

Convolutional networks



10-701 Introduction to Machine Learning

Geoff Gordon

Signals as tensors

- An n -dimensional signal is represented as a tensor with n indices
 - ▶ e.g., grayscale image: 2 indices (pixel i, j)
 - ▶ e.g., color image: 3 indices (last is channel: R, G, or B)
 - ▶ e.g., pressure in a volume: 3 indices (voxel i, j, k)
- Each index also called a ***mode***
 - ▶ helps with overloading of the word “dimension”
- List of index ranges: the ***shape*** of the tensor
 - ▶ e.g., RGB image at VGA resolution: (640, 480, 3)
 - ▶ shape (3, 4, 2, 5) tensor has 120 total elements
 - ▶ first index can be 0, 1, 2; second 0, 1, 2, 3; ...

Tensor operations

- Componentwise $+$, $\times$, $\cos$, ...

▶ with *broadcasting*:

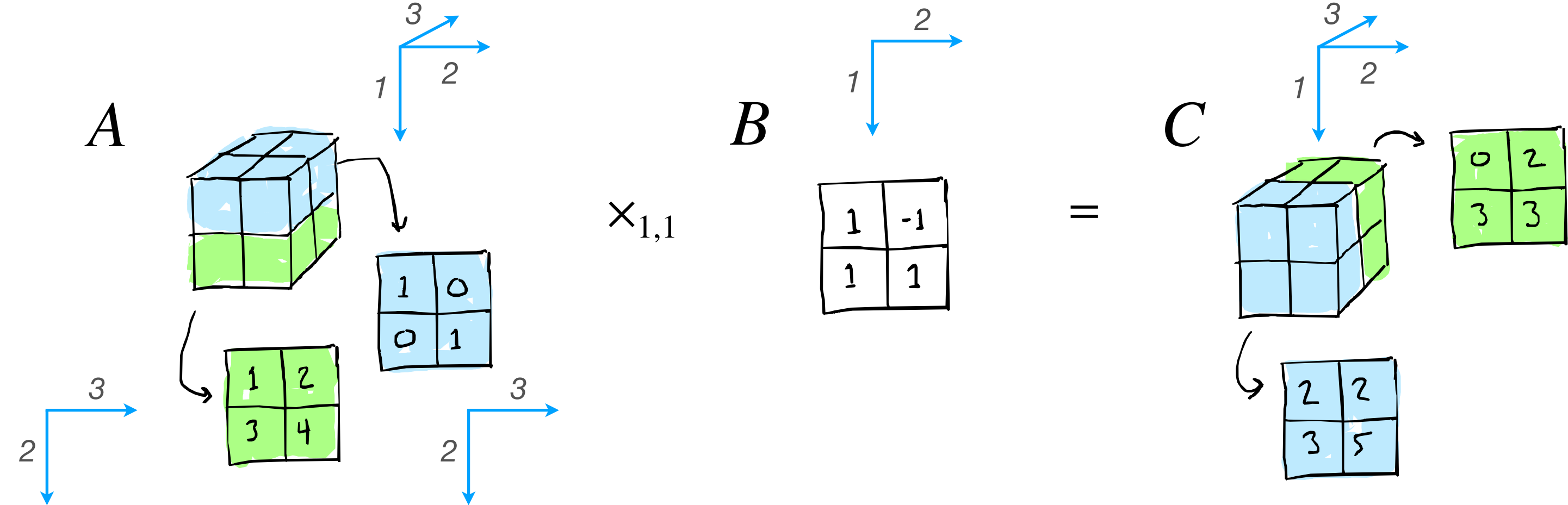
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \times (1 \ 2 \ 3) = \begin{pmatrix} 1 & 4 & 9 \\ 4 & 10 & 18 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$

- Implemented as NumPy `Array` or PyTorch `Tensor`

Tensor operations



- Contraction: e.g., $A \times_{3,2} B$ or $A \times_{1,1} B$
 - ▶ like matrix multiplication (precise definition below), but using mode 3 of A and mode 2 of B — regular matrix multiplication is $\times_{2,1}$
 - ▶ modes of $A \times_{3,2} B$: modes of A (except 3) then modes of B (except 2)
 - ▶ by convention, all of A 's modes first (in order) then B 's
- Represents a linear function of B with parameters A
 - ▶ or a linear function of A with parameters B

Elementwise notation

- (a_{ij}) or a_{ij} means a matrix whose i, j element is a_{ij}
 - ▶ range of i, j can be left implicit or specified separately:
e.g., $i \in \{1 \dots 5\}, j \in \{1 \dots 7\}$
- Similarly, a_{ijkl} means a 4-mode tensor
- So does $a_{ijk}b_{jkl}$ — the i, j, k, l element is $a_{ijk}b_{jkl}$
- So does $a_{ijk} + b_{jkl}$ (with $+$ instead of $\times$)
- The expression tells us how to compute a generic element of the result tensor from elements of the input tensors

Einstein summation

- We allow indices to appear in either raised or lowered positions: both a_i^j and a_j^i refer to 2-mode tensors (matrices) whose indices are i and j
- Only effect of raising and lowering is the ***Einstein summation convention***: any index that appears ***both*** raised and lowered in an expression gets summed out
 - ▶ in $a_{ij}^k b_{jk}^l c_{lm}$ indices i, j, m appear only lowered; k, l are both raised and lowered
 - ▶ so the result is a tensor f_{ijm} : element at index i, j, m is
$$\sum_k \sum_l a_{ijk} b_{jkl} c_{lm}$$
- Remaining indices keep their positions (raised or lowered)

Example

- If $\begin{matrix} j \rightarrow \\ \downarrow i \end{matrix}$
 $a_i^j = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad b_j = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \quad c^i = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

- then

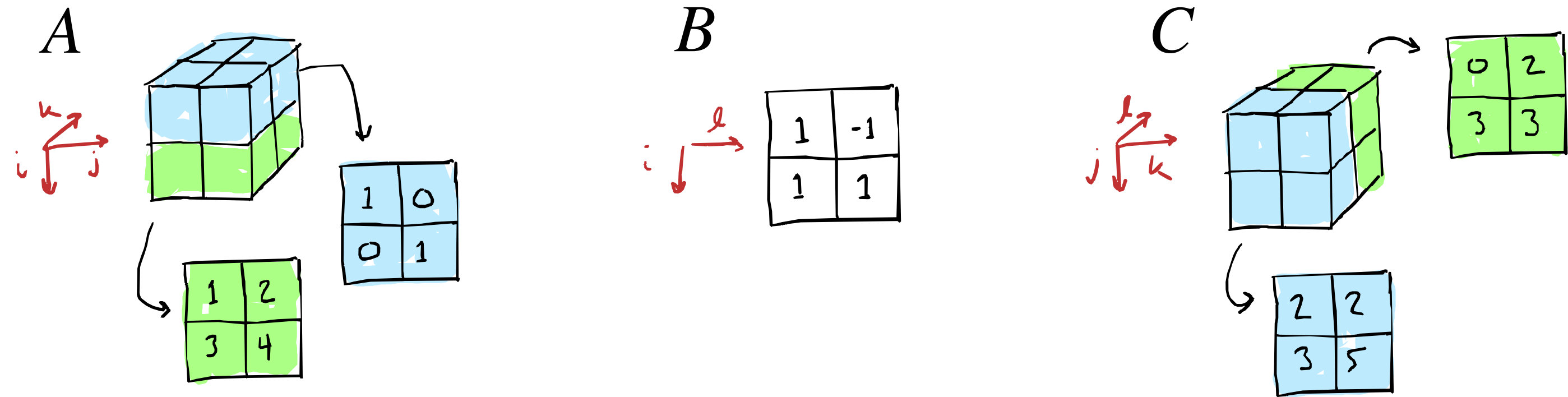
$$a_i^j b_j =$$

$$a_i^j c^i =$$

Transpose

- Changing around raised and lowered indices is a generalization of transpose
 - ▶ a_j^i and a_i^j have the same elements, as do x_j and x^j
 - ▶ convention: lowered indices in a 1- or 2-mode tensor correspond to rows, raised ones to columns
 - ▶ so x_j is a column vector (index runs through rows)
- With this convention:
 - ▶ Ax is $a_i^j x_j$
 - ▶ and $A^T y$ is $a_j^i y_i$ (i acts like column index when we raise it, and j acts like row index when we lower it; since the elements are the same, a_j^i represents A^T)
 - ▶ and $y^T A$ is $a_i^j y^i$ (raised index of y , leaving a_i^j alone)

Tensor contraction as Einstein summation



$$A_{ijk}B_l^i = C_{jkl}$$

- To get contraction, raise exactly one index instance
 - ▶ e.g., $A_{ijk}B_l^i$ is a 3-mode tensor C_{jkl}
 - ▶ result of contracting A and B along the i index
 - ▶ element j, k, l is $\sum_i A_{ijk}B_l^i$

Python *einsum*

*available in both
NumPy and PyTorch,
but some edge cases
have different behavior*

- A version of Einstein summation:
 - ▶ `np.einsum('ij,jk->ik', a, b)` is matrix multiplication
- Subscript string (first argument) formatted as $A, B \rightarrow C$
 - ▶ part A names the indices of first input tensor a
 - ▶ part B names the indices of second input tensor b
 - ▶ part C tells us indices of output tensor
 - ▶ indices that don't appear in output are summed out
- Variants
 - ▶ can have any number of input tensors
 - ▶ $\rightarrow C$ is optional (defaults to all non-duplicated indices in alphabetical order)
 - ▶ broadcasting: `np.einsum('...ij->...ji', a)` transposes last two modes

Linear functions in derivatives

- Revisiting convention from earlier: if we have two tensors y_{ij} and x_{klm} then derivative $\frac{dy}{dx}$ is a 5-mode tensor representing a linear function from dx to dy
- We can write this linear function as a tensor contraction or Einstein summation:

$$dy_{ij} = A_{ij}^{klm} dx_{klm}$$

- ▶ where A is the 5-mode tensor of coefficients
- Indices are $ijklm$, with last 3 indices (the ones corresponding to input x) raised
- Ex: Jacobian is $dy_i = A_i^j dx_j$

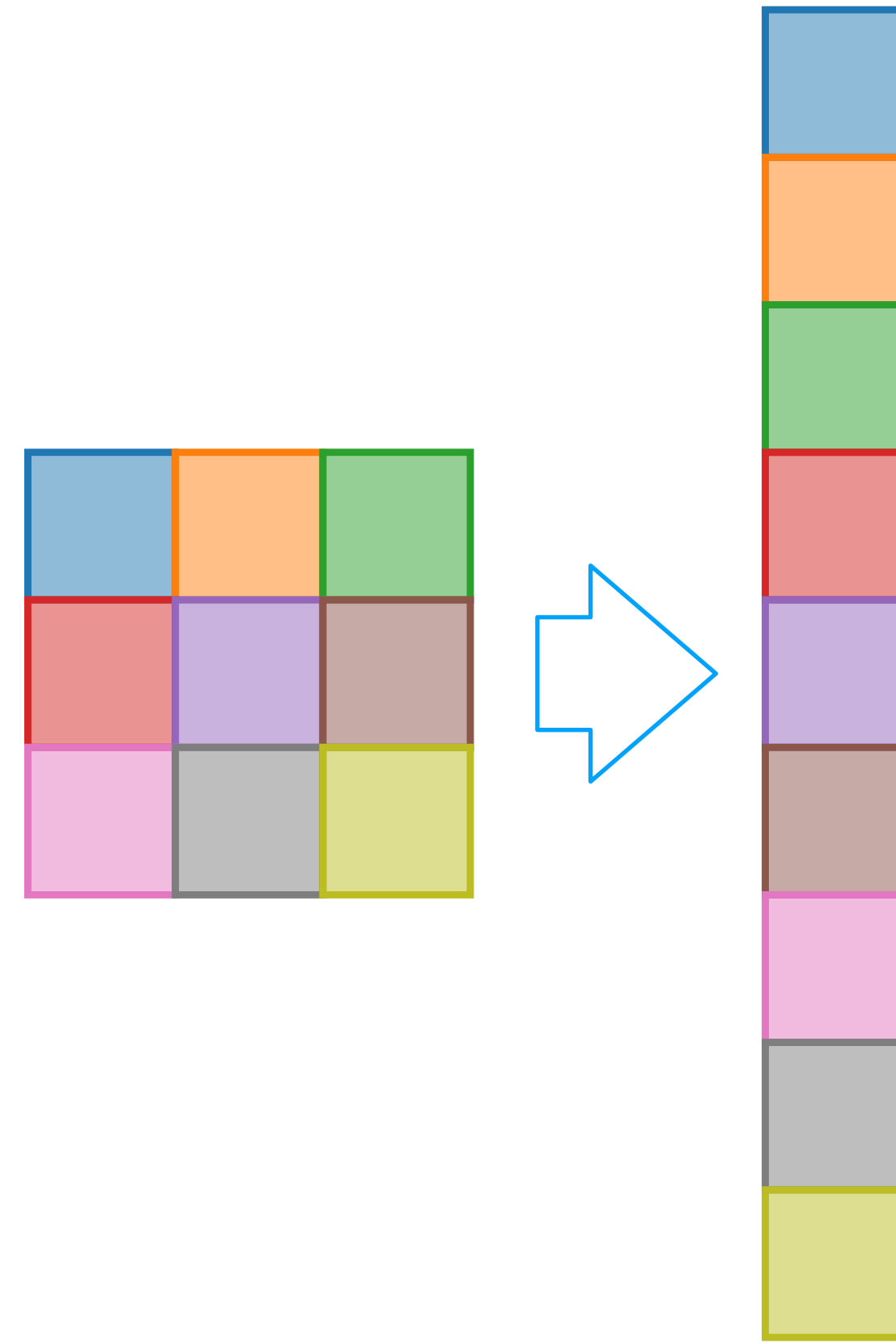
Batching

- Tensors make it easier to work with minibatches
- Just add a mode (typically last) that indexes samples within a batch
 - ▶ 640×480 images in batches of 17 $\rightarrow 640 \times 480 \times 17$
- Works with Python indexing conventions: if we drop the last index, we work with all samples at once

Convolutional neural networks

- We can use a convolution in a deep net layer
 - ▶ linear map $a \rightarrow a * b + c$ ($a, b \in \mathbb{R}^d$, $c \in \mathbb{R}$)
 - ▶ input a , weights b , bias c
 - ▶ learn b, c by SGD
- Variations: $b \in \mathbb{R}^k$ with $k \ll d$; omit c
- Nets with convolutions are *CNNs* or *convnets*
 - ▶ can also use the usual set of other blocks: fully connected linear, normalize, residual, pointwise nonlinearity, ...

Connecting to MLP layers

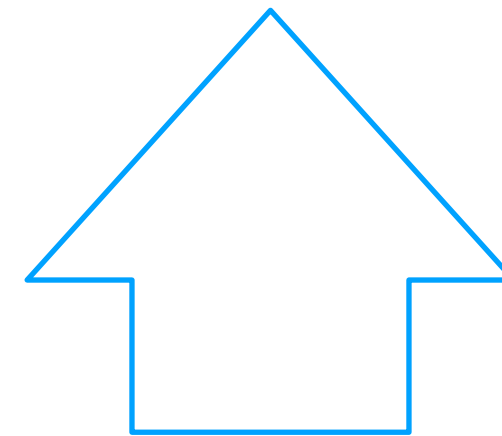


- To connect convolutional layers and MLP layers, need to unroll matrices or tensors into vectors (and vice versa)
 - ▶ called **vec** and **unvec**; use Python `flatten` & `reshape`

**Example:
detect ones
block of
length 3**

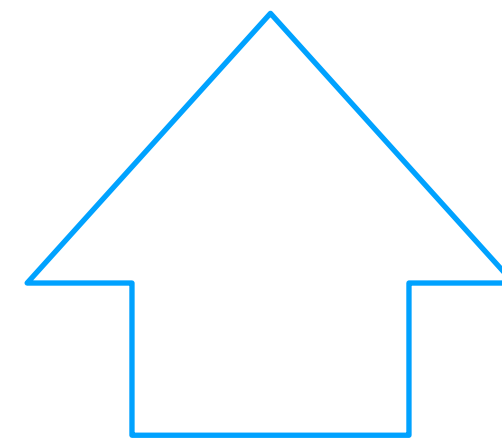
unlike this example, a real
network would **learn** the
weights and biases

$$z_3 = (0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0)$$



pointwise nonlinearity: $[z - 1]_+$

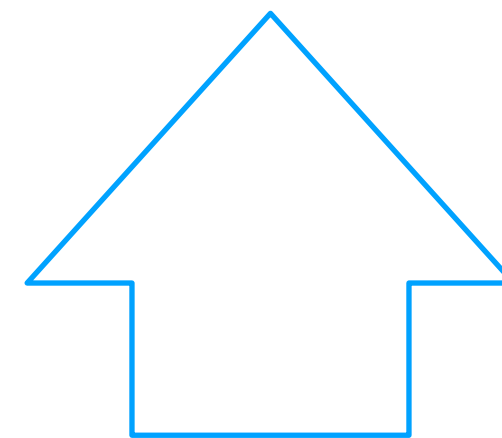
$$z_2 = (0 \ -1 \ 0 \ 0 \ 2 \ 0 \ 0 \ -1 \ 0 \ 0)$$



convolution: weights

$$(1 \ 0 \ 0 \ -1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)$$

$$z_1 = (0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ -1 \ 0 \ 0)$$



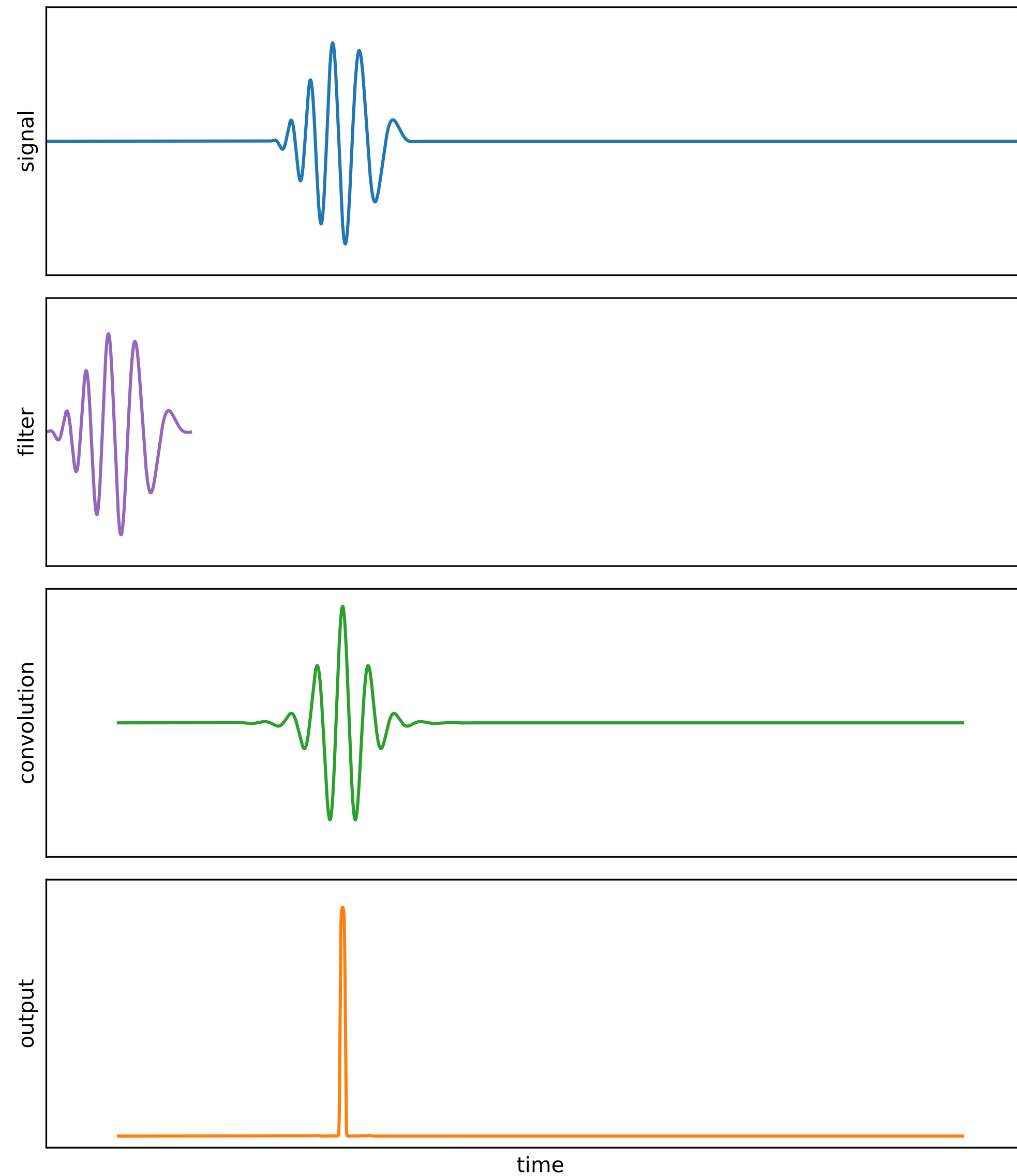
convolution: weights

$$(-1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)$$

$$x = (0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0)$$

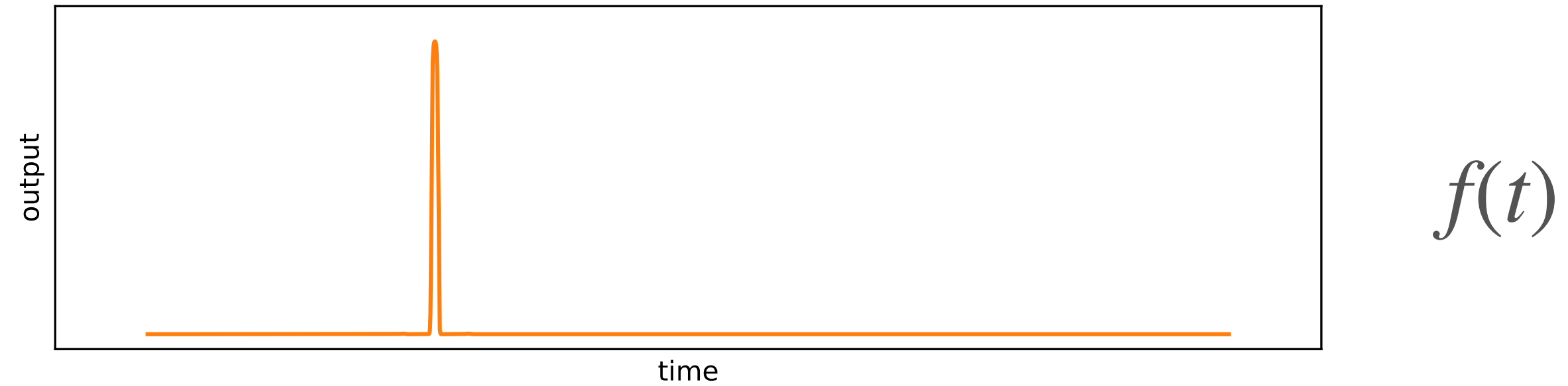
could group these two

Bigger version



- Same idea: detect a chirp in an audio signal
- Convolve input signal w/ chirp filter, then apply sigmoid pointwise

Pooling



max-pool:

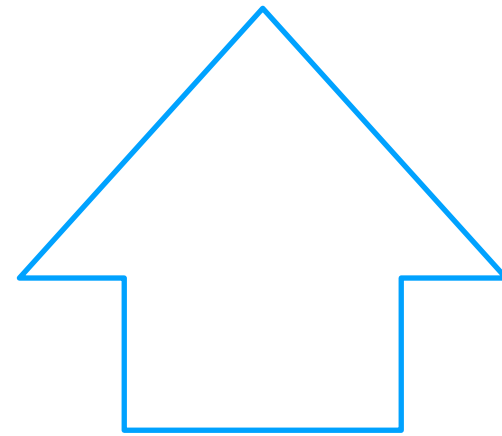
sum-pool:

- Convolution gives us translation *equivariance*
- If we want *invariance*,
 - ▶ e.g., find the average value of the signal
 - ▶ e.g., check whether the max of the signal is $\geq$ threshold
- Need another step — called *pooling*
 - ▶ e.g., max-pool a group of input units = make a single unit whose value is max of inputs

Pooling in action

Change from
localize to detect

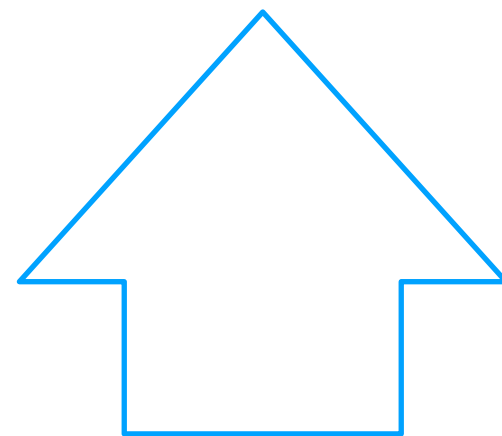
$$z_4 = (1)$$



$$\text{Max pool: } \max(0, 0, 0, 0, 1, 0, 0, 0, 0) = 1$$

$$\text{Sum pool: } 0 + 0 + 0 + 0 + 1 + 0 + 0 + 0 + 0 = 1$$

$$z_3 = (0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0)$$



$$\text{Convolution (w/ bias) followed by } [\cdot]_+$$

$$x = (0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0)$$

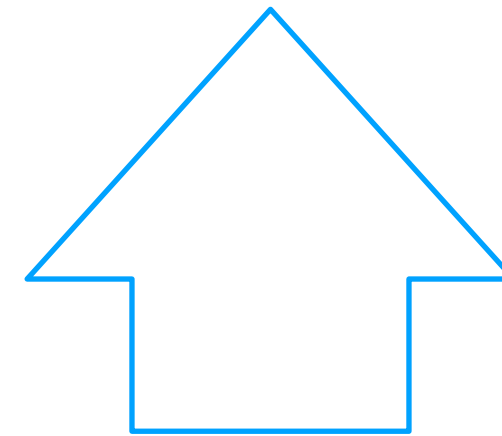
Local pooling

- Might want only *local* invariance (small shifts don't change output much)
- Can do *local* pooling — e.g., average or max of a block of adjacent values
- Can slide a window along signal as in convolution
- Or divide into blocks and pool each block (downsample)

