EECS 182/282A

Lecture 13- RNNs + Lect 14

Admin

→ What's up?

Source: Goodfellow — RNN chapter.

Recurrent Neural Networks.

History: Pre 2018. Two major successes in NNs.

① Images + Vision: CNNs.

② Languarge + Speech: RNNs.

How is language + speech different?

→ Time dependence

→ Also in control, time-series etc.

Notation: $t = 1, 2, \dots T$ time index

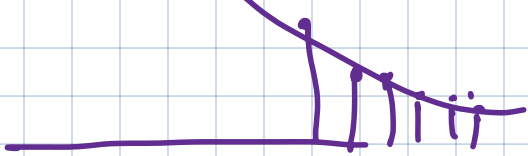
u_t: input at time t

y_t: output at time t

h_t: hidden state at time t

Causal sequence / Causal model





FIR Filter: Finite Impulse Response

$$y_T = \sum_{i=0}^{K-1} H_i \, u_{T-i}$$

← Convolution

H_i : the ith tap. . How many taps?

→ how much does the input from i steps ago factors in the input

→ How is a CNN like this?

IIR Filter: Infinite Impulse Response

$$h_t = f(h_{t-1}, u_t ; \theta)$$

updated state ↑ h_t — hidden state ↑ h_{t-1} — input ↑ u_t — parameters ↑ θ

State update equation.

$$y_t = g(h_t, u_t ; \theta)$$

output ↑ y_t — state ↑ h_t — input ↑ u_t — parameters ↑ θ.

Output equation

Example: LTI systems Linear Time Invariant.

$$h_t = A h_{t-1} + B u_t \qquad y_t = C \cdot h_t$$

Unroll: h_0, u_0 u_1 write h_1, write h_2, y_2.

$$h_t = A^t \cdot h_0 + \sum_{i=0}^{t-1} A^i B u_{t-i}$$

$$y_t = CA^t h_0 + \sum_{i=0}^{t-1} CA^i B u_{t-i} \quad \rightarrow \text{like FIR.}$$

How is this different from FIR?

Say $h_0 = 0$.

$H_i = CA^i \cdot B$.

Special case: A is nilpotent

How does memory help?

E.g. 1: Running avg.

$x_1, x_2 \cdots x_n$

$t \cdot \bar{x}_t = \bar{x}_{t-1}(t-1) + x_t$

Running avg

$$\bar{x}_t = \frac{1}{t}\left(\bar{x}_{t-1}(t-1) + x_t\right)$$

$\bar{x}_{t-1}$: state

x_t: input

State update

Running sum. $\bar{z}_t$

$\bar{z}_t = \bar{z}_{t-1} + x_t$

Eg. 2 Largest two elements of a seq.

