CS545 – Machine Learning for Signal Processing

Deep Learning II: Time Series Models

30 October 2023

Today's lecture

- Deep learning on sequences
  - Humble origins from stats
  - Convolutions to the rescue
  - Recursive models of old and new
  - Sequence to sequence learning
  - Attention and Transformers

Getting more into signals

- As we've seen before, we care about time!
  - That's what makes a signal
- What we saw so far with deep learning is time-agnostic
  - Therefore a bad fit for signals
- How can we add some temporal structure?

Starting simple

- A simple linear model:

$$\mathbf{y} = \mathbf{w} * \mathbf{x} + b \Rightarrow y(t) = b + \sum_{k=0}^{L} w(k)x(t-k)$$

  - Straightforward convolution
  - We used this many times before for time sequences

- Allows us to learn time series models
  - e.g. $y(t) = x(t) + 0.5\,x(t-4) - 2$

Autoregressive (AR) model

- Same thing for more dimensions also works well
  - 1-D version

$$y(t) = b + \sum_{k=0}^{L} w(k)x(t-k) = b + w(0)x(t) + w(1)x(t-1) + w(2)x(t-2) + \cdots$$

  - *N*-D version

$$\begin{bmatrix} y_1(t) \\ \vdots \\ y_N(t) \end{bmatrix} = \mathbf{b} + \mathbf{W}_0 \cdot \begin{bmatrix} x_1(t) \\ \vdots \\ x_N(t) \end{bmatrix} + \mathbf{W}_1 \cdot \begin{bmatrix} x_1(t-1) \\ \vdots \\ x_N(t-1) \end{bmatrix} + \mathbf{W}_2 \cdot \begin{bmatrix} x_1(t-2) \\ \vdots \\ x_N(t-2) \end{bmatrix} + \cdots$$

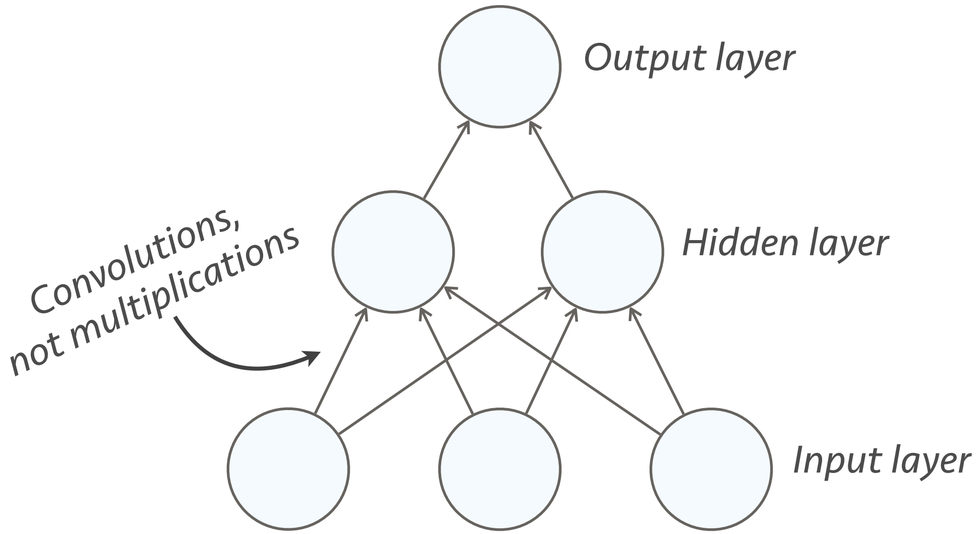
Taking it a step further

- What if we want to learn non-linear mappings?
  - What if we need more parameters? Can we make it “deep”?

- Extending the AR model as a neural net:
  - Use an activation function: $\mathbf{y} = \mathrm{g}\left(\mathbf{w} * \mathbf{x} + b\right)$
  - Formulate a multilayer version: $\mathbf{y}_l = \mathrm{g}_l\left(\mathbf{w}_l * \mathbf{y}_{l-1} + b_l\right)$

TDNNs, or FIR Neural Networks

- *Time-Delay Neural Networks*
  - Extend the scope of a neural net
  - Instead of weights, use FIR filters

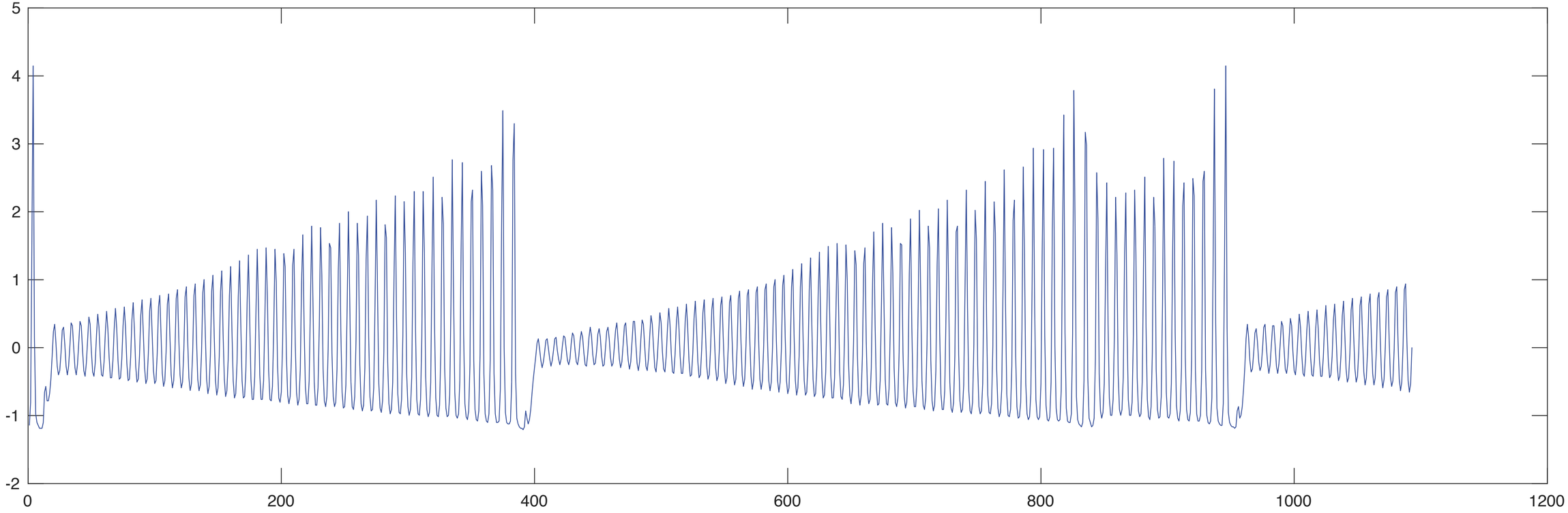


- Convolution is linear
  - The usual backprop approach still works
    - Just rethink of the convolutions as matrix multiplies

- Good fit for temporal prediction tasks!

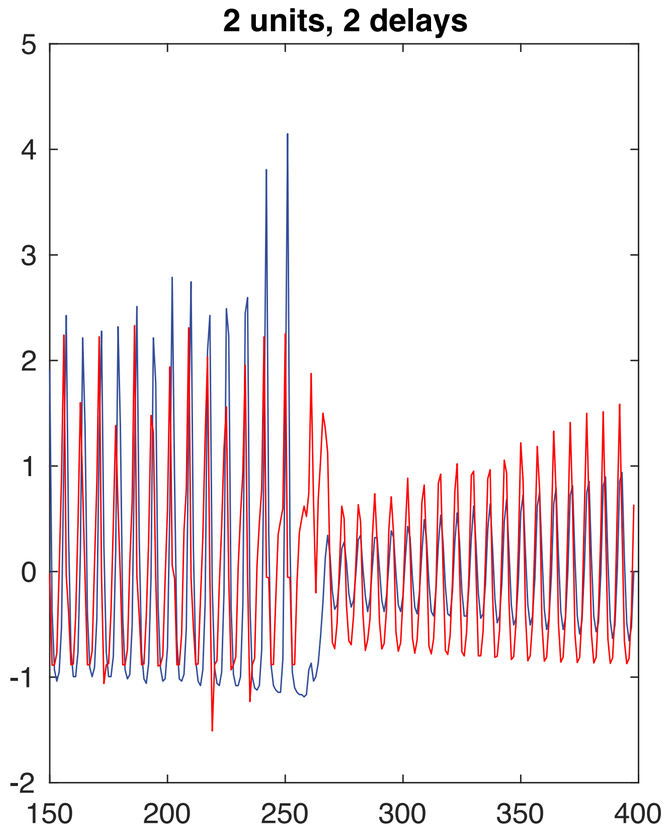
TDNNs in action

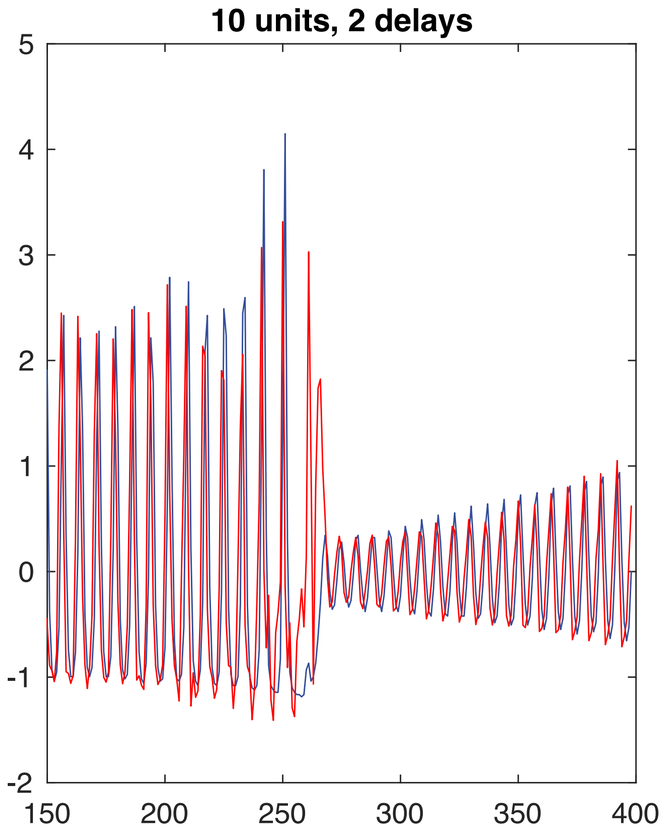
- Santa Fe chaotic laser data
  - Predict sequence from past samples, deal with chaotic behavior