***Local
pooling
example:
flexible-
length ones
block
detector***

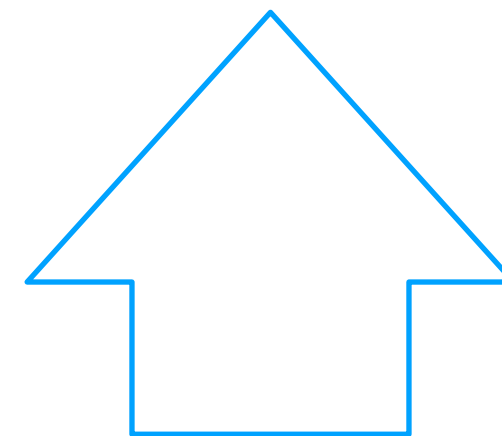
*from here, ReLU and pooling
can detect or localize*

$$z_3 = (0 \ -1 \ 2 \ -1 \ 0)$$



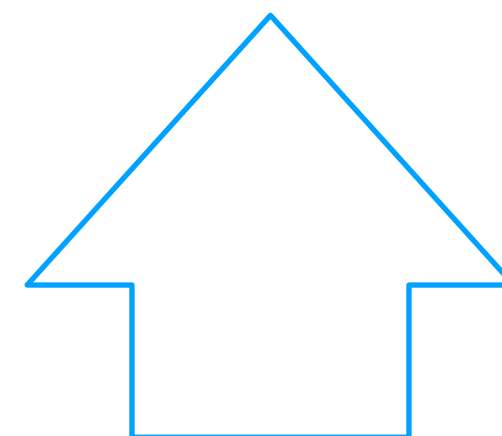
*convolution: weights
(1 -1)*

$$z_2 = (0 \ 0 \ 1 \ -1 \ 0)$$



sum pool in blocks of 2

$$z_1 = (0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ -1 \ 0 \ 0)$$



*convolution: weights
(-1 1)*

$$x = (0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0)$$

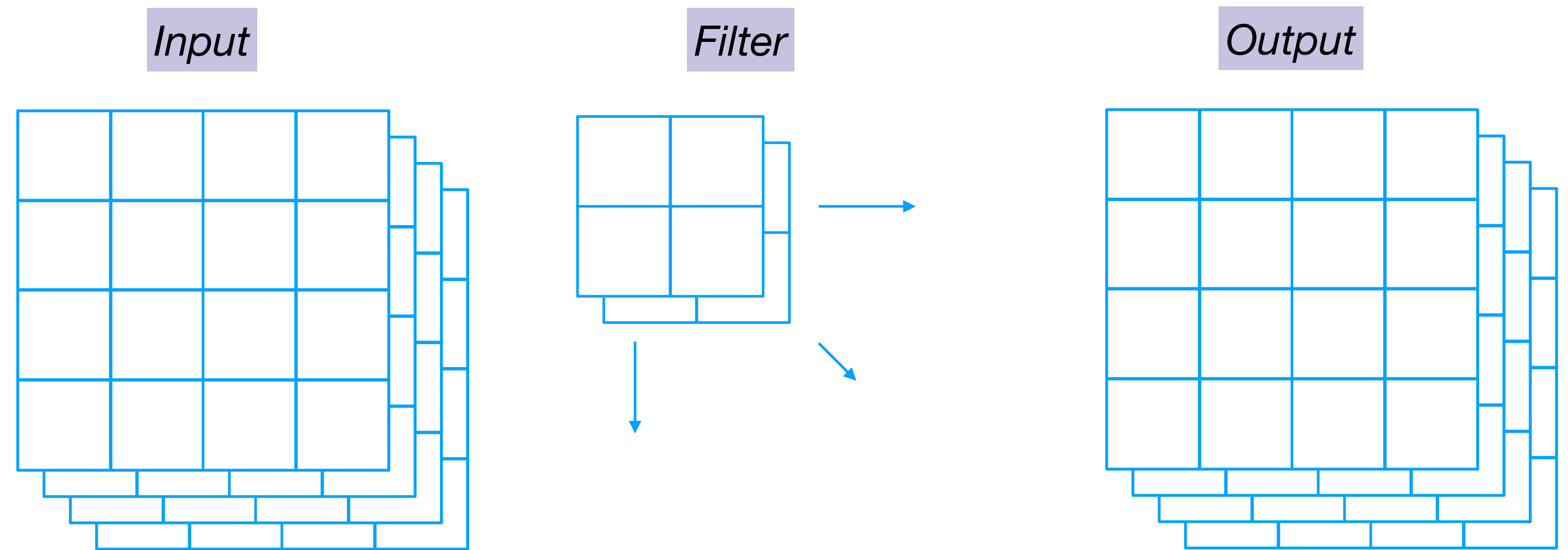
Channels

$$\text{e.g., } a \in \mathbb{R}^{640 \times 480}, b_i \in \mathbb{R}^{3 \times 3}, c_i \in \mathbb{R}$$

- Can do multiple convolutions in parallel
 - ▶ $a \rightarrow (a * b_1 + c_1, a * b_2 + c_2, \dots)$
 - ▶ each parameter set is a *head*, each output is a *channel*
 - ▶ different heads extract different kinds of info from signal
- Changes shape of activation tensor
 - ▶ e.g., if layer has 3d input: multiple heads $\rightarrow$ 4d output
 - ▶ e.g., image $\rightarrow$ (stack of band-pass images)
- If we think of a tensor as containing multiple channels, the channel index is also called *depth* or *layer* index

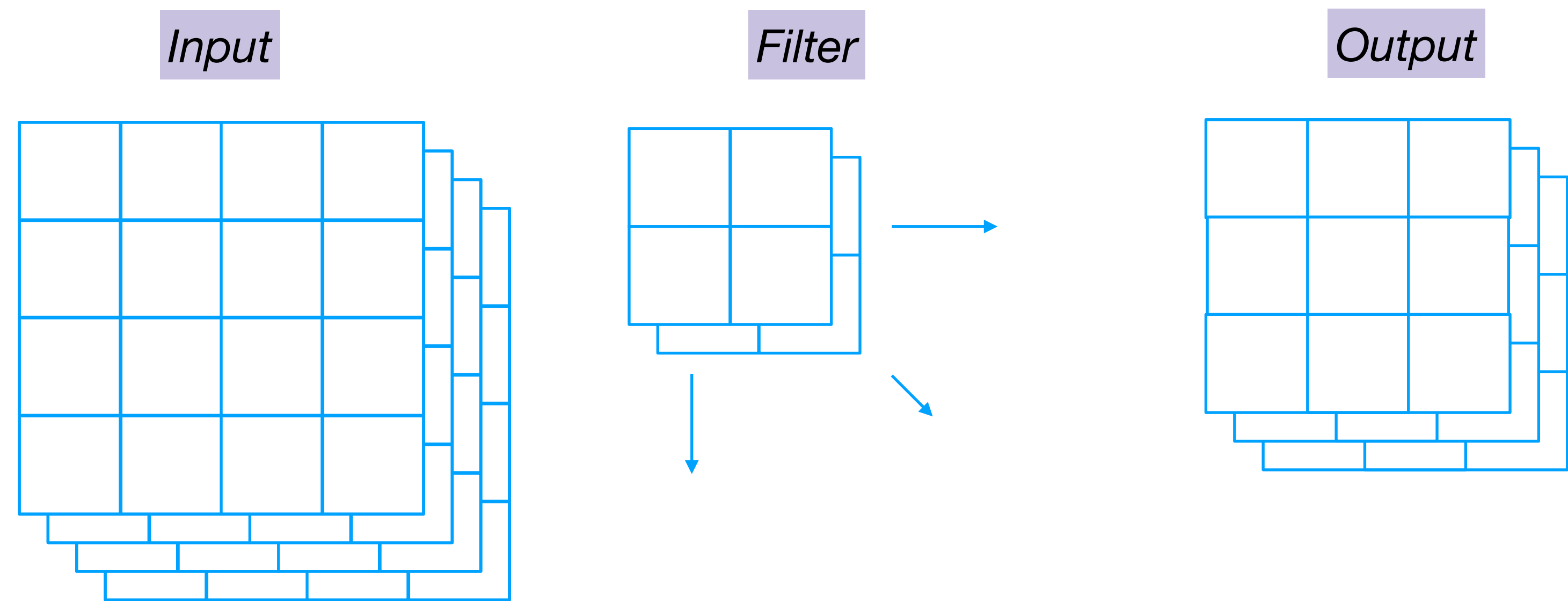
Standard convolution

*parameters like
dilation and stride
can be per-mode*



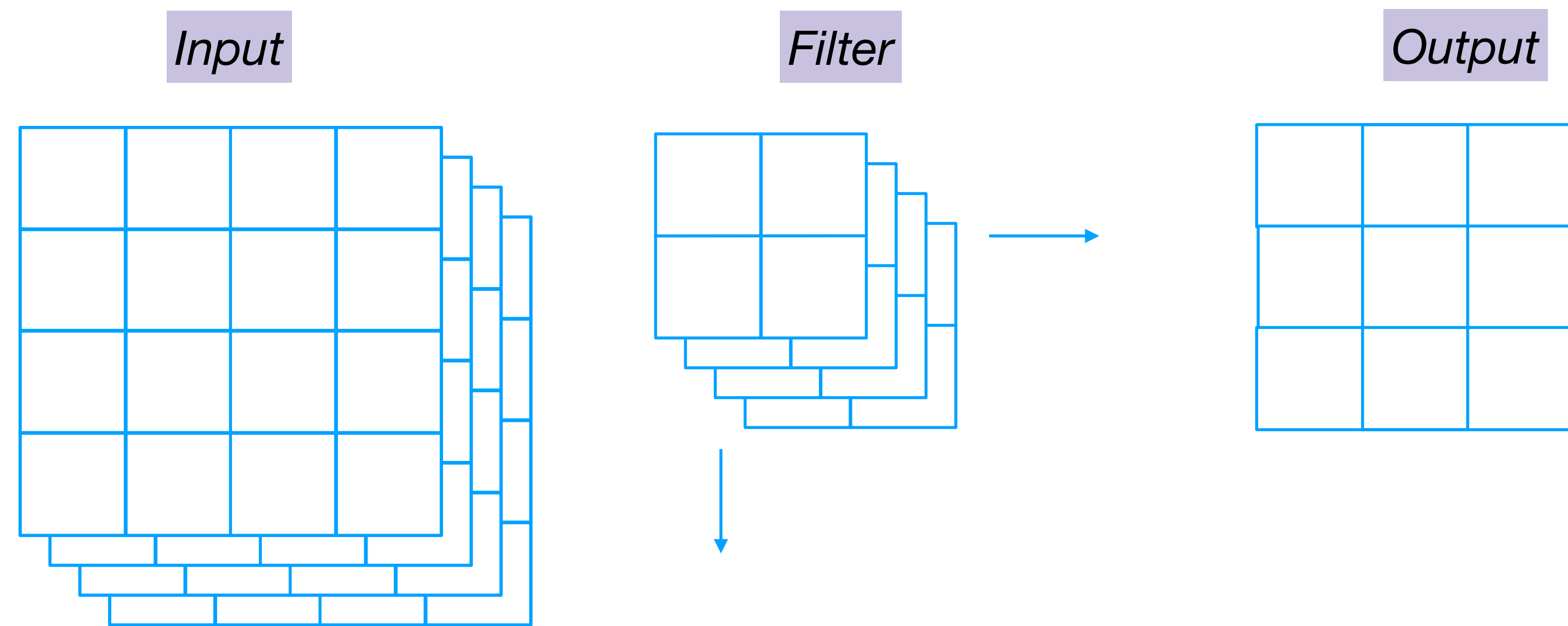
- Without channels, the only option is a **standard** convolution: filter has same number of modes as input
 - ▶ iterate over all positions in the input, placing the filter at each one and computing dot product
 - ▶ in this example, we're choosing the output to be the same size as the input (using padding or cyclic boundary conditions so that we don't have to skip filter placements near edges of input tensor)

Standard convolution (no padding)



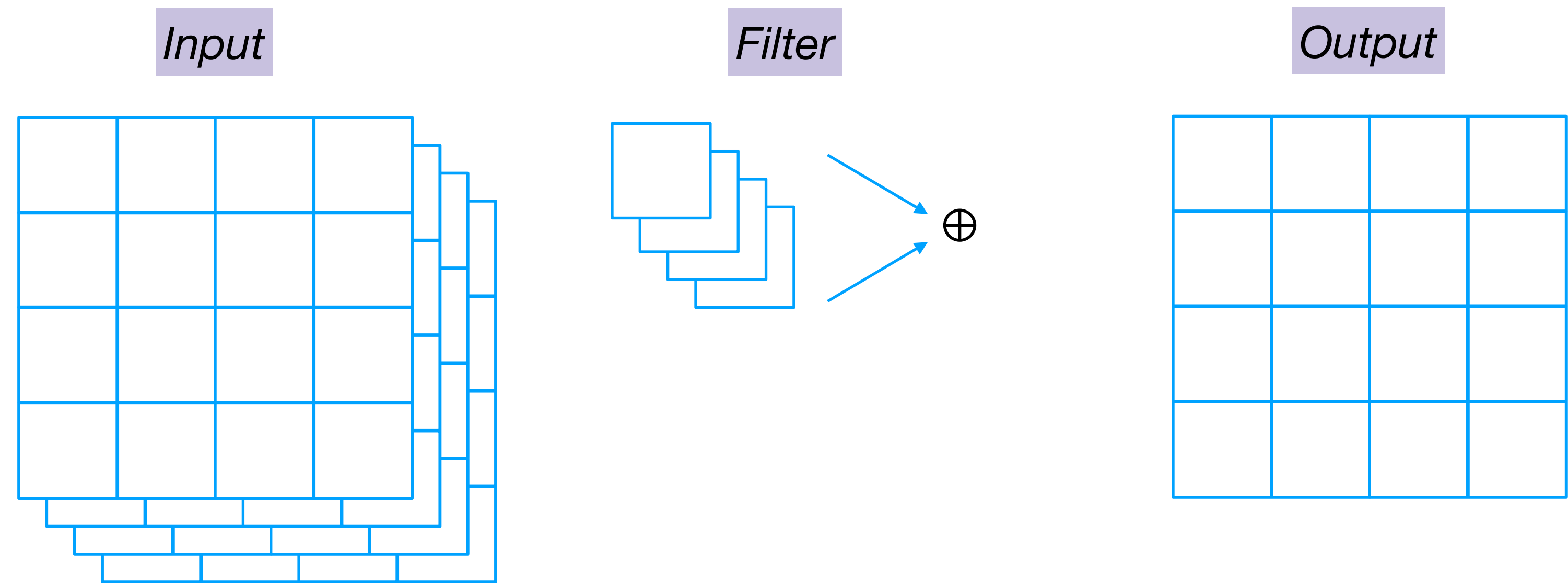
- If we use only **valid** positions (no padding or cyclic boundary), output is smaller than input
- Special case: if
$$\text{filter.shape}[-1] == \text{input.shape}[-1]$$
 - ▶ then last mode of output only has length 1
 - ▶ if desired we can drop this mode, reducing the number of modes of the output

Standard convolution (no padding)



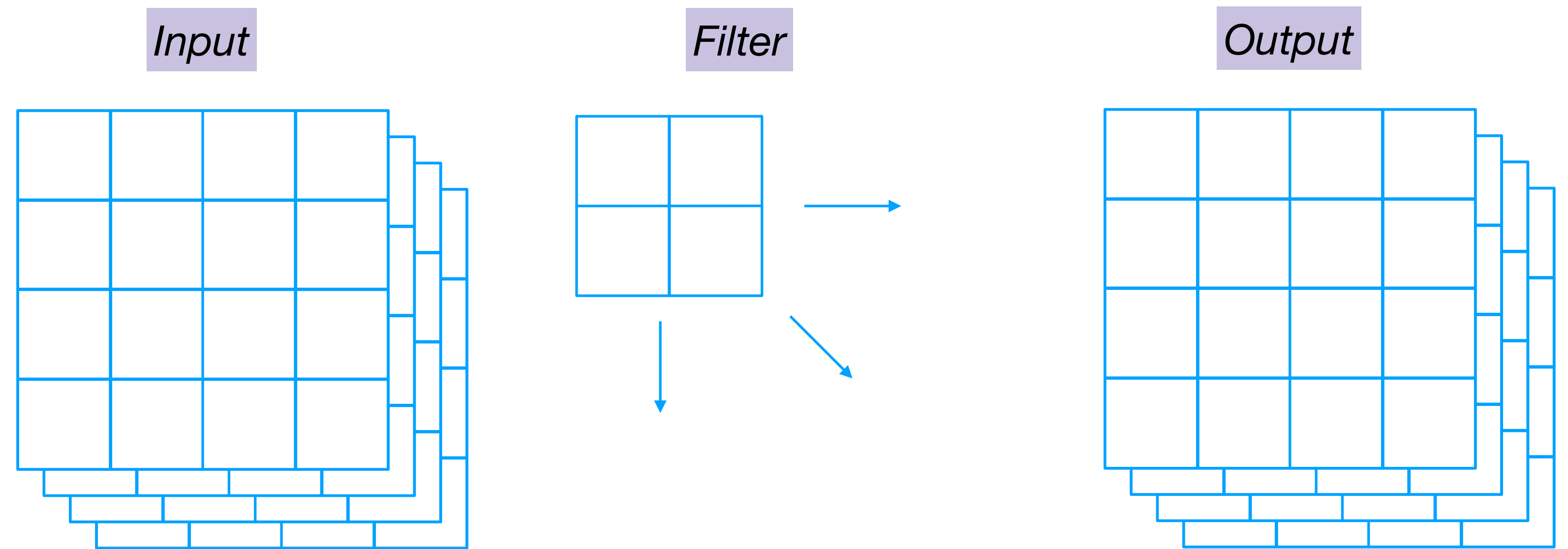
- If we use only **valid** positions (no padding or cyclic boundary), output is smaller than input
- Special case: if
$$\text{filter.shape}[-1] == \text{input.shape}[-1]$$
 - ▶ then last mode of output only has length 1
 - ▶ if desired we can drop this mode, reducing the number of modes of the output

Pointwise convolution



- Even specialer case: filter is $1 \times 1 \times \dots \times 1 \times \text{depth}$
 - ▶ keep all but one mode the same shape, lose last mode
- Called ***pointwise*** or (for 2d) 1×1 convolution
 - ▶ same as a weighted sum of slices / channels
- Makes a lot of sense if last mode is channel index: e.g., collapse RGB channels to get overall image intensity
 - ▶ common design: multi-head convolution adds a channel index, pointwise later removes it

Same filter on each channel

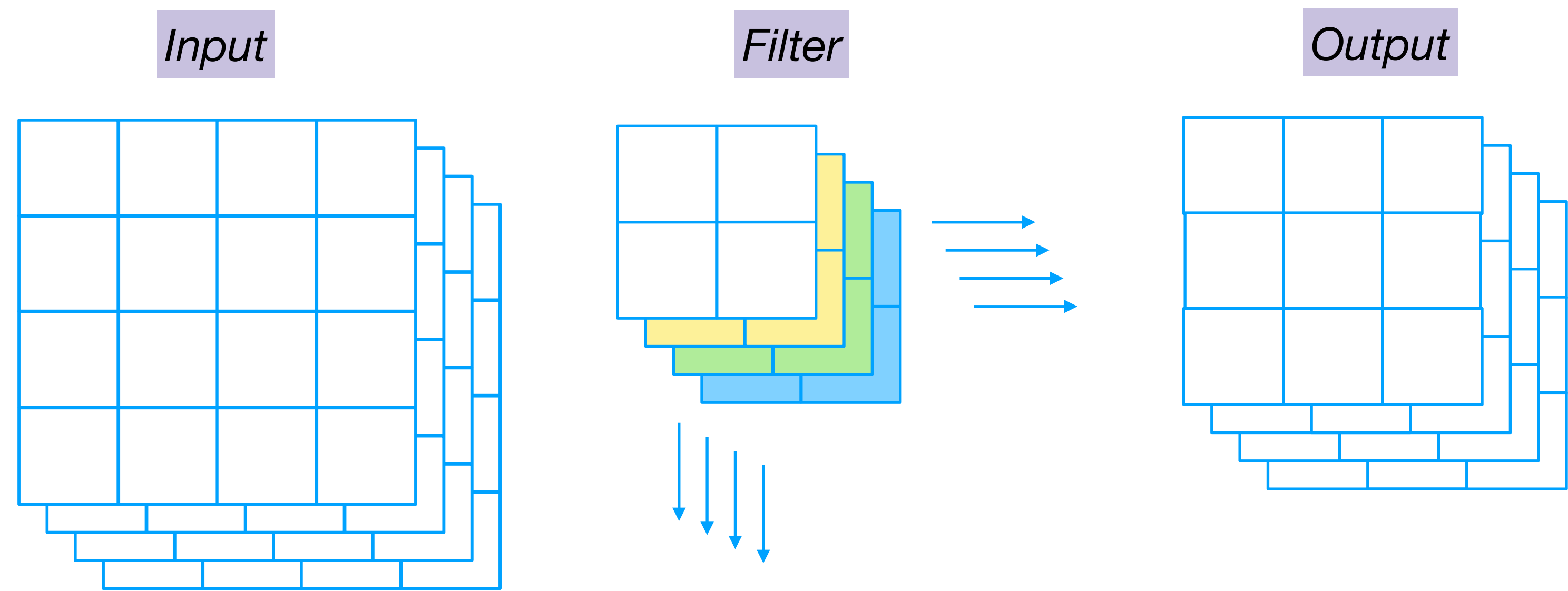


- If last mode is channel index, and if filter has just 1 channel, then standard convolution applies the same filter independently to each channel
 - ▶ e.g., blur RGB channels independently by applying a filter like

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

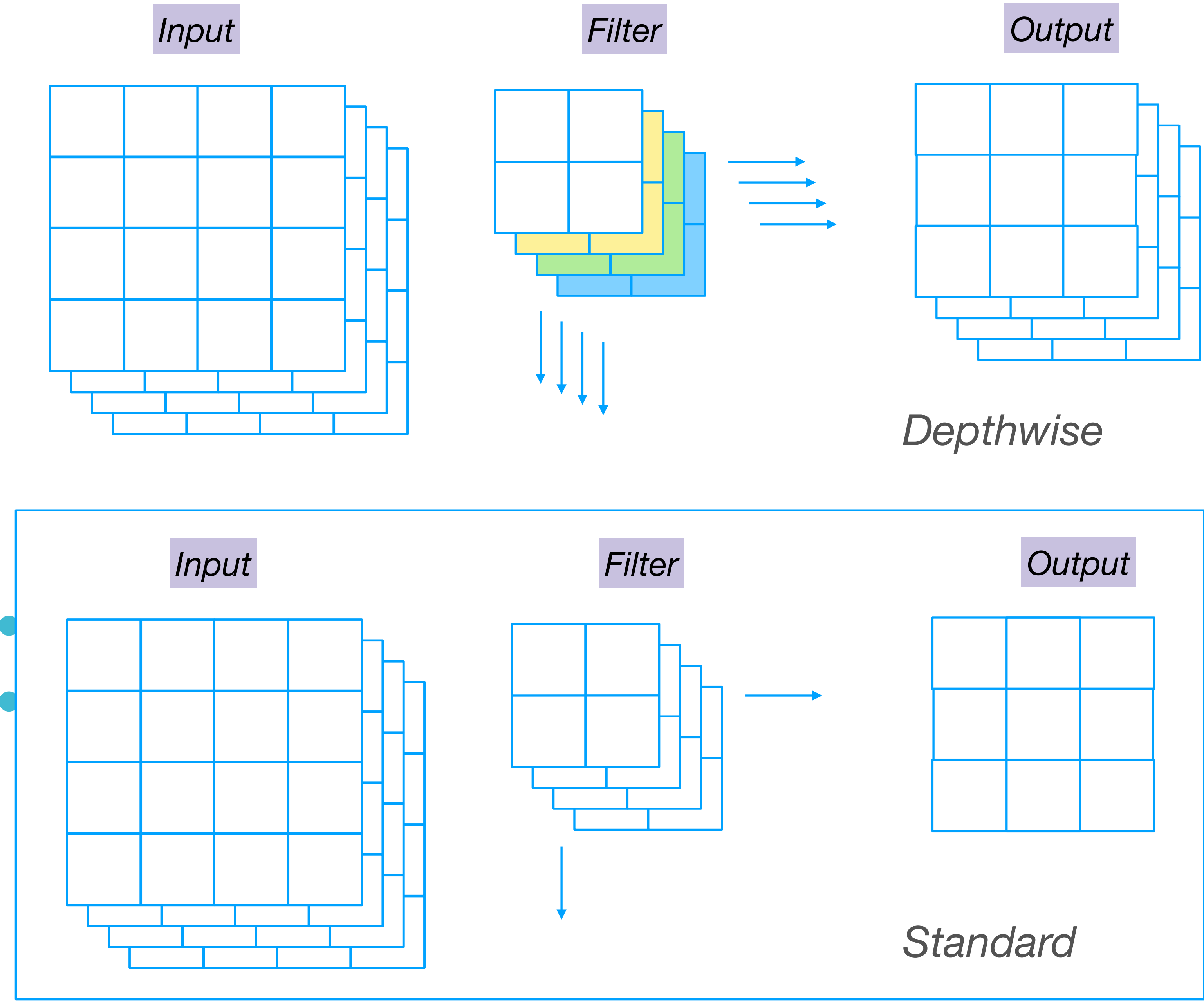
to each one

Depthwise convolution



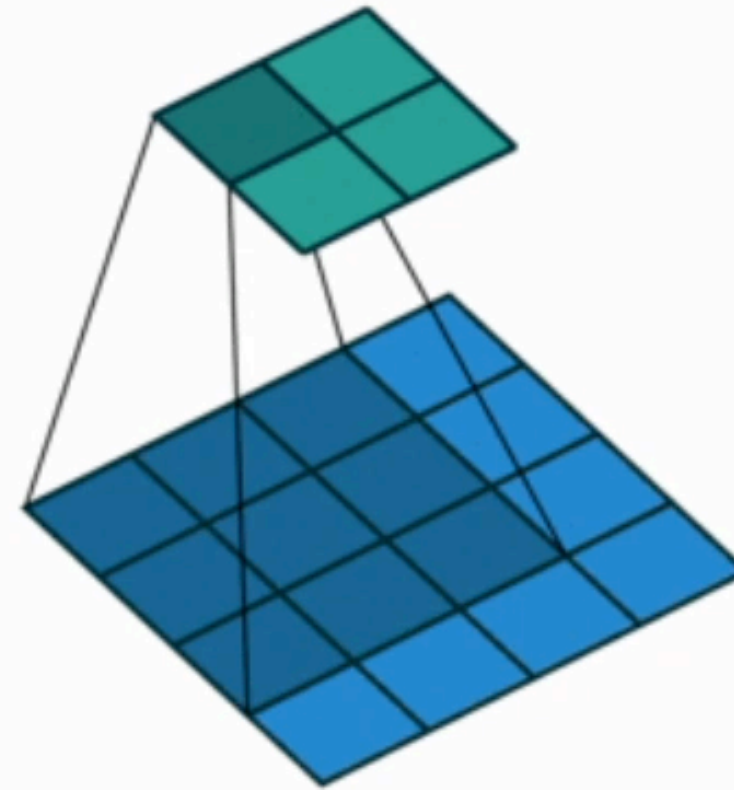
- Might also want to filter each channel separately
- **Depthwise:** filter has same number of channels as input, apply to each layer independently
 - ▶ this is *different* from a standard convolution: standard = depthwise followed by sum

