CS 182 Lecture 17: Generative Models 7/16

Probabilistic models: → $p(x)$ → unsupervised learning (learn the distribution) aka "generative".
→ $p(y|x)$
Key uses of generative modeling: - rep. learning - same uses as language modeling.
- actually generating things

"language modelling" images is possible - Pixel RNNs. → Autoregressive generative models
→ how should we read pixels?
In paper: scanline order. ← issue: pixels right above are considered "far away"!

$$p(x) = \prod_{i=1}^{h^2} p(x_i | x_1, \ldots, x_{i-1})$$

256-way softmax
→ generate one color channel at a time.
↳ Diagonal BiLSTM.

practical: Pixel CNN (much faster, only uses masked neighborhood of pixel)

robust: Pixel Transformer
↳ all pixels are equally "close" for self-attention.
↳ to reduce computation, only compute attention for nearby pixels ("modified" masked attention)

Training:
1. Divide x into $x_1, \ldots, x_n$ dimensions
2. Discretize each x_i → k values
3. Model $p(x)$ using chain rule
4. Use sequence model to model $p(x)$.

Using:
↳ Sampling ($x_1, x_2, \ldots$, etc.)
↳ Completion (feed in known values)
↳ Representation: like ELMo / BERT.

conditional autoregressive models - generate something conditioned on another thing.
↳ pass in context thru encoder prior to "kicking off" sequence model.

General design for generative models:
image → nontrivial model → loss → same image. (RNN, pixel CNN/RNN, Transformer)

Autoencoder: train network to encode image, then decode same image.
image → encoder → hidden state → decoder → recreated image.
↳ force a "bottleneck" structure to compress initially (classic) → non-linear dimensionality reduction; Antiquated. Dense!
↳ force a sparse, conditioned hidden state → describe input w/ "attribute" set, compressed/structured version, most values are zero.
↳ corrupt the AE input with noise

widely used: denoising autoencoder (fill in the blanks)
↳ (sort of) analogous to BERT.
↳ simple to implement, but many ad-hoc choices.

image → encoder → HS → decoder → image
↓ (h)
sparsity

Early days: autoencoders used for layerwise pretraining
↳ train deep networks w/o end-to-end woes
↳ advent of ReLU, BN, etc. put a stop to this

combine a bunch of trained AEs, reduce dim, speed up backprop

Simple sparsity loss: $\sum_{j=1}^{D} |h_j|$ (L1) ← each dimension
↓ loss
other: lifetime sparsity, spike/slab models

Today: less widely used
↳ representation: VAEs, contrastive learning
↳ generation: GANs, VAEs, autoregressive models
↳ viable option for "quick and dirty" rep. learning
↳ Problem: sampling is hard.

Representing LVMs?
↳ easy: FCNN
↳ better: transpose convolutions (upscaling, unpooling)

Latent Variable Models: A sum of GMMs $p(x) = \sum_z p(x|z)\,p(z)$ ← categorical variable.
→ assume $p(x)$ is super complex
→ assume $p(z)$ is simple (Gaussian)
→ $p(x|z) = N(\mu_{nn}(z), \sigma_{nn}(z))$. ← offloaded
if x is discrete: discretized logistic is common

$$p(x) = \int p(x|z)\,p(z)\,dz$$

becomes continuous due to NN
simple, simple
NN functions
simple!

$$p(y|x) = \sum_z p(y|x,z)\,p(z)$$

To train LVMs:
→ model $p_\theta(x)$
→ data $D = \{x_1, \ldots, x_N\}$
→ max likelihood fit:
intractable... no closed-form solution
$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i \log p_\theta(x_i)$

Expected log-likelihood:

$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i E_{z \sim p(z|x_i)} [\log p_\theta(x_i, z)]$$

calculate through probabilistic inference.
"guess" most likely z given x_i

LVM in DL: basically, turn random numbers into valid samples
↳ VAEs, GANs, normalizing flows, etc.