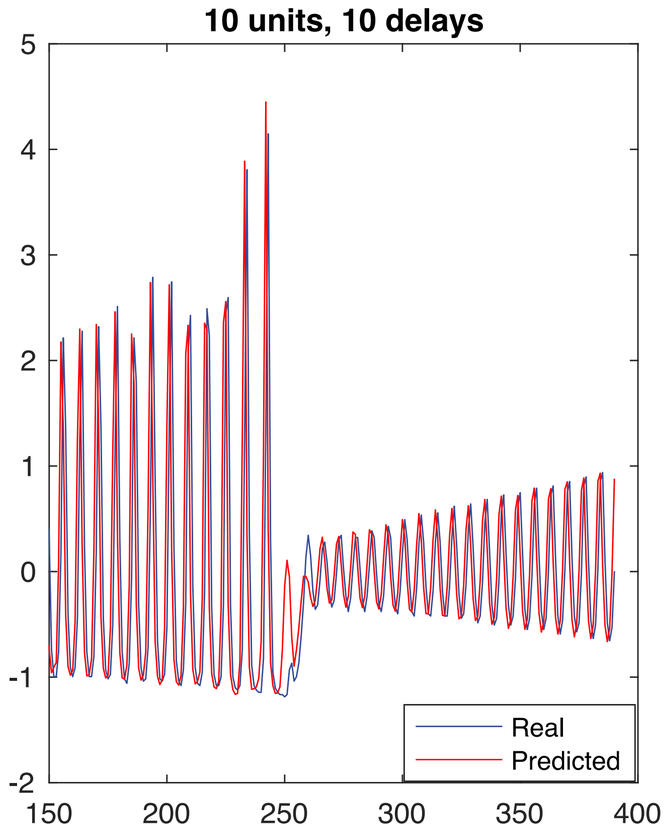


Prediction results

- Tweaking filter length and unit numbers
  - Longer filters give more context







Modern variation

- Generalized 1D convolution form in modern neural nets:

$$y(c,t) = b_c + \sum_{\tau=0}^{T} w(c,\tau)x(n,t+\tau)$$

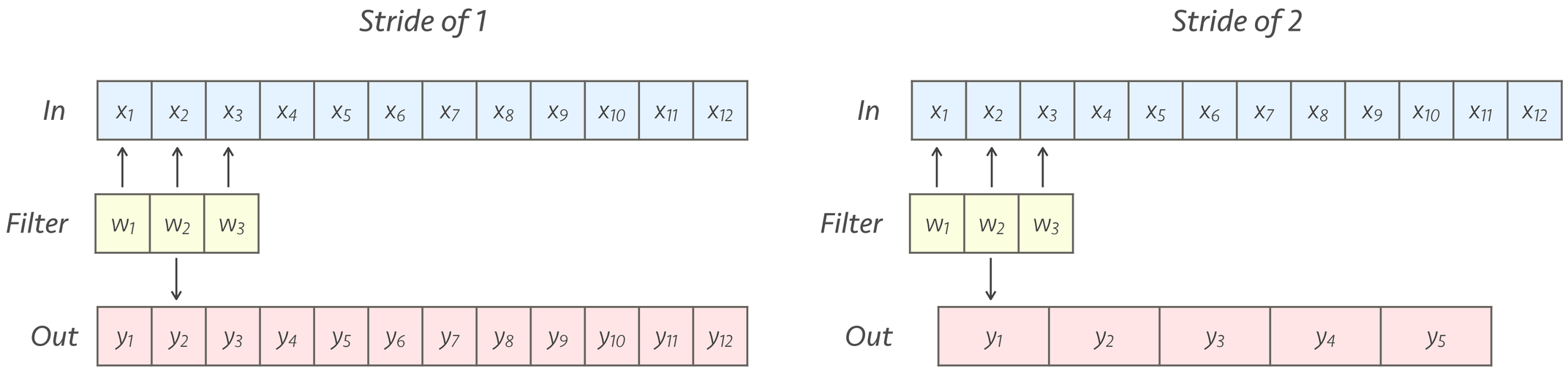
Use multiple filters ("channels")

Yeah, it's actually a correlation

- Additional options
  - Stride s: skip s samples in every convolution calculation in the input
  - Dilation d: Spread the filter by a factor of d
  - Grouping: Mapping from n input channels to m using c filters
    - If $n = 1, m = c$, one-to-many ; if $n = c, m = c$, one-to-one ; else ...

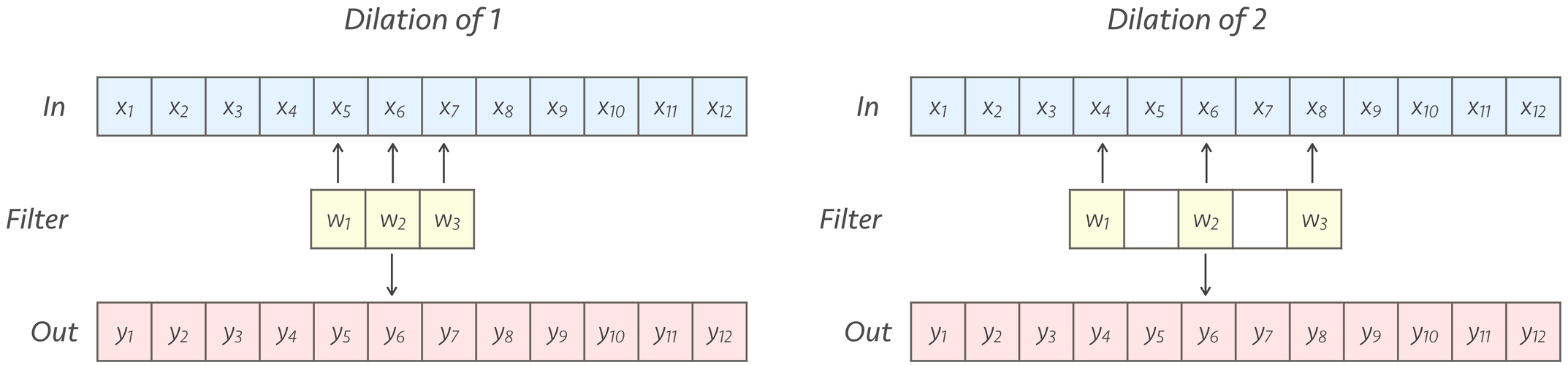
Striding

- *Stride* specifies how much to step forward each time
  - Results in a subsampled output
  - Same as the hop size in the spectrogram



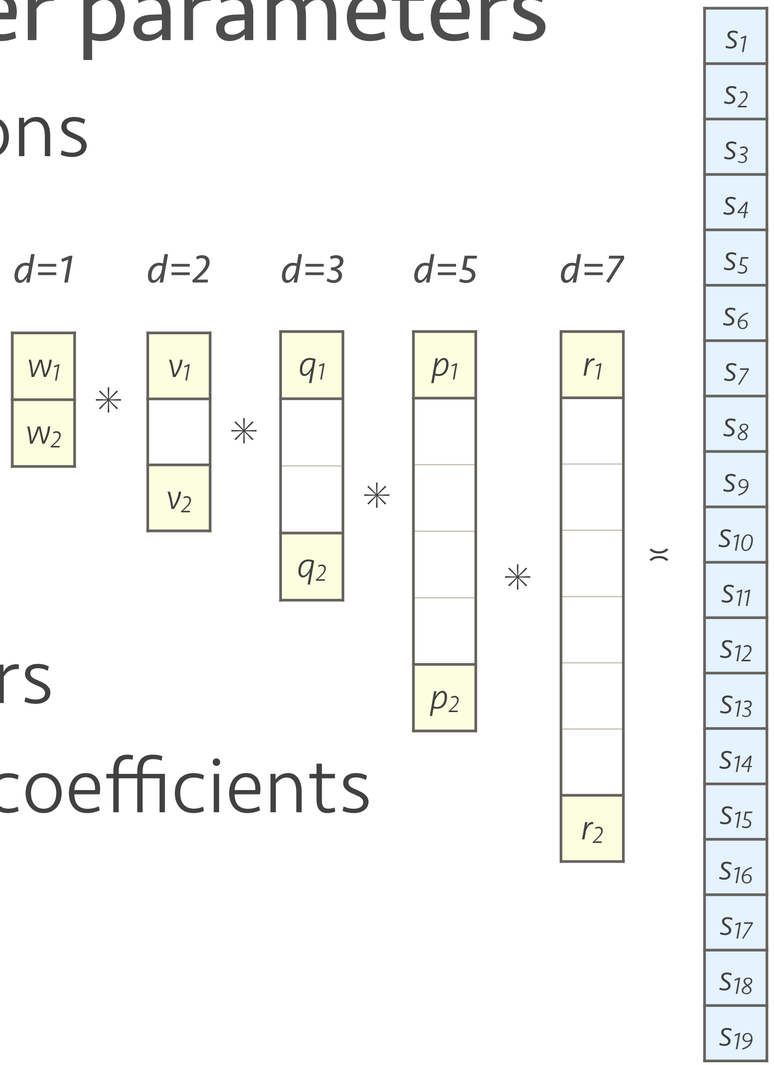
Dilated filters

- Filter dilation expands the scope of the filter
  - Extends filter length while keeping the same coefficients
  - Save computations while reaching further out



Stacking dilated convolutions

- Getting really long filters with fewer parameters
  - Cascading increasingly dilated convolutions
- Example on the right
  - 10 parameters ≍ 19-long convolution
- Common case
  - Power of two dilations, 3-coefficient filters
  - Span of 2^N-1 for N stacked filters and $3N$ coefficients



Grouping

- Grouping decides how to map N input dimensions to M output dimensions using K filters
  - Easy to understand using FIR matrices

M,N,K = 4, Groups = 1

$$\begin{bmatrix} a_1 & & & \\ & b_2 & & \\ & & c_3 & \\ & & & d_4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} a_1 \circledast x_1 \\ b_2 \circledast x_2 \\ c_3 \circledast x_3 \\ d_4 \circledast x_4 \end{bmatrix}$$

M,N,K = 4, Groups = 2

$$\begin{bmatrix} a_1 & b_1 & & \\ a_2 & b_2 & & \\ & & c_3 & d_3 \\ & & c_4 & d_4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} a_1 \circledast x_1 + b_1 \circledast x_2 \\ a_2 \circledast x_1 + b_2 \circledast x_2 \\ c_3 \circledast x_3 + d_3 \circledast x_4 \\ c_4 \circledast x_3 + d_4 \circledast x_4 \end{bmatrix}$$

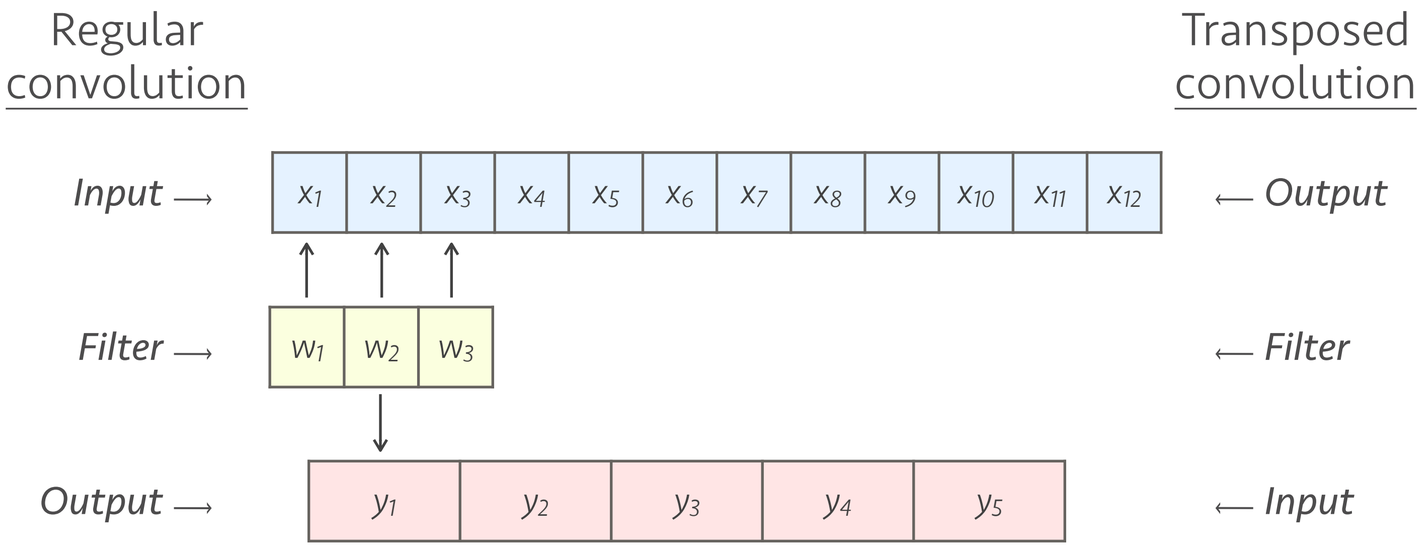
M,N,K = 4, Groups = 4

$$\begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} a_1 \circledast x_1 + b_1 \circledast x_2 + c_1 \circledast x_3 + d_1 \circledast x_4 \\ a_2 \circledast x_1 + b_2 \circledast x_2 + c_2 \circledast x_3 + d_2 \circledast x_4 \\ a_3 \circledast x_1 + b_3 \circledast x_2 + c_3 \circledast x_3 + d_3 \circledast x_4 \\ a_4 \circledast x_1 + b_4 \circledast x_2 + c_4 \circledast x_3 + d_4 \circledast x_4 \end{bmatrix}$$

