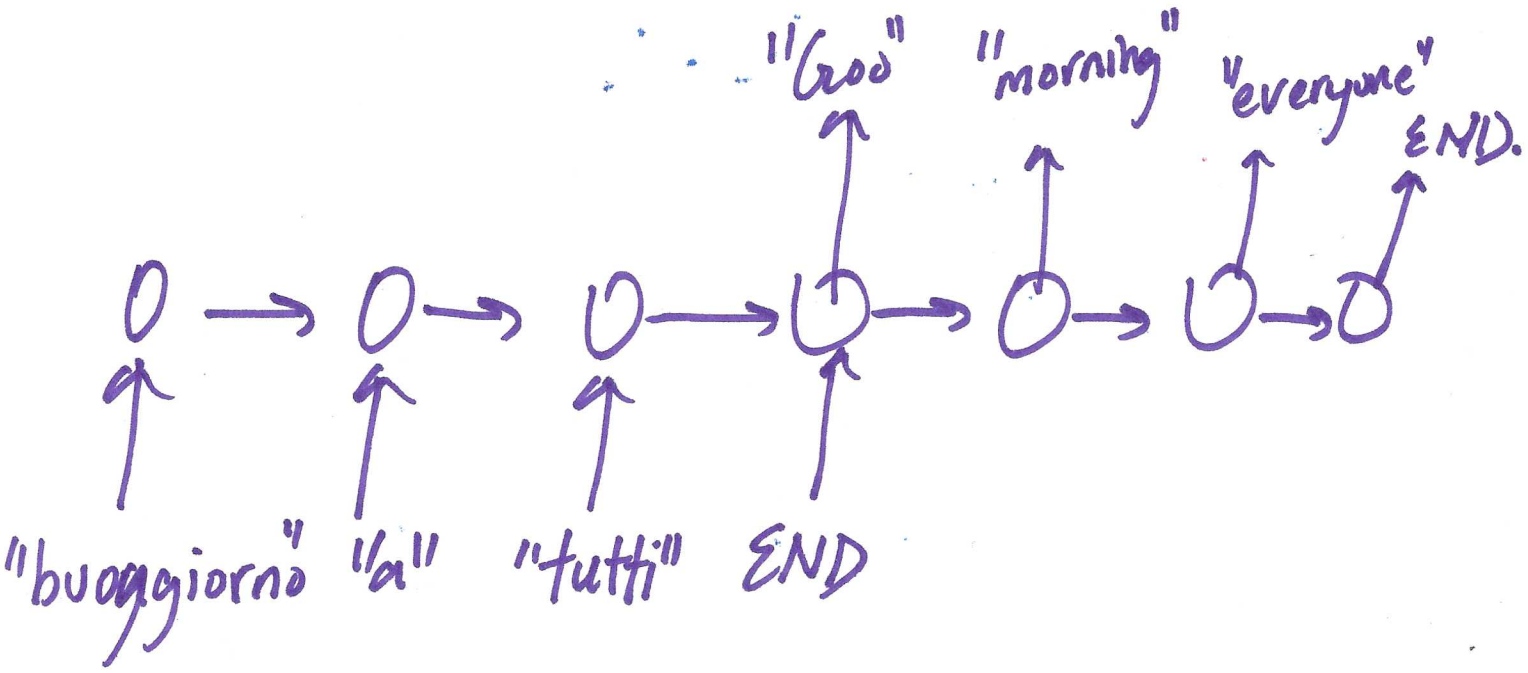
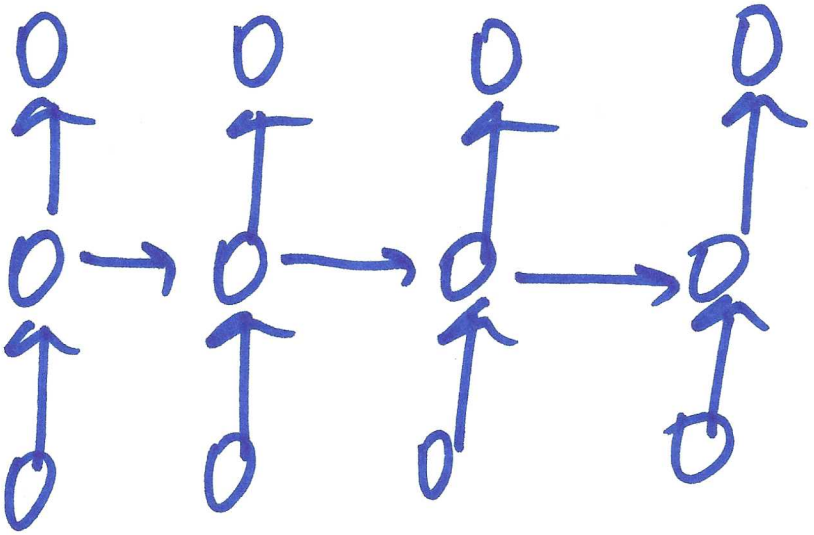
Many-to-many architecture. (6)

Example: Machine translation



Synch. many-to-many architecture.



e.g. frame-by-frame video classification. etc.

many other variants...

