Lecture 24

[Generating images by inverting CNNs]

A trained CNN yields $p(y|\vec{x})$

(y: class label, $\vec{x}$: image)

This can be used to produce a generative model:

$$p(\vec{x}|y) \sim \underbrace{p(\vec{x})}_{\text{prior for images}} p(y|\vec{x})$$

Need to sample the energy function:

$$E_c(\vec{x}) = \log p(y=c|\vec{x}) + \log p(\vec{x}) .$$

Can use Langevin dynamics:

$$(1) \quad \vec{x}_{t+1} = \vec{x}_t + \frac{\epsilon}{2} \underbrace{\vec{\nabla} E_c(\vec{x}_t)}_{\text{-force}} + \mathcal{N}(\vec{0}, \epsilon' \mathbb{I})$$

or even

$$(2) \quad \vec{x}_{t+1} = \vec{x}_t + \epsilon_1 \frac{\partial \log p(\vec{x}_t)}{\partial \vec{x}_t} + \epsilon_2 \underbrace{\frac{\partial \log p(y=c|\vec{x}_t)}{\partial \vec{x}_t}}_{\text{Jacobian}} + \mathcal{N}(\vec{0}, \epsilon_3 \mathbb{I})$$

If $\epsilon_3 = 0$, the method generates the 'most likely' image for class c.

Image priors

- Gaussian prior: $p(\vec{x}) = \mathcal{N}(\vec{x} \mid \vec{0}, \mathbb{I})$

  ↑ for centered image pixels

$$\nabla_{\vec{x}} \log p(\vec{x}) = \nabla_{\vec{x}} \left[-\frac{|\vec{x}|^2}{2} \right] = -\vec{x}$$

Using $\epsilon_2 = 1$, $\epsilon_3 = 0$, we obtain: (Eq. (2))

$$\vec{x}_{t+1} = (1 - \epsilon_1)\vec{x}_t + \frac{\partial \log p(y = c \mid \vec{x}_t)}{\partial \vec{x}_t}$$

Total variation (TV) prior:

$$TV(\vec{x}) = \sum_{\underbrace{k}_{\text{channel}}} \sum_{\underbrace{i,j}_{\substack{\text{row } i, \\ \text{column } j}}} \left[(x_{ijk} - x_{i+1,jk})^2 + (x_{ijk} - x_{i,j+1,k})^2 \right]$$

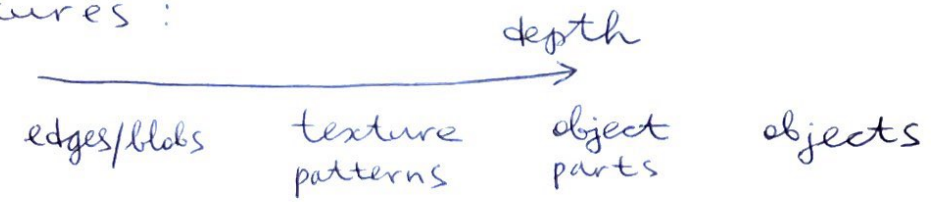
One can use $p(\vec{x}) \sim e^{-TV(\vec{x})}$ as the prior; penalizes high-freq. artifacts

Visualizing CNN features

Start with random image and optimize it to maximize the activation of a particular neuron ⇒ activation maximization

Goal: discover neuron functions

When applied to the AlexNet CNN, this procedure identifies a hierarchy of features:



Alternatively, one can simply search for images in a DB that maximally activate a given neuron ⇒ yields more 'relatable' images.

Deep Dream

Goal: generate versions of an input image that emphasize/exaggerate certain features

Patterns of activity of neurons in a given layer = different features in the image

Then define $\langle y_\ell \rangle = \frac{1}{HWC} \sum_{h,w,c} y_{\ell,hwc}(\vec{x})$

(ℓ: layer index; $y_{\ell,hwc}$: feature score)

Use $\sum_{\ell \in L} \langle y_\ell(\vec{x}) \rangle$ as the loss function, (part of)

to be optimized via grad descent

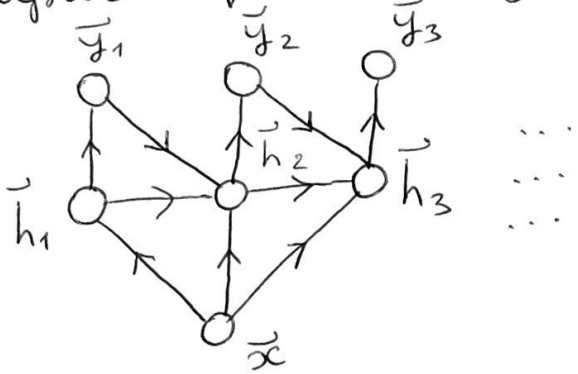
This procedure results in a combination of the original image with "hallucinations"