$u_1, u_2, \cdots \quad u_t, \cdots$

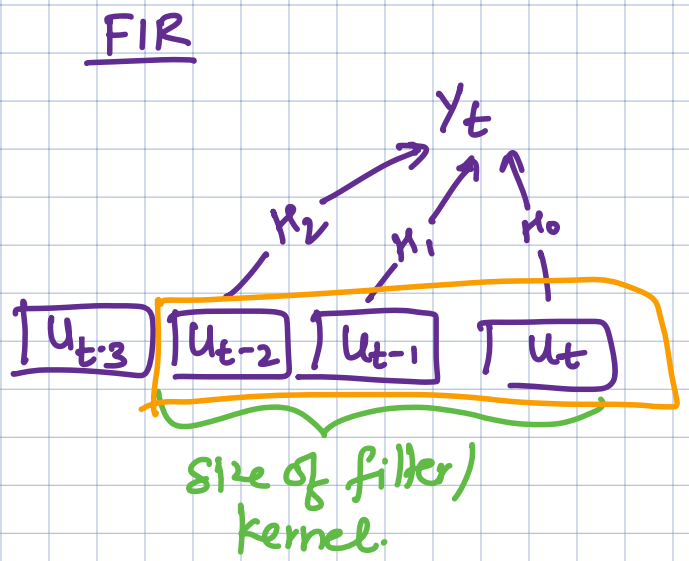
a_t : max

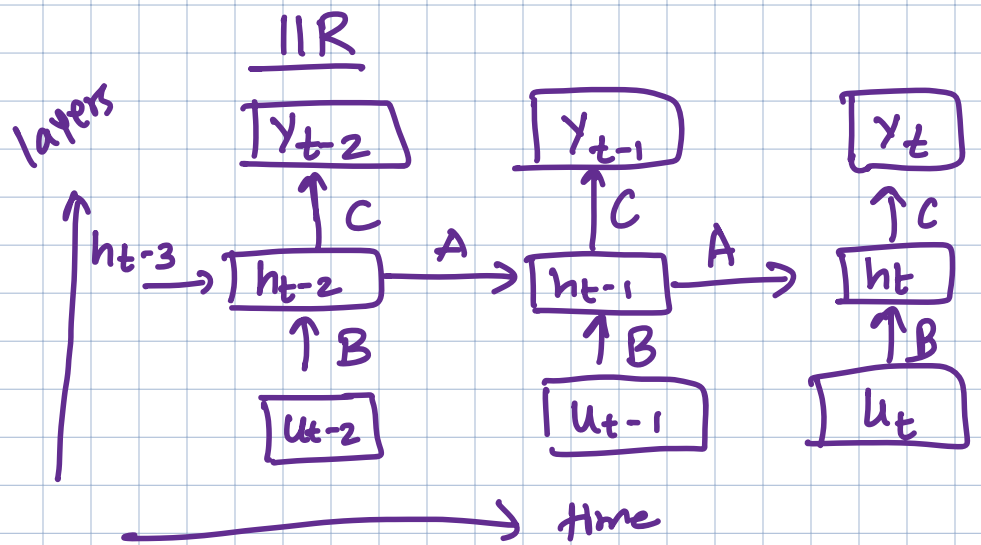
b_t : second

$$n_t = \min(n_{t-1} + 1, 2)$$

$$a_t = \begin{cases} u_t & n_{t-1} = 0 \\ \max(a_{t-1}, u_t) & n_{t-1} \geq 1 \end{cases}$$

$$b_t = \begin{cases} \min(a_{t-1}, u_t) & n_{t-1} = 1 \\ \max\{b_{t-1}, \min(a_{t-1}, u_t)\} & n_{t-1} = 2. \end{cases}$$





Signal Processing

FIR Filter

IIR Filter

- Finite dim hidden state inside filter
- Sequential + Causal

Neural Nets

CNN

RNN

Weight Sharing

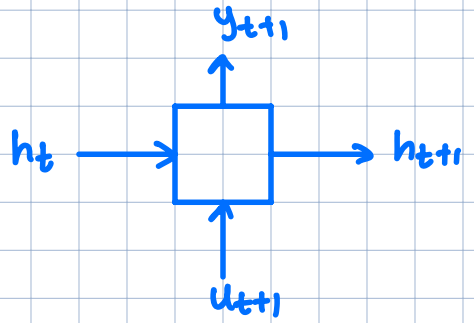
Across space.

Across time

Basic building block:

$$\vec{h}_{t+1} = W \cdot \vec{h}_t + B \cdot \vec{u}_{t+1}$$ Update state causally

$$\vec{y}_{t+1} = C \cdot \vec{h}_{t+1} + D \cdot \vec{u}_{t+1}$$ Generate filter outp



Counterpart of a conv.

Approach: Keep all the design ideas and principles from CNNs.