Simple RNN architecture (7)

x_t: Input at time $t \in \mathbb{R}^{d \times 1}$

s_t: State at time $t \in \mathbb{R}^{N \times 1}$

y_t: Output at time $t \in \mathbb{R}^{d \times 1}$.

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

s_{t-1}: recurrence

$W, U \in \mathbb{R}^{N \times d}$ are trainable weights.

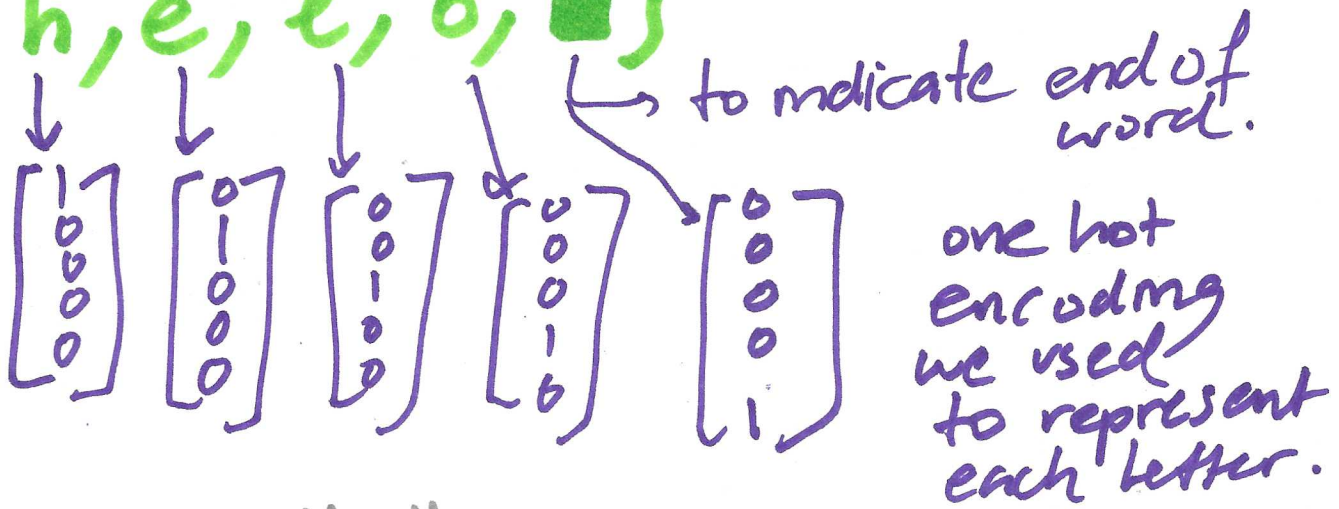
$$y_t = Vs_t$$

(one can also do sigmoid, hard decisions (step func.) etc.

e.g. $y_t = \sigma(Vs_t)$ where σ is the sigmoid act. function.

Ex: An RNN trained to generate words over the alphabet

$\{h, e, l, o, \blacksquare\}$



E.g: $x_1 =$ "e", $x_2 =$ "e"

(First two letters of the word are "ee" what will come after is likely "l", indicating eel (a certain kind of fish))

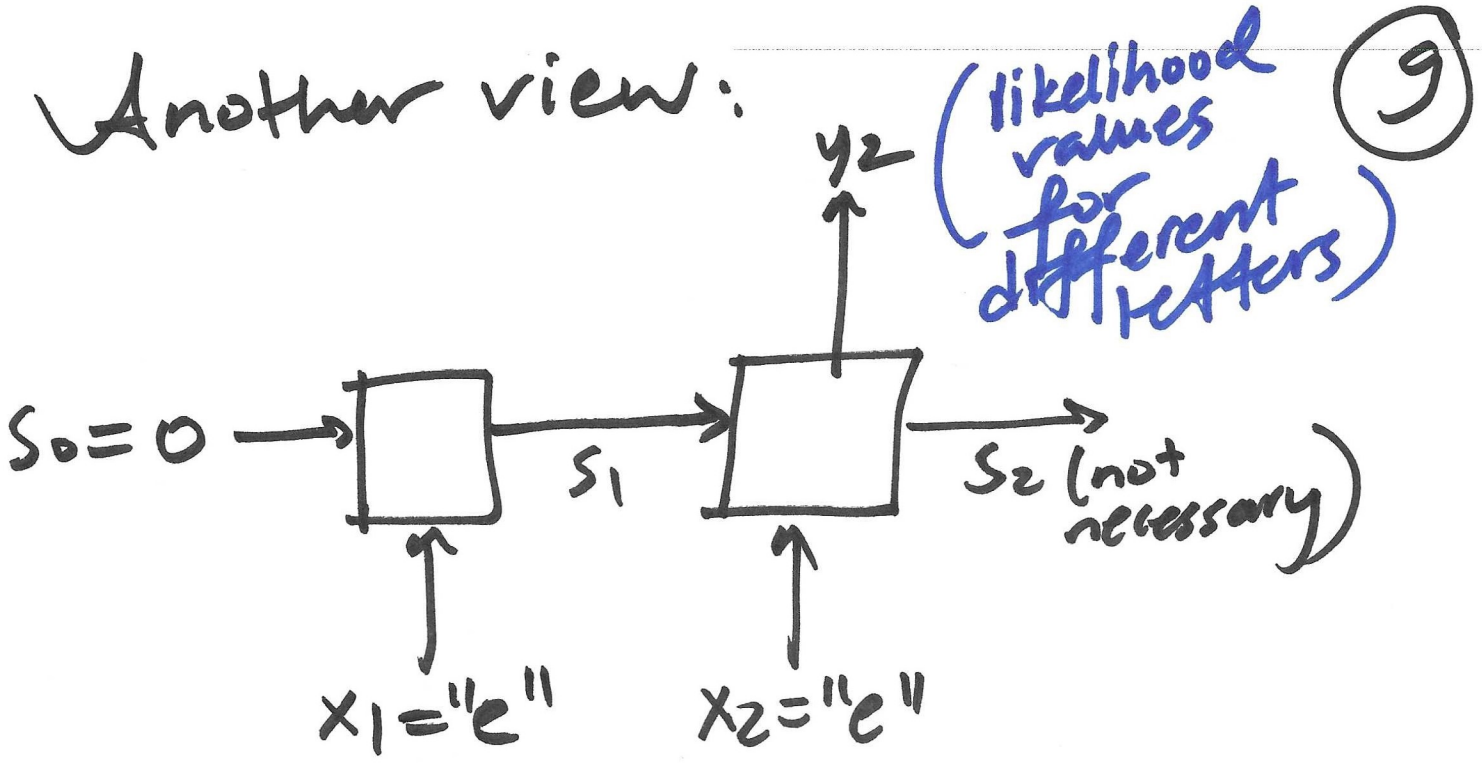
We are interested in predicting what comes after "ee", which we determine through y_2.