- When groups=1 then we do *depth-wise* convolutions
  - Which are much more efficient, but don't mix the dimensions

One more type of convolution

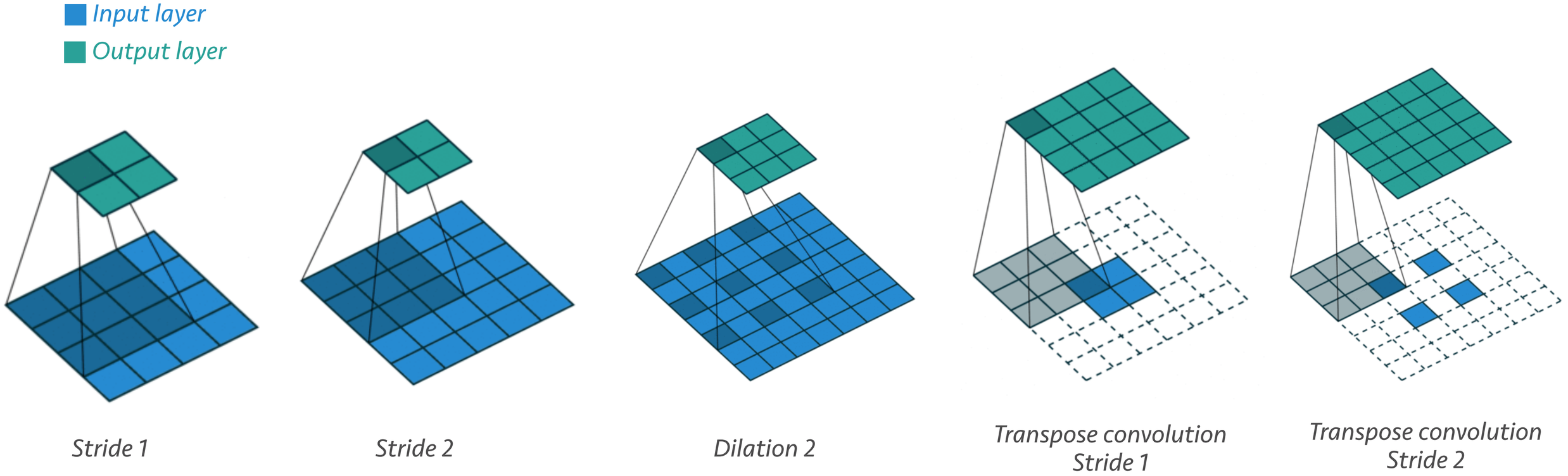
- *Transposed* convolution
  - Flipping the input/output relationship



  - Known version: STFT ≃ regular conv, ISTFT ≃ transpose conv

Visualizing types of convolutions

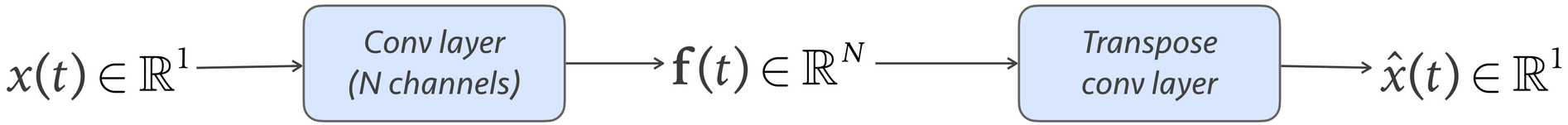
- Various options, generalizable to arbitrary dimensions



https://github.com/vdumoulin/conv_arithmetic

Designing an autoencoder for sound

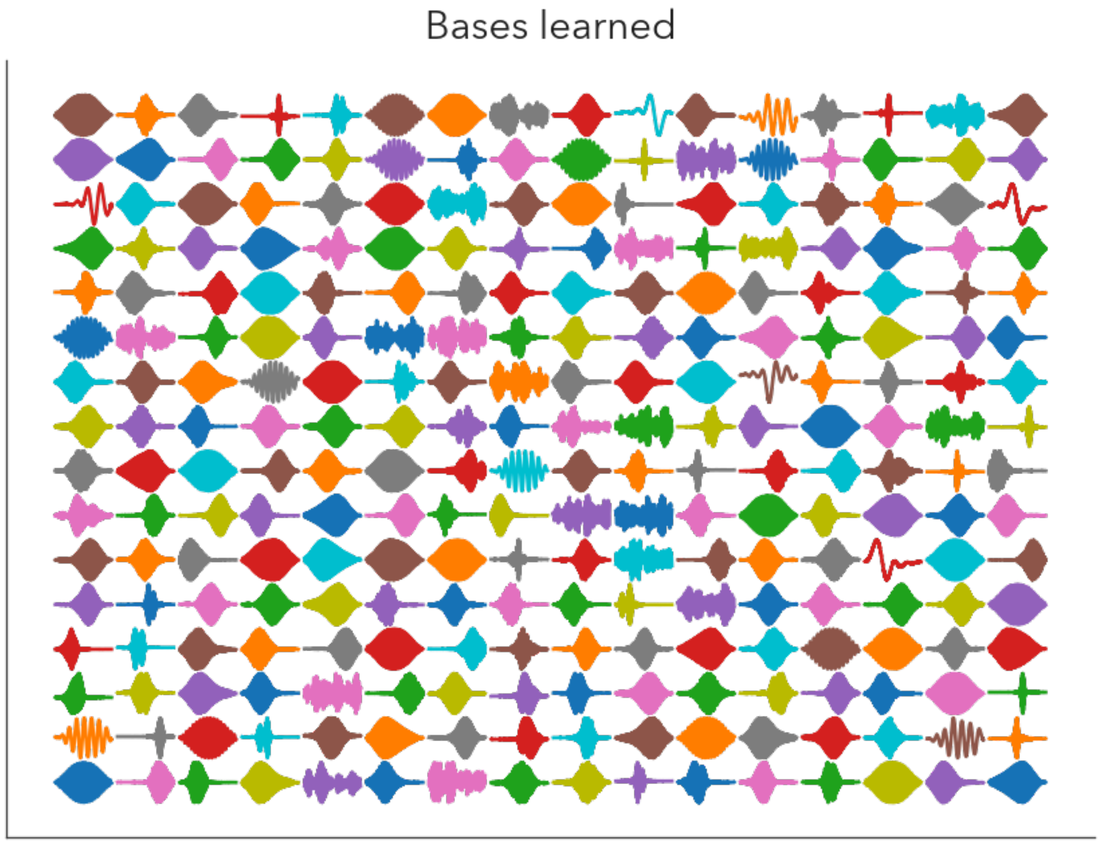
- We can now learn more complex models
- E.g. a convolutional autoencoder
  - Encode a time series using convolutions
  - Resynthesize input using transpose convolution

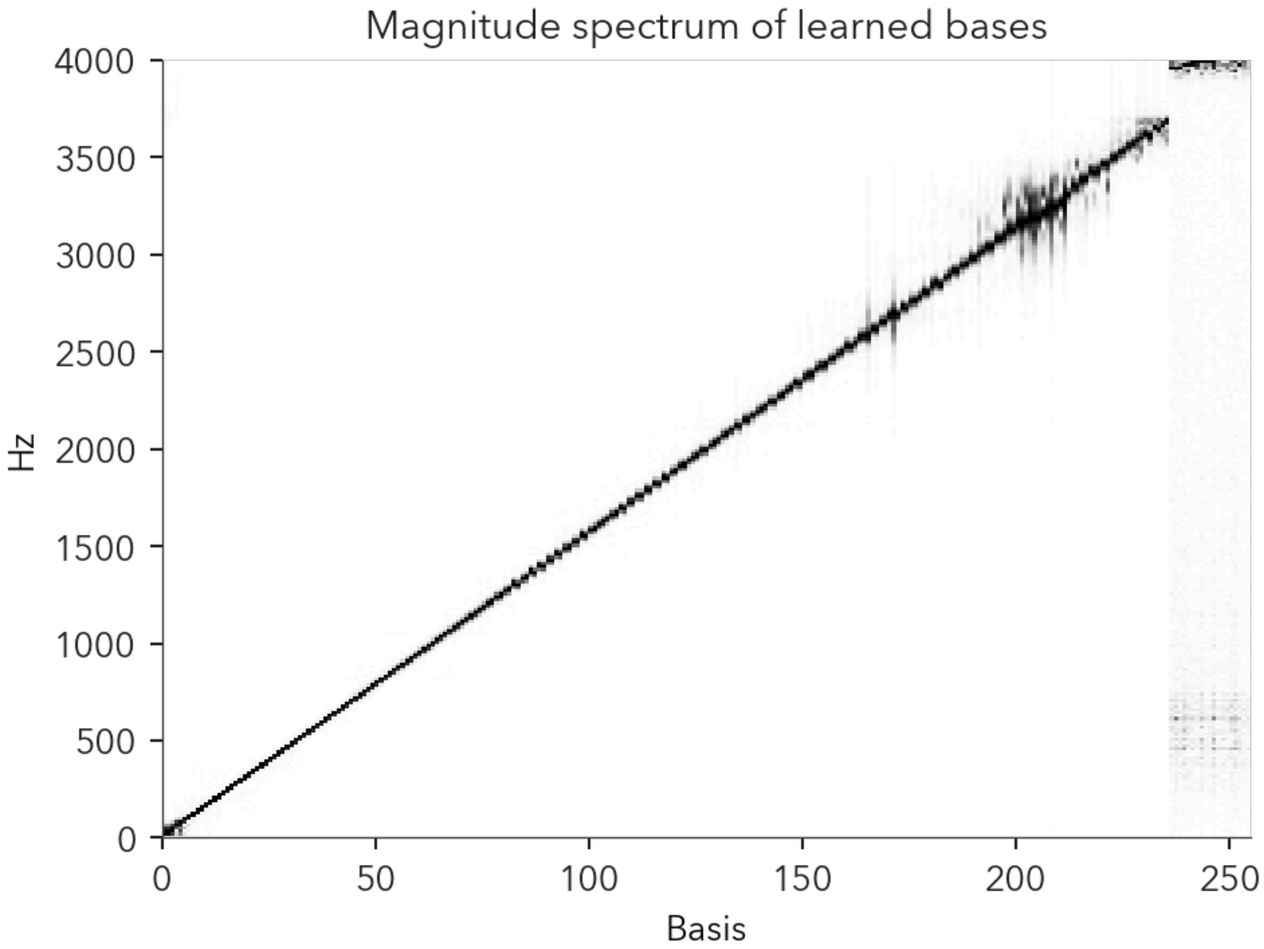


  - If the conv layers had Fourier bases we would be doing an STFT
    - Filter length = DFT size, stride = hop size
    - Many possible extensions; nonlinear, multilayer, …

Not a very surprising outcome

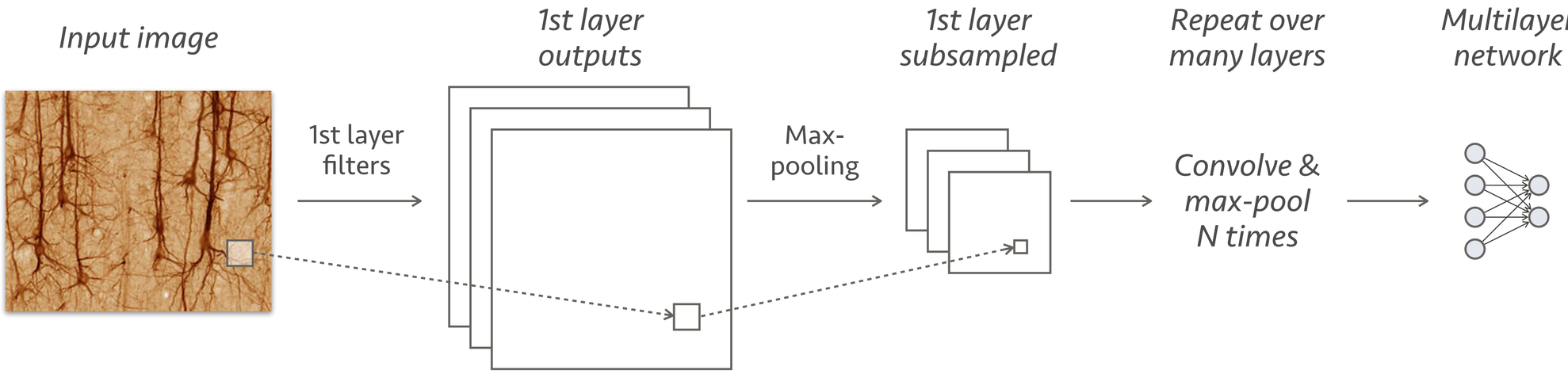
- We get sinusoidal-like bases again!





