CS/DS 541: Class 2

Jacob Whitehill

Linear regression (aka 2-layer NN)

Linear regression

- Let dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$

- Want to create a neural network — which we can also treat as a “machine” — to estimate each $y^{(i)}$ with high accuracy.

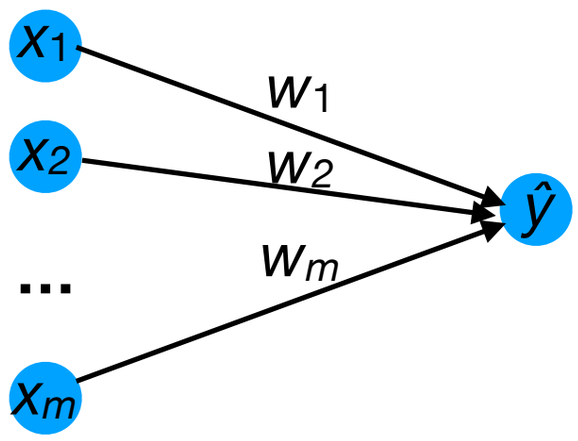
- Let us define the machine by a function g (with parameters $\mathbf{w}$) whose output $\hat{y}$ is linear in its inputs:

$$\hat{y} \doteq g(\mathbf{x}; \mathbf{w}) \doteq \sum_{j=1}^{m} x_j w_j = \mathbf{x}^\top \mathbf{w}$$

Linear regression

- Note that this function is equivalent to a 2-layer neural network (with no activation function):

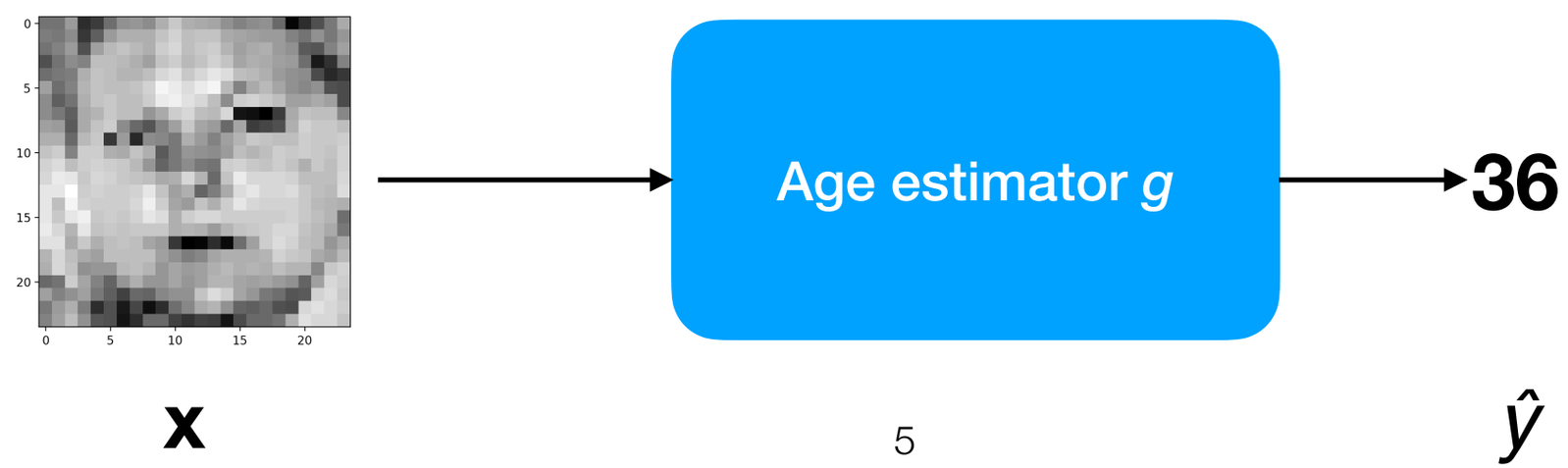
$$\hat{y} \doteq g(\mathbf{x}; \mathbf{w}) \doteq \sum_{j=1}^{m} x_j w_j = \mathbf{x}^\top \mathbf{w}$$



Linear regression

- Example application: automated face analysis to estimate a person's age from their face image.

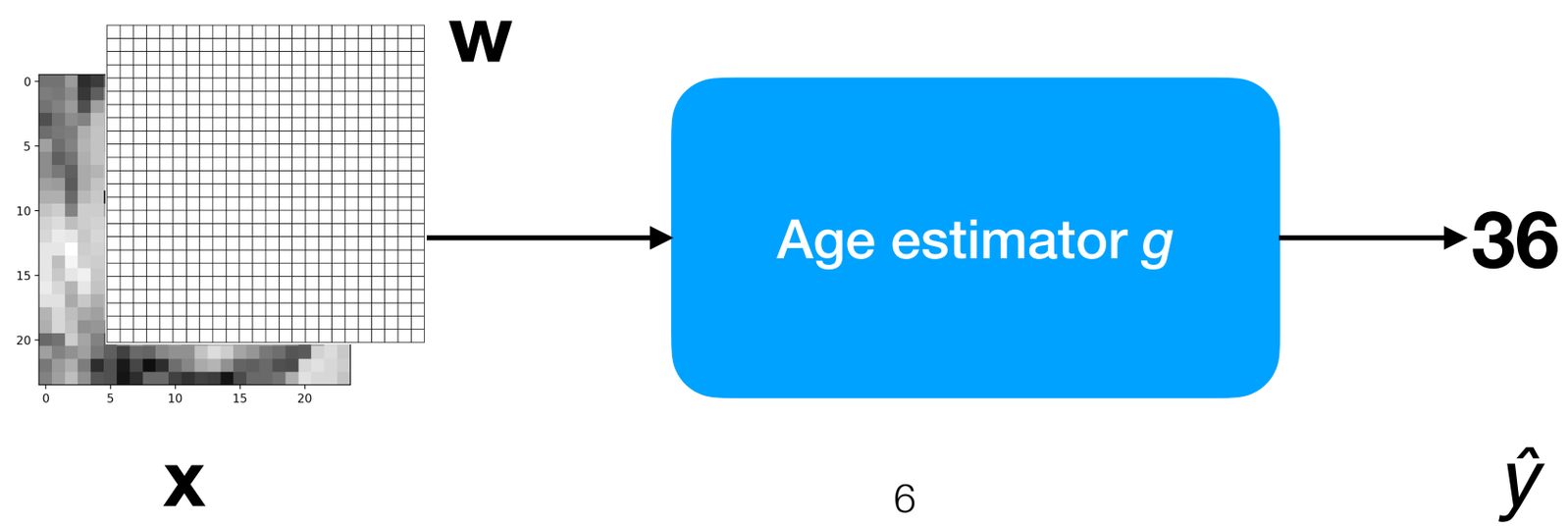
- Here, we treat the image **x** as a *vector* (even though it represents a 2-d image).



Linear regression

- Example application: automated face analysis to estimate a person's age from their face image.

- Vector **w** represent an "overlay image" that weights the different pixel intensities of **x**.



Linear regression

- Imagine a 2x2 pixel “image” **x** and a weight matrix **w**:

2	5
0	3

x

1	3
2	4

w

- Then $\hat{y} = 2*1 + 5*3 + 0*2 + 3*4 = 29$

Linear regression

- Given our dataset $\mathcal{D}$, we want to optimize **w**.

- Let's choose each "weight" w_j to minimize the **mean squared error** (MSE) of our predictions.

- We can define the **loss** function that we seek to minimize:

$$
\begin{aligned}
f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) &= \frac{1}{2n} \sum_{i=1}^{n} \left(g(\mathbf{x}^{(i)}; \mathbf{w}) - y^{(i)} \right)^2 \\
&= \frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2
\end{aligned}
$$

Linear regression

- **w** is an unconstrained real-valued vector; hence, we can use differential calculus to find the minimum of f_{MSE}.

- Just derive the gradient of f_{MSE} w.r.t. **w**, set to 0, and solve.

- Since f_{MSE} is a convex function, we are guaranteed that this critical point is a global minimum.

Solving for **w**

- The gradient of f_{MSE} is thus:

$$\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) = \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right]$$

Solving for w

- The gradient of f_{MSE} is thus:

$$\begin{aligned}
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{2n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left[\left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right]
\end{aligned}$$

Solving for w

- The gradient of f_{MSE} is thus:

$$\begin{aligned}
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{2n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left[\left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)
\end{aligned}$$

Solving for **w**

- By setting to 0, splitting the sum apart, and solving, we reach the solution:

$$\nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left({\mathbf{x}^{(i)}}^{\top} \mathbf{w} - y^{(i)} \right)$$

Solving for **w**

- By setting to 0, splitting the sum apart, and solving, we reach the solution:

$$\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left({\mathbf{x}^{(i)}}^{\top} \mathbf{w} - y^{(i)} \right)$$

$$0 = \sum_{i} \mathbf{x}^{(i)} {\mathbf{x}^{(i)}}^{\top} \mathbf{w} - \sum_{i} \mathbf{x}^{(i)} y^{(i)}$$

Solving for **w**

- By setting to 0, splitting the sum apart, and solving, we reach the solution:

$$\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left({\mathbf{x}^{(i)}}^{\top} \mathbf{w} - y^{(i)} \right)$$

$$0 = \sum_{i} \mathbf{x}^{(i)} {\mathbf{x}^{(i)}}^{\top} \mathbf{w} - \sum_{i} \mathbf{x}^{(i)} y^{(i)}$$

$$\sum_{i} \mathbf{x}^{(i)} {\mathbf{x}^{(i)}}^{\top} \mathbf{w} = \sum_{i} \mathbf{x}^{(i)} y^{(i)}$$

Solving for **w**

- By setting to 0, splitting the sum apart, and solving, we reach the solution:

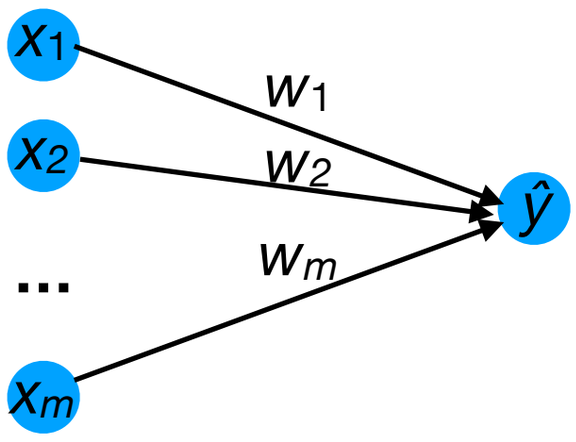
$$\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left({\mathbf{x}^{(i)}}^{\top} \mathbf{w} - y^{(i)} \right)$$

$$\begin{aligned} 0 &= \sum_i \mathbf{x}^{(i)} {\mathbf{x}^{(i)}}^{\top} \mathbf{w} - \sum_i \mathbf{x}^{(i)} y^{(i)} \\ \sum_i \mathbf{x}^{(i)} {\mathbf{x}^{(i)}}^{\top} \mathbf{w} &= \sum_i \mathbf{x}^{(i)} y^{(i)} \\ \mathbf{w} &= \left(\sum_i \mathbf{x}^{(i)} {\mathbf{x}^{(i)}}^{\top} \right)^{-1} \sum_i \mathbf{x}^{(i)} y^{(i)} \end{aligned}$$

Solving for **w**

- We have found the optimal **w** to minimize f_{MSE}.
- In other words, we can optimize a 2-layer linear NN for the MSE loss using the closed formula:

$$\mathbf{w} = \left(\sum_i \mathbf{x}^{(i)} \mathbf{x}^{(i)\top} \right)^{-1} \sum_i \mathbf{x}^{(i)} y^{(i)}$$



Matrix notation

- We can also derive the same solution using matrix notation:

- Let's define a matrix **X** to contain all the training images:

$$\mathbf{X} = \begin{bmatrix} | & & | \\ \mathbf{x}^{(1)} & \cdots & \mathbf{x}^{(n)} \\ | & & | \end{bmatrix}$$

  - In statistics, **X** is called the **design matrix**.*

- Let's define vector **y** to contain all the training labels:

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

* Actually, statistics literature typically defines this as $\mathbf{X}^T$.

Matrix notation

- Using summation notation, we derived:

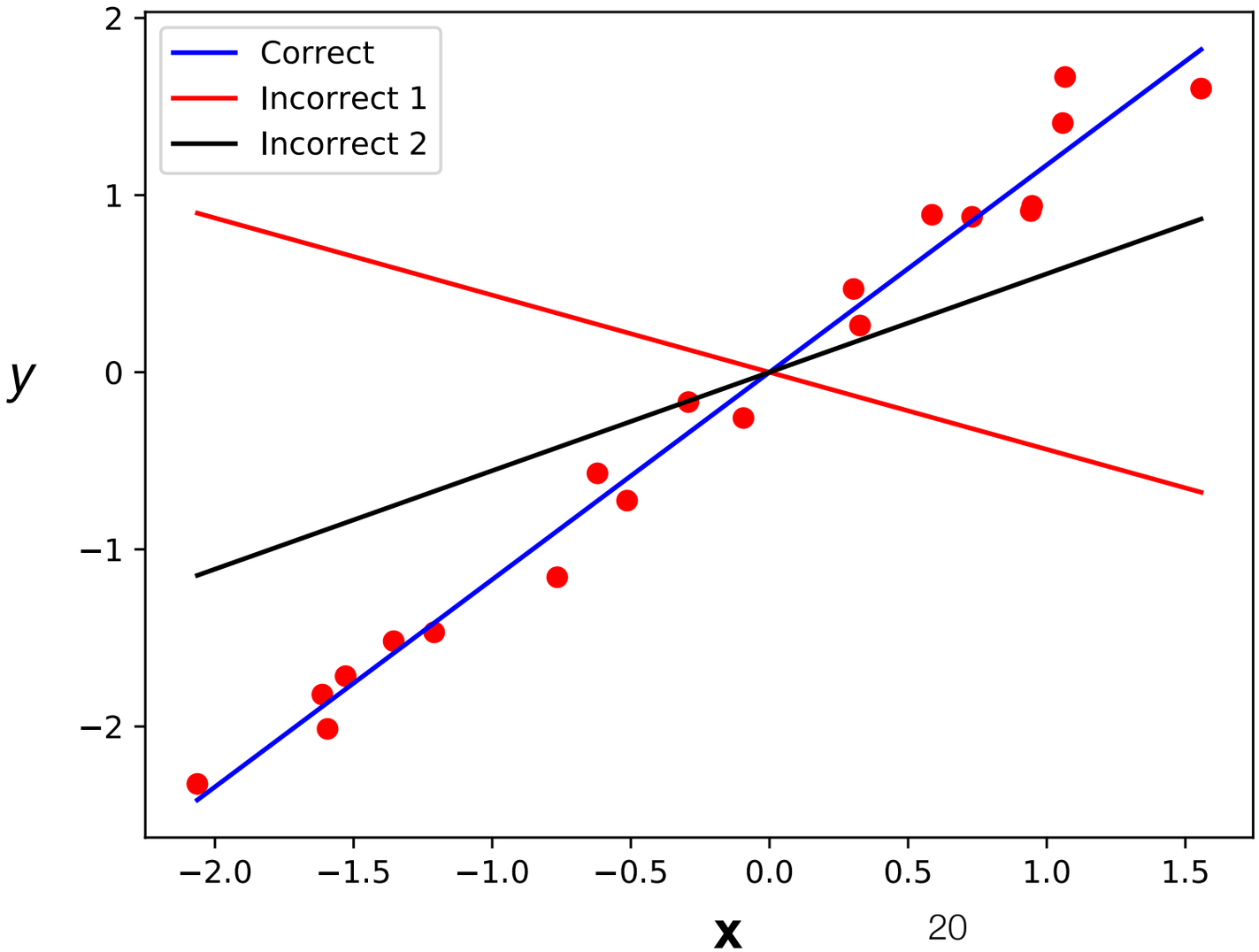
$$\mathbf{w} = \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} {\mathbf{x}^{(i)}}^\top \right)^{-1} \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} y^{(i)} \right)$$