$$y_2 = \sigma(Vs_2) = \sigma(V \tanh(Ux_2 + Ws_1))$$

$$s_1 = \tanh(Ux_1 + Ws_0)$$

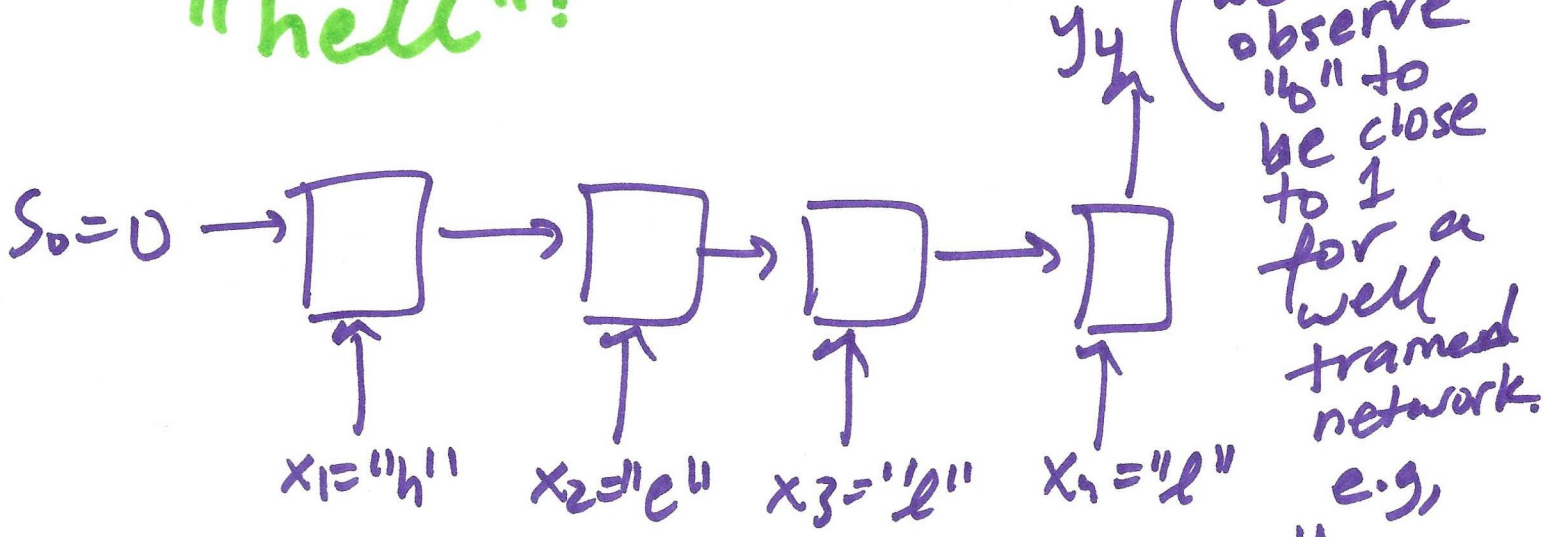
$s_0 =$ some fixed 5-dim vector (e.g. all zero vector).