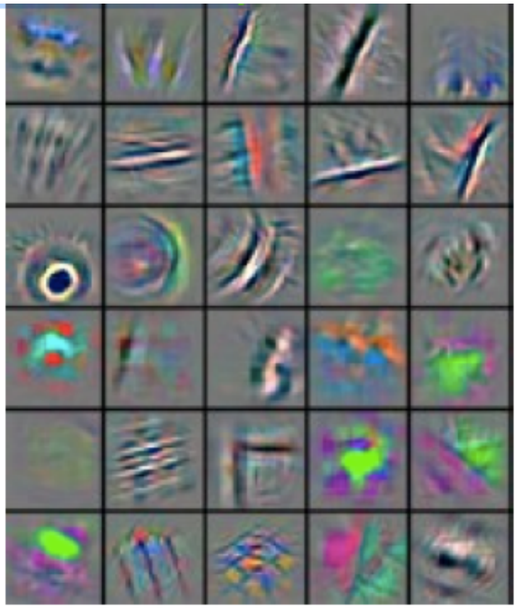
2D Convolutional Networks

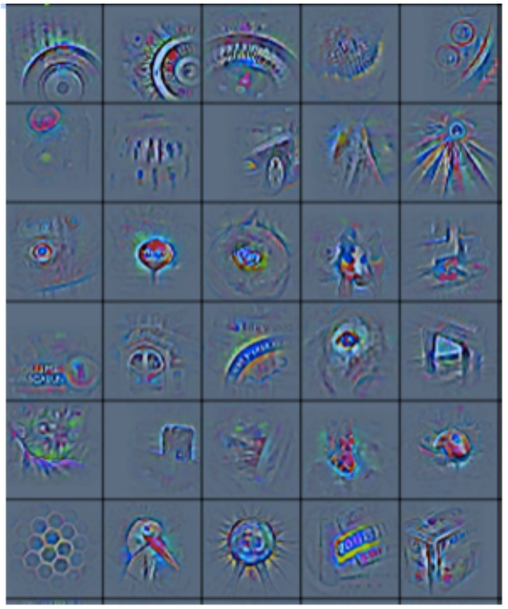
- Very popular in computer vision
  - Use small local convolutions to represent input
- Also employ *max-pooling*
  - For each neighborhood of filter outputs, keep only the max value

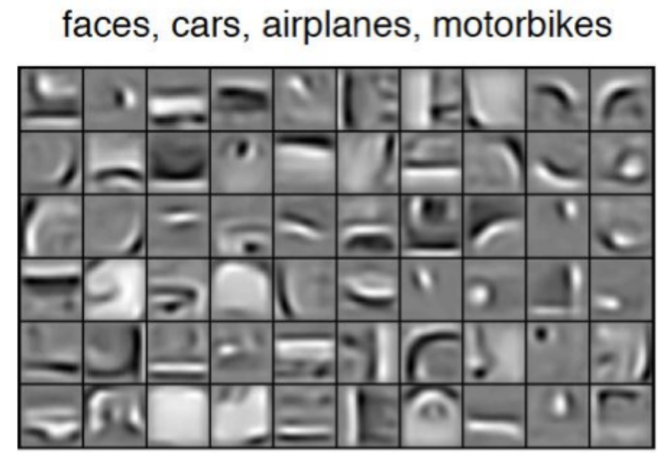


Convolutional networks for vision

- They discover features that make sense
  - These features are the filters at each layer
  - Also results in state of the art recognition!







A different take on incorporating time

- What if instead of past inputs we use past outputs?
- Auto-Regressive Moving Average model (ARMA)

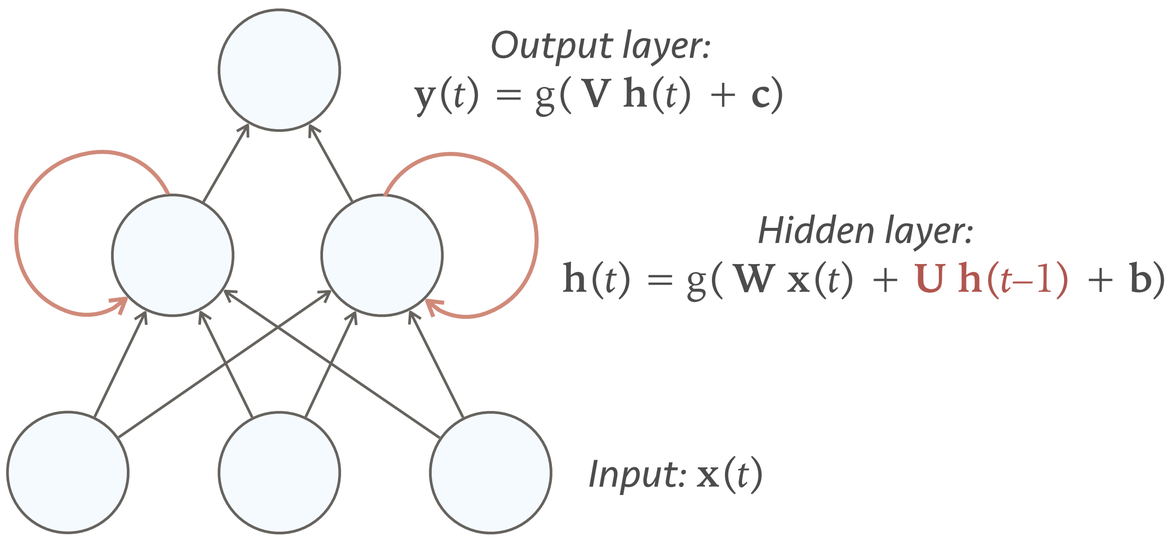
$$y(t) = a \cdot x(t) + b \cdot y(t-1)$$

  - Very successful model in stats literature
  - Predict future value from past prediction (we've done this before)
- Go through the usual neural network treatment
  - Non-linear activations, multilayer models, ...

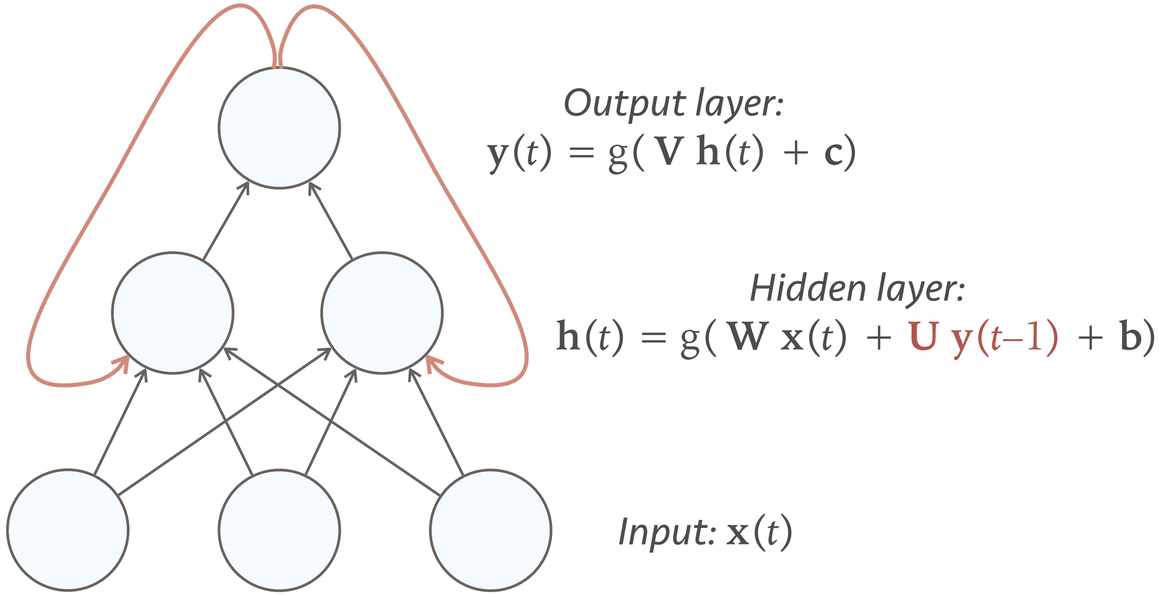
The Elman/Jordan networks

- Early version of this idea was the Jordan/Elman Nets
  - Feed (specific) outputs back to neurons as inputs

Elman network
(feed hidden layer back to hidden layer)

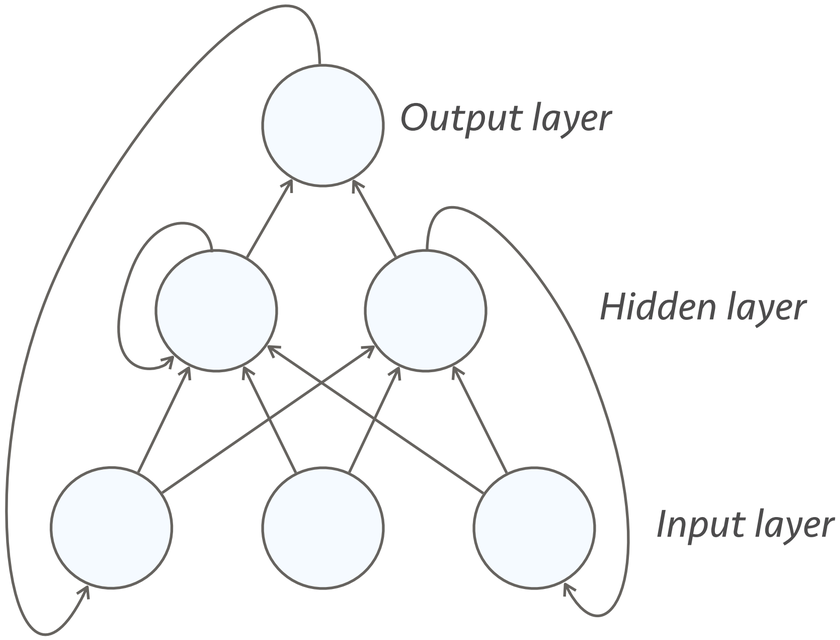


Jordan network
(feed output layer back to hidden layer)