- Using matrix notation, we can write the solution as:

$$\mathbf{w} = \left(\mathbf{X}\mathbf{X}^\top \right)^{-1} \mathbf{X}\mathbf{y} \qquad \text{where } \mathbf{X} = \begin{bmatrix} | & & | \\ \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \\ | & & | \end{bmatrix}$$

1-d example

- Linear regression finds the weight vector **w** that minimizes the f_{MSE}. Here's an example where each **x** is just 1-d…

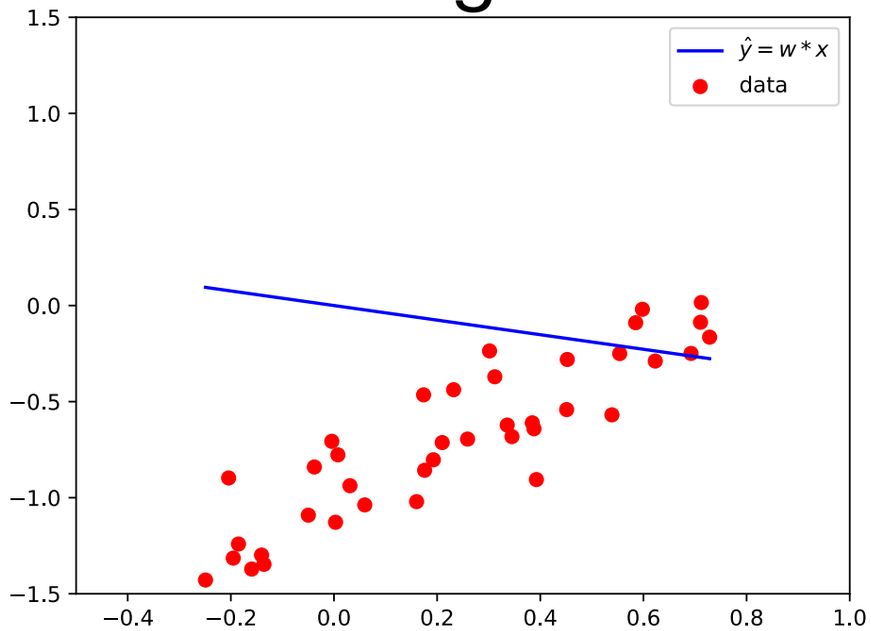


The best **w** is the one such that $f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}})$ is as small as possible, where each $\hat{y} = \mathbf{x}^{\mathsf{T}}\mathbf{w}$.

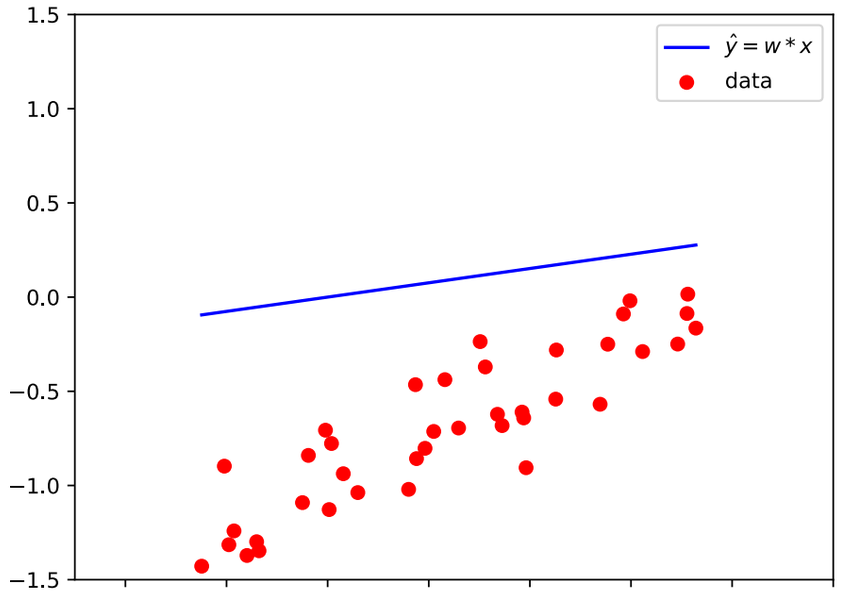
Exercise

- Which of the following regression lines would be predicted using the model described above?

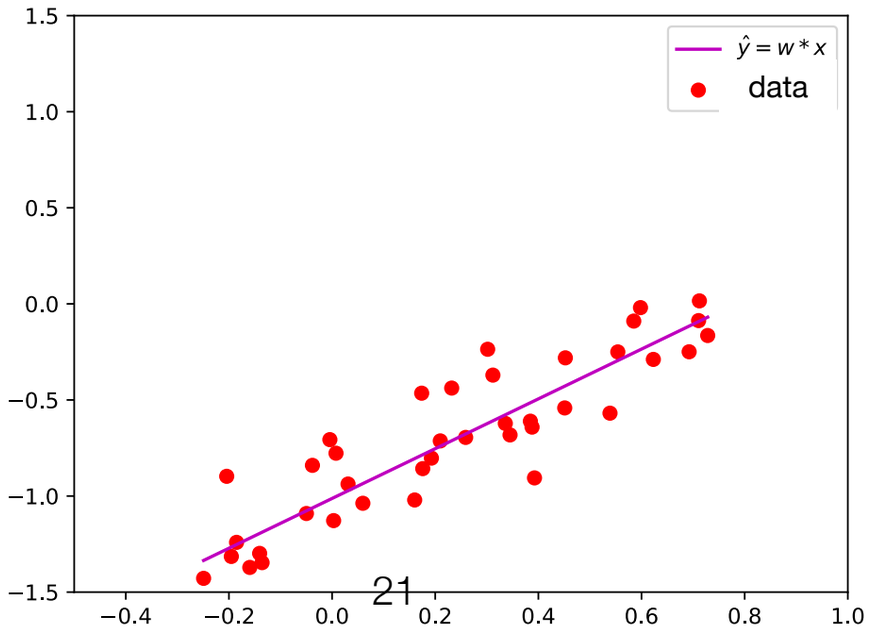
1.



2.



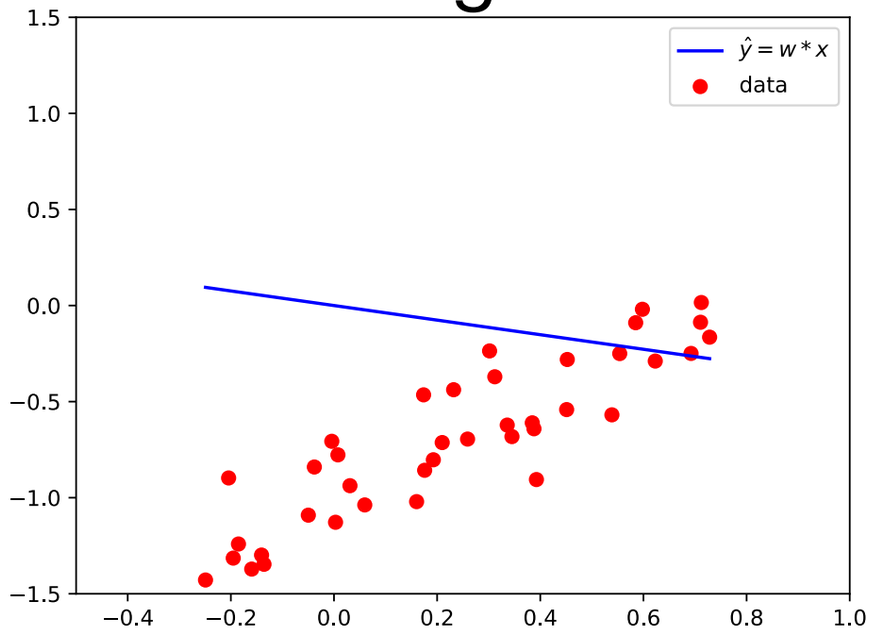
3.



Exercise

- Which of the following regression lines would be predicted using the model described above?

1.



Notice that the model enforces that (*x*,*y*)=(0,0) lie in the graph. Because of this constraint, the model learns the wrong slope to minimize the MSE.

Bias term

- In order to account for target values with non-zero mean, we can add a bias term to our model:

$$\hat{y} = \mathbf{x}^\top \mathbf{w} + b$$

- We can then compute the gradient w.r.t. both **w** and *b*, set to 0, and then solve the resulting system of equations.

$$\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}, b) = \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} + b - y^{(i)} \right)^2 \right]$$

$$\nabla_{b} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}, b) = \nabla_{b} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} + b - y^{(i)} \right)^2 \right]$$

Gradient descent

Linear regression

- Linear regression is one of the few ML algorithms that has an analytical solution:

$$\mathbf{w} = \left(\mathbf{X}\mathbf{X}^\top\right)^{-1}\mathbf{X}\mathbf{y}$$

- **Analytical solution**: there is a closed formula for the answer.

Linear regression

- Alternatively, linear regression can be solved numerically using gradient descent.

- **Numerical solution**: need to iterate (according to some algorithm) many times to *approximate* the optimal value.

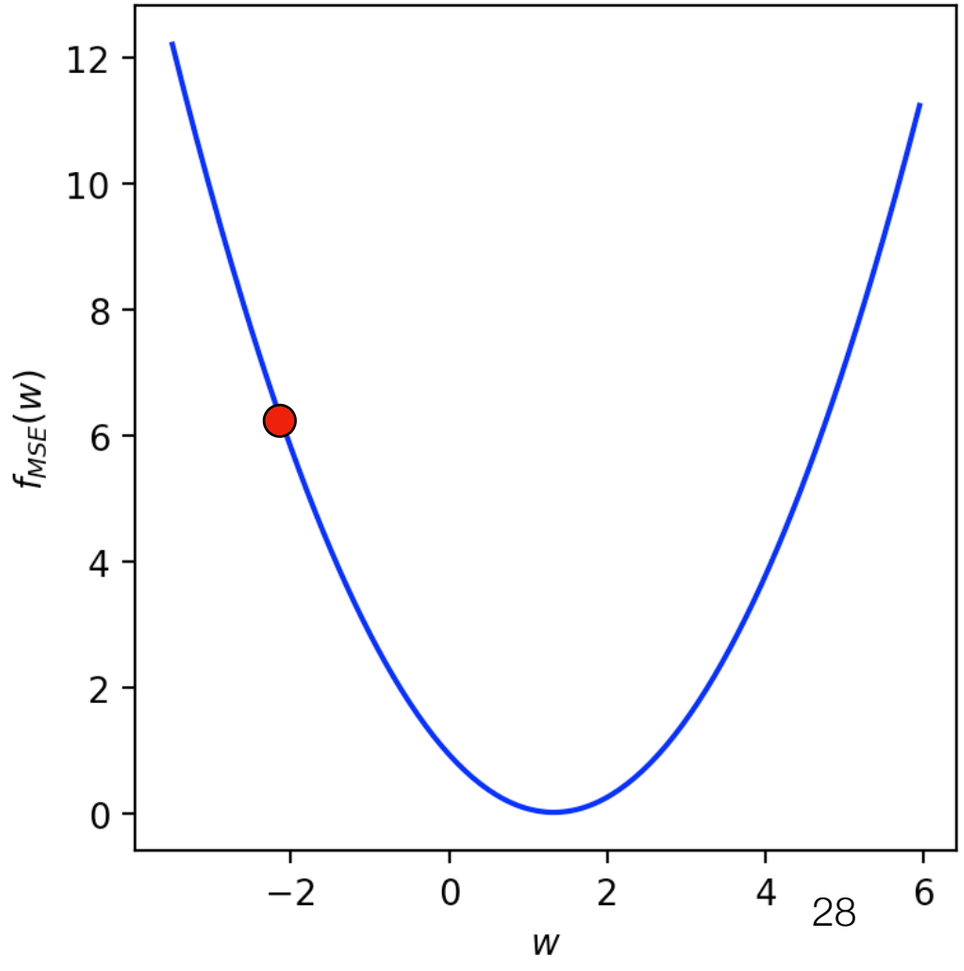
- Gradient descent is more laborious to code than the one-shot solution, but it generalizes to a wide variety of ML models.

Gradient descent

- Gradient descent is a **hill climbing algorithm** that uses the gradient (aka slope) to decide which way to “move” **w** to reduce the objective function (e.g., f_{MSE}).

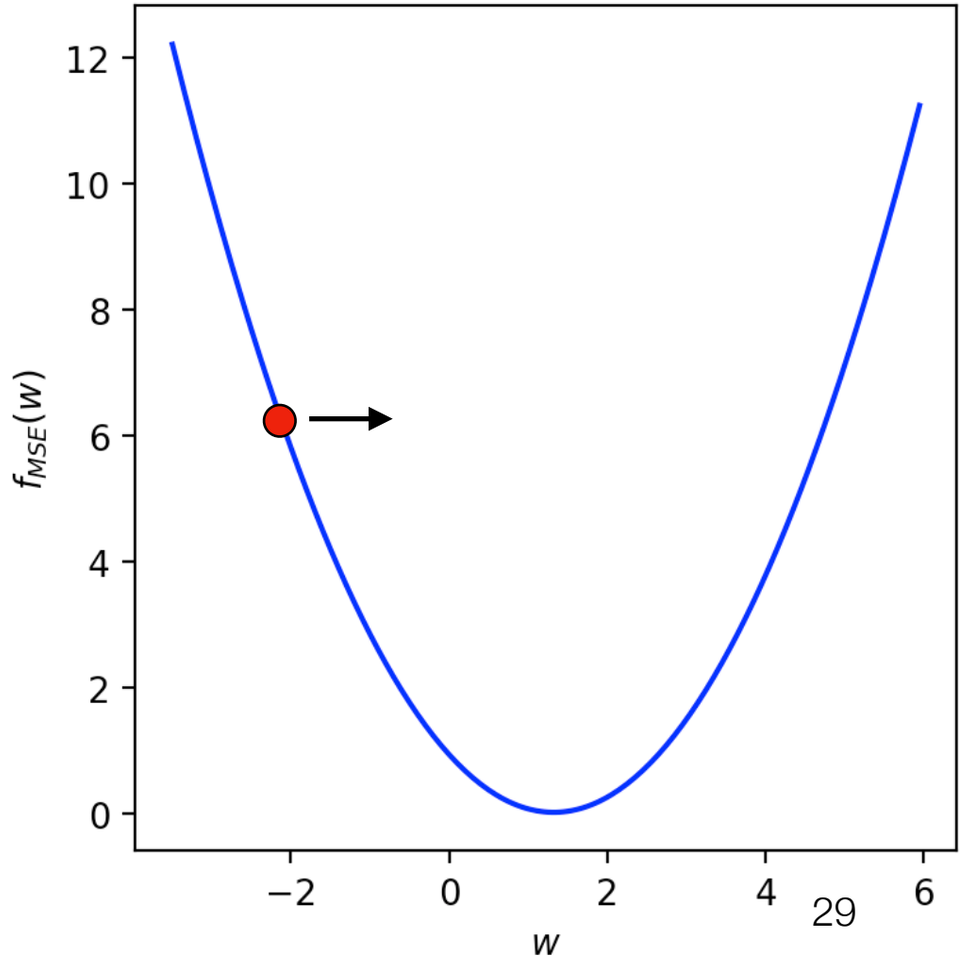
Gradient descent

- Suppose we just guess an initial value for **w** (e.g., -2.1).
- How can we make it better — increase it or decrease it?



