University of Illinois at Urbana-Champaign

CS598PS – Machine Learning for Signal Processing

Deep Learning II: Time Series & Generative Models

9 November 2020

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

- Generative models
  - Making signals out of noise

Getting more into signals

- As we've seen before, we care about time!
  - That's what makes a signal
- What we have with deep learning so far 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

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

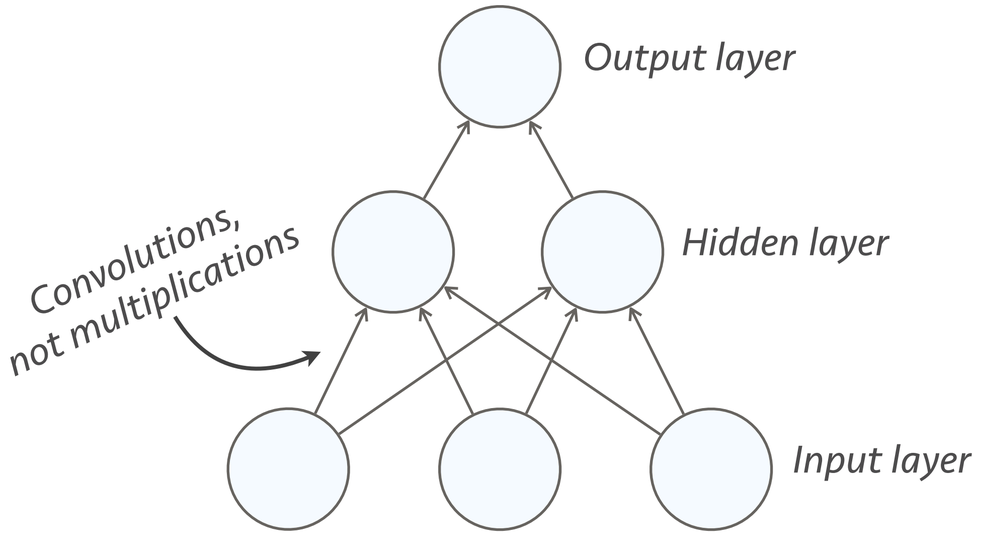
Taking it a step further

- 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

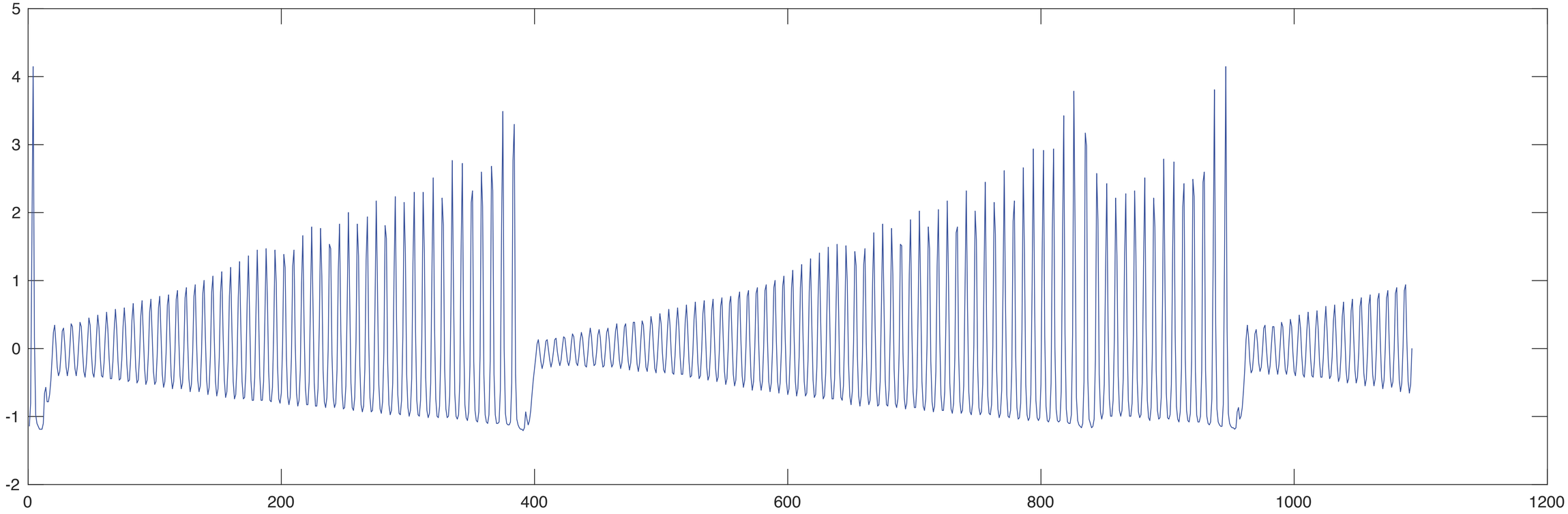


- Convolution is linear
  - The usual backprop approach still works
    - Just pretend the convolutions are matrix multiplies

- Good fit for temporal prediction tasks!

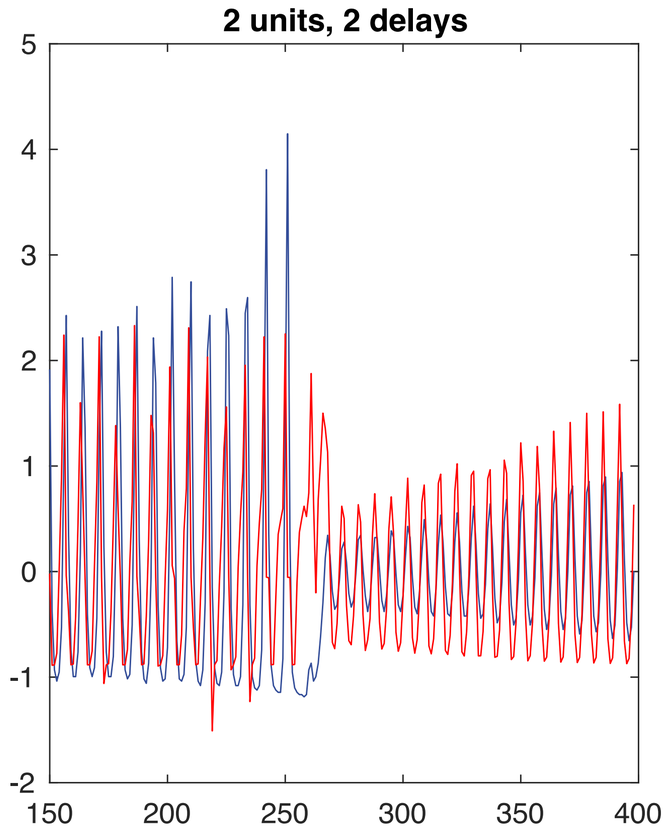
TDNNs in action

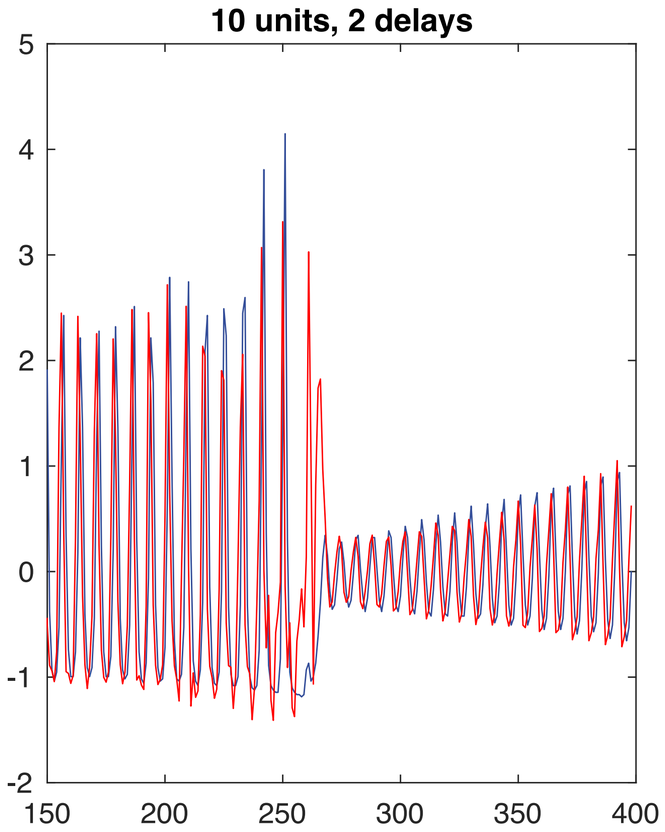
- Santa Fe chaotic laser data
  - Predict sequence from past samples, deal with chaotic behavior

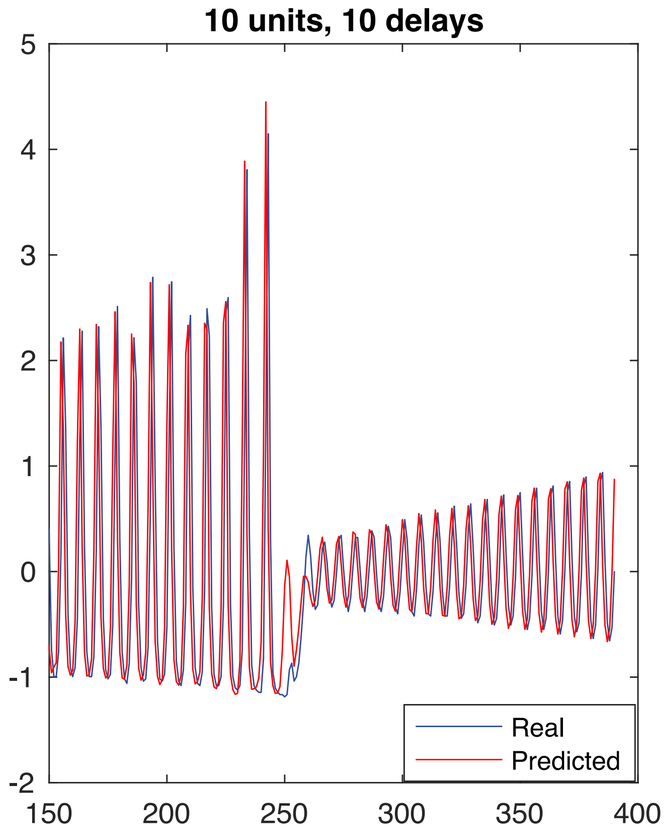


Prediction results

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







Modern variation

- Generalized convolution form in modern neural nets:

