Maximum Likelihood Learning

Volodymyr Kuleshov

Cornell Tech

Lecture 4

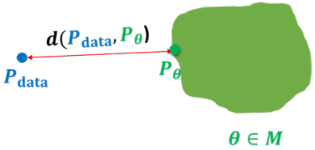
Announcements

- Assignment is up and is due 2 weeks from now.
- Please ask any questions on Piazza.

Learning a generative model

- We are given a training set of examples, e.g., images of dogs

$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



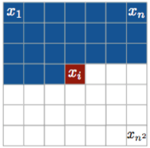
Model family

- We want to learn a probability distribution $p(x)$ over images x such that
  - **Generation:** If we sample $x_{new} \sim p(x)$, x_{new} should look like a dog (*sampling*)
  - **Density estimation:** $p(x)$ should be high if x looks like a dog, and low otherwise (*anomaly detection*)
  - **Unsupervised representation learning:** We should be able to learn what these images have in common, e.g., ears, tail, etc. (*features*)
- First question: how to represent $p_\theta(x)$. Second question: **how to learn it**.

Lecture Outline

1. Wrap-up of modern autoregressive models
    - PixelRNN, PixelCNN
    - WaveNet
2. Learning as density estimation
3. Density estimation as optimization
    - Monte Carlo estimation
    - Gradient descent
4. Statistical issues and the bias/variance tradeoff

Pixel RNN (Oord et al., 2016)



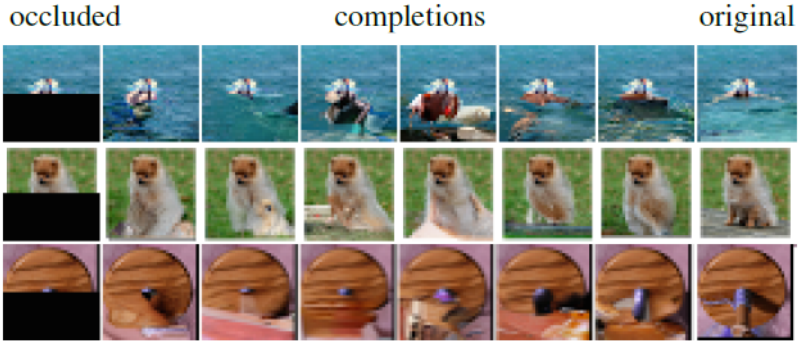
1. Model images pixel by pixel using raster scan order
2. Each pixel conditional $p(x_t \mid x_{1:t-1})$ needs to specify 3 colors

$$p(x_t \mid x_{1:t-1}) = p(x_t^{red} \mid x_{1:t-1})p(x_t^{green} \mid x_{1:t-1}, x_t^{red})p(x_t^{blue} \mid x_{1:t-1}, x_t^{red}, x_t^{green})$$

   and each conditional is a categorical random variable with 256 possible values
3. Conditionals modeled using RNN variants. LSTMs + masking (like MADE)

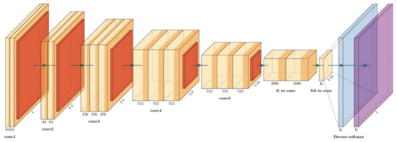
$$P(x_t \mid x_{1:t-1}) = P(x_t \mid h_{t-1}, x_{t-1}) \qquad h_t = f(h_{t-1}, x_{t-1})$$

Pixel RNN



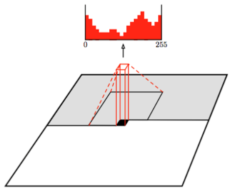
Results on downsampled ImageNet. Very slow: sequential likelihood evaluation.

Convolutional Architectures



Convolutions are natural for image data and easy to parallelize on modern hardware.

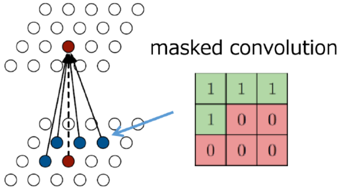
PixelCNN (Oord et al., 2016)

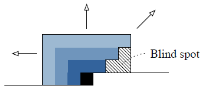


Idea: Use convolutional architecture to predict next pixel given context (a neighborhood of pixels).

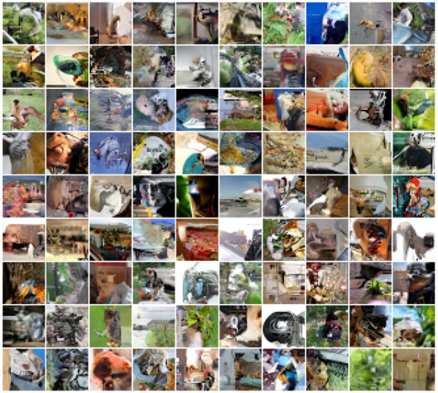
$$P(x_t \mid x_{1:t-1}) = P(x_t \mid x_{\text{neighborhood}}) = \text{conv}(x_{\text{neighborhood}})$$

Challenge: Has to be autoregressive. **Soln:** Mask future pixels in conv filter.





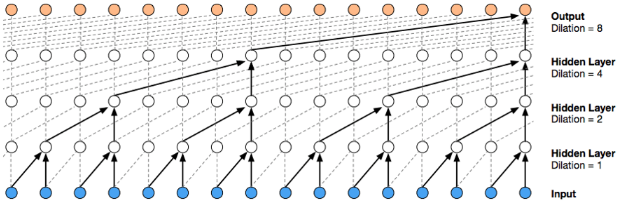
PixelCNN



Samples from the model trained on Imagenet (32×32 pixels). Similar performance to PixelRNN, but much faster.

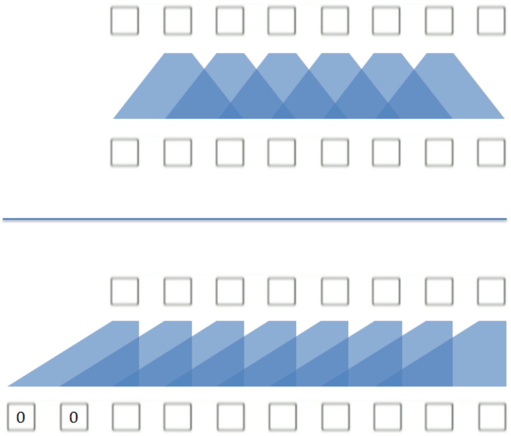
WaveNet (Oord et al., 2016)

State of the art model for speech:



Causal Convolutions

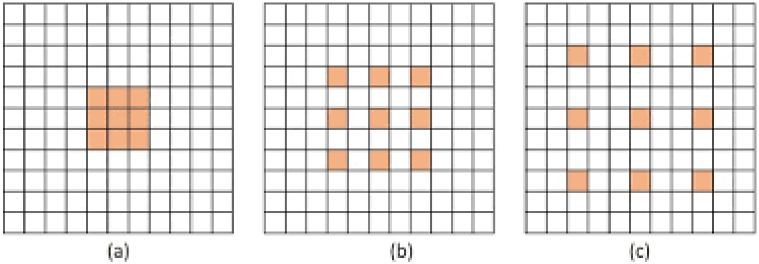
Regular convolutions (top) use filters that touch symmetrical input region.



Causal convolutions (bottom) mask part of filter that touches the future.

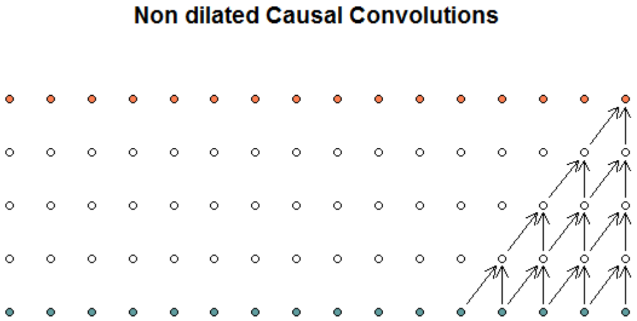
Dilated Convolutions

Dilated convolutions introduce "holes" into the convolution filters:



Dilated Convolutions

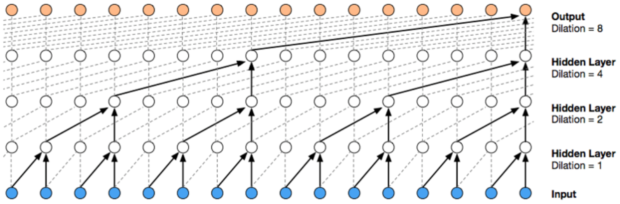
Normal convolutions in Wavenet would look like this:



Dilated convolutions increase the receptive field: kernel only touches the signal at every 2^d entries.

WaveNet (Oord et al., 2016)

State of the art model for speech:



Dilated convolutions increase the receptive field: kernel only touches the signal at every 2^d entries.

Summary of Autoregressive Models

- Easy to sample from
  1. Sample $\overline{x}_0 \sim p(x_0)$
  2. Sample $\overline{x}_1 \sim p(x_1 \mid x_0 = \overline{x}_0)$
  3. $\cdots$
- Easy to compute probability $p(x = \overline{x})$
  1. Compute $p(x_0 = \overline{x}_0)$
  2. Compute $p(x_1 = \overline{x}_1 \mid x_0 = \overline{x}_0)$
  3. Multiply together (sum their logarithms)
  4. $\cdots$
  5. Ideally, can compute all these terms in parallel for fast training
- We can often construct autoregressive models by adapting feed-forward and discriminative models using ideas such as masking.
- Next: learning

Lecture Outline

1. Wrap-up of modern autoregressive models
    - PixelRNN, PixelCNN
    - WaveNet
2. **Learning as density estimation**
3. Density estimation as optimization
    - Monte Carlo estimation
    - Gradient descent
4. Statistical issues and the bias/variance tradeoff

Setting

- Lets assume that the domain is governed by some underlying distribution P_{data}
- We are given a dataset $\mathcal{D}$ of m samples from P_{data}
  - Each sample is an assignment of values to (a subset of) the variables, e.g., $(X_{\mathrm{bank}} = 1, X_{\mathrm{dollar}} = 0, ..., Y = 1)$ or pixel intensities.
- The standard assumption is that the data instances are **independent and identically distributed (IID)**
- We are also given a family of models $\mathcal{M}$, and our task is to learn some "good" model $\hat{\mathcal{M}} \in \mathcal{M}$ (i.e., in this family) that defines a distribution $p_{\hat{\mathcal{M}}}$
  - For example, all Bayes nets with a given graph structure, for all possible choices of the CPD tables
  - For example, a FVSBN for all possible choices of the logistic regression parameters. $\mathcal{M} = \{P_\theta, \theta \in \Theta\}$, $\theta =$ concatenation of all logistic regression coefficients

Goal of learning

- The goal of learning is to return a model $\hat{\mathcal{M}}$ that precisely captures the distribution P_{data} from which our data was sampled
- This is in general not achievable because of
    - limited data only provides a rough approximation of the true underlying distribution
    - computational reasons
- Example. Suppose we represent each image with a vector X of 784 binary variables (black vs. white pixel). How many possible states (= possible images) in the model? $2^{784} \approx 10^{236}$. Even 10^7 training examples provide *extremely* sparse coverage!
- We want to select $\hat{\mathcal{M}}$ to construct the "best" approximation to the underlying distribution P_{data}
- What is "best"?

What is "best"?

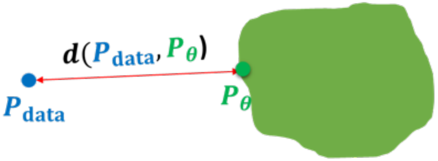
This depends on what we want to do

1. Density estimation: we are interested in the full distribution (so later we can compute whatever conditional probabilities we want)
2. Specific prediction tasks: we are using the distribution to make a prediction
    - Is this email spam or not?
    - Predict next frame in a video
3. Structure or knowledge discovery: we are interested in the model itself
    - How do some genes interact with each other?
    - What causes cancer?

Learning as density estimation

- We want to learn the full distribution so that later we can answer *any* probabilistic inference query
- In this setting we can view the learning problem as **density estimation**
- We want to construct P_θ as "close" as possible to P_{data} (recall we assume we are given a dataset $\mathcal{D}$ of samples from P_{data})

$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



$\theta \in M$

Model family

- How do we evaluate "closeness"?

KL-divergence

- How should we measure distance between distributions?
- The **Kullback-Leibler divergence** (KL-divergence) between two distributions p and q is defined as

$$D(p\|q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

- $D(p \,\|\, q) \geq 0$ for all p, q, with equality if and only if $p = q$. Proof:

$$\mathbf{E}_{\mathbf{x}\sim p}\left[-\log \frac{q(\mathbf{x})}{p(\mathbf{x})}\right] \geq -\log\left(\mathbf{E}_{\mathbf{x}\sim p}\left[\frac{q(\mathbf{x})}{p(\mathbf{x})}\right]\right) = -\log\left(\sum_{\mathbf{x}} p(\mathbf{x})\frac{q(\mathbf{x})}{p(\mathbf{x})}\right) = 0$$

- Notice that KL-divergence is **asymmetric**, i.e., $D(p\|q) \neq D(q\|p)$
- Measures the expected number of extra bits required to describe *samples from* $p(\mathbf{x})$ using a code based on q instead of p

Learning as density estimation

- We want to learn the full distribution so that later we can answer *any* probabilistic inference query
- In this setting we can view the learning problem as **density estimation**
- We want to construct P_θ as "close" as possible to P_{data} (recall we assume we are given a dataset $\mathcal{D}$ of samples from P_{data})
- How do we evaluate "closeness"?
- **KL-divergence** is one possibility:

$$\mathbf{D}(P_{\mathrm{data}}||P_\theta) = \mathbf{E}_{\mathbf{x}\sim P_{\mathrm{data}}}\left[\log\left(\frac{P_{\mathrm{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}\right)\right] = \sum_{\mathbf{x}} P_{\mathrm{data}}(\mathbf{x})\log\frac{P_{\mathrm{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}$$

- $\mathbf{D}(P_{\mathrm{data}}||P_\theta) = 0$ iff the two distributions are the same.
- It measures the "compression loss" (in bits) of using P_θ instead of P_{data}.

Expected log-likelihood

- We can simplify this somewhat:

$$\begin{aligned}\mathbf{D}(P_{\mathrm{data}}||P_\theta) &= \mathbf{E}_{\mathbf{x}\sim P_{\mathrm{data}}}\left[\log\left(\frac{P_{\mathrm{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}\right)\right] \\ &= \mathbf{E}_{\mathbf{x}\sim P_{\mathrm{data}}}\left[\log P_{\mathrm{data}}(\mathbf{x})\right] - \mathbf{E}_{\mathbf{x}\sim P_{\mathrm{data}}}\left[\log P_\theta(\mathbf{x})\right]\end{aligned}$$

- The first term does not depend on P_θ.
- Then, *minimizing* KL divergence is equivalent to *maximizing* the **expected log-likelihood**

$$\arg\min_{P_\theta} \mathbf{D}(P_{\mathrm{data}}||P_\theta) = \arg\min_{P_\theta} -\mathbf{E}_{\mathbf{x}\sim P_{\mathrm{data}}}\left[\log P_\theta(\mathbf{x})\right] = \arg\max_{P_\theta} \mathbf{E}_{\mathbf{x}\sim P_{\mathrm{data}}}\left[\log P_\theta(\mathbf{x})\right]$$

  - Asks that P_θ assign high probability to instances sampled from P_{data}, so as to reflect the true distribution
  - Because of log, samples $\mathbf{x}$ where $P_\theta(\mathbf{x}) \approx 0$ weigh heavily in objective
- Although we can now compare models, since we are ignoring $\mathbf{H}(P_{\mathrm{data}})$, we don't know how close we are to the optimum
- Problem: In general we do not know P_{data}.

Monte Carlo Estimation

1. Express the quantity of interest as the expected value of a random variable.

$$E_{x\sim P}[g(x)] = \sum_x g(x)P(x)$$

2. Generate T samples $\mathbf{x}^1, \ldots, \mathbf{x}^T$ from the distribution P with respect to which the expectation was taken.
3. Estimate the expected value from the samples using:

$$\hat{g}(\mathbf{x}^1, \cdots, \mathbf{x}^T) \triangleq \frac{1}{T}\sum_{t=1}^{T} g(\mathbf{x}^t)$$

where $\mathbf{x}^1, \ldots, \mathbf{x}^T$ are independent samples from P. Note: $\hat{g}$ is a random variable. Why?

Properties of the Monte Carlo Estimate

- **Unbiased:**

$$E_P[\hat{g}] = E_P[g(x)]$$

- **Convergence:** By law of large numbers

$$\hat{g} = \frac{1}{T}\sum_{t=1}^{T} g(x^t) \rightarrow E_P[g(x)] \text{ for } T \rightarrow \infty$$

- **Variance:**

$$V_P[\hat{g}] = V_P\left[\frac{1}{T}\sum_{t=1}^{T} g(x^t)\right] = \frac{V_P[g(x)]}{T}$$

Thus, variance of the estimator can be reduced by increasing the number of samples.

Maximum likelihood estimation

- Approximate the expected log-likelihood

$$\mathbf{E}_{\mathbf{x}\sim P_{\mathrm{data}}}\left[\log P_\theta(\mathbf{x})\right]$$

with the *empirical log-likelihood*:

$$\mathbf{E}_{\mathcal{D}}\left[\log P_\theta(\mathbf{x})\right] = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

- **Maximum likelihood learning** is then:

$$\max_{P_\theta} \; \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

- Equivalently, maximize probability of the data under model $P_\theta(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}) = \prod_{\mathbf{x}\in\mathcal{D}} P_\theta(\mathbf{x})$

Example

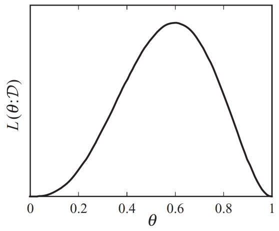
Single variable example: A biased coin

- Two outcomes: *heads* (H) and *tails* (T)
- Data set: Tosses of the biased coin, e.g., $\mathcal{D} = \{H, H, T, H, T\}$
- Assumption: the process is controlled by a probability distribution $P_{\rm data}(x)$ where $x \in \{H, T\}$
- Class of models $\mathcal{M}$: Bernoulli distributions over $x \in \{H, T\}$.
- Example learning task: How should we choose $P_\theta(x)$ from $\mathcal{M}$ if 60 out of 100 tosses are heads in $\mathcal{D}$?

MLE scoring for the coin example

We represent our model: $P_\theta(x = H) = \theta$ and $P_\theta(x = T) = 1 - \theta$

- Example data: $\mathcal{D} = \{H, H, T, H, T\}$
- Likelihood of data $= \prod_i P_\theta(x_i) = \theta \cdot \theta \cdot (1-\theta) \cdot \theta \cdot (1-\theta)$



- Optimize for θ which makes $\mathcal{D}$ most likely. What is the solution in this case?

MLE scoring for the coin example: Analytical derivation

Distribution: $P_\theta(x = H) = \theta$ and $P_\theta(x = T) = 1 - \theta$

- More generally, log-likelihood function

$$\begin{aligned} L(\theta) &= \theta^{\#heads} \cdot (1-\theta)^{\#tails} \\ \log L(\theta) &= \log(\theta^{\#heads} \cdot (1-\theta)^{\#tails}) \\ &= \#heads \cdot \log(\theta) + \#tails \cdot \log(1-\theta) \end{aligned}$$

- MLE Goal: Find $\theta^* \in [0,1]$ such that $\log L(\theta^*)$ is maximum.
- Differentiate the log-likelihood function with respect to θ and set the derivative to zero. We get:

$$\theta^* = \frac{\#heads}{\#heads + \#tails}$$

Extending the MLE principle to a Bayesian network

Given an autoregressive model with n variables and factorization

$$P_\theta(\mathbf{x}) = \prod_{i=1}^{n} p(x_i | pa(x_i); \theta_i)$$

Training data $\mathcal{D} = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$. Maximum likelihood estimate of the parameters?

- Decomposition of Likelihood function

$$L(\theta, \mathcal{D}) \quad = \quad \sum_{j=1}^{m} \log P_\theta(\mathbf{x}^{(j)}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i)$$

- Goal : maximize $\arg\max_\theta L(\theta, \mathcal{D}) = \arg\max_\theta \log L(\theta, \mathcal{D})$
- Each term is a normal conditional log-likelihood and can be optimized independently.
- For classical Bayes Net, conditionals are exponential families and have closed form solutions.

Extending the MLE principle to a Neural model

Given an autoregressive model with n variables and factorization

$$P_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_{\text{neural}}(x_i | pa(x_i); \theta_i)$$

Training data $\mathcal{D} = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$. Maximum likelihood estimate of the parameters?

- Decomposition of Likelihood function

$$L(\theta, \mathcal{D}) \quad = \quad \prod_{j=1}^{m} P_\theta(\mathbf{x}^{(j)}) = \prod_{j=1}^{m} \prod_{i=1}^{n} p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i)$$

- Goal : maximize $\arg\max_\theta L(\theta, \mathcal{D}) = \arg\max_\theta \log L(\theta, \mathcal{D})$
- We no longer have a closed form solution!

MLE Learning: Gradient Descent

$$L(\theta, \mathcal{D}) = \prod_{j=1}^{m} P_\theta(\mathbf{x}^{(j)}) = \prod_{j=1}^{m}\prod_{i=1}^{n} p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i)$$

Goal : maximize $\arg\max_\theta L(\theta, \mathcal{D}) = \arg\max_\theta \log L(\theta, \mathcal{D})$

$$\ell(\theta) = \log L(\theta, \mathcal{D}) = \sum_{j=1}^{m}\sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i)$$

1. Initialize θ^0 at random
2. Compute $\nabla_\theta \ell(\theta)$ (by back propagation)
3. $\theta^{t+1} = \theta^t + \alpha_t \nabla_\theta \ell(\theta)$

Non-convex optimization problem, but often works well in practice

MLE Learning: Stochastic Gradient Descent

$$\ell(\theta) = \log L(\theta, \mathcal{D}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i)$$

1. Initialize θ^0 at random
2. Compute $\nabla_\theta \ell(\theta)$ (by back propagation)
3. $\theta^{t+1} = \theta^t + \alpha_t \nabla_\theta \ell(\theta)$

$$\nabla_\theta \ell(\theta) = \sum_{j=1}^{m} \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i)$$

What if $m = |\mathcal{D}|$ is huge?

$$\begin{aligned} \nabla_\theta \ell(\theta) &= m \sum_{j=1}^{m} \frac{1}{m} \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i) \\ &= m E_{x^{(j)} \sim \mathcal{D}} \left[\sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i) \right] \end{aligned}$$

Monte Carlo: Sample $x^{(j)} \sim \mathcal{D}; \nabla_\theta \ell(\theta) \approx m \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i)$

Parallelization in Autoregressive Models

Out objective function is:

$$\ell(\theta) = \log L(\theta, \mathcal{D}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i)$$

If we use a recurrent neural network model, each term has the form

$$P(x_t \mid x_{1:t-1}) = P(x_t \mid h_{t-1}, x_{t-1}) \qquad h_t = f(h_{t-1}, x_{t-1}).$$

Before we can evaluate and/or compute gradient, we need to process each term sequentially.
This is why we want feed-forward models like NADE, MADE, or PixelCNN:

$$P(x_t \mid x_{1:t-1}) = P(x_t \mid x_{\text{neighborhood}}) = \text{conv}(x_{\text{neighborhood}})$$

Lecture Outline

1. Wrap-up of modern autoregressive models
    - PixelRNN, PixelCNN
    - WaveNet
2. Learning as density estimation
3. Density estimation as optimization
    - Monte Carlo estimation
    - Gradient descent
4. **Statistical issues and the bias/variance tradeoff**

Empirical Risk and Overfitting

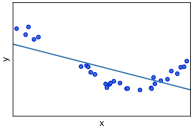
- Empirical risk minimization can easily **overfit** the data
  - Extreme example: The data is the model (remember all training data).
- Generalization: the data is a sample, usually there is vast amount of samples that you have never seen. Your model should generalize well to these "never-seen" samples.
- Thus, we typically restrict the **hypothesis space** of distributions that we search over

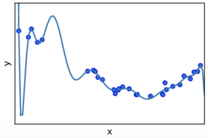
Bias-Variance trade off

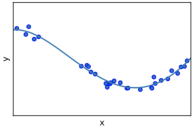
- If the hypothesis space is very limited, it might not be able to represent P_{data}, even with unlimited data
  - This type of limitation is called **bias**, as the learning is limited on how close it can approximate the target distribution
- If we select a highly expressive hypothesis class, we might represent better the data
  - When we have small amount of data, multiple models can fit well, or even better than the true model. Moreover, small perturbations on $\mathcal{D}$ will result in very different estimates
  - This limitation is call the **variance**.

Bias-Variance trade off

- There is an inherent **bias-variance trade off** when selecting the hypothesis class. Error in learning due to both things: bias and variance.
- Hypothesis space: linear relationship
    - Does it fit well? Underfits
- Hypothesis space: high degree polynomial
    - Overfits
- Hypothesis space: low degree polynomial
    - Right tradeoff







How to avoid overfitting?

- Hard constraints, e.g. by selecting a less expressive hypothesis class:
  - Bayesian networks with at most d parents
  - Smaller neural networks with less parameters
  - Weight sharing
- Soft preference for "simpler" models: **Occam Razor**.
- Augment the objective function with **regularization**:

$$objective(\mathbf{x}, \mathcal{M}) = loss(\mathbf{x}, \mathcal{M}) + R(\mathcal{M})$$

- Evaluate generalization performance on a held-out validation set. Log-likelihood should be similar on both training and validation set if there is no overfitting (as in discriminative modeling!)

Conditional generative models

- Suppose we want to generate a set of variables $\mathbf{Y}$ given some others $\mathbf{X}$, e.g., text to speech
- We concentrate on modeling $p(\mathbf{Y}|\mathbf{X})$, and use a **conditional** loss function
$$-\log P_\theta(\mathbf{y} \mid \mathbf{x}).$$
- Since the loss function only depends on $P_\theta(\mathbf{y} \mid \mathbf{x})$, suffices to estimate the conditional distribution, not the joint

Input : image

$\mapsto$

Brown horse in grass field

Output: caption

Recap

- For autoregressive models, it is easy to compute $p_\theta(x)$
    - Ideally, evaluate in parallel each conditional $\log p_{\text{neural}}(x_i^{(j)} | pa(x_i)^{(j)}; \theta_i)$. Not like RNNs.
- Natural to train them via maximum likelihood
- Higher log-likelihood doesn't necessarily mean better looking samples
- Other ways of measuring similarity are possible (Generative Adversarial Networks, GANs)