Depthwise convolution

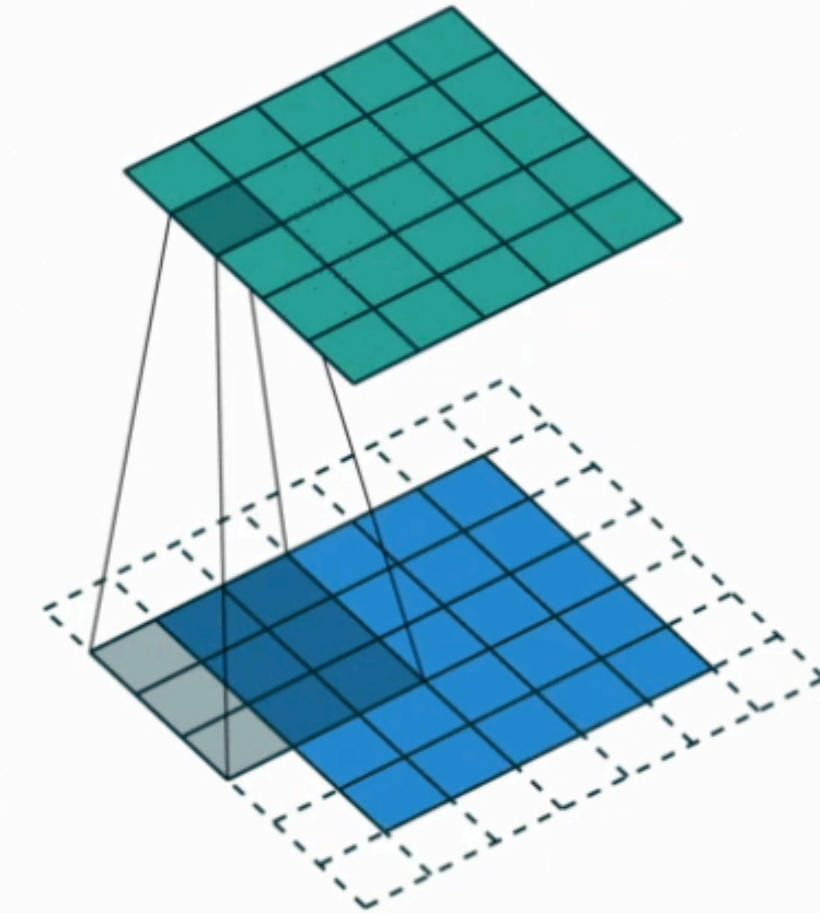


Visualizations

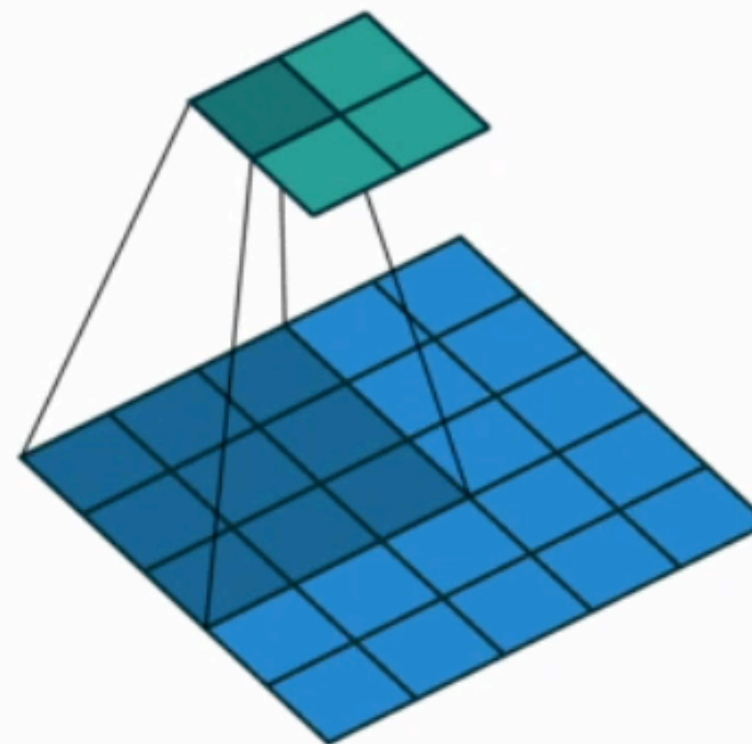
padding = 0, stride = 1



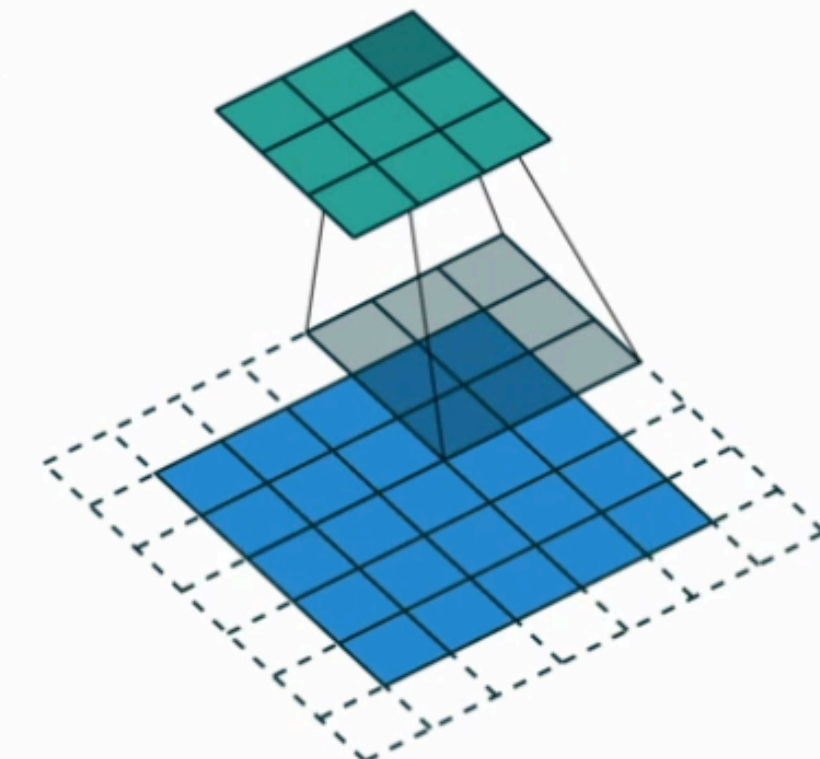
padding = 1, stride = 1



padding = 0, stride = 2

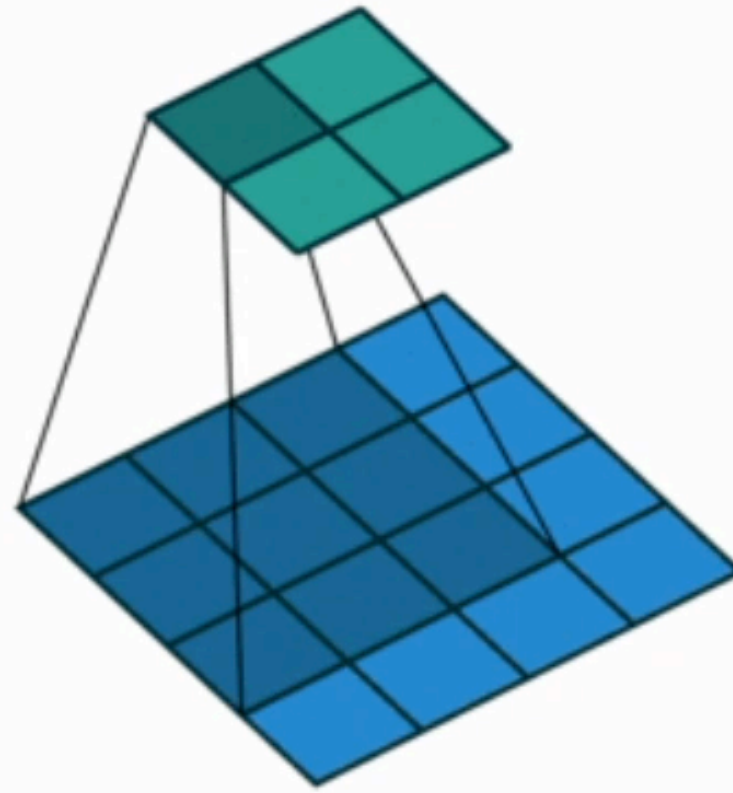


padding = 1, stride = 2

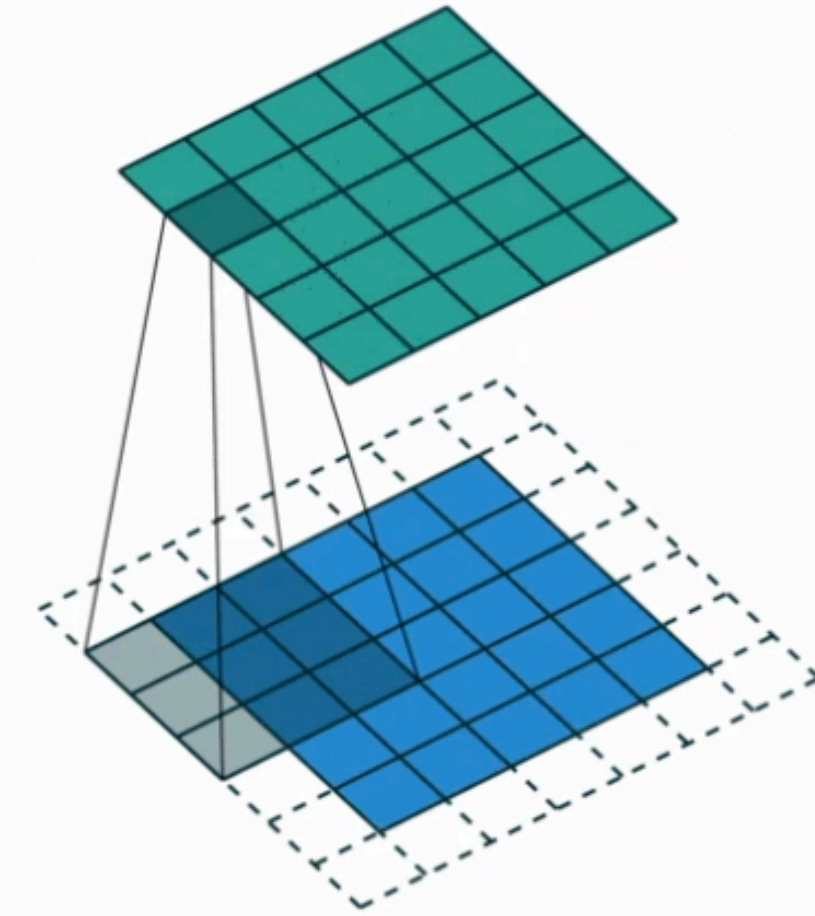


Visualizations

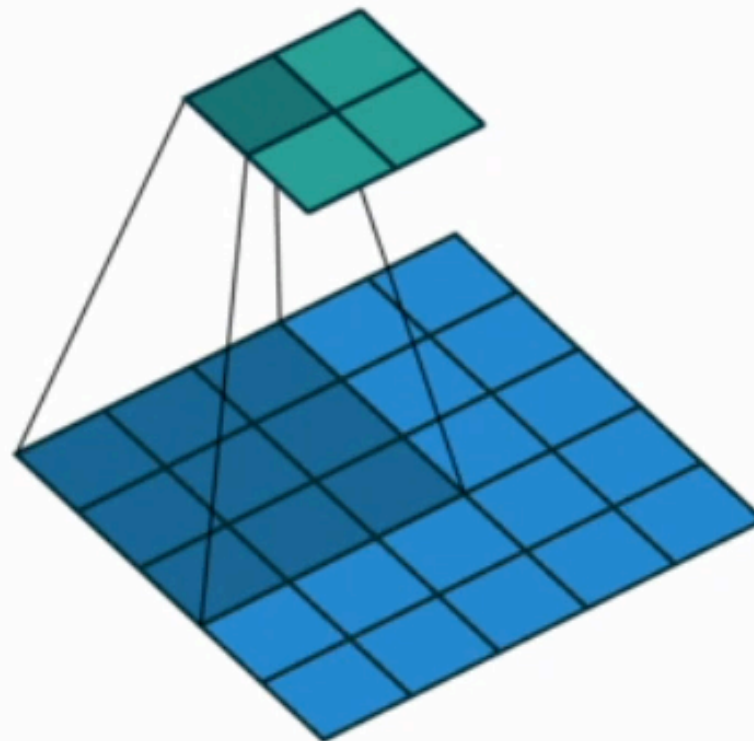
padding = 0, stride = 1



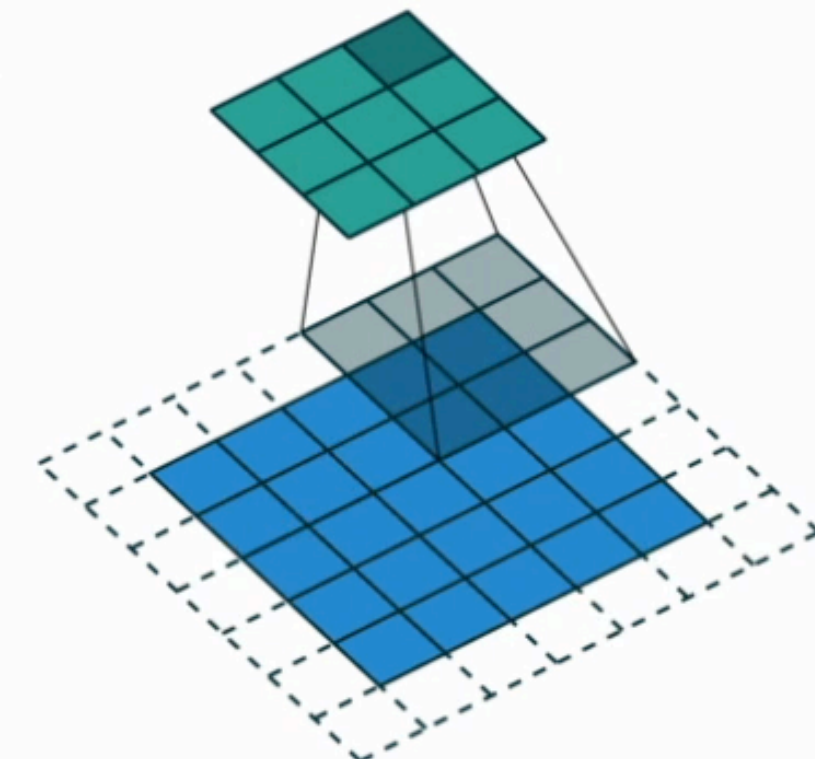
padding = 1, stride = 1



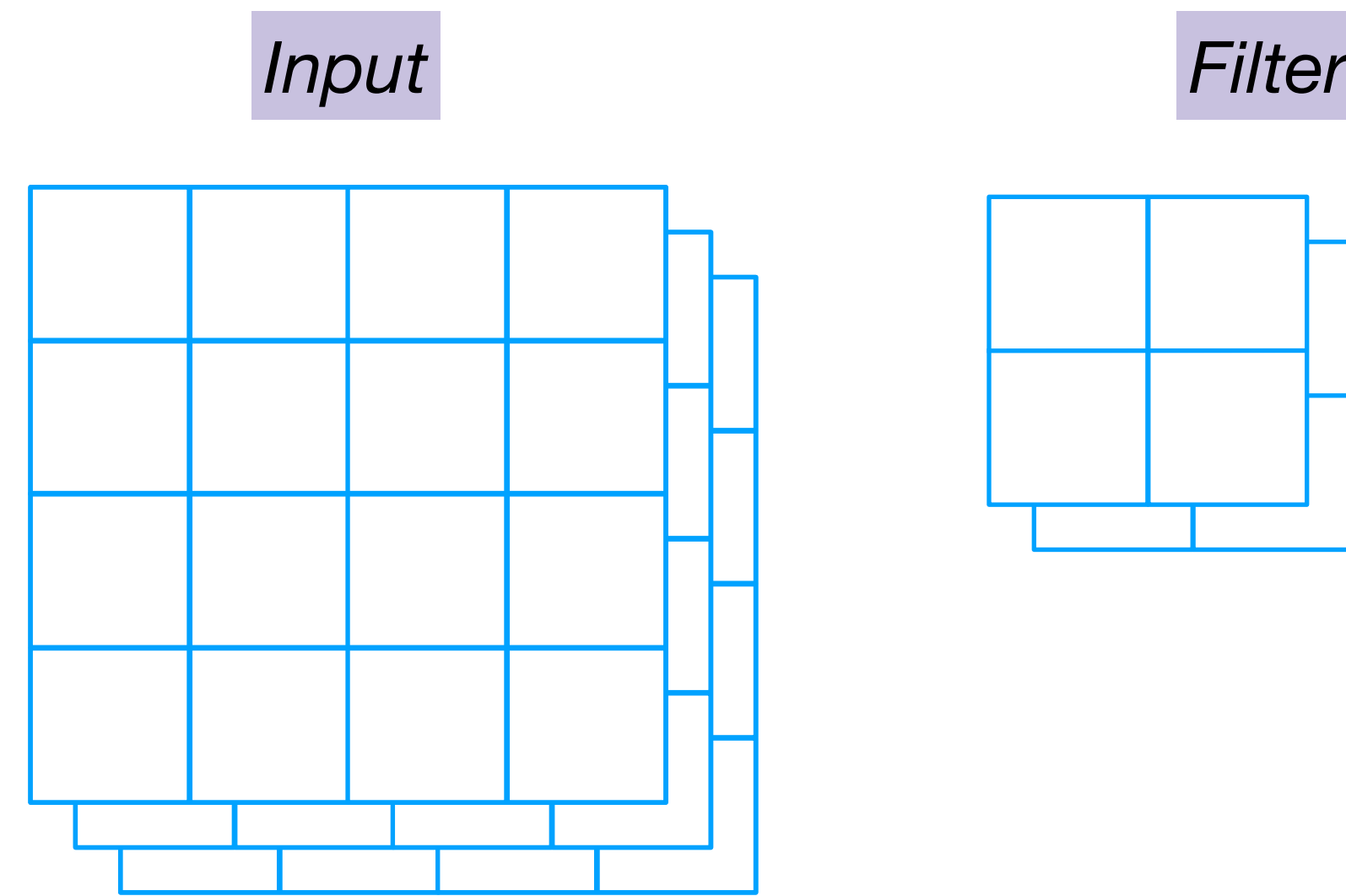
padding = 0, stride = 2



padding = 1, stride = 2



Calculating sizes



- With all the options, it can be tricky to figure out what size the output of a convolution will be
 - ▶ e.g., given $4 \times 4 \times 3$ input and $2 \times 2 \times 2$ filter
 - ▶ what size is output for a standard, valid convolution?
 - ▶ what if we zero-pad the last (depth) mode by 1 at end?
 - ▶ what if we use stride 2, 2, 1 (with no padding)?

Try it

- Suppose our input is $13 \times 13 \times 5$, our filter is $3 \times 3 \times 5$, and we compute only valid output positions. Suppose we zero-pad by 1 on all sides of the images (i.e., in the first 2 modes) and use a stride of 2 in all modes. What shape is the output?
 - A. $6 \times 6 \times 1$
 - B. $7 \times 7 \times 1$
 - C. $13 \times 13 \times 5$
 - D. $13 \times 13 \times 1$
 - E. $7 \times 7 \times 5$

Putting it together

- Convolution and pooling work well together: tunable between equivariance and invariance
- Often do convolutions alternating with nonlinearities, occasional pooling to reduce size of a layer, occasional multiple heads or pointwise convolutions to change number of modes of tensor
- Sometimes increase number of channels as we downsample (else we force the net to discard a lot of information)
- Can use vec or unvec to glue between convolutional layers and plain MLP layers
- Common pattern: more smaller channels as we go deeper (using multi-head convolution, subsampling by pool or stride)
- Add in residual connections to help gradients not explode or vanish

Interactive example

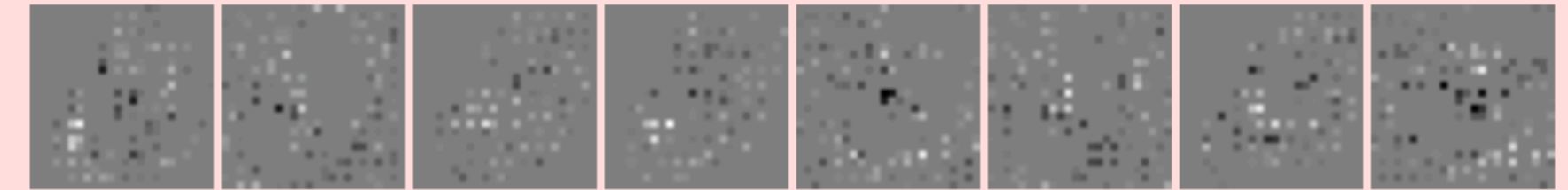
<https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html>

conv (24x24x8)
filter size 5x5x1, stride 1
max activation: 3.60709, min: -2.93768
max gradient: 0.14545, min: -0.15263
parameters: $8 \times 5 \times 5 \times 1 + 8 = 208$

Activations:



Activation Gradients:



Weights:

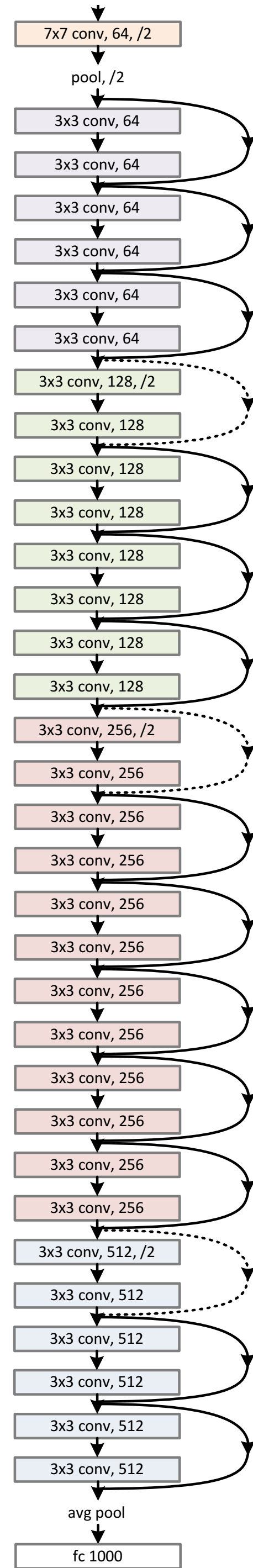
(■)(■)(■)(■)(■)(■)(■)(■)

Weight Gradients:

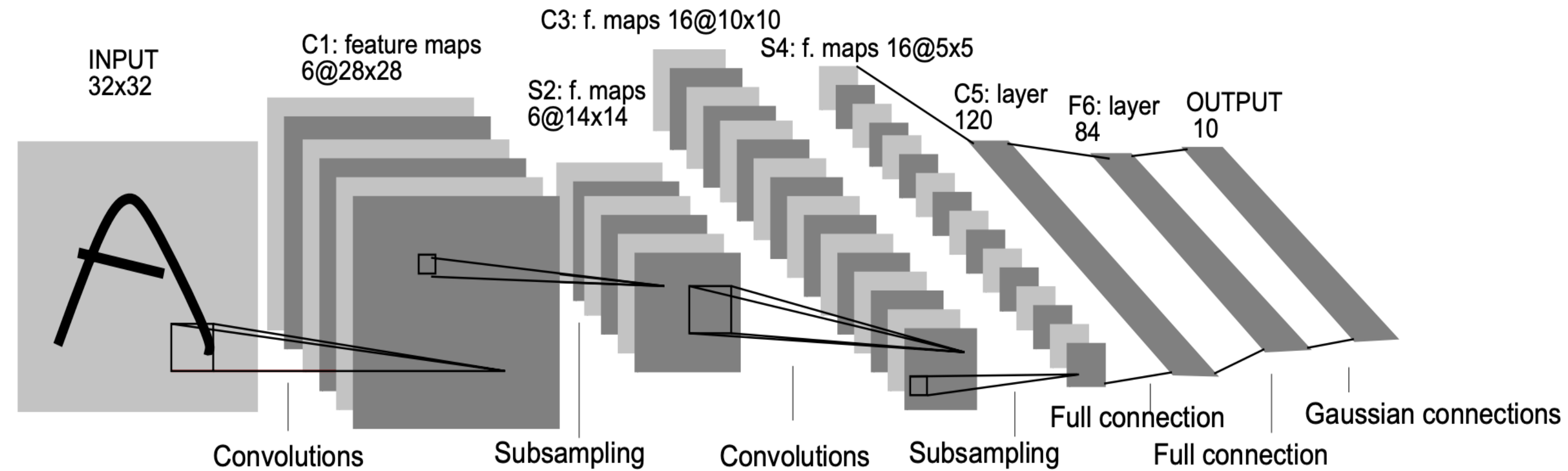
(■)(■)(■)(■)(■)(■)(■)(■)

- CNN for MNIST, trained by Javascript in browser
- $2 \times (\text{convolution} + \text{ReLU} + \text{max-pool}) + \text{FC} + \text{softmax}$
- Enough for 99% validation accuracy

Lots of well-known convolutional networks



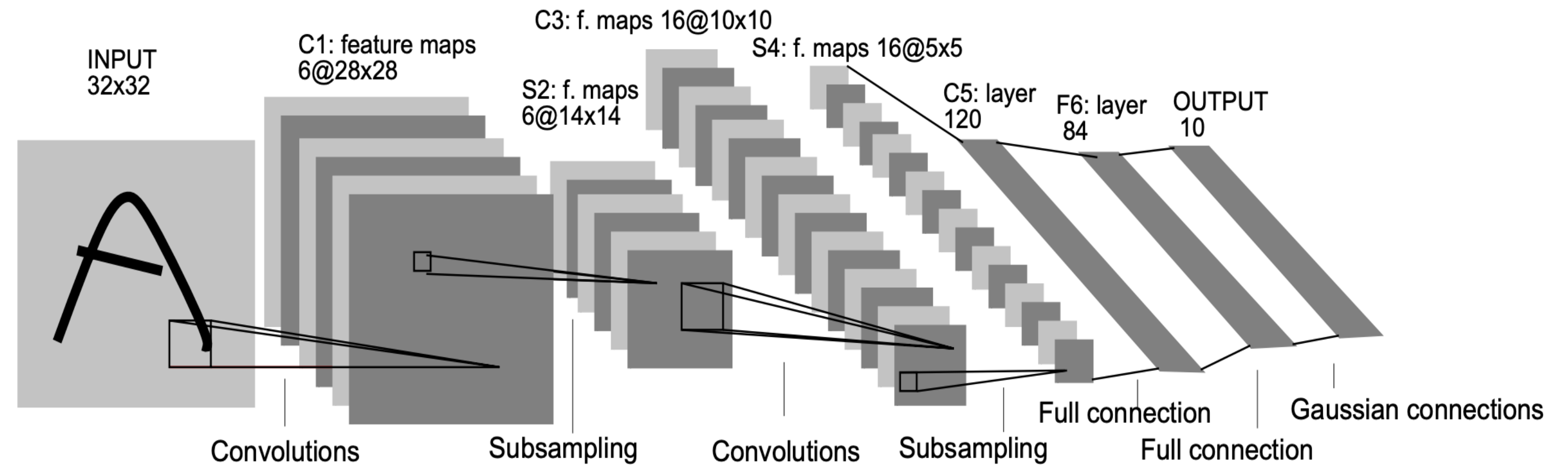
[He et al., 2015]