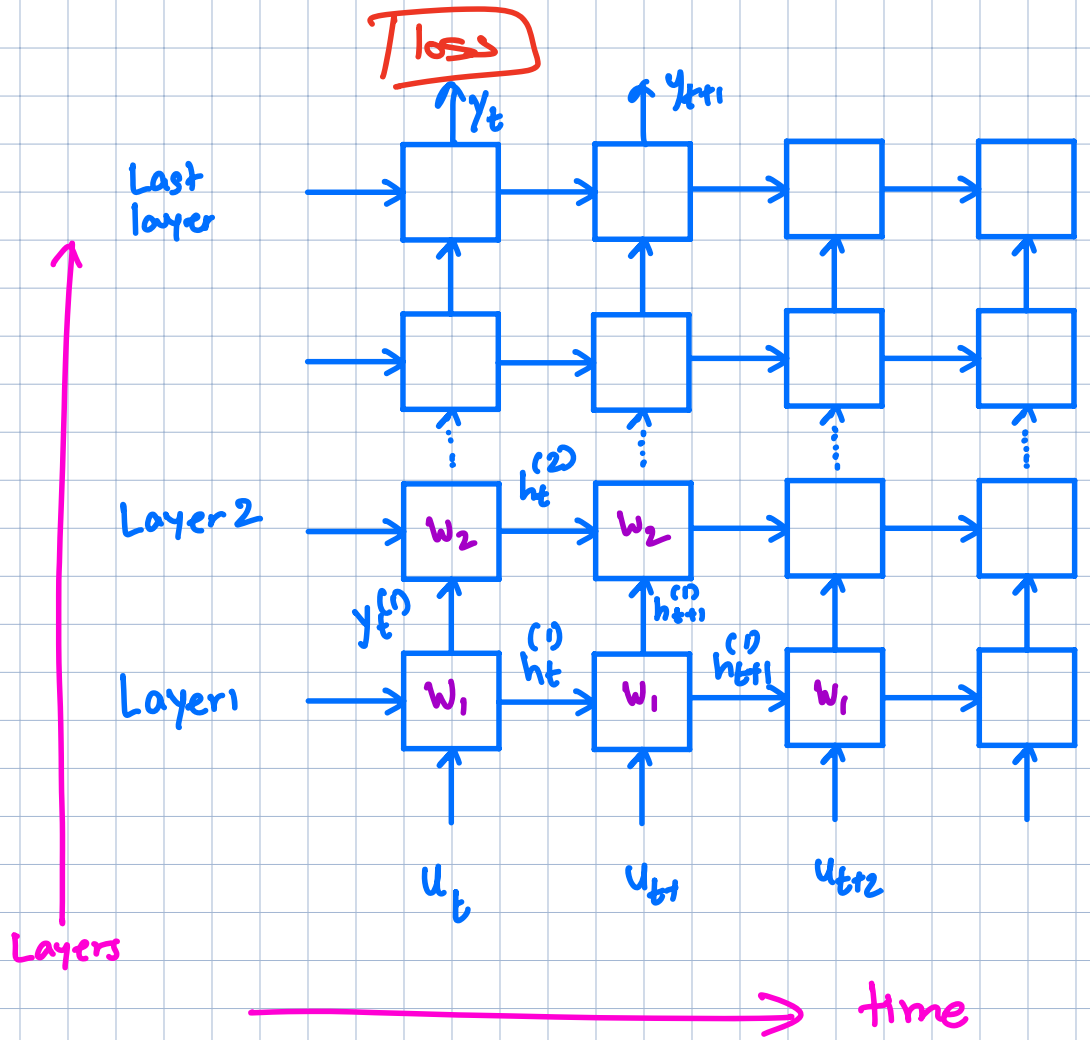
Just replace space with time.

Enforce causality

"Residual" connection

$$y_{t+1} = C h_{t+1} + \underbrace{D u_{t+1}}$$

like a residual connection



- Vertical Residual Connections
- Vertical 1×1 MLPs
- Vertical normalization.

  e.g. Layer Norm (i.e. input wise normalization)

→ Cannot normalize with future.

→ No Batch norm
- ↳ Wt. sharing across time
- ↳ Sequences not same length. So you can't avg across both.

Question:

Traditional RNN perspective: Non-linearity in state update.

$$\vec{h}_{t+1} = W\vec{h}_t + B\vec{u}_{t+1} + \vec{b}_h$$
$$\vec{y}_{t+1} = C \cdot \vec{h}_{t+1} + \vec{b}_y$$

$\vec{b}_h$, $\vec{b}_y$ → bias

With non-lin.

$$\vec{h}_{t+1} = \sigma\left(W\vec{h}_t + B\vec{u}_{t+1} + \vec{b}_h\right)$$
$$\vec{y}_{t+1} = C \cdot \vec{h}_{t+1} + \vec{b}_y$$

Typically: sigmoid or tanh.

