Neural Autoregressive Distribution Estimation

Instructor: John Thickstun

Discussion Board: Available on Ed!

Zoom Link: Available on Canvas

Instructor Contact: thickstn@cs.washington.edu

Course Webpage: https://courses.cs.washington.edu/courses/cse599i/20au/

Autoregressive Modeling

- Factor the joint distribution into conditionals:

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t | x_{<t}).$$

- Learn the conditional distributions $\hat{p}(x_t | x_{<t})$.

- Linear autoregressive model:

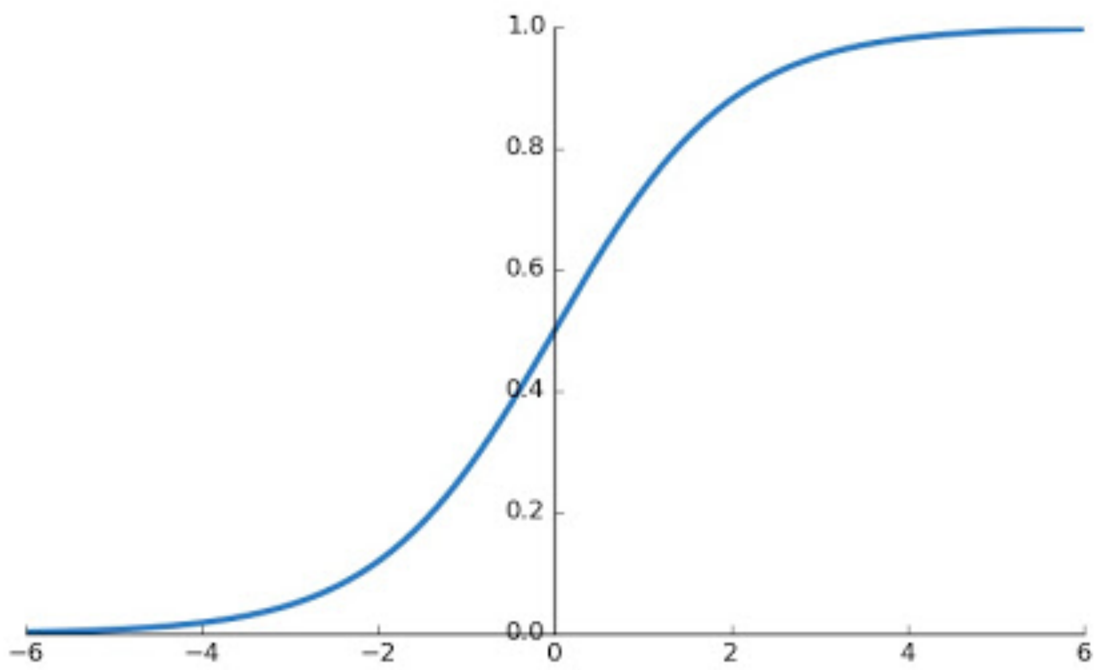
$$x_t \sim \mathcal{N}(f(x_{<t}), \sigma^2),$$
$$f(x_{<t}) = \rho(x_{t-1} - \mu) + \mu.$$

Binary Autoregressive Models

Definition 1. *Let $P \in \mathbb{R}^T$ and define $f_t : \mathbb{R}^{(t-1)} \to \mathbb{R}$ by $f_t(x_{<t}) = \langle P_{<t}, x_{<t} \rangle$. The* ***fully-visible sigmoid belief network*** *is given by*

$$x_t \sim \text{Bernoulli}\left(\text{sigmoid}(f_t(x_{<t}))\right),$$
$$x_1 \sim \text{Bernoulli}(P_0).$$

- Sigmoid function $\text{sigmoid}(u) = 1/(1+e^{-u})$
- A log-linear autoregressive model.



Masked Autoregressive Models

Definition 1. *Let $P \in \mathbb{R}^T$ and define $f_t : \mathbb{R}^{(t-1)} \to \mathbb{R}$ by $f_t(x_{<t}) = \langle P_{<t}, x_{<t} \rangle$. The* ***fully-visible sigmoid belief network*** *is given by*

$$x_t \sim \text{Bernoulli}\left(\text{sigmoid}(f_t(x_{<t}))\right),$$
$$x_1 \sim \text{Bernoulli}(P_0).$$

- Let $\mathbf{m} \in \{0,1\}^{T \times T}$ such that $\mathbf{m}_{t,s} = \mathbf{1}_{s<t}$.