Another view:



9

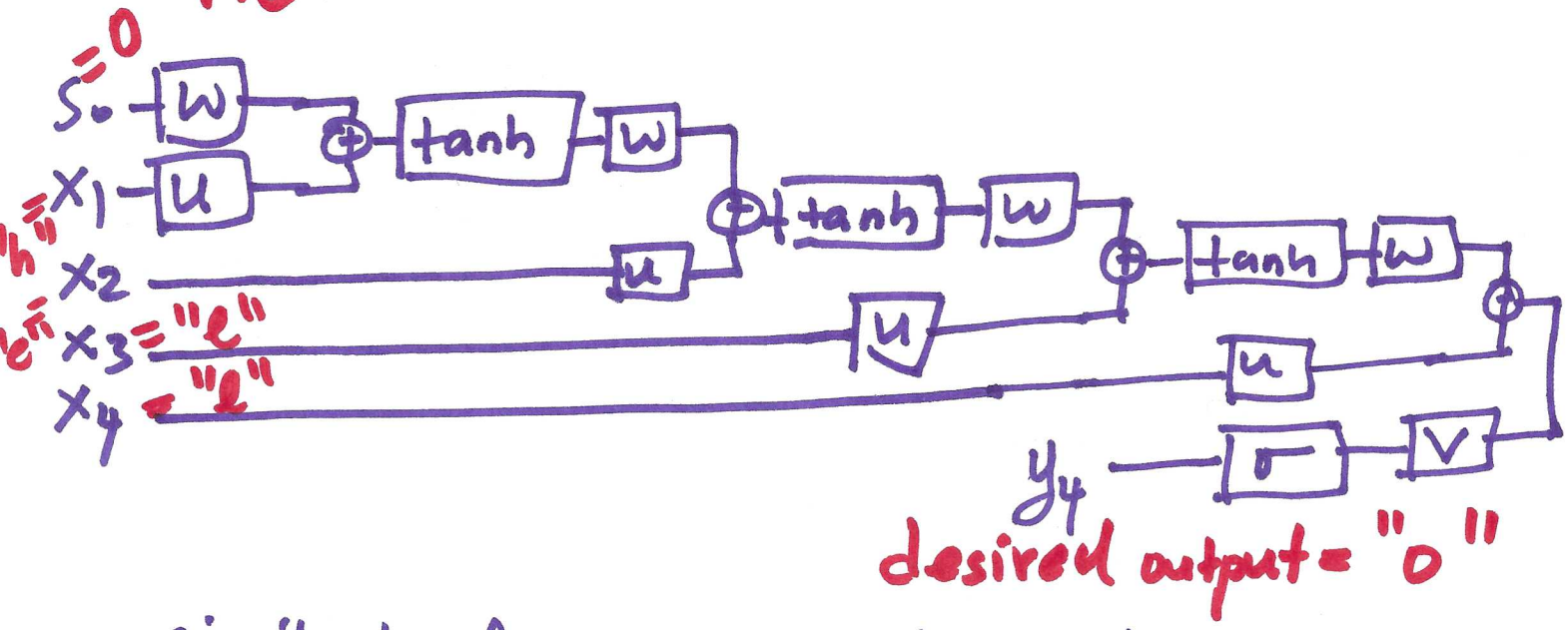
Same network for initial sequence "hell":



Note that "o" = $[0, 0, 0, 1, 0]$

Training RNNs

View each example above as one instance of an ordinary feedforward NN. For example, for the "hello" prediction, the network is:



Similarly for any such ~~desired~~ inputs - desired output pairs the RNN can be unrolled any number of times and trained using standard backpropagation.

The resulting algorithm is referred to as backpropagation through time (BPTT).

Long-Short Term Memory

(11)

As we consider longer sequences, we face the vanishing & exploding gradient problems in the standard RNN structure.

LSTMs alleviate (though not entirely solve) the vanishing & exploding ∇ problems of RNNs.

We simply replace the basic recurrence block with another block. Define

x_t: Input at time t

y_t: Output at time t

s_t: State at time t

As usual. LSTMs define several new "gates":

"Forget" Gate: (how much of the previous state to forget)

$$f_t = \sigma(U^f x_t + W^f y_{t-1} + b^f)$$

Input Gate: (how much of the new candidate state to keep)

$$i_t = \sigma(U^i x_t + W^i y_{t-1} + b^i)$$

Output Gate: (how much of the calculated output to expose)

$$o_t = \sigma(U^o x_t + W^o y_{t-1} + b^o)$$

New candidate state:

$$g = \tanh(U^g x_t + W^g y_{t-1} + b^g)$$

New state: → Hadamard (element-wise) product

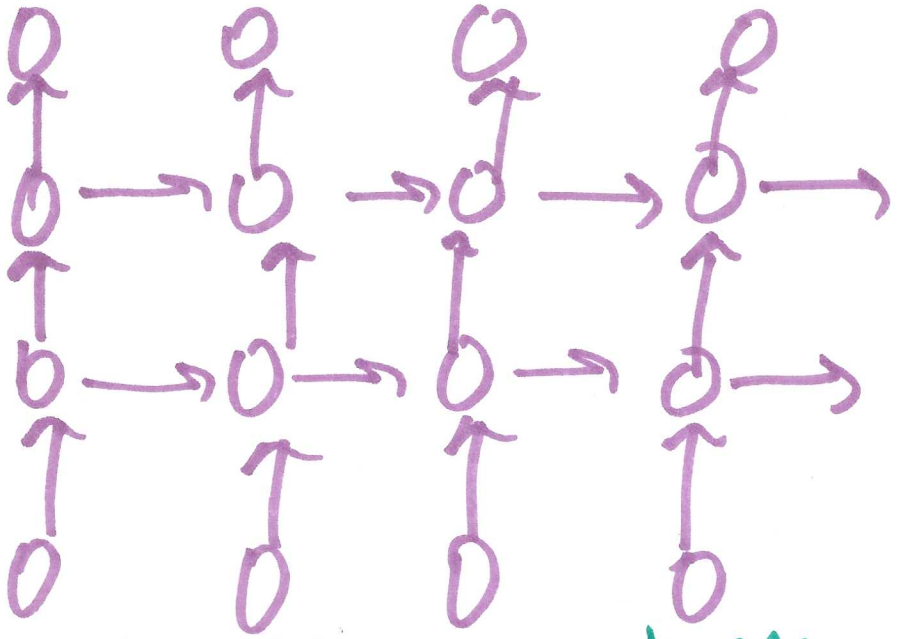
$$s_t = s_{t-1} \odot f_t + g \odot i_t$$

output:

$$y_t = \tanh(s_t) \odot o_t$$

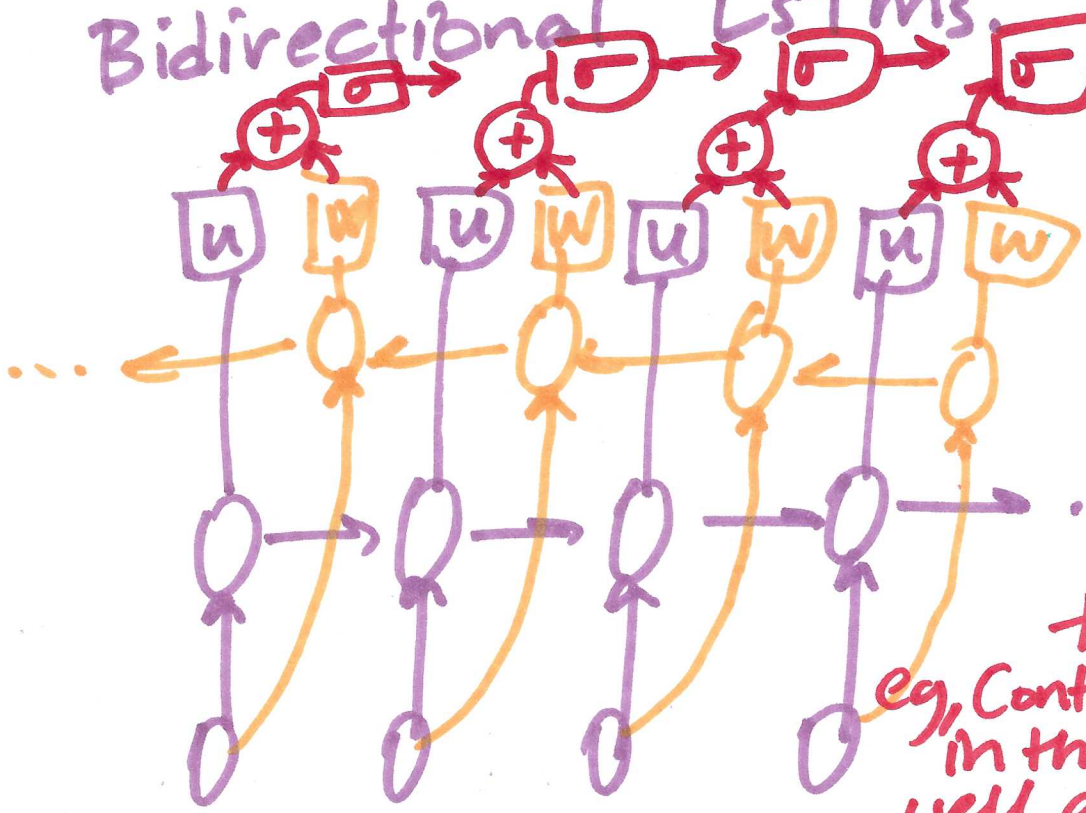
- All weights, biases are trainable.
- Peephole LSTMs include a trainable weight × previous state to the induced local fields.

Stacked LSTMs



* motivation: capturing more complex features (as in deep vs shallow NNs).
* Not much gains to have >3 stacks.

Bidirectional LSTMs



- Information flows in both negative & positive time directions.
- Useful in machine translation.

eg, Context can be hidden in the beginning as well as at the end of a sentence.

L22&23

ECE/CS 559 Lecture 23 11/19/2024

Last time: Hierarchical Autoencoders

① Generative-Adversarial Networks (GANs)

Motivation: In both density estimation and auto-encoders, we use likelihoods / ELBO losses:
- These are problematic in high dimensions, where the sparsity of the data makes it hard to be accurate.
- They evaluate each data point individually, instead of seeing whether the generation overall is good.