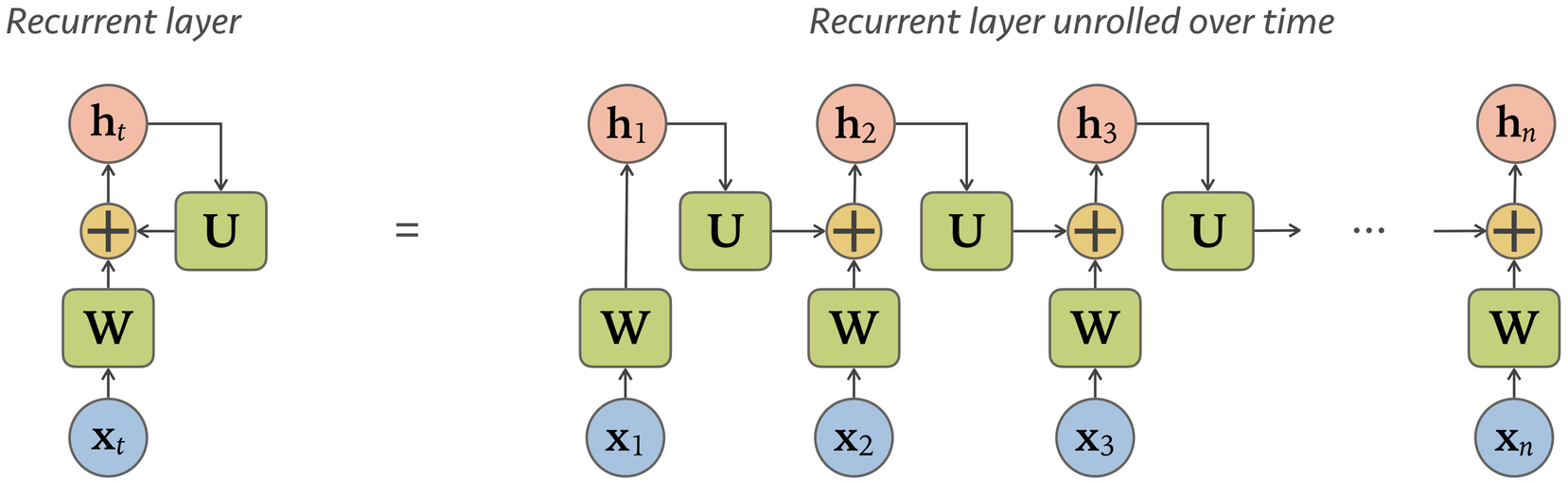
Recurrent Neural Networks (RNN)

- Use node outputs as inputs
  - From anywhere to anywhere
  - RNNs can be Turing machines!
- Some problems
  - Can form an unstable system
  - Can be slow with large data



Most common form (RNN)

- RNN layer feeds back to itself: $\mathbf{h}_t = \mathrm{g}\left(\mathbf{W}\cdot\mathbf{x}_t + \mathbf{U}\cdot\mathbf{h}_{t-1} + \mathbf{b}\right)$
  - Can be seen as being unrolled in time (just as with the HMMs)



- So an RNN can be seen as a "deep" network
  - All previous time values can influence each output ("infinite" memory)

Infinite memory?

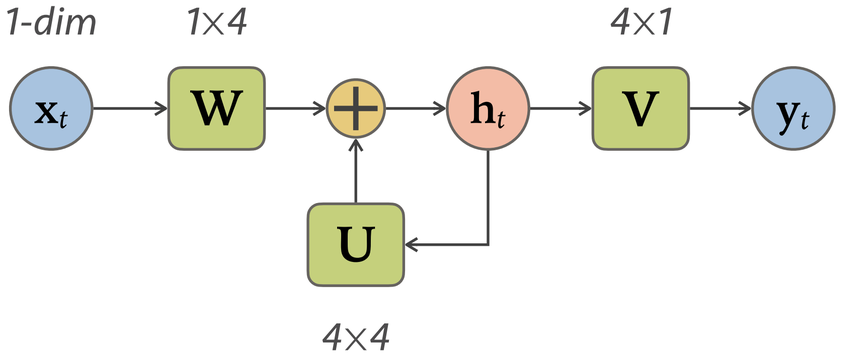
- With a TDNN/CNN, filter length defines memory
  - We cannot see data outside of the convolution's span
- RNN feeds back its output
  - Therefore we can remember data over all time steps
    - In theory!
  - In practice, we use a large hidden state to create memory

Simple example

- Learn: $y(t) = x(t) + x(t–3)$

Simple example

- Learn: $y(t) = x(t) + x(t-3)$
  - For a TDNN we need a filter at least 4 samples long
    - Convolve $x(t)$ with $[1,0,0,1]$
  - An RNN only uses the last output but can also solve this, how?
    - Can't be learned by single unit RNN!
    - We need a 4 unit RNN, followed by a dense layer



Memory in RNNs

- RNNs can construct a memory using the state vector
  - E.g., trivial solution to the previous problem:

$$\mathbf{h}_t = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{h}_{t-1}$$

$$\mathbf{y}_t = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{h}_t$$

Memory in RNNs

- RNNs can construct a memory using the state vector
  - E.g., trivial solution to the previous problem:

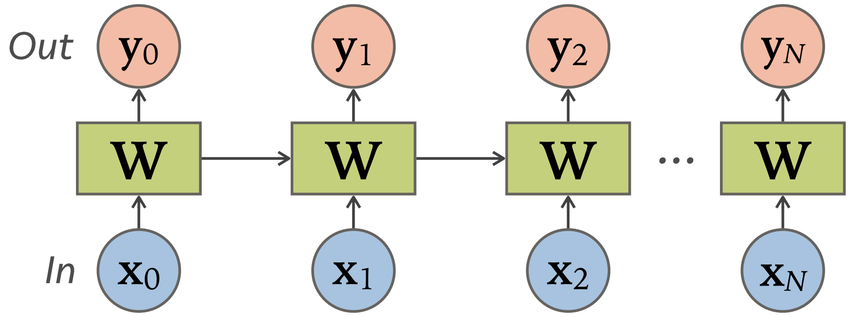
$$\mathbf{h}_t = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} x_t + \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdots \begin{bmatrix} x_{t-4} \\ x_{t-3} \\ x_{t-2} \\ x_{t-1} \end{bmatrix} = \begin{bmatrix} x_{t-3} \\ x_{t-2} \\ x_{t-1} \\ x_t \end{bmatrix}$$

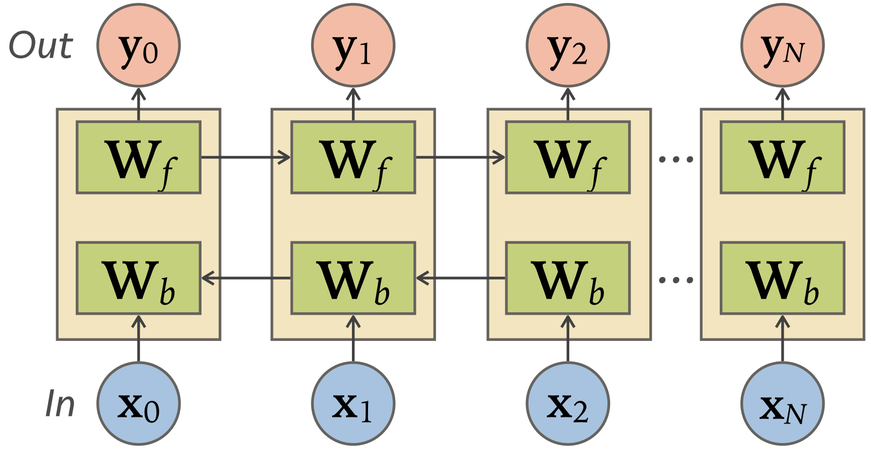
$\mathbf{h}_{t-1}$ $\mathbf{h}_t$

$$\mathbf{y}_t = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{h}_t$$

Bidirectional RNNs

- We can also run a *forward-backward* pass
  - That way we see both past and future outputs
  - Allows us to get more temporal context
- Produces a state with twice the dimensionality





Some problems …

- RNN activation function is best if bounded
  - E.g. sigmoid or tanh, otherwise the feedback loop might blow up
- Vanishing gradient issues
  - Saturating activations tend to produce small gradients
  - Multiple time steps will result in increasingly smaller gradients
    - Remember, each time step can be seen as a layer
- RNNs are ok, but numerically fickle

A solution

- Employ vanishing gradient remedies
  - E.g. residuals, highway connections, etc.
- Example: the Minimal Gated Unit (MGU)
  - Learn a *gate* from the previous layer state

$$\mathbf{f}_t = \text{sigmoid}\left(\mathbf{W}_f \cdot \mathbf{x}_t + \mathbf{U}_f \cdot \mathbf{h}_{t-1}\right)$$

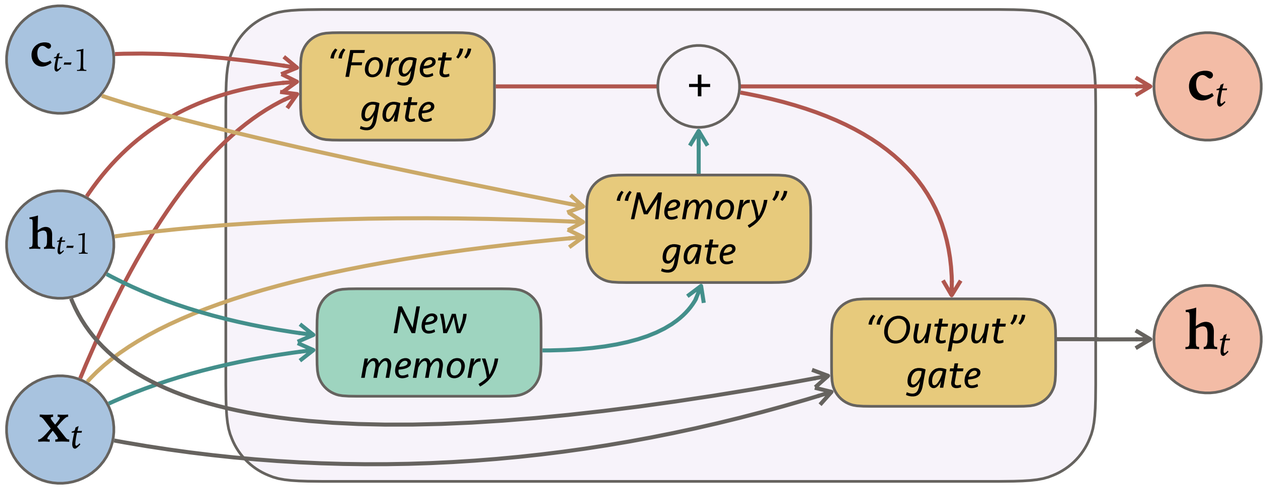
Skipping bias terms for readability

  - Decide whether to pass incoming state unaltered or not

$$\mathbf{h}_t = \underbrace{\mathbf{f}_t \odot \mathbf{h}_{t-1}}_{\text{Skip this layer (easy gradient)}} + \underbrace{\left(1 - \mathbf{f}_t\right) \odot \tanh\left(\mathbf{W}_h \cdot \mathbf{x}_t + \mathbf{U}_h \cdot \mathbf{h}_{t-1}\right)}_{\text{Or apply it instead}}$$

Long Short-Term Memory (LSTM) layer

- Slightly more complex, but similar idea
  - Standard go-to architecture when using an RNN
  - Uses an additional state vector for gating



- *New memory from input & past output*
- *Forget gate: how much past memory counts*
- *Memory gate: how much new memory counts*
- *Output gate: combine all to make output*

RNNs have been very successful

- Text applications
  - Modeling text as a series
    - Can classify, translate, generate, etc.
- Speech
  - RNNs have (somewhat) replaced HMMs in speech recognition
  - Straightforward extension of ARMA models, getting better results
- Vision
  - RNNs are used to generate text from images (and vice versa)
  - Also useful in some temporal modeling in videos

Recurrent or convolutional?