Gradient descent

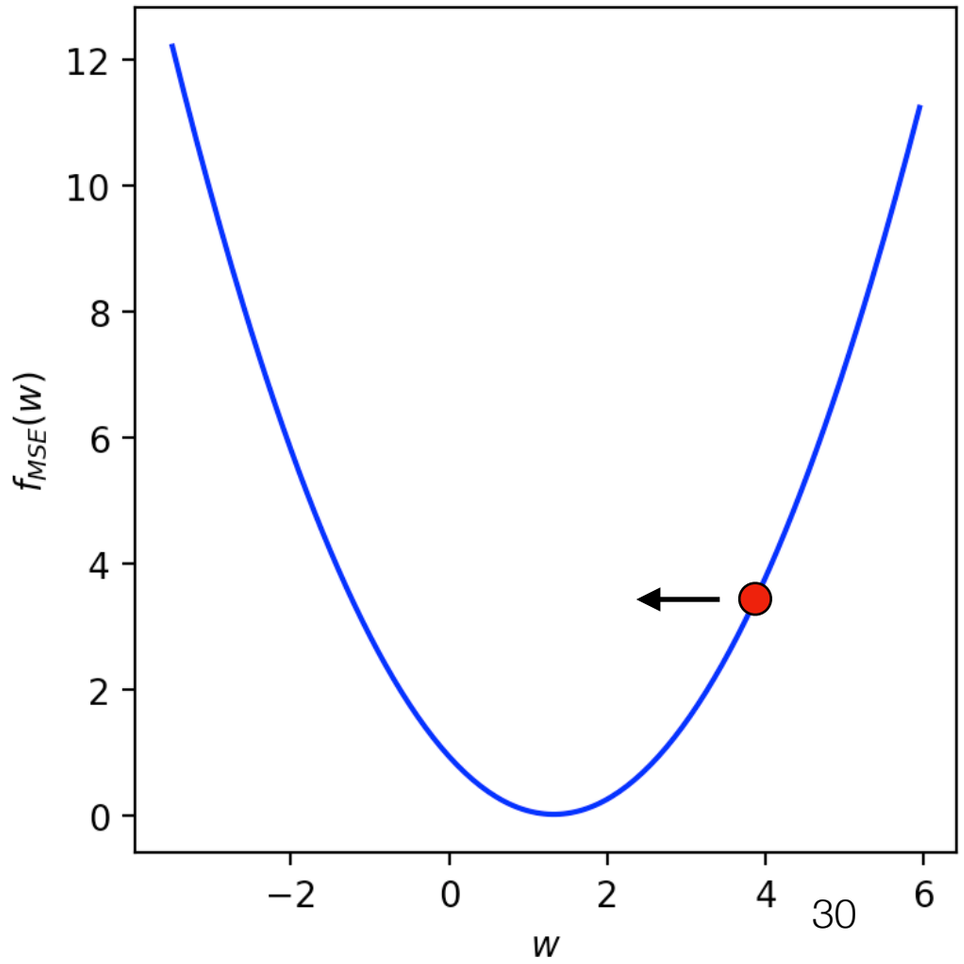
- Suppose we just guess an initial value for **w** (e.g., -2.1).
- How can we make it better — **increase** it or decrease it?
  - What does the **slope** of f_{MSE} tell us to do?



The slope at f_{MSE}(-2.1) is *negative*, i.e., we can *decrease* our cost by *increasing* w.

Gradient descent

- Or maybe our initial guess for **w** was 3.9.
- How can we make it better — increase it or **decrease** it?
  - What does the **slope** of f_{MSE} tell us to do?



The slope at $f_{MSE}(3.9)$ is *positive*, i.e., we can *decrease* our cost by *decreasing* w.

Gradient descent

- How do we know the slope? Compute the **gradient** of f_{MSE} w.r.t. **w**:

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{2n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left[\left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right) \\
&= \frac{1}{n} \mathbf{X} \left(\mathbf{X}^{\top} \mathbf{w} - \mathbf{y} \right)
\end{aligned}
$$

Gradient descent

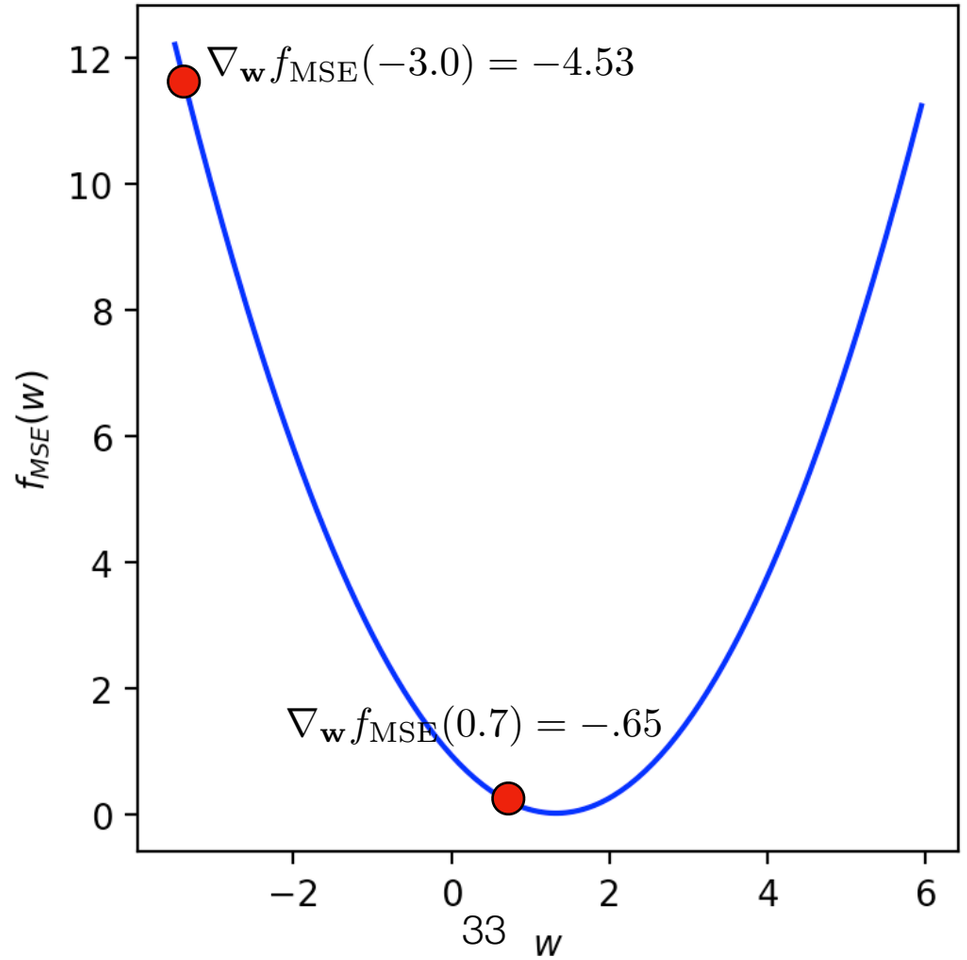
- How do we know the slope? Compute the **gradient** of f_{MSE} w.r.t. **w**:

$$\begin{aligned}
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{2n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left[\left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right) \\
&= \frac{1}{n} \mathbf{X} \left(\mathbf{X}^{\top} \mathbf{w} - \mathbf{y} \right)
\end{aligned}$$

- Then plug in the current value of **w**.
  (Note that **X** and **y** are computed from the data and are constant.)

Gradient descent

- How *far* do we "move" left or right?
  - Notice that, in the graph below, the **magnitude** of the slope (aka gradient) gives an indication of how far we need to go to reach the optimal **w**.

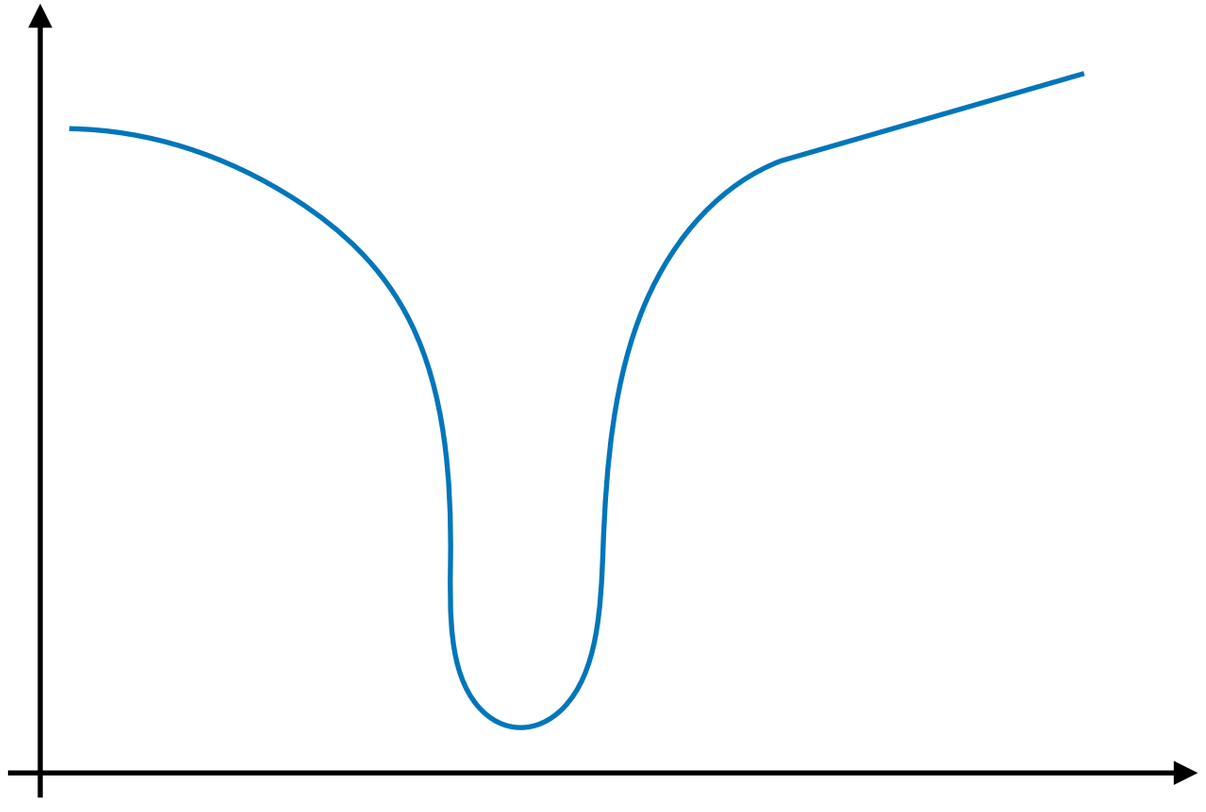


Exercise

- Draw on paper a function such that this property is false.

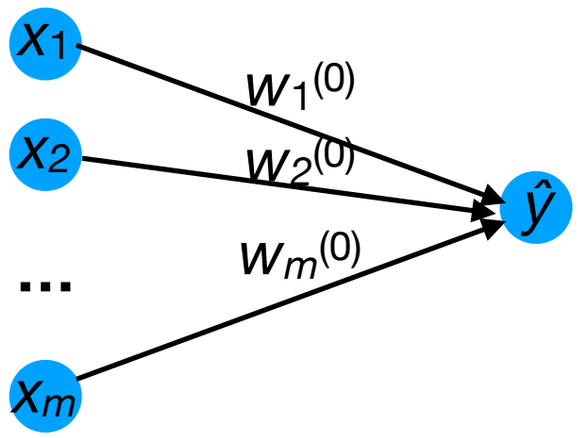
Exercise

- Draw on paper a function such that this property is false.



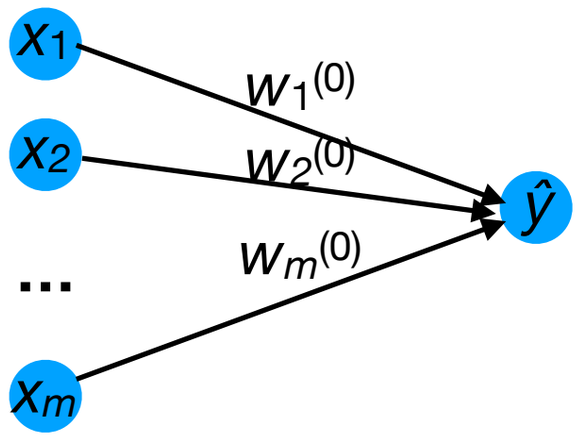
Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.



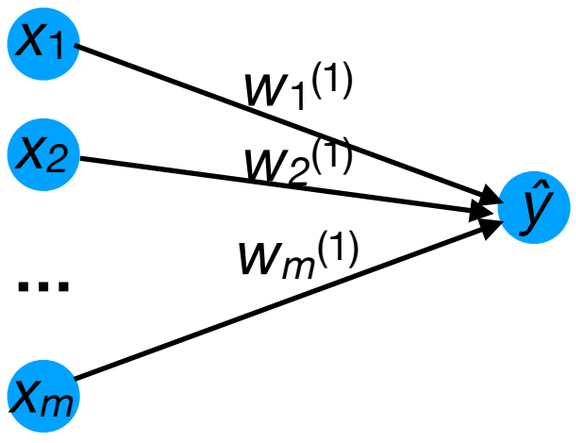
Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$



Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Update **w** by moving opposite the gradient, multiplied by a **learning rate** ε. $\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$



Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Update **w** by moving opposite the gradient, multiplied by a **learning rate** ε. $\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Repeat…

$$\mathbf{w}^{(2)} \leftarrow \mathbf{w}^{(1)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(1)})$$
$$\mathbf{w}^{(3)} \leftarrow \mathbf{w}^{(2)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(2)})$$
$$\mathbf{...}$$
$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(t-1)})$$

Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Update **w** by moving opposite the gradient, multiplied by a **learning rate** ε. $\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Repeat…

$$\mathbf{w}^{(2)} \leftarrow \mathbf{w}^{(1)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(1)})$$

$$\mathbf{w}^{(3)} \leftarrow \mathbf{w}^{(2)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(2)})$$

$$\ldots$$

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(t-1)})$$

- …until some convergence condition (e.g., #iterations, tolerance in function value diff, tolerance in weight diff, etc.).

Gradient descent demos

- Demo video.

Stochastic gradient descent

Gradient descent

- With gradient descent, we only update the weights after scanning the *entire* training set.
  - This is slow.
- If the training set contains 20K examples, then the gradient is an *average* over 20K images.
  - How much would the gradient really change if we just used, say, 10K images? 5K images? 128 images?

$$\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) = \frac{1}{n}\mathbf{X}(\mathbf{X}^\top \mathbf{w} - \mathbf{y})$$

Average over entire training set.

Stochastic gradient descent

- This is the idea behind **stochastic gradient descent** (SGD):
  - Randomly sample a small ($\ll n$) **mini-batch** (or sometimes just **batch**) of training examples.
  - Estimate the gradient on just the mini-batch.
  - Update weights based on *mini-batch* gradient estimate.
  - Repeat.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.
- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

- Procedure:

1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
2. Randomize the order of the examples in the training set.
3. For *e* = 0 to numEpochs:

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

- Procedure:

1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
2. Randomize the order of the examples in the training set.
3. For *e* = 0 to numEpochs:
   I. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.
- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *e* = 0 to numEpochs:
     I. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
        A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

- Procedure:

1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
2. Randomize the order of the examples in the training set.
3. For *e* = 0 to numEpochs:
   I. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
      A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.
      B. Compute the gradient on this mini-batch: $\frac{1}{\tilde{n}} \sum_{i \in \mathcal{J}} \nabla_{\mathbf{W}} f(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}; \mathbf{W})$

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.
- Procedure:

1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
2. Randomize the order of the examples in the training set.
3. For *e* = 0 to numEpochs:
   I. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
      A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.
      B. Compute the gradient on this mini-batch: $\frac{1}{\tilde{n}} \sum_{i \in \mathcal{J}} \nabla_{\mathbf{W}} f(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}; \mathbf{W})$
      C. Update the weights based on the current mini-batch gradient.

SGD versus GD: example

- Suppose our training set contains *n*=8 examples.
- Here is how *regular* gradient descent would proceed:
  - **Initialize weights $\mathbf{w}^{(0)}$ to random values.**

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - **Compute gradient on all n examples.**

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - Compute gradient on all n examples.
    - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \nabla_{\mathbf{w}} f$

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains *n*=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - **Compute gradient on all *n* examples.**
    - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \nabla_{\mathbf{w}} f$

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - Compute gradient on all n examples.
    - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \nabla_{\mathbf{w}} f$

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - **Initialize weights $\mathbf{w}^{(0)}$ to random values.**

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - **Randomize the order of the training data.**

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): **e=1**
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**

Training examples

4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): **e=1**
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples

4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): **e=1**
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples

4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): **e=1**
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples

4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): **e=1**
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples

4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): **e=1**
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples

4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): **e=1**
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples

4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): **e=1**
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples

4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): e=2
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples

4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains *n*=8 examples with $\tilde{n} = 2$.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (*e*=1, …, *E*): e=2
    - For each round (*r*=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

  ...

Training examples

4
1
3
5
7
6
8
2

Stochastic gradient descent

- Despite “noise” (statistical inaccuracy) in the mini-batch gradient estimates, we will still converge to local minimum.

- Training can be much faster than regular gradient descent because we adjust the weights *many times* per epoch.

SGD: learning rates

- With SGD, our learning rate ϵ needs to be **annealed** (reduced slowly over time) to guarantee convergence.
  - Otherwise we might just oscillate forever in weight space.
- Necessary conditions:

$$\lim_{T \to \infty} \sum_{t=1}^{T} |\epsilon_t|^2 < \infty$$

Not too big: sum of squared learning rates converges.

SGD: learning rates

- With SGD, our learning rate ϵ needs to be **annealed** (reduced slowly over time) to guarantee convergence.
  - Otherwise we might just oscillate forever in weight space.
- Necessary conditions:

$$\lim_{T\to\infty} \sum_{t=1}^{T} |\epsilon_t|^2 < \infty$$

$$\lim_{T\to\infty} \sum_{t=1}^{T} |\epsilon_t| = \infty$$

Not too small: sum of absolute learning rates grows to infinity.

SGD: learning rates

- One common learning rate "schedule" is to multiply ϵ by $c \in (0, 1)$ every *k* rounds.
    - This is called **exponential decay**.
- Another possibility (which avoids the issue) is to set the number of epochs *T* to a finite number.
    - SGD may not fully converge, but the machine might still perform well.
- There are many other strategies.

Probabilistic machine learning

Variational Auto-Encoders (Kingma & Welling 2014)

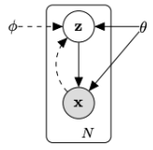


Figure 1: The type of directed graphical model under consideration. Solid lines denote the generative model $p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, dashed lines denote the variational approximation $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ to the intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The variational parameters $\boldsymbol{\phi}$ are learned jointly with the generative model parameters $\boldsymbol{\theta}$.

straightforward to extend this scenario to the case where we also perform variational inference on the global parameters; that algorithm is put in the appendix, but experiments with that case are left to future work. Note that our method can be applied to online, non-stationary settings, e.g. streaming data, but here we assume a fixed dataset for simplicity.

1. *Intractability*: the case where the integral of the marginal likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, d\mathbf{z}$ is intractable (so we cannot evaluate or differentiate the marginal likelihood), where the true posterior density $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})/p_{\boldsymbol{\theta}}(\mathbf{x})$ is intractable (so the EM algorithm cannot be used), and where the required integrals for any reasonable mean-field VB algorithm are also intractable. These intractabilities are quite common and appear in cases of moderately complicated likelihood functions $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, e.g. a neural network with a nonlinear hidden layer.

1. *Intractability*: the case where the integral of the marginal likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, d\mathbf{z}$ is intractable (so we cannot evaluate or differentiate the marginal likelihood), where the true posterior density $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})/p_{\boldsymbol{\theta}}(\mathbf{x})$ is intractable (so the EM algorithm cannot be used), and where the required integrals for any reasonable mean-field VB algorithm are also intractable. These intractabilities are quite common and appear in cases of moderately complicated likelihood functions $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, e.g. a neural network with a nonlinear hidden layer.

2. *A large dataset*: we have so much data that batch optimization is too costly; we would like to make parameter updates using small minibatches or even single datapoints. Sampling-based solutions, e.g. Monte Carlo EM, would in general be too slow, since it involves a typically expensive sampling loop per datapoint.

We are interested in, and propose a solution to, three related problems in the above scenario:

1. Efficient approximate ML or MAP estimation for the parameters $\boldsymbol{\theta}$. The parameters can be of interest themselves, e.g. if we are analyzing some natural process. They also allow us to mimic the hidden random process and generate artificial data that resembles the real data.
2. Efficient approximate posterior inference of the latent variable $\mathbf{z}$ given an observed value $\mathbf{x}$ for a choice of parameters $\boldsymbol{\theta}$. This is useful for coding or data representation tasks.
3. Efficient approximate marginal inference of the variable $\mathbf{x}$. This allows us to perform all kinds of inference tasks where a prior over $\mathbf{x}$ is required. Common applications in computer vision include image denoising, inpainting and super-resolution.

Normalizing Flows (Rezende & Mohamed 2015)

tion). For example, if $q_\phi(z)$ is a Gaussian distribution $\mathcal{N}(z|\mu, \sigma^2)$, with $\phi = \{\mu, \sigma^2\}$, then the location-scale transformation using the standard Normal as a base distribution allows us to reparameterize $\mathbf{z}$ as:

$$z \sim \mathcal{N}(z|\mu, \sigma^2) \Leftrightarrow z = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

- **Backpropagation with Monte Carlo**. We can now differentiate (backpropagation) w.r.t. the parameters ϕ of the variational distribution using a Monte Carlo approximation with draws from the base distribution:

$$\nabla_\phi \mathbb{E}_{q_\phi(z)}[f_\theta(z)] \Leftrightarrow \mathbb{E}_{\mathcal{N}(\epsilon|0,1)}[\nabla_\phi f_\theta(\mu + \sigma\epsilon)]\,.$$

tion). For example, if $q_\phi(z)$ is a Gaussian distribution $\mathcal{N}(z|\mu, \sigma^2)$, with $\phi = \{\mu, \sigma^2\}$, then the location-scale transformation using the standard Normal as a base distribution allows us to reparameterize $\mathbf{z}$ as:

$$z \sim \mathcal{N}(z|\mu, \sigma^2) \Leftrightarrow z = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

- **Backpropagation with Monte Carlo**. We can now differentiate (backpropagation) w.r.t. the parameters ϕ of the variational distribution using a Monte Carlo approximation with draws from the base distribution:

$$\nabla_\phi \mathbb{E}_{q_\phi(z)}[f_\theta(z)] \Leftrightarrow \mathbb{E}_{\mathcal{N}(\epsilon|0,1)}[\nabla_\phi f_\theta(\mu + \sigma\epsilon)]\,.$$

where the Lth Gaussian distribution is not dependent on any other random variables. The prior over latent variables is a unit Gaussian $p(\mathbf{z}_l) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and the observation likelihood $p_\theta(\mathbf{x}|\mathbf{z})$ is any appropriate distribution that is conditioned on $\mathbf{z}_1$ and is also parameterized by a deep neural network (figure 2). This model class is very general and includes other models such as factor analysis and PCA, non-linear factor analysis, and non-linear Gaussian belief networks as special cases (Rezende et al., 2014).

DLGMs use continuous latent variables and is a model class perfectly suited to fast amortized variational inference using the lower bound (3) and stochastic backpropagation.

In this paper, we study deep latent Gaussian models (DLGM), which are a general class of deep directed graphical models that consist of a hierarchy of L layers of Gaussian latent variables $\mathbf{z}_l$ for layer l. Each layer of latent variables is dependent on the layer above in a non-linear way, and for DLGMs, this non-linear dependency is specified by deep neural networks. The joint probability model is:

$$p(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_L) = p\left(\mathbf{x}|f_0(\mathbf{z}_1)\right) \prod_{l=1}^{L} p\left(\mathbf{z}_l|f_l(\mathbf{z}_{l+1})\right) \quad (4)$$

By repeatedly applying the rule for change of variables, the initial density 'flows' through the sequence of invertible mappings. At the end of this sequence we obtain a valid probability distribution and hence this type of flow is referred to as a normalizing flow.

3.1. Finite Flows

The basic rule for transformation of densities considers an invertible, smooth mapping $f : \mathbb{R}^d \to \mathbb{R}^d$ with inverse

Probability theory

- Many machine learning algorithms, and some deep learning methods, are based on probability theory.
- Probabilities provide a natural way of expressing our **uncertainty** about a particular value, e.g.:
  - The ground-truth y we are trying to estimate.
  - Our estimate $\hat{y}$ of the ground-truth.

Quantifying uncertainty

- **Frequentist** probabilities:
  - Ask a large group of randomly selected people to **label** the face as smiling or not.
  - Count the number of labels for “smile” and divide by the total number of labels.
  - The ratio is the probability of “smile" for that face image.

Quantifying uncertainty

- **Bayesian** probabilities (“beliefs”):
  - Ask one person how much she/he believes the image is smiling, quantified as a number between 0 and 1.
  - The “belief” score is the probability of “smile" for that face image.

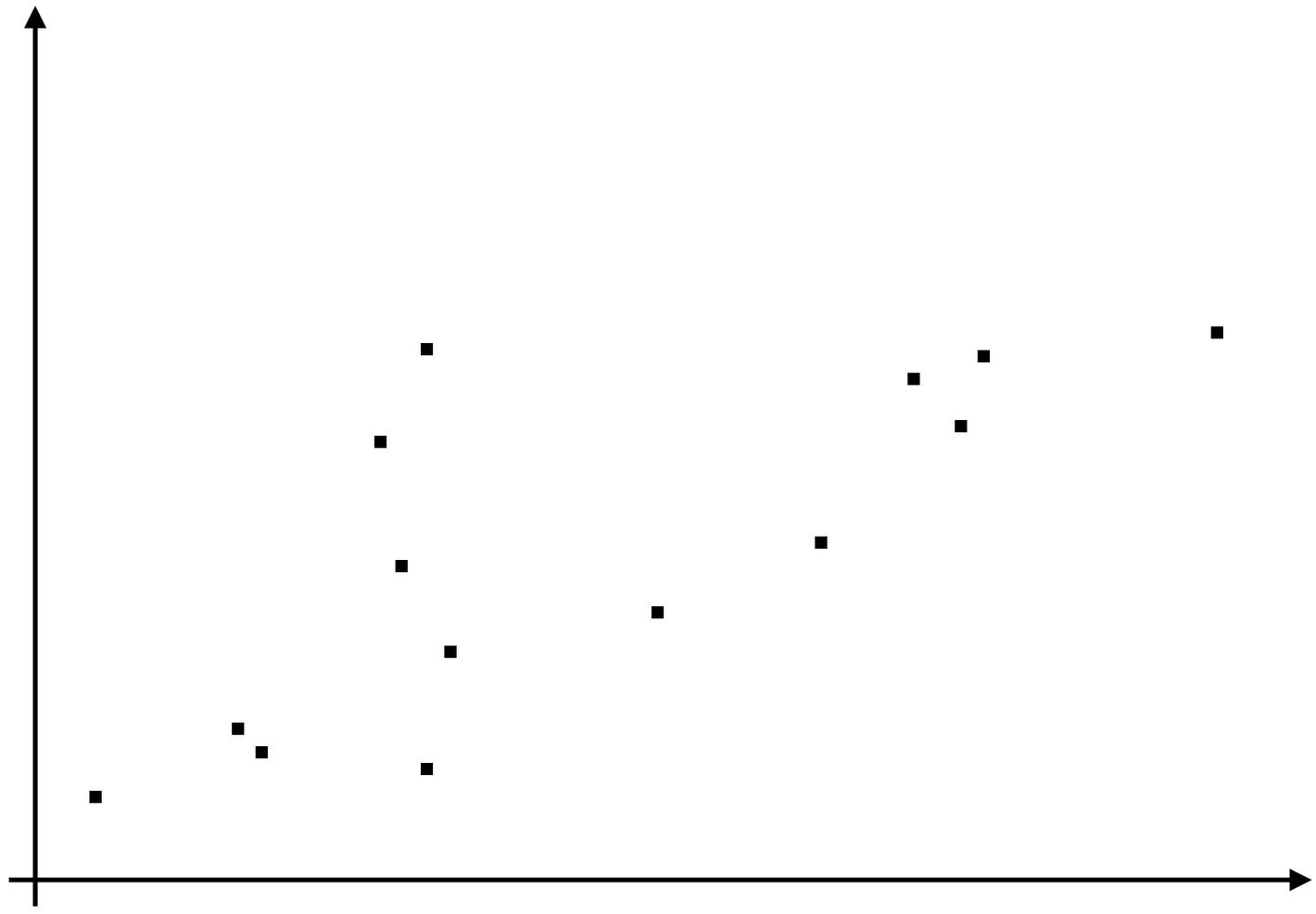
Quantifying uncertainty

- Many a debate has been waged about the validity of Bayesian versus frequentist probabilities.

- For the most part we will steer clear of these debates and use both tools when they are useful.

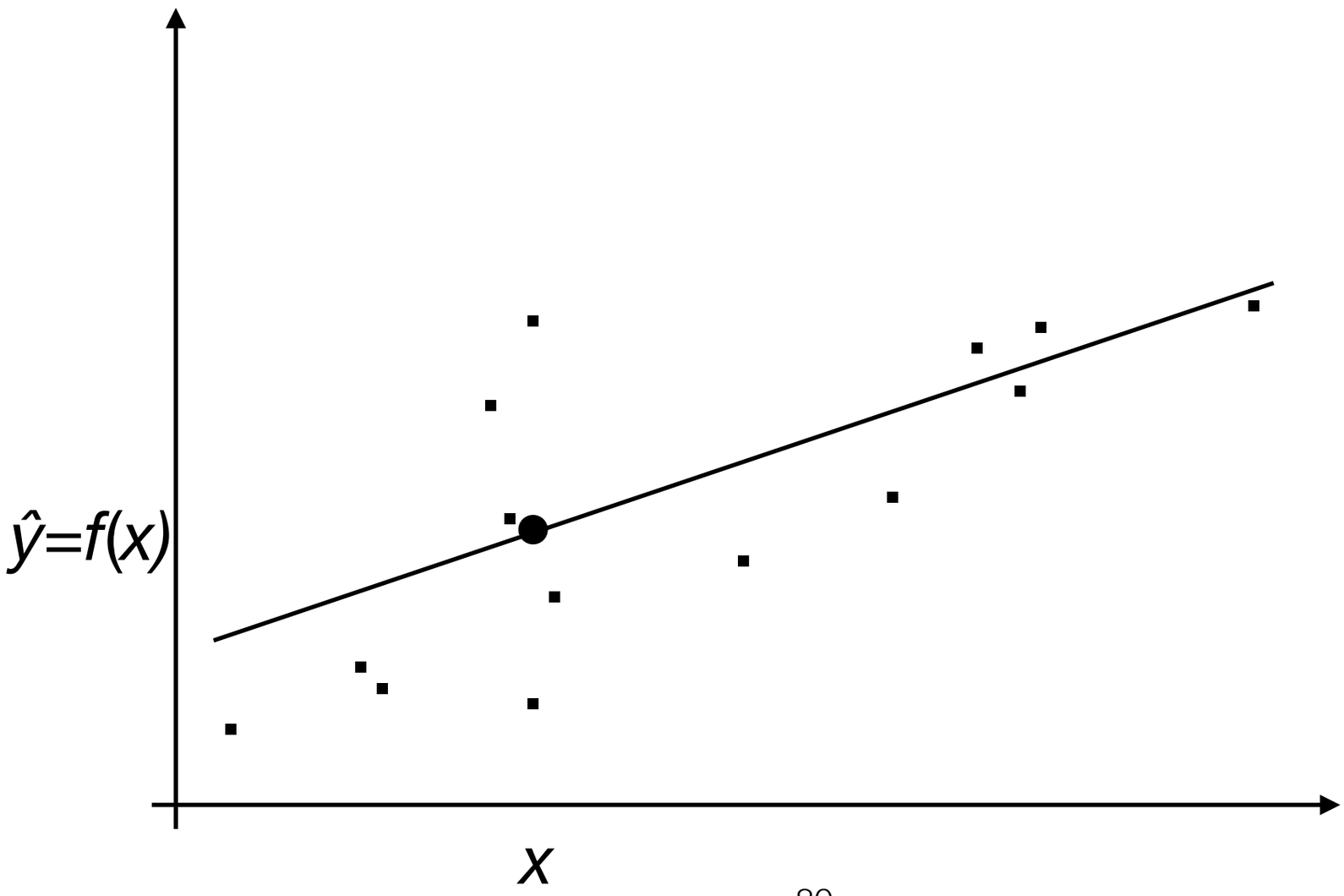
Uncertainty of estimation

- Sometimes we may be very uncertain about our prediction of the target value *y* from the input *x*.



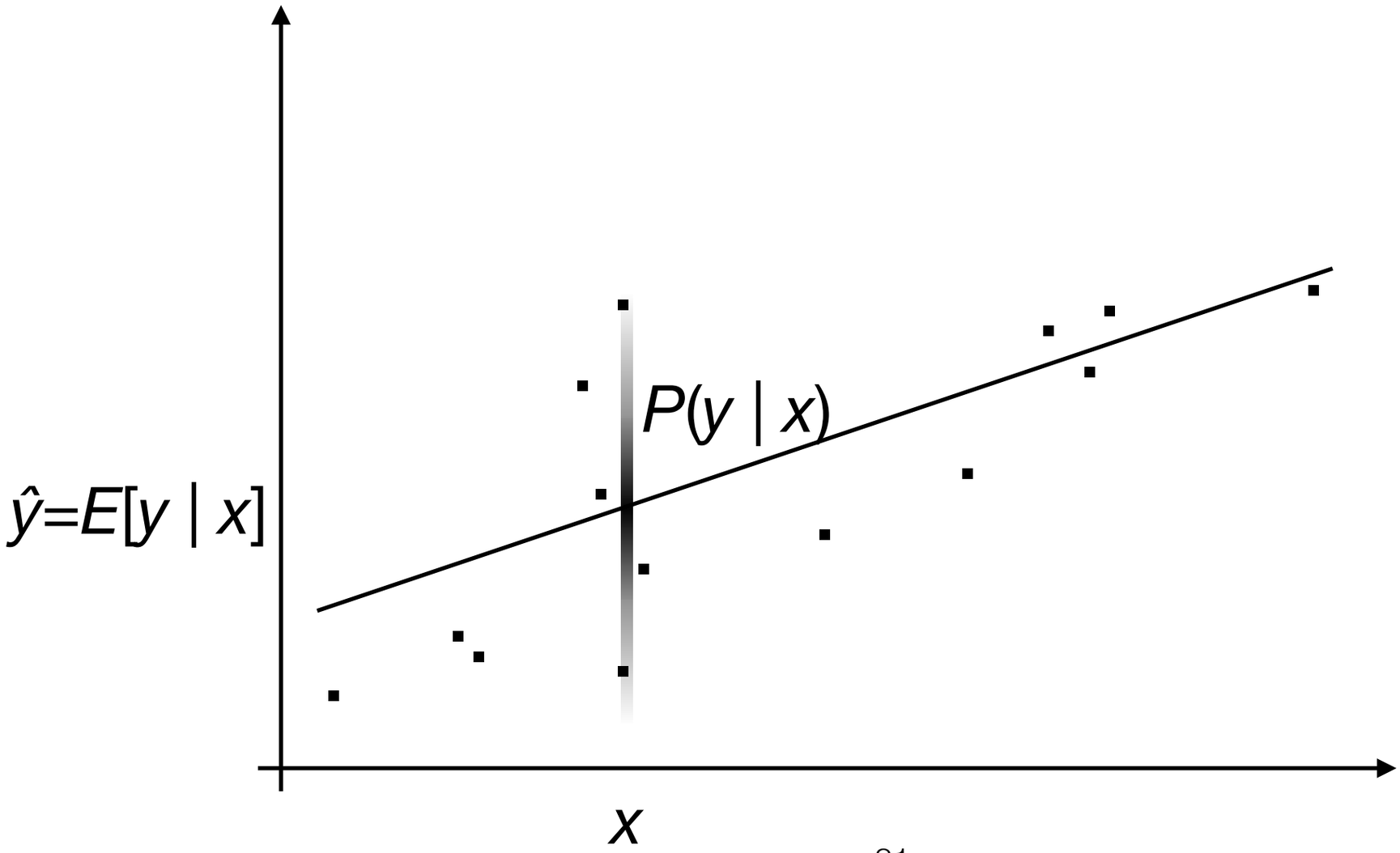
Uncertainty of estimation

- Rather than just giving a point estimate $\hat{y}=f(x)$…



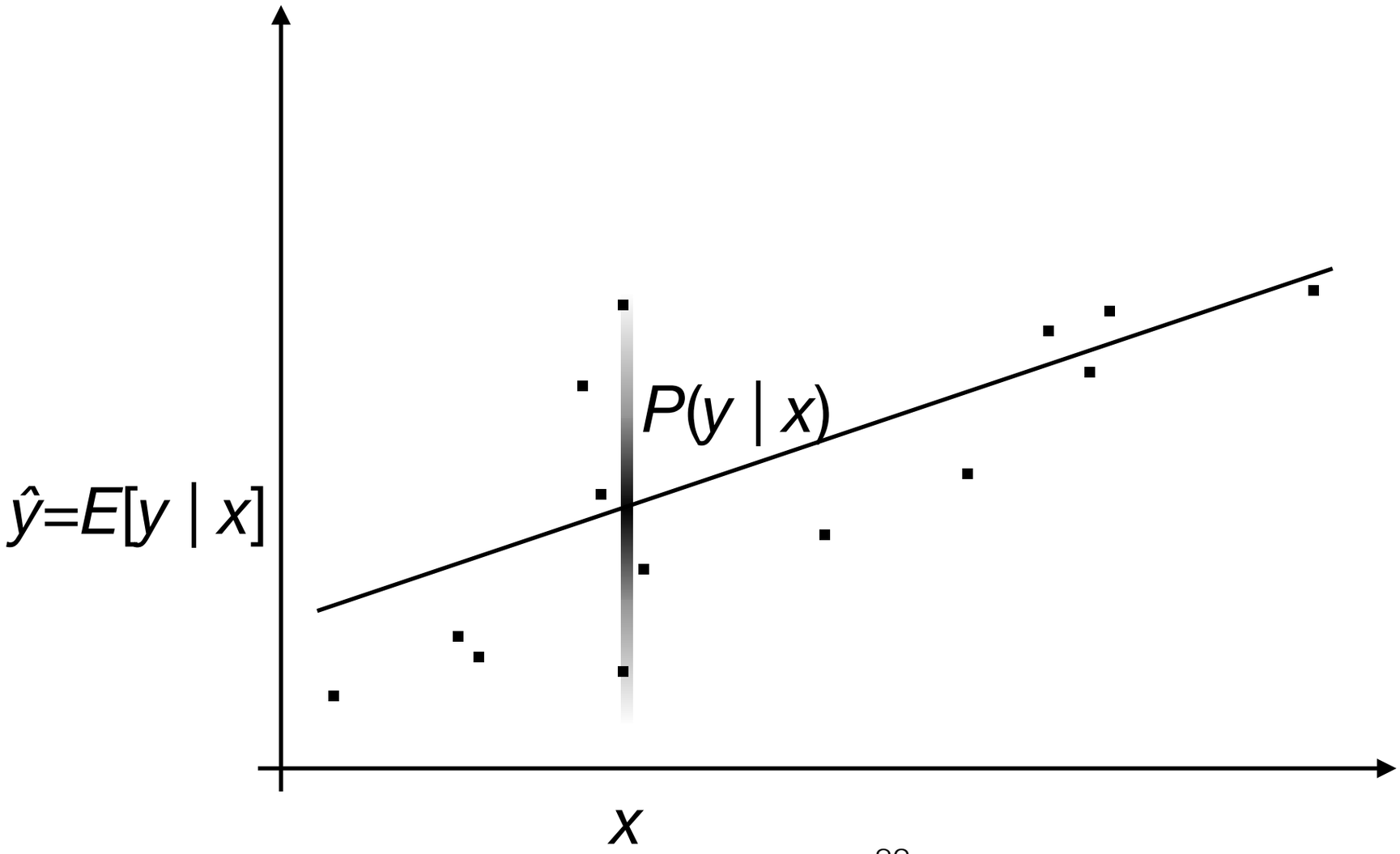
Uncertainty of estimation

- …we can estimate an entire probability distribution $P(y \mid x)$ that expresses our (conditional) uncertainty.



Uncertainty of estimation

- Indeed, it turns out that the optimal parameters for a conditional Gaussian probability model are **exactly the same** as for linear regression with minimal MSE (i.e., 2-layer NN).



Probabilistic deep learning

- Neural networks can be used in various ways to make probabilistic predictions:
  - For regression, estimate both the expected value and the variance of the prediction.
  - Model a high-dimensional distribution using a probabilistic latent variable model (LVM) — akin to factor analysis but deeper.

Random variables

- A **random variable*** (**RV**) X (with sample space Ω) has a value we are unsure about, maybe because (a) it is decided by some random process or (b) it is hidden from us.

- Types of sample spaces Ω:
  - Finite, e.g., { 0, 1 }, { red, blue, green }.
  - Countable, e.g., $\mathbb{Z}_{\geq 0}$
  - Uncountable, e.g., $\mathbb{R}$

* This is a practical definition for the purposes of this course, not a formal definition based on measure theory.

Random variables

- RVs are typically written as capital letters, e.g., *X*, *Y*.
- Once the value of RV *X* has been "sampled", "selected", "instantiated", or "realized" (by a random number generator, generative process, God, etc.), it takes a specific value from the sample space.
- The values the RV can take are typically written as lower-case letters, e.g., *x*, *y*.

Random variables

- The probability that a random variable *X* takes a particular value is determined by a:

  - **Probability mass function** (**PMF**) for finite or countable sample spaces.

  - **Probability density function** (**PDF**) for uncountable sample spaces.

PMF

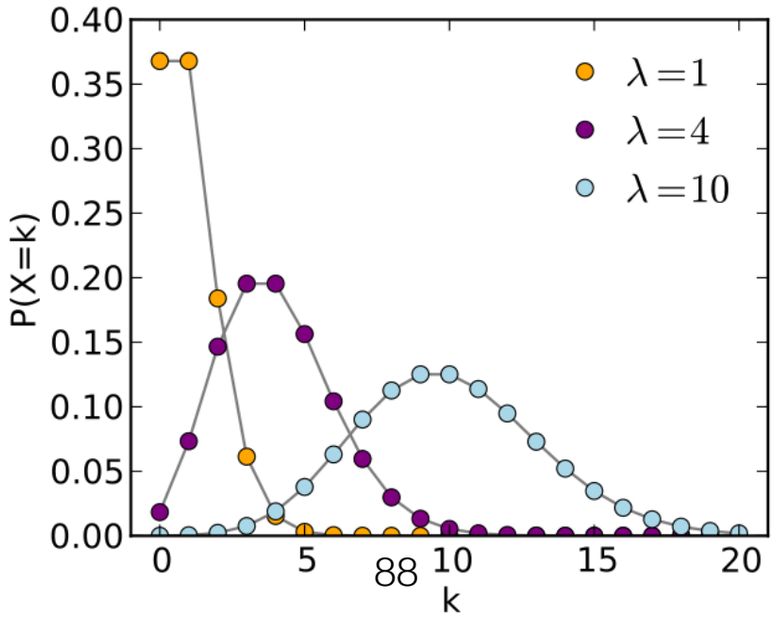
- Example 1 (finite): 
- Let RV *X* be the outcome of rolling a 6-sided die.
- If *X* is fair, then: $P(X = i) = \frac{1}{6} \quad \forall i \in \{1, \ldots, 6\}$

PMF

- Example 2 (countable): 
  - Let RV *X* be the number of TCP/IP packets that arrive in 1 second.
  - We can model the count of packets with a Poisson distribution:

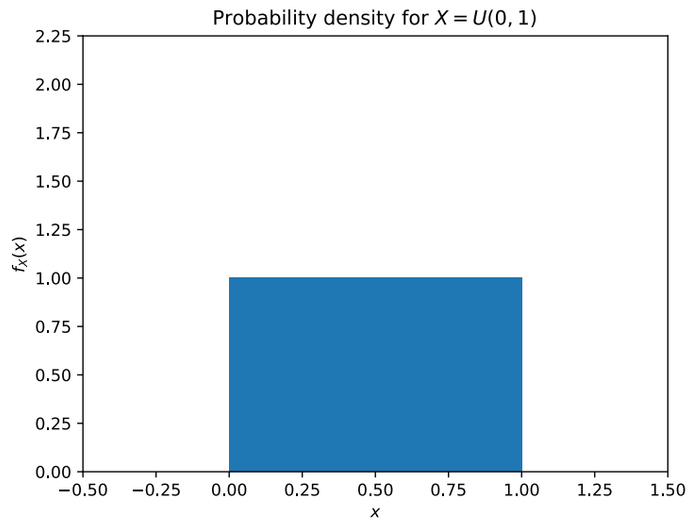
$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

where parameter λ specifies the *rate* of the packet arrivals.



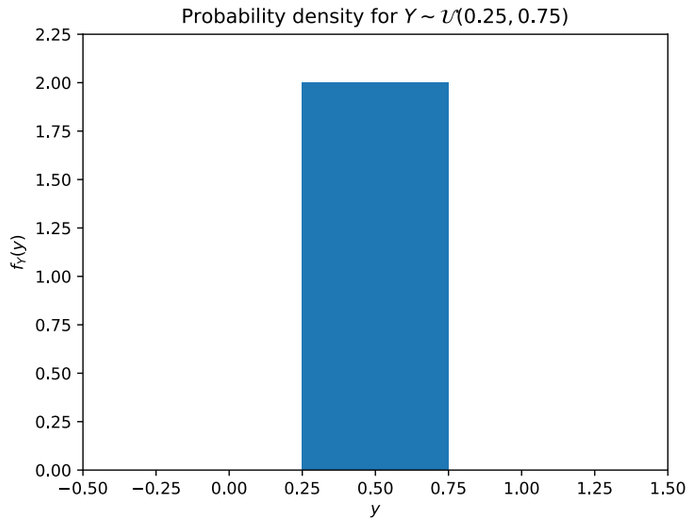
PDF

- Example 1:
  - Let *X* be a uniformly-distributed RV over Ω=[0,1].
    - Then $f_X(x) = 1 \quad \forall x \in \Omega$



- Example 2:
  - Let *Y* be a uniformly-distributed RV over Ω=[1/4, 3/4].
    - Then $f_Y(y) = 2 \quad \forall y \in \Omega$

Note that the PDF can exceed 1.

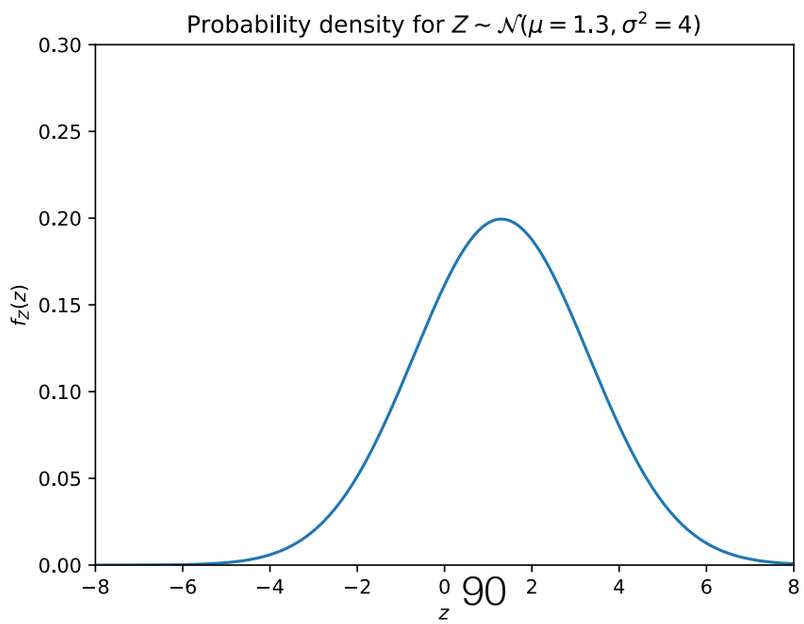


PDF

- Example 3:
  - Let Z be a **normally** (aka **Gaussian**) distributed RV with mean 1.5 and variance 4, i.e., $Z \sim \mathcal{N}(z; \mu = 1.5, \sigma^2 = 4)$.
  - Then

$$f_Z(z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right)$$

where μ is the mean and σ^2 is the variance.



Probability distributions

- In this course, we will relax notation and use “probability distribution” to mean either the PDF or PMF of a RV (as appropriate).

- As a notational shortcut, we use $P(x)$ to mean $P(X=x)$ or $f_X(x)$.

Joint probability distributions

- For multiple random variables *X*, *Y*, …, we can construct the joint probability distribution *P*(*x*, *y*, …) to mean the probability that (*X*=*x*) ^ (*Y*=*y*) ^ ….

- Note that *P* must still sum to 1 over all possible joint values (*x*, *y*, …).

Joint probability distributions

- Example in 2-D — crayons:

  - Let *X* be the color (red, blue, green, white).
  - Let *Y* be the intensity (low, medium, high).

	Red	Blue	Green	White
Low	0.1	0.05	0.025	0.2
Med	0.075	0.05	0.1	0
High	0.25	0.05	0.075	0.025

https://www.nytimes.com/2017/03/31/business/crayon-dandelion-crayola.html

Joint probability distributions

- Exercise:
  - What is the overall probability of obtaining a white crayon?

	Red	Blue	Green	White
Low	0.1	0.05	0.025	0.2
Med	0.075	0.05	0.1	0
High	0.25	0.05	0.075	0.025

https://www.nytimes.com/2017/03/31/business/crayon-dandelion-crayola.html

Joint probability distributions

- Example in 2-D — crayons:

  - From the joint distribution we can compute the **marginal distributions** $P(x)$ and $P(y)$.

$$P(x) = \sum_y P(x, y) \quad P(y) = \sum_x P(x, y)$$

	Red	Blue	Green	White	***P(y)***
Low	0.1	0.05	0.025	0.2	**0.375**
Med	0.075	0.05	0.1	0	**0.225**
High	0.25	0.05	0.075	0.025	**0.4**
P(x)	**0.425**	**0.15**	**0.2**	**0.225**	

https://www.nytimes.com/2017/03/31/business/crayon-dandelion-crayola.html

Law of total probability

- This is also called the **law of total probability**:
  - For any RVs *X* and *Y*:

$$P(x) = \sum_y P(x, y)$$

https://www.nytimes.com/2017/03/31/business/crayon-dandelion-crayola.html

Joint probability distributions

- In machine learning, we often use joint distributions of many variables that are part of a collection, e.g.:
  - Sequence (W_1, W_2, …, W_T) of words in a sentence.
    - W_t is the t^{th} RV in the sequence.
  - Grid (I_{11}, …, I_{1M}, …, I_{N1}, …, I_{NM}) of the pixels in an N x M image.
    - I_{ij} is the RV corresponding to location (i, j).

Conditional probability distributions

- Sometimes the value of one RV is predictive of the value of another RV.

- Examples:

  - If I know a person's height *H*, then I have some information about their weight *W*.

  - If I know how much cholesterol *C* a person eats, then I have some information about their chance of coronary heart disease *D*.

Conditional probability distributions

- We can form a **conditional probability distribution** of RV *X* **given** the value of RV *Y*:

$$P(x \mid y)$$

- Examples:
  - Height given weight: *P*(*h* | *w*)
  - Heart disease given cholesterol: *P*(*d* | *c*)

Conditional probability distributions

- We can form a **conditional probability distribution** of RV *X* **given** the value of RV *Y*:

$$P(x \mid y)$$

"given"

- Examples:
  - Height given weight: $P(h \mid w)$
  - Heart disease given cholesterol: $P(d \mid c)$

Conditional probability distributions

- More generally, we can form a conditional probability distribution of X_1, …, X_n given the values of Y_1, …, Y_m:

$$P(x_1, \ldots, x_n \mid y_1, \ldots, y_m)$$

Conditional probability distributions

- A conditional probability distribution is related to the joint probability distribution as follows:

$$P(x \mid y)P(y) = P(x, y)$$

- It follows that:

$$P(x \mid y, z)P(y \mid z) = P(x, y \mid z)$$

Conditional probability distributions

- More generally:

$$P(x_1, \ldots, x_n \mid y_1, \ldots, y_m) P(y_1, \ldots, y_m) = P(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

- And also:

$$P(x_1, \ldots, x_n \mid y_1, \ldots, y_m, z_1, \ldots, z_p) P(y_1, \ldots, y_m \mid z_1, \ldots, z_p) = P(x_1, \ldots, x_n, y_1, \ldots, y_m \mid z_1, \ldots, z_p)$$

Conditional probability distributions

- Note that the same joint probability can be factored in different ways, e.g.:

$$P(x, y, z) = P(x, y \mid z)P(z)$$
$$= P(x \mid y, z)P(y, z)$$

Conditional probability distributions

- Exercises:

1. $P(a, b, c, d) = P(a, c) * ?$
2. $P(w_1, w_2, w_3) = P(w_3 \mid w_1) * ? * ?$
3. $P(x_1, x_2, x_3) = P(x_1) * ? * P(x_3 \mid x_1, x_2)$
4. $P(x_1, \ldots, x_n) = P(x_1) * ? * ? * \ldots * ?$

n-1 terms

Conditional probability distributions

- Exercises:

1. $P(a, b, c, d) = P(a, c) * P(b, d \mid a, c)$
2. $P(w_1, w_2, w_3) = P(w_3 \mid w_1) * P(w_1) * P(w_2 \mid w_1, w_3)$
3. $P(x_1, x_2, x_3) = P(x_1) * P(x_2 \mid x_1) * P(x_3 \mid x_1, x_2)$
4. $P(x_1, \ldots, x_n) = P(x_1) * \prod_{i=2}^{n} P(x_i \mid x_1, \ldots, x_{i-1})$

Independence

- RVs *X* and *Y* are independent iff $P(x, y) = P(x)P(y)\ \forall x, y$, i.e., the joint distribution equals the product of the marginal distributions.

- Note that this implies that $P(x \mid y) = P(x)$ and $P(y \mid x) = P(y)$ since $P(x, y) = P(x \mid y)P(y) = P(y \mid x)P(x)$ by definition of conditional probability.

Conditional independence

- RVs *X* and *Y* are conditionally independent given RV *Z* iff:

$$P(x, y \mid z) = P(x \mid z)P(y \mid z) \ \forall x, y, z$$

- Note that this implies:

$$P(x \mid y, z) = P(x \mid z)$$

- In words: “If I already know the value of *Z*, then knowing *Y* tells me nothing *further* about *X*”.

Conditional independence

- More generally: $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$ are conditionally independent given $Z_1, \ldots, Z_p$ iff:

$$P(x_1, \ldots, x_n, y_1, \ldots, y_m \mid z_1, \ldots, z_p)$$
$$= P(x_1, \ldots, x_n \mid z_1, \ldots, z_p) P(y_1, \ldots, y_m \mid z_1, \ldots, z_p)$$

Bayes' rule

- It is often useful to compute $P(x \mid y)$ in terms of $P(y \mid x)$.
  - For example, if X represents a student's skill level, and Y is their test score, it's often easier to compute $P(y \mid x)$. But given a student's test score Y, we really want to know $P(x \mid y)$.
- Bayes' rule:

$$P(x \mid y) = \frac{P(x, y)}{P(y)} = \frac{P(y|x)P(x)}{P(y)}$$

Bayes' rule

- We can also generalize Bayes' rule to cases where we always condition on a tertiary variable Z:

$$P(x \mid y, z) = \frac{P(y \mid x, z)P(x \mid z)}{P(y|z)}$$

Probabilistic inference

Probabilistic graphical models

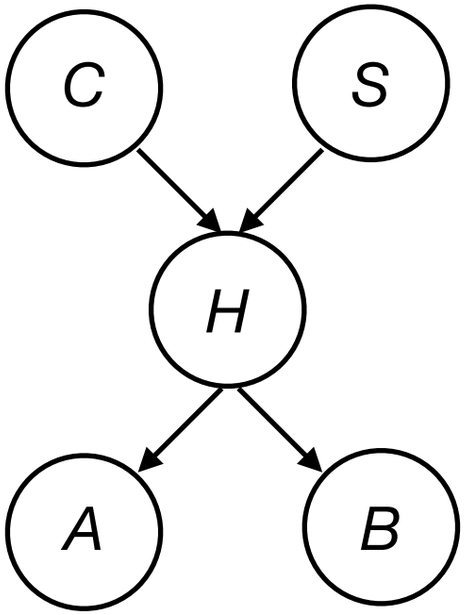
- To express the conditional independence relationships between multiple RVs, it is useful to represent their dependencies in a graph.

- A formal theory of probabilistic graphical models (Pearl 1998) has been devised.

    - Conditional independence can be determined via the principle of **d-separation** (beyond the scope of this course).

Probabilistic graphical models

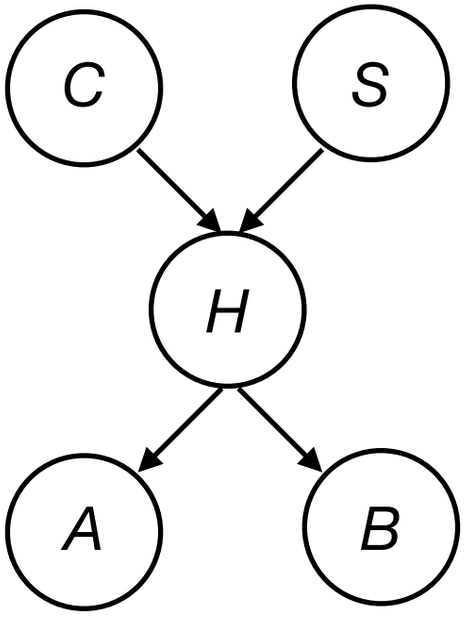
- Example 1 — medical diagnosis:



- *C*: whether the patient eats **c**aviar.
- *S*: the patient's **s**ex
- *H*: whether the patient has **h**igh cholesterol
- *A*: whether the patient will have a heart **a**ttack.
- *B*: whether the patient has shortness of **b**reath.

Probabilistic graphical models

- Example 1 — medical diagnosis:



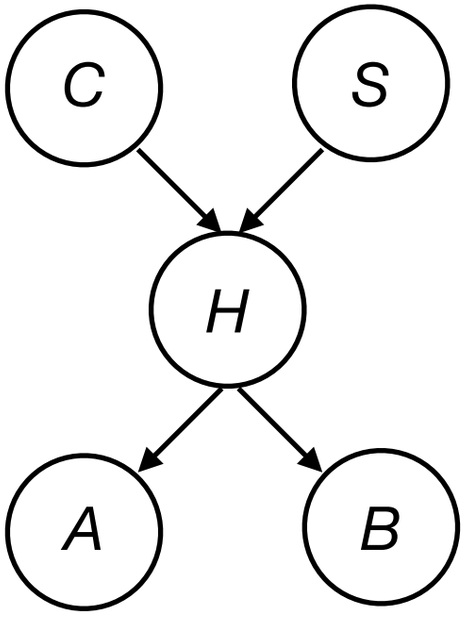
- This model implies that:

$$P(a, b \mid h, c, s) = P(a, b \mid h) \text{ and}$$

$$P(c, s \mid h, a, b) = P(c, s \mid h)$$

Probabilistic graphical models

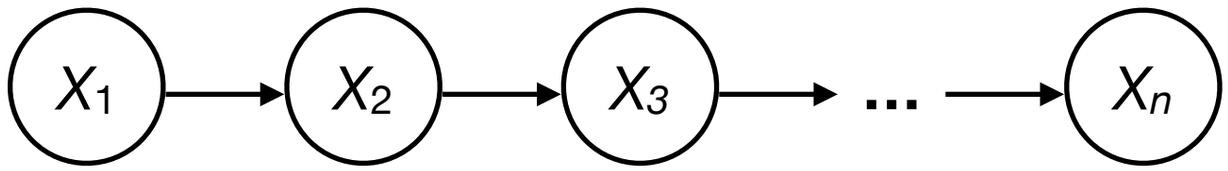
- Example 1 — medical diagnosis:



- In words, “If I want to know the probability the patient will have a heart attack *A*, and I already know the patient has high cholesterol *H*, then the patient’s sex and whether she/he eats caviar *C* is irrelevant.”

Probabilistic graphical models

- Example 2 — Markov chain:



- This chain-like model of X_1, …, X_n implies that:

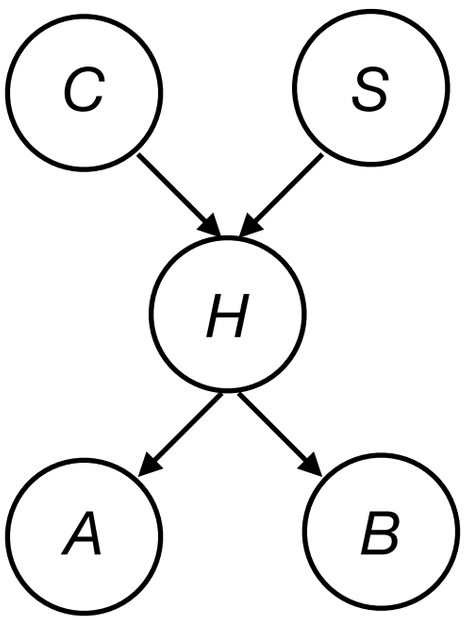
$$P(x_i \mid x_1, \ldots, x_{i-1}) = P(x_i \mid x_{i-1}) \text{ and}$$
$$P(x_i \mid x_{i+1}, \ldots, x_n) = P(x_i \mid x_{i+1})$$

- In words, “If I want to know the value of X_i and I already know $X_{i\text{-}1}$, then the values of any ‘earlier’ X’s are irrelevant.”

Probabilistic inference

- Given a model with multiple RVs and how they are related to each other, we can **infer** the values of other RVs.

- For the medical diagnosis example, suppose we knew the conditional probability distributions:



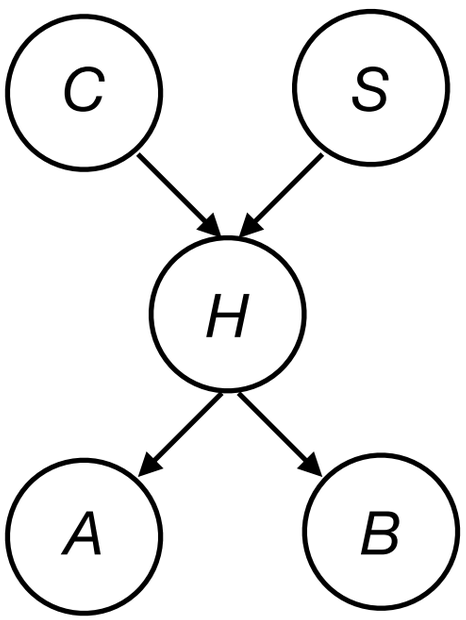
$P(H=0)$		
	$C=0$	$C=1$
S=Ma		
S=Fe		

$P(H=1)$		
	$C=0$	$C=1$
S=Ma	0.3	0.6
S=Fe	0.2	0.25

Probabilistic inference

- Given a model with multiple RVs and how they are related to each other, we can **infer** the values of other RVs.

- For the medical diagnosis example, suppose we knew the conditional probability distributions:



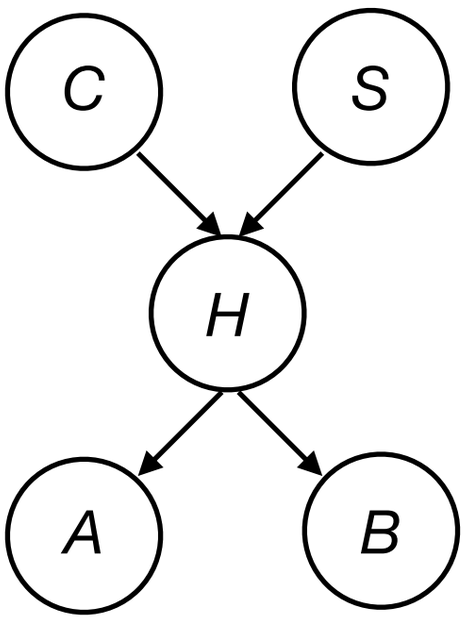
$P(H=0)$	$C=0$	$C=1$
S=Ma	0.7	0.4
S=Fe	0.8	0.75

$P(H=1)$	$C=0$	$C=1$
S=Ma	0.3	0.6
S=Fe	0.2	0.25

Probabilistic inference

- Given a model with multiple RVs and how they are related to each other, we can **infer** the values of other RVs.

- For the medical diagnosis example, suppose we knew the conditional probability distributions:



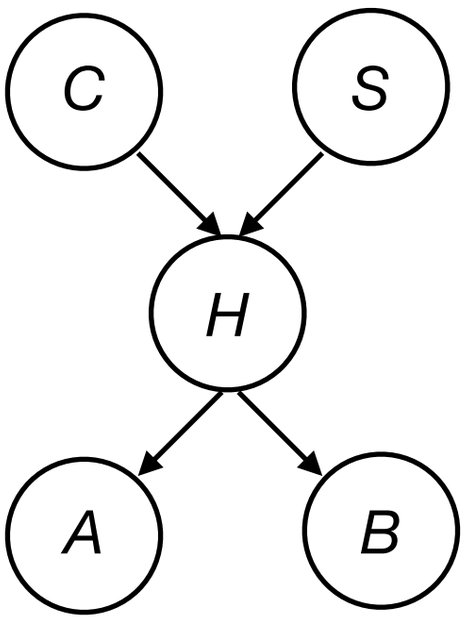
$P(A=0)$	
$H=0$	0.75
$H=1$	0.2

$P(A=1)$	
$H=0$	0.25
$H=1$	0.8

Probabilistic inference

- Given a model with multiple RVs and how they are related to each other, we can **infer** the values of other RVs.

- For the medical diagnosis example, suppose we knew the conditional probability distributions:



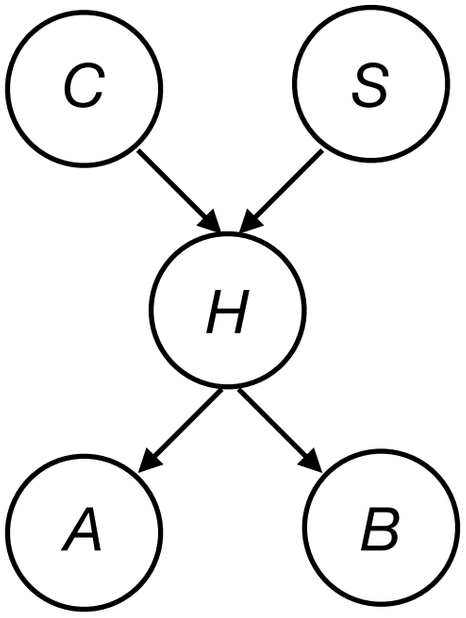
$P(B=0)$	
$H=0$	0.9
$H=1$	0.1

$P(B=1)$	
$H=0$	0.1
$H=1$	0.9

Probabilistic inference

- Suppose we meet a male patient who eats caviar.

- What is the **posterior probability** that H=1, i.e., $P(H=1 \mid C=1, S=\text{Ma})$? (*Posterior* means after observing C, S.)



$P(H=0)$

	C=0	C=1
S=Ma	0.7	0.4
S=Fe	0.8	0.75

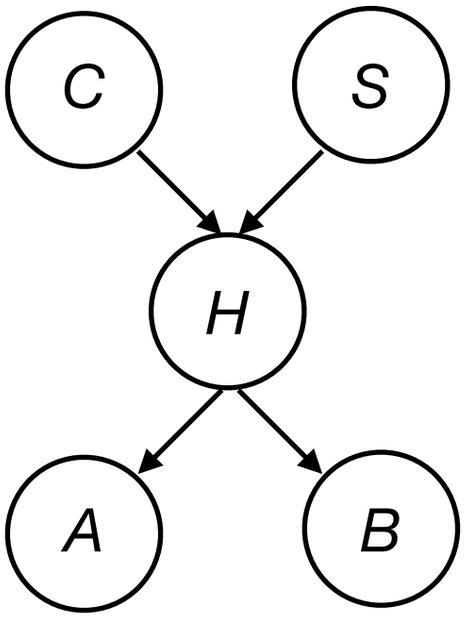
$P(H=1)$

	C=0	C=1
S=Ma	0.3	0.6
S=Fe	0.2	0.25

Probabilistic inference

- Suppose we meet a male patient who eats caviar.

- What is the **posterior probability** that H=1, i.e., $P(H=1 \mid C=1, S=\text{Ma})$? (*Posterior* means after observing C, S.)



$P(H=0)$

	C=0	C=1
S=Ma	0.7	0.4
S=Fe	0.8	0.75

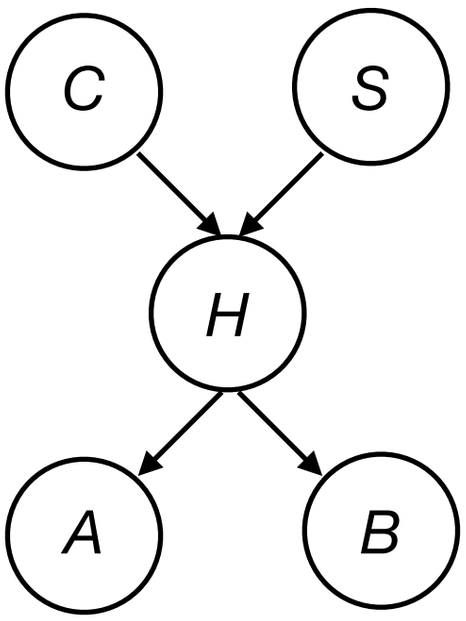
$P(H=1)$

	C=0	C=1
S=Ma	0.3	**0.6**
S=Fe	0.2	0.25

Just consult the conditional probability distribution.

Probabilistic inference

- What if we also know that the patient is short of breath?



P(*H*=0)

	C=0	*C*=1
S=Ma	0.7	0.4
S=Fe	0.8	0.75

P(*H*=1)

	C=0	*C*=1
S=Ma	0.3	**0.6**
S=Fe	0.2	0.25

P(*B*=0)

H=0	0.9
H=1	0.1

P(*B*=1)

H=0	0.1
H=1	0.9

Conditional independence from the graphical model:

$P(b \mid h, c, s) = P(b \mid h)$

Probabilistic inference

$$P(H = 1 \mid C = 1, S = \text{Ma}, B = 1)$$

$$= \frac{P(B = 1 \mid H = 1, C = 1, S = \text{Ma}) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Bayes' rule

$P(H=0)$	$C=0$	$C=1$
S=Ma	0.7	0.4
S=Fe	0.8	0.75

$P(H=1)$	$C=0$	$C=1$
S=Ma	0.3	**0.6**
S=Fe	0.2	0.25

$P(B=0)$	
H=0	0.9
H=1	0.1

$P(B=1)$	
H=0	0.1
H=1	0.9

Conditional independence from the graphical model:

$$P(b \mid h, c, s) = P(b \mid h)$$

Probabilistic inference

$$P(H = 1 \mid C = 1, S = \text{Ma}, B = 1)$$

$$= \frac{P(B = 1 \mid H = 1, C = 1, S = \text{Ma}) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Bayes' rule

$$= \frac{P(B = 1 \mid H = 1) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Conditional independence

$P(H=0)$	$C=0$	$C=1$
S=Ma	0.7	0.4
S=Fe	0.8	0.75

$P(H=1)$	$C=0$	$C=1$
S=Ma	0.3	**0.6**
S=Fe	0.2	0.25

$P(B=0)$	
H=0	0.9
H=1	0.1

$P(B=1)$	
H=0	0.1
H=1	0.9

Conditional independence from the graphical model:

$$P(b \mid h, c, s) = P(b \mid h)$$

Probabilistic inference

$$P(H = 1 \mid C = 1, S = \text{Ma}, B = 1)$$

$$= \frac{P(B = 1 \mid H = 1, C = 1, S = \text{Ma}) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Bayes' rule

$$= \frac{P(B = 1 \mid H = 1) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Conditional independence

$$= \frac{0.9 * 0.6}{\sum_{h=0}^{1} P(B = 1, H = h \mid C = 1, S = \text{Ma})}$$

Law of total probability

P(*H*=0)

	C=0	*C*=1
S=Ma	0.7	0.4
S=Fe	0.8	0.75

P(*H*=1)

	C=0	*C*=1
S=Ma	0.3	**0.6**
S=Fe	0.2	0.25

P(*B*=0)

H=0	0.9
H=1	0.1

P(*B*=1)

H=0	0.1
H=1	0.9

Conditional independence from the graphical model:

$P(b \mid h, c, s) = P(b \mid h)$

Probabilistic inference

$$P(H = 1 \mid C = 1, S = \text{Ma}, B = 1)$$

$$= \frac{P(B = 1 \mid H = 1, C = 1, S = \text{Ma}) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Bayes' rule

$$= \frac{P(B = 1 \mid H = 1) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Conditional independence

$$= \frac{0.9 * 0.6}{\sum_{h=0}^{1} P(B = 1, H = h \mid C = 1, S = \text{Ma})}$$

Law of total probability

$$= \frac{0.54}{\sum_{h=0}^{1} P(B = 1 \mid H = h, C = 1, S = \text{Ma}) P(H = h \mid C = 1, S = \text{Ma})}$$

Def. of cond. prob.

Probabilistic inference

$$P(H = 1 \mid C = 1, S = \text{Ma}, B = 1)$$

$$= \frac{P(B = 1 \mid H = 1, C = 1, S = \text{Ma}) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Bayes' rule

$$= \frac{P(B = 1 \mid H = 1) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Conditional independence

$$= \frac{0.9 * 0.6}{\sum_{h=0}^{1} P(B = 1, H = h \mid C = 1, S = \text{Ma})}$$

Law of total probability

$$= \frac{0.54}{\sum_{h=0}^{1} P(B = 1 \mid H = h, C = 1, S = \text{Ma}) P(H = h \mid C = 1, S = \text{Ma})}$$

Def. of cond. prob.

$$= \frac{0.54}{\sum_{h=0}^{1} P(B = 1 \mid H = h) P(H = h \mid C = 1, S = \text{Ma})}$$

Conditional independence

Probabilistic inference

$$P(H = 1 \mid C = 1, S = \text{Ma}, B = 1)$$

$$= \frac{P(B = 1 \mid H = 1, C = 1, S = \text{Ma}) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Bayes' rule

$$= \frac{P(B = 1 \mid H = 1) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$

Conditional independence

$$= \frac{0.9 * 0.6}{\sum_{h=0}^{1} P(B = 1, H = h \mid C = 1, S = \text{Ma})}$$

Law of total probability

$$= \frac{0.54}{\sum_{h=0}^{1} P(B = 1 \mid H = h, C = 1, S = \text{Ma}) P(H = h \mid C = 1, S = \text{Ma})}$$

Def. of cond. prob.

$$= \frac{0.54}{\sum_{h=0}^{1} P(B = 1 \mid H = h) P(H = h \mid C = 1, S = \text{Ma})}$$

Conditional independence

$$= \frac{0.54}{0.1 * 0.4 + 0.9 * 0.6}$$

Probabilistic inference

$$P(H = 1 \mid C = 1, S = \text{Ma}, B = 1)$$

$$= \frac{P(B = 1 \mid H = 1, C = 1, S = \text{Ma}) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$ **Bayes' rule**

$$= \frac{P(B = 1 \mid H = 1) P(H = 1 \mid C = 1, S = \text{Ma})}{P(B = 1 \mid C = 1, S = \text{Ma})}$$ **Conditional independence**

$$= \frac{0.9 * 0.6}{\sum_{h=0}^{1} P(B = 1, H = h \mid C = 1, S = \text{Ma})}$$ **Law of total probability**

$$= \frac{0.54}{\sum_{h=0}^{1} P(B = 1 \mid H = h, C = 1, S = \text{Ma}) P(H = h \mid C = 1, S = \text{Ma})}$$ **Def. of cond. prob.**

$$= \frac{0.54}{\sum_{h=0}^{1} P(B = 1 \mid H = h) P(H = h \mid C = 1, S = \text{Ma})}$$ **Conditional independence**

$$= \frac{0.54}{0.1 * 0.4 + 0.9 * 0.6}$$

$$= \frac{0.54}{0.04 + 0.54}$$

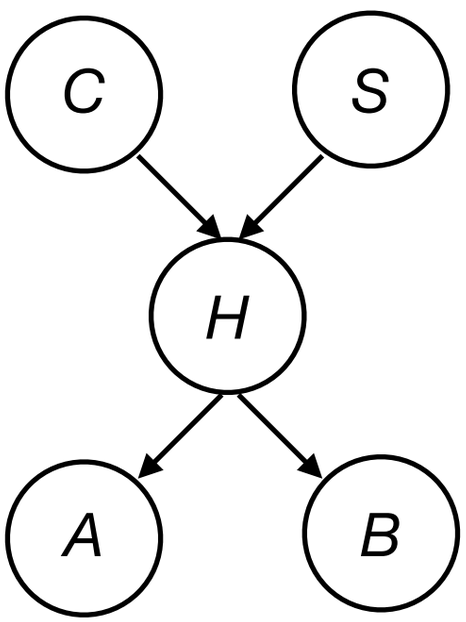
$$\approx 0.93$$

Maximum likelihood estimation (MLE)

Parameters in probability distributions

- Most probabilistic models have parameters we want to estimate.

- For example, the conditional probabilities for medical diagnosis are all parameters that must be learned.



$P(H=0)$

	$C=0$	$C=1$
S=Ma	0.7	0.4
S=Fe	0.8	0.75

$P(H=1)$

	$C=0$	$C=1$
S=Ma	0.3	0.6
S=Fe	0.2	0.25

$P(B=0)$

$H=0$	0.9
$H=1$	0.1

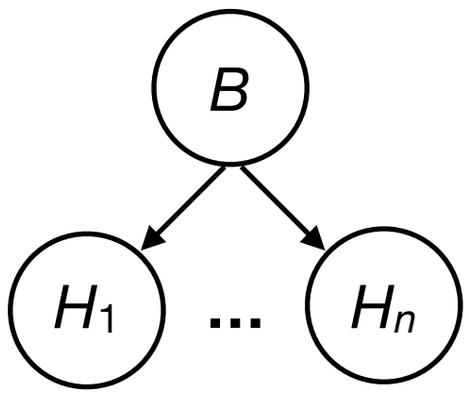
$P(B=1)$

$H=0$	0.1
$H=1$	0.9

Parameters in probability distributions

- Most probabilistic models have parameters we want to estimate.

- As another example, we might want to estimate the bias B of a coin after observing n coin flips $H_1, \ldots, H_n$:



$$P(H_i = 1 \mid b) = b$$

Conditional independence:

$$P(h_i \mid b, h_1, \ldots, h_{i-1}, h_{i+1}, \ldots, h_n) = \mathrm{P}(h_i \mid b)$$

- What is a principled approach to estimating B?

Maximum likelihood estimation (MLE)

- Maximum likelihood estimation (MLE):
  - The value of a latent variable is estimated as the one that makes the observed data as likely (probable) as possible.
- The **likelihood** of H_1, …, H_n given B is:

$$P(h_1, \ldots, h_n \mid b) = P(h_1 \mid b) \prod_{i=2}^{n} P(h_i \mid b, h_1, \ldots, h_{i-1})$$

Maximum likelihood estimation (MLE)

- Maximum likelihood estimation (MLE):
  - The value of a latent variable is estimated as the one that makes the observed data as likely (probable) as possible.

- The **likelihood** of $H_1, \ldots, H_n$ given B is:

$$P(h_1, \ldots, h_n \mid b) = P(h_1 \mid b) \prod_{i=2}^{n} P(h_i \mid b, h_1, \ldots, h_{i-1})$$

$$= P(h_1 \mid b) \prod_{i=2}^{n} P(h_i \mid b)$$

Conditional independence

Maximum likelihood estimation (MLE)

- Maximum likelihood estimation (MLE):
  - The value of a latent variable is estimated as the one that makes the observed data as likely (probable) as possible.

- The **likelihood** of $H_1, \ldots, H_n$ given B is:

$$P(h_1, \ldots, h_n \mid b) = P(h_1 \mid b) \prod_{i=2}^{n} P(h_i \mid b, h_1, \ldots, h_{i-1})$$

$$= P(h_1 \mid b) \prod_{i=2}^{n} P(h_i \mid b) \quad \textbf{Conditional independence}$$

$$= \prod_{i=1}^{n} P(h_i \mid b)$$

Maximum likelihood estimation (MLE)

- We can express the probability of each h_i given b as:

$$P(h_i \mid b) = b^{h_i}(1-b)^{1-h_i}$$

Maximum likelihood estimation (MLE)

- We can express the probability of each h_i given b as:

$$P(h_i \mid b) = b^{h_i}(1-b)^{1-h_i}$$

$$= b \text{ if } h_i = 1 \quad \text{or}$$

$$(1-b) \text{ if } h_i = 0$$

The exponent "chooses" the correct probability for H_i=1 or H_i=0.

Maximum likelihood estimation (MLE)

- We seek to maximize the probability of $h_1, \ldots, h_n$ by optimizing b.

- It's often easier instead to optimize the **log-likelihood**.

$$\arg\max_b P(h_1, \ldots, h_n \mid b) = \arg\max_b \log P(h_1, \ldots, h_n \mid b)^*$$

Maximum likelihood estimation (MLE)

- We seek to maximize the probability of h_1, …, h_n by optimizing b.

- It's often easier instead to optimize the **log-likelihood**.

$$\arg\max_b P(h_1, \ldots, h_n \mid b) = \arg\max_b \log P(h_1, \ldots, h_n \mid b)^*$$

* assuming the probability is never exactly 0.

Maximum likelihood estimation (MLE)

- We seek to maximize the probability of h_1, …, h_n by optimizing b.

- It's often easier instead to optimize the **log-likelihood**.

$$\arg\max_b P(h_1, \ldots, h_n \mid b) = \arg\max_b \log P(h_1, \ldots, h_n \mid b)$$

$$\log P(h_1, \ldots, h_n \mid b) = \log \prod_{i=1}^{n} P(h_i \mid b)$$ due to conditional independence

Maximum likelihood estimation (MLE)

- We seek to maximize the probability of h_1, …, h_n by optimizing b.

- It's often easier instead to optimize the **log-likelihood**.

$$\arg\max_b P(h_1, \ldots, h_n \mid b) = \arg\max_b \log P(h_1, \ldots, h_n \mid b)$$

$$\log P(h_1, \ldots, h_n \mid b) = \log \prod_{i=1}^{n} P(h_i \mid b)$$

$$= \sum_{i=1}^{n} \log P(h_i \mid b)$$

Maximum likelihood estimation (MLE)

- We seek to maximize the probability of h_1, …, h_n by optimizing b.

- It's often easier instead to optimize the **log-likelihood**.

$$\arg\max_b P(h_1, \ldots, h_n \mid b) = \arg\max_b \log P(h_1, \ldots, h_n \mid b)$$

$$\begin{aligned}
\log P(h_1, \ldots, h_n \mid b) &= \log \prod_{i=1}^{n} P(h_i \mid b) \\
&= \sum_{i=1}^{n} \log P(h_i \mid b) \\
&= \sum_{i=1}^{n} \log b^{h_i} (1-b)^{1-h_i}
\end{aligned}$$

Maximum likelihood estimation (MLE)

- We seek to maximize the probability of $h_1, \ldots, h_n$ by optimizing b.

- It's often easier instead to optimize the **log-likelihood**.

$$\arg\max_b P(h_1, \ldots, h_n \mid b) = \arg\max_b \log P(h_1, \ldots, h_n \mid b)$$

$$\begin{aligned}
\log P(h_1, \ldots, h_n \mid b) &= \log \prod_{i=1}^{n} P(h_i \mid b) \\
&= \sum_{i=1}^{n} \log P(h_i \mid b) \\
&= \sum_{i=1}^{n} \log b^{h_i} (1-b)^{1-h_i} \\
&= \sum_{i=1}^{n} h_i \log b + (1-h_i) \log(1-b)
\end{aligned}$$

Maximum likelihood estimation (MLE)

- We seek to maximize the probability of $h_1, \ldots, h_n$ by optimizing b.

- It's often easier instead to optimize the **log-likelihood**.

$$\arg\max_b P(h_1, \ldots, h_n \mid b) = \arg\max_b \log P(h_1, \ldots, h_n \mid b)$$

$$\begin{aligned}
\log P(h_1, \ldots, h_n \mid b) &= \log \prod_{i=1}^{n} P(h_i \mid b) \\
&= \sum_{i=1}^{n} \log P(h_i \mid b) \\
&= \sum_{i=1}^{n} \log b^{h_i} (1-b)^{1-h_i} \\
&= \sum_{i=1}^{n} h_i \log b + (1-h_i) \log(1-b) \\
&= n_1 \log b + (n-n_1) \log(1-b)
\end{aligned}$$

n_1 is number of heads.

Maximum likelihood estimation (MLE)

- We can now differentiate w.r.t. *b*, set to 0, and solve to obtain the MLE of *B*:

$$\nabla_b \left[n_1 \log b + (n - n_1) \log(1 - b) \right] = \frac{n_1}{b} - \frac{(n - n_1)}{1 - b}$$

$b = ?$

Maximum likelihood estimation (MLE)

- We can now differentiate w.r.t. *b*, set to 0, and solve to obtain the MLE of *B*:

$$\nabla_b \left[n_1 \log b + (n - n_1) \log(1 - b) \right] = \frac{n_1}{b} - \frac{(n - n_1)}{1 - b}$$

$$(1 - b)n_1 - b(n - n_1) = 0$$

$$n_1 - bn_1 - bn + bn_1 = 0$$

$$n_1 = bn$$

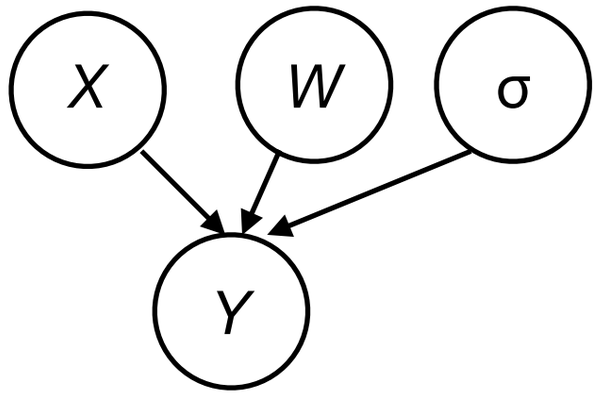
$$b = \frac{n_1}{n}$$

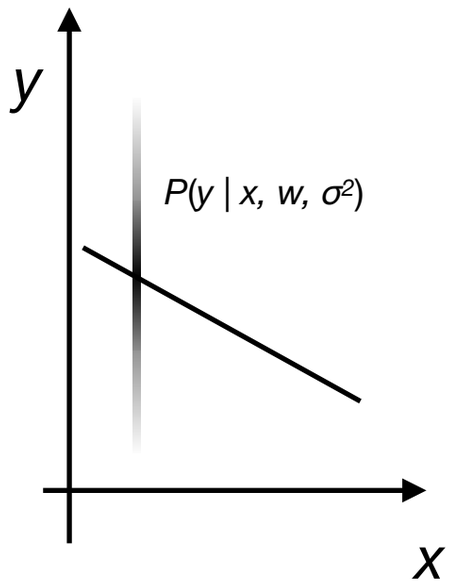
The MLE for *B* is the fraction of coin flips that are heads.

Linear-Gaussian models

Linear-Gaussian model

- Let's consider a different model that contains **real-valued** RVs (not just from a finite sample space).

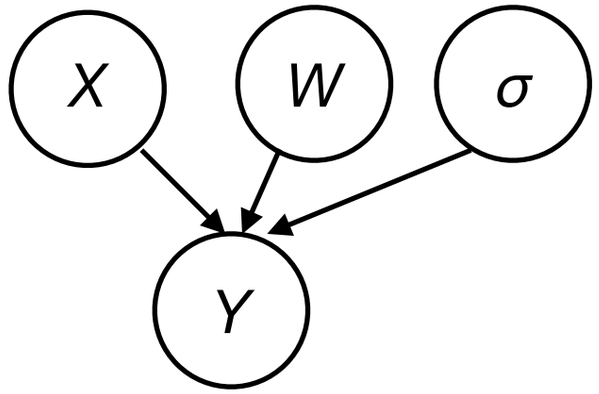


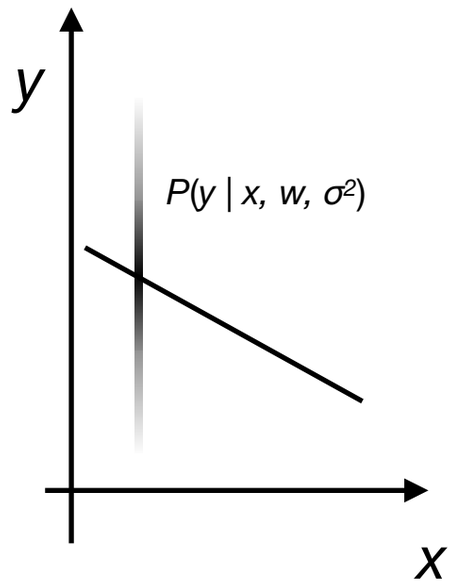


- *X* is some feature vector (e.g., face image).
- *Y* is some outcome variable (e.g., age).
- *W* is a vector of weights that characterize how *Y* is related to *X*.
- σ expresses how uncertain we are about *Y* after seeing *X*.

Linear-Gaussian model

- Suppose we model the relationship between X, W, σ, and Y such that:

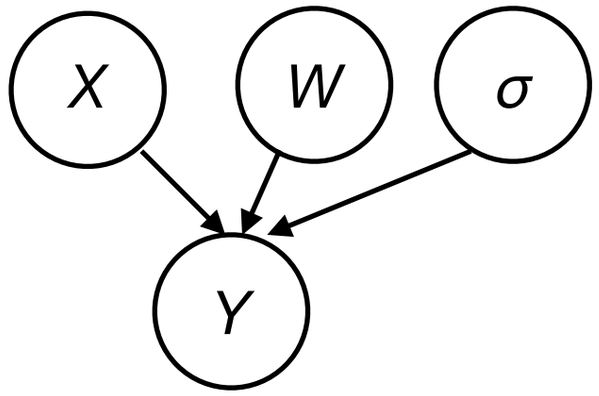




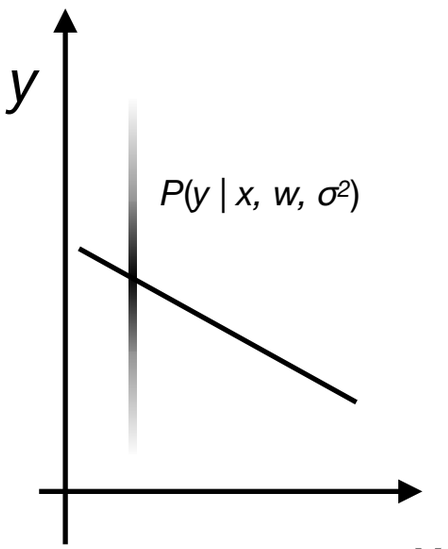
- Y is a normal/Gaussian random variable.
- The expected value of Y is $x^{\mathsf{T}}w$.
- The variance of Y is constant (σ^2) for all possible x.

Linear-Gaussian model

- Suppose we model the relationship between X, W, σ, and Y such that:



$$P(y \mid \mathbf{w}, \mathbf{x}) = \mathcal{N}(y; \mathbf{x}^\top \mathbf{w}, \sigma^2)$$



- If we collect a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$, what is the MLE for W and σ?

Linear-Gaussian model

$$P(y \mid \mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(y; \mathbf{x}^\top \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \mathbf{x}^\top \mathbf{w})^2}{2\sigma^2}\right)$$

Linear-Gaussian model

$$P(y \mid \mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(y; \mathbf{x}^\top \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \mathbf{x}^\top \mathbf{w})^2}{2\sigma^2}\right)$$

$$P(\mathcal{D} \mid \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2)$$

Conditional independence

Linear-Gaussian model

$$P(y \mid \mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(y; \mathbf{x}^\top \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \mathbf{x}^\top \mathbf{w})^2}{2\sigma^2}\right)$$

$$P(\mathcal{D} \mid \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2)$$

Conditional independence

$$\log P(\mathcal{D} \mid \mathbf{w}, \sigma^2) = \log \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2)$$

Linear-Gaussian model

$$P(y \mid \mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(y; \mathbf{x}^\top \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \mathbf{x}^\top \mathbf{w})^2}{2\sigma^2}\right)$$

$$P(\mathcal{D} \mid \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2)$$

Conditional independence

$$\log P(\mathcal{D} \mid \mathbf{w}, \sigma^2) = \log \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2)$$

$$= \sum_{i=1}^{n} \log P(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2)$$

Linear-Gaussian model

- MLE for **w**:

$$\mathbf{w} = \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} \mathbf{x}^{(i)\top} \right)^{-1} \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} y^{(i)} \right)$$

This is the same solution as for linear regression, but derived as the MLE of a probabilistic model (instead of the minimum MSE).

Linear-Gaussian model

- MLE for σ^2:

$$\sigma^2 = \frac{1}{n}\sum^{n}(\mathbf{x}^{(i)\top}\mathbf{w} - y^{(i)})^2$$
$$i{=}1$$

This is the sum of squared residuals of the predictions w.r.t. ground-truth.