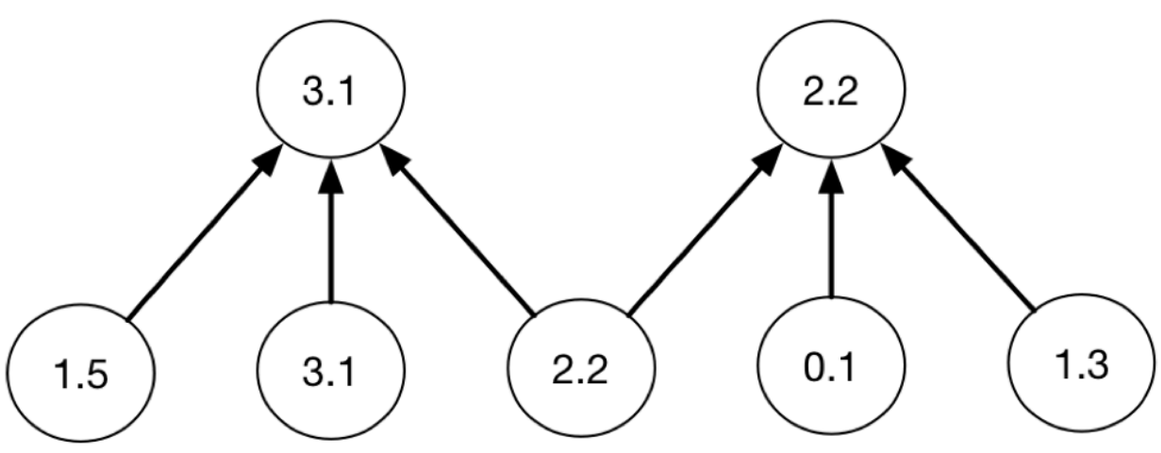


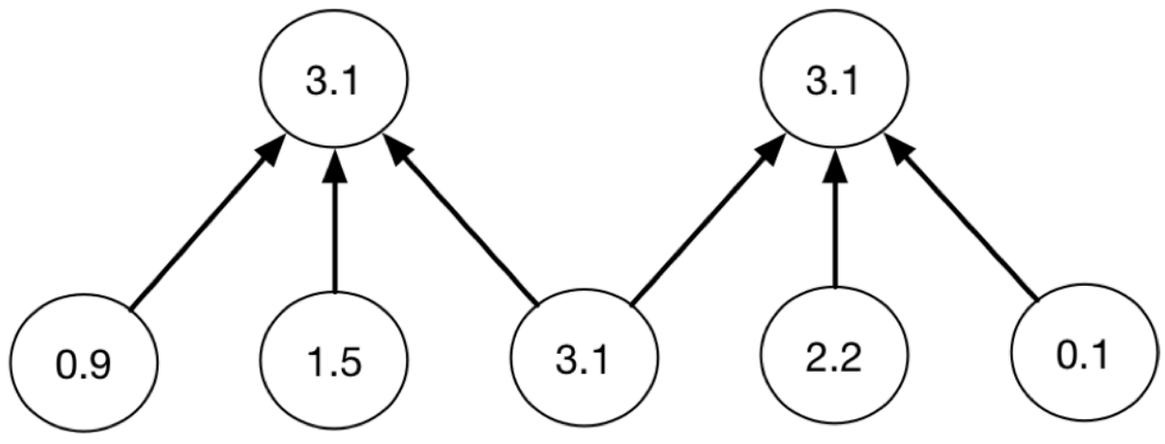


Equivariance and Invariance

We said the network's responses should be robust to translations of the input. But this can mean two different things.

- Convolution layers are equivariant: if you translate the inputs, the outputs are translated by the same amount.
- We'd like the network's predictions to be invariant: if you translate the inputs, the prediction should not change.
- Pooling layers provide invariance to small translations.





Convolution Layers

Each layer consists of several feature maps, each of which is an array. For the input layer, the feature maps are usually called channels.

- If the input layer represents a grayscale image, it consists of one channel. If it represents a color image, it consists of three channels.

Each unit is connected to each unit within its receptive field in the previous layer. This includes *all* of the previous layer's feature maps.

Convolution Layers

For simplicity, focus on 1-D signals (e.g. audio waveforms). Suppose the convolution layer's input has J feature maps and its output has I feature maps. Let t index the locations. Suppose the convolution kernels have radius R, i.e. dimension $K = 2R + 1$.

Each unit in a convolution layer receives inputs from all the units in its receptive field in the previous layer:

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

In terms of convolution,

$$\mathbf{y}_i = \sum_j \mathbf{x}_j * \text{flip}(\mathbf{w}_{i,j}).$$

Convolution Layers

For simplicity, focus on 1-D signals (e.g. audio waveforms). Suppose the convolution layer's input has J feature maps and its output has I feature maps. Let t index the locations. Suppose the convolution kernels have radius R, i.e. dimension $K = 2R + 1$.

Each unit in a convolution layer receives inputs from all the units in its receptive field in the previous layer:

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

Backprop Updates (Optional)

How do we train a conv net? With backprop, of course!

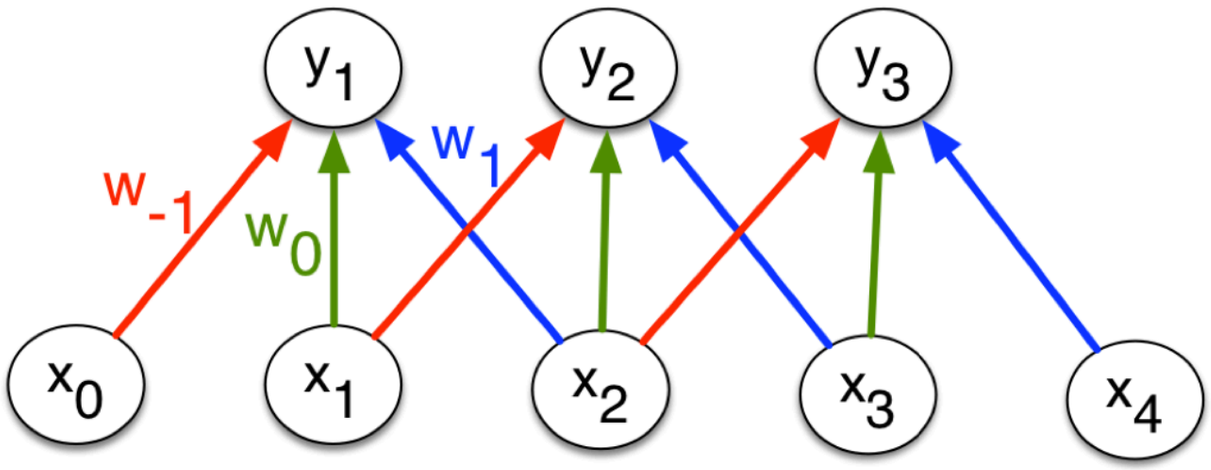
Recall what we need to do. Backprop is a message passing procedure, where each layer knows how to pass messages backwards through the computation graph. Let's determine the updates for convolution layers.

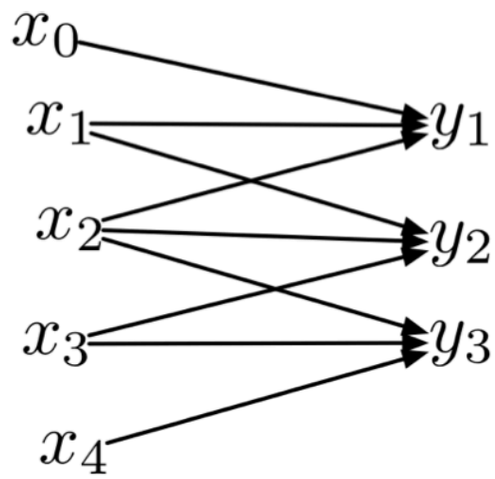
- We assume we are given the loss derivatives $\overline{y_{i,t}}$ with respect to the output units.
- We need to compute the cost derivatives with respect to the input units and with respect to the weights.

The only new feature is: how do we do backprop with tied weights?

Backprop Updates (Optional)

Consider the computation graph for the inputs:





Each input unit influences all the output units that have it within their receptive fields. Using the multivariate Chain Rule, we need to sum together the derivative terms for all these edges.

Backprop Updates (Optional)

Recall the formula for the convolution layer:

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

We compute the derivatives, which requires summing over all the outputs units which have the input unit in their receptive field:

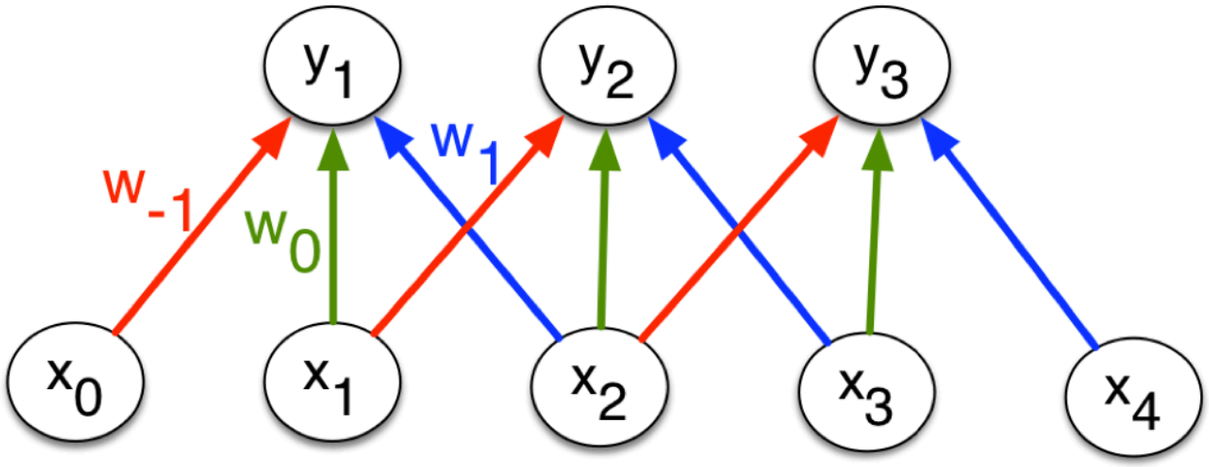
$$\begin{aligned}
\overline{x_{j,t}} &= \sum_{\tau} \overline{y_{i,t-\tau}} \frac{\partial y_{i,t-\tau}}{\partial x_{j,t}} \\
&= \sum_{\tau} \overline{y_{i,t-\tau}} \, w_{i,j,\tau}
\end{aligned}$$

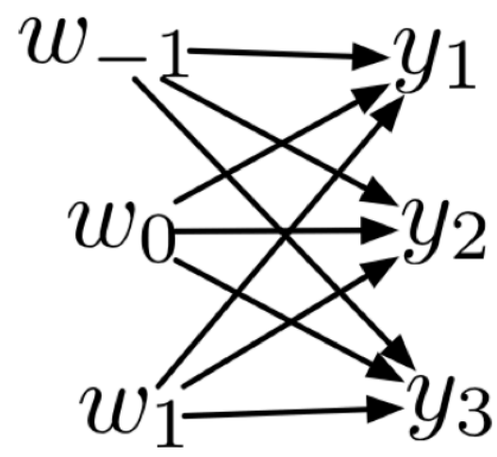
Written in terms of convolution,

$$\overline{\mathbf{x}_j} = \overline{\mathbf{y}_i} * \mathbf{w}_{i,j}.$$

Backprop Updates (Optional)

Consider the computation graph for the weights:





Each of the weights affects all the output units for the corresponding input and output feature maps.

Backprop Updates (Optional)

Recall the formula for the convolution layer:

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

We compute the derivatives, which requires summing over all spatial locations:

$$\begin{aligned} \overline{w_{i,j,\tau}} &= \sum_t \overline{y_{i,t}} \frac{\partial y_{i,t}}{\partial w_{i,j,\tau}} \\ &= \sum_t \overline{y_{i,t}} \, x_{j,t+\tau} \end{aligned}$$

Backprop Updates (Optional)

Recall the formula for the convolution layer:

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

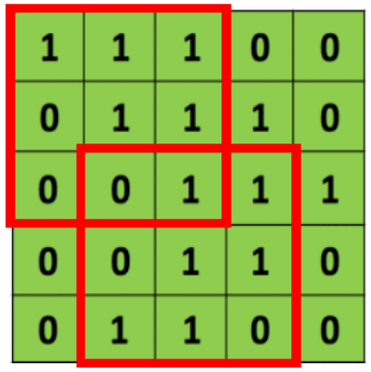
We compute the derivatives, which requires summing over all the outputs units which have the input unit in their receptive field:

$$\begin{aligned}
\overline{x_{j,t}} &= \sum_{\tau} \overline{y_{i,t-\tau}} \frac{\partial y_{i,t-\tau}}{\partial x_{j,t}} \\
&= \sum_{\tau} \overline{y_{i,t-\tau}} \, w_{i,j,\tau}
\end{aligned}$$

CNN examples

Convolution

Input Image
5×5



3×3 patch

Filter (Kernel)
3×3

Question: how many 3×3 patches does a 5×5 image have?

Answer: $(5-3+1)\times(5-3+1)=9.$

Convolution

Input Image
5×5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter (Kernel)
3×3

1	0	1
0	1	0
1	0	1

Convolution:

4	3	4
2	4	3
2	3	4

Result
3×3

The value 4 is the inner product of the patch

1	1	1
0	1	1
0	0	1

and the filter

1	0	1
0	1	0
1	0	1

.

Convolution

Input Image
5×5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter (Kernel)
3×3

1	0	1
0	1	0
1	0	1

Convolution:

4	3	4
2	4	3
2	3	4

Result
3×3

The value 3 is the inner product of the patch

1	1	0
1	1	1
0	1	1

and the filter

1	0	1
0	1	0
1	0	1

.