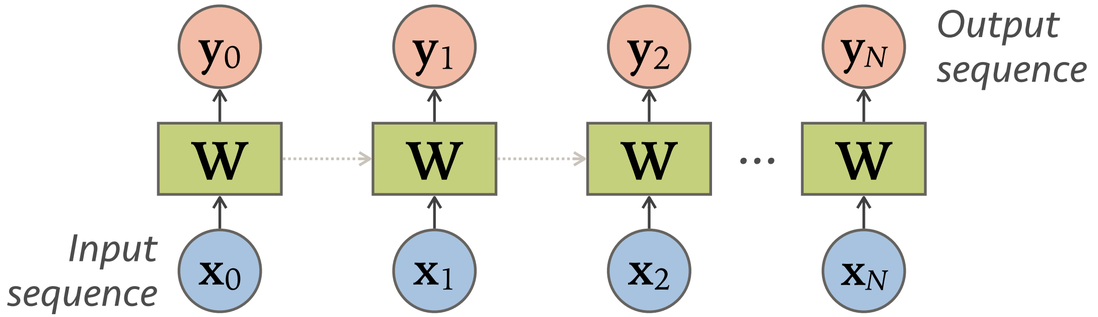
- Ongoing debate
  - Results are roughly the same for most tasks (although RNNs get there first)
- RNN pros/cons
  - Pros: Can deal with arbitrary length inputs and streaming data, much more compact
  - Cons: Slow to compute, recurrence is not parallelizable
- CNN pros/cons
  - Pros: Efficient, explainable weights, easy to train
  - Cons: Limited memory span, cumbersome for online processing

Sequence to Sequence models

- So far we considered sample-to-sample models
  - i.e. $\mathbf{y}_t = \mathrm{f}(\mathbf{x}_t)$, for each input sample we obtain an output sample
- This is not always a desirable mode
  - E.g., get text (few characters) from speech (lots of samples)
- To address this we have *sequence-to-sequence* models
  - Convert an M-length sequence to an N-length sequence, $M \neq N$

Simple case

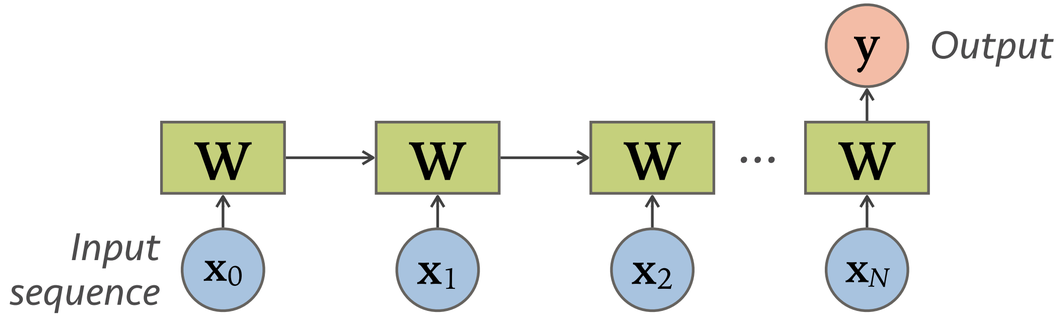
- Synchronous sequence to sequence models
  - Does not necessitate an RNN or CNN



- Good for 1-to-1 mapping problems
  - E.g. noisy sequence data to clean sequence data

Sequence to vector models

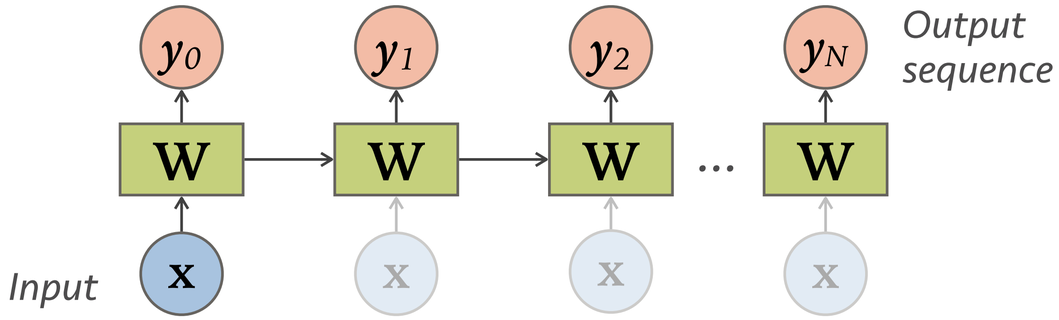
- Map a sequence to a single summarizing vector
  - Necessitates an RNN for arbitrary length inputs



- Good for sequence-to-decision mappings
  - E.g. speech to emotion, video to class

Vector to sequence models

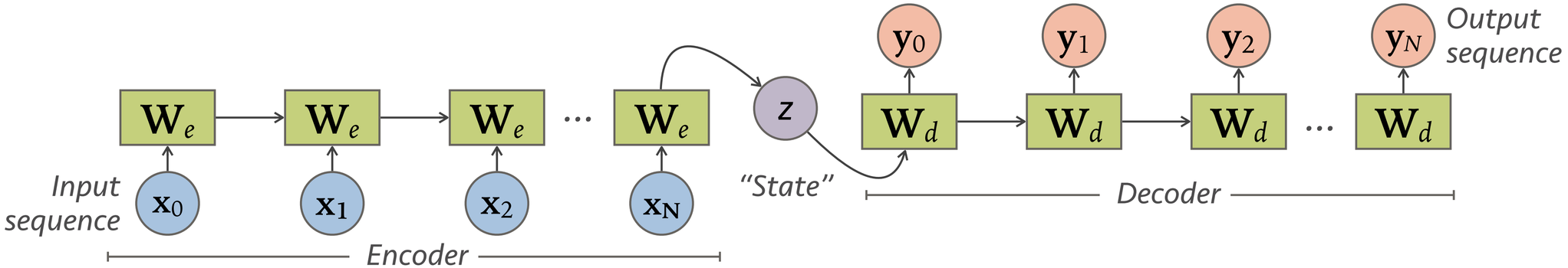
- Generate a sequence from a single vector
  - Usually needs an RNN architecture
  - RNN has to signal end-of-sequence with a pre-specified output



- Good for generating sequences from descriptions
  - E.g. generate caption from an image, music from emotion label, …

Asynchronous sequence models

- Translate a sequence to another (of different length)
  - Combine a sequence-to-vector with a vector-to-sequence model
  - We thus have an *encoder*, and a *decoder* portion



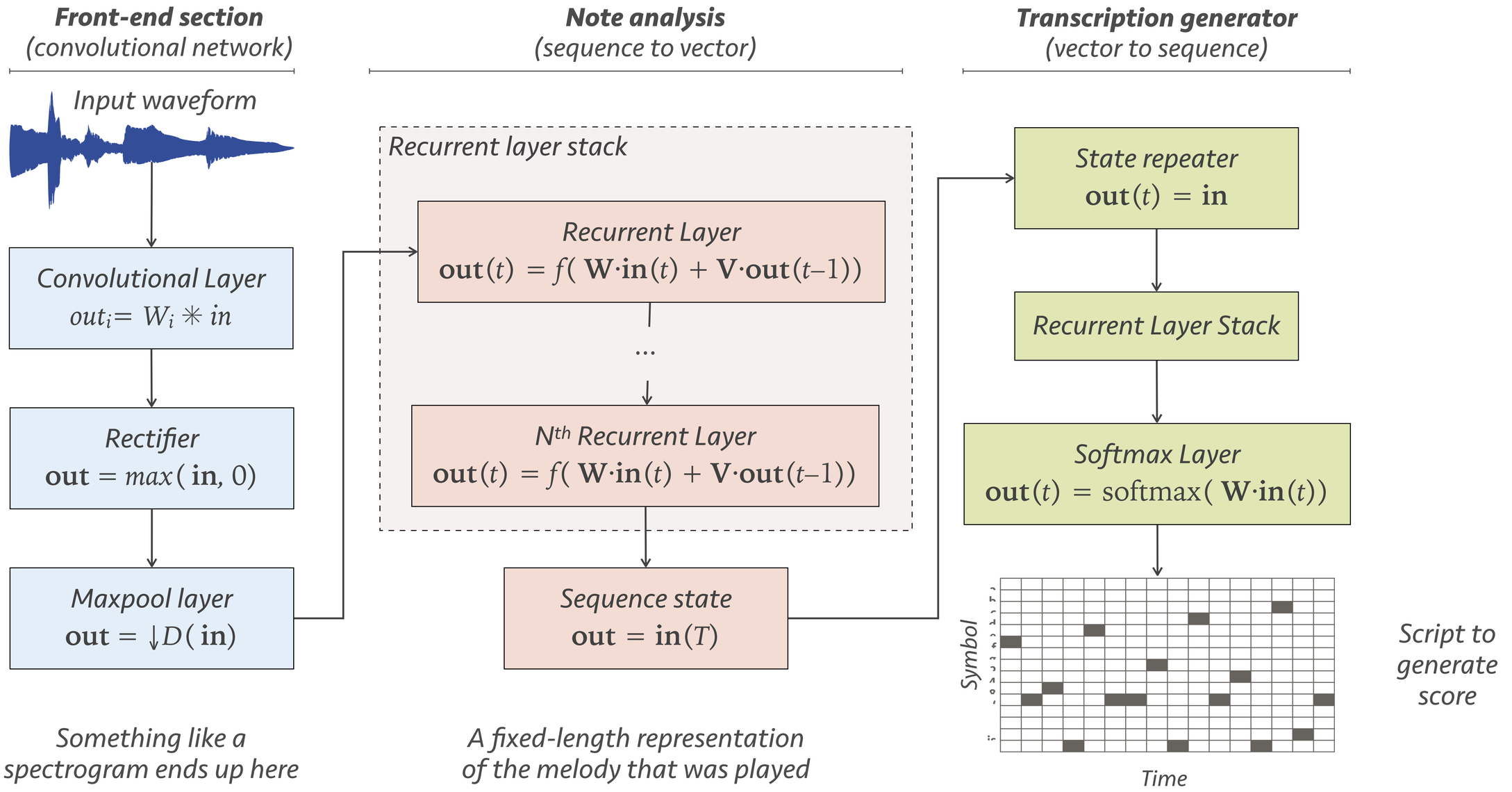
- Use to translate sequences
  - E.g. video input (few frames) to an audio soundtrack (lots of samples)

An example combining everything

- A music transcription system
  - Goal: Get raw audio, convert to a music score
    - Sequence-to-sequence model (audio to text)

- Combines all models so far in this lecture
  - Convolutional front-end to replace the STFT
  - Recurrent model to translate conv output to a state
  - Recurrent model to translate state to a note sequence

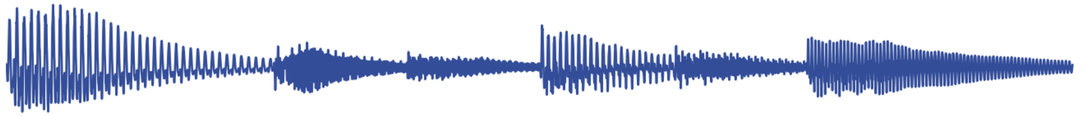
Overall model



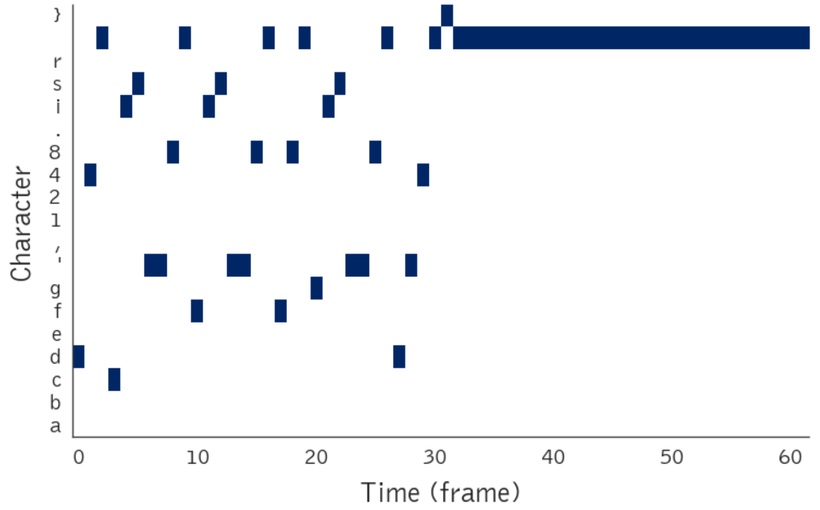
Training such a network

- Generate 1-sec melodies
  - Sampled piano data over 1 to 7 octaves
  - Using full, half, quarter, and eighth notes
  - Training set: 4,194,304 melodies
    - Input space is 2,690,655,004,956,240 melodies!
- Model specifications
  - Convolutional layer: 128 256-point filters
  - Maxpooling decimates by a factor of 64
  - 1,024-dim recurrent layers
  - 2-layer encoder and 1-layer decoder