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

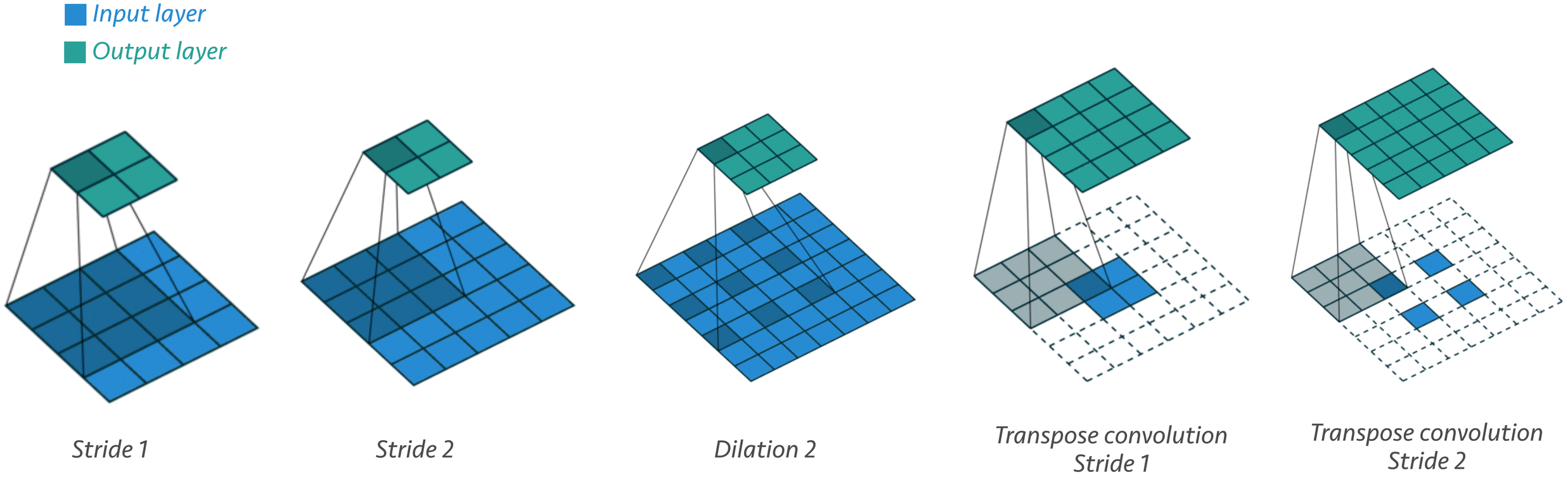
Use multiple filters ("channels")

Yeah, it's actually a correlation

- Additional options
  - Stride s: skip s samples in every convolution calculation in the input
  - Dilation d: Subsample input 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 ...

Different types of convolutions

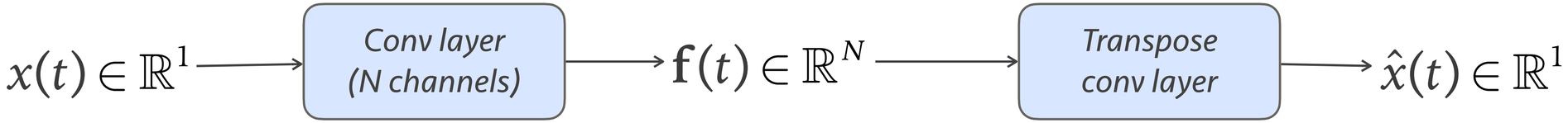
- Various options, generalizable to arbitrary dimensions



https://github.com/vdumoulin/conv_arithmetic

Designing an autoencoder for sound

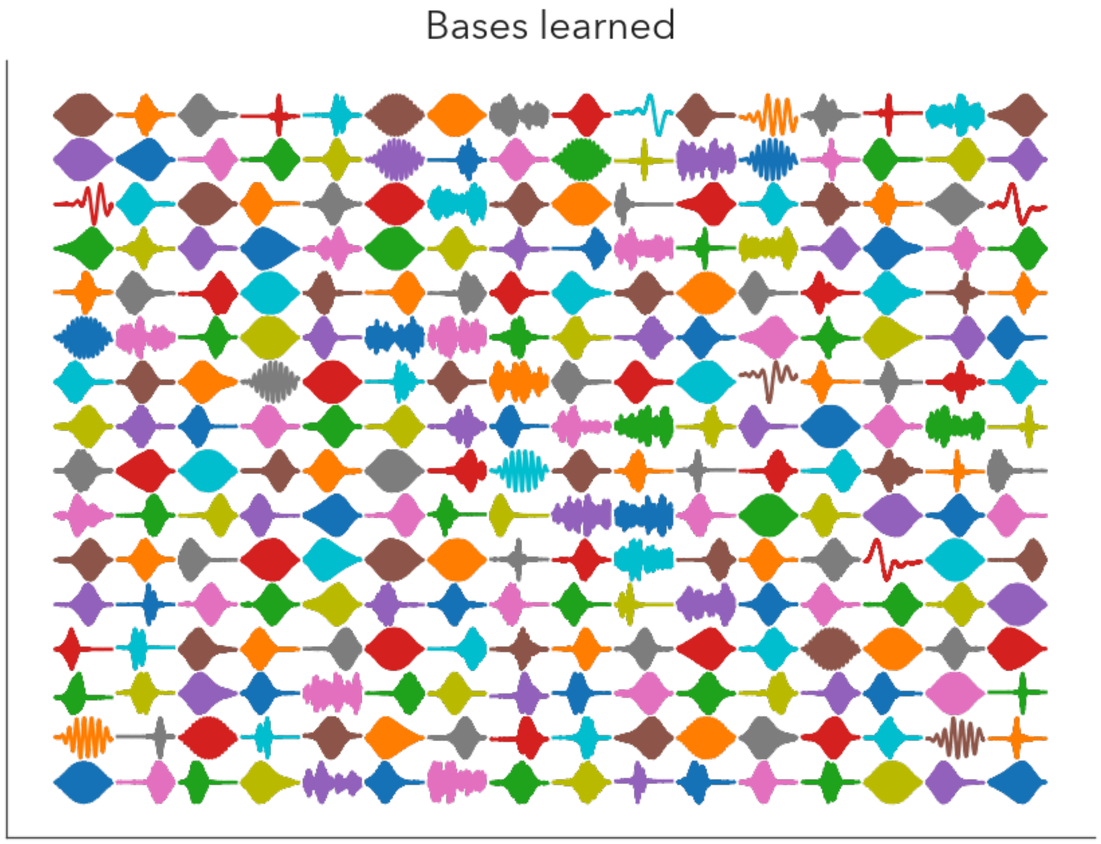
- We can now learn more complex models
- E.g. a convolutional autoencoder
  - Encode a time series using convolutions
  - Resynthesize input using transpose convolution

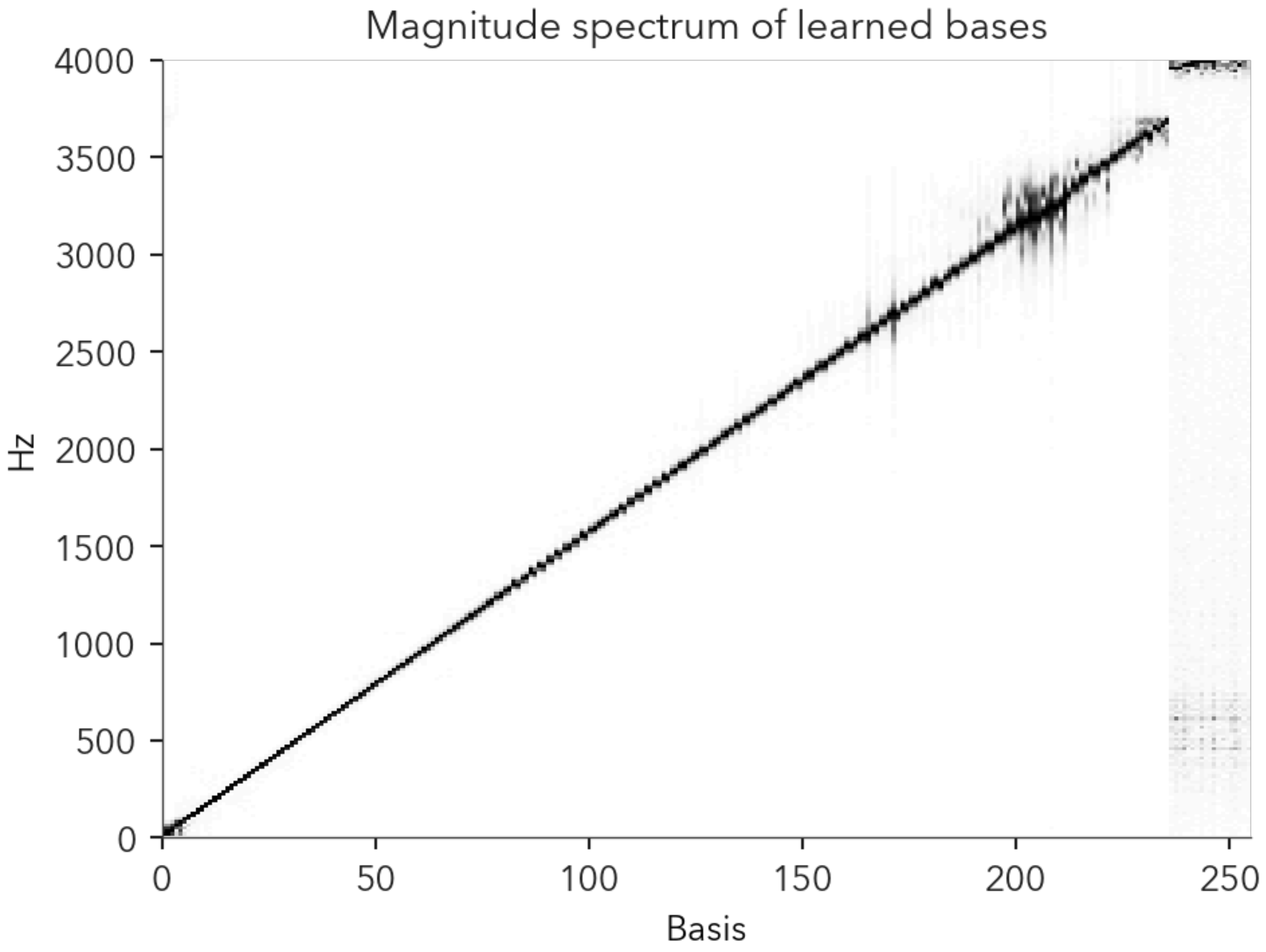


  - If the conv layers had Fourier bases we would be doing an STFT
    - Filter length = DFT size, stride = hop size
    - Many possible extensions; nonlinear, multilayer, …