- Intuition: If a very good discriminator cannot distinguish actual images from those made by a generator, then the generator is quite good. Note:
  - No P_X, instead the discriminator judges images (perceptual)
  - The discriminator judges the overall (not individual) performance

Model
- Let S be a coin toss: $P_S(s) = \begin{cases} \frac{1}{2} & s=H \text{ (actual)} \\ \frac{1}{2} & s=-1 \text{ (fake)} \end{cases}$
- The discriminator produces $P_{disc}(s \mid x)$, judging probability of actual/fake
- The generator samples x from $P_{gen}(x)$, by sampling z then $x=G(z)$
- The discriminator sees x from data, P_{pop}, if $s=+1$, or generator if $s=-1$

$$P_X(x) = P_S(+1)\cdot P_{pop}(x) + P_S(-1)\cdot P_{gen}(x)$$
(seen by discriminator) $\quad P_{X|S}(x|+1) \quad P_{X|S}(x|-1)$

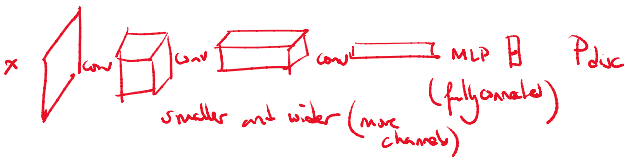
- We measure the performance of the discriminator via binary cross-entropy: $\mathbb{E}_S\left[\mathbb{E}_{x|S}\left[-\log P_{disc}(S|x)\right]\right]$
- Empirically, generate as many z's as x's in Data, match each x to a z, then calculate:

$$R(W) = \frac{1}{|Data|}\sum_{x\in Data} -\frac{1}{2}\log \underbrace{P_{disc}(+1|x)}_{D(x)} - \frac{1}{2}\log \underbrace{P_{disc}(-1 \mid G(z))}_{1-D(G(z))}$$

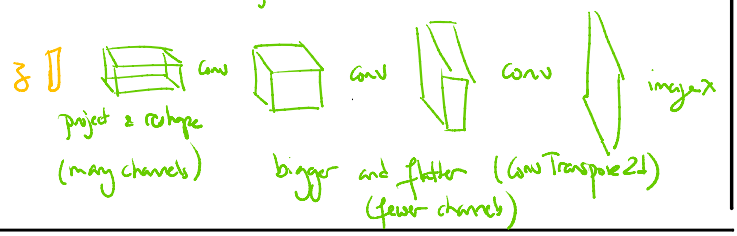
- Goal: Discriminator tries to ↓ cross-entropy and generator tries to ↑. (get better at telling apart) (get better at fooling)

$$\arg\max_{P_{gen}} \min_{P_{disc}} R(W)$$
all weights (disc & gen)

- What do the discriminator and generator look like.
- Discriminator: CNNs, standard



- Generator: CNNs, in reverse



② Analyzing GANs

Why do we think this would work?
- Let's assume infinite data + universality (can approximate all)

$$\arg\max_{P_{gen}} \min_{P_{disc}} \mathbb{E}_S\left[\mathbb{E}_{x|S}\left[-\log P_{disc}(S|x)\right]\right]$$

actual $P(S|x) = \frac{P(S)\cdot P(x|S)}{P(x) = \frac{1}{2}P_{pop} + \frac{1}{2}P_{gen}}$

$$\arg\max_{P_{gen}} \frac{1}{2}\mathbb{E}_{x\sim P_{pop}}\left[-\log P(+1|x)\right] + \frac{1}{2}\mathbb{E}_{x\sim P_{gen}}\left[-\log P(-1|x)\right]$$
$P(+1|x) = \frac{\frac{1}{2}\cdot P_{pop}}{\frac{1}{2}P_{pop}+\frac{1}{2}P_{gen}}$ $\quad P(-1|x) = \frac{P_{gen}}{\frac{1}{2}P_{pop}+\frac{1}{2}P_{gen}}$

KL Divergence

$$\arg\max_{P_{gen}} KL\left(P_{pop} \,\|\, \tfrac{1}{2}(P_{pop}+P_{gen})\right) + KL\left(P_{gen} \,\|\, \tfrac{1}{2}(P_{pop}+P_{gen})\right)$$

$$\arg\max_{P_{gen}} JSD(P_{pop}, P_{gen}) \Rightarrow \boxed{P_{gen}^* = P_{pop}}$$

Jensen-Shannon Divergence

③ Conditional Generation

- All a form of translation, e.g., image-to-image

style change x' → x

(a) With paired images (x', x) given x' generate x

$$\underset{gen}{\operatorname{argmin}} \mathbb{E}\left[\|x - G(z,x')\|_1\right] + \underbrace{\lambda_{max}}_{disc}\underbrace{\mathbb{E}\left[-\log P_{disc}(S|x)\right]}_{R_{disc}}$$