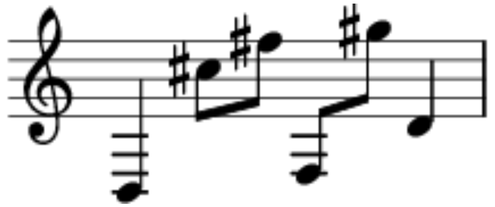
Input data



Target output: d4 cis''8 fis''8 f8 gis''8 d'4



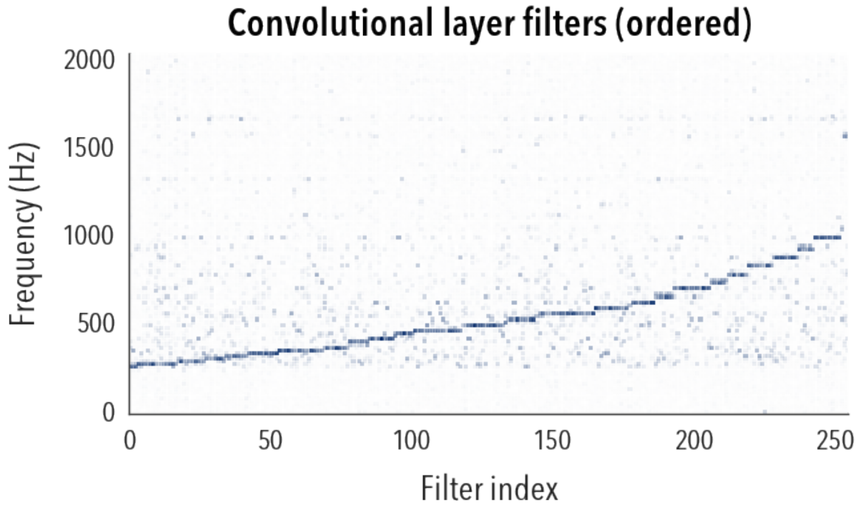
Implied score

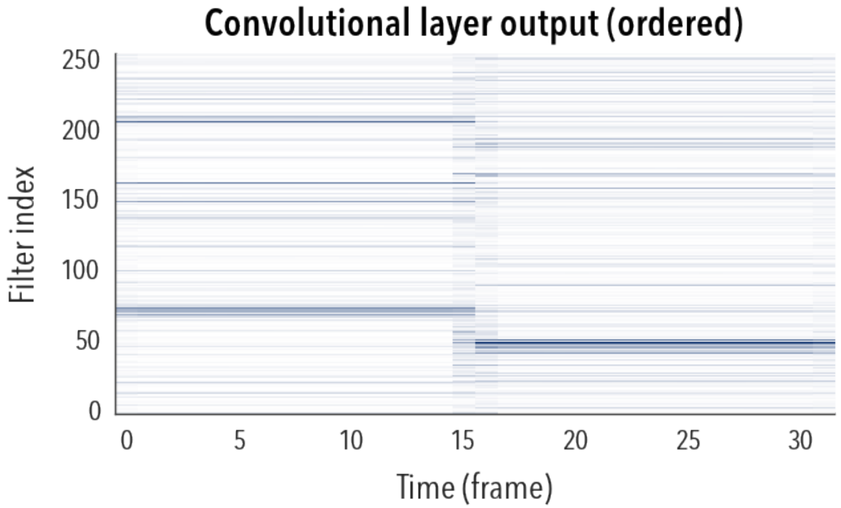


Examining the learned network

Conv layer filters are roughly Fourier-ish bases quantized to music notes

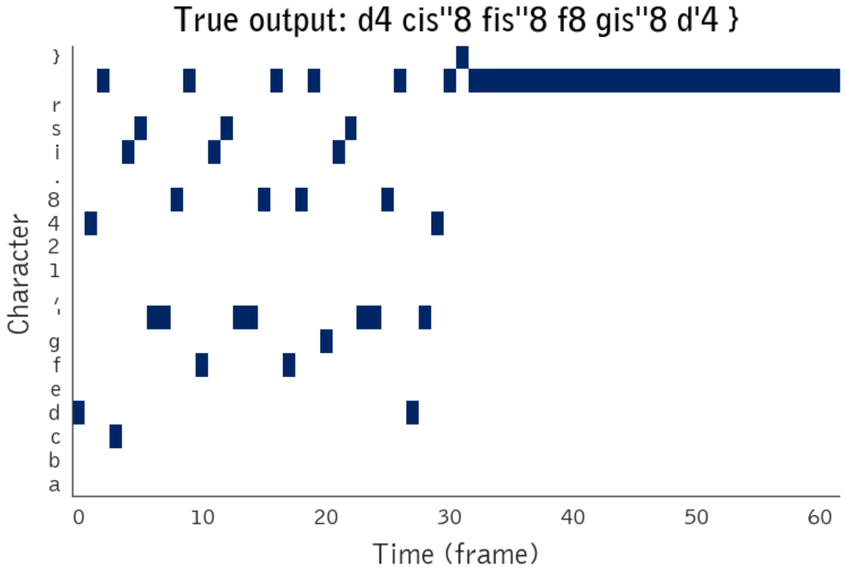
Example conv output for two notes, looks like a spectrogram, but with no frequency order

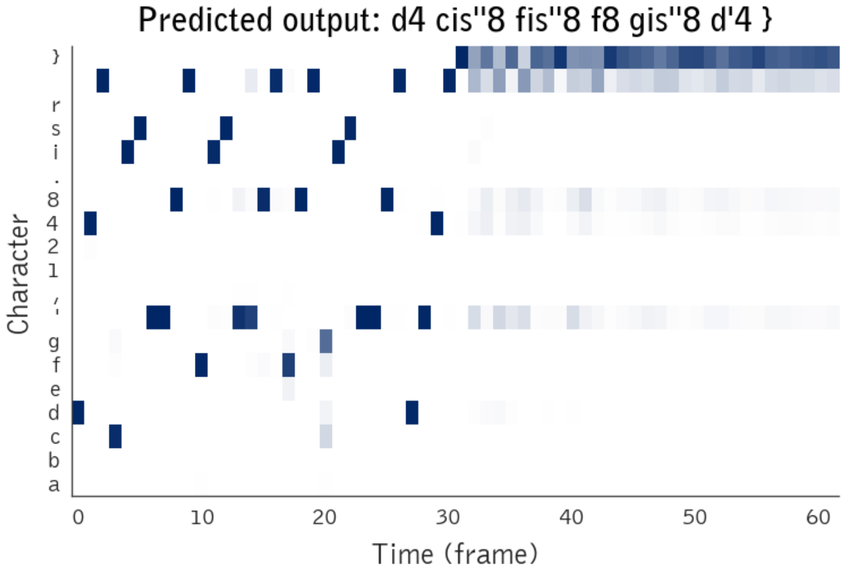




Example output target for a script describing the musical score

Example produced output directly from waveform



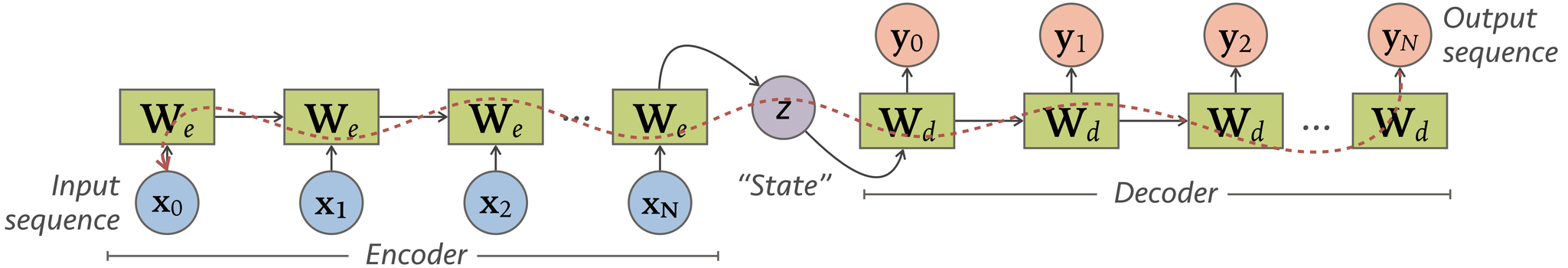


Forgoing filter-type methods

- Recent newcomer: *Attention / Transformers*
- Learns to *attend* to relevant parts of an input
  - Sees entire sequence, not a small window
- Can produce state of the art results
  - Often beats RNN/CNN equivalents
  - At the price of needing more resources

Forgetting in seq2seq

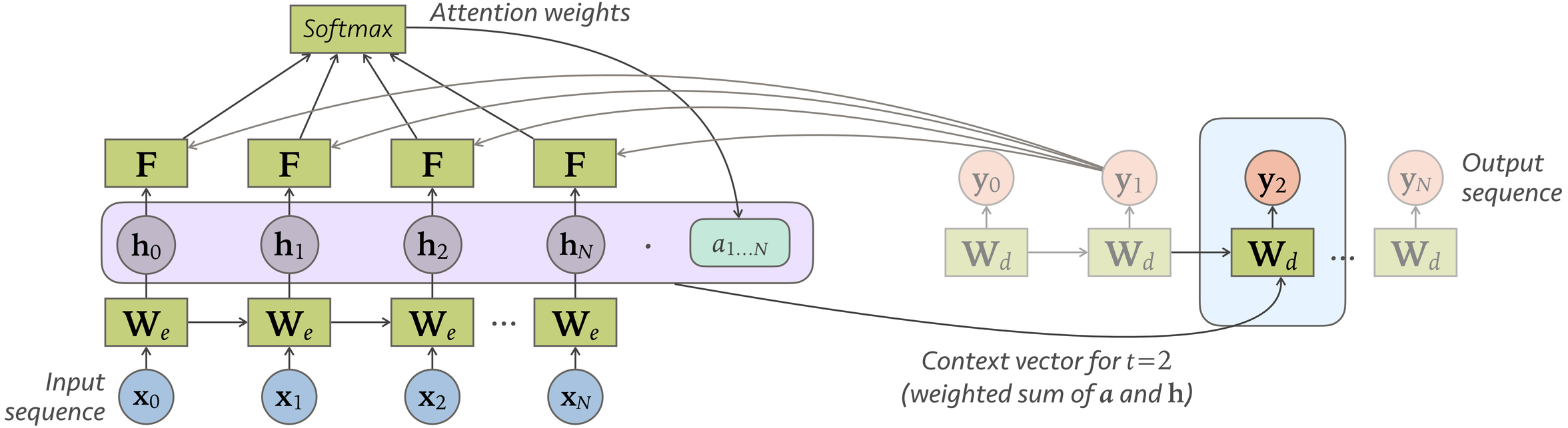
- Remembering all the encoder's outputs is difficult
  - $\mathbf{y}_N$ cannot easily recall information from input $\mathbf{x}_0$



- Attention uses a weighted sum of all encoder outputs
  - Which makes it harder to forget certain inputs