Not a very surprising outcome

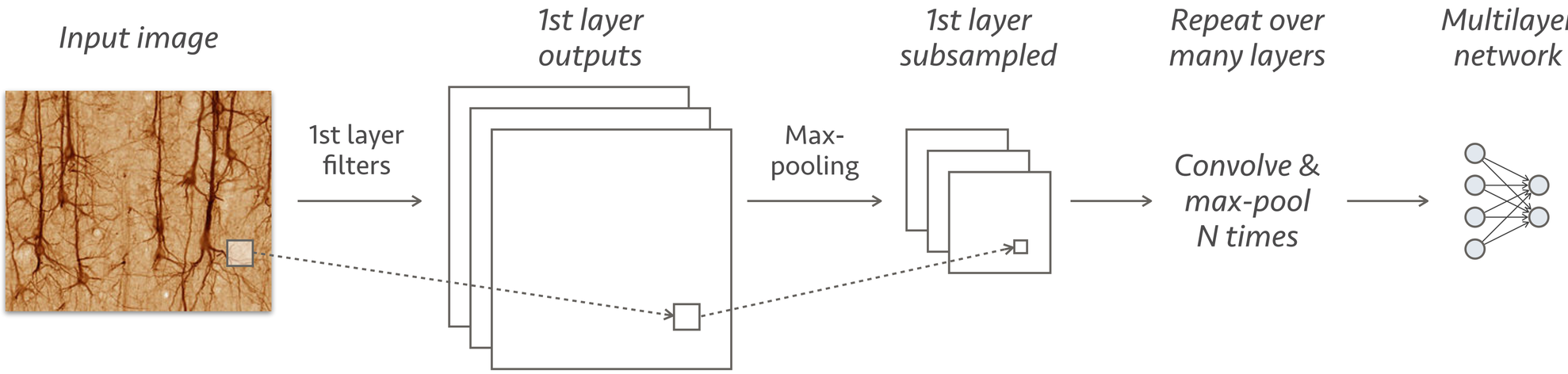
- We get sinusoidal-like bases again!





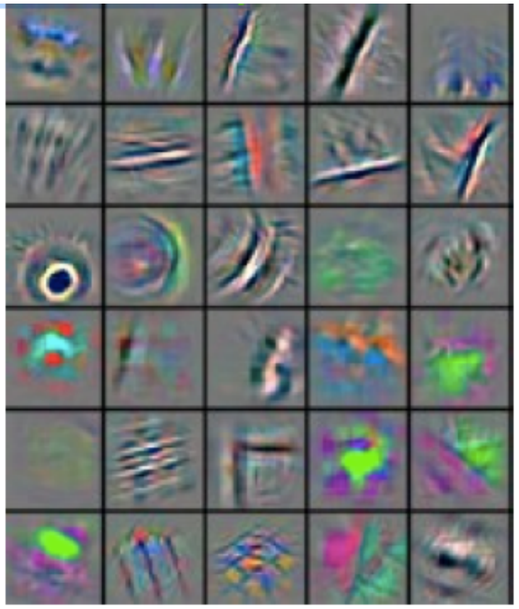
Convolutional Networks

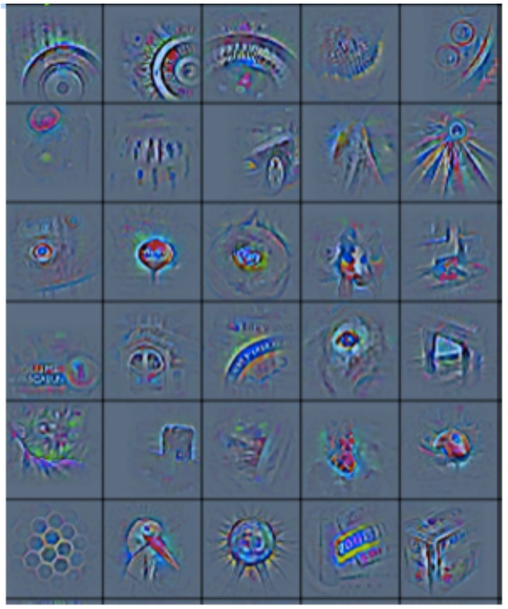
- Very popular in computer vision
  - Use small local convolutions to represent input
- Also employ *max-pooling*
  - For each neighborhood of filter outputs, keep only the max value

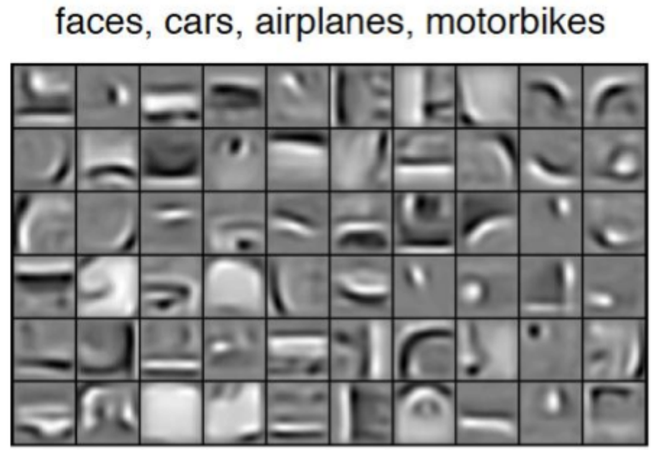


Convolutional networks for vision

- 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)

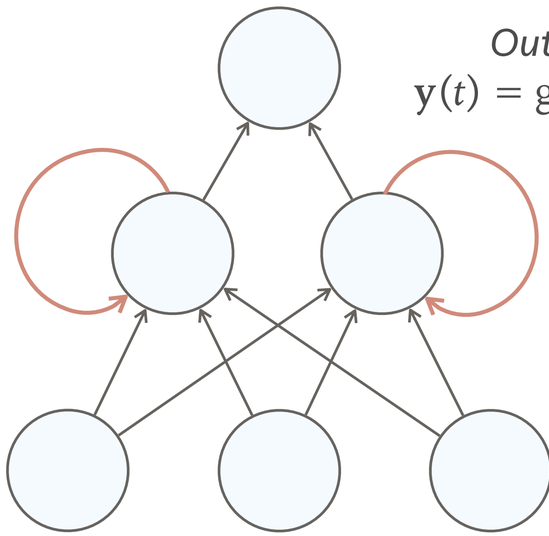
$$y(t) = a \cdot x(t) + b \cdot y(t-1)$$

  - 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)

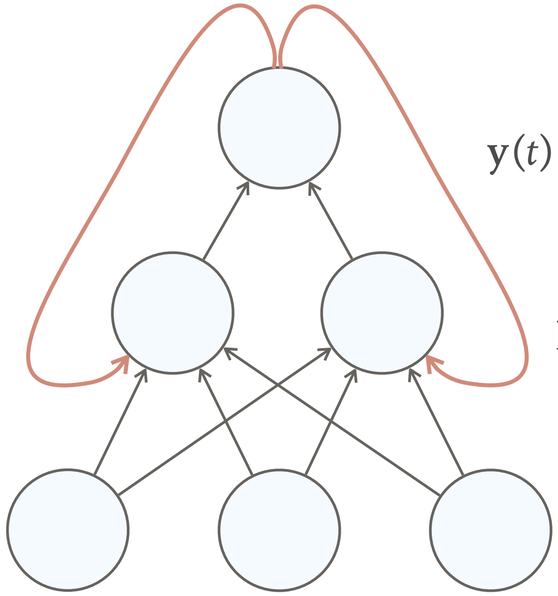


Output layer:
$\mathbf{y}(t) = \mathrm{g}(\mathbf{V}\,\mathbf{h}(t) + \mathbf{c})$

Hidden layer:
$\mathbf{h}(t) = \mathrm{g}(\mathbf{W}\,\mathbf{x}(t) + \mathbf{U}\,\mathbf{h}(t-1) + \mathbf{b})$

Input: $\mathbf{x}(t)$

Jordan network
(feed output layer back to hidden layer)



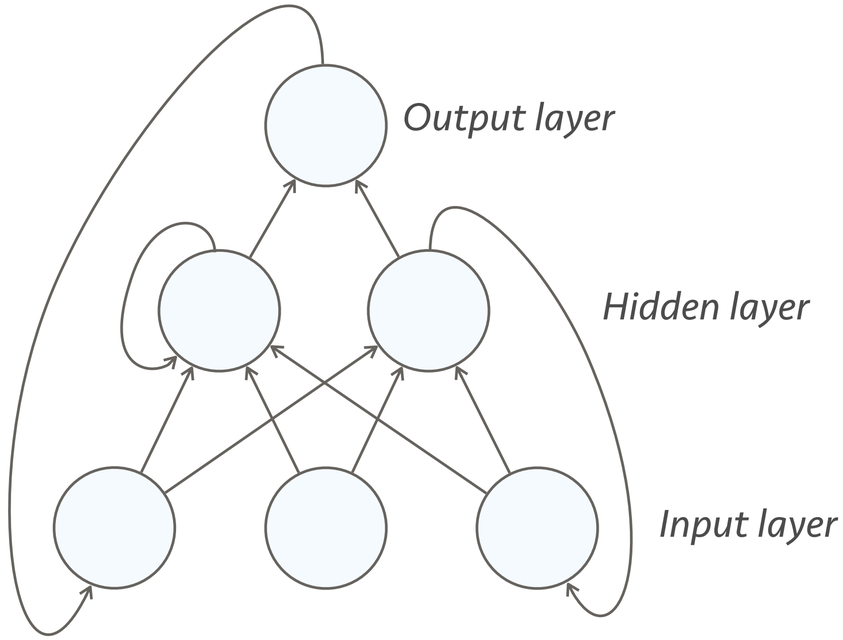
Output layer:
$\mathbf{y}(t) = \mathrm{g}(\mathbf{V}\,\mathbf{h}(t) + \mathbf{c})$

Hidden layer:
$\mathbf{h}(t) = \mathrm{g}(\mathbf{W}\,\mathbf{x}(t) + \mathbf{U}\,\mathbf{y}(t-1) + \mathbf{b})$