[LeCun et al., 1998]

- These features have appeared in *many* famous models (SOTA at their time)
 - ▶ e.g., LeNet (convolution and pooling, for MNIST digit classification)
 - ▶ e.g., ResNet (convolution, pooling, batch norm, residual connections, for ImageNet)

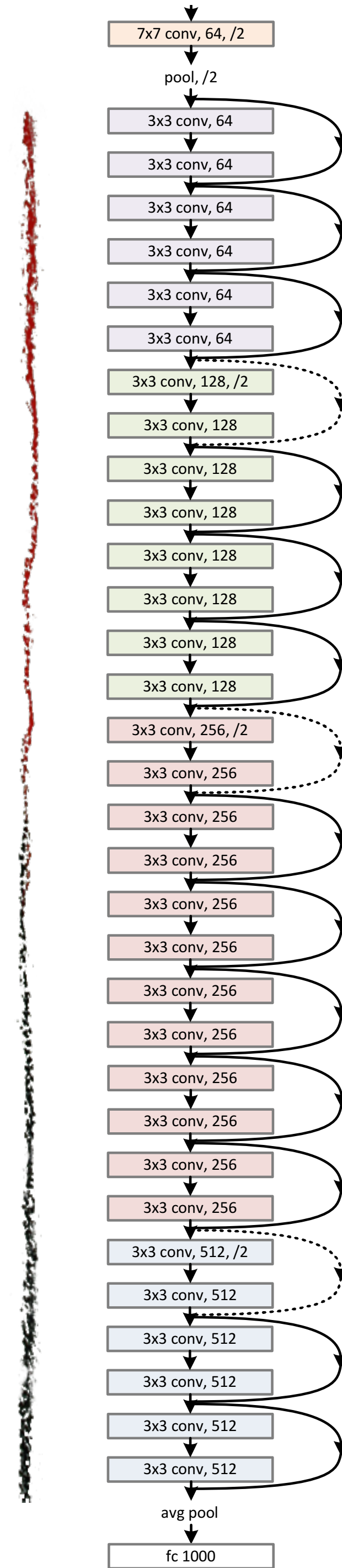
LeNet



[LeCun et al., 1998]

- Convolutions are 5×5 with no padding
- Subsampling is average-pooling followed by a pointwise linear function and nonlinearity
- Nonlinearities are all tanh, symmetric version of sigmoid
- C3 uses a hand-designed sparse connection pattern
- After S4, we vec the $5 \times 5 \times 16$ tensor, use MLP layers
- Final nonlinearity is not commonly used any more: it unvecs the F6 activation to a 12×7 image and compares to 10 digit templates using Gaussian RBFs

ResNet-34



- 3×3 conv, 128, /2 means a 3×3 convolution with stride 2, 128 filters
- At the top is a max-pool, and at the bottom we pool by averaging each channel to a single number
- Residual connections throughout
 - ▶ dashed ones incorporate downsampling of the identity path via a 1×1 convolution of stride 2

Learning objectives— CNNs

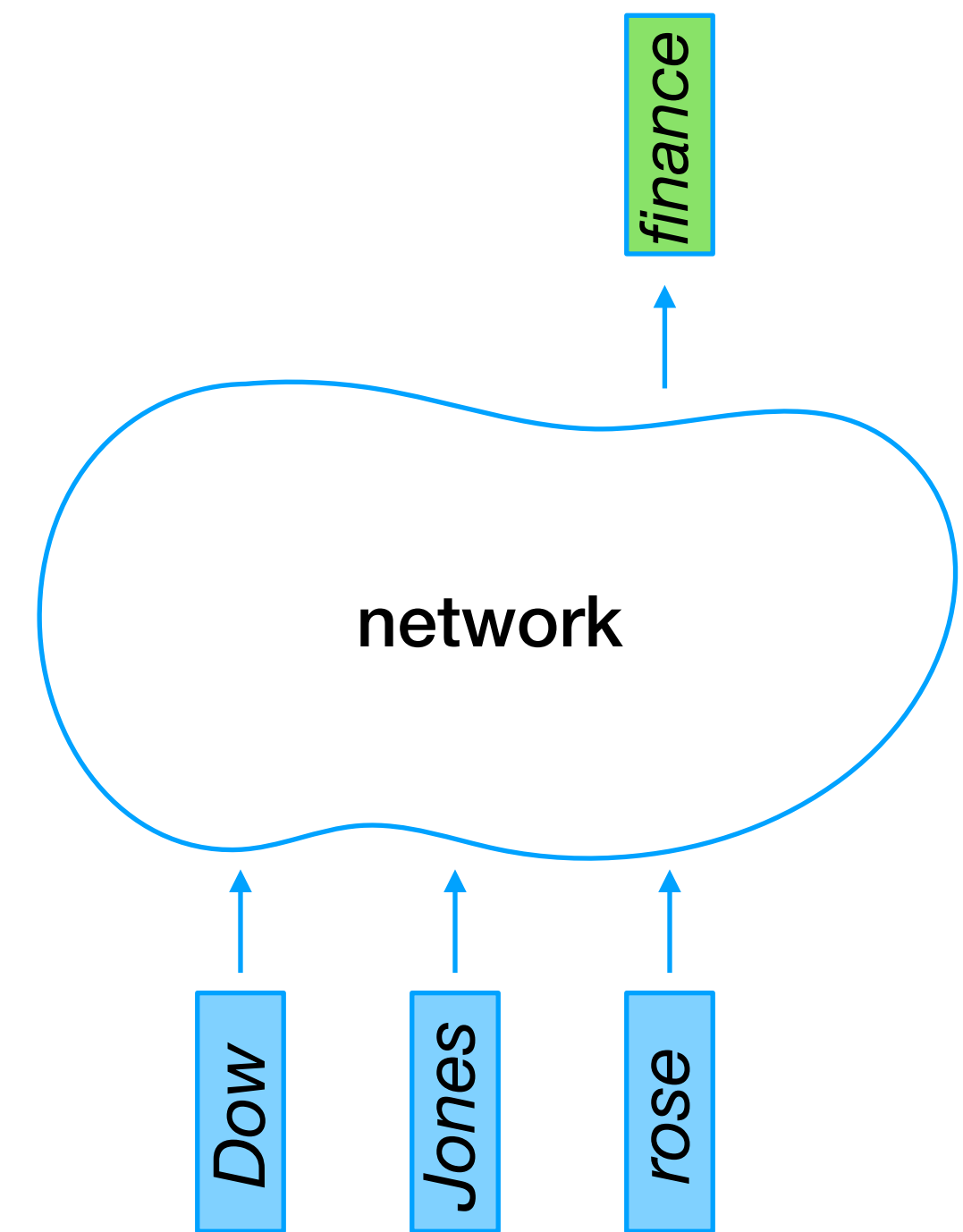
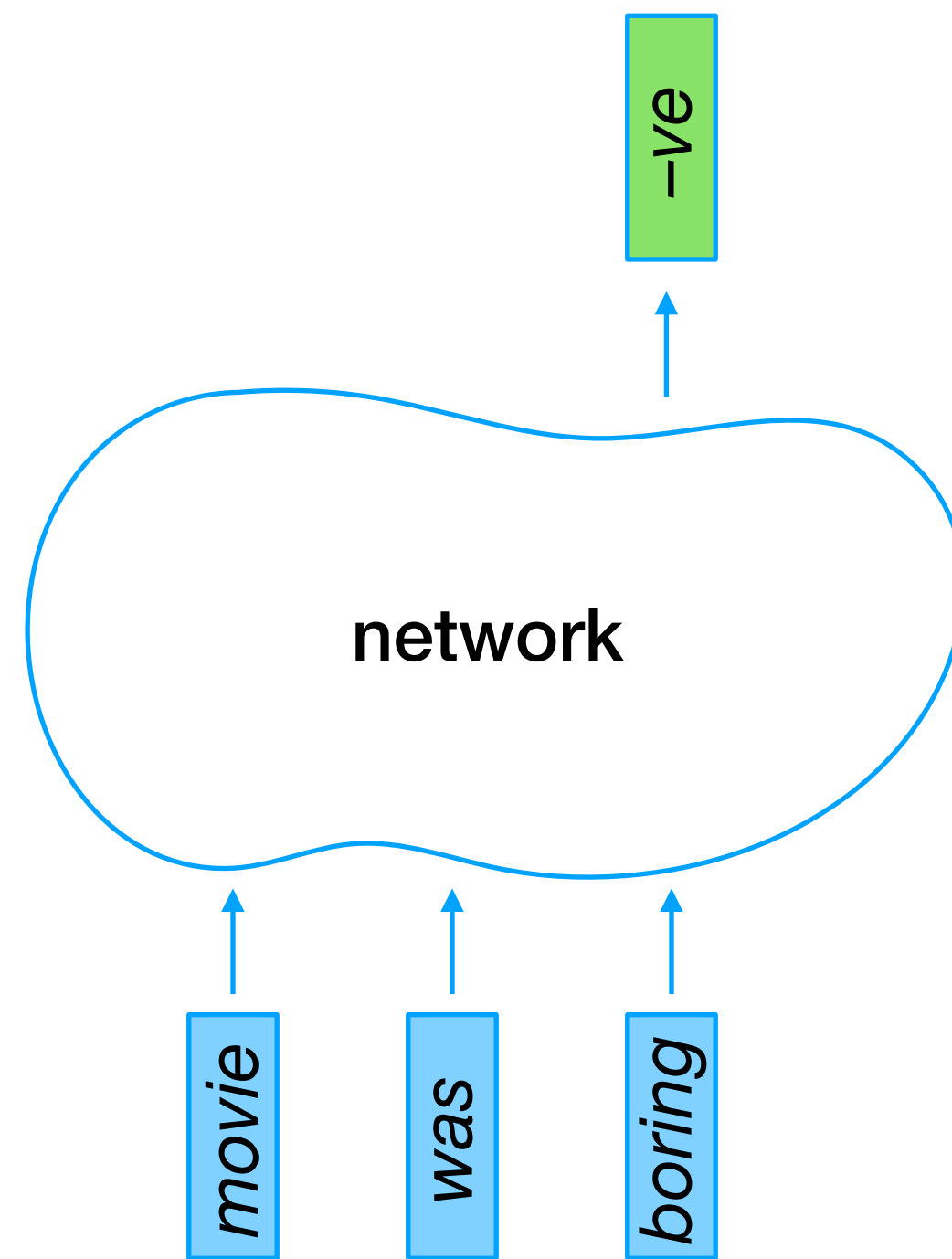
- What is a (continuous time or space) signal? How does it relate to a (discrete time or space) sampled vector, matrix, or tensor?
- What are important properties of convolutional layers? (invariance/equivariance, few parameters, fast forward/backward pass)
- Explain variations: output size, padding, dilation, stride
- What kinds of filters can convolution+nonlinearity represent? (blur, sharpen, edge/corner detection, band-pass, ...)
- What kinds of operations control layer shape and size? (pooling, vec/unvec, multiple heads, 1×1 convolution)
- Explain tensor indexing, broadcasting, elementwise notation, Einstein summation, representation of derivatives
- Carry out (small) tensor operations and convolutions by hand, including predicting the shape of output tensor

Recurrent networks



10-701 Introduction to Machine Learning
Geoff Gordon

Sequence prediction



- Models up to now have *fixed-shape* inputs
- Many language tasks (like sentiment classification, topic classification) take *variable-length* sequence as input
- Need models that can scale to fit

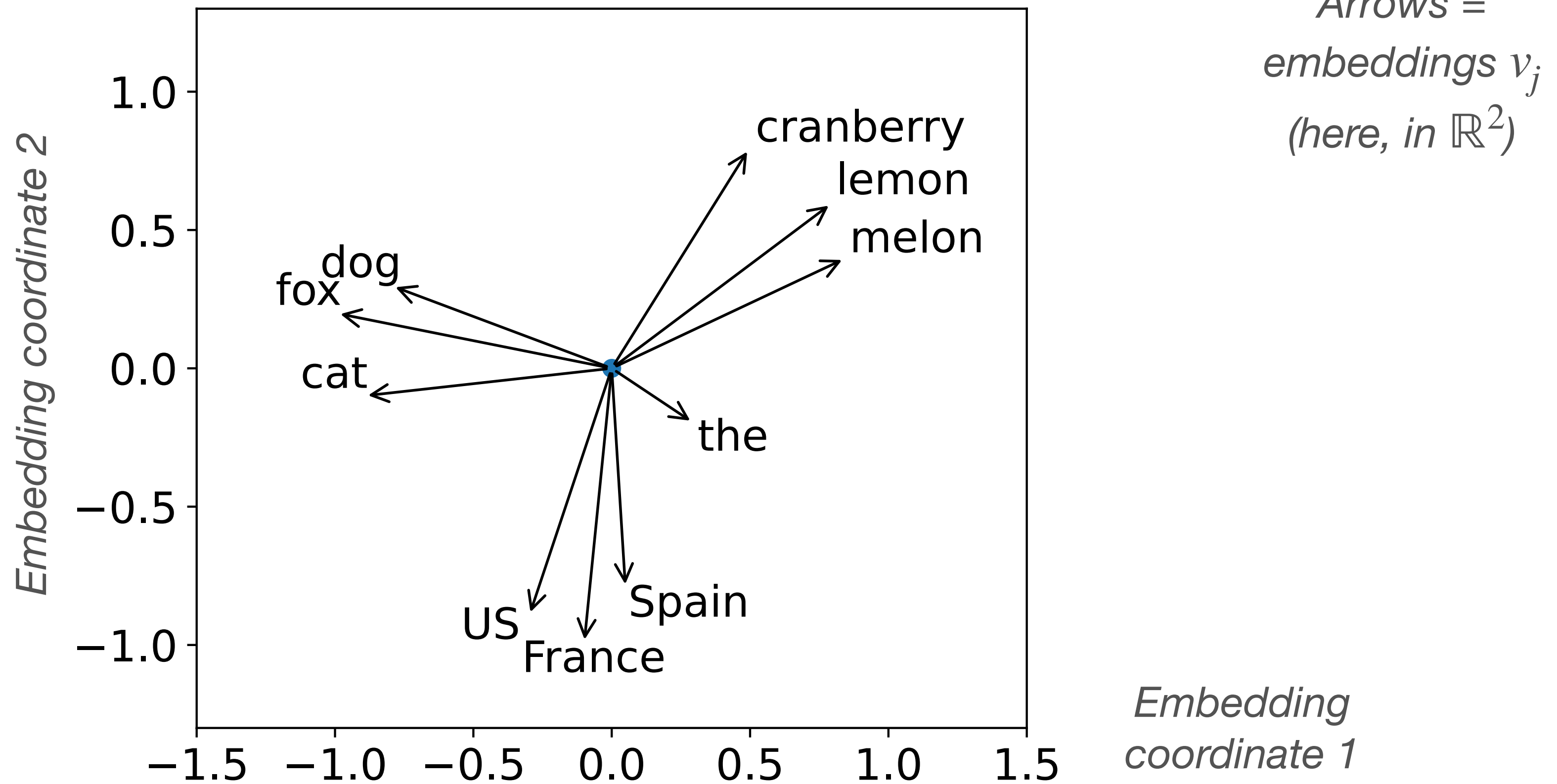
Discrete- valued sequences

To be or not to be, that is the question

One nation indivisible

- Another way language differs: **discrete** tokens instead of continuous pixel values
 - ▶ characters
 - ▶ words
 - ▶ word parts
 - ▶ class labels — potentially 1000s with relationships among them

Embedding

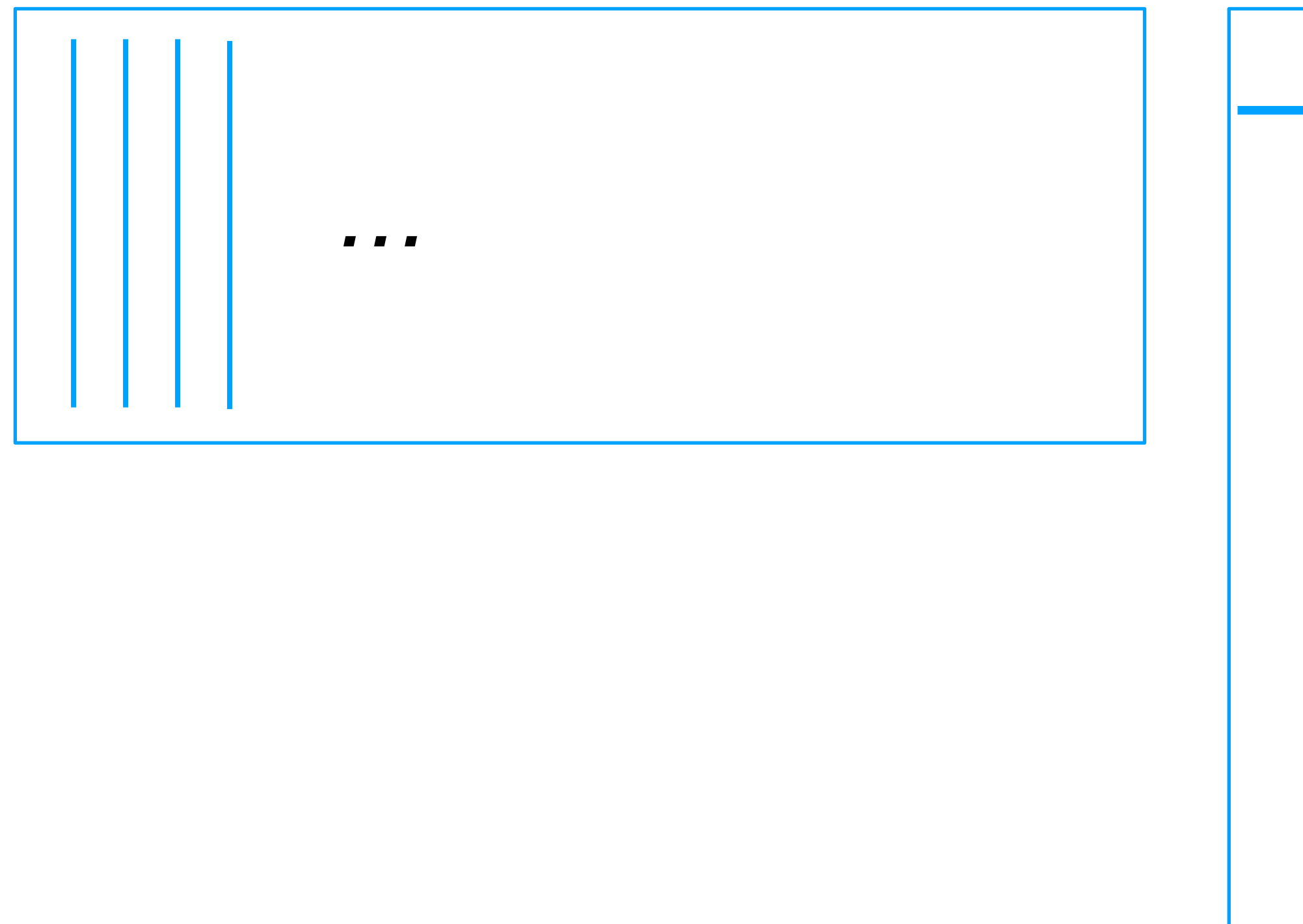


- Handle tokens by *embedding*: replace by vector $\in \mathbb{R}^d$ (d might be 512, 2048, 10k — large but $\ll$ vocabulary size)
 - ▶ ideally: relationships among tokens reflected in vectors
 - ▶ dog and cat embedded near each other
 - ▶ Paris – France + Spain $\approx$ Madrid

Learning embeddings

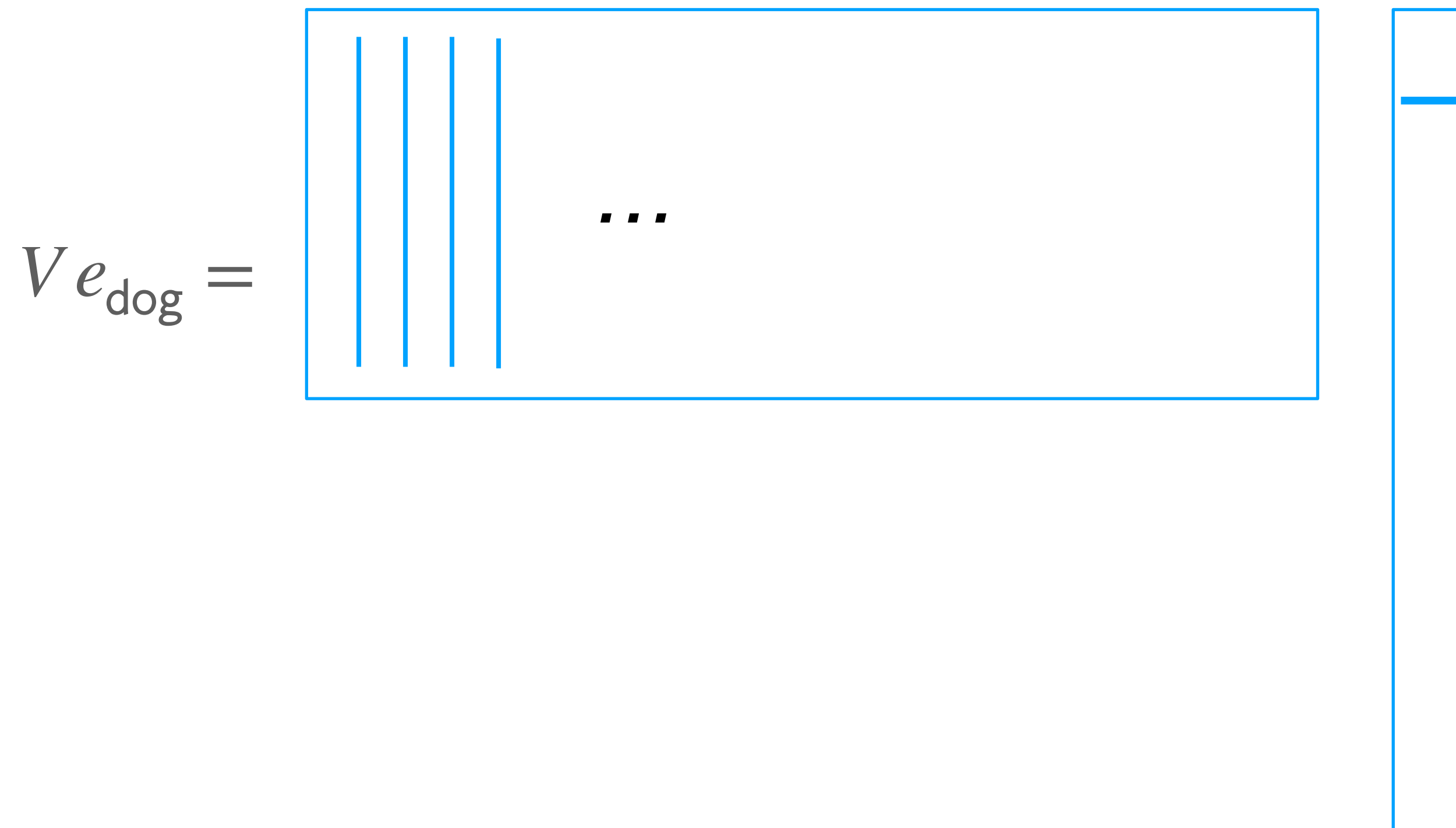
- If each raw token = one-hot vector, and each embedded token = arbitrary vector, this is a ***linear map***
- $\mathbb{R}^{\text{vocabulary size}} \rightarrow \mathbb{R}^{\text{embedding dimension}}$
- Embedding of “dog” = Ve_{dog}

$$Ve_{\text{dog}} =$$

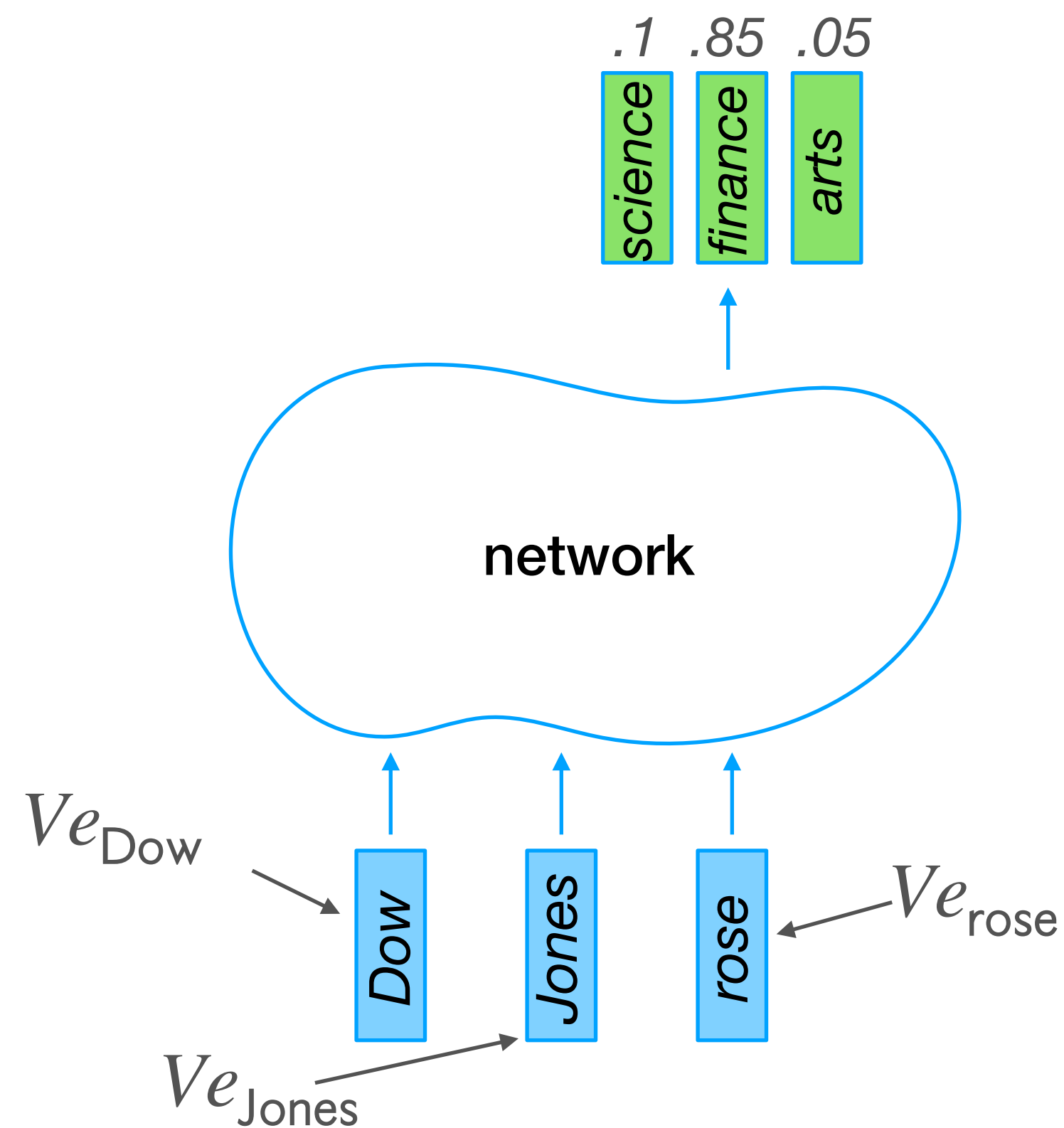


Learning embeddings

- If we present tokens as one-hot (for a neural net model in any task), first layer automatically learns embedding $V = W_1$
- SGD step updates embeddings of current tokens to make current prediction more accurate
- We'll talk later about more ways to learn embeddings