- Then $f(x_{<t})$ can be equivalently defined by

$$f(x_{<t}) \equiv \langle P_{<t}, x_{<t} \rangle = \langle P, \mathbf{m}_t \odot x \rangle.$$

Discrete Autoregressive Models

Definition 2. *Let* $\boldsymbol{P} \in \mathbb{R}^{T \times d \times d}$ *and define* $f_t : \mathbb{R}^{(t-1) \times d} \to \mathbb{R}^d$ *by* $f_t(\boldsymbol{x}_{<t}) = \sum_{s<t} \boldsymbol{P}_s \boldsymbol{x}_s$.

The ***fully-visible softmax belief network*** *is given by*

$$\boldsymbol{w}_t \sim \text{Categorical}\left(\text{softmax}\left(f_t(\boldsymbol{x}_{<t})\right)\right),$$
$$\boldsymbol{w}_1 \sim \text{Categorical}(\boldsymbol{P}_{0,0}).$$

- Softmax function $\text{softmax} : \mathbb{R}^d \to \mathbb{R}^d$ defined by

$$\text{softmax}(\mathbf{u}) = \frac{e^{\mathbf{u}}}{\sum_{k=1}^d e^{\mathbf{u}_k}}.$$

- What is the input encoding $x_t \in \mathbb{R}^d$ of the discrete inputs $w_t \in \mathcal{V}$?

Input Encoding

- Discrete sequences $\mathbf{w} \in \mathcal{V}^T$.
- Tokens $w_t \in \mathcal{V}$ for some finite vocabulary $\mathcal{V}$, where $|\mathcal{V}| = d$.
- Need to encode tokens w as vectors x in a Euclidean space.
- Two popular options:
    1. One-hot encoding: $x \in \mathbb{R}^d$ where $x_k = \mathbf{1}_{k=w}$.
    2. Word Embeddings (word2vec, Glove, etc.): $x_k \in \mathbb{R}^k$ $(k < d)$.

Training a FVSBN

Definition 2. *Let $\boldsymbol{P} \in \mathbb{R}^{T\times d\times d}$ and define $f_t : \mathbb{R}^{(t-1)\times d} \to \mathbb{R}^d$ by $f_t(\boldsymbol{x}_{<t}) = \sum_{s<t} \boldsymbol{P}_s \boldsymbol{x}_s$.*

*The **fully-visible softmax belief network** is given by*

$$\boldsymbol{w}_t \sim \mathrm{Categorical}\left(\mathrm{softmax}\left(f\left(\boldsymbol{x}_{<t}\right)\right)\right),$$
$$\boldsymbol{w}_1 \sim \mathrm{Categorical}(\boldsymbol{P}_{0,0}).$$

- Maximize the likelihood! Cross-entropy minimization:

$$\frac{1}{n}\sum_{i=1}^{n} -\log p(\mathbf{w}^i) = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{T} -\log p(\mathbf{w}_t^i|\mathbf{w}_{<t}^i).$$

Neural Parameterization (NADE)

- Fully-Visible Softmax Belief Network:

$$f_t(\mathbf{x}_{<t}) = \sum_{s<t} \mathbf{P}_s \mathbf{x}_s, \qquad \mathbf{P} \in \mathbb{R}^{T\times d\times d}.$$

- Neural Autoregressive Distribution Estimation (NADE):

$$\mathbf{h}_t = \text{sigmoid}\left(\mathbf{c}_t + \sum_{s<t} \mathbf{V}_s \mathbf{x}_s\right), \qquad \mathbf{V} \in \mathbb{R}^{T\times k\times d}, \mathbf{c} \in \mathbb{R}^{T\times k},$$

$$f_t(\mathbf{x}_{<t}) = \mathbf{b}_t + \sum_{s\leq t} \mathbf{W}_s \mathbf{h}_s, \qquad \mathbf{W} \in \mathbb{R}^{T\times d\times k}, \mathbf{b} \in \mathbb{R}^{T\times d}.$$

Training a NADE

- Maximize the likelihood:

$$\frac{1}{n}\sum_{i=1}^{n} -\log p_\theta(\mathbf{w}^i) = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{T} -\log p_\theta(w_t^i|w_{<t}^i).$$

- Optimize with SGD (choose a random j, t):

$$\theta^{(i)} = \theta^{(i-1)} + \eta\nabla_\theta \log p_\theta(w_t^j|w_{<t}^j).$$

- Consider adding momentum (e.g. NAG) and adaptivity (Adam).

Back-Propagation Through Time

- NADE models see a limited window of context/history (order p).