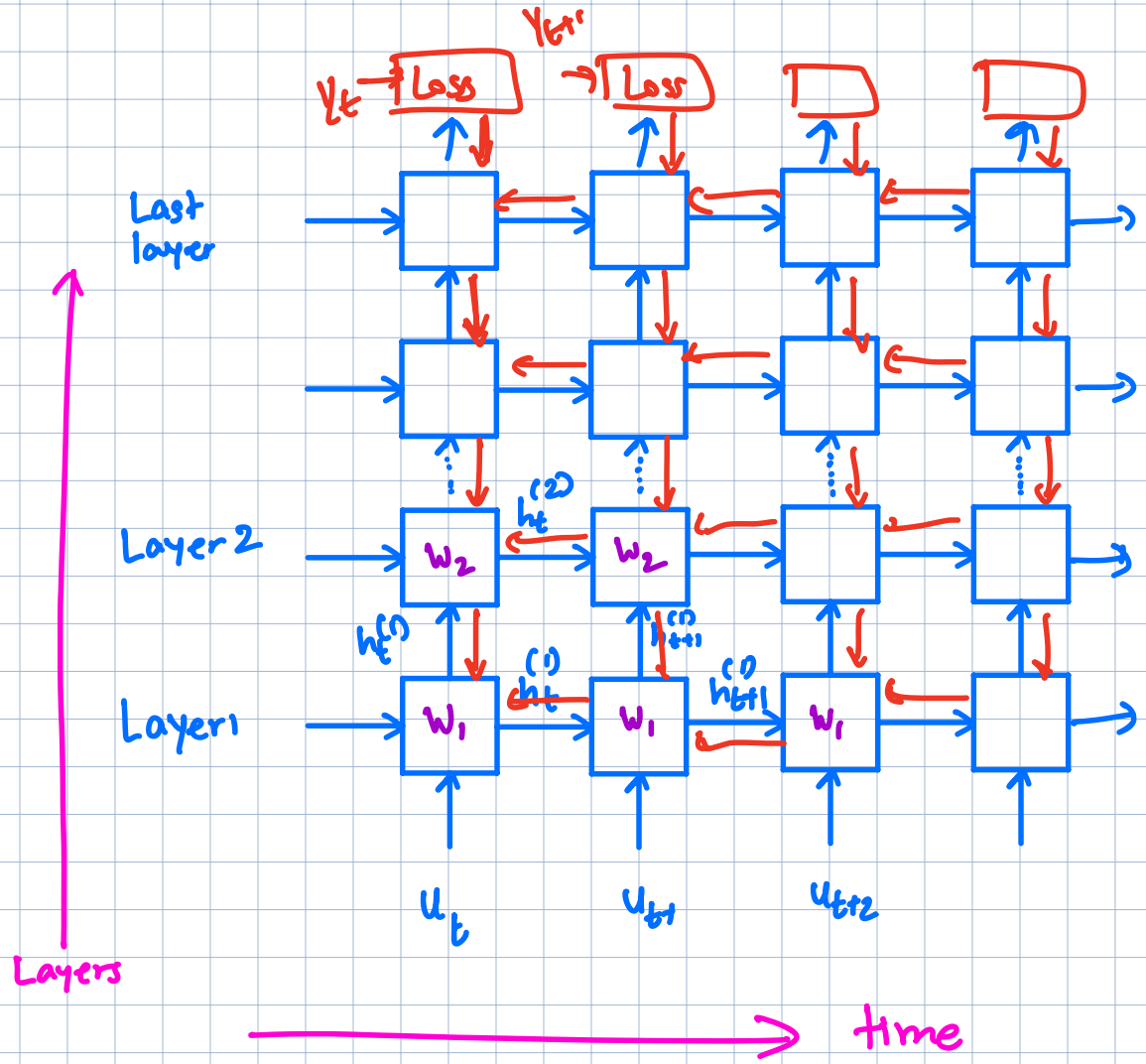
Why?

"Vanilla" RNN : the hidden state is the output

$$h_{t+1}^{(l)} = \sigma\left(W^{(l)}\vec{h}_t + B^{(l)} \cdot \vec{h}_{t+1} + \vec{b}\right)$$

state input

Backprop?



Gradients flow

- Down through depth
- Backward through time

Backpropagation through time. BPTT

Challenge?

$$J_t = \frac{\partial h_t}{\partial h_{t-1}}$$

$$\frac{\partial l_t}{\partial h_s} = \frac{\partial l_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{s+1}}{\partial h_s}$$

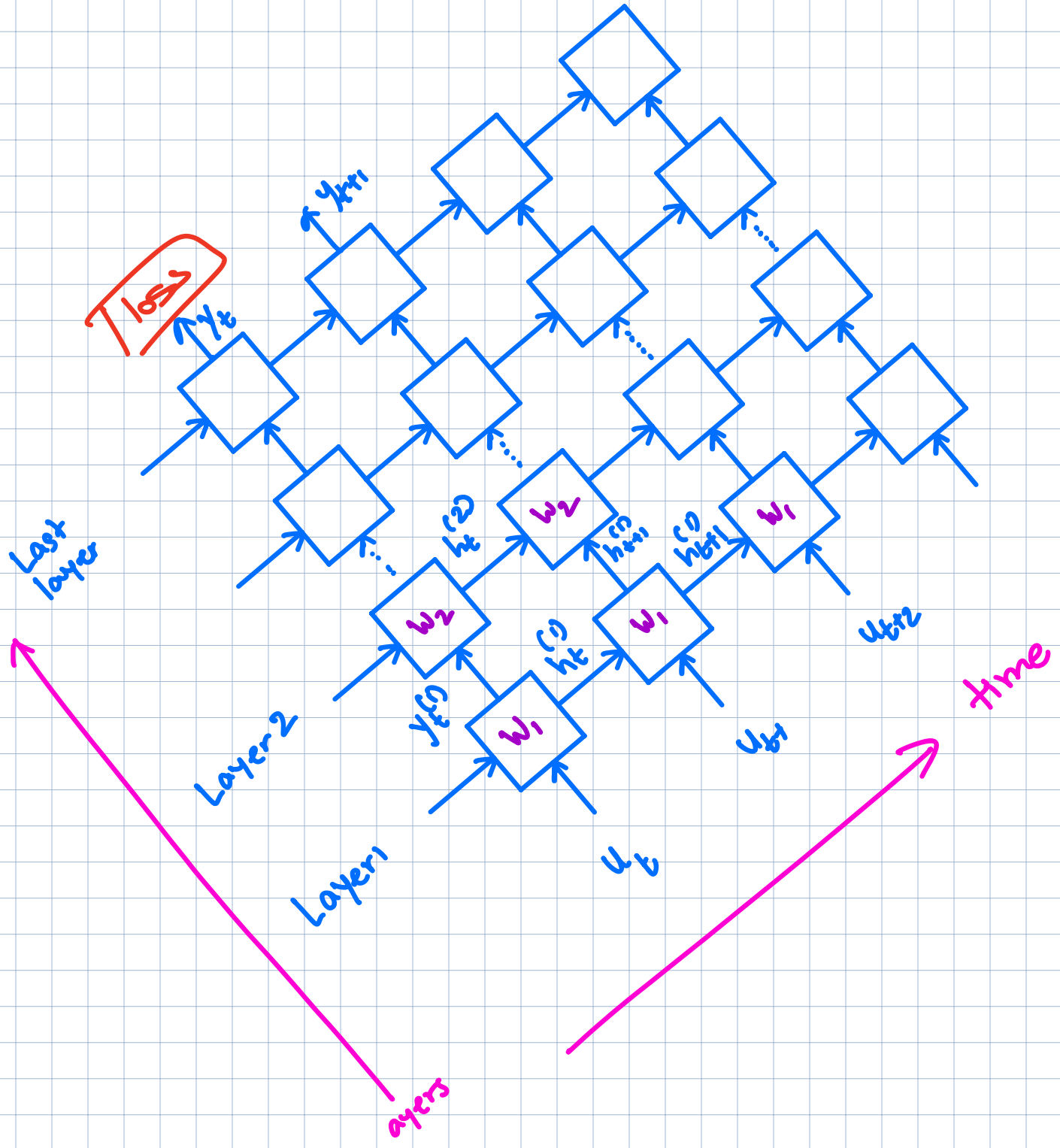
$$= \frac{\partial l_t}{\partial h_t} \cdot J_t \cdot J_{t-1} \cdots J_{s+1}$$

Forward pass → stable

Backward pass → exponentially decaying.

So what is the problem?

Traditional answer: Hard to learn long range dependencies.


Loss
y_t
y_{t+1}
W_2
W_1
$h_t^{(2)}$
$h_{t+1}^{(2)}$
$h_t^{(1)}$
$h_{t+1}^{(1)}$
$y_t^{(2)}$
u_t
u_{t+1}
u_{t+2}
Last layer
Layer 2
Layer 1
layers
time

Kalman Filtering

Controlled Linear Gaussian System.

$$x_{t+1} = Ax_t + Bu_{t+1} + w_{t+1} \qquad w_t \sim N(0, K_w)$$

$$y_{t+1} = Cx_{t+1} + Du_{t+1} + v_{t+1} \qquad v_t \sim N(0, K_v)$$

$\hat{x}_t = E[x_t \mid u_{1:t}, y_{1:t}]$ $\longrightarrow$ Expected value if you know all controls + observations till current time.

$$\hat{x}_{t+1} = \tilde{A}\hat{x}_t + \tilde{B}u_{t+1} + Fy_{t+1}$$

$$\tilde{A} = (I - FC)A \qquad \tilde{B} = (I - FC)B - FD$$

$$F = PC^T(CPC^T + K_v)^{-1}$$

and P satisfies

$$P = APA^T + K_w - APC^T(CPC^T + K_v)^{-1}CPA^T$$

Consider now you only have data

Traces: $x_{1:T}, u_{1:T}, y_{1:T}$

Say we have many, each of a different length

Don't know A, B, C, D, K_w, K_v.

Can we set this up as a deep learning problem?

- $\bar{A}, \bar{B}, F$ are learnable parameters (like linear transform in RNN)

→ Cannot have a loss on these

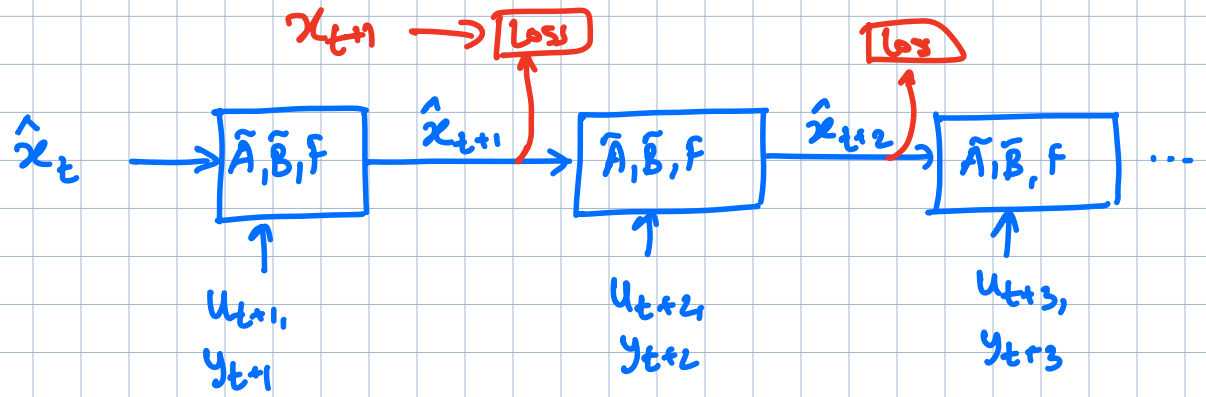
→ So what do we do?

- loss on x_t → $\|x_t - \hat{x}_t\|^2$.
- Run with u_t, y_t as inputs.

→ Actually: $\frac{1}{B}\sum_{i=1}^{B} \|x_i - \hat{x}_i\|^2$

B: Block size

→ How much can you fit in memory?



Now, what if you don't have x_t either.

Only controls and observations.

u_t data
y_t data.

Issue 1: We cannot uniquely identify $\tilde{A}, \tilde{B}$ etc.

Why?

→ Change of basis.

$$x_{t+1} = Ax_t + Bu_{t+1} + w_{t+1} \qquad w_t \sim N(0, K_w)$$

$$y_{t+1} = Cx_{t+1} + Du_{t+1} + v_{t+1} \qquad v_t \sim N(0, K_v)$$

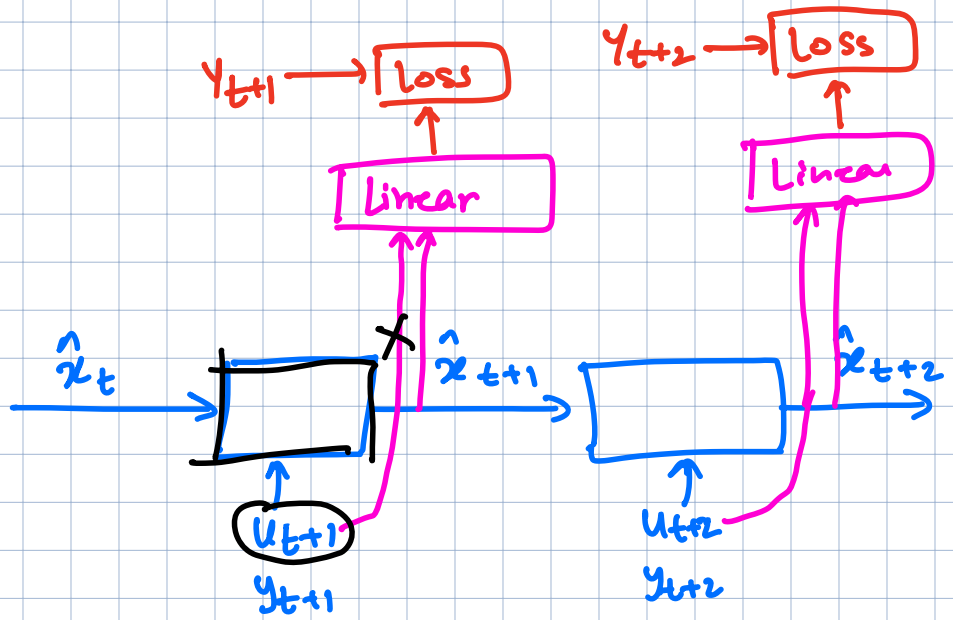
$$Mx_{t+1} = MAM^{-1}Mx_t + MBu_{t+1} + Mw_{t+1}$$

$$\tilde{x}_{t+1} = MAM^{-1}\tilde{x}_t + MBu_{t+1} + \tilde{w}_{t+1} \qquad \tilde{w}_t \sim N(0, \text{appropriately modified } K_w)$$

$$y_{t+1} = \underline{CM^{-1}}\tilde{x}_{t+1} + Du_{t+1} + v_{t+1}$$

Φ

Assume A is diagonal. Then.



Self-supervision.

→ Challenge: Need labels to train.

But don't always have them.

→ Make your own labels! (from data)

Lessons from Kalman Filtering example.

1. We can learn a partial pattern that can be useful
2. Might need scaffolding.
3. Generic idea: "next-thing" prediction in causal sequence modeling.

Step back: Classical ML.

Unsupervised Learning

1. Dimensionality Reduction
2. Clustering

PCA

$$X = \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_n^T \end{bmatrix}$$

1. Compute SVD $X = U \Sigma V^T$
2. $\vec{v}_1, \vec{v}_2 \dots \vec{v}_k$ are top k singular vectors.

   Project into this basis.

$$\vec{x} \longrightarrow \begin{bmatrix} \vec{v}_1^T \\ \vdots \\ \vec{v}_k^T \end{bmatrix} \cdot \vec{x} \longrightarrow k \times 1 \text{ vector}$$

But what if we wanted to do this DL style?

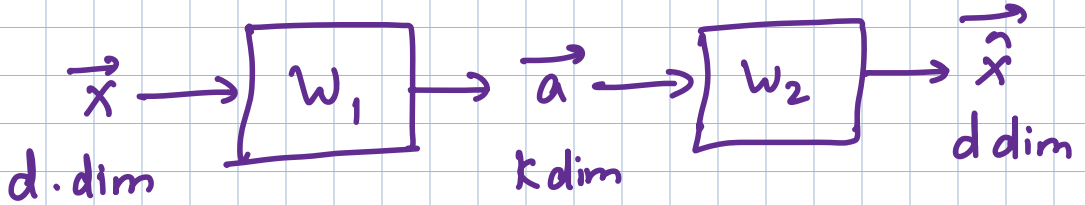
But what if we wanted to do this DL style?

→ Learn $v_1, v_2 \cdots v_k$

Eckart-Young: $\hat{X}_k = \sum_{i=1}^{k} \sigma_i \vec{u_i} \vec{v_i}^T$ is the rank-k matrix that minimizes

$$\min_{\text{Rank}(Y) \le k} \|X - Y\|_F^2$$

How would I learn this?



Dimensionality reduction

Reconstruction

Minimize $\|\vec{x} - \hat{\vec{x}}\|_2^2$

Autoencoders

- Labels are x_i itself

- Encoder – decoder architecture
- Bottleneck in the middle

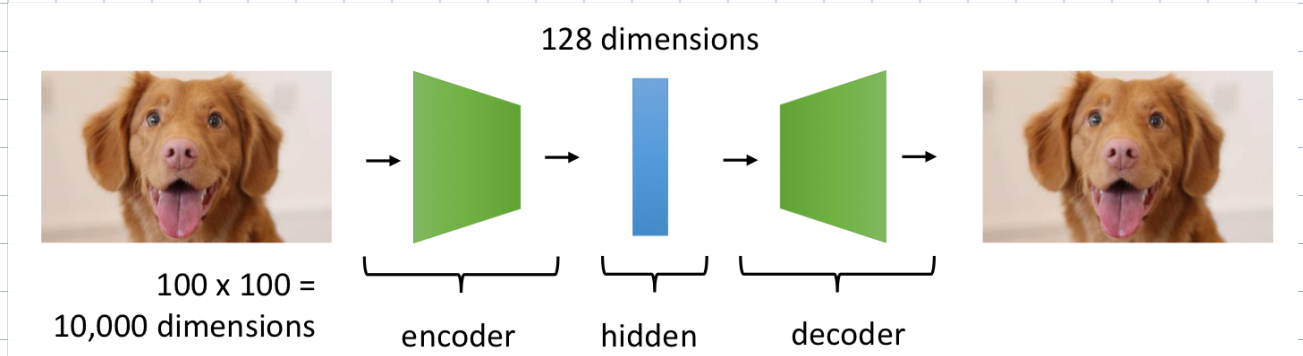
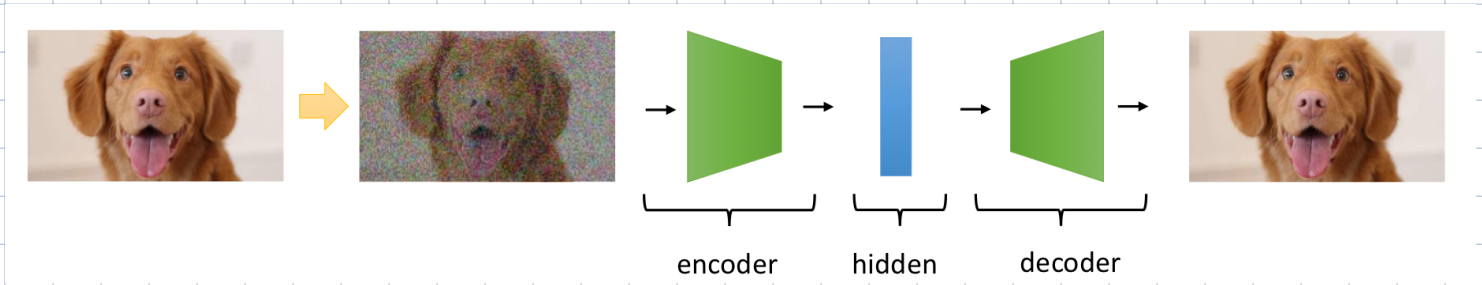


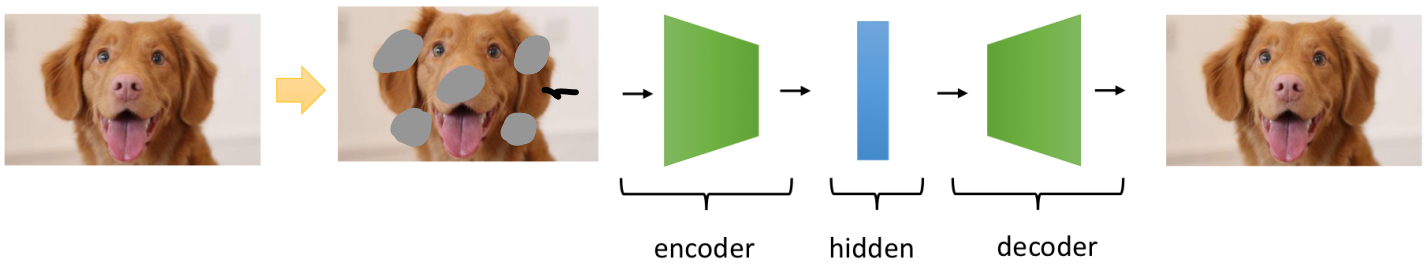
Fig. Credit
Levine 182 lecture
2021

Desired output
Use relevant loss.

Denoising autoencoder



→ Use data augmentation



Masked autoencoder

Sparse autoencoder