Input: $\mathbf{x}(t)$

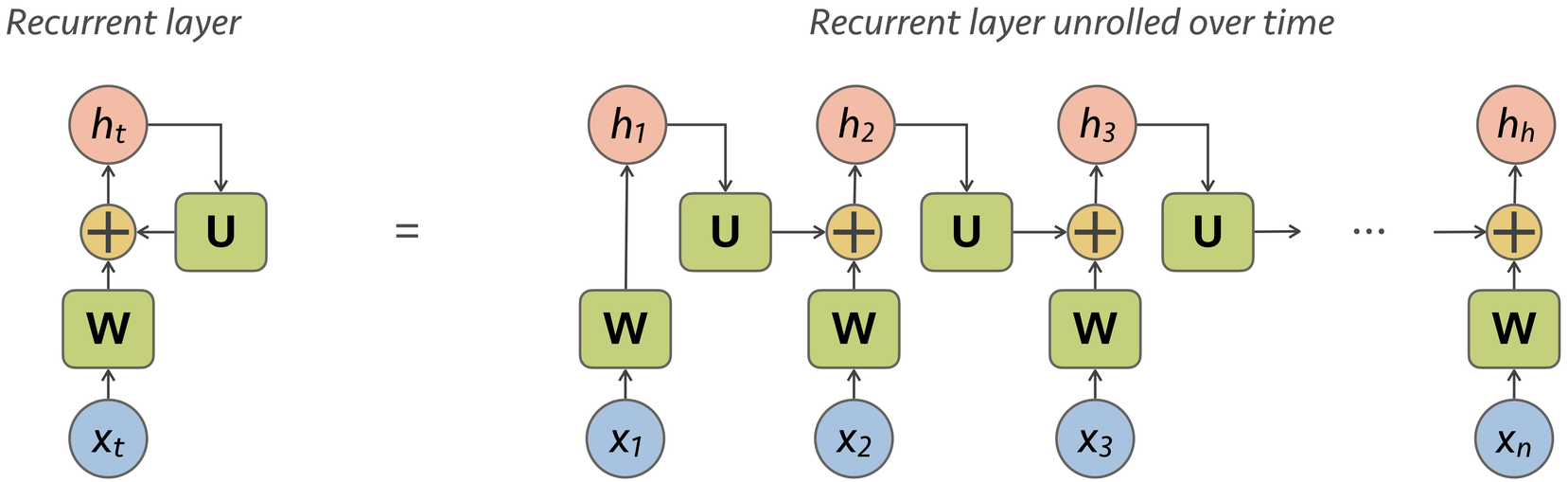
Recurrent Neural Networks (RNN)

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



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 "carry" necessary data over all time steps
    - In theory!
  - In practice, we use a large hidden state to create memory

Simple example

- Learn: $y(t) = x(t) + x(t–3)$

Memory in RNNs

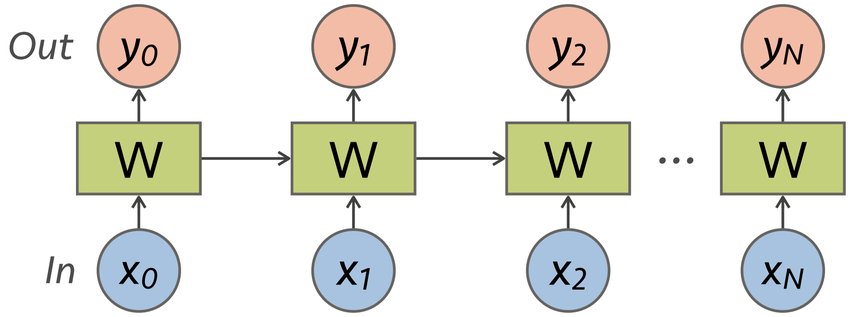
- RNNs can construct a memory using the state vector
  - E.g., trivial solution to the previous problem:

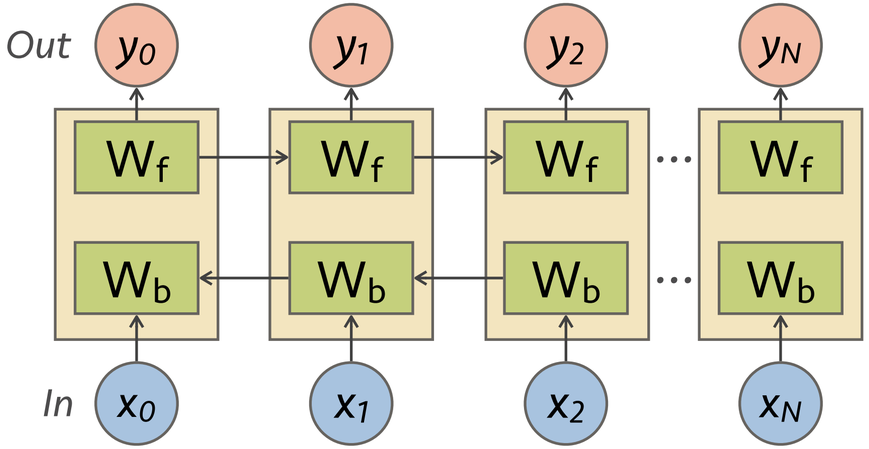
$$\mathbf{h}_t = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot \mathbf{x}_t + \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{h}_{t-1} \Rightarrow \mathbf{h}_t = \begin{bmatrix} \mathbf{x}_{t-3} \\ \mathbf{x}_{t-2} \\ \mathbf{x}_{t-1} \\ \mathbf{x}_t \end{bmatrix}$$

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

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 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 difficult

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

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

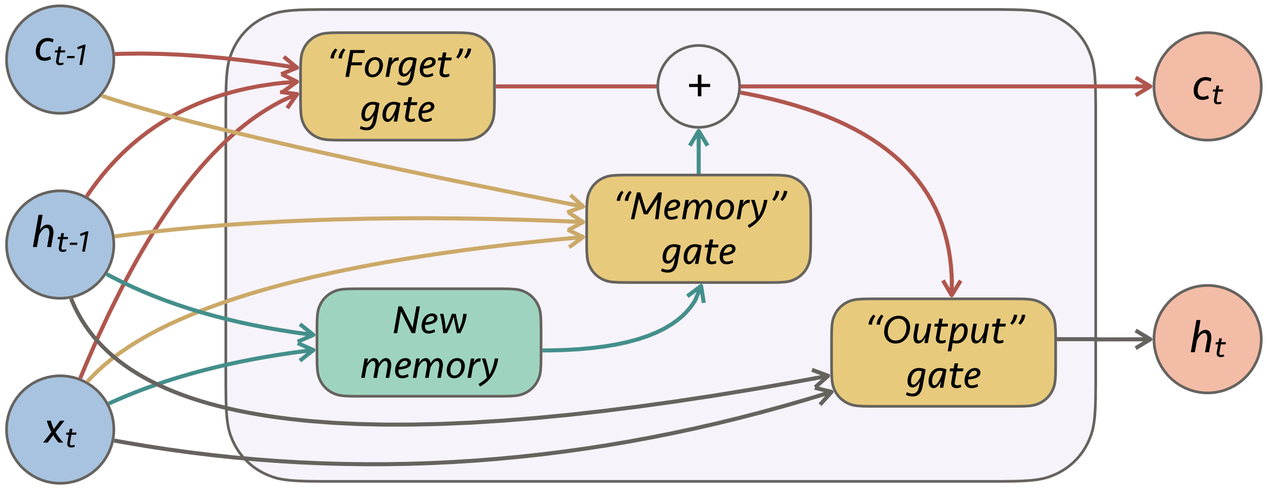
Skipping bias terms for readability

  - 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?

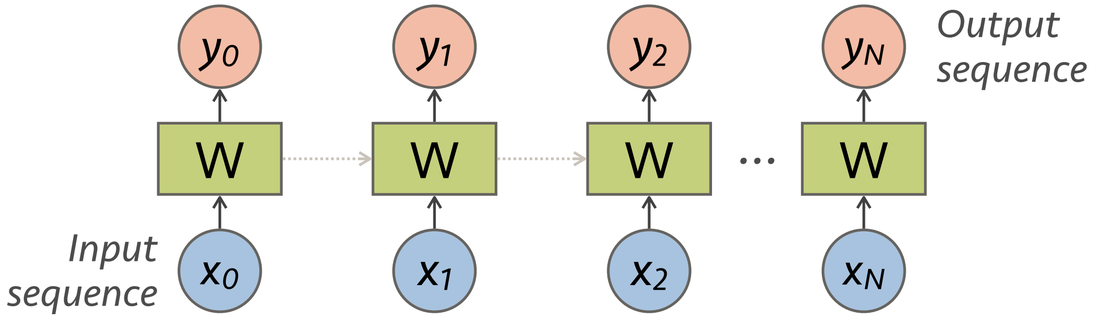
- Ongoing debate
  - Results are roughly the same for most tasks (although RNNs get there first)
- RNN pros/cons
  - Pros: Can deal with arbitrary length inputs, much more compact
  - Cons: Slow to compute, recurrence is not parallelizable
- CNN pros/cons
  - Pros: Efficient, explainable weights, easy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

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

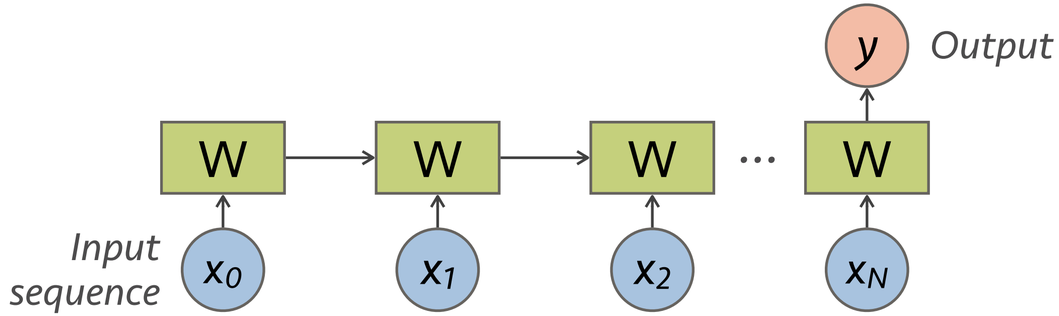
- Synchronous sequence to sequence models
  - Does not necessitate an RNN or CNN



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

Sequence to vector models

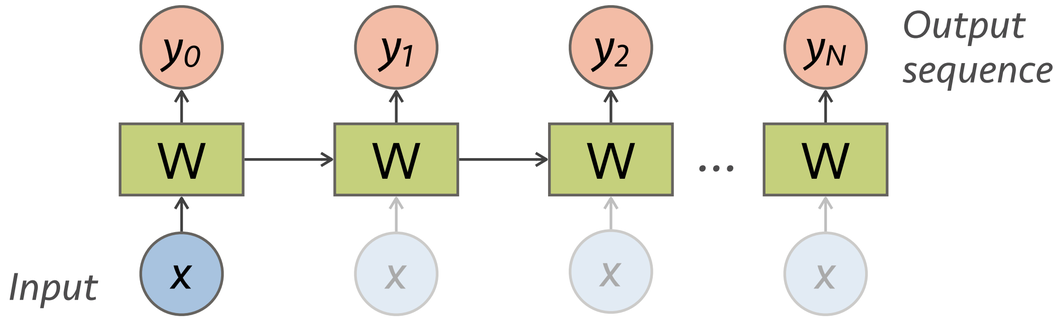
- Map a sequence to a single summarizing vector
  - Necessitates an RNN for arbitrary length inputs



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

Vector to sequence models

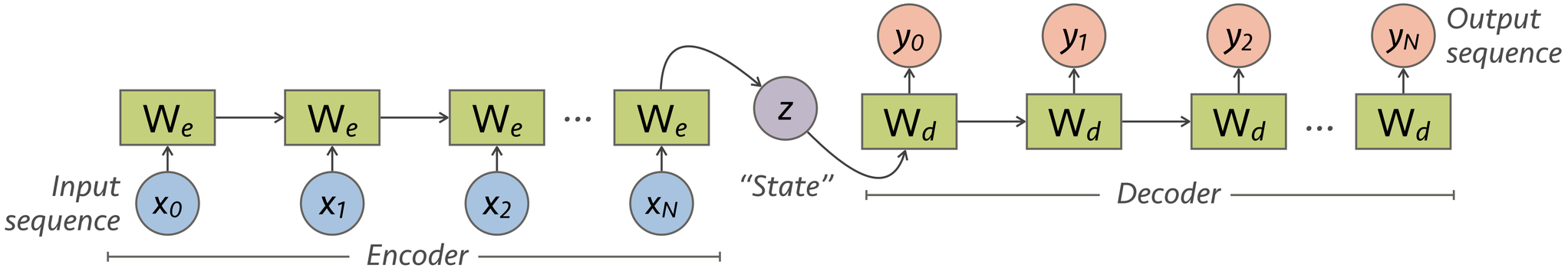
- Generate a sequence from a single vector
  - Needs an RNN architecture
  - RNN has to signal end-of-sequence with a pre-specified output



- Good for generating sequences from descriptions
  - E.g. generate images from a state vector, music from emotion, ...

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



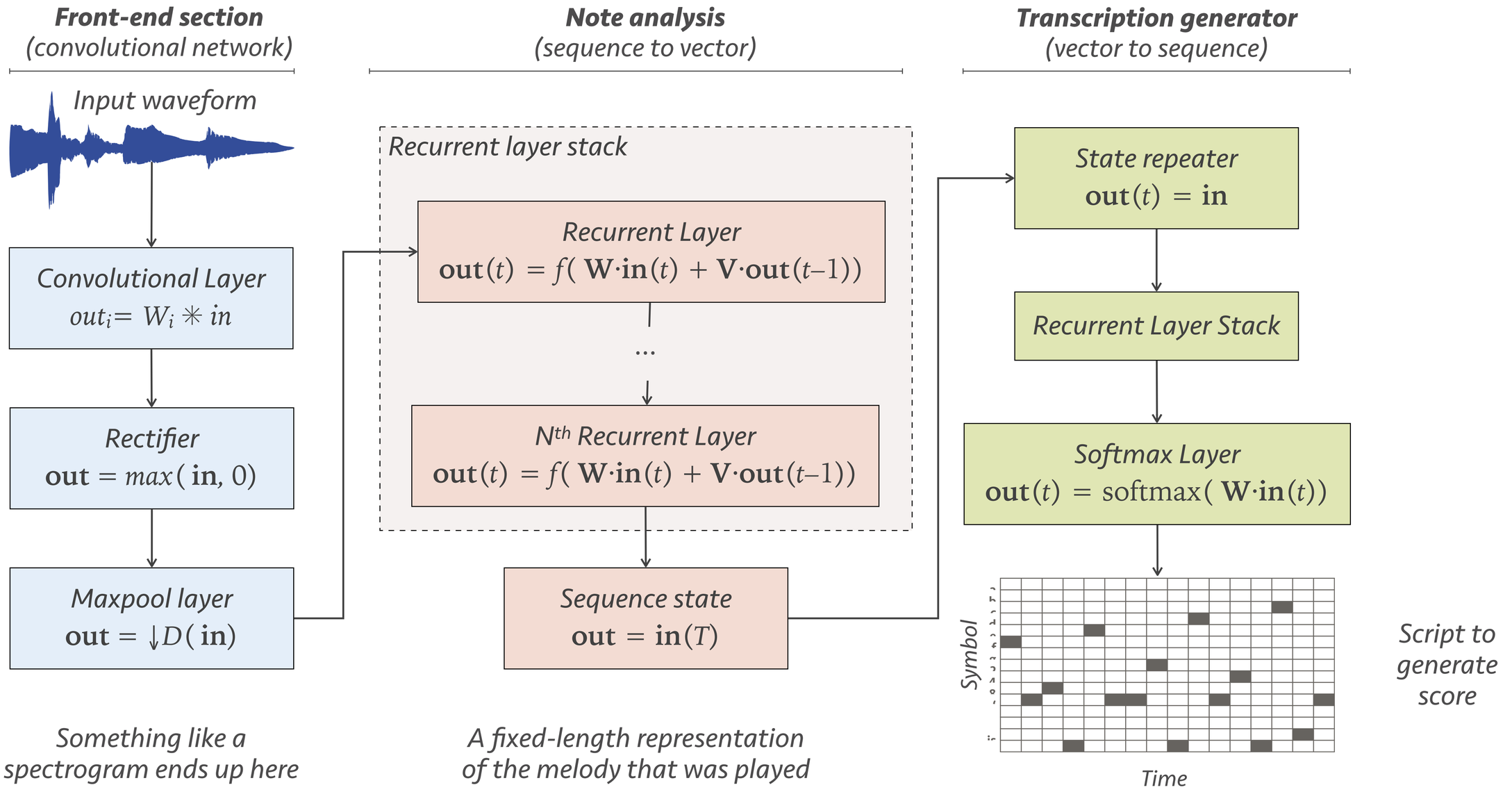
- Use to translate sequences
  - E.g. video input (few frames) to an audio soundtrack (lots of samples)