[Recurrent neural networks (RNNs)]

vec2seq models: $\mathbb{R}^D \to \mathbb{R}^{T \times C}$ ← T vectors of size C; T determined dynamically

fixed size of the input vector

Output sequence: $\vec{y}_{1:T} = \{\vec{y}_1, \dots, \vec{y}_T\}$



computes $p(\vec{y}_{1:T} | \vec{x})$ via hidden states $\vec{h}_{1:T}$

$$p(\vec{y}_{1:T} | \vec{x}) = \sum_{\vec{h}_{1:T}} p(\vec{y}_{1:T}, \vec{h}_{1:T} | \vec{x}) =$$

$$= \sum_{\vec{h}_{1:T}} \prod_{t=1}^{T} p(\vec{y}_t | \vec{h}_t)\, p(\vec{h}_t | \vec{h}_{t-1}, \vec{y}_{t-1}, \vec{x})$$

Init. cond.: $p(\vec{h}_1 | \vec{h}_0, \vec{y}_0, \vec{x}) = \underbrace{f(\vec{h}_1 | \vec{x})}_{\text{initial hidden state distribution}}$

Output distributions:

$$p(\vec{y}_t | \vec{h}_t) = \text{Cat}(\vec{y}_t | \text{softmax}(\underbrace{W_{hy}}_{\text{hidden} \to \text{output weights}} \vec{h}_t + \underbrace{\vec{b}_y}_{\text{output biases}}))$$

or

$$p(\vec{y}_t | \vec{h}_t) = \mathcal{N}(\vec{y}_t | W_{hy} \vec{h}_t + \vec{b}_y, \sigma^2 \mathbb{I})$$

$p(\vec{h}_t | \vec{h}_{t-1}, \vec{y}_{t-1}, \vec{x})$ is computed deterministically:

$$\vec{h}_t = f(\vec{h}_{t-1}, \vec{y}_{t-1}, \vec{x}) \quad \text{or,}$$

more explicitly,

$$\vec{h}_t = g(W_1 \vec{x} + W_2 \vec{y}_{t-1} + W_3 \vec{h}_{t-1} + \vec{b}_h)$$

→ RNN can be viewed as a generalization of a standard Markov model

→ Training RNN: use labeled data as usual

→ Using RNN in a generative mode: sample from $p(\vec{y}_t | \vec{h}_t)$ to generate $\vec{\tilde{y}}_t$, compute $\vec{h}_{t+1} = f(\vec{h}_t, \vec{\tilde{y}}_t, \vec{x})$, sample from $p(\vec{y}_{t+1} | \vec{h}_{t+1})$, etc.

[Noise comes from the stochasticity in the outputs.]

Applications:

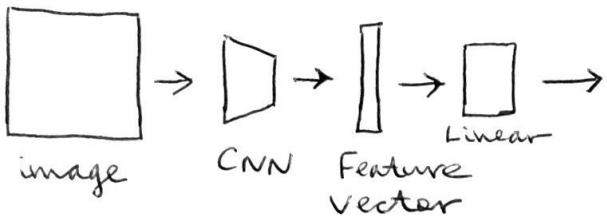
(1.) Language modeling: $\vec{x} = \vec{0}$, learn

$p(y_1, \ldots, y_T)$ = sentence

↑ tokens

(2.) Image captioning: $\vec{x}$ = image embedding produced by CNN

$p(y_1, \ldots, y_T)$ = image caption



Seq2vec models: $R^{TD} \to R^{C}$

TD: variable-size sequence of vectors

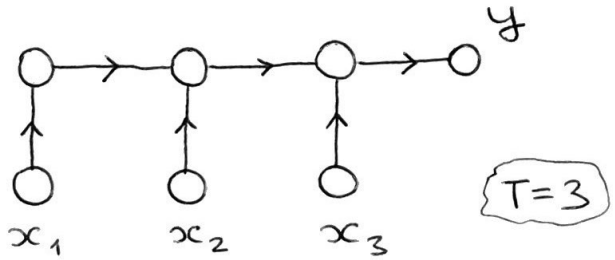
C: fixed-size output

Focus on the class-label output:

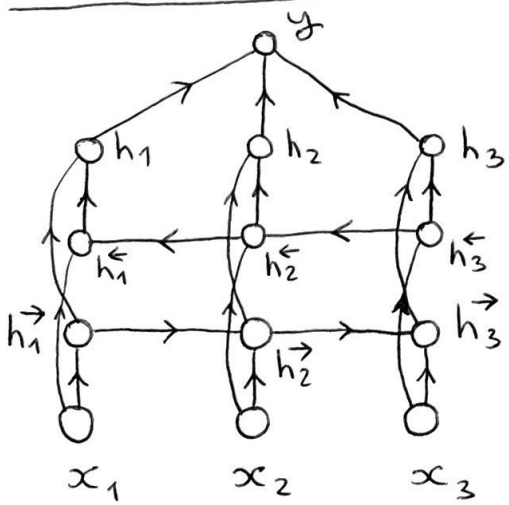
$$y \in \{1, \ldots, C\}$$

$$p(y \mid \vec{x}_{1:T}) = \text{Cat}(y \mid \text{softmax}(W_{hy} \vec{h}_T + \vec{b}_y))$$