- Mini-batch SGD over time steps (one call to the model):

$$\begin{aligned}\theta^{(i)} &= \theta^{(i-1)} + \eta\nabla_\theta \log p_\theta(w_t^j, \ldots, w_{t+p}^j) \\ &= \theta^{(i-1)}) + \eta\nabla_\theta \sum_{s=1}^{p} \log p_\theta(w_{t+s}^j | w_t^j, \ldots, w_{t+s-1}^j).\end{aligned}$$

- Also mini-batch over k data points (k calls to the model):

$$\theta^{(i)} = \theta^{(i-1)}) + \eta\nabla_\theta \sum_{j=1}^{k}\sum_{s=1}^{p} \log p_\theta(w_{t+s}^j | w_t^j, \ldots, w_{t+s-1}^j).$$

5-Minute Break

BPTT in NLP: The Dirty Details

- Suppose we have an NLP corpus of documents (e.g. Wikitext-2).
- Concatenate all the documents: one long sequence of length T.
- Chunk the sequence up into segments of length p.
- Treat the corpus as $n = T/p$ data points, each of length p.

Weight-Sharing

- Neural Autoregressive Distribution Estimator (order p):

$$\mathbf{h}_t = \text{sigmoid}\left(\mathbf{c}_t + \sum_{s<t} \mathbf{V}_s \mathbf{x}_s\right), \qquad \mathbf{V} \in \mathbb{R}^{p\times k\times d}, \mathbf{c} \in \mathbb{R}^{p\times k},$$

$$f_t(\mathbf{x}_{<t}) = \mathbf{b}_t + \sum_{s\leq t} \mathbf{W}_s \mathbf{h}_s, \qquad \mathbf{W} \in \mathbb{R}^{p\times d\times k}, \mathbf{b} \in \mathbb{R}^{p\times d}.$$

- Set $\mathbf{V}_s = (\mathbf{W}_s)^T$.

Deep NADE

- Multiple hidden layers (compare to a deep fully-connected network)

$$\mathbf{h}_{0,t} = \mathrm{ReLU}\left(\mathbf{c}_{0,t} + \sum_{s<t} (\mathbf{W}_s)^T \mathbf{x}_s\right), \qquad \mathbf{W} \in \mathbb{R}^{p\times d\times k}, \mathbf{c}_0 \in \mathbb{R}^{p\times k},$$

$$\mathbf{h}_{\ell,t} = \mathrm{ReLU}\left(\mathbf{c}_{\ell,t} + \sum_{s\leq t} \mathbf{V}_{\ell,s}\mathbf{h}_{\ell-1,s}\right), \qquad \mathbf{V} \in \mathbb{R}^{(L-1)\times p\times k\times k}, \mathbf{c} \in \mathbb{R}^{(L-1)\times p\times k},$$

$$f_{\theta,t}(\mathbf{x}_{<t}) = \mathbf{b}_t + \sum_{s\leq t} \mathbf{W}_s \mathbf{h}_{L,s}, \qquad \mathbf{b} \in \mathbb{R}^{p\times d}.$$

A Recurrent NADE

- Parameter counts in NADE scale with the autoregressive order p.
- We can share weights using a recursively defined model (RNN):

$$
\begin{aligned}
\mathbf{h}_0 &= \mathbf{W}_{\text{init}}, & \mathbf{W}_{\text{init}} &\in \mathbb{R}^k, \\
\mathbf{h}_t &= \text{sigmoid}(\mathbf{c} + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t), & \mathbf{W}_h &\in \mathbb{R}^{k \times k}, \mathbf{W}_x \in \mathbb{R}^{k \times d}, \mathbf{c} \in \mathbb{R}^k, \\
f_t(\mathbf{x}_{<t}) &= \mathbf{b} + \mathbf{W}_o \mathbf{h}_{t-1}, & \mathbf{W}_o &\in \mathbb{R}^{d \times k}, \mathbf{b} \in \mathbb{R}^d.
\end{aligned}
$$

- No dependency on p! Can set $\mathbf{W}_x = \mathbf{W}_o^T$ for further sharing.

Hyper-Parameter Selection

- In general: make the model very wide and very deep. Use large p.
- Don't regularize with model size! Regularize by (in order of effectiveness):
    1. Getting more training data.
    2. Dataset augmentation (e.g. input/embedding dropout)
    3. Early stopping/L2 regularization/dropout
- Use validation data, train multiple models (pay the Amazon tax).
- If your model is constrained by GPU memory, you are on the right track.

Exposure Bias

- Factor the joint distribution into conditionals:

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t | x_{<t}).$$

- Learn the conditional distributions $\hat{p}(x_t | x_{<t})$.

- Iteratively sample $\hat{x}_t \sim \hat{p}(\cdot | \hat{x}_{<t})$ to construct

- The data we condition on to predict has the wrong distribution!

Logistics

- Homework 1 is out on the website.
- If you downloaded it early, get a fresh copy: I made some corrections.
- You could start looking at the code for Problem 8 (except transformers.py)