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

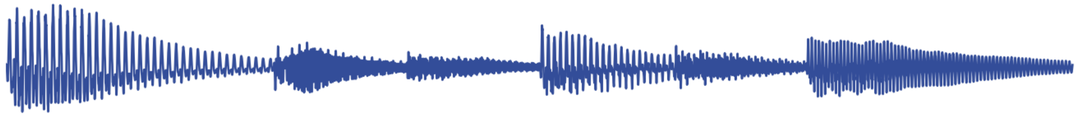
Overall model



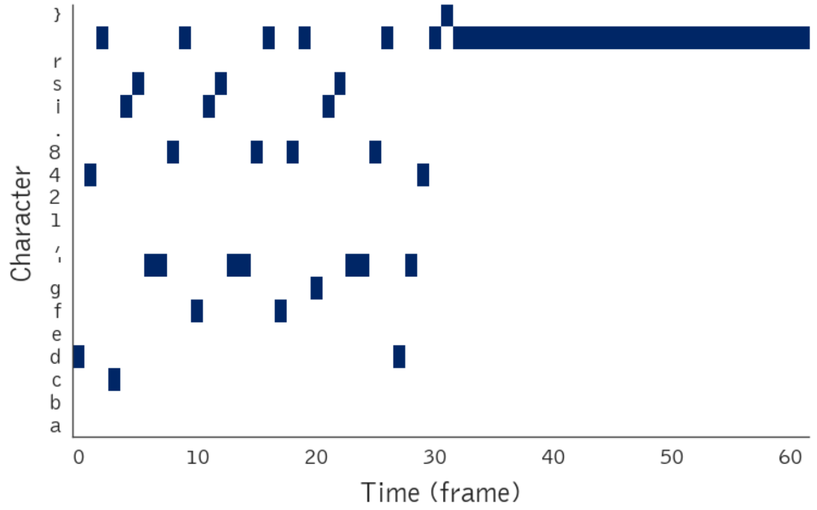
Training such a network

- 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

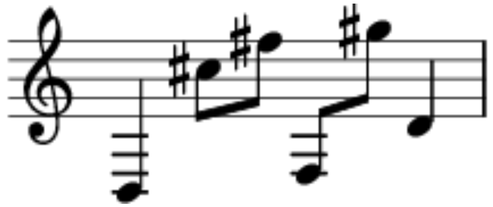
Input data



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



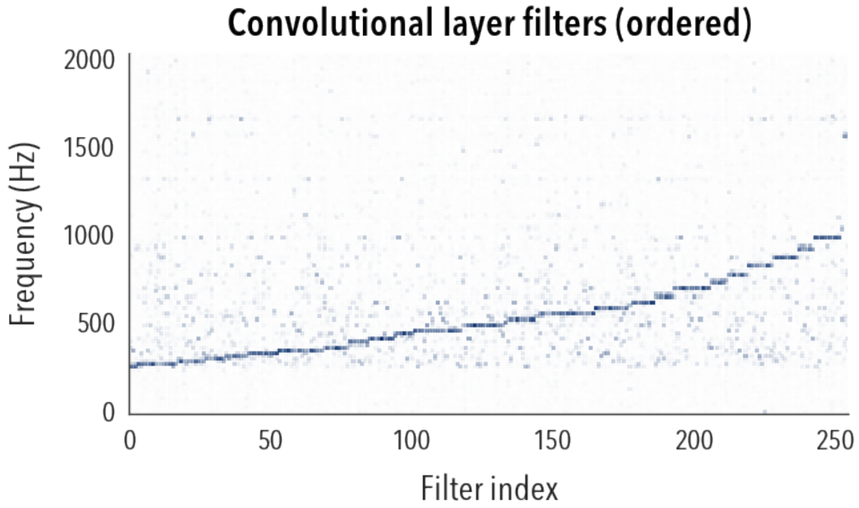
Implied score

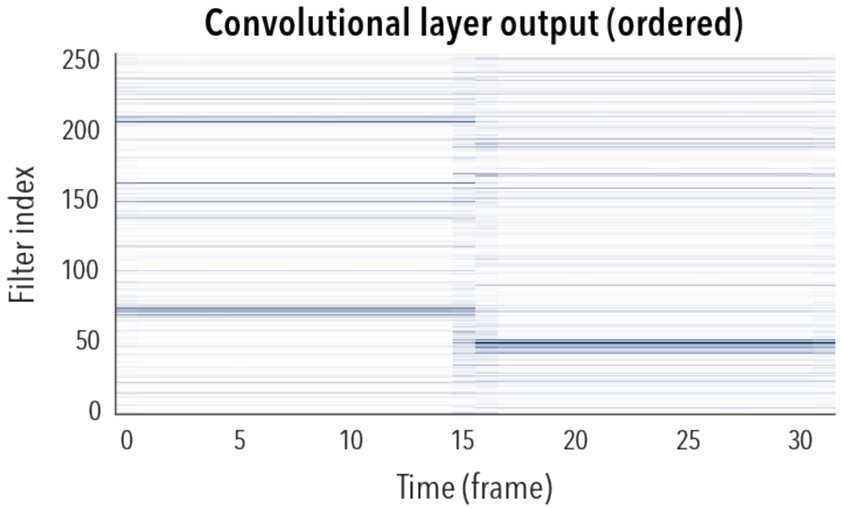


Examining the learned network

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

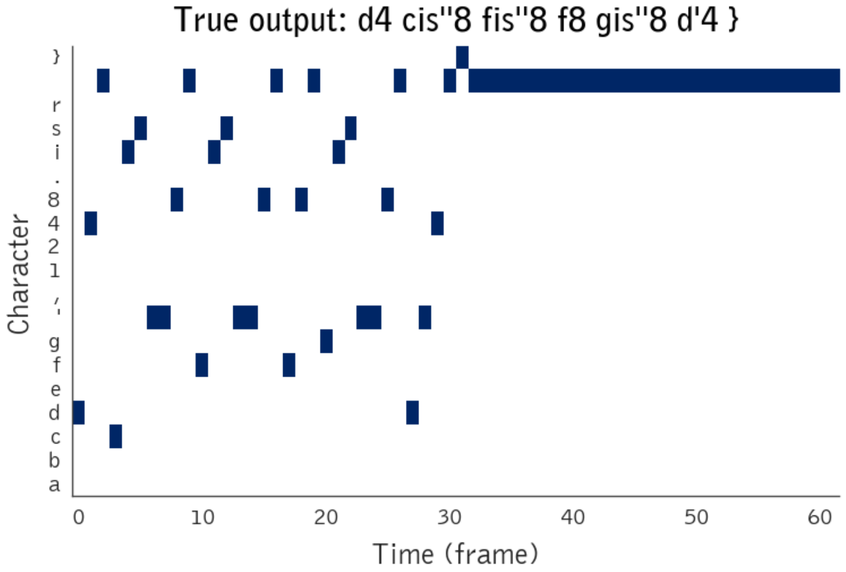
Example conv output for two notes, looks like a spectrogram, but with no frequency order

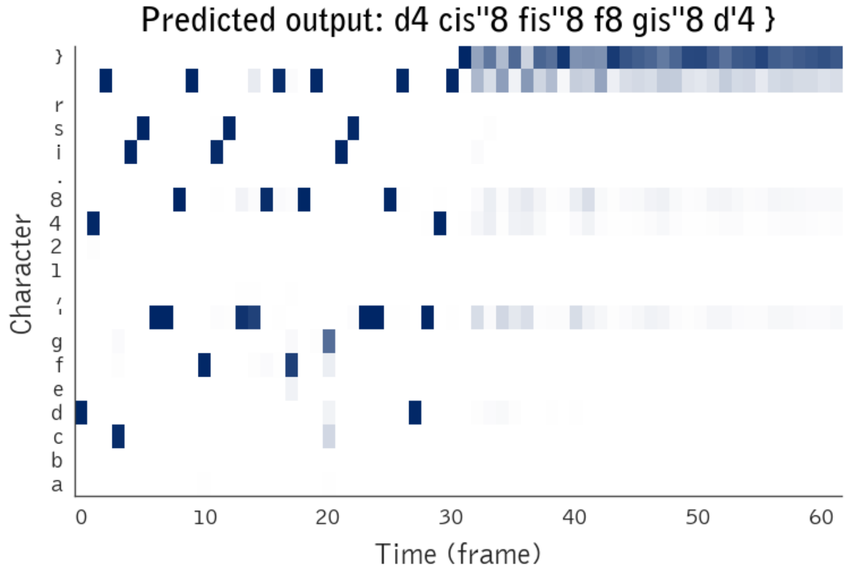




Example output target for a script describing the musical score

Example produced output directly from waveform





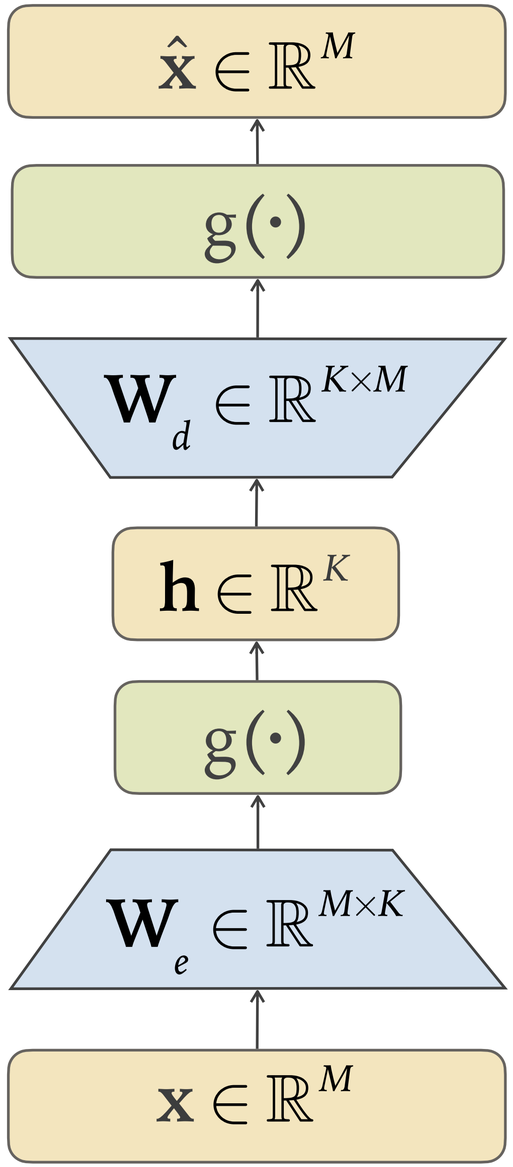
Generative Models

- Generative models are good for synthesizing data
  - E.g., speech synthesis, image generation, videos, etc.

- Two dominant models
  - Variational Auto-Encoders (VAEs)
  - Generative Adversarial Networks (GANs)

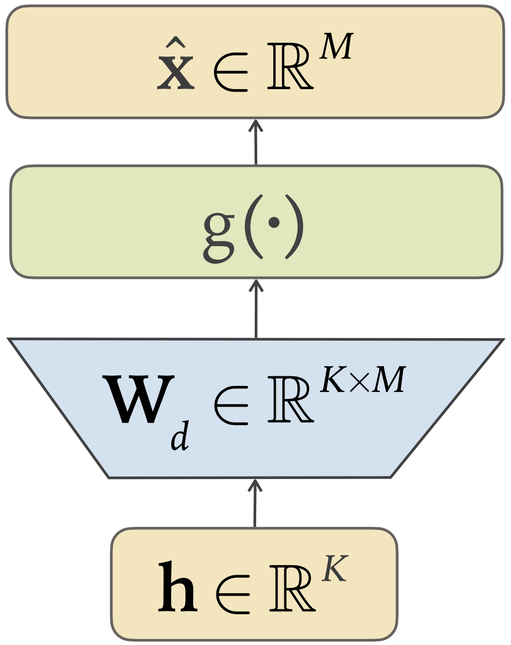
Starting with an autoencoder

- Consider an autoencoder network
  - High-dimensional input $\mathbf{x}$
  - Transformed to a low-dimensional $\mathbf{h}$
    - Via an *encoder* $\mathrm{g}(\mathbf{W}_e \cdot \mathbf{x})$
  - And back to high-dim approximation of input
    - Via a *decoder* $\mathrm{g}(\mathbf{W}_d \cdot \mathbf{h})$
- $\mathbf{h}$ is a low-dim representation of $\mathbf{x}$
  - Like PCA, but not orthogonal



Hallucinating new data

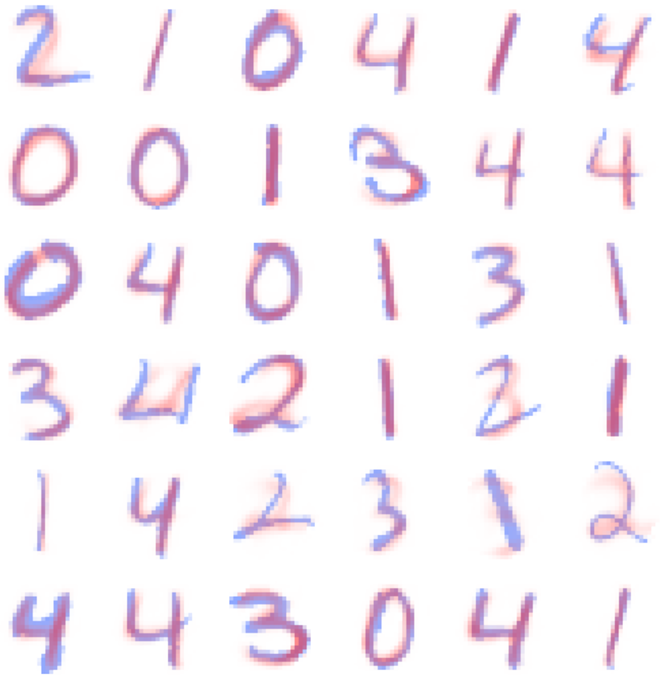
- Trained autoencoder has two parts
  - Encoder: from high-d to low-d
  - Decoder: from low-d to high-d
- The decoder on its own can be used to generate new data
  - Generate random values for $\mathbf{h}$
  - Pass them through decoder



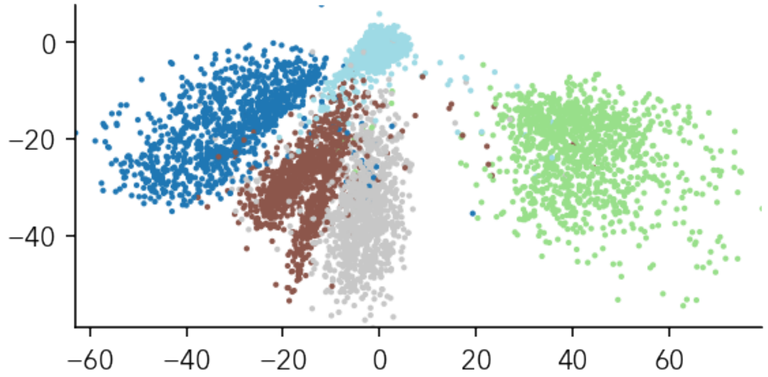
MNIST example

- Train on digits {0,1,2,3,4}, use 2d latent representation
  - Latent dimension is not easy to understand

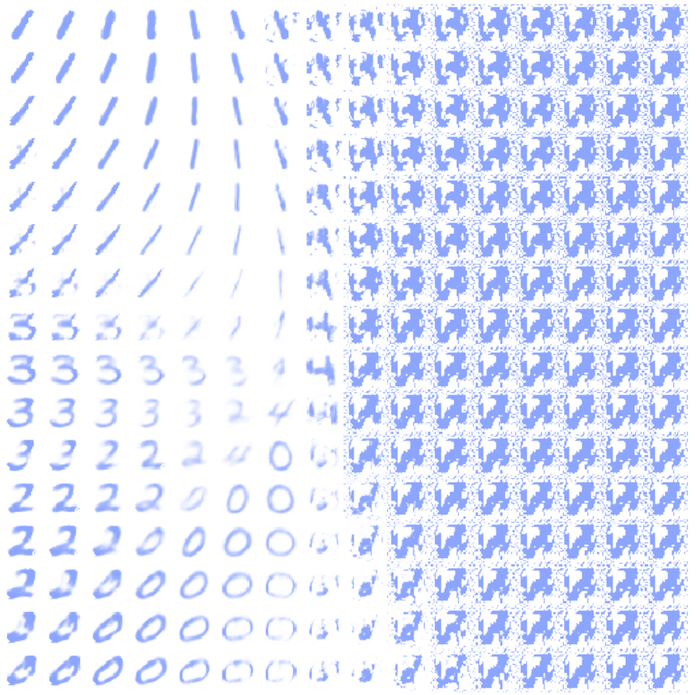
Input/approximations



Latent projection



Latent space

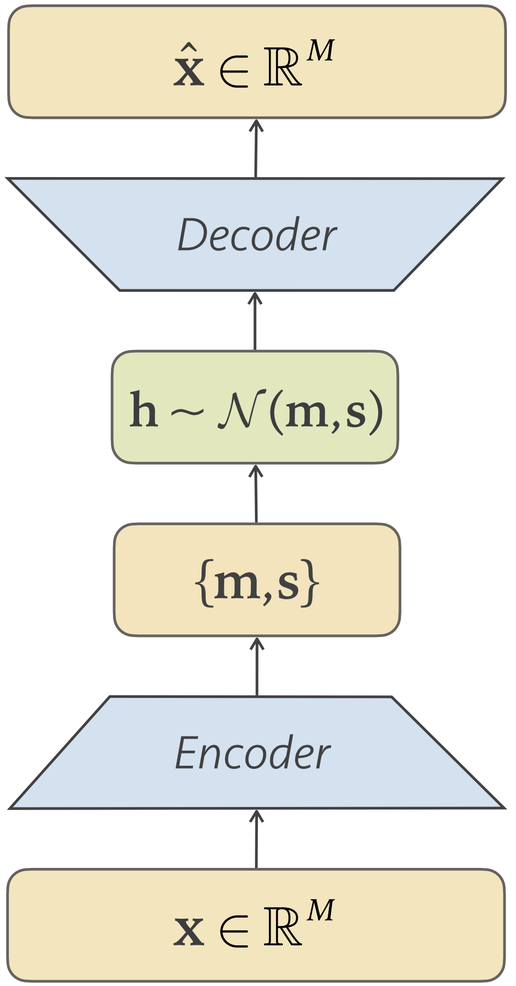


Some problems

- How can we sample plausible $\mathbf{h}$ values?
  - There are gaps which generate garbage data
- How is $\mathbf{h}$ distributed?
  - What range do I sample from?
- What if multiple $\mathbf{x}$'s collapse to a single $\mathbf{h}$?
  - Can I guarantee that the encoder is smooth?

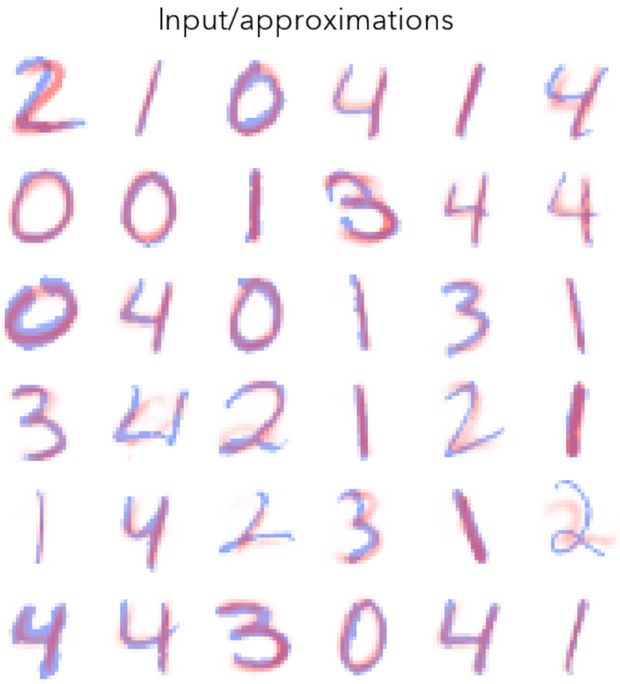
Variational Auto-Encoder (VAE)

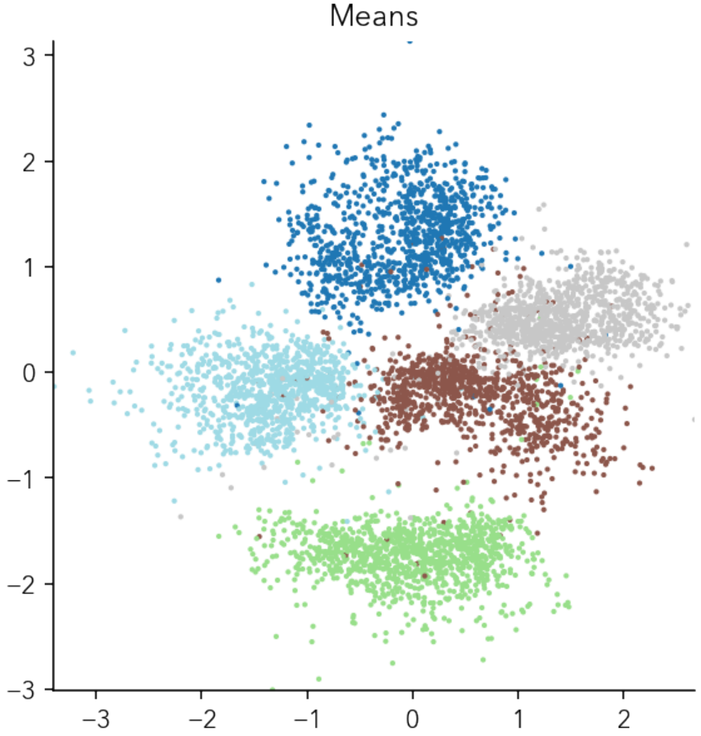
- Force $\mathbf{h}$ to be Gaussian distributed
  - Helps us to know its structure
  - Creates a more efficient encoding (more later)
- How to do this?
  - Encoder maps each input $\mathbf{x}$ to a mean and variance
  - We sample a Gaussian with them
  - We present that sample to the decoder
- We additionally minimize: $\mathrm{KL}\Big(\mathcal{N}(\mathbf{m},\mathbf{s}) \,\|\, \mathcal{N}(\mathbf{0},\mathbf{I})\Big)$
  - i.e. tend to generate $\mathbf{m} = \mathbf{0}$, $\mathbf{s} = \mathbf{I}$

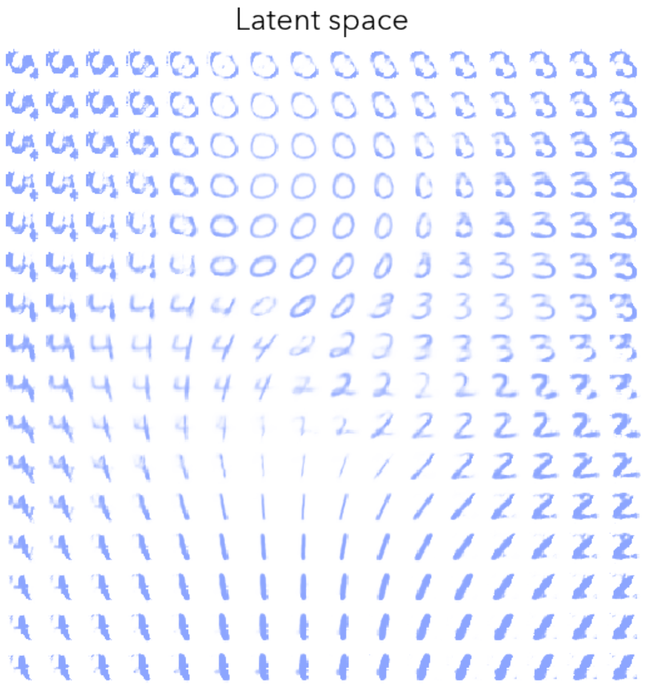


VAE MNIST example

- More convenient results
  - New latent representation is more compact and predictable
  - Sampled latent space generates continuous outputs







Key points to make

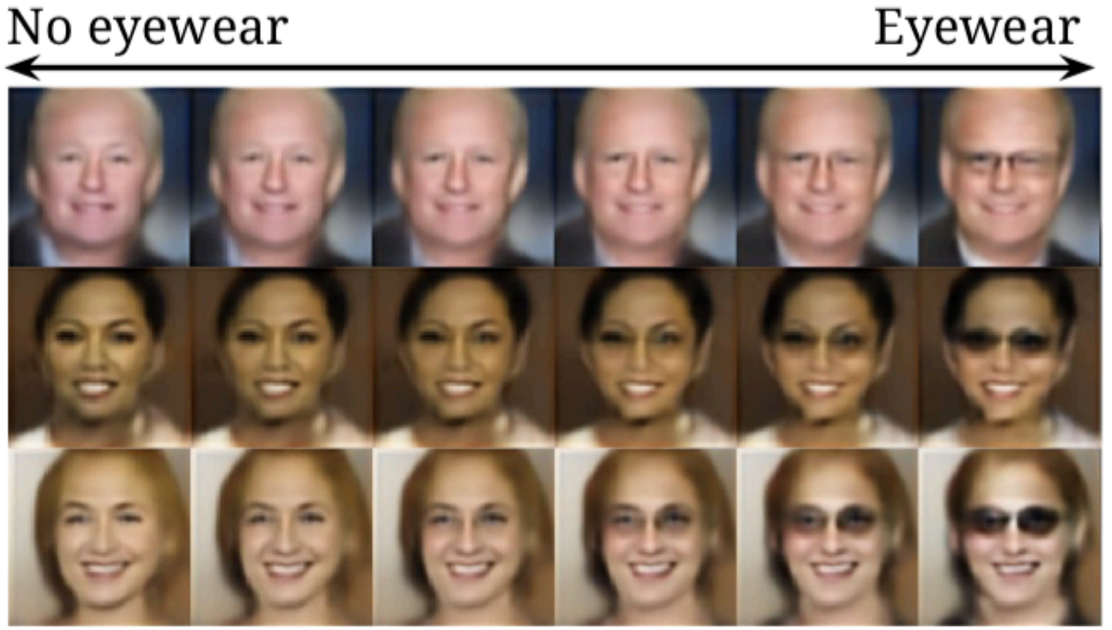
- We now have a distribution for $\mathbf{h}$
  - We know where to sample from
  - We additionally minimize gaps in the latent space

- The sampling process creates a more efficient encoder
  - Noise in the process forces the same data point to map to an entire region, and not a single point (we avoid overfitting)

Conditional VAEs

- If we also have labels for our data we can use them
  - Provide label as additional input to the decoder
  - This allows us to specify elements of data to generate

- E.g. Attribute2Image model
  - Learn faces with attributes
  - Sample and impose constraints



VAEs in action

- Popular for generating images
  - Generate new images from old
  - Generate new images with custom attributes (e.g. age, gender, …)
- Also used for making music
  - Interpolate between melodies
  - Generate new sequences from old
    - Can use RNNs in VAE model too!

An alternative approach

- Use a neural network to teach a neural network
  - Avoid selecting a specific cost function
  - Instead of comparing inputs/outputs, use a teacher network

- Why is this good?
  - We don’t have to use a simplistic measure (e.g. L2 reconstruction)
  - We can have a constantly evolving model

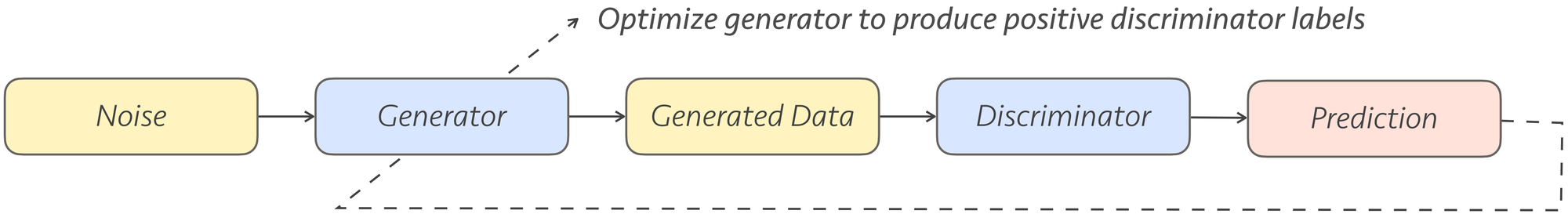
Generative Adversarial Networks (GANs)

- GANs are composed out of two networks
  - The *Generator*, creates new data points from noise input
  - The *Discriminator*, classifies data points as generated or real

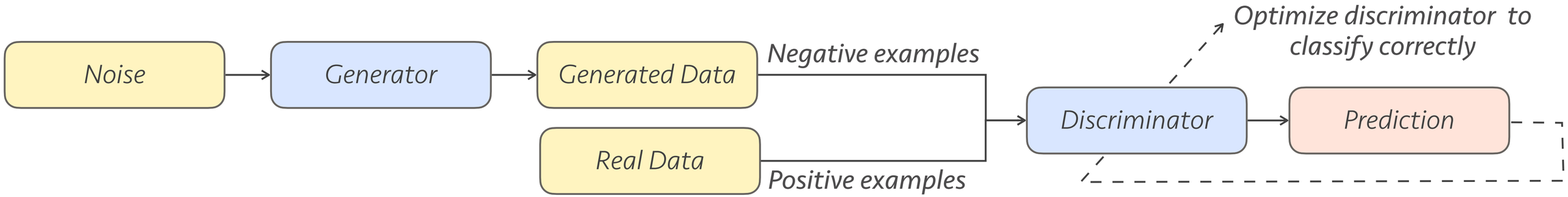
- Adversarial training process
  - Train the generator such that it fools the discriminator
  - Train the discriminator to detect generator's data
    - Through this process they both constantly improve each other

Training a GAN

- Train the generator to fool the discriminator
  - Improves the quality of generated data



- Train the discriminator to detect generated data
  - Makes the discriminator pickier, which will improve generator



How to set this up

- Original formulation, minimax setup

$$\max D = E_{\mathbf{x}} \log\Big[D(\mathbf{x})\Big] + E_{\mathbf{z}} \log\Big[1 - D\Big(G(\mathbf{z})\Big)\Big]$$

$$\min G = E_{\mathbf{z}} \log\Big[1 - D\Big(G(\mathbf{z})\Big)\Big]$$

- Wasserstein GANs ⟵ *Work better!*
  - Boils down to a simpler problem
    - Train the discriminator *N* times using regular classification loss
    - Train the generator *M* times using discriminator's loss
  - Get rid of the logs while we're at it

Some problems

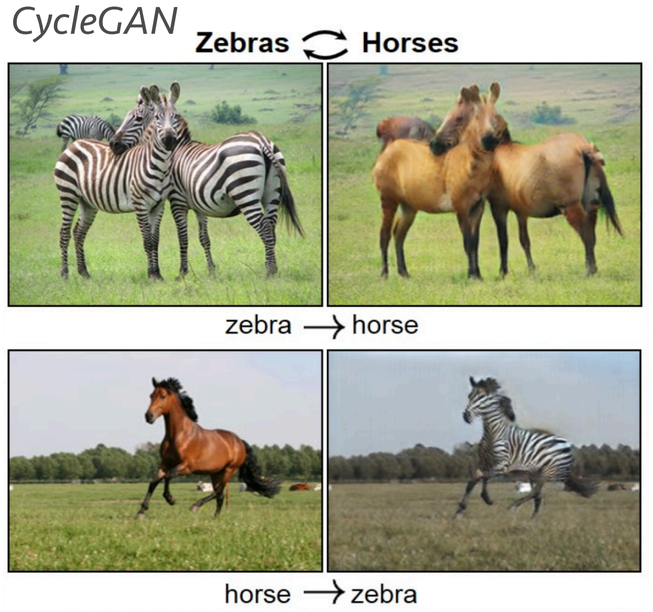
- Generated data are not guaranteed to be right
  - They will most likely fool the discriminator (is that enough?)
- Mode collapse
  - The generator constantly produces one passable data point
    - There is no guarantee of having variance in the generated data
  - Training can be tricky
- Cannot map to a describable latent space

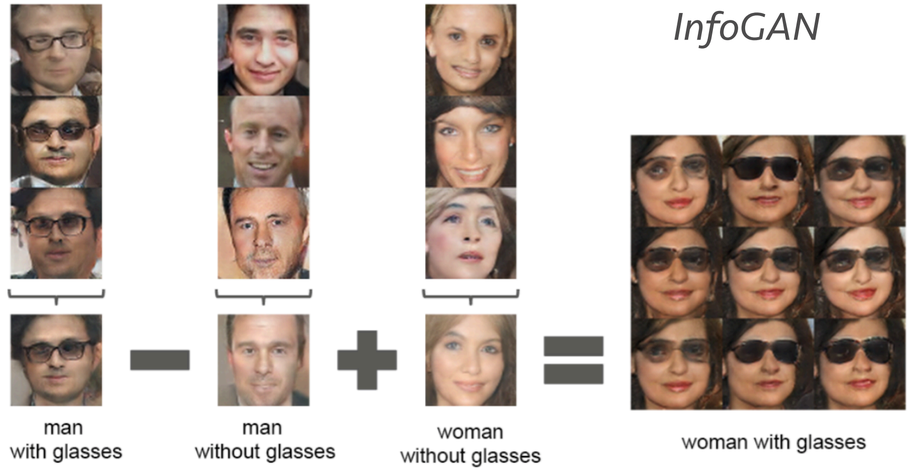
GANs in action

- Very popular in vision/graphics
  - E.g. fake celebrity generation
- Some success in other domains
  - Very useful when we cannot easily describe the data we want to make

Many elaborations

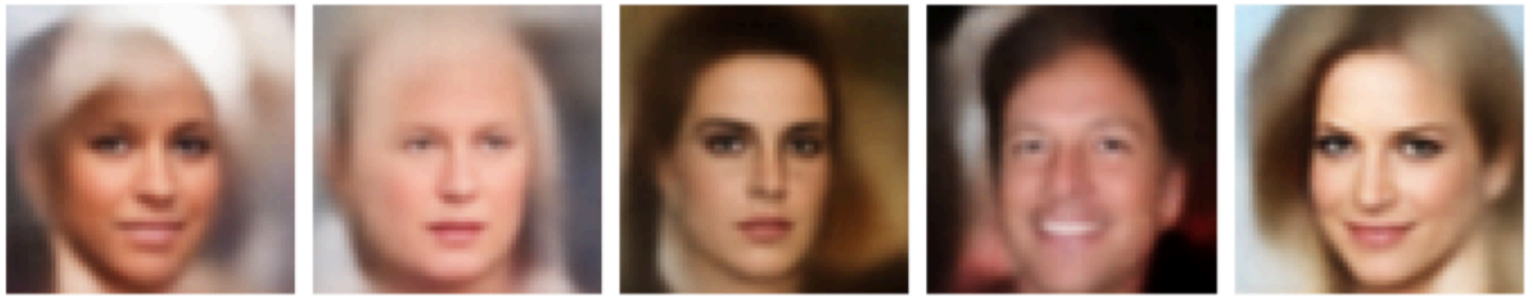
- CycleGAN
  - Make an autoencoder, provide additional output to a discriminator of other style
  - Then run it backwards
- InfoGAN
  - Force part of representation to use a latent code that adds information
- Many more …





VAEs or GANs?

- VAEs are known to produce “fuzzier” outputs



- GANs are known for sharper (but weirder) outputs

Putting it all together

- We can also make hybrids
- VAE_{dis} model
  - Train a GAN, use GAN's discriminator to train a VAE
- VAE/GAN
  - Combine the two, use a VAE to generate fake samples for GAN

Recap

- Incorporating time to deep learning models
  - CNNs, LSTMs, sequence models
- Generative models
  - Variational Auto-Encoders
  - Generative Adversarial Networks

Reading material

- Most basic material is here:
  - https://www.deeplearningbook.org

- Google relevant papers for all the rest
  - Sorry, there is no concentrated reference, things move fast!

Next lecture

- Graph Signal Processing and Graph Neural Nets
  - What do you do if your data is a graph
    - Or sampled irregularly