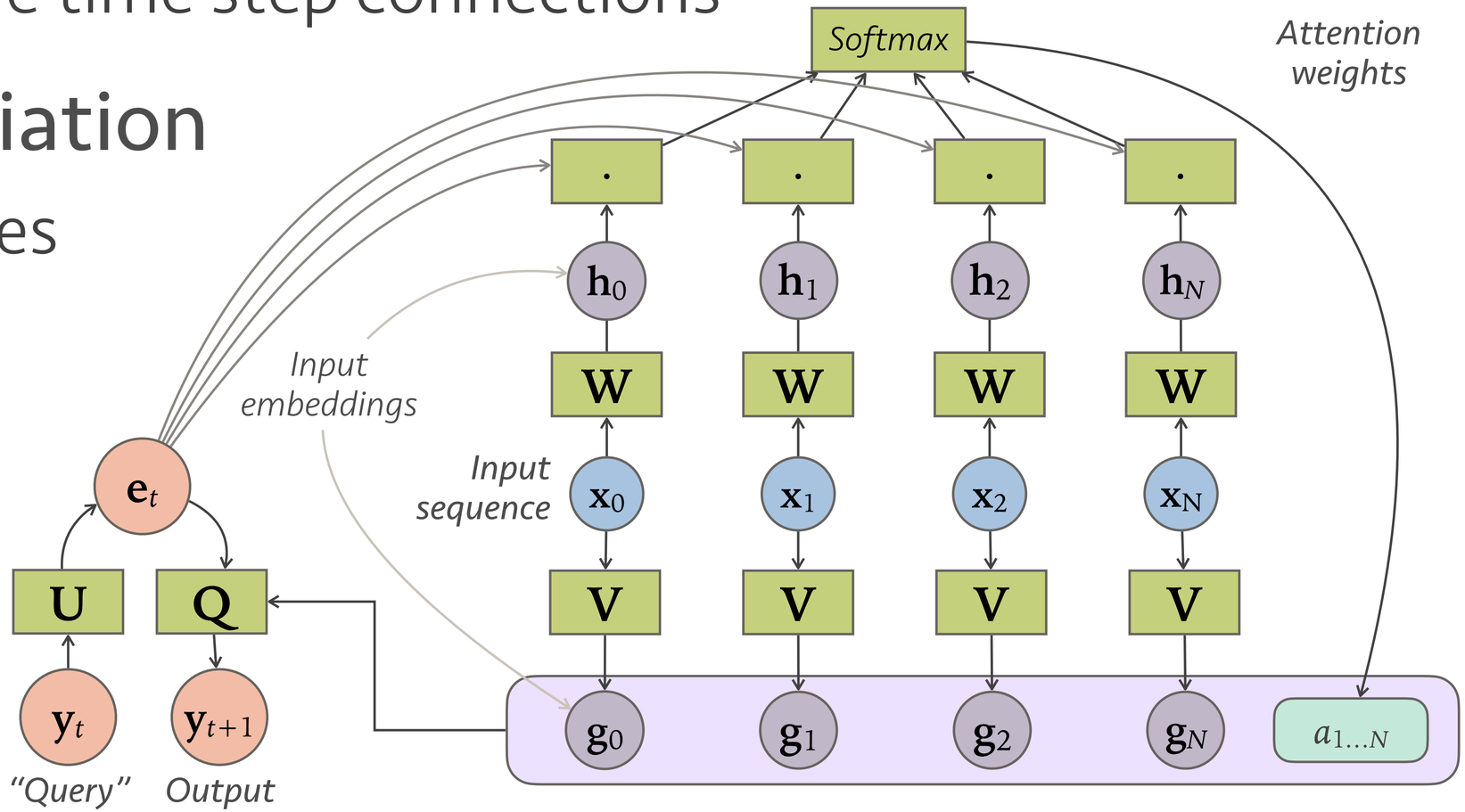
Attention

- Attention tells us where to focus for each output
  - Provides a convenient map highlighting input/output relationships



Getting rid of the RNN too

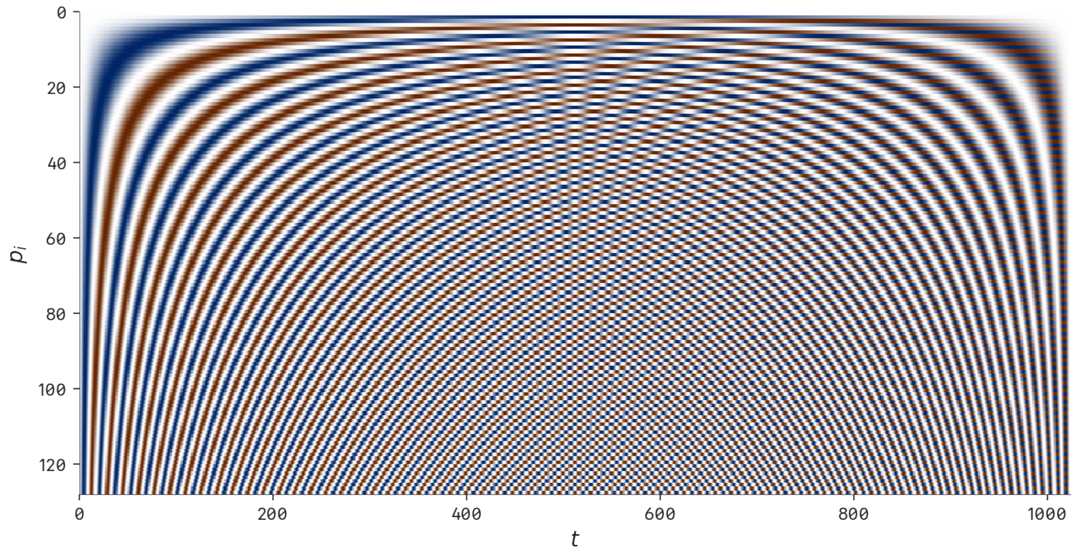
- *End-to-end memory* networks
  - Severing consecutive time step connections
- *Self-attention* variation
  - Use all input samples as queries
  - Can serve as a layer



What if time really matters?

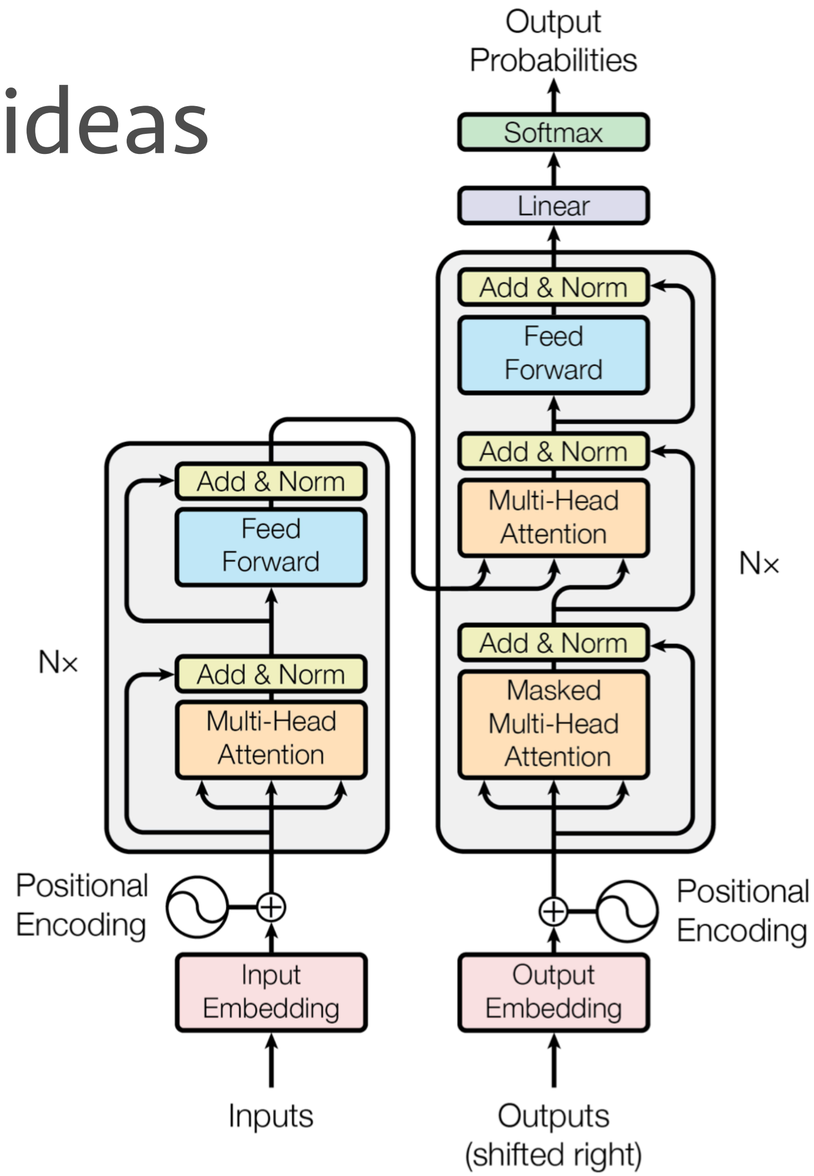
- *Positional encodings*
  - Use a code to express time of each element
- A bad idea: use time as a feature: $\mathbf{x}_t \rightarrow \{t, \mathbf{x}\}$
  - High-dimensional $\mathbf{x}$ will be much more influential than scalar t
- Better take: $\mathbf{x}_t \rightarrow \{\mathbf{p}_t, \mathbf{x}\}$

$$\mathbf{p}_t = \begin{bmatrix} sin(\omega_1 t + \varphi_1) \\ \vdots \\ sin(\omega_N t + \varphi_N) \end{bmatrix}$$



The Transformer

- Putting together all of the previous ideas
  - Self-attention, positional encodings, …
- Multi-head attention
  - Concatenating multiple attentions
- Template behind popular models
  - GPT-3, BERT, etc.



Generalizations

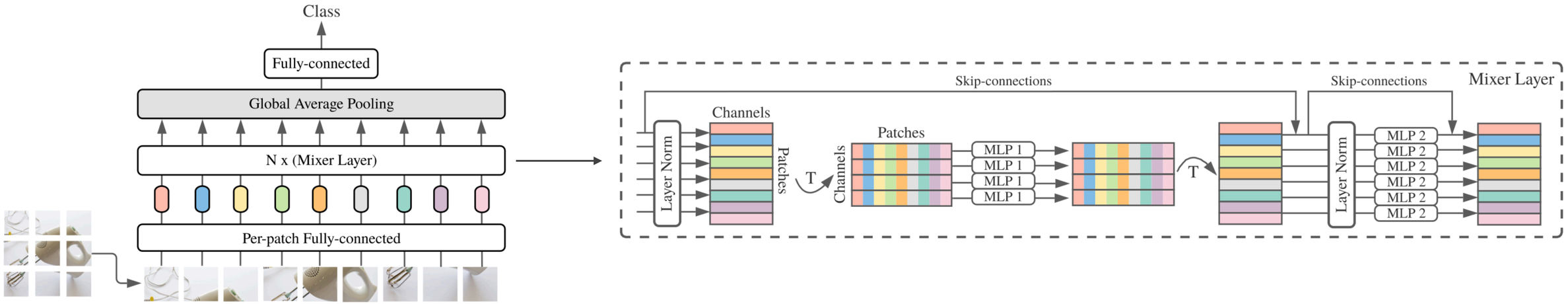
- Sparse attention models
  - Sparse-tranformers, longformers, reformers, graph attention, …
  - Use a sparse representation, not encoding all similarities
- Performers
  - Linearly-scaling attention
- Resulting models can work on signal lengths!
  - E.g. OpenAI's Jukebox synthesizing novel music

Ella Fitzgerald & Celine Dion

Justin Bieber

Back to multilayer perceptrons?

- One can forgo CNNs/RNNs/Attention and use MLPs!
  - Apply simple MLPs on neighborhood patches of input
    - Uses local structure
  - Then apply simple MLPs on combinations of these and repeat
    - Looks at global structure



Recap

- Incorporating time in deep learning models
  - Convolution-based
  - Recurrent models
- Sequence-to-sequence models
- Attention and Transformers

Reading material

- Most basic material is here:
  - https://www.deeplearningbook.org
- More advanced models:
  - WaveNet: https://arxiv.org/pdf/1609.03499.pdf
  - Transformer: https://arxiv.org/abs/1706.03762
  - MLP-Mixer: https://arxiv.org/pdf/2105.01601.pdf

Next lecture

- Generative Models!
  - VAEs, GANs, Flows, Diffusion, AR models, et al.