(b) with unpaired images $\{x'\}$ $\{x\}$

Cycle GAN: 2 generators & 2 discriminators (for x & x')

$$\mathbb{E}\left[\|G_2(G_1(z_1,x')) - x'\|_1\right] + \mathbb{E}\left[\|G_1(G_2(z_2,x)) - x\|_1\right] \text{ (cycle loss)} + \lambda R_{disc'} + \lambda R_{disc}$$

L19&20

ECE/CS 559 Lecture 19/20 Th 10/31; Th 11/7

Last time: Hebbian updates

① Generative Models

- All unsupervised learning is about understanding $P(x)$
- Generative models attempt to create more x:
  - They can do this by modeling $P(x)$ & sampling from it.
  - Or, they can directly model the sampling mechanism
    e.g., sample z (some standard distribution), then let $x = f(z) + \text{noise}$
- Conditional generative models extend to generation via prompts: $P(x|y)$

Examples: y = caption, x = image (diffusion models)
y = question, x = answer (transformers)

② Density Estimation

Given Data x, assume i.i.d. from P.

Parametric methods: Let $P(x) = f(x; w)$

Maximize likelihood: $\arg\max_w P(\text{Data}) = \prod_{x \in \text{Data}} P(x)$

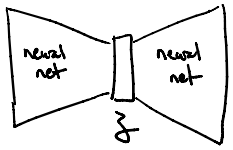
$\Leftrightarrow \arg\min_w \sum_{x \in \text{Data}} -\log P(x)$ ← log loss

Non-parametric methods: Allow # of parameters (size of w) to grow with data size. Example, kernel density estimation

Works well when x discrete or small dimensional.
Otherwise, requires regularization, e.g. smooth densities.

③ Autoencoders (Variational; VAEs)

- They try to capture $P(x)$ by reducing the "essence" of x to a simple representation z (encoding)
  then generating x from a sample of z (decoding)



$f(z)$, approximates x (+ noise)

Generally: $P_{enc}(z|x)$ $P_{prior}(z)$ $P_{dec}(x|z)$

- Why sampling $z \sim P_{prior}(z)$ then letting $x \sim f(z)$ work?
  How do we choose P_{enc}, P_{dec}?

④ Derived distributions:

- Say $x \in \mathbb{R}^1$. Let $z \sim P_z(z)$ and let $x = f(z)$.
- What is $P_x(x)$? Let's assume P's are densities, f monotonic)
- $F_x(x) = P(X \le x) = P(f(z) \le x) = P(z \le f^{-1}(x))$ decreasing

thus $P_x(x) = \frac{dF}{dx} = \frac{d}{dx} F_z(f^{-1}(x)) = \frac{d}{dz} F_z \cdot \frac{dz}{dx} = P_z(f^{-1}(x)) \cdot \left|\frac{df^{-1}}{dx}\right|$

Example: $P_z \sim \text{Uniform}[0,1]$ $x = -\log z$ $f^{-1}(x) = e^{-x}$

$\Rightarrow P_x(x) = \begin{cases} -1 \cdot (-e^{-x}) & e^{-x} \in (0,1) \\ 0 & \text{otherwise} \end{cases} = \begin{cases} e^{-x} & x \ge 0 \\ 0 & x < 0 \end{cases}$

- Can create any new random variable from standard random variables (e.g. uniform, Gaussian, etc.)

⑤ From maximum likelihood to ELBO

How do we train an autoencoder?

Idea 1: Fix $P_{prior}(z)$ and only train the decoder $P_{dec}(x|z)$

by minimizing $\sum_{x \in \text{Data}, z \sim P_{prior}} -\log P_{dec}(x|z)$

Issue: Creates disconnect between x & z (close x's $\not\Rightarrow$ close z's)

Idea 2: Fix $P_{enc}(z|x)$ and train $P_{prior}(z)$ & $P_{dec}(x|z)$

by minimizing $\sum_{x \in \text{Data}, z \sim P_{enc}(\cdot|x)} -\log P_{prior}(z) \cdot P_{dec}(x|z)$

approximates $\mathbb{E}_x \mathbb{E}_{z \sim P_{enc}|x} -\log P_{prior}(z) P_{dec}(x|z)$ $\Leftrightarrow$

$\mathbb{E}_x[\log \frac{1}{P_{gen}(x)}] + \mathbb{E}_x[\mathbb{E}_{z \sim P_{enc}|x}[\log \frac{1}{P_{gen}(z|x)}]] - \mathbb{E}_x[\mathbb{E}_{z \sim P_{enc}|x}[\log \frac{1}{P_{enc}(z|x)}]]$

implicit $P_{gen}(x)$ $P_{gen}(z|x)$

min at $P_{gen}(x) = P_x(x)$ min at $P_{gen}(z|x) = P_{enc}(z|x)$ $-$ Entropy (Penc)

Issue: If the neural net of the decoder cannot easily "invert" the choice of P_{enc}, it won't work well.

Idea 3 Train P_{enc} and P_{dec} jointly.

Issue: P_{enc} in the loss is just a sampling distribution.