Convolution

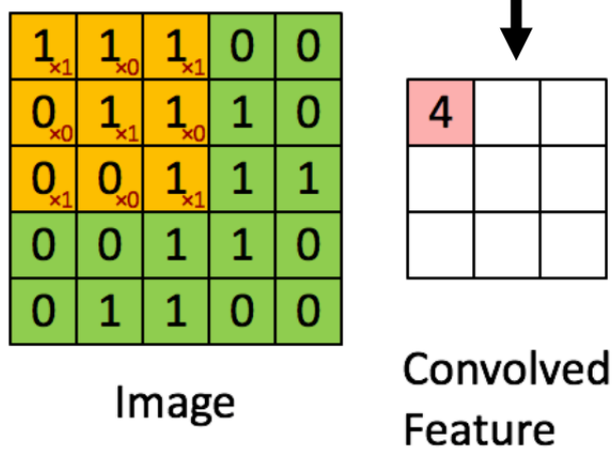
Input Image
5×5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter (Kernel)
3×3

1	0	1
0	1	0
1	0	1

Convolution:



Result
3×3

Convolution

Input Image
5×5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter (Kernel)
3×3

1	0	1
0	1	0
1	0	1

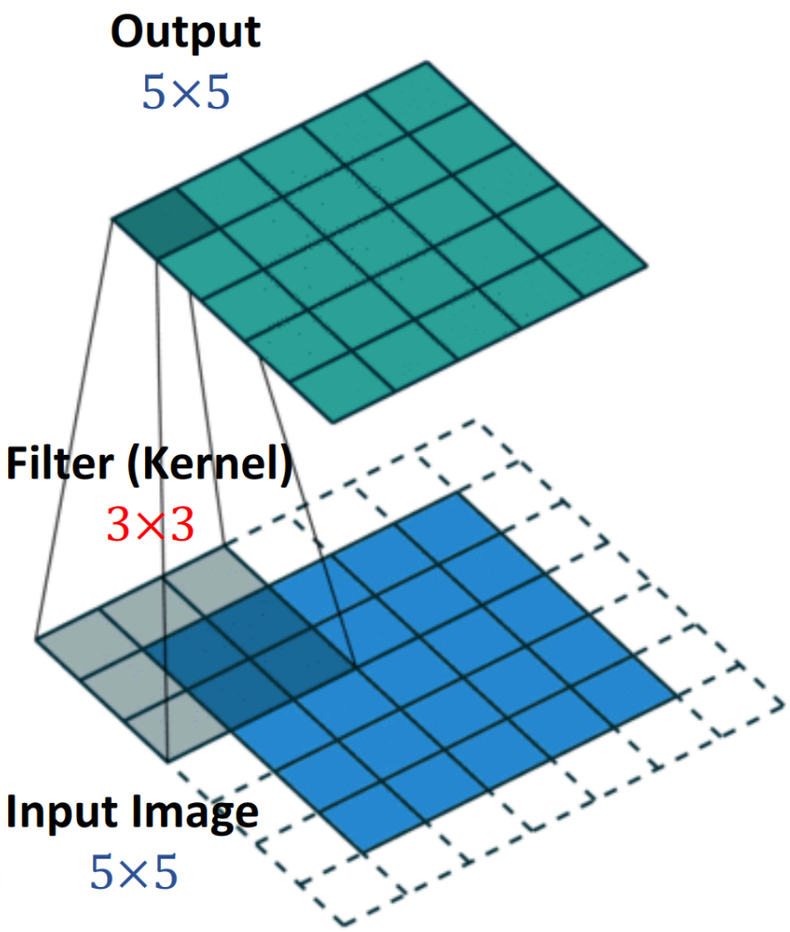
Convolution:

4	3	4
2	4	3
2	3	4

Dimensions:

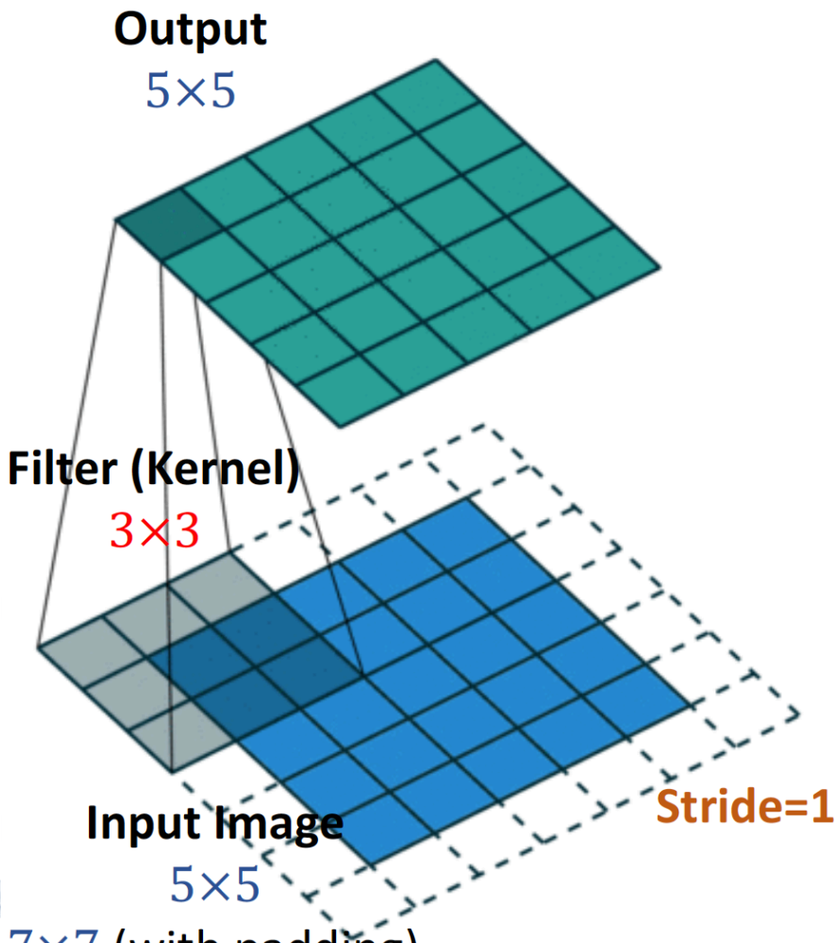
- Input: $d_1 \times d_2$
- Filter: $k_1 \times k_2$
- Output: $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1)$

Convolution: Zero Padding



- Add a “boarder” of all-zeros.
- Increase the input shape:
  - From $d_1 \times d_2$ to $(d_1+2)\times(d_2+2)$.
- If the filter is 3×3, the output is $d_1 \times d_2$.

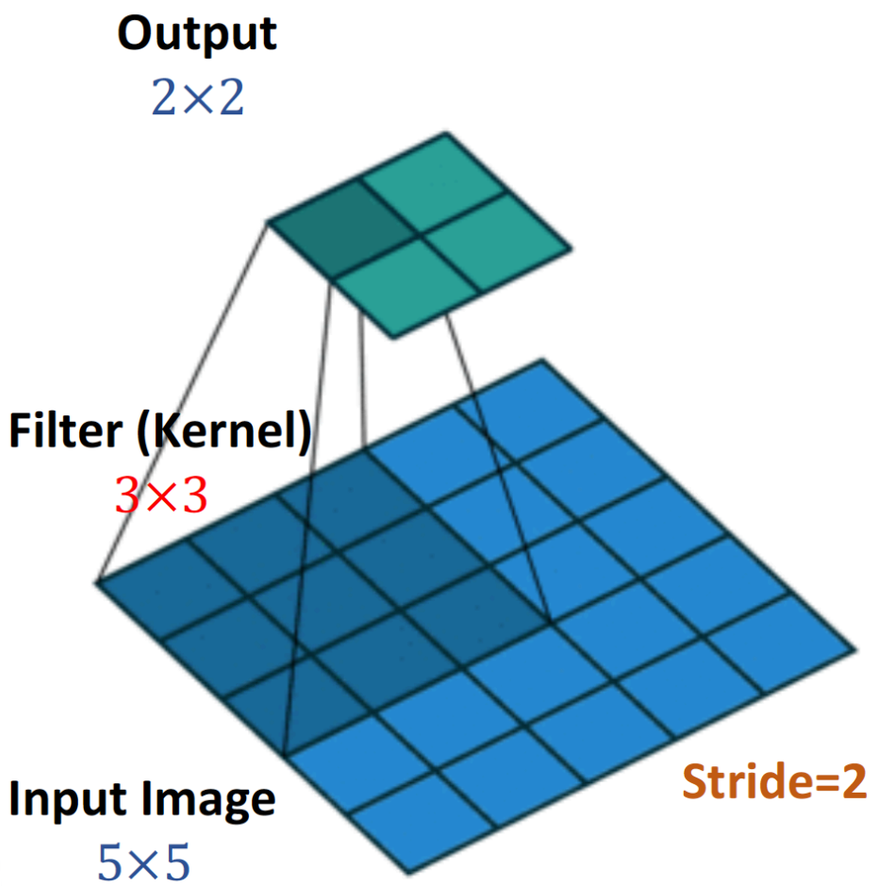
Convolution: Stride



7×7 (with padding)

- In the previous examples, the stride is 1.
  - The filter moves 1 step each time.

Convolution: Stride



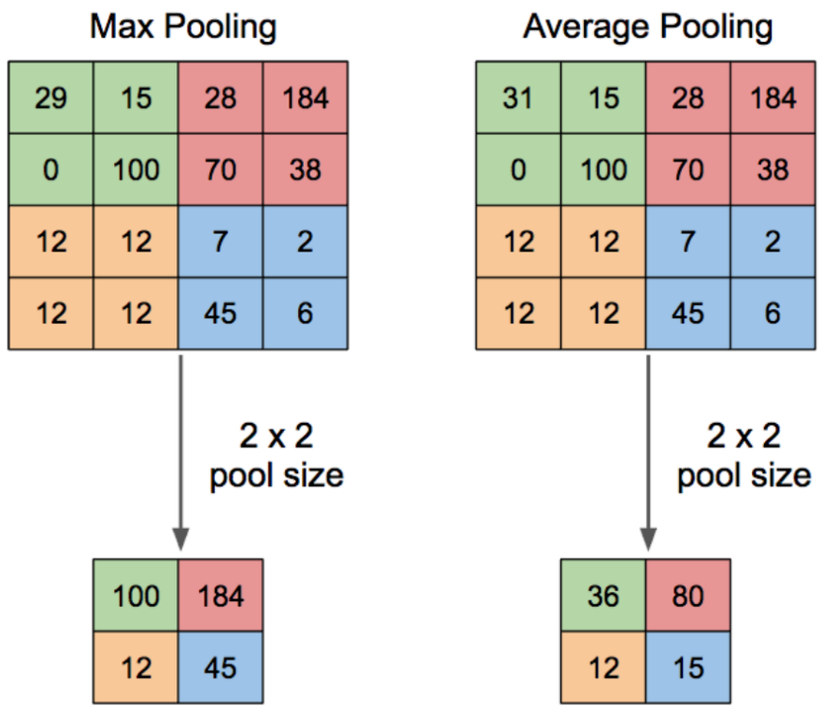
- In the previous examples, the stride is 1.
  - The filter moves 1 step each time.
- The stride can be 2 or even larger.

Dimensions:

- Input: $d_1 \times d_2$
- Filter: $k_1 \times k_2$
- Stride: s
- Output: $\left(\left\lfloor \frac{d_1 - k_1}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{d_2 - k_2}{s} \right\rfloor + 1\right)$

Pooling

- Pooling reduces the dimensionality of each feature map.
- E.g., Max Pooling, Average Pooling, etc.



Settings:

- Average Pooling.
- Pool size $= 2\times2$.
- No overlap (stride $= 2\times2$).

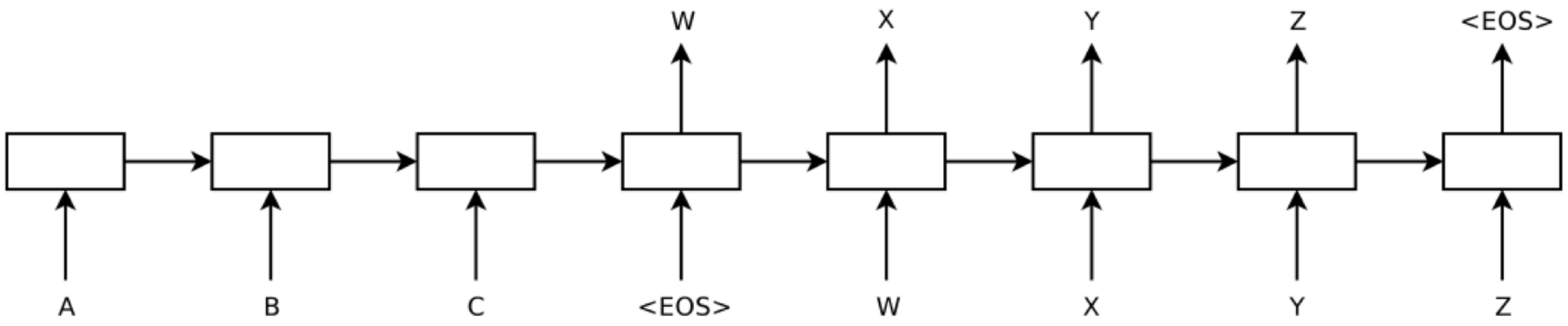
Hyper-Parameters for Network Structure

- Convolutional layers
  - # Filters
  - Filter shape
  - Stride
  - Zero-padding
- Pooling layers
  - MaxPool or AvgPool
  - Pool size
  - Pool stride
- Fully-connected layers
  - Width
- Activation functions
  - ReLU
  - SoftMax
  - Sigmoid
- Loss function.
  - Cross-entropy for classification
  - L1 or L2 for regression (the labels are continuous)
- Optimization algorithm (and its hyper-parameters, e.g., learning rate).
  - SGD
  - SGD with momentum
  - AdaGrad
  - RMSprop
- Random initialization.
  - Gaussian or uniform?
  - Scaling

Attention

Attention-Based Machine Translation

- Remember the encoder/decoder architecture for machine translation:



- The network reads a sentence and stores all the information in its hidden units.
- Some sentences can be really long. Can we really store all the information in a vector of hidden units?
  - Let's make things easier by letting the decoder refer to the input sentence.

Attention-Based Machine Translation

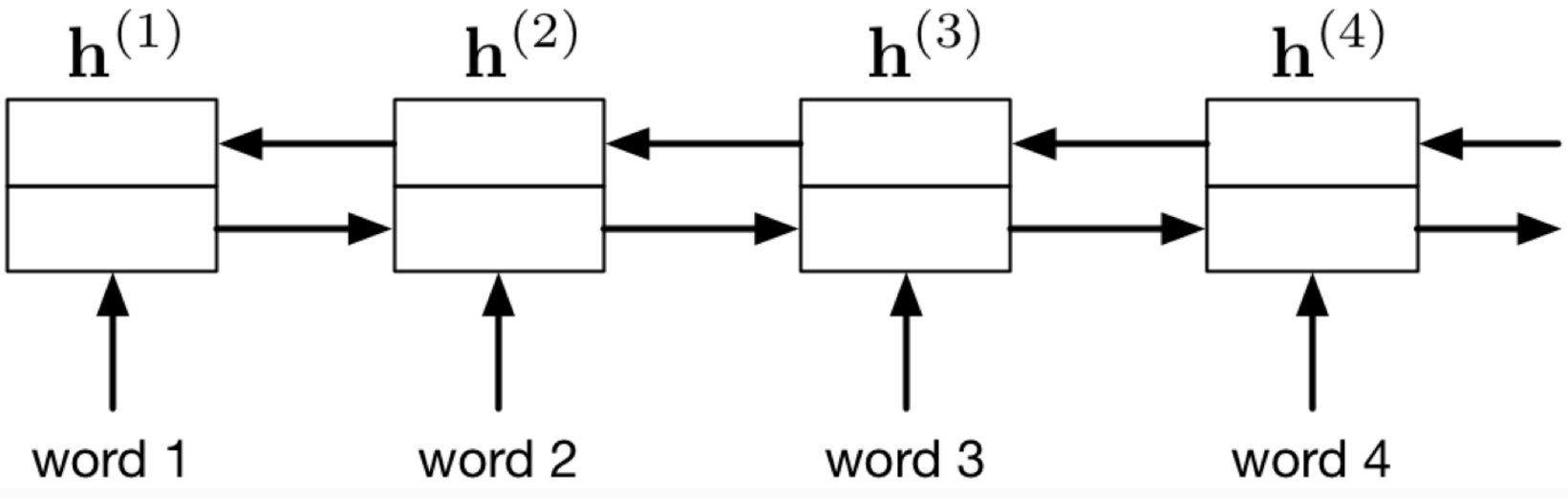
- We'll look at the translation model from the classic paper:

  Bahdanau et al., Neural machine translation by jointly learning to align and translate. ICLR, 2015.

- Basic idea: each output word comes from one word, or a handful of words, from the input. Maybe we can learn to attend to only the relevant ones as we produce the output.

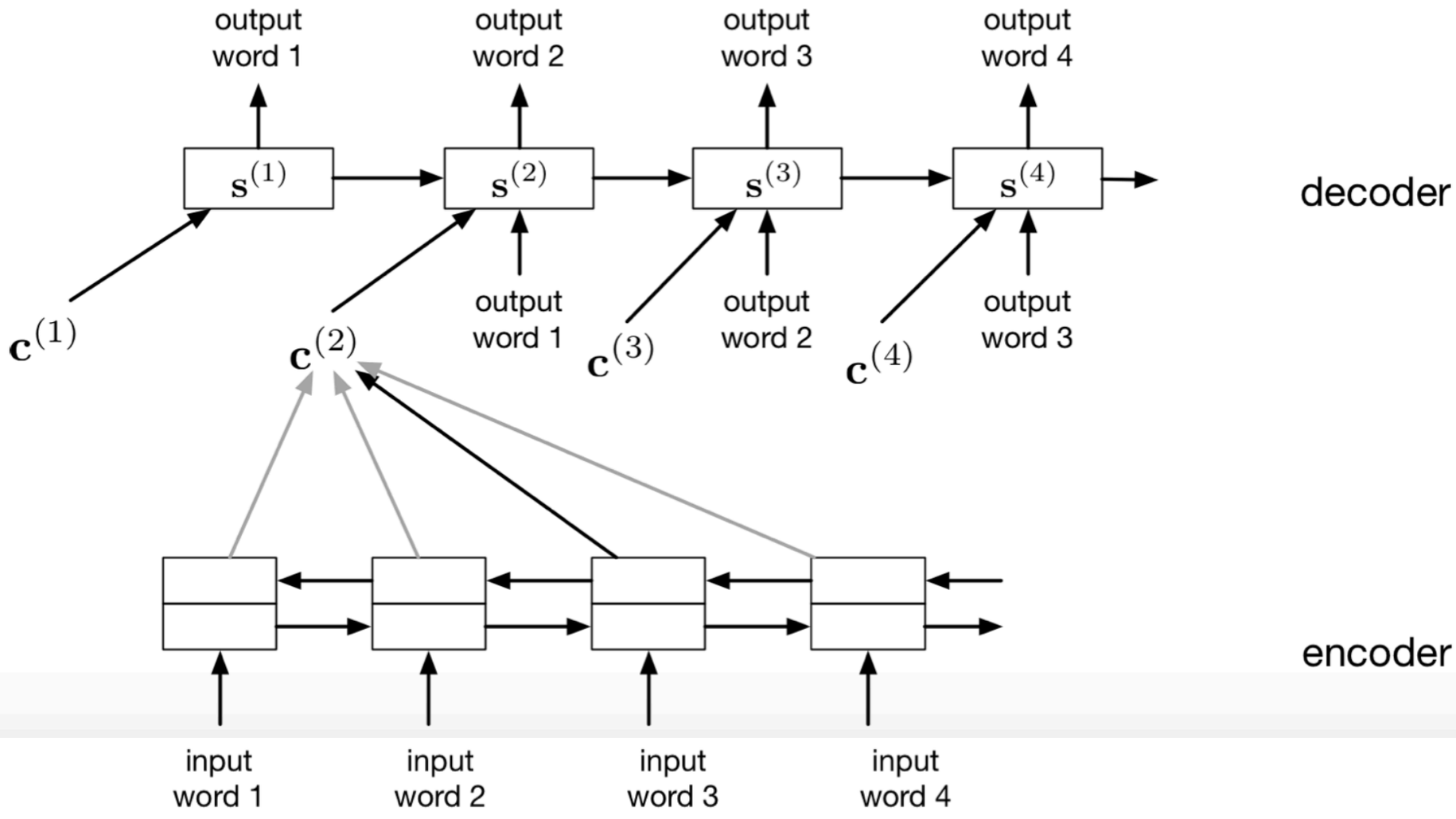
Attention-Based Machine Translation

- The model has both an encoder and a decoder. The encoder computes an annotation of each word in the input.
- It takes the form of a bidirectional RNN. This just means we have an RNN that runs forwards and an RNN that runs backwards, and we concantenate their hidden vectors.
    - The idea: information earlier or later in the sentence can help disambiguate a word, so we need both directions.
    - The RNN uses an LSTM-like architecture called gated recurrent units.



Attention-Based Machine Translation

- The decoder network is also an RNN. Like the encoder/decoder translation model, it makes predictions one word at a time, and its predictions are fed back in as inputs.
- The difference is that it also receives a context vector $\mathbf{c}^{(t)}$ at each time step, which is computed by attending to the inputs.



Attention-Based Machine Translation

- The context vector is computed as a weighted average of the encoder's annotations.

$$\mathbf{c}^{(i)} = \sum_j \alpha_{ij} h^{(j)}$$

- The attention weights are computed as a softmax, where the inputs depend on the annotation and the decoder's state:

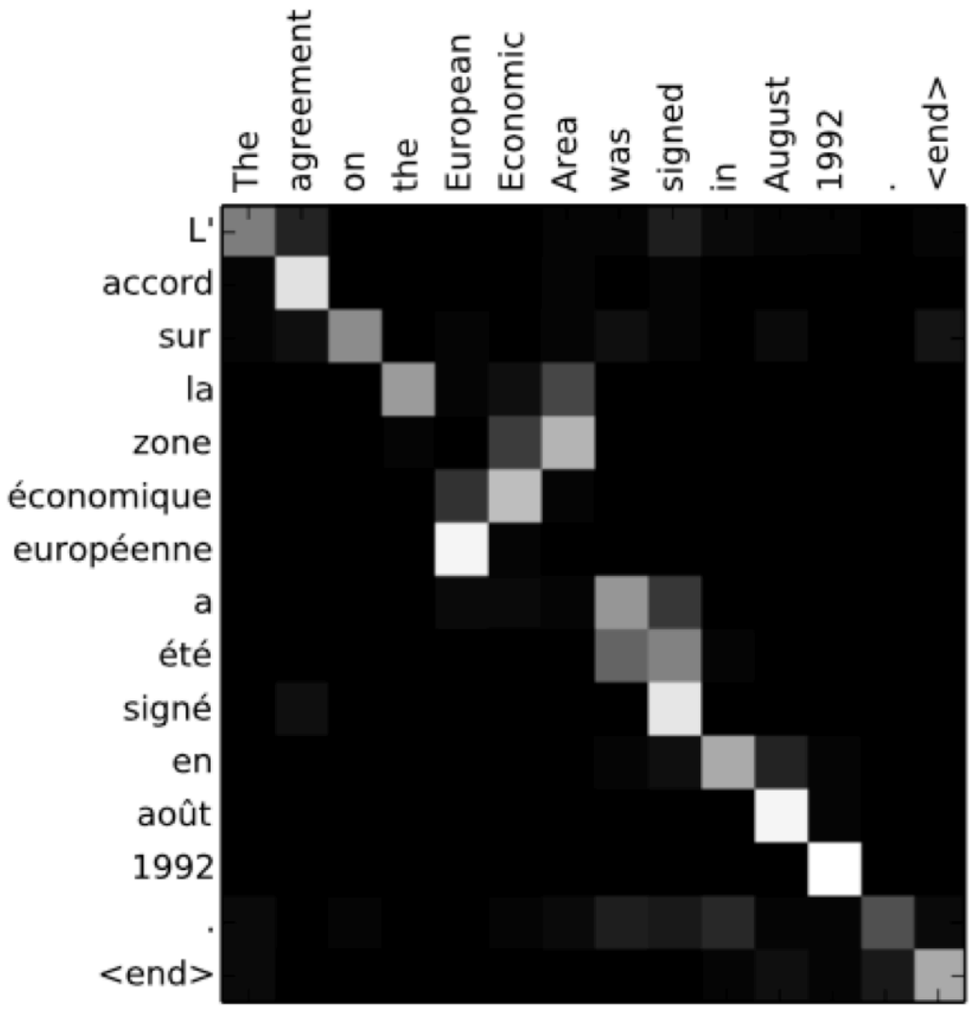
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

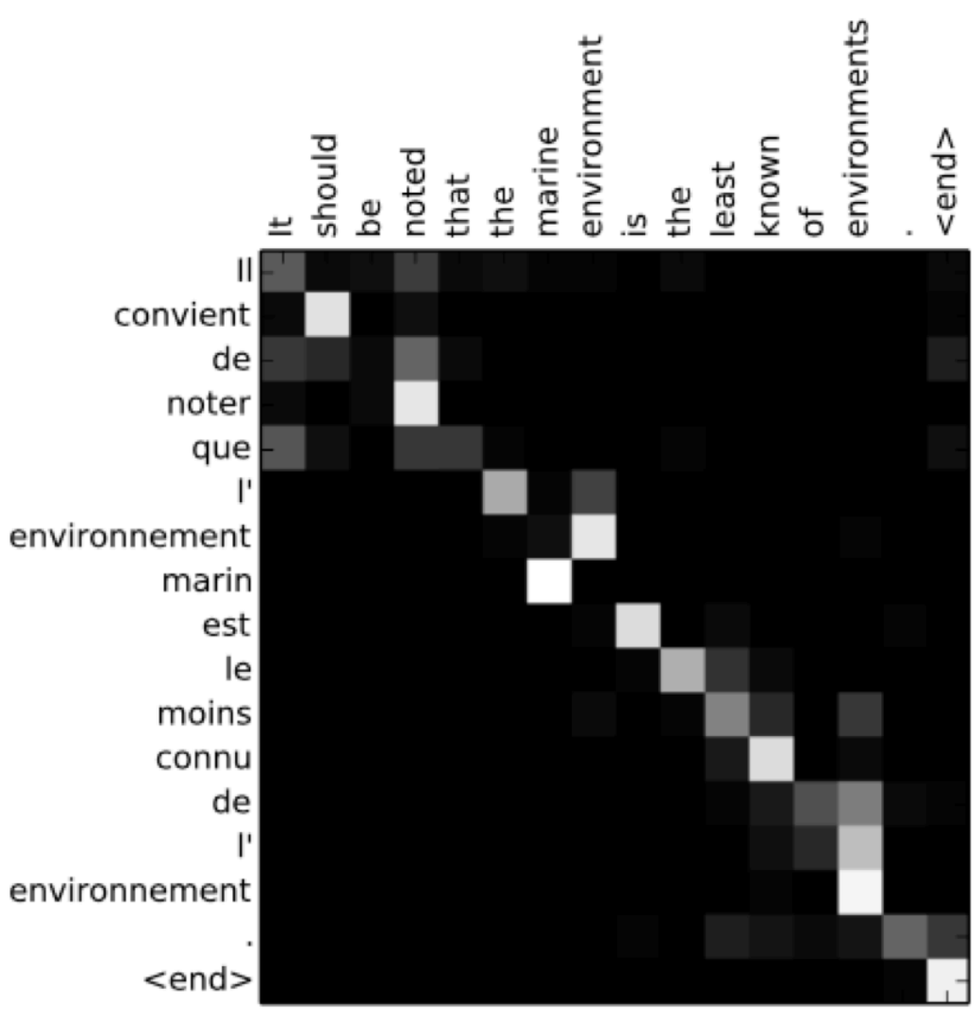
$$e_{ij} = a(\mathbf{s}^{(i-1)}, \mathbf{h}^{(j)})$$

- Note that the attention function depends on the annotation vector, rather than the position in the sentence. This means it's a form of content-based addressing.
  - My language model tells me the next word should be an adjective. Find me an adjective in the input.

Attention-Based Machine Translation

- Here's a visualization of the attention maps at each time step.

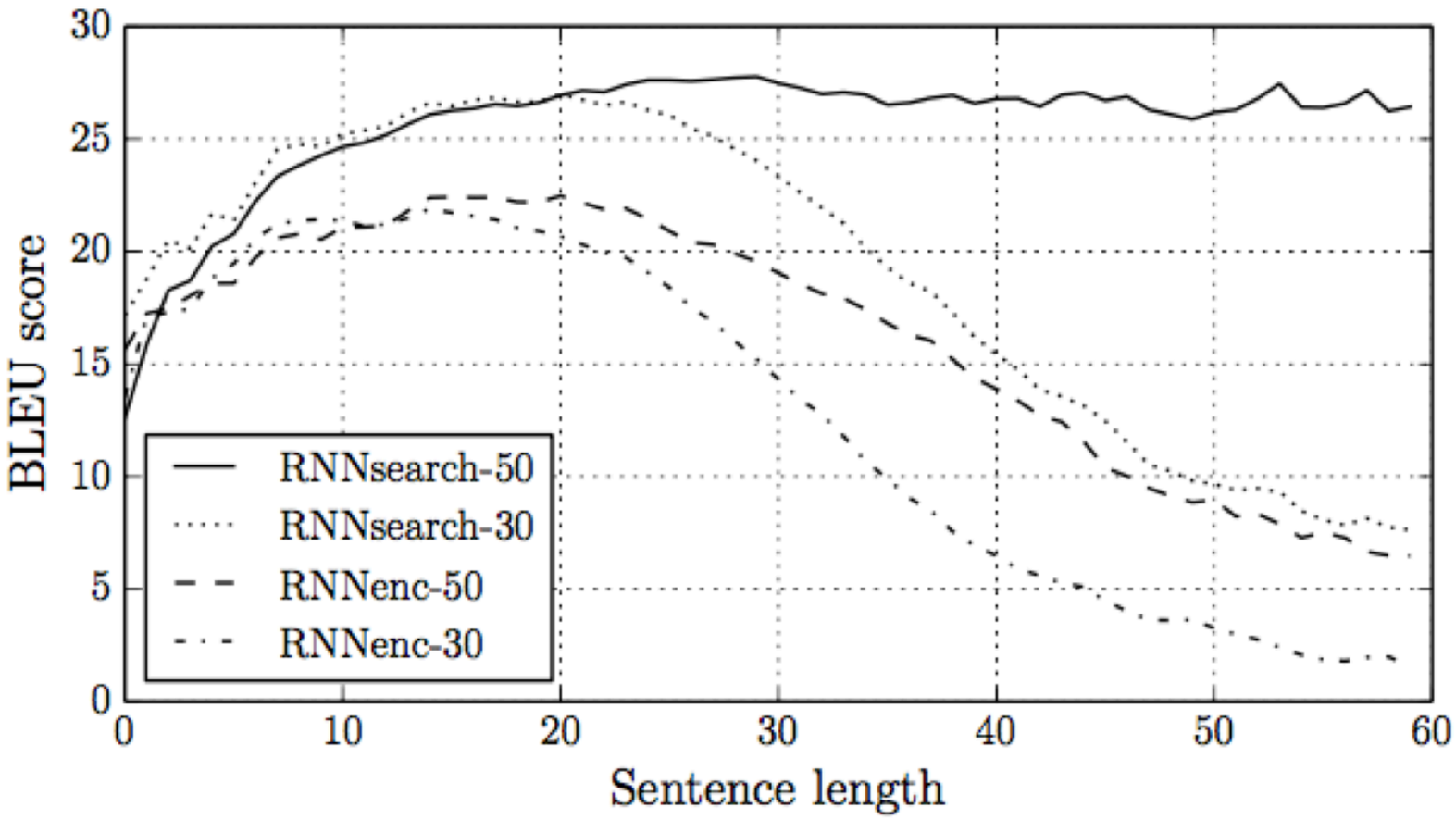




- Nothing forces the model to go linearly through the input sentence, but somehow it learns to do it.
  - It's not perfectly linear — e.g., French adjectives can come after the nouns.

Attention-Based Machine Translation

- The attention-based translation model does much better than the encoder/decoder model on long sentences.



Attention-Based Caption Generation

- Attention can also be used to understand images.
- We humans can't process a whole visual scene at once.
  - The fovea of the eye gives us high-acuity vision in only a tiny region of our field of view.
  - Instead, we must integrate information from a series of glimpses.
- The next few slides are based on this paper from the UofT machine learning group:

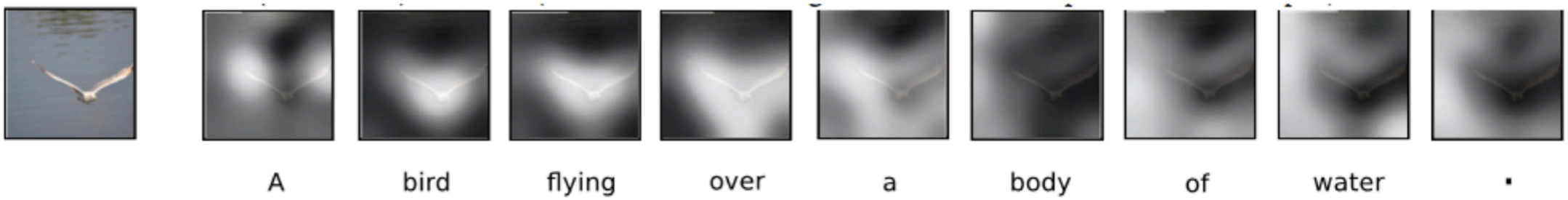
  Xu et al. Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention. ICML, 2015.

Attention-Based Caption Generation

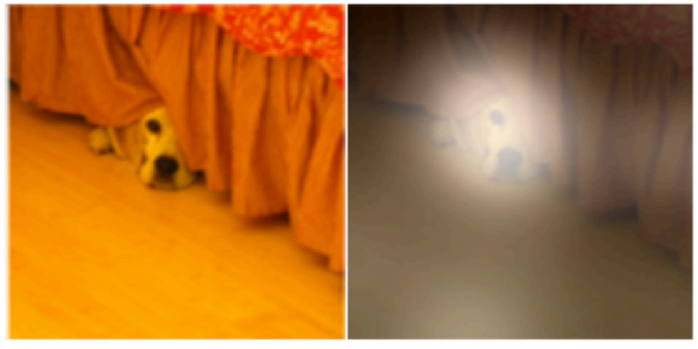
- The caption generation task: take an image as input, and produce a sentence describing the image.
- **Encoder:** a classification conv net (VGGNet, similar to AlexNet). This computes a bunch of feature maps over the image.
- **Decoder:** an attention-based RNN, analogous to the decoder in the translation model
  - In each time step, the decoder computes an attention map over the entire image, effectively deciding which regions to focus on.
  - It receives a context vector, which is the weighted average of the conv net features.

Attention-Based Caption Generation

- This lets us understand where the network is looking as it generates a sentence.

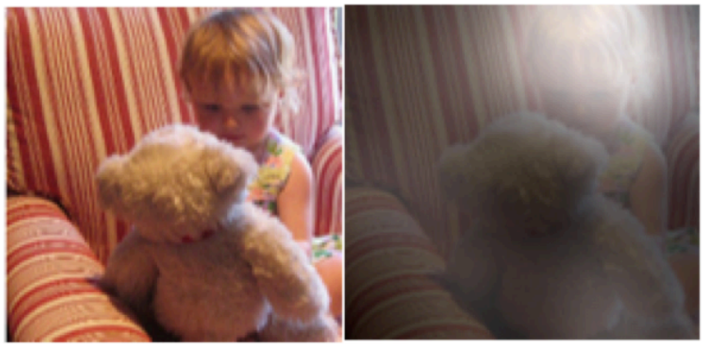


A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Attention-Based Caption Generation

- This can also help us understand the network's mistakes.

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and
a hat on a skateboard.

A person is standing on a beach
with a surfboard.

A woman is sitting at a table
with a large pizza.

A man is talking on his cell phone
while another man watches.

Computational Cost and Parallelism

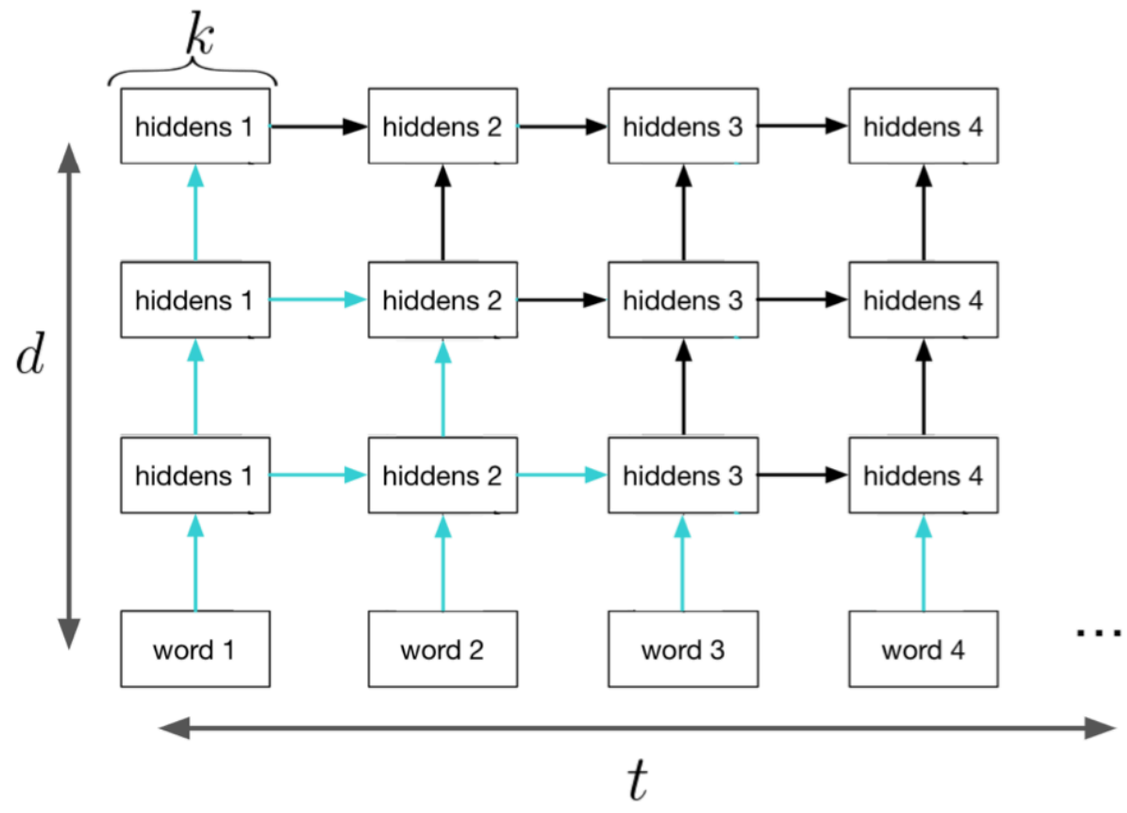
- There are a few things we should consider when designing an RNN.
- Computational cost:
  - **Number of connections.** How many add-multiply operations for the forward and backward pass.
  - **Number of time steps.** How many copies of hidden units to store for Backpropgation Through Time.
  - **Number of sequential operations.** The computations cannot be parallelized. (The part of the model that requires a for loop).
- **Maximum path length across time**: the shortest path length between the first encoder input and the last decoder output.
  - It tells us how easy it is for the RNN to remember / retreive information from the input sequence.

Computational Cost and Parallelism

- There are a few things we should consider when designing an RNN.
- Computational cost:
  - **Number of connections.** How many add-multiply operations for the forward and backward pass.
  - **Number of time steps.** How many copies of hidden units to store for Backpropgation Through Time.
  - **Number of sequential operations.** The computations cannot be parallelized. (The part of the model that requires a for loop).
- **Maximum path length across time**: the shortest path length between the first encoder input and the last decoder output.
  - It tells us how easy it is for the RNN to remember / retreive information from the input sequence.

Computational Cost and Parallelism

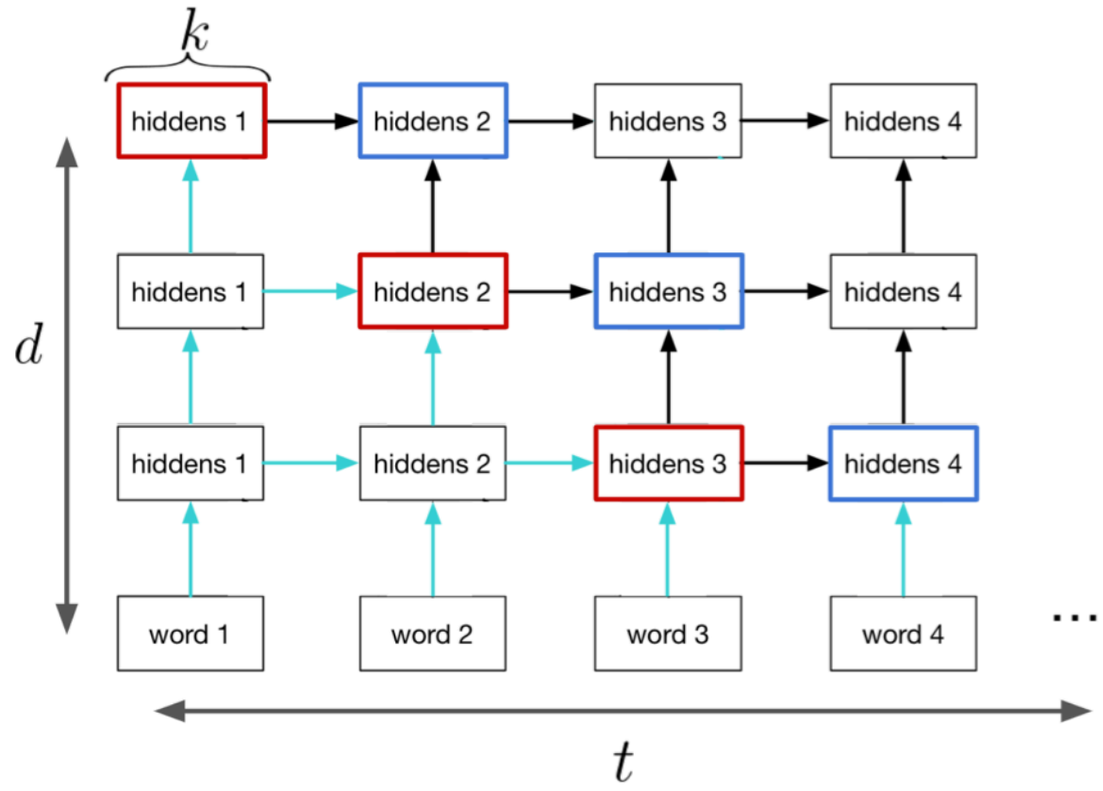
- Consider a standard d layer RNN with k hidden units, training on a sequence of length t.



- There are k^2 connections for each hidden-to-hidden connection. A total of $t \times k^2 \times d$ connections.
- We need to store all $t \times k \times d$ hidden units during training.
- Only $k \times d$ hidden units need to be stored at test time.

Computational Cost and Parallelism

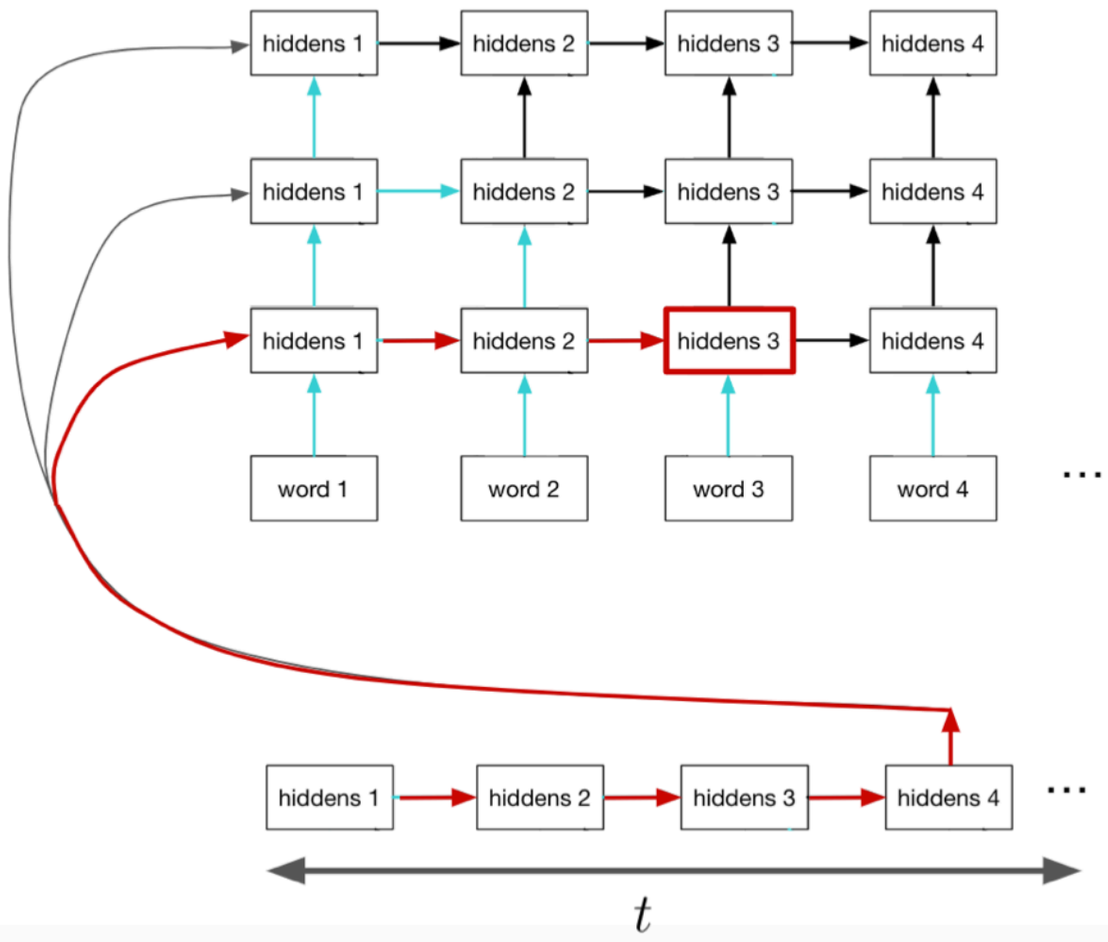
- Consider a standard d layer RNN with k hidden units, training on a sequence of length t.

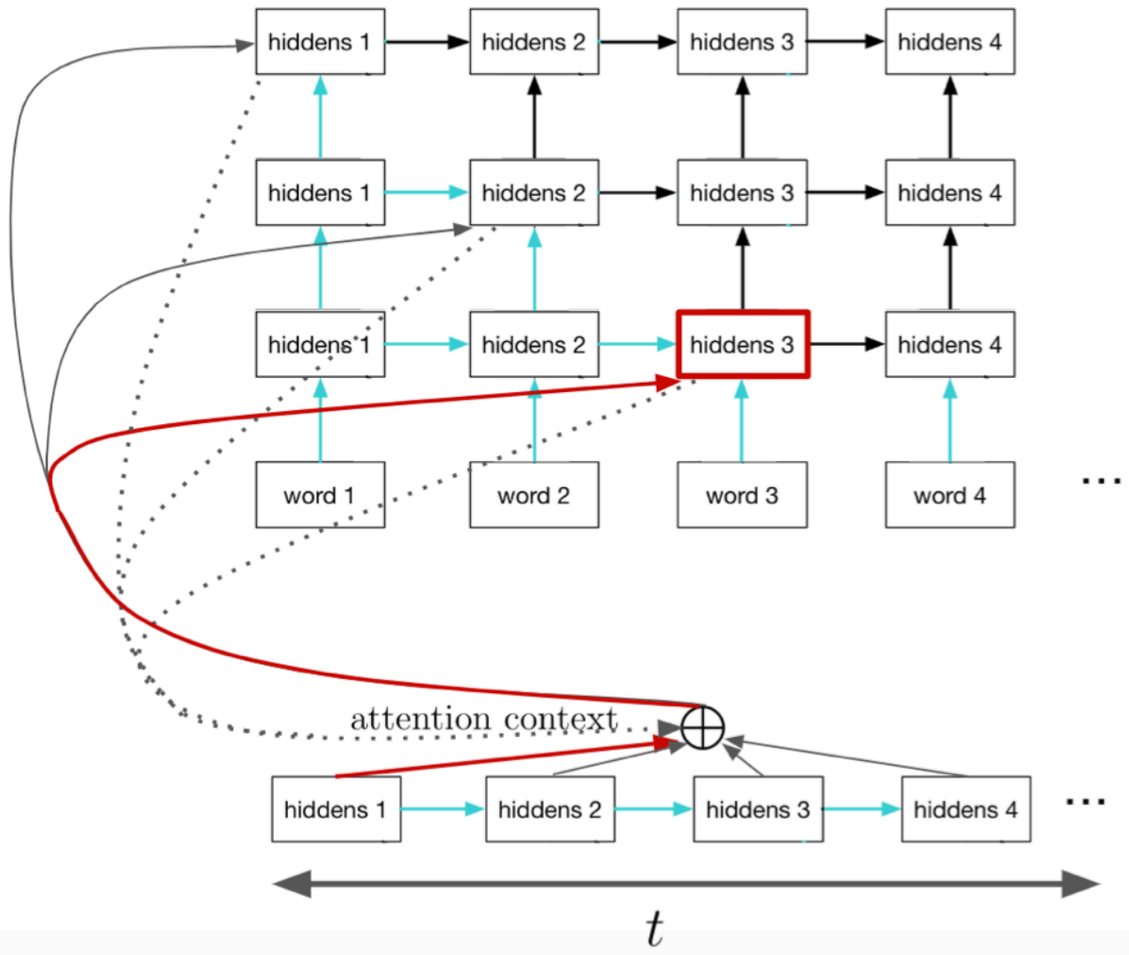


- Both the input embeddings and the outputs of an RNN can be computed in parallel.
- The blue hidden units are independent given the red.
- The numer of sequential operation is still propotional to t.

Computational Cost and Parallelism

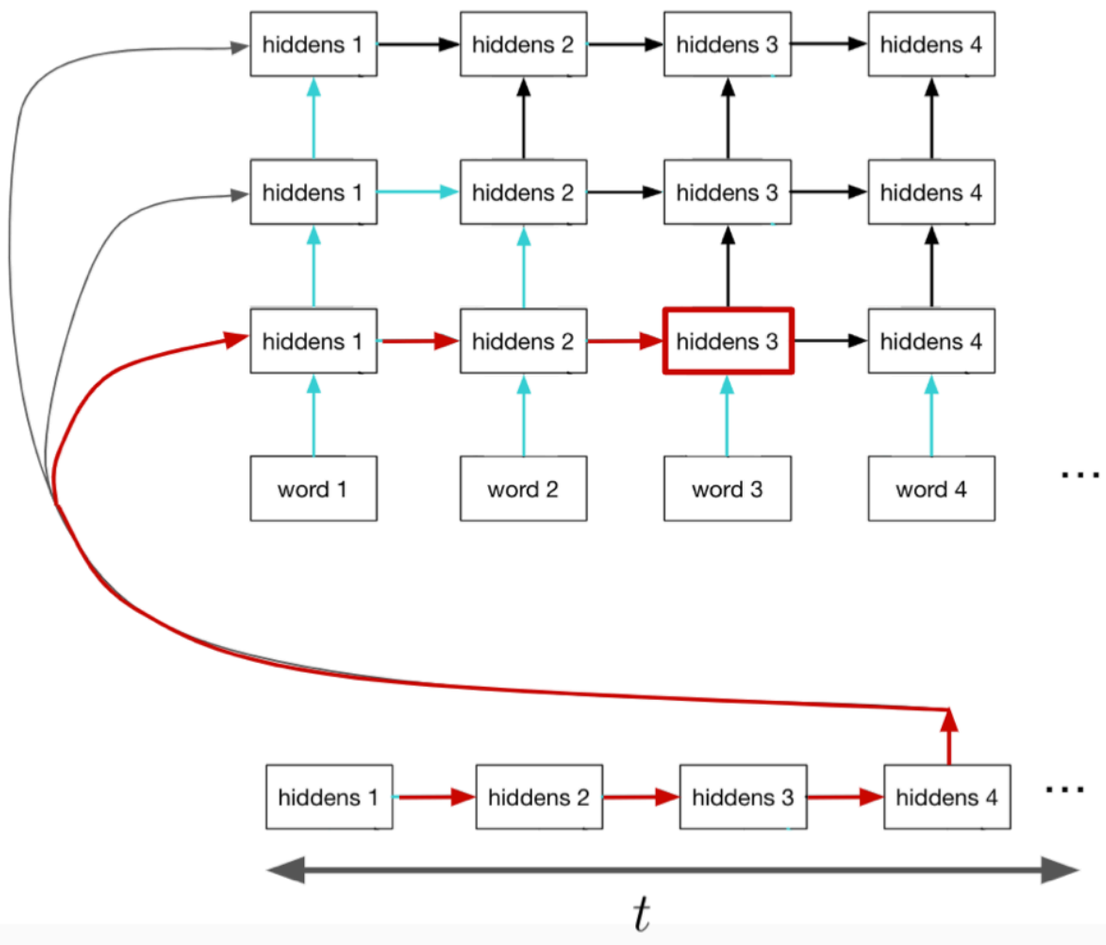
- In the standard encoder-decoder RNN, the maximum path length across time is propotional to the number of time steps.
- Attention-based RNNs have a constant path length between the encoder inputs and the decoder hidden states.
    - Learning becomes easier if all the information are present in the inputs.

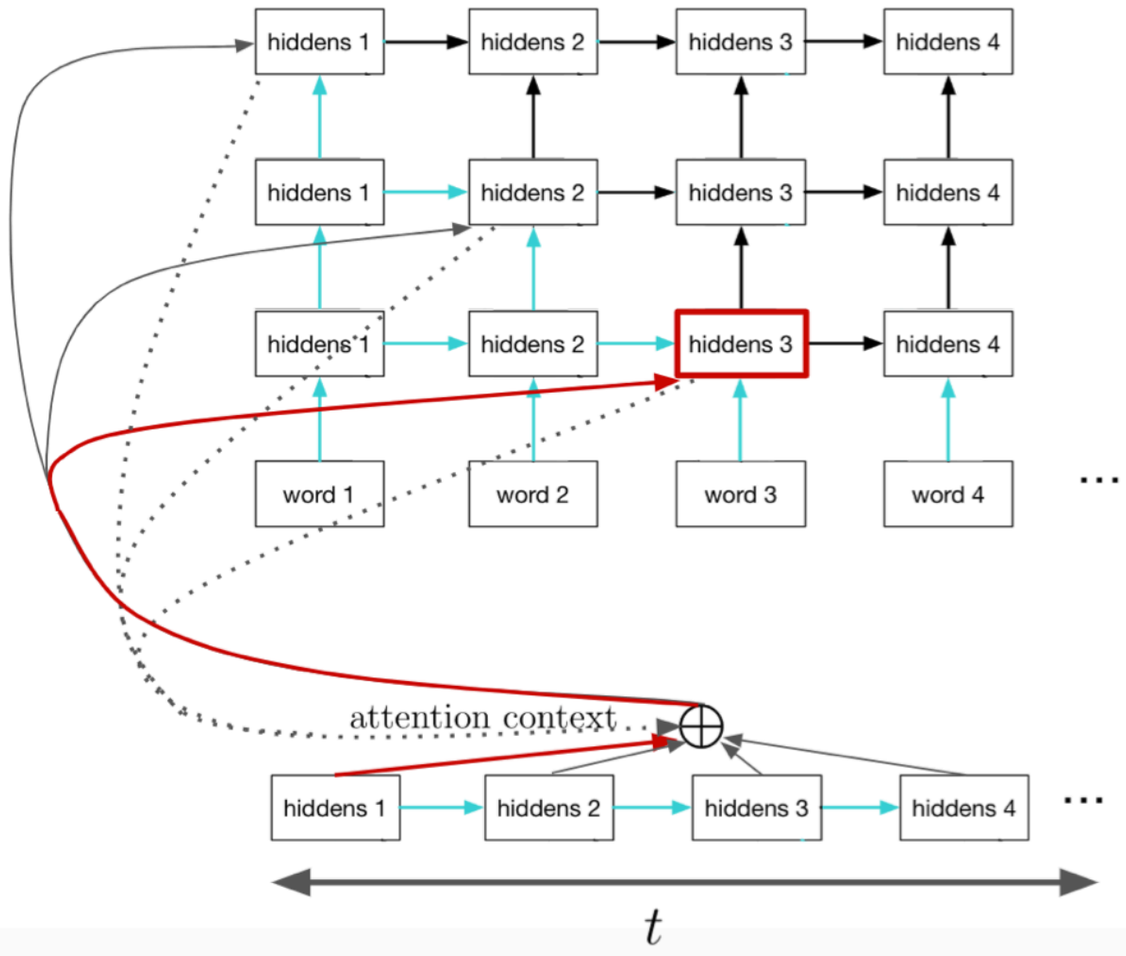




Computational Cost and Parallelism

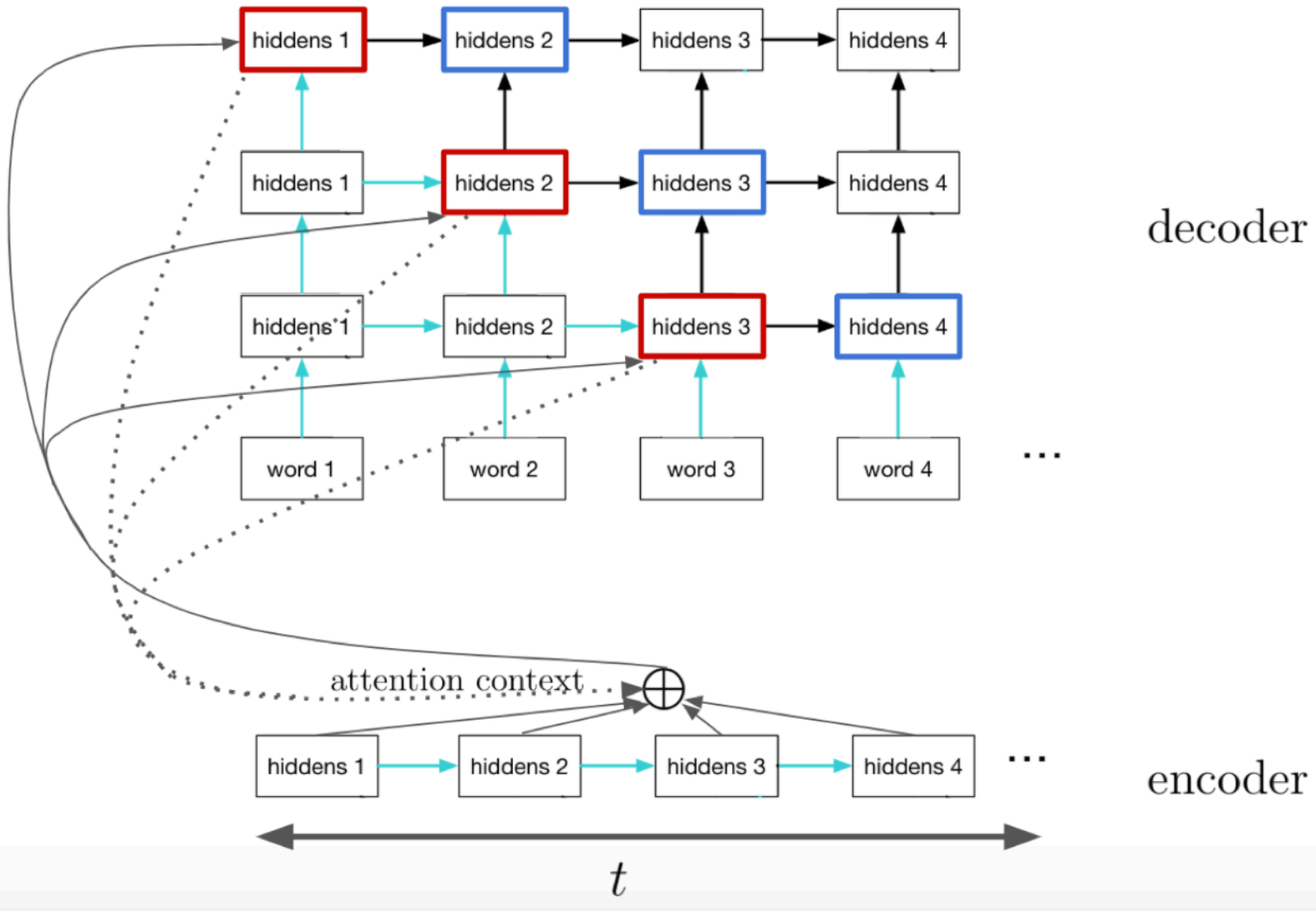
- In the standard encoder-decoder RNN, the maximum path length across time is propotional to the number of time steps.
- Attention-based RNNs have a constant path length between the encoder inputs and the decoder hidden states.
    - Learning becomes easier if all the information are present in the inputs.





Computational Cost and Parallelism

- Attention-based RNNs achieves efficient content-based addressing at the cost of re-computing context vectors at each time step.
  - *Bahdanau et. al.* computes context vector over the entire input sequence of length t using a neural network of k^2 connections.
  - Computing the context vectors adds a $t \times k^2$ cost at each time step.



Computational Cost and Parallelism

- In summary:
  - t: sequence length, d: # layers and k: # neurons at each layer.

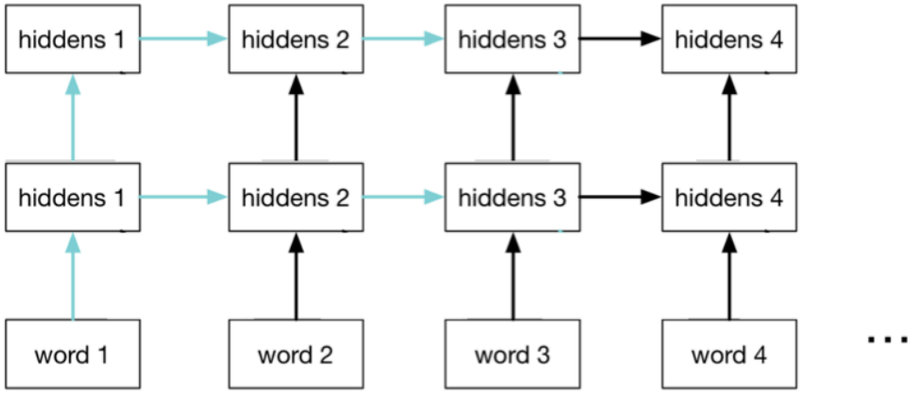
Model	training complexity	training memory	test complexity	test memory
RNN	$t \times k^2 \times d$	$t \times k \times d$	$t \times k^2 \times d$	$k \times d$
RNN+attn.	$t^2 \times k^2 \times d$	$t^2 \times k \times d$	$t^2 \times k^2 \times d$	$t \times k \times d$

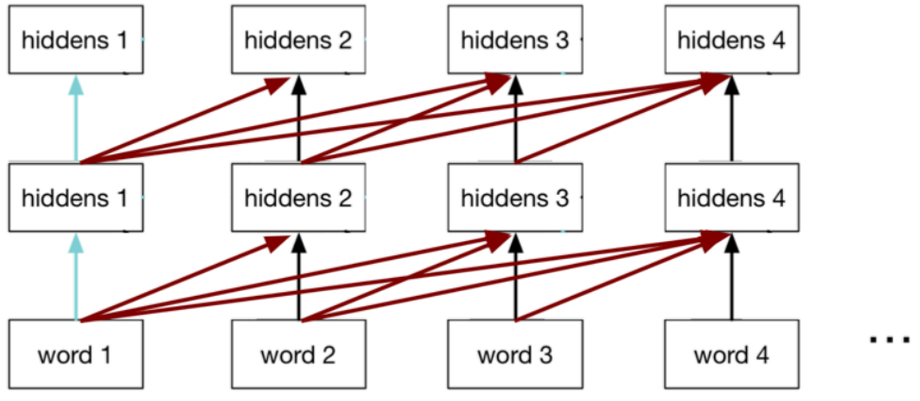
- Attention needs to re-compute context vectors at every time step.
- Attention has the benefit of reducing the maximum path length between long range dependencies of the input and the target sentences.

Model	sequential operations	maximum path length across time
RNN	t	t
RNN+attn.	t	1

Improve Parallelism

- RNNs are sequential in the sequence length t due to the number hidden-to-hidden lateral connections.
  - RNN architecture limits the parallelism potential for longer sequences.
- The idea: remove the lateral connections. We will have a deep autoregressive model, where the hidden units depends on all the previous time steps.

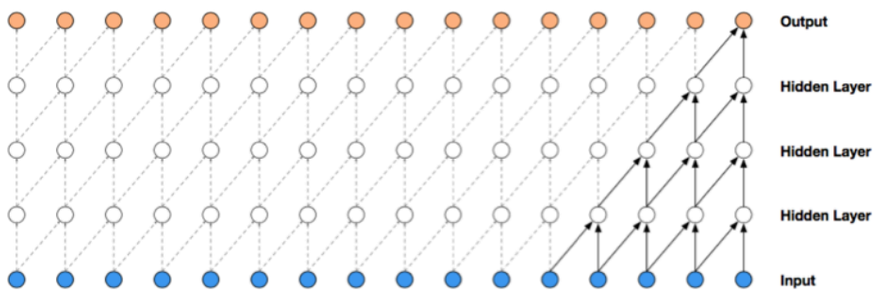


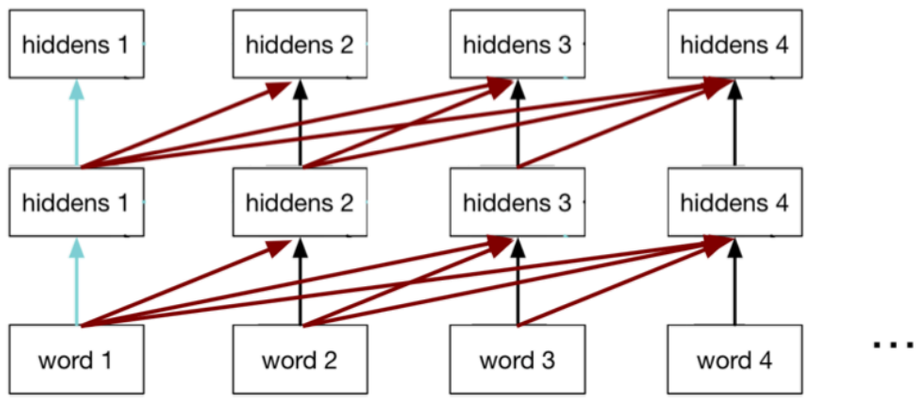


- Benefit: the number of sequential operations are now independent of the sequence length.

Attention is All You Need

- Autoregressive models like PixelCNN and WaveNet from Lecture 15 used a fixed context window with causal convolution.
- We would like our model to have access to the entire history at each hidden layer.
- But the context will have different input length at each time step. Max or average pooling are not very effective.
- We can use attention to aggregate the context information by attending to one or a few important tokens from the past history.





Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

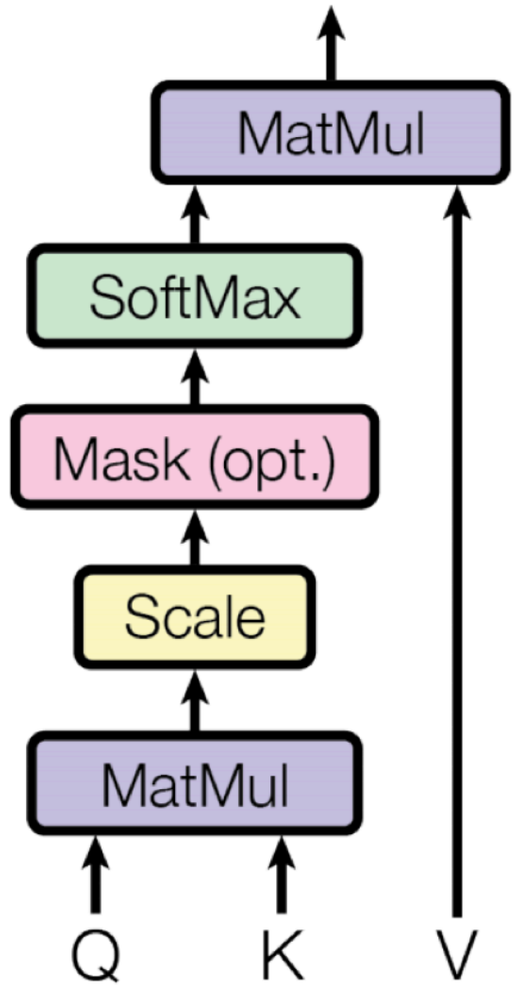
Attention is All You Need

- In general, Attention mappings can be described as a function of a query and a set of key-value pairs.
- Transformers use a "Scaled Dot-Product Attention" to obtain the context vector:

$$\mathbf{c}^{(t)} = \text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V,$$

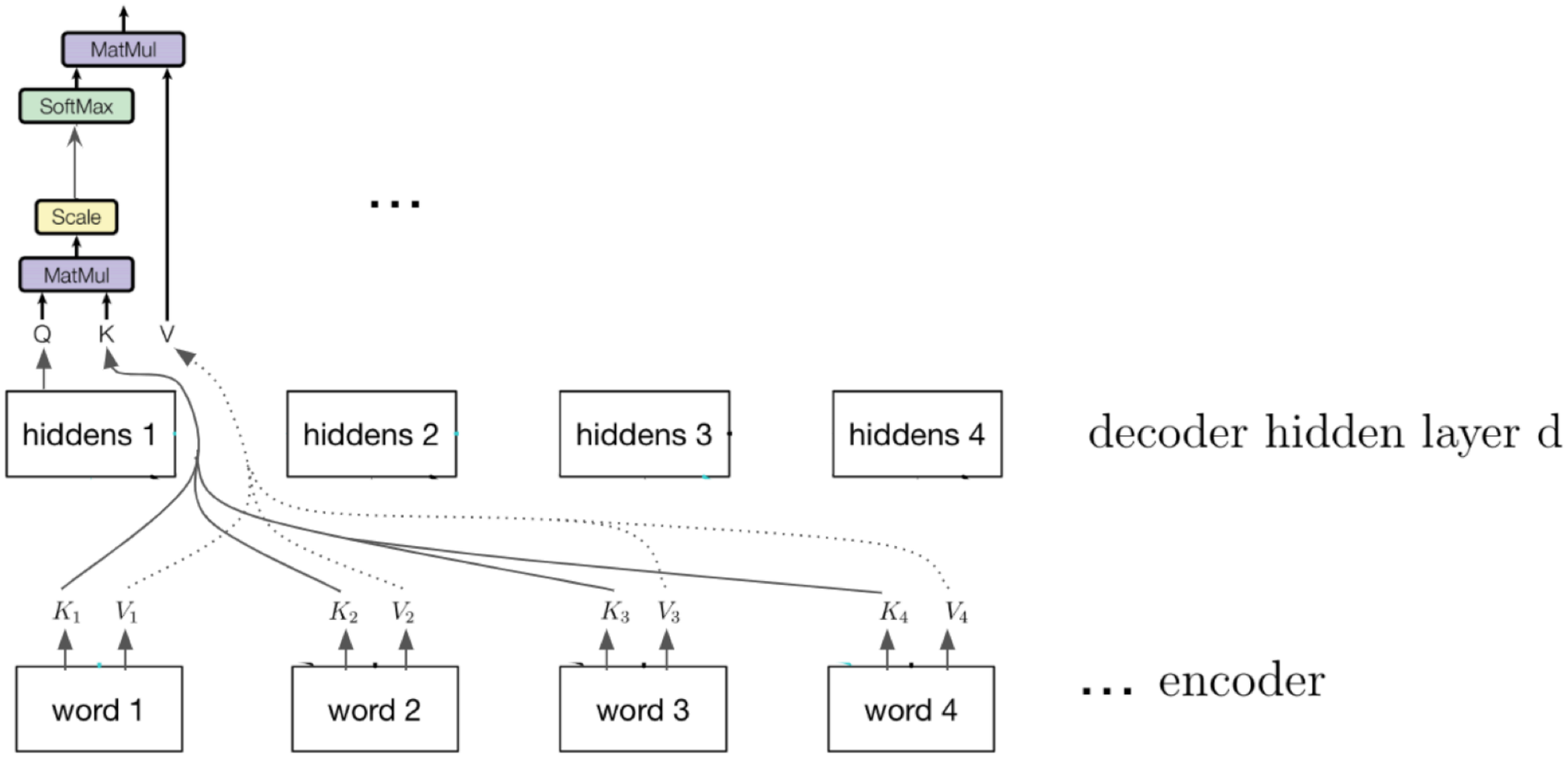
scaled by square root of the key dimension d_K.

- Invalid connections to the future inputs are masked out to preserve the autoregressive property.



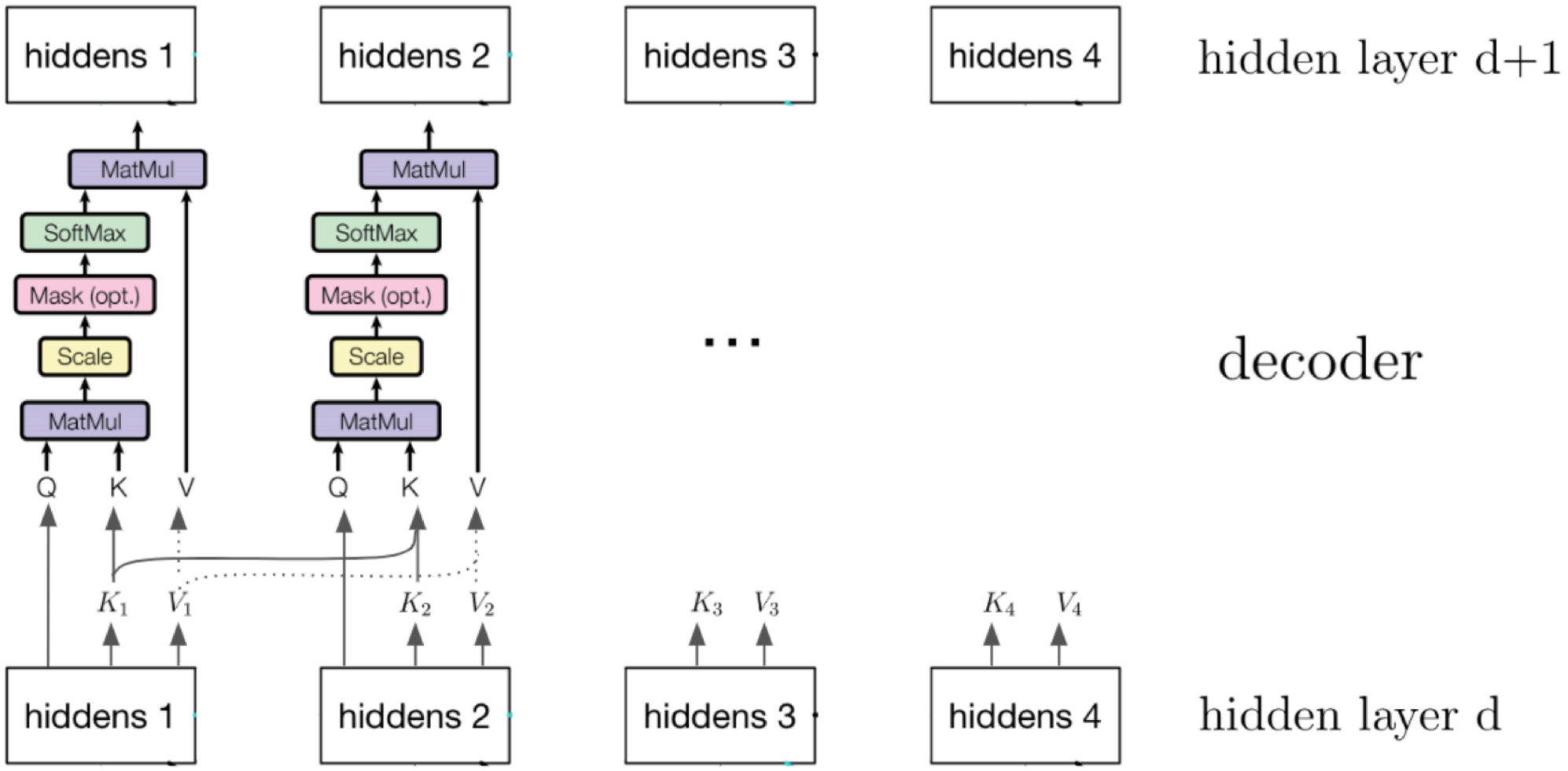
Attention is All You Need

- Transformer models attend to both the encoder annotations and its previous hidden layers.
- When attending to the encoder annotations, the model computes the key-value pairs using linearly transformed the encoder outputs.

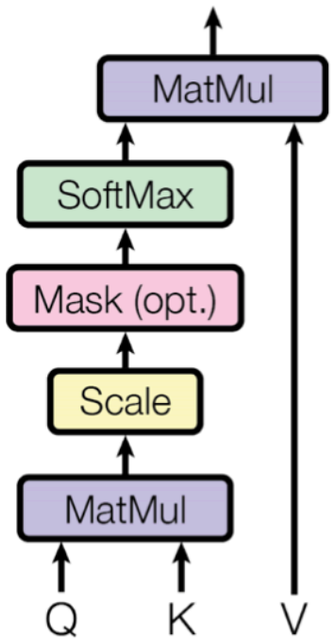


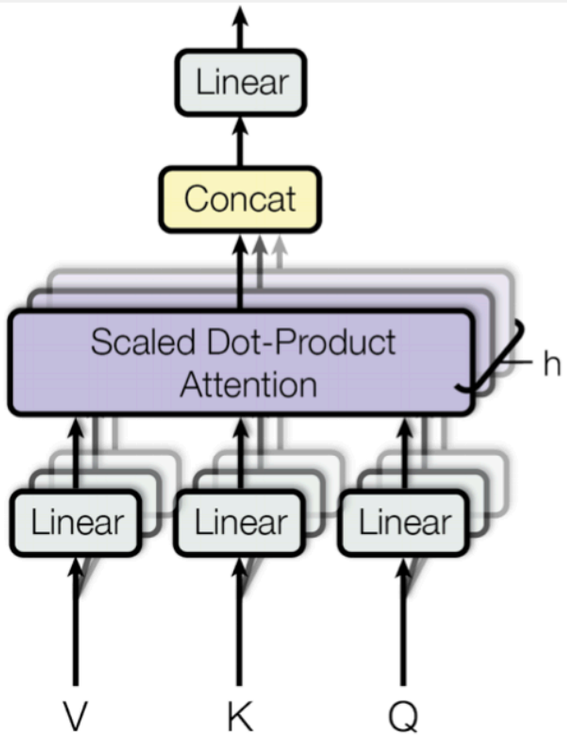
Attention is All You Need

- Transformer models also use "self-attention" on its previous hidden layers.
- When applying attention to the previous hidden layers, the casual structure is preserved.



Attention is All You Need





- The Scaled Dot-Product Attention attends to one or few entries in the input key-value pairs.
  - Humans can attend to many things simultaneously.
- The idea: apply Scaled Dot-Product Attention multiple times on the linearly transformed inputs.

$$\text{MultiHead}(Q, K, V) = \text{concat}\left(\mathbf{c}_1, \cdots, \mathbf{c}_h\right) W^O,$$

$$\mathbf{c}_i = \text{attention}(QW_i^Q, KW_i^K, VW_i^V).$$

Positional Encoding

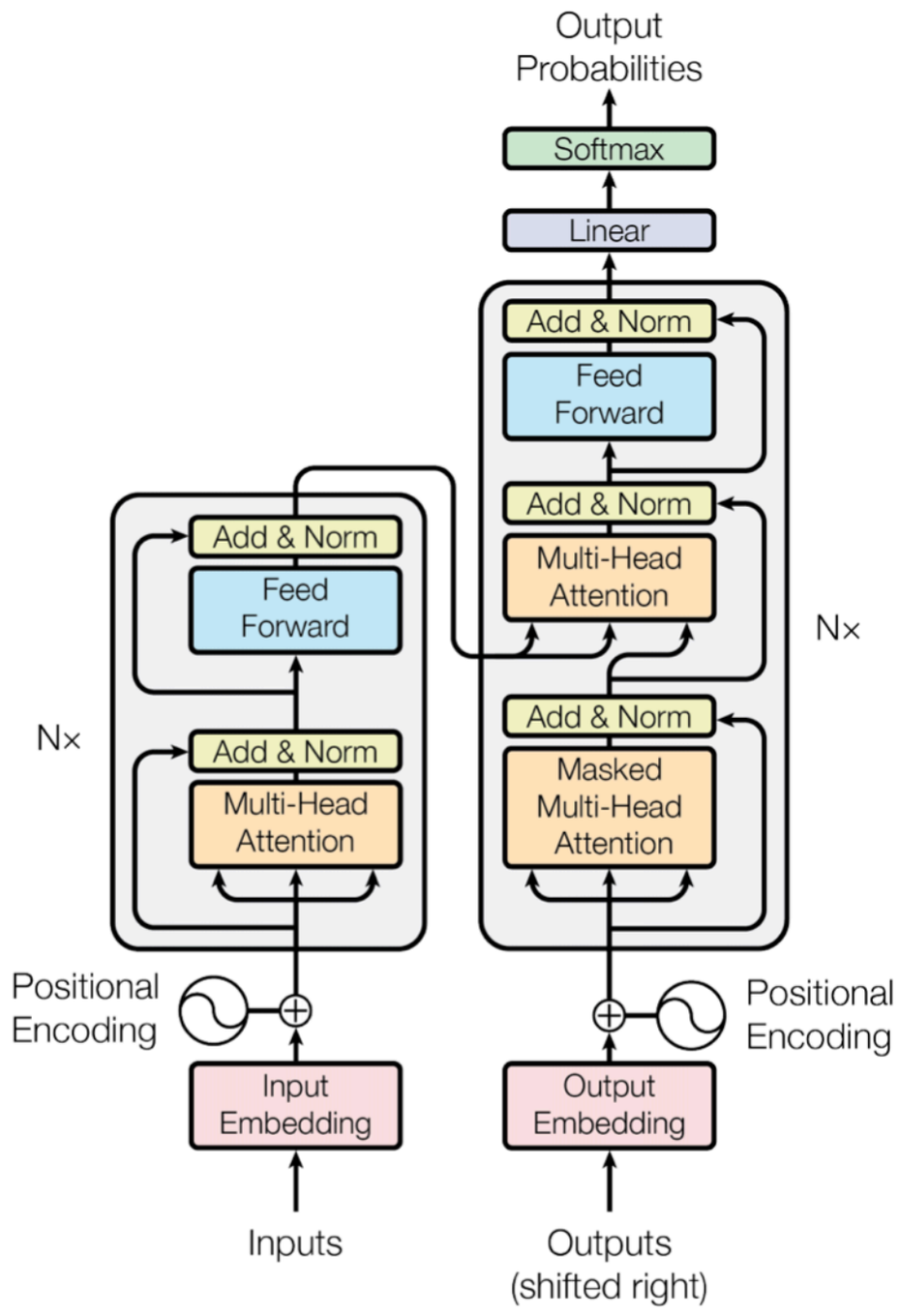
- Unlike RNNs and CNNs encoders, the attention encoder outputs do not depend on the order of the inputs. (Why?)
- The order of the sequence conveys important information for the machine translation tasks and language modeling.
- The idea: add positional information of a input token in the sequence into the input embedding vectors.

$$\mathrm{PE}_{pos,2i} = sin(pos/10000^{2i/d_{emb}}),$$
$$\mathrm{PE}_{pos,2i+1} = cos(pos/10000^{2i/d_{emb}}),$$

- The final input embeddings are the concatenation of the learnable embedding and the postional encoding.

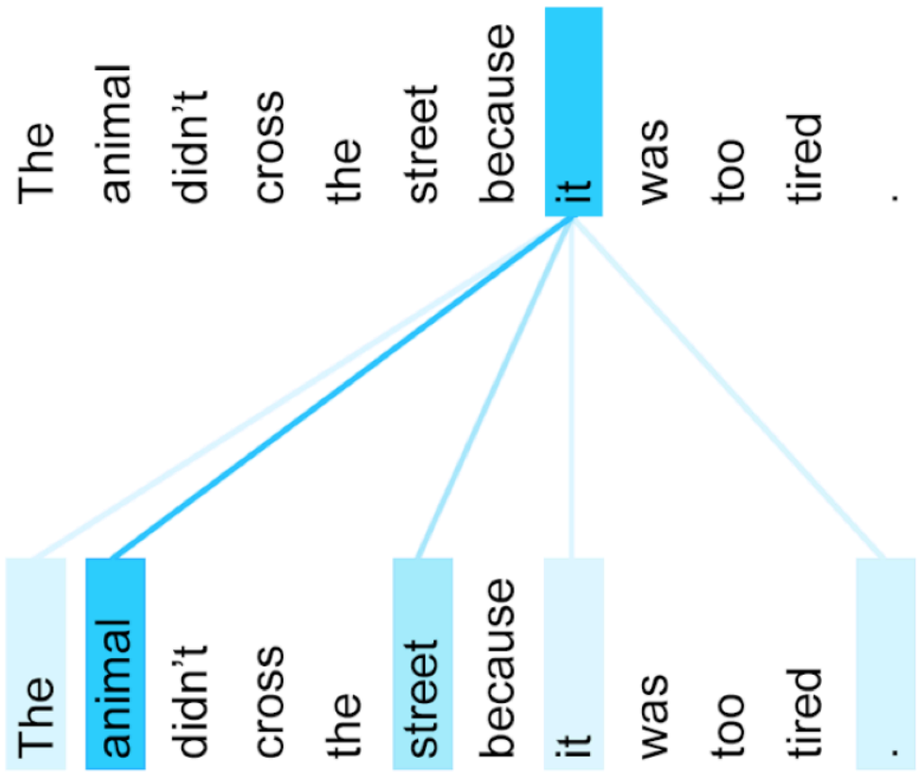
Transformer Machine Translation

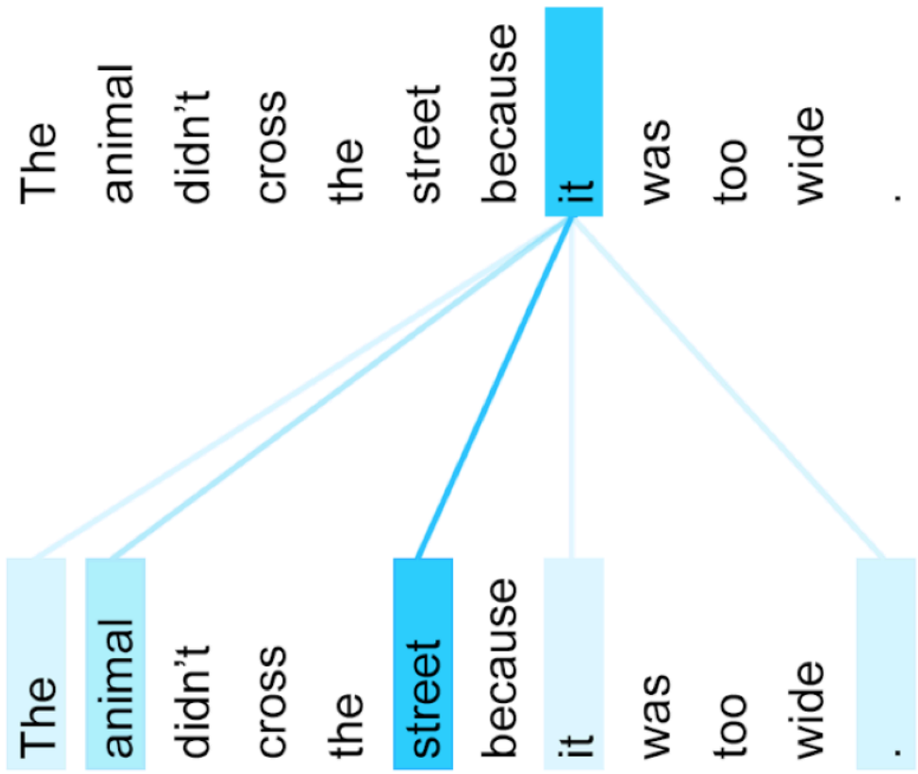
- Transformer has a encoder-decoder architecture similar to the previous RNN models.
  - except all the recurrent connections are replaced by the attention modules.
- The transfomer model uses N stacked self-attention layers.
- Skip-connections help preserve the positional and identity information from the input sequences.



Transformer Machine Translation

- Self-attention layers learnt "it" could refer to different entities in the different contexts.





- Visualization of the 5th to 6th self-attention layer in the encoder.

`https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html`

Transformer Machine Translation

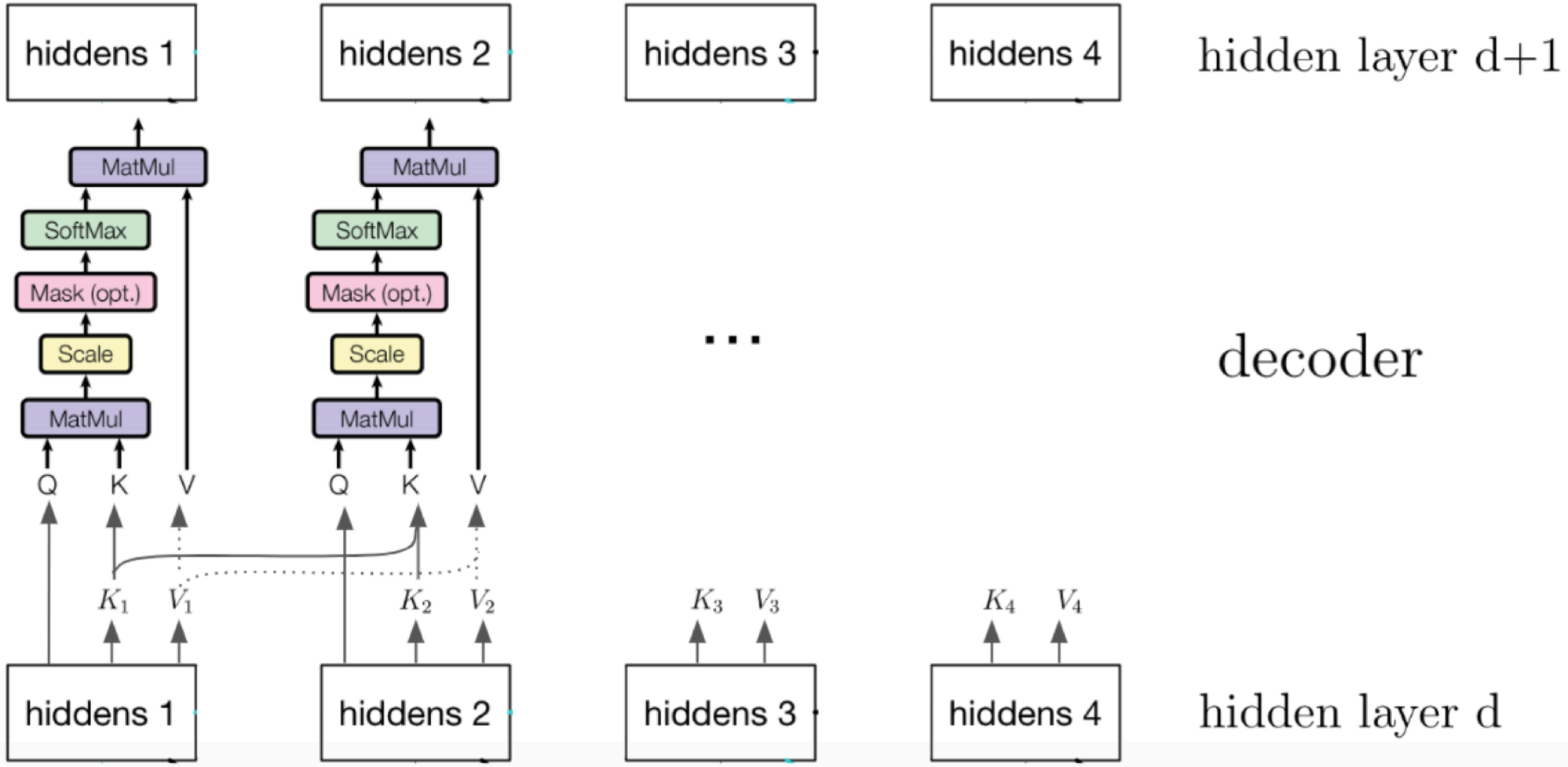
- BLEU scores of state-of-the-art models on the WMT14 English-to-German translation task

Translation Model	Training time	BLEU (diff. from MOSES)
Transformer (large)	3 days on 8 GPU	**28.4** **(+7.8)**
Transformer (small)	1 day on 1 GPU	24.9 (+4.3)
GNMT + Mixture of Experts	1 day on 64 GPUs	26.0 (+5.4)
ConvS2S (FB)	18 days on 1 GPU	25.1 (+4.5)
GNMT	1 day on 96 GPUs	24.6 (+4.0)

Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

Computational Cost and Parallelism

- Self-attention allows the model to learn to access information from the past hidden layer, but decoding is very expensive.
- When generating sentences, the computation in the self-attention decoder grows as the sequence gets longer.



Computational Cost and Parallelism

- t: sequence length, d: # layers and k: # neurons at each layer.

Model	training complexity	training memory	test complexity	test memory
RNN	$t \times k^2 \times d$	$t \times k \times d$	$t \times k^2 \times d$	$k \times d$
RNN+attn.	$t^2 \times k^2 \times d$	$t^2 \times k \times d$	$t^2 \times k^2 \times d$	$t \times k \times d$
transformer	$t^2 \times k \times d$	$t \times k \times d$	$t^2 \times k \times d$	$t \times k \times d$

- **Transformer vs RNN**: There is a trade-off between the sequencial operations and decoding complexity.
  - The sequential operations in transformers are independent of sequence length, but they are very expensive to decode.
  - Transformers can learn faster than RNNs on parallel processing hardwards for longer sequences.

Model	sequential operations	maximum path length across time
RNN	t	t
RNN+attn.	t	1
transformer	d	1

Transformer Language Pre-training

- Similar to pre-training computer vision models on ImageNet, we can pre-train a language model for NLP tasks.
  - The pre-trained model is then fine-tuned on textual entailment, question answering, semantic similarity assessment, and document classification.