T: last time index



Bi-directional RNN:



$$\vec{h}_t = \varphi(W_{xh}^{\rightarrow} \vec{x}_t + W_{hh}^{\rightarrow} \vec{h}_{t-1} + \vec{b}_h^{\rightarrow}),$$

$$\vec{h}_t = \varphi(W_{xh}^{\leftarrow} \vec{x}_t + W_{hh}^{\leftarrow} \vec{h}_{t-1} + \vec{b}_h^{\leftarrow}),$$

$$\vec{h} = \underbrace{[\vec{h}_t, \vec{h}_t]}_{\text{concatenation}}$$

$$\langle \vec{h} \rangle = \underbrace{\frac{1}{T} \sum_{t=1}^{T} \vec{h}_t}_{\text{average}}$$

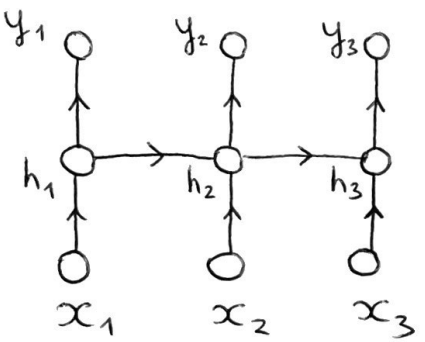
Finally, $p(y|\vec{x}_{1:T}) = \text{Cat}(y|\text{softmax}(W\langle\vec{h}\rangle + \vec{b}_y))$

Applications: text to sentiment

seq2seq models: $R^{TD} \to R^{T'C}$

E.g., $T = T'$ $\Rightarrow$ aligned input/output
$T \neq T'$ $\Rightarrow$ unaligned —"—/—"—

(1.) Aligned case: dense sequence labeling (one label per location)



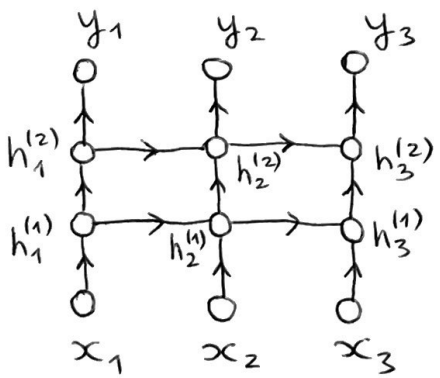
uni-directional RNN (bi-directional RNN possible as well)

$$p(\vec{y}_{1:T}|\vec{x}_{1:T}) = \sum_{\vec{h}_{1:T}} \prod_{t=1}^{T} p(\vec{y}_t|\vec{h}_t)$$

$$\vec{h}_t = f(\vec{h}_{t-1}, \vec{x}_t)$$

$\vec{h}_1 = f_0(\vec{x}_1)$ initial condition

Deep RNN: multiple layers of hidden nodes

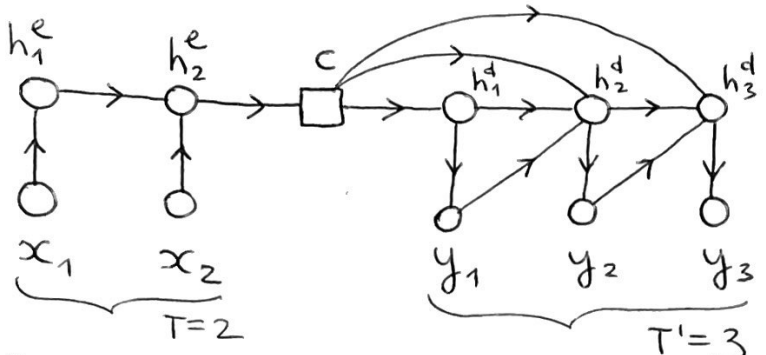


$$\vec{h}_t^{(\ell)} = f_\ell(W_{xh}^{\ell}\vec{h}_t^{(\ell-1)} + W_{hh}^{\ell}\vec{h}_{t-1}^{(\ell)} + \vec{b}_h^{(\ell)})$$

] layers

↑ hidden layer updates

Unaligned case: encoder-decoder architecture



Context vector $\vec{c} = f_e(\vec{x}_{1:T})$

RNN decoder $\vec{y}_{1:T} = f_d(\vec{c})$

Application: machine translation