Hard to optimize: bad local minima / P_{enc} can degenerate by colluding w/ P_{dec}

Idea 4: Include P_{enc} in the loss: $\mathbb{E}\left[-\log \frac{P_{prior}(z) P_{dec}(x|z)}{P_{enc}(z|x)}\right]$

"max-entropy" effect

Evidence Lower Bound (ELBO) Loss $< \log P(x)$

(c) Example: Gaussian Variational Auto encoder

$p_{prior}(z) \propto \exp\left(-\frac{\|z\|^2}{2}\right)$ $\quad \mathcal{N}(0, I)$

$p_{encoder}(z|x) \propto \exp\left(-\frac{\|z - g(x;w)\|^2}{2}\right)$ $\quad \mathcal{N}(g(x;w), I)$

$p_{decoder}(x|z) \propto \exp\left(-\frac{\|x - f(z;w)\|^2}{2}\right)$ $\quad \mathcal{N}(f(z;w), I)$

ELBO has explicit form: $\mathbb{E}\left[\frac{\|z\|^2}{2} - \frac{\|z - g(x;w)\|^2}{2}\right] = \frac{1}{2}\mathbb{E}\left[\|g(x;w)\|^2\right]$

(minimize representation norm)

$+ \frac{1}{2}\mathbb{E}\left[\|x - f(\underbrace{g(x;w) + N}_{\text{encode}}; w)\|^2\right]$ $\quad N \sim \mathcal{N}(0, I)$

(minimize reconstruction error) — decode

This looks almost like kmeans and it's not a coincidence! The similarity is closer if we quantize z.

(4) Hierarchical VAEs

- Instead of one representation stage, have multiple:

$x \to z_1 \to z_2 \to z_3 \to z_2 \to z_1 \to x$

encode | decode

- Why? Allows for gradual tweaks, making it easier to train.

Example: Diffusion models use encoders that just add noise.

ELBO: $\log \frac{p_{prior}(z_3)\, p_{dec}(z_2|z_3)\, p_{dec}(z_1|z_2)\, p_{dec}(x|z_1)}{p_{enc}(z_3|z_2)\, p_{enc}(z_2|z_1)\, p_{enc}(z_1|x)}$

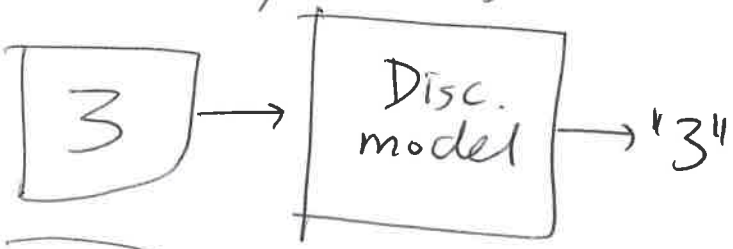
ECE/CS 559 - Neural Networks.

Generative Models.

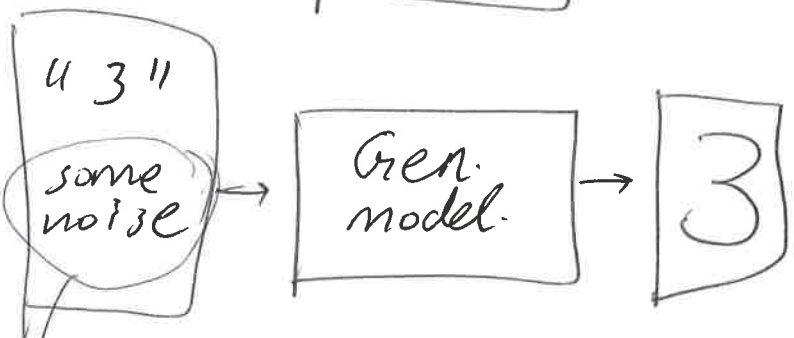
(1)

* Two main approaches in machine learning: generative vs. discriminative approach.
* Observation: X, Target: Y (e.g. an input image) (e.g. a class label).
* A discriminative model can provide $P(Y|X=x)$ (e.g. given the input, what are the likelihoods of different class labels).
* Whereas in a generative model, we are interested in finding $P(X|Y=y)$ Given label, generate some samples belonging to that label (kind of an inverse problem):

Discriminative model:



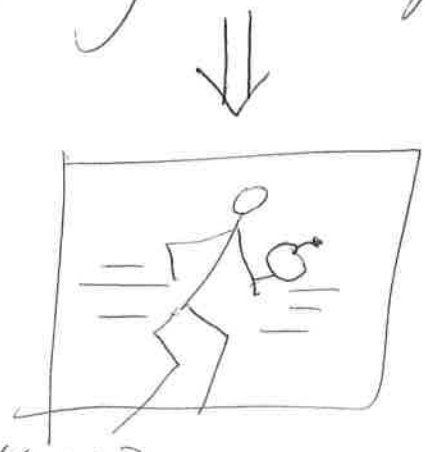
Generative model



* Noise could be considered as a "seed". Different noises would generate different images of 3's.

Applications of generative models (2)

- Low to high resolution image synthesis.
- Text to image translation
  "A guy running with an apple on his hand".



- Speech synthesis.
- Error correction

: