Maximum Likelihood Learning

Volodymyr Kuleshov

Cornell Tech

Lecture 4

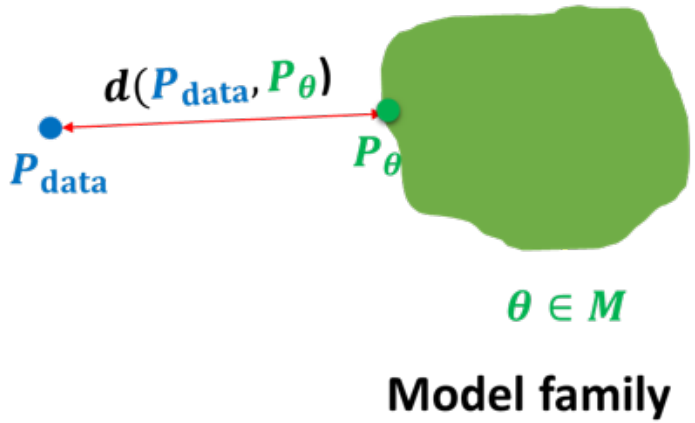
Announcements

- Assignment 1 was released on Monday and is due on 2/13.
  - Please submit the assignment on Gradescope.
  - You can submit in teams of 1-2.
- The project proposal will be due on 2/22.
- I will release next week a sign-up sheet for student presentations.

The Task of Generative Modeling

Suppose we are given a training set of examples, e.g., images of dogs

$$x_i \sim P_{\text{data}}$$
$$i = 1, 2, \ldots, n$$



Recall: A generative modeling algorithm has three components:

- **Model Family:** How do we parameterize a probability distribution over the data? What is the set of distributions we want to try to learn?
- **Learning Objective:** How do we quantify if a particular model in our family is a good fit?
- **Optimization:** How do we optimize the objective?

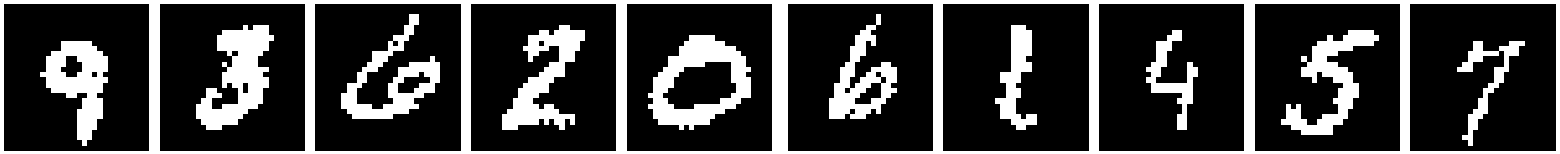
Step 1: How to represent $p(x)$ (last lecture). Step 2: how to learn it (today).

Lecture Outline

1. Learning via Maximum Likelihood
    - The Learning Problem
    - The KL-Divergence as a Learning Objective
    - From KL-Divergence to Maximum Likelihood
2. Maximizing the Likelihood from Data
    - Monte Carlo Estimation
    - Gradient Descent
3. Statistical Issues and the Bias/Variance Tradeoff

Running Example: A Generative Model for MNIST

Suppose we are given a dataset $\mathcal{D}$ of handwritten digits (binarized MNIST)



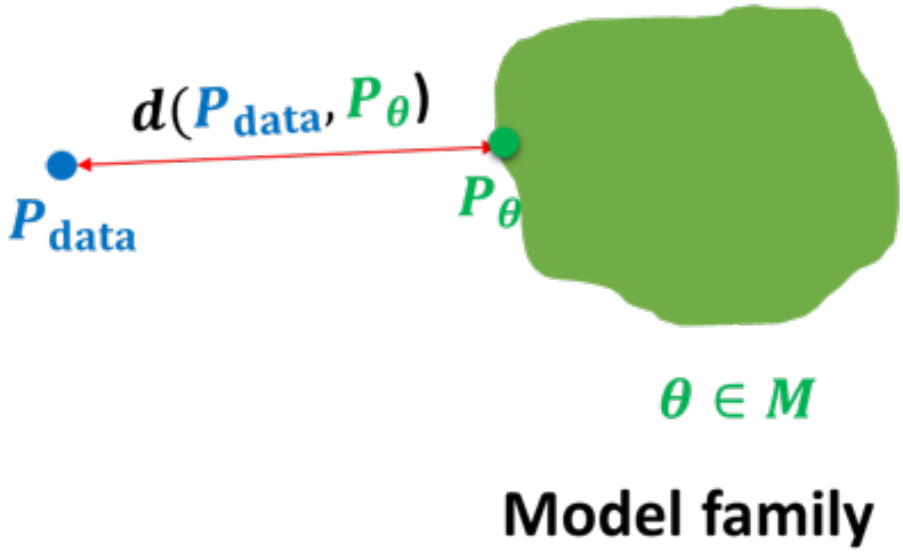
- Each image has $n = 28 \times 28 = 784$ pixels. Each pixel can either be black (0) or white (1).
- **Our Goal:** Learn a probability distribution $p(x) = p(x_1, \cdots, x_{784})$ over $x \in \{0, 1\}^{784}$ such that when $x \sim p(x)$, x looks like a digit
- Two step process:
  1. Parameterize a model family $\{p_\theta(x), \theta \in \Theta\}$ [This lecture]
  2. Search for model parameters θ based on training data $\mathcal{D}$ [Next lecture]

Formalizing Assumptions on the Data

We start by making some standard assumptions on the data.

- We assume that the data comes from a probability distribution P_{data}
- Our dataset $\mathcal{D}$ consists of m samples from P_{data}
  - Each sample is an assignment of values to (a subset of) the variables, e.g., $(X_{\text{bank}} = 1, X_{\text{dollar}} = 0, ..., Y = 1)$ or pixel intensities.

$$x_i \sim P_{\text{data}}$$
$$i = 1, 2, \ldots, n$$

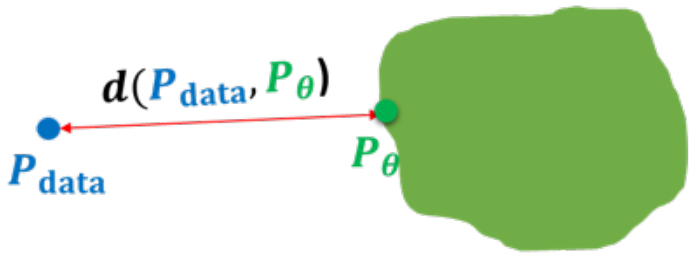


Formalizing the Task of Learning

Next, we want to define what "learning" means:

- We are also given a family of models $\mathcal{M}$, and our task is to learn some "good" model $P_\theta \in \mathcal{M}$ (i.e., in this family) that defines a distribution P_θ
  - 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
- The goal of learning is to return a model P_θ that precisely captures the distribution P_{data} from which our data was sampled

$x_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



What Defines a Good Model?

We may be interested in multiple tasks:

1. **Density Estimation**: we are interested in the full distribution (for anomaly detection, missing value imputation, etc.)
2. **Prediction**: we are using the distribution for supervised learning
   - Is this email spam or not?
   - Predict next frame in a video
3. **Structure discovery**: we are interested in the model itself
   - How do some genes interact with each other?
   - What causes cancer?
4. **Generation**: sample new data points that look "good".

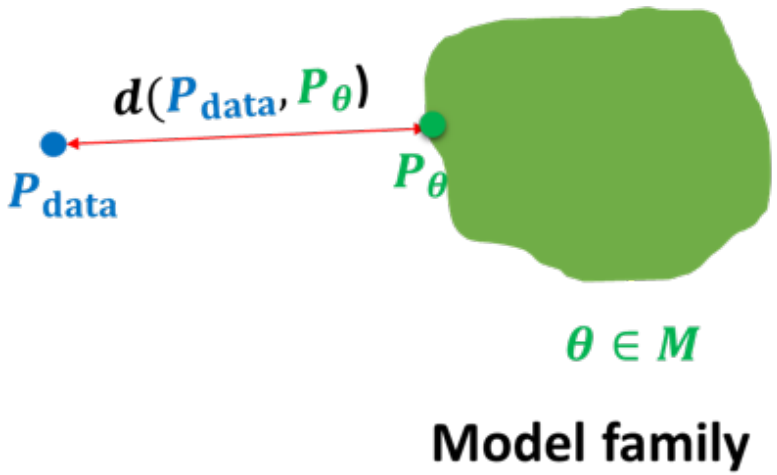
Our strategy will be to define an objective that balances among all these tasks.

Choosing a Learning Objective

We want to learn the full distribution so that later we can answer *any* probabilistic inference query

- In this setting we formulate a learning problem inspired by **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})

$$x_i \sim P_{\text{data}}$$
$$i = 1, 2, \ldots, n$$



- Our goal is to choose an objective $d(P_{\text{data}}, P_\theta)$ that captures "closeness" between $P_{\text{data}}, P_\theta$

A Learning Objective Based on the KL-Divergence

How should we measure "closeness" between distributions $P_{\text{data}}, P_\theta$?

- 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:

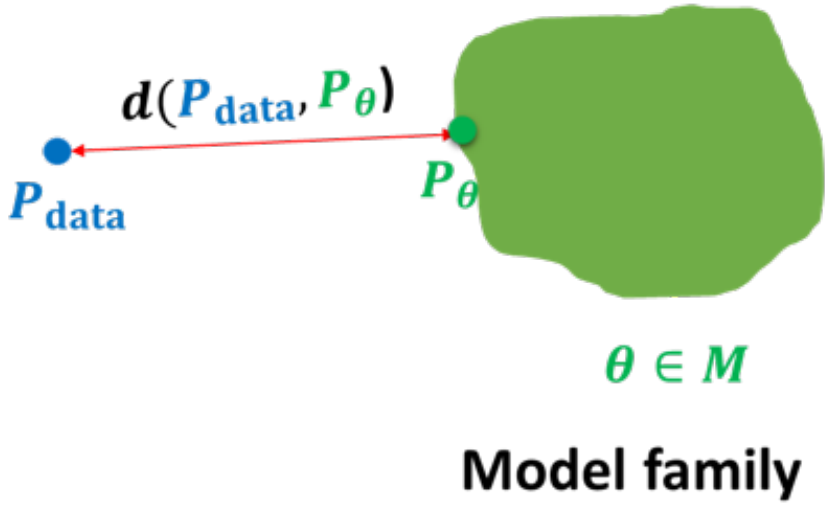
$$\mathbf{E}_{\mathbf{x}\sim p}\left[-\log \frac{q(\mathbf{x})}{p(\mathbf{x})}\right] \geq -\log\left(\mathbf{E}_{\mathbf{x}\sim p}\left[\frac{q(\mathbf{x})}{p(\mathbf{x})}\right]\right) = -\log\left(\sum_{\mathbf{x}} p(\mathbf{x})\frac{q(\mathbf{x})}{p(\mathbf{x})}\right) = 0$$

- Hence, this is a sensible metric to quantify similarity of p and q
- Notice that KL-divergence is **asymmetric**, i.e., $D(p\|q) \neq D(q\|p)$

Learning as KL-Divergence Minimization

We are going to use the KL-Divergence to estimate the similarity between the distributions $P_{\text{data}}, P_\theta$

$$x_i \sim P_{\text{data}}$$
$$i = 1, 2, \ldots, n$$



- The **KL-divergence** between $P_{\text{data}}, P_\theta$ is:

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

- $\mathbf{D}(P_{\text{data}} || P_\theta) = 0$ iff the two distributions are the same.

From KL-Divergence to Log-Likelihood

- We can simplify this somewhat:

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

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

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

  - 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 $\mathbf{H}(P_{\text{data}})$, we don't know how close we are to the optimum
- Problem: In general we do not know P_{data}.

Lecture Outline

1. Learning via Maximum Likelihood
    - The Learning Problem
    - The KL-Divergence as a Learning Objective
    - From KL-Divergence to Maximum Likelihood
2. **Maximizing the Likelihood from Data**
    - Monte Carlo Estimation
    - Gradient Descent
3. Statistical Issues and the Bias/Variance Tradeoff

Monte Carlo Estimation

To derive a learning objective, we will use *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 an *estimator* and a random variable.

Properties of the Monte Carlo Estimator

- **Unbiased:**

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

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

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

- **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.

Empirical Maximum Likelihood

We can now use Monte Carlo to derive a practical learning objective:

- We approximate the expected log-likelihood

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

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

An Example: Flipping a Biased Coin

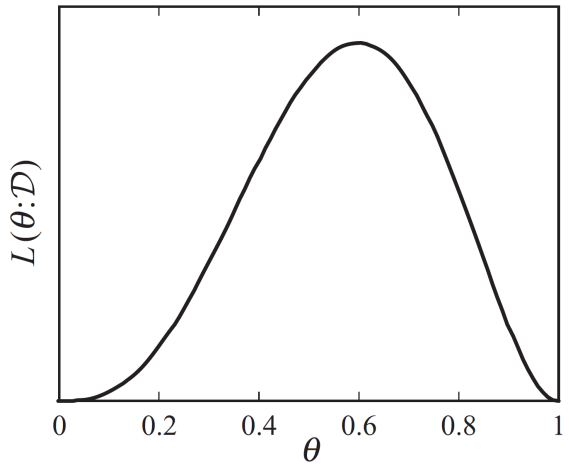
Let's start with a simple example: flipping a biased coin

- 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 $P_{\text{data}}(x)$ 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 $\mathcal{D}$?

Maximum Likelihood Estimation for the Biased Coin

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 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

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

- 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

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

- 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

Our 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

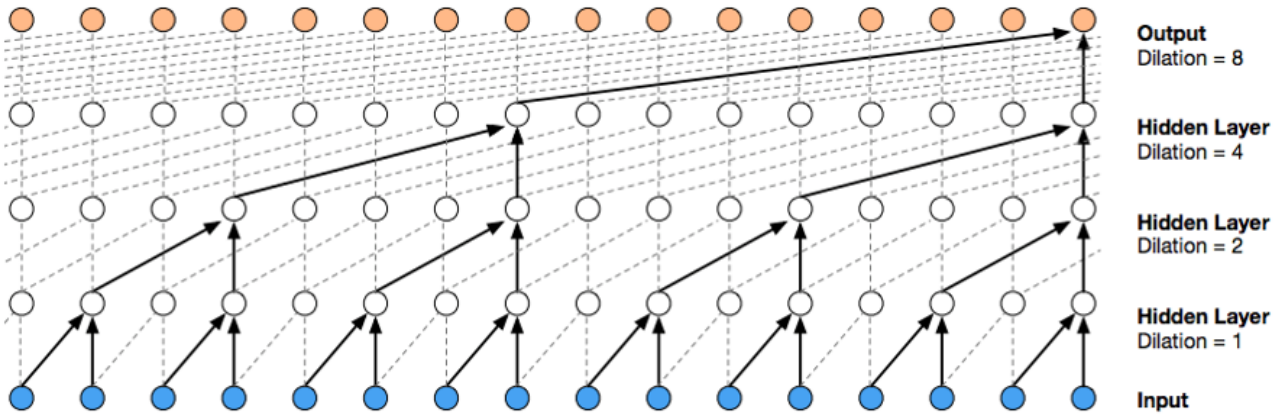
$$P(x_t \mid x_{1:t-1}) = P(x_t \mid h_{t-1}, x_{t-1}) \qquad h_t = f(h_{t-1}, x_{t-1}).$$

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 WaveNet:

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

Recall: WaveNet (Oord et al., 2016)

WaveNet is a state of the art model for speech:



WaveNet uses convolutions to parameterize a MADE-like masked autoencoder model mapping inputs x to model parameters $\hat{x}_i$.

Lecture Outline

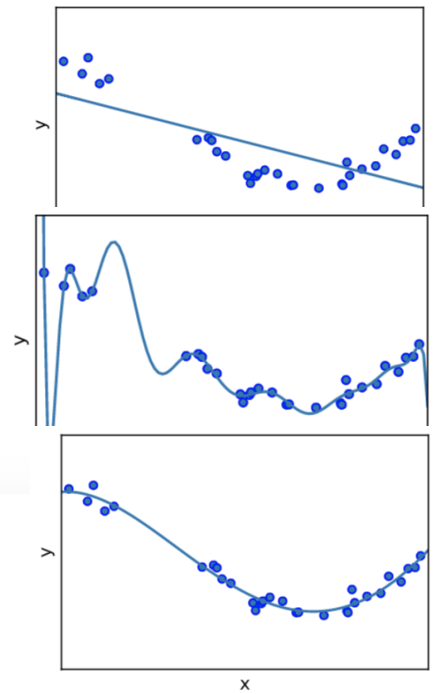
1. Learning via Maximum Likelihood
    - The Learning Problem
    - The KL-Divergence as a Learning Objective
    - From KL-Divergence to Maximum Likelihood
2. Maximizing the Likelihood from Data
    - Monte Carlo Estimation
    - Gradient Descent
3. **Statistical Issues and the Bias/Variance Tradeoff**

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)