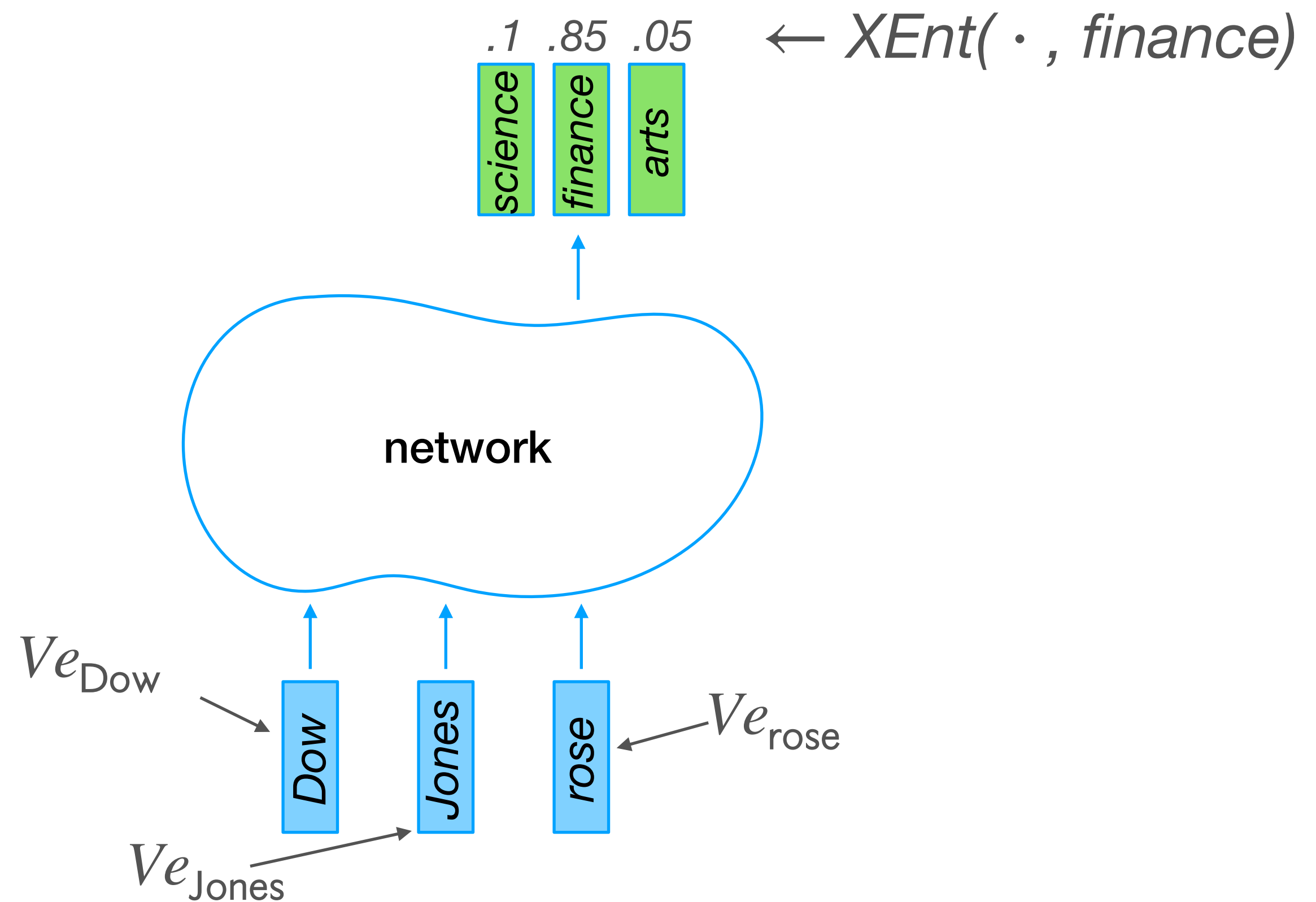


Sequences of tokens



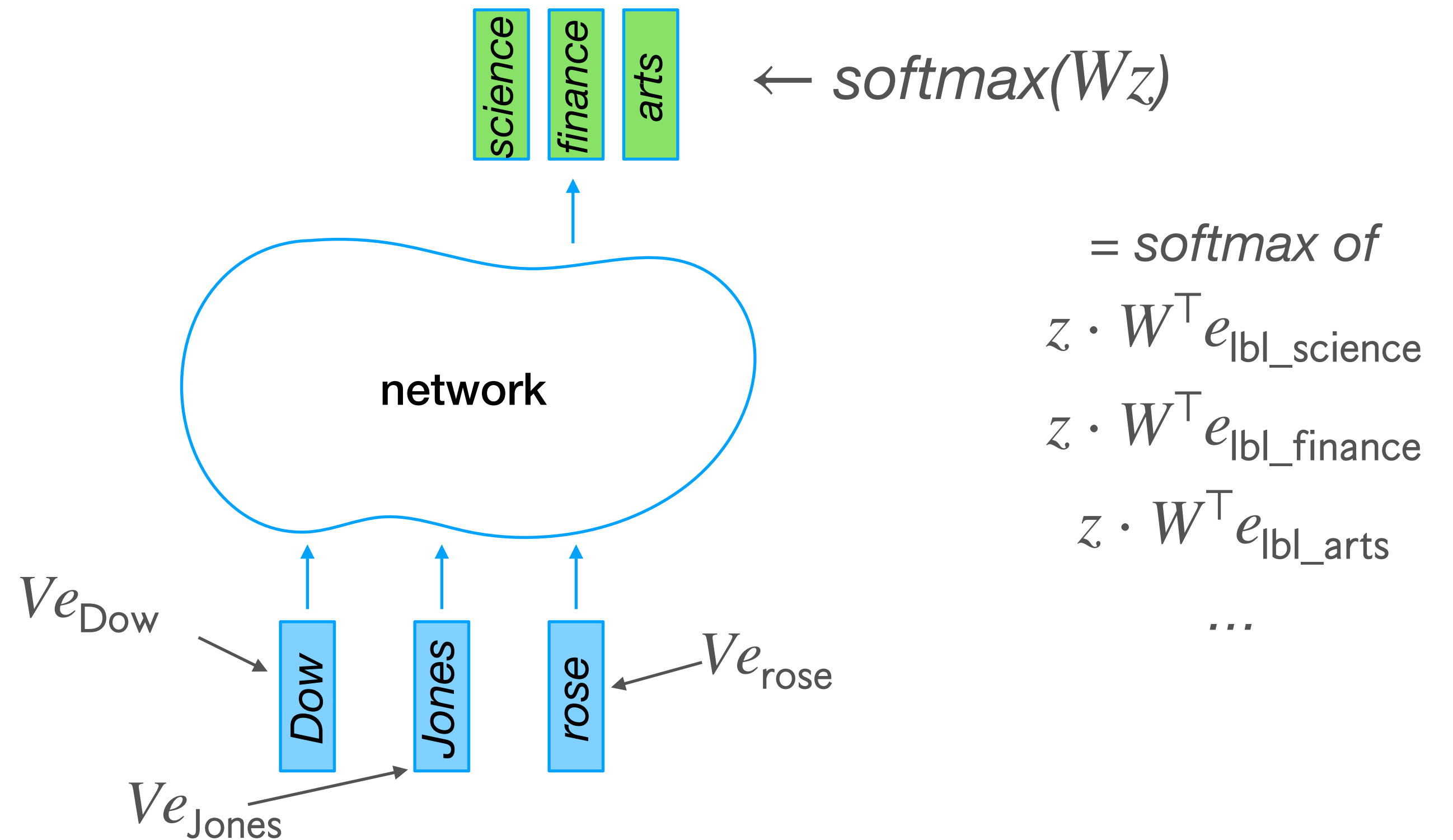
- Putting it together:
 - ▶ first encode the input as a sequence of tokens
 - ▶ then embed each token
 - ▶ then pass tokens to the model one by one
 - ▶ finally read off predictions at output node

Train



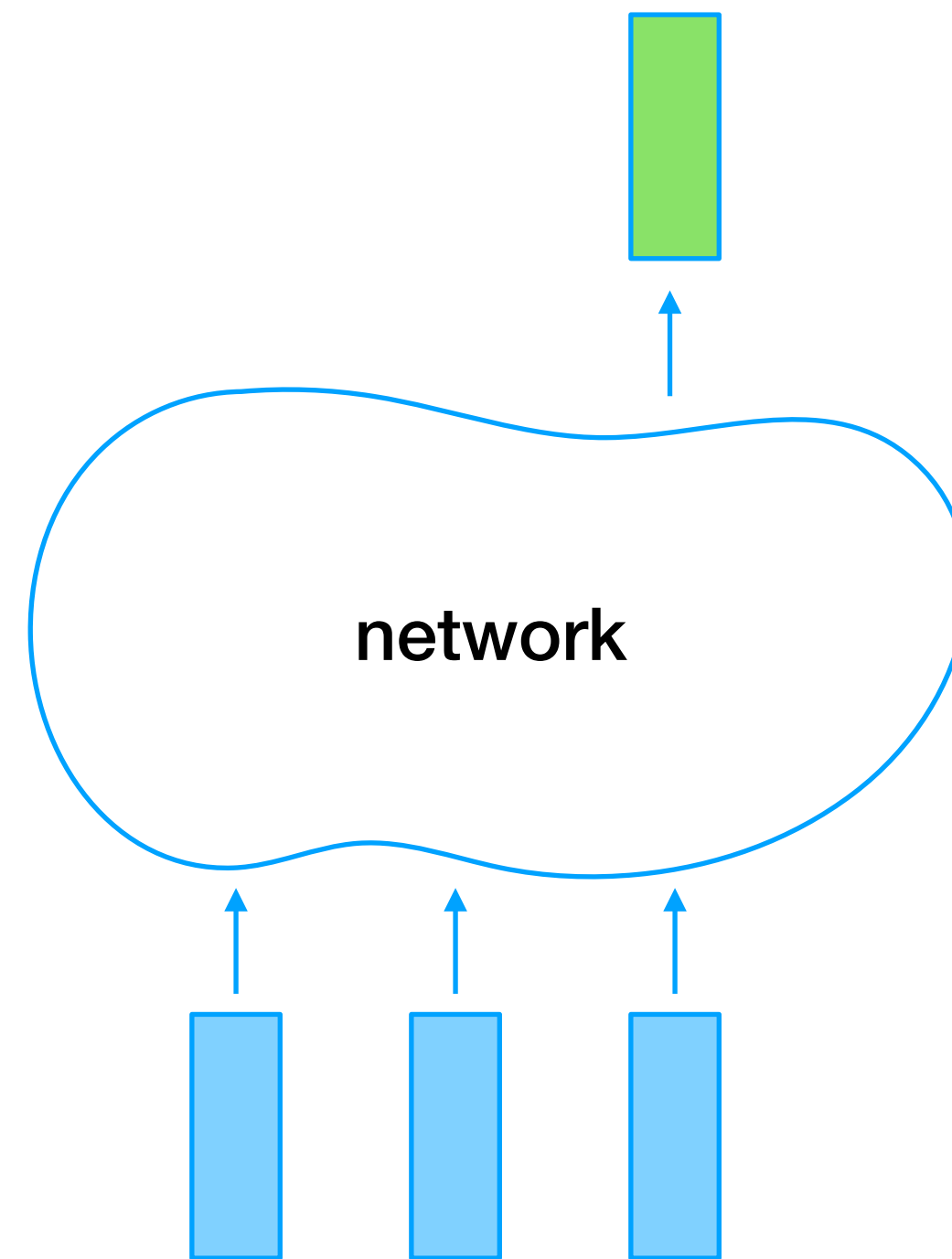
- Apply loss to prediction (maybe target was *finance*)
- Backprop through network
- Use gradient to update parameters (SGD, Adam, ...)
 - ▶ including embeddings of tokens that appeared in input

Class label embeddings

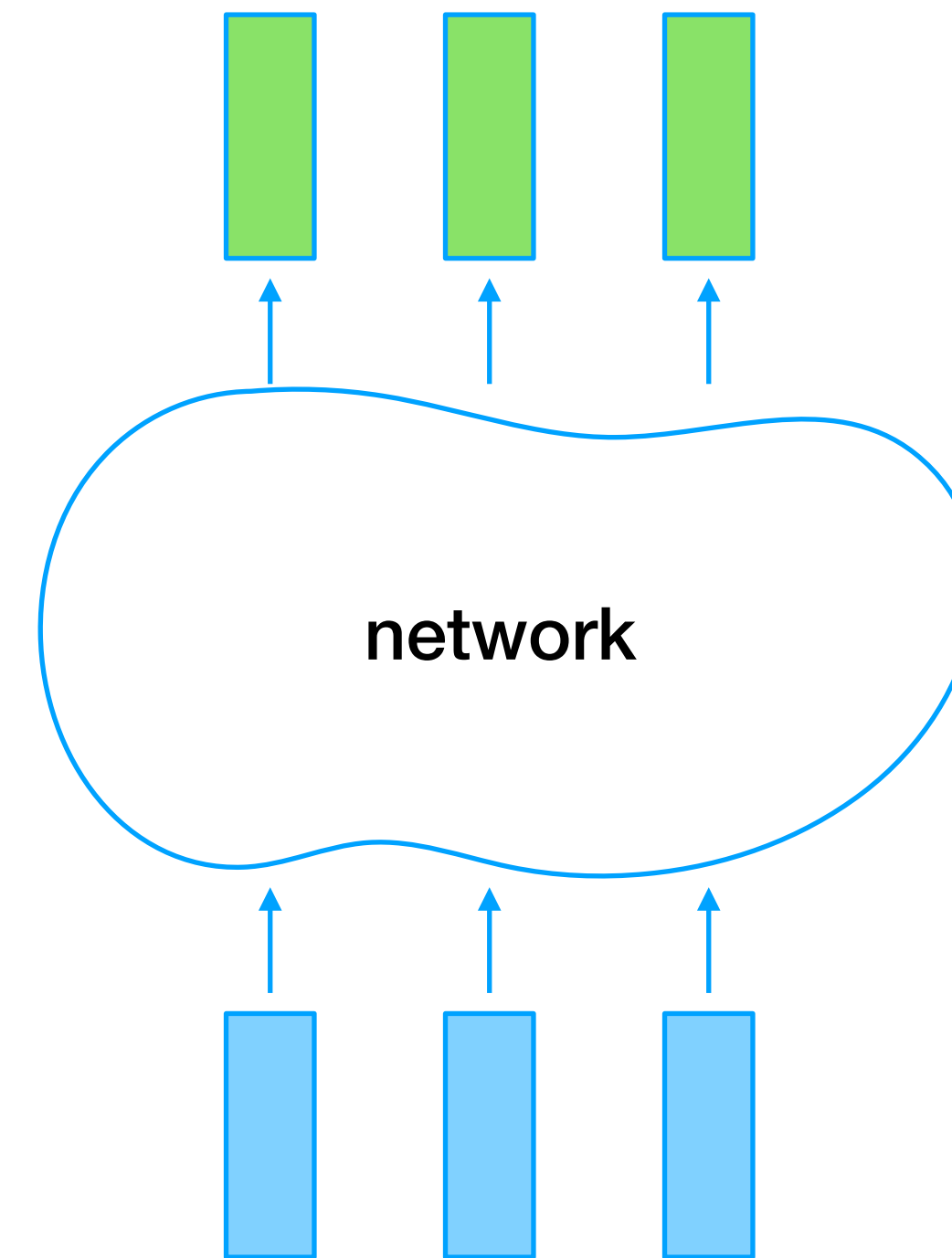


- Can even interpret class labels as tokens with embeddings
 - ▶ if penultimate layer activations are z
 - ▶ and last layer is $\text{softmax}(Wz)$
 - ▶ then we can think of i th row of W as embedding of i th label
 - ▶ and we put highest probability on the label whose embedding has highest dot product with z

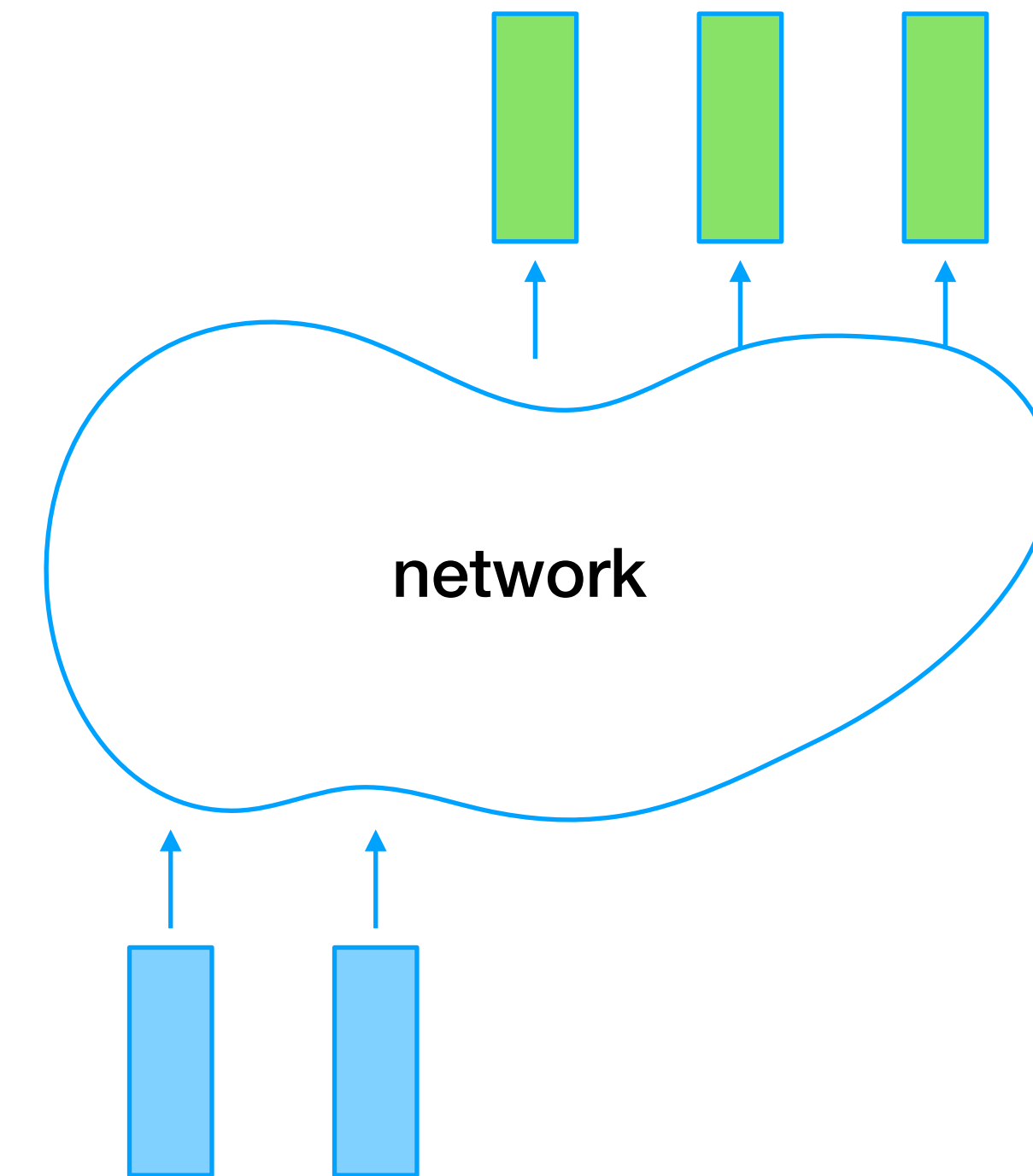
Sequence prediction tasks



*many → one
classification, regression
(e.g., sentiment analysis)*



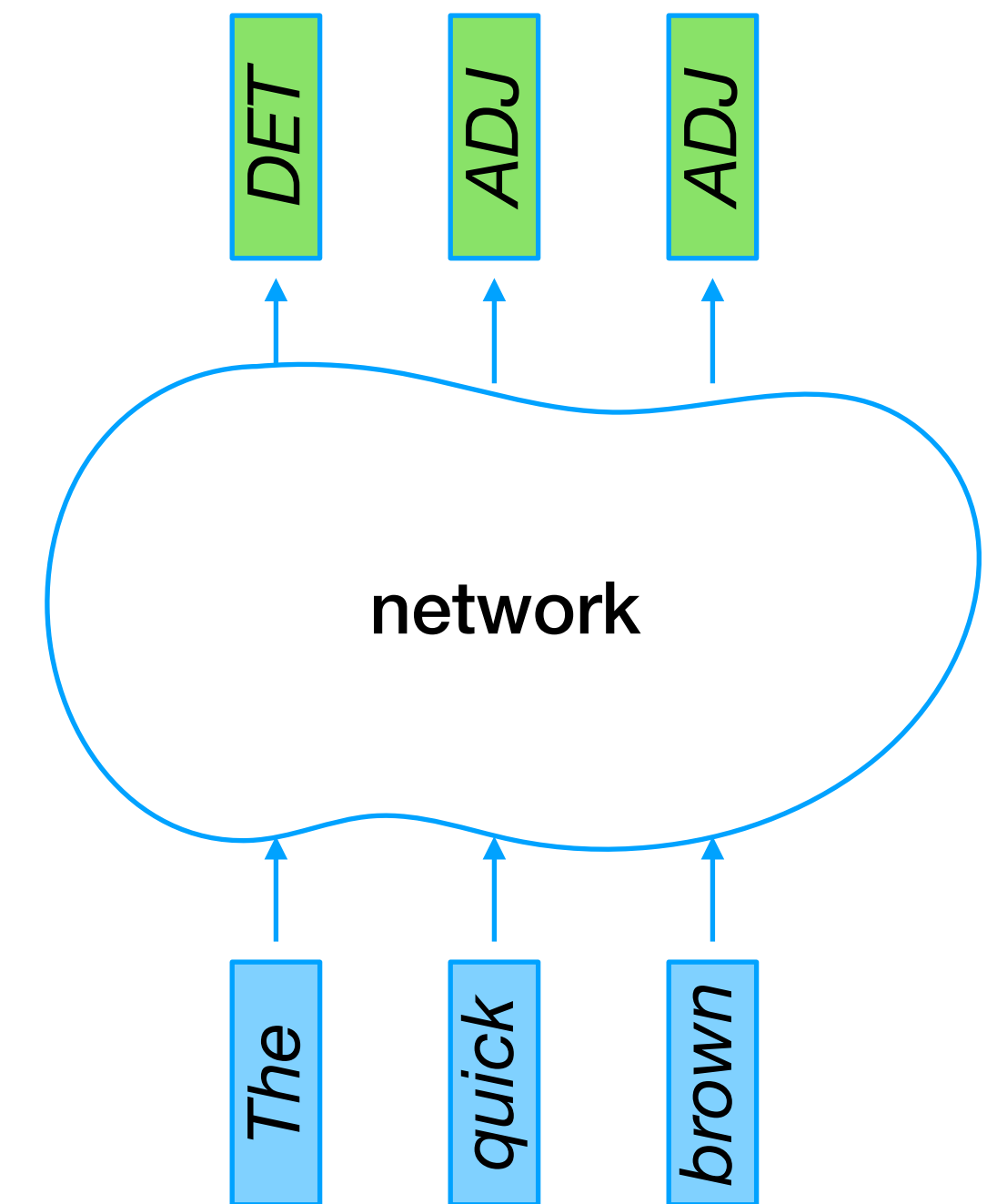
*many → many (corresp.)
transduction
(e.g., POS tagging)*



*many → many
generation
(e.g., language model)*

Part of speech tagging

- Example application: POS tagging
- Map each word (in context) to its part of speech (determiner, adjective, noun, verb, ...)
- A transduction task: output tokens are matched with corresponding input tokens



OCR

*another
transduction task*

*figure credit: (Chatzis
& Demiris, 2013)*



- Could try to recognize each character independently, but accuracy will be higher if we use context

Masked token prediction

another
transduction task



The quick brown fox jumped over the lazy dog

- For each token index i :
 - ▶ *input*: a window of tokens near index i but **not** token i :
 $j \in \{i - k, \dots, i + k\} \setminus \{i\}$ (the **context**)
 - ▶ *target*: the masked token i
- Could also mask more than 1 token in a context window

Masked token prediction

another
transduction task

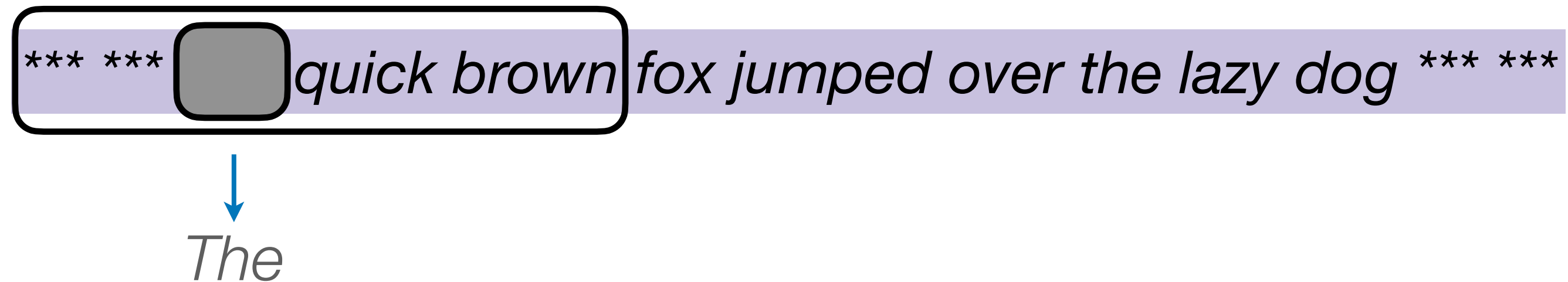


*** ** The quick brown fox jumped over the lazy dog *** **

- For each token index i :
 - ▶ *input*: a window of tokens near index i but **not** token i :
 $j \in \{i - k, \dots, i + k\} \setminus \{i\}$ (the **context**)
 - ▶ *target*: the masked token i
- Could also mask more than 1 token in a context window

Masked token prediction

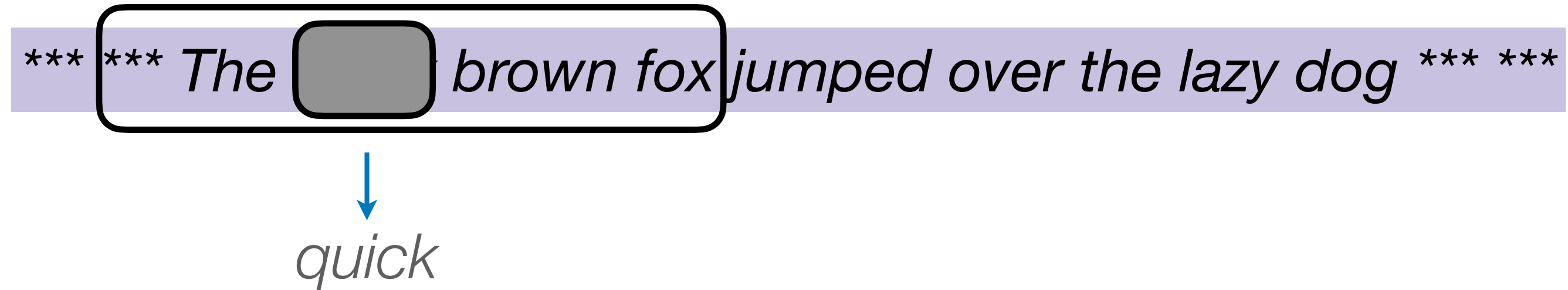
another
transduction task



- For each token index i :
 - ▶ *input*: a window of tokens near index i but **not** token i :
 $j \in \{i - k, \dots, i + k\} \setminus \{i\}$ (the **context**)
 - ▶ *target*: the masked token i
- Could also mask more than 1 token in a context window

Masked token prediction

another
transduction task



- For each token index i :
 - ▶ *input*: a window of tokens near index i but **not** token i :
 $j \in \{i - k, \dots, i + k\} \setminus \{i\}$ (the **context**)
 - ▶ *target*: the masked token i
- Could also mask more than 1 token in a context window

Masked token prediction

another
transduction task

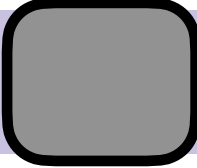


- For each token index i :
 - ▶ *input*: a window of tokens near index i but **not** token i :
 $j \in \{i - k, \dots, i + k\} \setminus \{i\}$ (the **context**)
 - ▶ *target*: the masked token i
- Could also mask more than 1 token in a context window

Masked token prediction

another transduction task



*** ** The quick brown fox jumped over the lazy  *** **

↓
dog

- For each token index i :
 - ▶ *input*: a window of tokens near index i but **not** token i :
 $j \in \{i - k, \dots, i + k\} \setminus \{i\}$ (the **context**)
 - ▶ *target*: the masked token i
- Could also mask more than 1 token in a context window

A generation task: language modeling

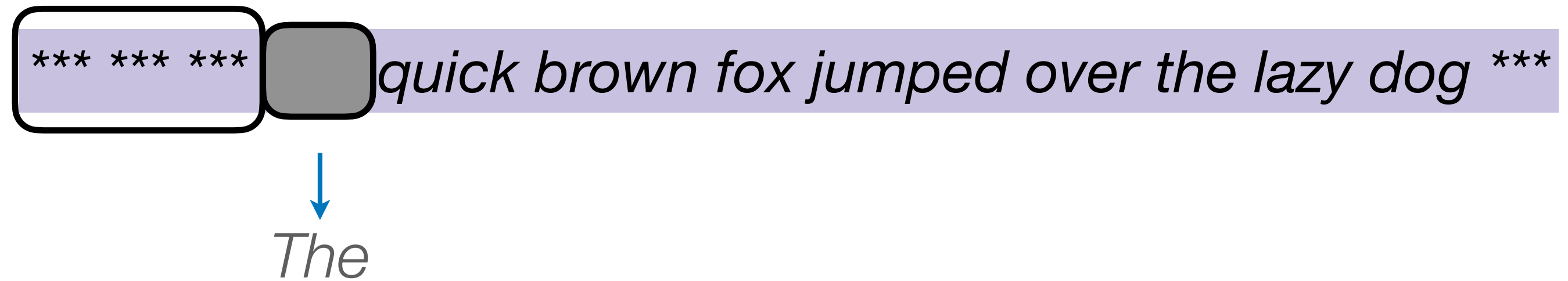
- Given a prompt (a string of tokens)
 - ▶ e.g., “ $3 \times 17 - 9 =$ ” or “It was a dark and stormy night.”
- Predict a completion (another string of tokens)
 - ▶ e.g., “42” or “Suddenly, a shot rang out.”
 - ▶ goal: concatenation of prompt and completion should have high probability in training set distribution
 - ▶ i.e., $\max P(\text{completion} \mid \text{prompt})$
- Often trained on the one-step version (just one token in the completion): ***next-token prediction***
 - ▶ we’ll talk more later about using next-token prediction repeatedly to generate a longer completion

Next token prediction

*** *** *** *The quick brown fox jumped over the lazy dog* ***

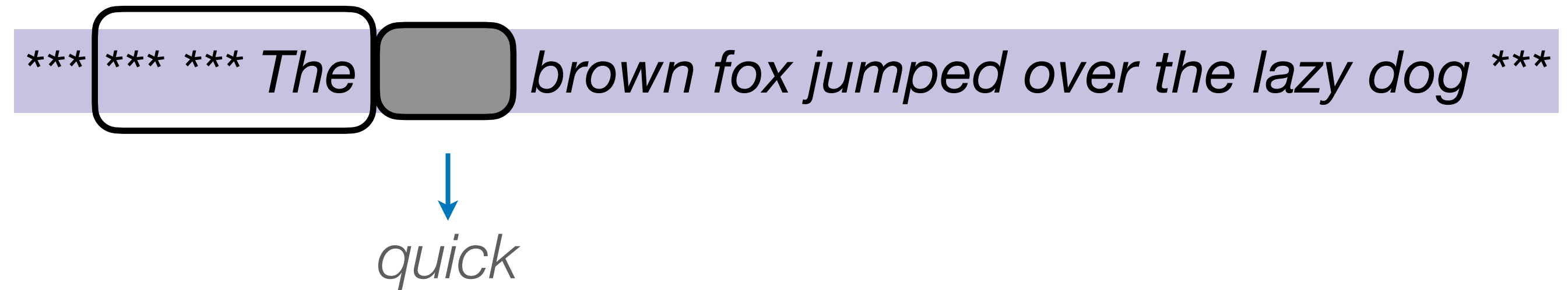
- For each token index i :
 - ▶ *input*: a window of tokens before index i but **not** token i :
 $j \in \{i - k, \dots, i - 1\}$ (the **context**)
 - ▶ *target*: the next token i

Next token prediction



- For each token index i :
 - ▶ *input*: a window of tokens before index i but **not** token i :
 $j \in \{i - k, \dots, i - 1\}$ (the **context**)
 - ▶ *target*: the next token i

Next token prediction



- For each token index i :
 - ▶ *input*: a window of tokens before index i but **not** token i :
 $j \in \{i - k, \dots, i - 1\}$ (the **context**)
 - ▶ *target*: the next token i

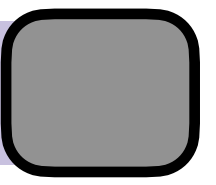
Next token prediction



- For each token index i :
 - ▶ *input*: a window of tokens before index i but **not** token i :
 $j \in \{i - k, \dots, i - 1\}$ (the **context**)
 - ▶ *target*: the next token i

Next token prediction

*** *** *** The quick brown fox jumped over the lazy dog



- For each token index i :
 - ▶ *input*: a window of tokens before index i but **not** token i :
 $j \in \{i - k, \dots, i - 1\}$ (the **context**)
 - ▶ *target*: the next token i

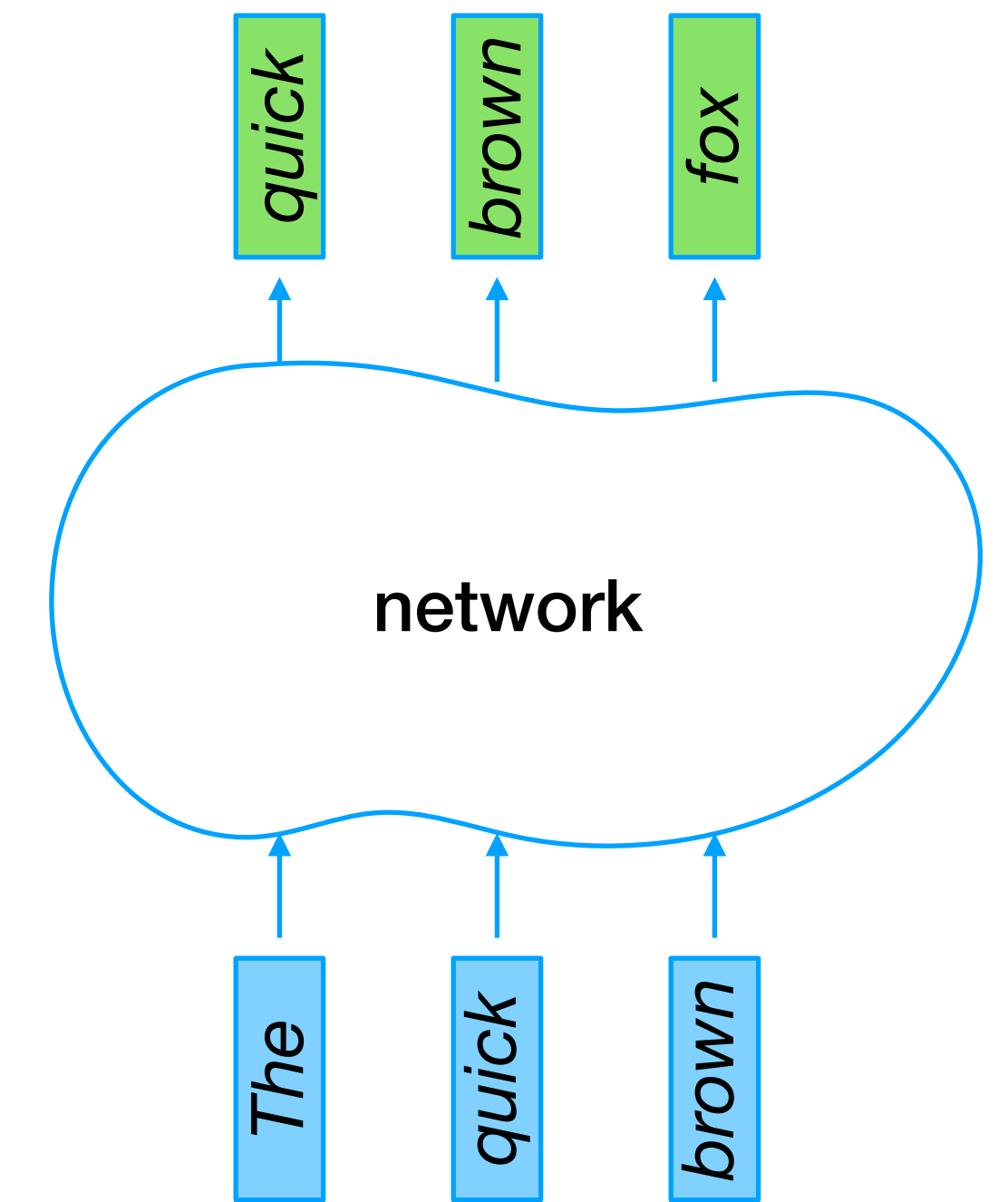
Recurrent neural network (RNN)

- Oldest sequence model
- Replicate a common network block for each step
 - ▶ input x_t , hidden state z_t , output y_t
- Hidden state at each step becomes another input to next step
 - ▶ initialize hidden state (e.g., to zero) before first step
 - ▶ weight matrix V connects steps, W gives embeddings

$$z_0 = 0$$

$$z_t = \sigma(Vz_{t-1} + Wx_t + b) \quad t = 1..T$$

note: no time index



This is an **Elman network**, a type of **simple recurrent network**, applied to next-token prediction

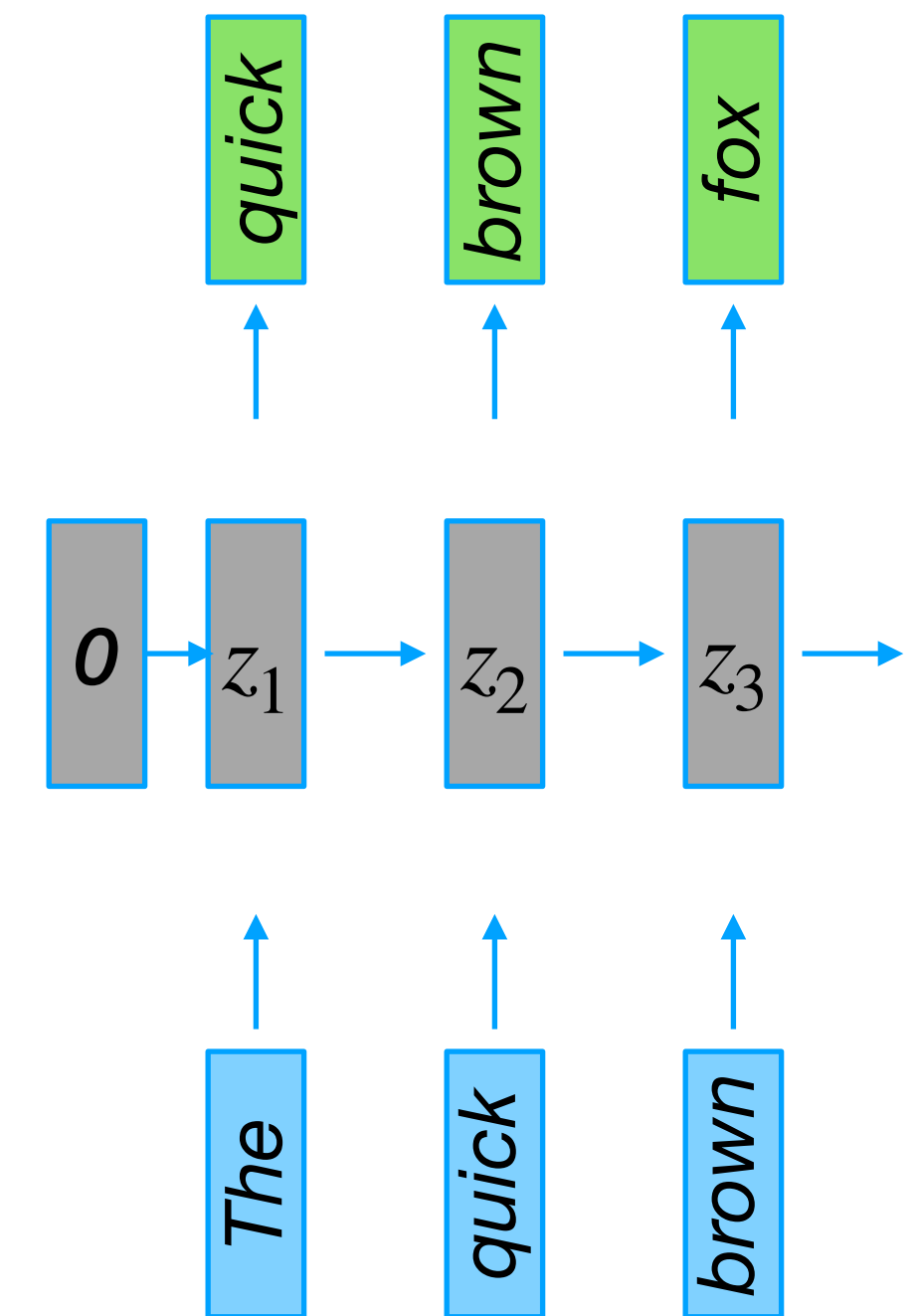
Recurrent neural network (RNN)

- Oldest sequence model
- Replicate a common network block for each step
 - ▶ input x_t , hidden state z_t , output y_t
- Hidden state at each step becomes another input to next step
 - ▶ initialize hidden state (e.g., to zero) before first step
 - ▶ weight matrix V connects steps, W gives embeddings

$$z_0 = 0$$

$$z_t = \sigma(Vz_{t-1} + Wx_t + b) \quad t = 1..T$$

note: no time index



*This is an **Elman network**, a type of **simple recurrent network**, applied to next-token prediction*

A tiny Elman network on {A, B}*

model and parameters

embed

$$e_t = E \text{onehot}(x_t)$$

recur

$$z_t = Wh_{t-1} + e_t + b$$

activate

$$h_t = \text{ReLU}(z_t)$$

read out

$$o_t = Vh_t + c$$

predict

$$p_t = \text{softmax}(o_t)$$

loss

$$L = \sum_t -y_t \cdot \ln p_t, \quad y_t = \text{onehot}(x_{t+1})$$

Parameters

$$E = \begin{bmatrix} 1.00 & -0.50 \\ -0.50 & 1.00 \end{bmatrix}$$

e_A

e_B

$$W = \begin{bmatrix} 0.60 & -0.40 \\ 0.40 & 0.60 \end{bmatrix}$$

c

$$b = \begin{bmatrix} 0.30 \\ -0.20 \end{bmatrix}$$

logit A

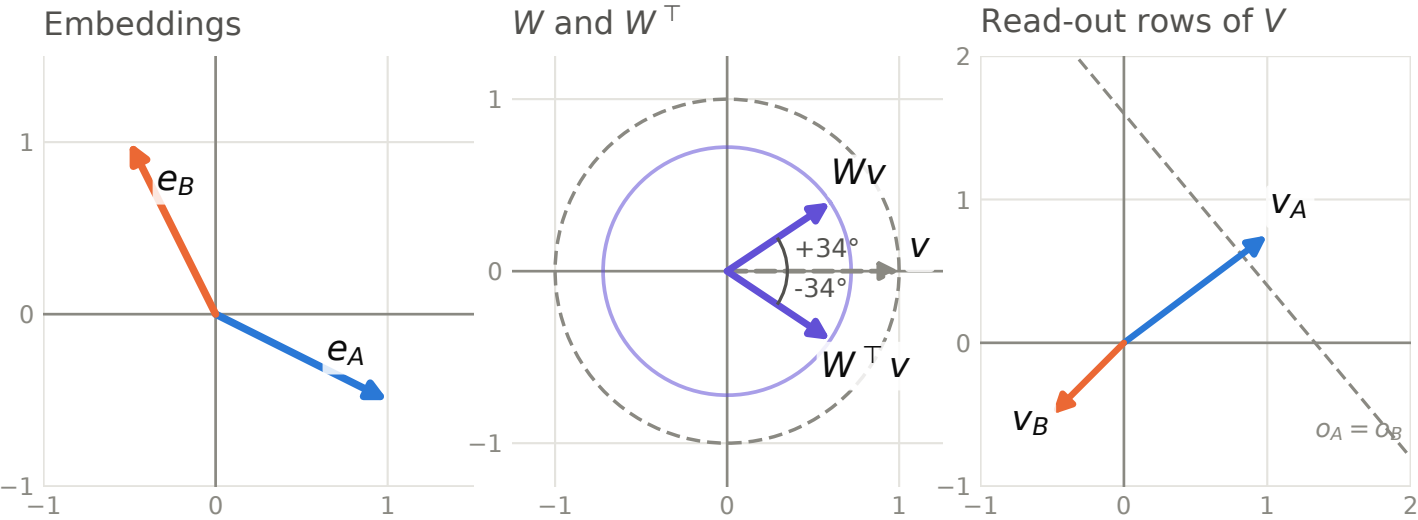
logit B

$$V = \begin{bmatrix} 1.00 & 0.75 \\ -0.50 & -0.50 \end{bmatrix}$$

$$c = \begin{bmatrix} -2.00 \\ 0.00 \end{bmatrix}$$

$$h_0 = (0, 0)$$

The columns of E are the embeddings of A and B; the rows of V give the logits for A and B. W rotates by 33.7° and scales by 0.72, so each step turns the old state a little and shrinks it.



Columns of E: the vector added to the state when the network reads that token.

W turns a vector counter-clockwise (forward pass). W^T turns it clockwise by the same angle (backward pass). Both shrink it by 0.72: the solid circle.

Each logit is the state dotted with a row of V, plus c. Dashed: states where the two logits tie. The network predicts A above that line and B below it.

Forward pass, step 1 of 4

read $x_1 = A$, predict x_2

sequence A A B B A filled: input x_1 outlined: target x_2

1 **embed**

$$e_1 = E \text{onehot}(x_1)$$

$$\begin{bmatrix} 1.00 & -0.50 \\ -0.50 & 1.00 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1.00 \\ -0.50 \end{bmatrix}$$

2 **carry the state**

$$Wh_0$$

$$\begin{bmatrix} 0.60 & -0.40 \\ 0.40 & 0.60 \end{bmatrix} \begin{bmatrix} 0.00 \\ 0.00 \end{bmatrix} = \begin{bmatrix} 0.00 \\ 0.00 \end{bmatrix}$$

the start state is zero

3 **add, then ReLU**

$$h_1 = \text{ReLU}(z_1)$$

$$\begin{bmatrix} 0.00 \\ 0.00 \end{bmatrix} + \begin{bmatrix} 1.00 \\ -0.50 \end{bmatrix} + \begin{bmatrix} 0.30 \\ -0.20 \end{bmatrix} = \begin{bmatrix} 1.30 \\ -0.70 \end{bmatrix} \xrightarrow{\text{ReLU}} \begin{bmatrix} 1.30 \\ 0.00 \end{bmatrix}$$

unit 2 is negative: clipped

4 **read out**

$$o_1 = Vh_1 + c$$

$$\begin{bmatrix} 1.00 & 0.75 \\ -0.50 & -0.50 \end{bmatrix} \begin{bmatrix} 1.30 \\ 0.00 \end{bmatrix} + \begin{bmatrix} -2.00 \\ 0.00 \end{bmatrix} = \begin{bmatrix} -0.70 \\ -0.65 \end{bmatrix}$$

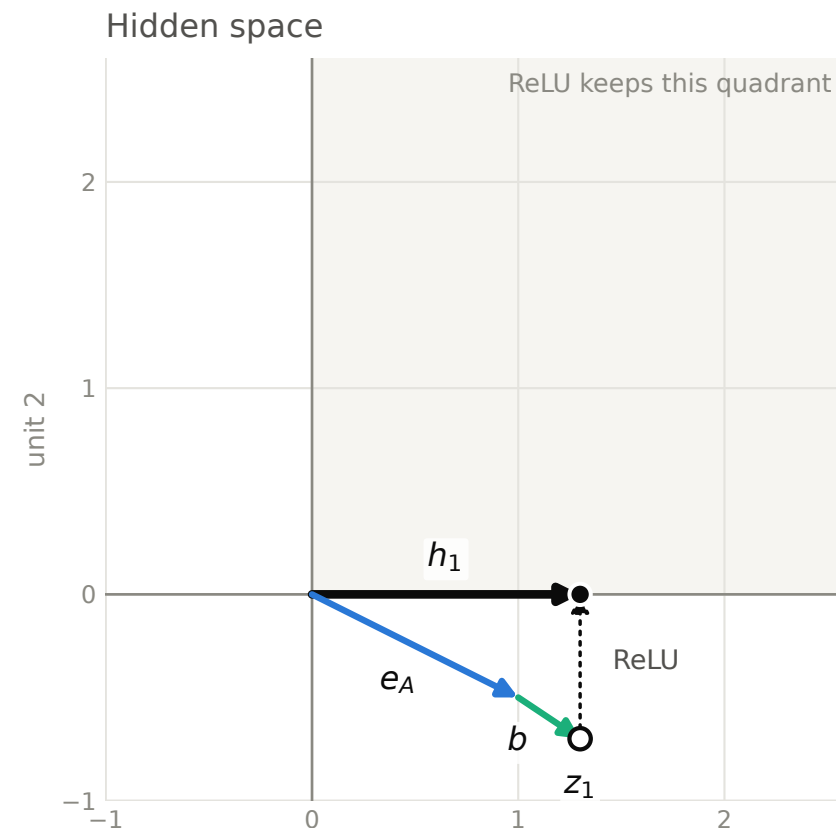
5 **predict**

$$p_1 = \text{softmax}(o_1)$$

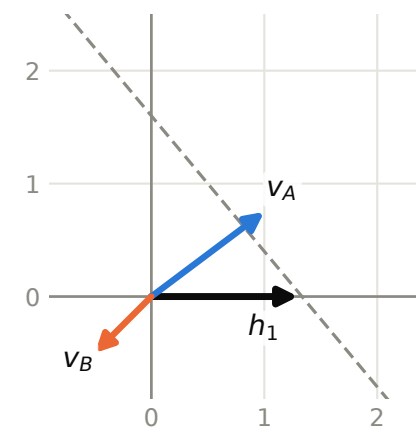
$$\text{softmax} \begin{bmatrix} -0.70 \\ -0.65 \end{bmatrix} = \begin{bmatrix} 0.49 \\ 0.51 \end{bmatrix}$$

target $y_1 = \text{onehot}(A) = (1, 0)$

$$L_1 = -y_1 \cdot \ln p_1 = -\ln 0.49 = 0.72$$



Read-out: h_1 against the rows of V



Prediction for x_2

$p(A)$ 0.49

$p(B)$ 0.51

state is below the dashed tie line $\rightarrow$ B more likely

Forward pass, step 2 of 4

read $x_2 = A$, predict x_3

sequence A A B B A filled: input x_2 outlined: target x_3

1

embed
 $e_2 = E \text{onehot}(x_2)$

$$\begin{matrix} & E & & \text{onehot}(A) & & e_A \\ \begin{bmatrix} 1.00 & -0.50 \\ -0.50 & 1.00 \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \end{bmatrix} & = & \begin{bmatrix} 1.00 \\ -0.50 \end{bmatrix} \end{matrix}$$

2

carry the state
 Wh_1
rotate and shrink

$$\begin{matrix} & W & & h_1 & & Wh_1 \\ \begin{bmatrix} 0.60 & -0.40 \\ 0.40 & 0.60 \end{bmatrix} & \begin{bmatrix} 1.30 \\ 0.00 \end{bmatrix} & = & \begin{bmatrix} 0.78 \\ 0.52 \end{bmatrix} \end{matrix}$$

3

add, then ReLU
 $h_2 = \text{ReLU}(z_2)$
unit 2 is negative: clipped

$$\begin{matrix} & Wh_1 & & e_A & & b & & z_2 & & h_2 \\ \begin{bmatrix} 0.78 \\ 0.52 \end{bmatrix} + \begin{bmatrix} 1.00 \\ -0.50 \end{bmatrix} + \begin{bmatrix} 0.30 \\ -0.20 \end{bmatrix} = \begin{bmatrix} 2.08 \\ -0.18 \end{bmatrix} \xrightarrow{\text{ReLU}} \begin{bmatrix} 2.08 \\ 0.00 \end{bmatrix} \end{matrix}$$

4

read out
 $o_2 = Vh_2 + c$

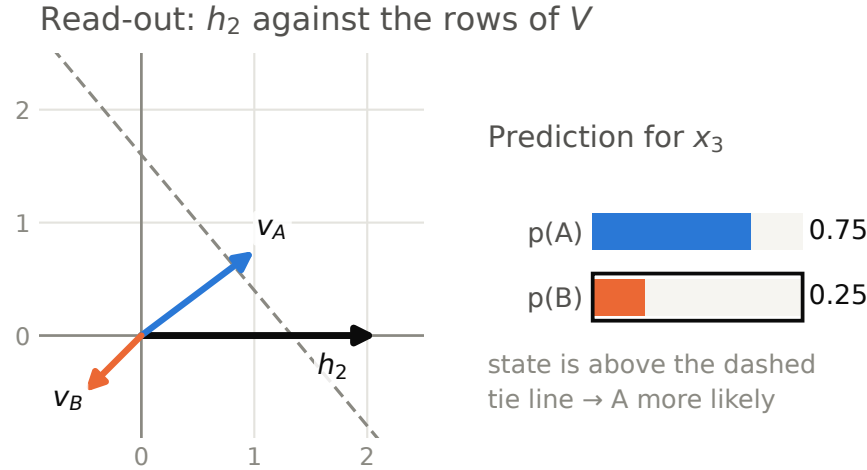
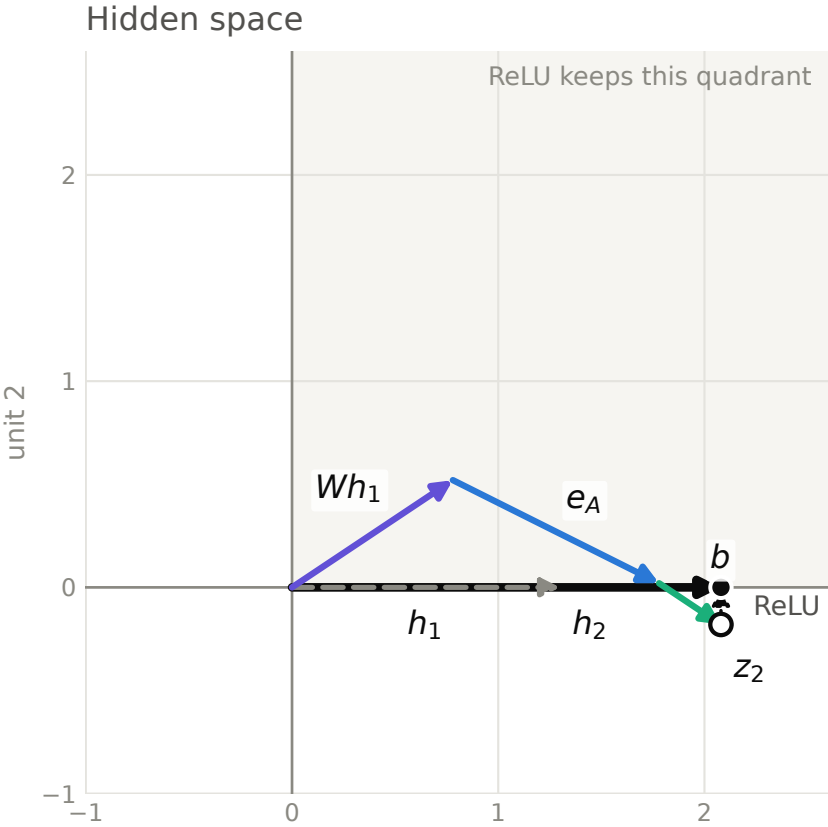
$$\begin{matrix} & V & & h_2 & & c & & o_2 \text{ (logits)} \\ \begin{bmatrix} 1.00 & 0.75 \\ -0.50 & -0.50 \end{bmatrix} \begin{bmatrix} 2.08 \\ 0.00 \end{bmatrix} + \begin{bmatrix} -2.00 \\ 0.00 \end{bmatrix} = \begin{bmatrix} 0.08 \\ -1.04 \end{bmatrix} \end{matrix}$$

5

predict
 $p_2 = \text{softmax}(o_2)$

$$\text{softmax} \begin{bmatrix} 0.08 \\ -1.04 \end{bmatrix} = \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix}$$

target $y_2 = \text{onehot}(B) = (0, 1)$
 $L_2 = -y_2 \cdot \ln p_2 = -\ln 0.25 = 1.40$



Forward pass, step 3 of 4

read $x_3 = B$, predict x_4

sequence A A B B A filled: input x_3 outlined: target x_4

- 1 embed

$e_3 = E \text{onehot}(x_3)$

$$\begin{matrix} E & \text{onehot}(B) & e_B \\ \begin{bmatrix} 1.00 & -0.50 \\ -0.50 & 1.00 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} & = \begin{bmatrix} -0.50 \\ 1.00 \end{bmatrix} \end{matrix}$$
- 2 carry the state

Wh_2
rotate and shrink

$$\begin{matrix} W & h_2 & Wh_2 \\ \begin{bmatrix} 0.60 & -0.40 \\ 0.40 & 0.60 \end{bmatrix} & \begin{bmatrix} 2.08 \\ 0.00 \end{bmatrix} & = \begin{bmatrix} 1.25 \\ 0.83 \end{bmatrix} \end{matrix}$$
- 3 add, then ReLU

$h_3 = \text{ReLU}(z_3)$
both units stay active

$$\begin{matrix} Wh_2 & e_B & b & z_3 & h_3 \\ \begin{bmatrix} 1.25 \\ 0.83 \end{bmatrix} + \begin{bmatrix} -0.50 \\ 1.00 \end{bmatrix} + \begin{bmatrix} 0.30 \\ -0.20 \end{bmatrix} = \begin{bmatrix} 1.05 \\ 1.63 \end{bmatrix} \xrightarrow{\text{ReLU}} \begin{bmatrix} 1.05 \\ 1.63 \end{bmatrix} \end{matrix}$$
- 4 read out

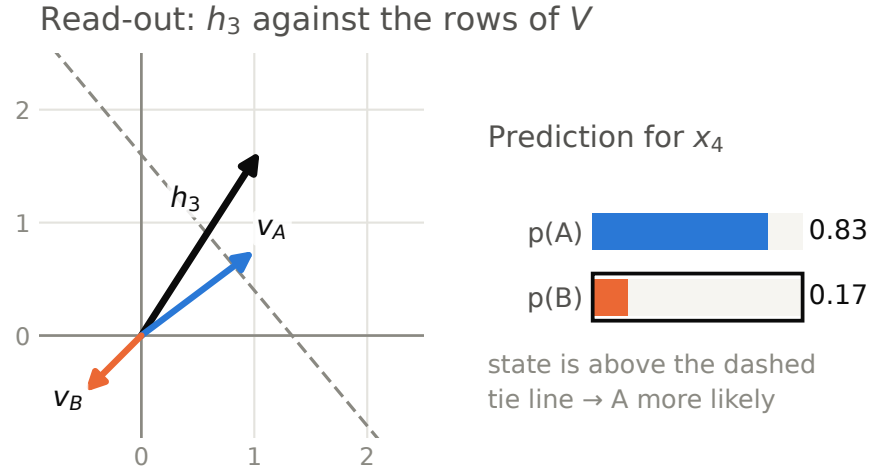
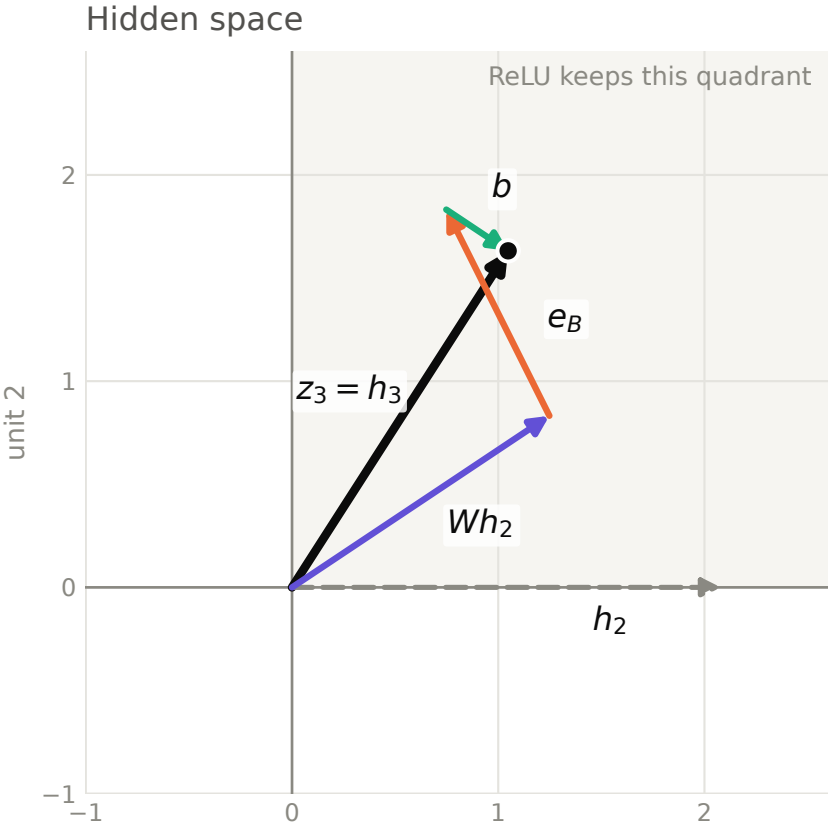
$o_3 = Vh_3 + c$

$$\begin{matrix} V & h_3 & c & o_3 \text{ (logits)} \\ \begin{bmatrix} 1.00 & 0.75 \\ -0.50 & -0.50 \end{bmatrix} \begin{bmatrix} 1.05 \\ 1.63 \end{bmatrix} + \begin{bmatrix} -2.00 \\ 0.00 \end{bmatrix} = \begin{bmatrix} 0.27 \\ -1.34 \end{bmatrix} \end{matrix}$$
- 5 predict

$p_3 = \text{softmax}(o_3)$

$$\begin{matrix} o_3 & p_3 \\ \text{softmax} \begin{bmatrix} 0.27 \\ -1.34 \end{bmatrix} = \begin{bmatrix} 0.83 \\ 0.17 \end{bmatrix} \end{matrix}$$

target $y_3 = \text{onehot}(B) = (0, 1)$
 $L_3 = -y_3 \cdot \ln p_3 = -\ln 0.17 = 1.79$



Forward pass, step 4 of 4

read $x_4 = B$, predict x_5

sequence A A B B A filled: input x_4 outlined: target x_5

1

embed
 $e_4 = E \text{onehot}(x_4)$

$$\begin{matrix} E & \text{onehot}(B) & e_B \\ \begin{bmatrix} 1.00 & -0.50 \\ -0.50 & 1.00 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} & = \begin{bmatrix} -0.50 \\ 1.00 \end{bmatrix} \end{matrix}$$

2

carry the state
 Wh_3
rotate and shrink

$$\begin{matrix} W & h_3 & Wh_3 \\ \begin{bmatrix} 0.60 & -0.40 \\ 0.40 & 0.60 \end{bmatrix} & \begin{bmatrix} 1.05 \\ 1.63 \end{bmatrix} & = \begin{bmatrix} -0.02 \\ 1.40 \end{bmatrix} \end{matrix}$$

3

add, then ReLU
 $h_4 = \text{ReLU}(z_4)$
unit 1 is negative: clipped

$$\begin{matrix} Wh_3 & e_B & b & z_4 & h_4 \\ \begin{bmatrix} -0.02 \\ 1.40 \end{bmatrix} + \begin{bmatrix} -0.50 \\ 1.00 \end{bmatrix} + \begin{bmatrix} 0.30 \\ -0.20 \end{bmatrix} = \begin{bmatrix} -0.22 \\ 2.20 \end{bmatrix} \xrightarrow{\text{ReLU}} \begin{bmatrix} 0.00 \\ 2.20 \end{bmatrix} \end{matrix}$$

4

read out
 $o_4 = Vh_4 + c$

$$\begin{matrix} V & h_4 & c & o_4 \text{ (logits)} \\ \begin{bmatrix} 1.00 & 0.75 \\ -0.50 & -0.50 \end{bmatrix} \begin{bmatrix} 0.00 \\ 2.20 \end{bmatrix} + \begin{bmatrix} -2.00 \\ 0.00 \end{bmatrix} = \begin{bmatrix} -0.35 \\ -1.10 \end{bmatrix} \end{matrix}$$

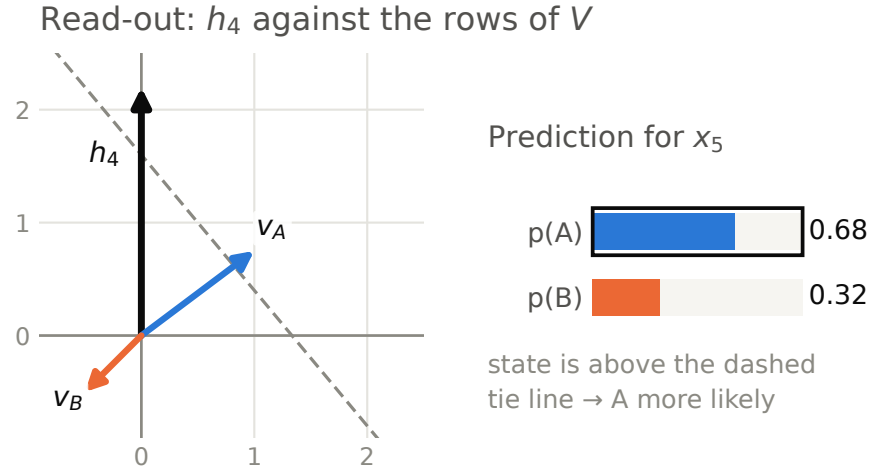
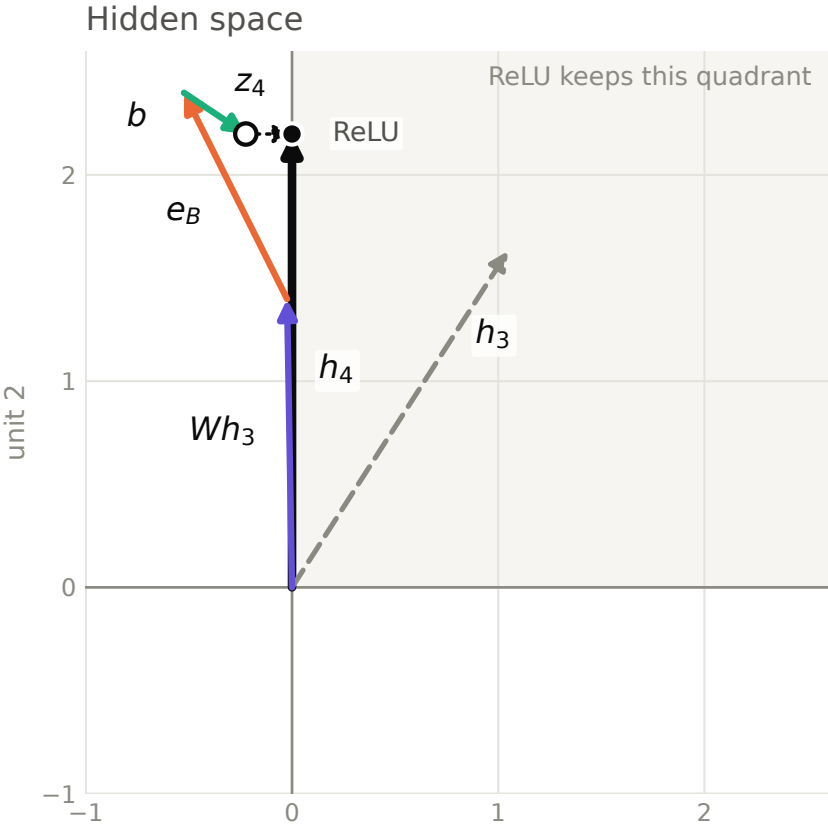
5

predict
 $p_4 = \text{softmax}(o_4)$

$$\begin{matrix} o_4 & p_4 \\ \text{softmax} \begin{bmatrix} -0.35 \\ -1.10 \end{bmatrix} = \begin{bmatrix} 0.68 \\ 0.32 \end{bmatrix} \end{matrix}$$

target $y_4 = \text{onehot}(A) = (1, 0)$

$L_4 = -y_4 \cdot \ln p_4 = -\ln 0.68 = 0.39$

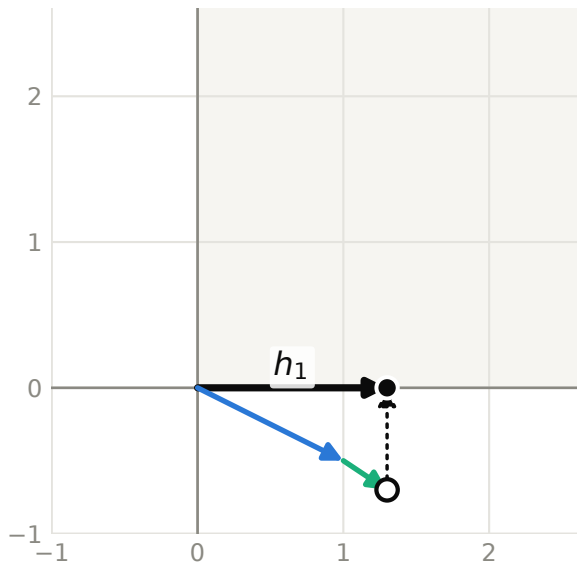


Forward pass, unrolled

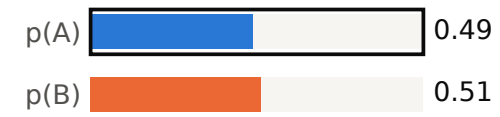
total loss L = 4.30

previous state Wh_{t-1} $+e_A$ $+e_B$ $+b$ h_t (after ReLU)

t = 1 input **A**

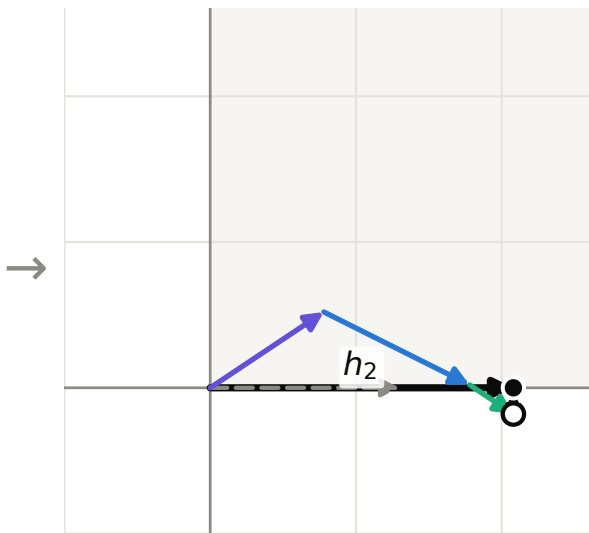


$h_1 = (1.30, 0.00)$



target A $L_1 = 0.72$

t = 2 input **A**

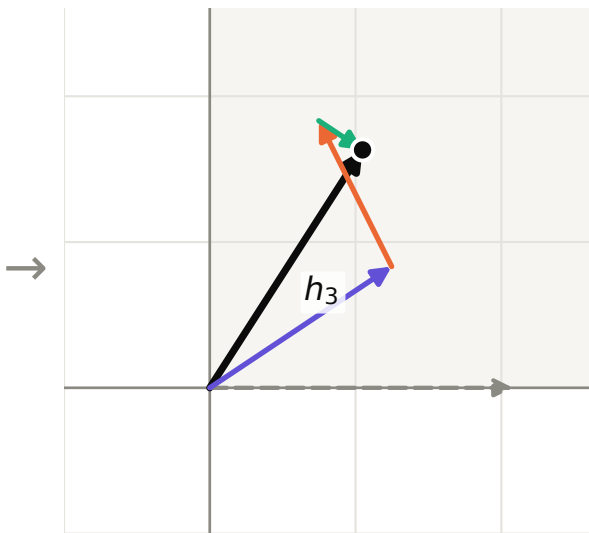


$h_2 = (2.08, 0.00)$



target B $L_2 = 1.40$

t = 3 input **B**

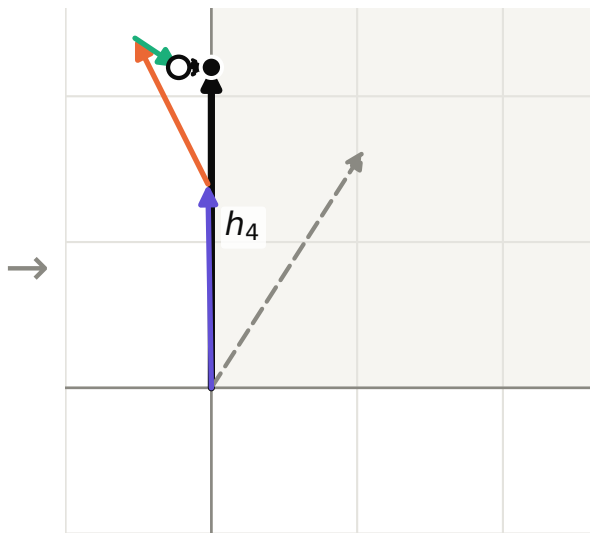


$h_3 = (1.05, 1.63)$

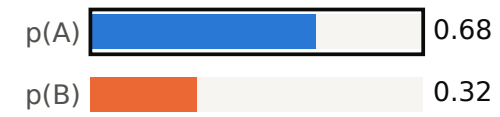


target B $L_3 = 1.79$

t = 4 input **B**



$h_4 = (0.00, 2.20)$



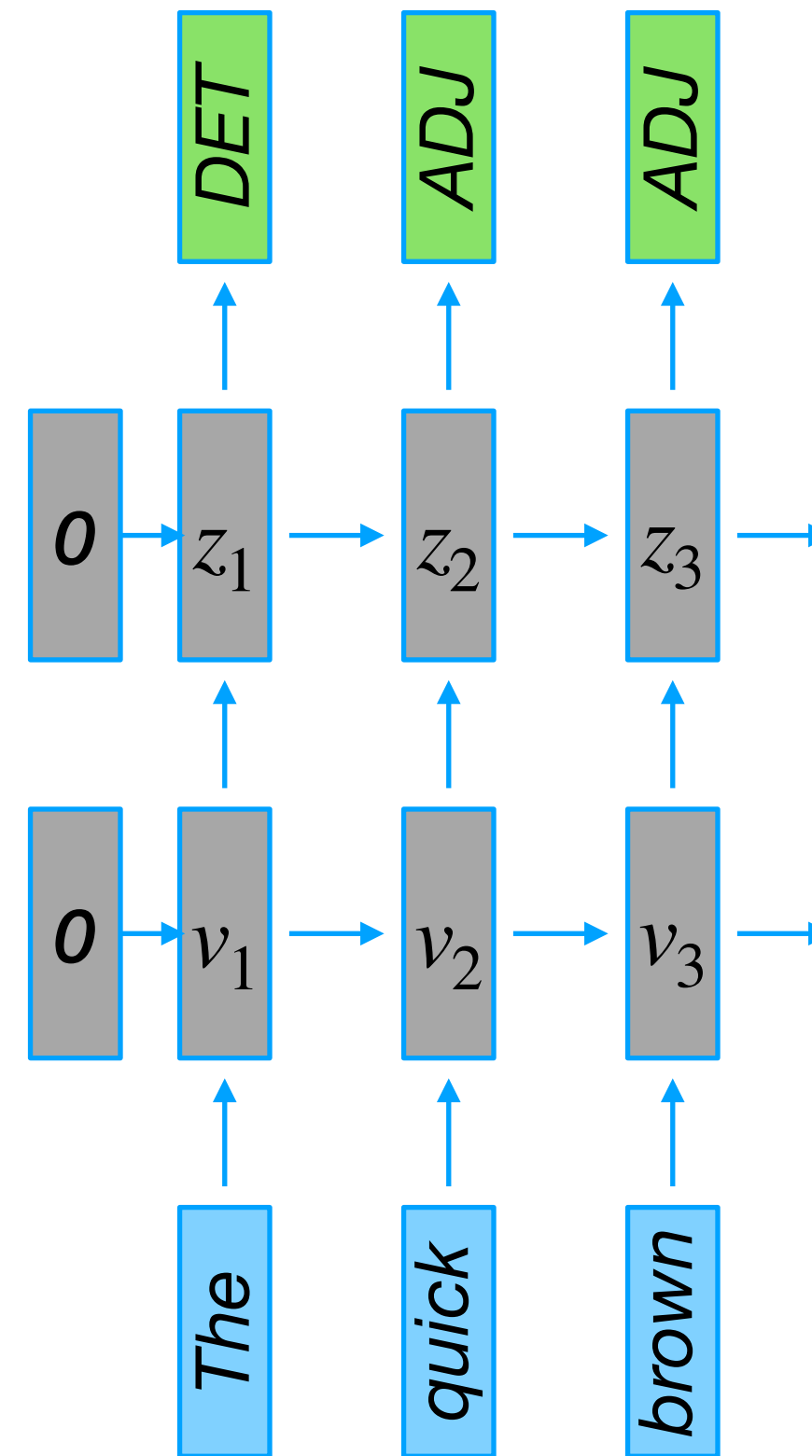
target A $L_4 = 0.39$

Example

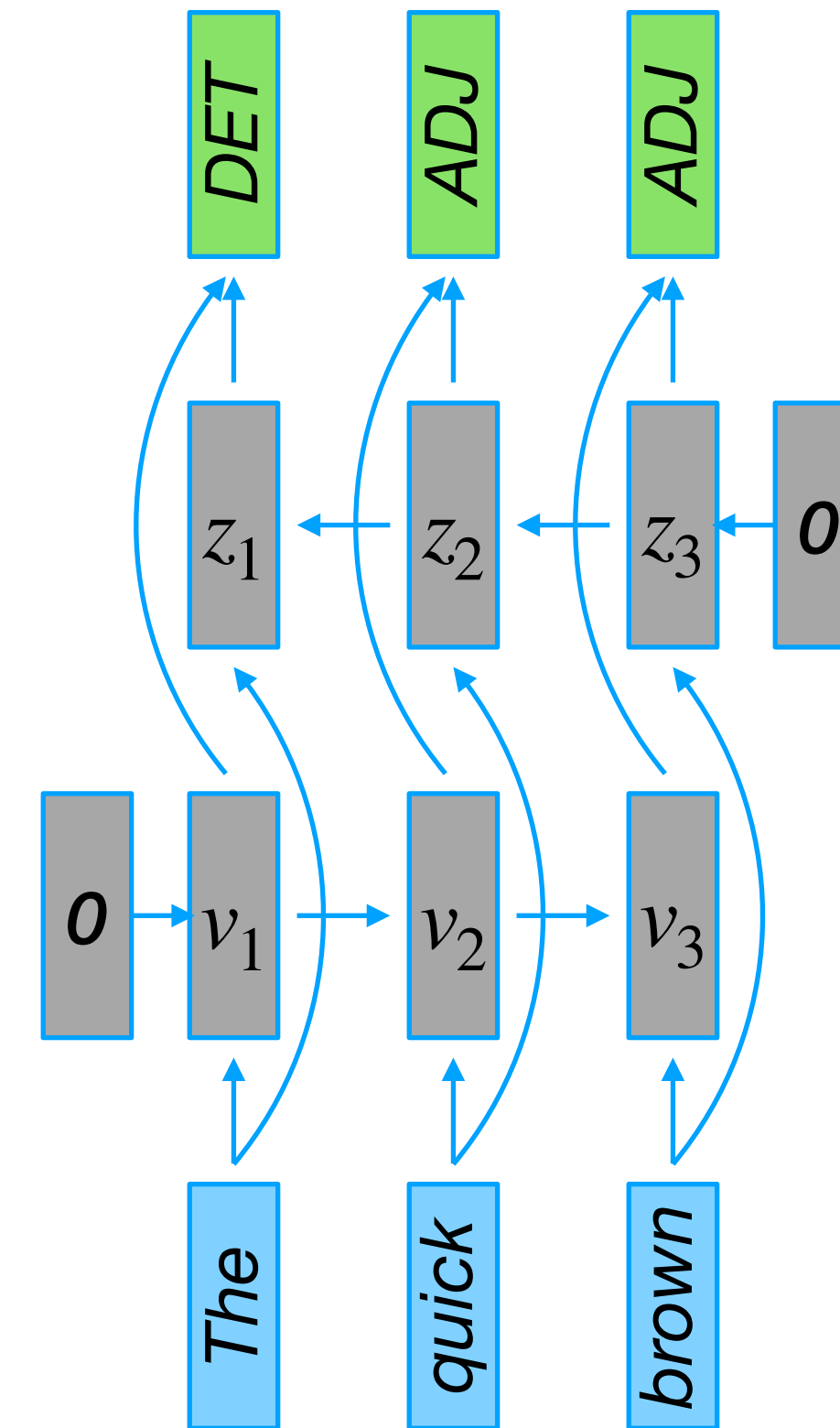
- A tiny Elman RNN
 - ▶ task: see a sequence of tokens ABBABAB..., predict next token at each step
 - ▶ 2-dim hidden states, ReLU activation, cross-entropy loss

Variants

*shown here on
POS tagging
(transduction)*



stacked



bidirectional

- Lots of variants of RNNs — incl. multiple kinds of memory, see below

Training RNNs

*note: vanishing/
exploding gradients!*

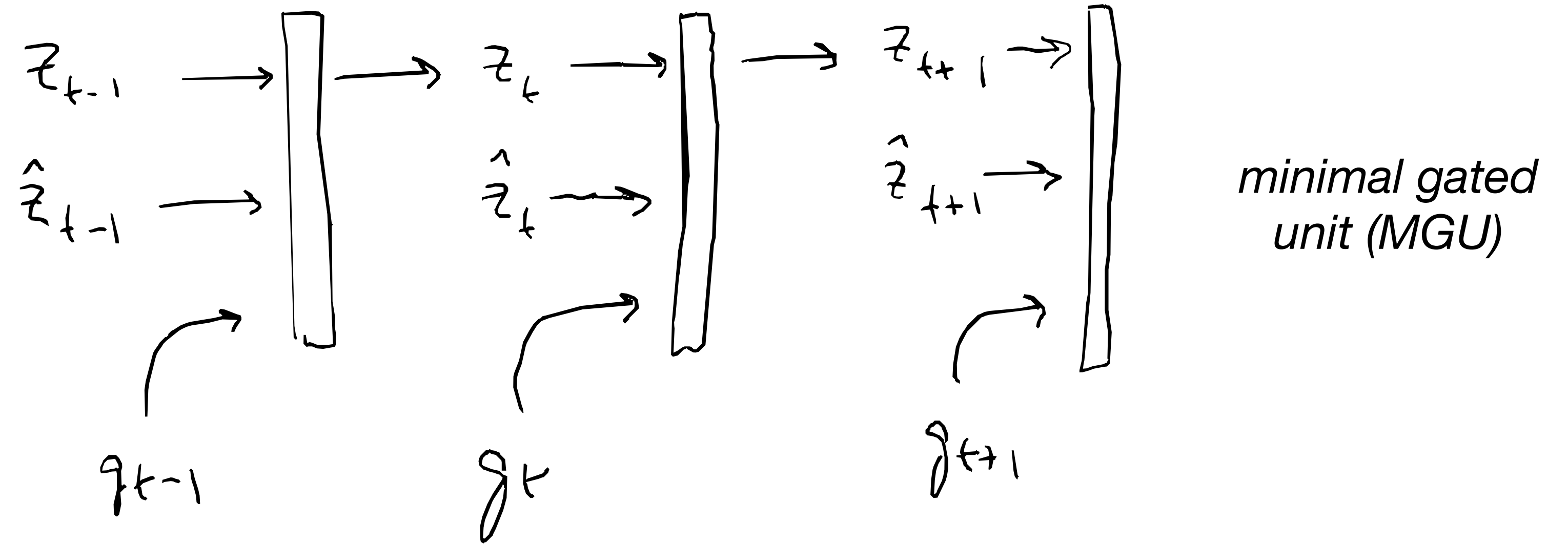
*(same tools as
before + one more
below)*

- Once we unroll it, RNN is just like any other network
 - ▶ forward pass from $t = 1..T$
 - ▶ backward pass from $t = T..1$ (*backprop through time*)
 - ▶ SGD step
- Similar to deep feedforward net (MLP)
 - ▶ difference #1: same parameters V, W, b at every step
 - ▶ autodiff handles parameter sharing automatically
 - ▶ or we can manually sum gradients (total derivative rule)
 - ▶ difference #2: multiple output nodes
 - ▶ loss = sum(losses at each output), so gradients backprop from each output and sum at every parameter
 - ▶ have to run all time steps before we can backprop

Example backward pass

Memory

Unlike residual connection, to couple long distances, need to train gate to be near 1 (don't replace) in between



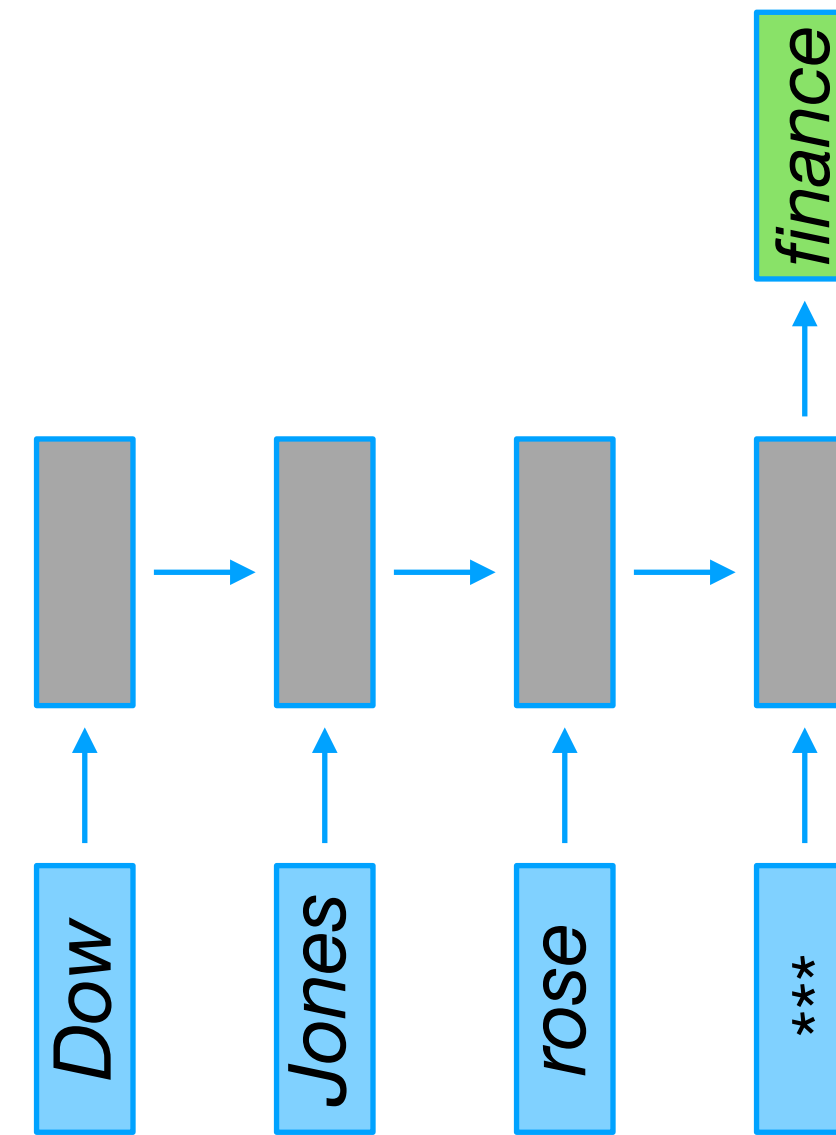
$$\text{gating: } z_{t+1} = g_t \circ z_t + (1 - g_t) \circ \hat{z}_t$$

both gate g_t and candidate update $\hat{z}_t$ are of form $f(Vz_{t-1} + Wx_t + b)$

output y_t could depend on z_t, x_t

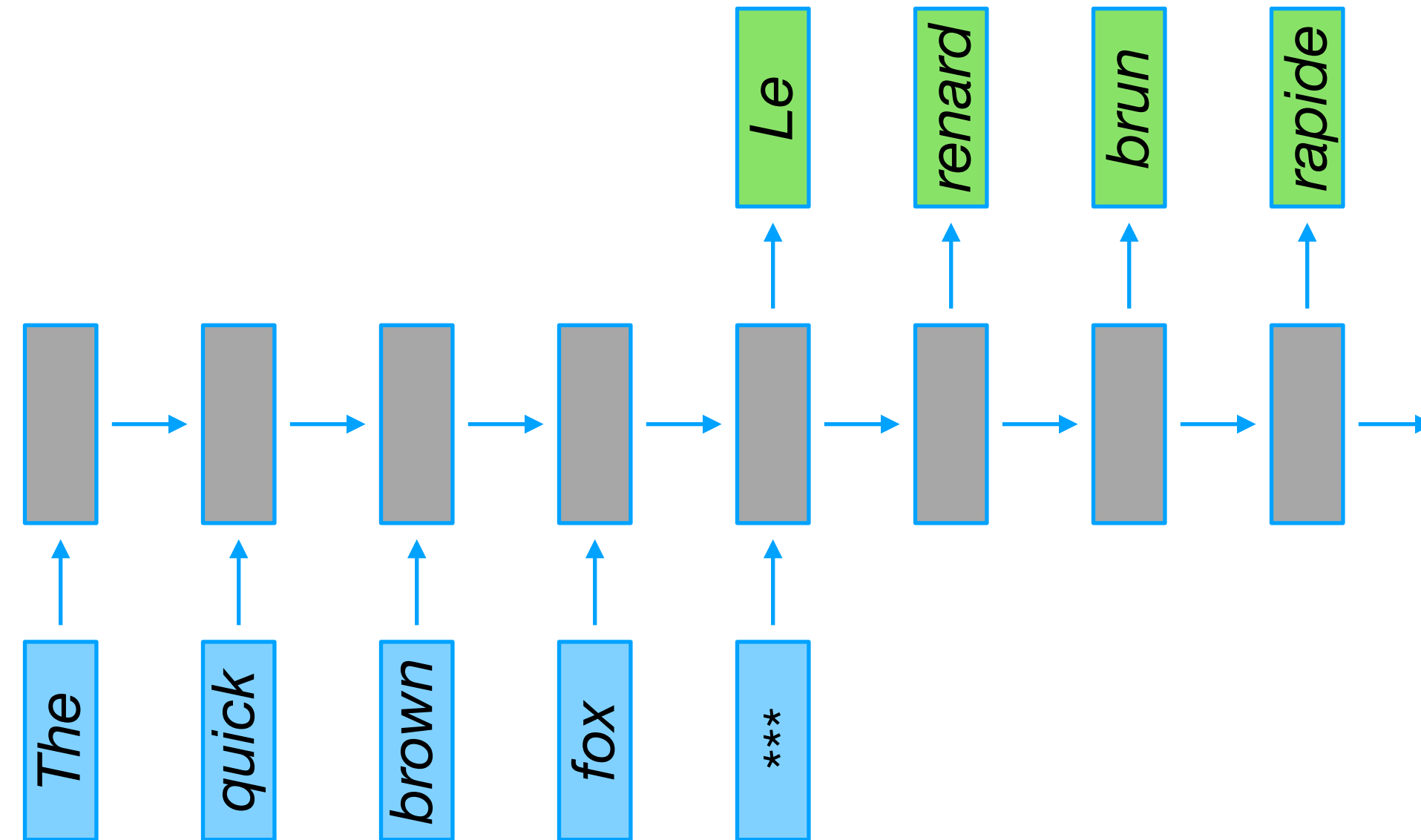
- Use *gating* to update hidden state
 - ▶ differentiable decision to store into z_t
 - ▶ like a controllable residual connection that lets us connect storage to retrieval
- Examples: LSTM, GRU, MGU

RNN for many:1 tasks



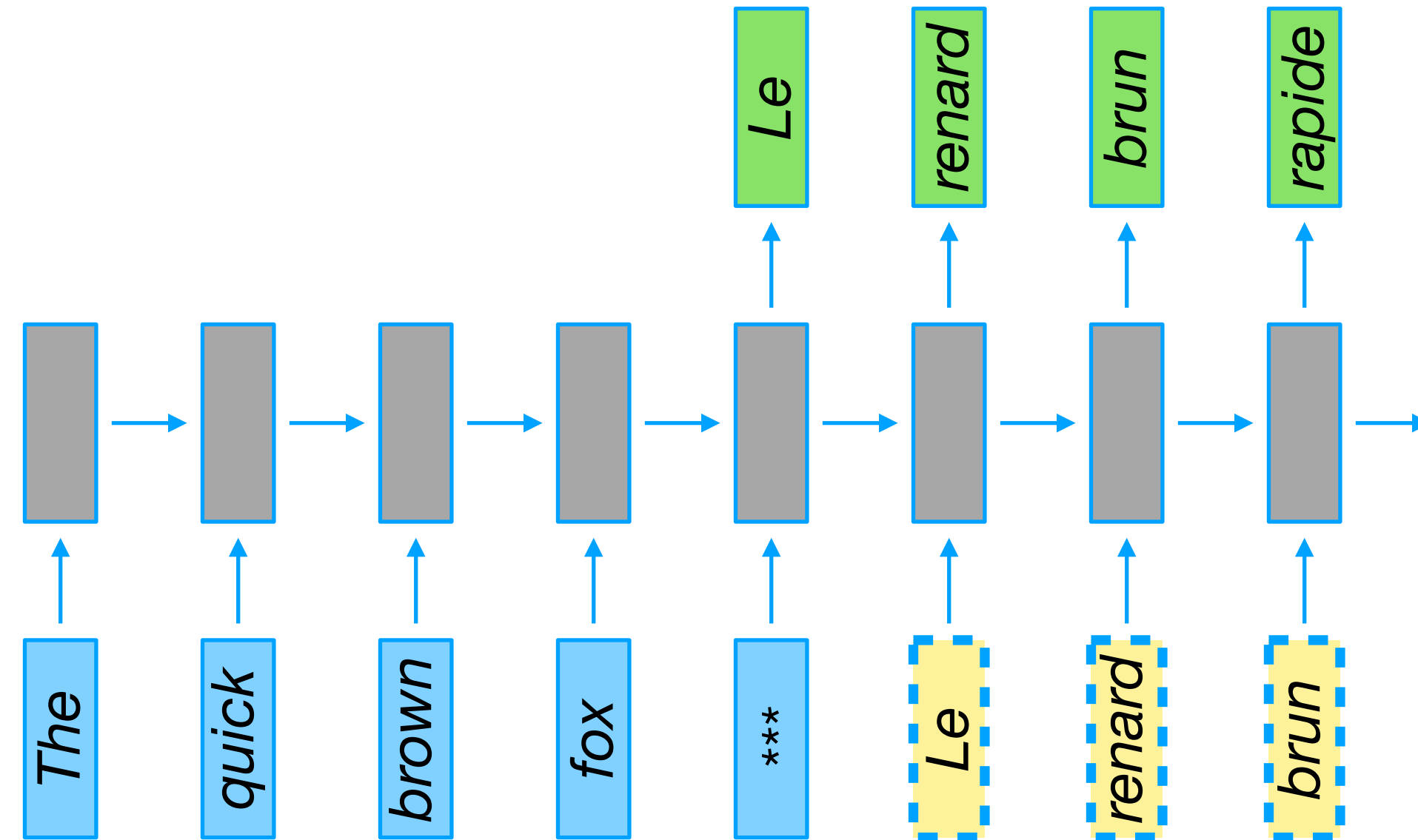
- To handle many:1 tasks like classification, most common setup is to end with a special token like ***, predict label based on corresponding hidden vector

RNN for generation



- To generate an arbitrary-length output, we can use RNN as a generative model
- E.g., language model: process input tokens (prompt), mark end of prompt w/ ***, continue after *** with next-token prediction
 - ▶ each time step generates one new token; stop when we generate *** again

RNN for generation

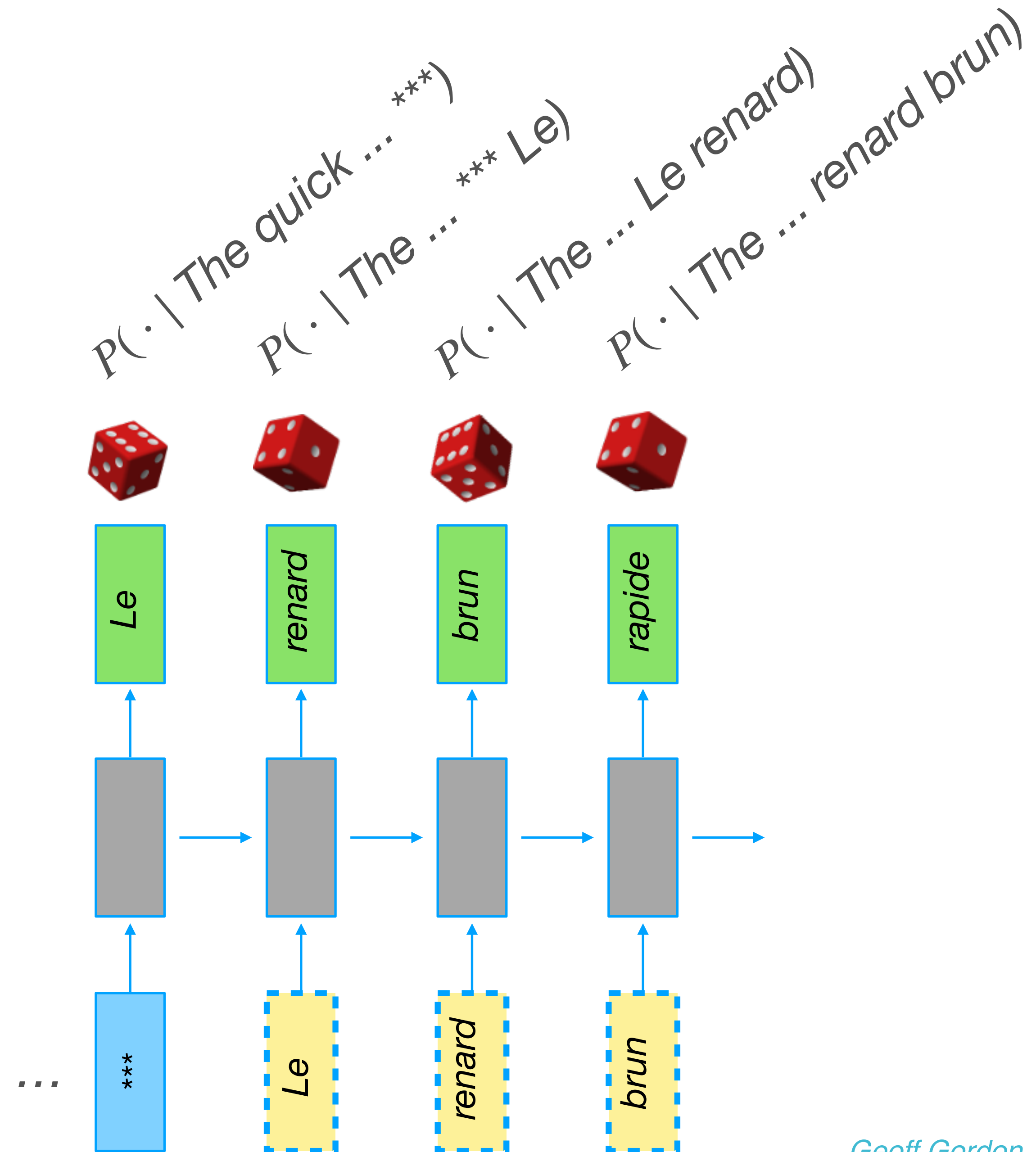


- To generate an arbitrary-length output, we can use RNN as a generative model
- E.g., language model: process input tokens (prompt), mark end of prompt w/ ***, continue after *** with next-token prediction
 - ▶ each time step generates one new token; stop when we generate *** again

$$P(\text{Le renard brun rapide} \mid \text{prompt}) = P(\text{Le} \mid \text{prompt}) \cdot P(\text{renard} \mid \text{prompt, Le}) \cdot \dots$$

Generation process (sequence completion)

- Output token at each step is a sample from *conditional probability distribution* given preceding tokens
- Like a weighted die with 50k sides
 - ▶ weights are model output
 - ▶ roll, pick the token that lands face up
- Copy this token to input (condition on it) to generate next

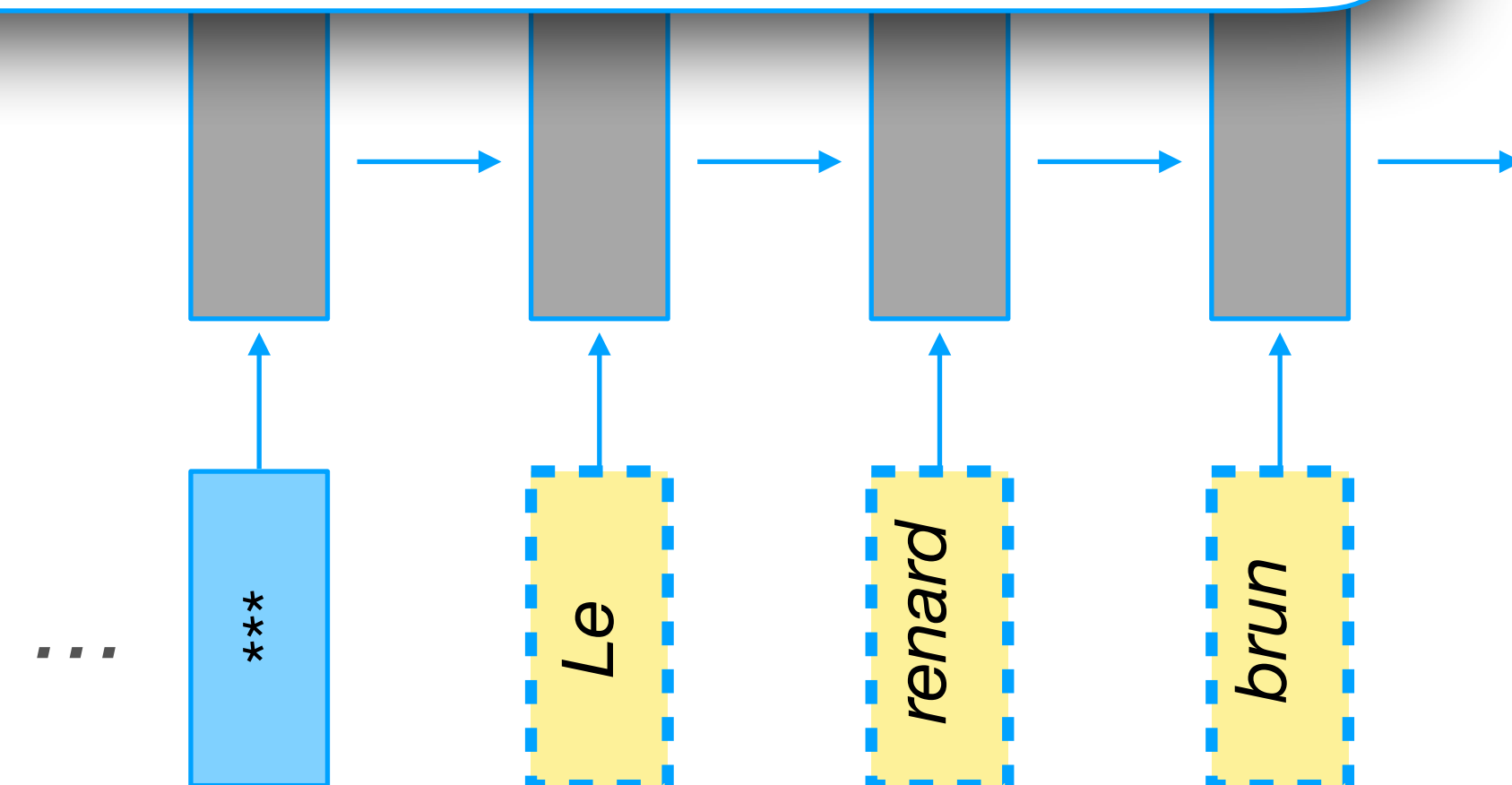


$$P(\text{Le renard brun rapide} \mid \text{prompt}) = P(\text{Le} \mid \text{prompt}) \cdot P(\text{renard} \mid \text{prompt, Le}) \cdot \dots$$

Generation process (sequence completion)

- Output token at each step is a sample from *conditional probability distribution* prece
- Like a
with 5
 - ▶ wei
outp
 - ▶ roll,
that lands face up
- Copy this token to input (condition on it) to generate next

Sequence completion like this is **extremely flexible**: translation, solving math problems, generating robot trajectories, synthesizing images — the basis of many highly popular AI models



Sampling from an RNN-LM

Training data (Shakespeare's As You Like It)

*VIOLA: Why, Salisbury must find his flesh and thought
That which I am not apt, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.*

*KING LEAR: O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.*

Sampling from an RNN-LM

Sample (RNN-LM)

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is but young and tender; and, for your love, I would be loath to foil him, as I must, for my own honour, if he come in: therefore, out of my love to you, I came hither to acquaint you withal, that either you might stay him from his intendment or brook such disgrace well as he shall run into, in that it is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Sampling from an RNN-LM

Sample (RNN-LM)

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter.
I am given, sir, secretly to understand that your younger brother
Orlando hath a disposition to come in disguised against me to try a
fall. To-morrow I will be gone without seeing him, and he escapes me
without seeing me. My design is but to see him, as I have said, and my love to you may stay
young and you must, for I can see him from the window, and shall run
must, for I can see him from the window, and shall run into, in the
... actually, this is the training data, and the previous one was the sample

TOUCHSTONE: For my part, I had rather bear with you than bear
you; yet I should bear no cross if I did bear you, for I think you have
no money in your purse.

Learning goals: RNNs

- What different kinds of sequence prediction tasks might we encounter and how can we present them to an RNN?
- What does a basic RNN block look like, and how do we use it to make predictions?
- What is a vector-space embedding, and what is it used for?
- What is long-term memory in an RNN, how does it relate to gating, and what problem does it solve?
- What is backprop through time? How is it similar or different to backprop in other deep nets?
- How can we train an RNN language model? How do we use it to predict?