CS/DS 541: Class 2

Jacob Whitehill

Linear regression (aka 2-layer NN)

Linear regression

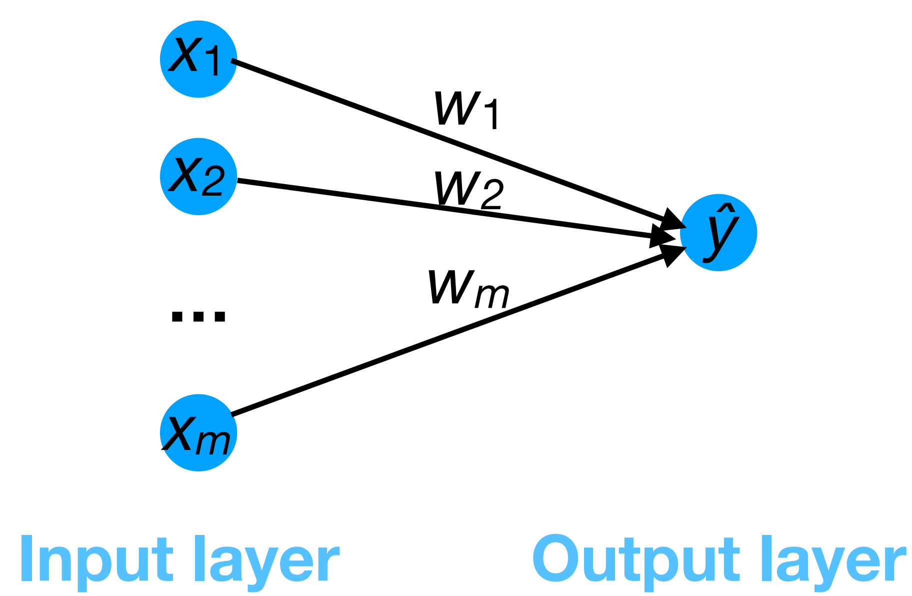
- Let dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
- Want to create a neural network — which we can also treat as a “machine” — to estimate each $y^{(i)}$ with high accuracy.
- Let us define the machine by a function g (with parameters $\mathbf{w}$) whose output $\hat{y}$ is linear 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

- 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:

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

Solving for **w**

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

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

Matrix notation

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

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

$$\mathbf{X} = \left[\begin{array}{ccc} | & & | \\ \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \\ | & & | \end{array} \right]$$

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

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

$$\mathbf{y} = \left[\begin{array}{c} y^{(1)} \\ \vdots \\ y^{(n)} \end{array} \right]$$

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

Matrix notation

- Using summation notation, we derived:

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

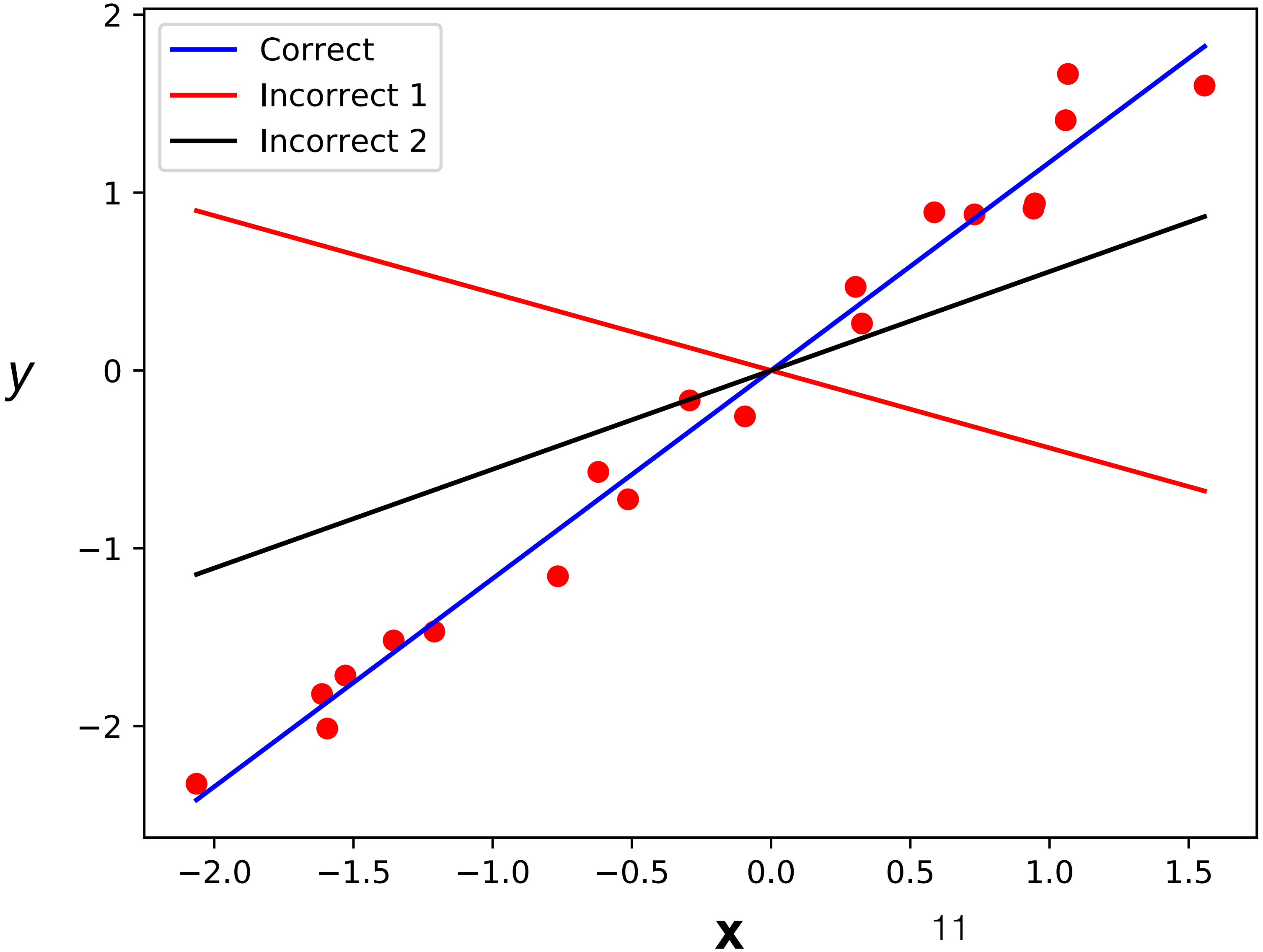
- Using matrix notation, we can write the solution as:

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

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

1-d example

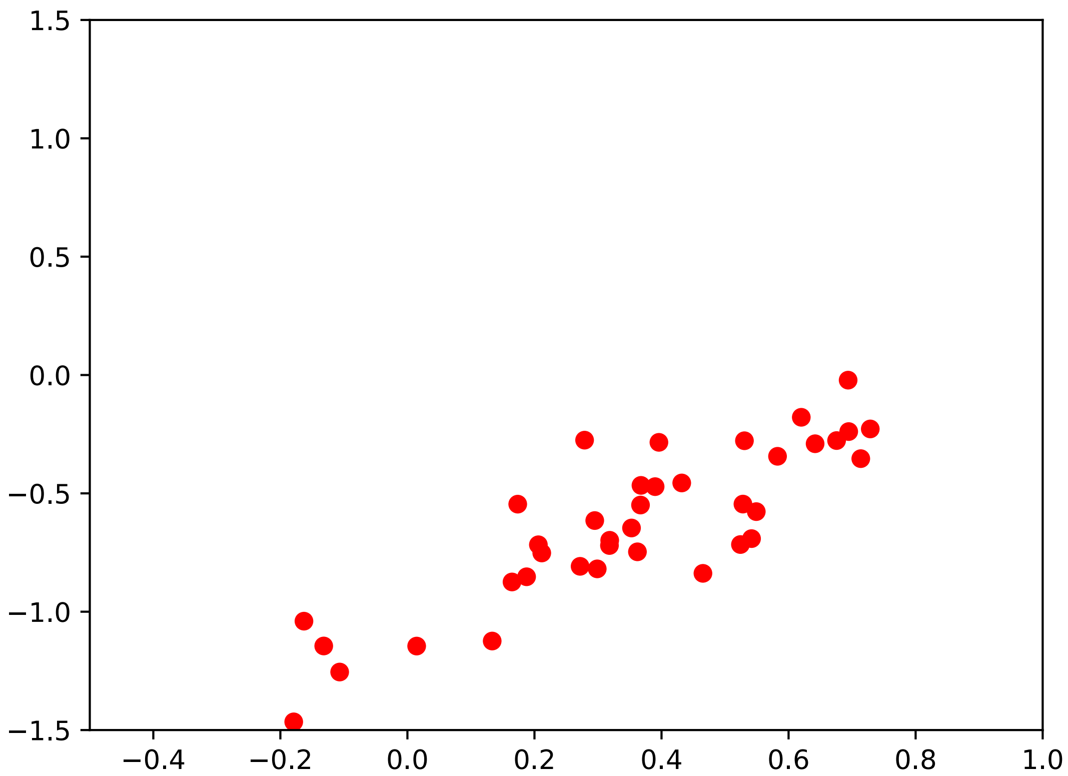
- Linear regression finds the weight vector **w** that minimizes the f_{MSE}. Here's an example where each **x** is just 1-d…



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

Exercise

- Suppose we use linear regression (as defined above) to model the relationship between *x* and *y*.



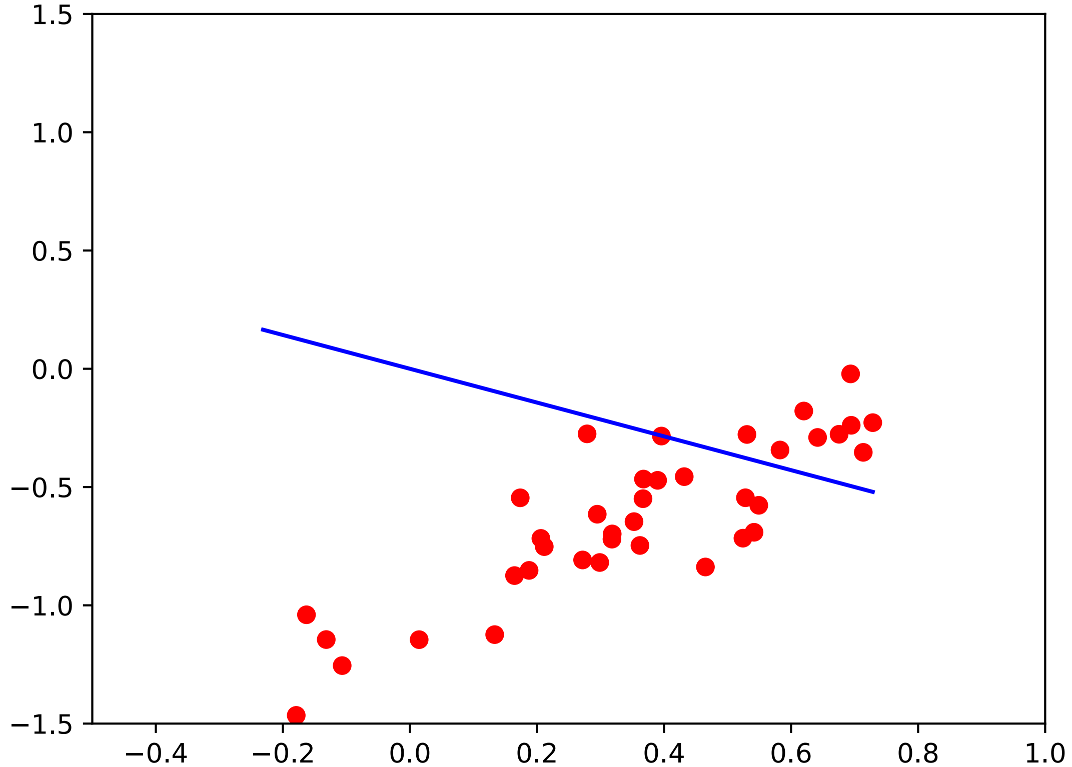
- What will be the sign of the regression weight *w* that is learned?

    1. *w* is positive
    2. *w* is 0
    3. *w* is negative

Jacob Whitehill, WPI

Exercise

- Suppose we use linear regression (as defined above) to model the relationship between x and y.



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.

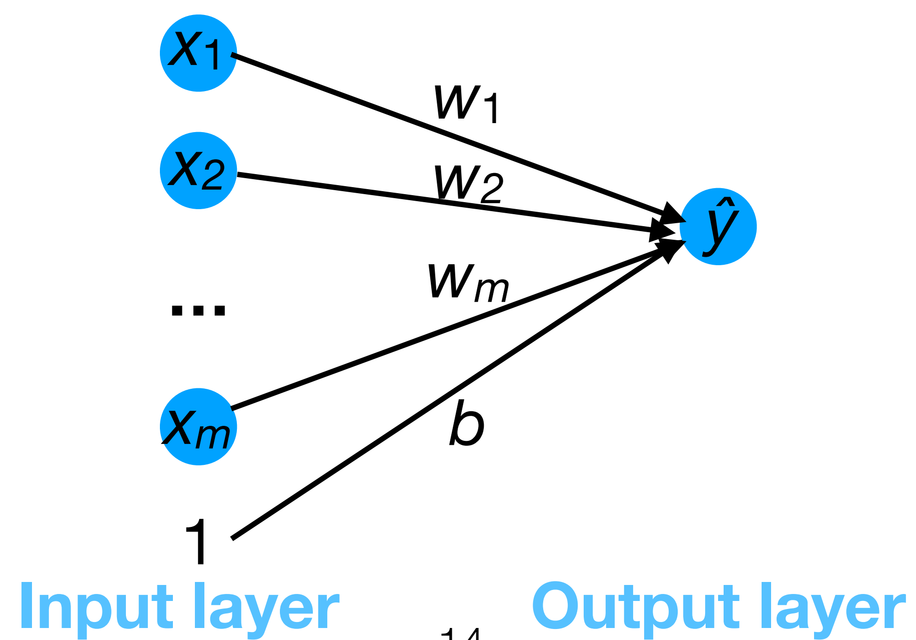
- What will be the sign of the regression weight w that is learned?

1. w is positive
2. w is 0
3. w is negative

Bias term

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

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



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

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

Bias term

- Alternatively, we can implicitly include a bias term by augmenting each input vector **x** with a 1 at the end:

$$\tilde{\mathbf{x}} = \left[\begin{array}{c} \mathbf{x} \\ 1 \end{array} \right]$$

- Correspondingly, our weight vector **w** will have an extra component (bias term) at the end.

$$\tilde{\mathbf{w}} = \left[\begin{array}{c} \mathbf{w} \\ b \end{array} \right]$$

Bias term

- To see why, notice that:

$$
\begin{aligned}
\hat{y} &= \tilde{\mathbf{x}}^\top \tilde{\mathbf{w}} \\
&= \left[\begin{array}{c} \mathbf{x} \\ 1 \end{array} \right]^\top \left[\begin{array}{c} \mathbf{w} \\ b \end{array} \right] \\
&= \left[\begin{array}{cc} \mathbf{x}^\top & 1 \end{array} \right] \left[\begin{array}{c} \mathbf{w} \\ b \end{array} \right] \\
&= \mathbf{x}^\top \mathbf{w} + b
\end{aligned}
$$

Bias term

- We can find the optimal **w** and *b* based on all the training data using matrix notation.

- First define an augmented design matrix:

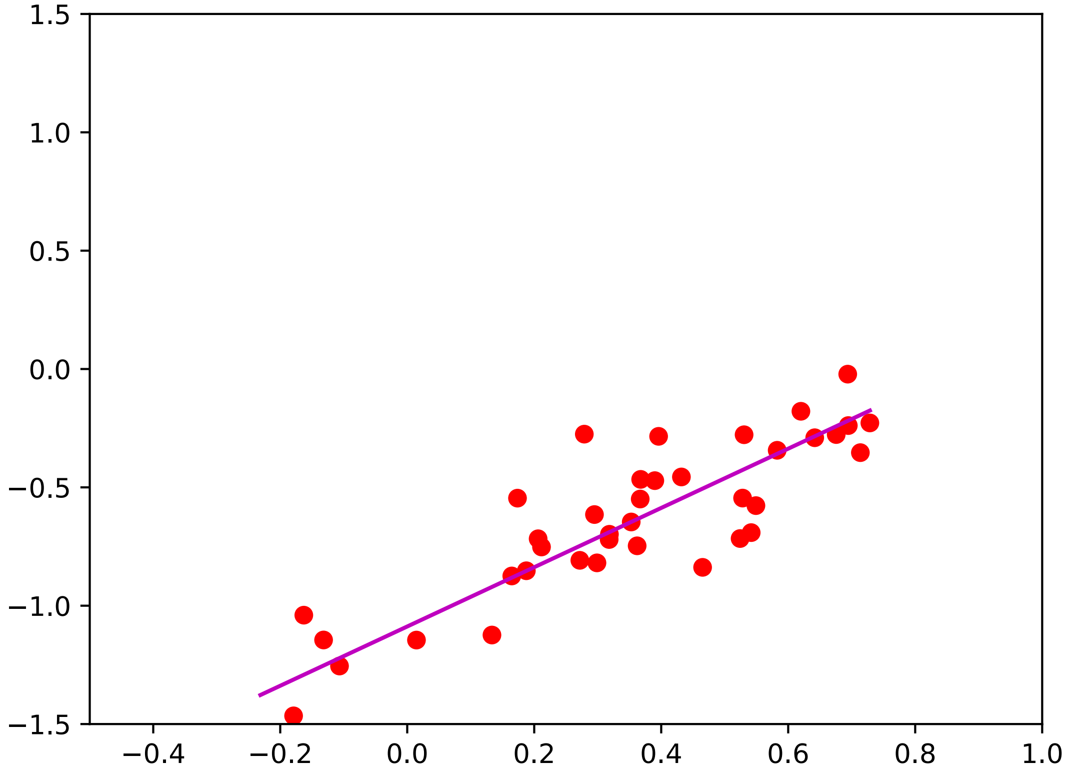
$$\tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \\ 1 & \dots & 1 \end{bmatrix}$$

- Then compute:

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

Bias term

- With this more powerful model, the regression line is now learned as desired:



Demo

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

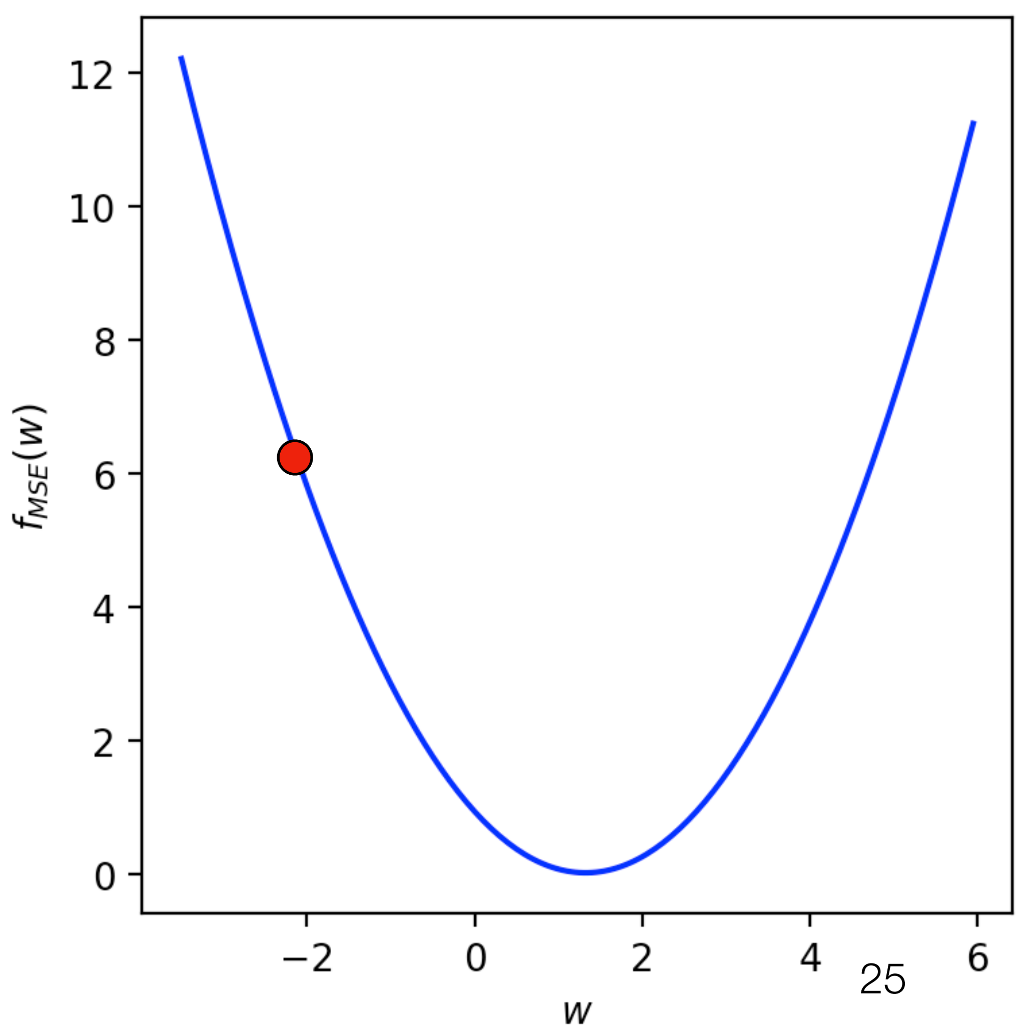
- Alternatively, linear regression can be solved numerically using gradient descent.
- **Numerical solution**: need to iterate (according to some algorithm) many times to *approximate* the optimal value.
- Gradient descent is more laborious to code than the one-shot solution, but it generalizes to a 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” $\mathbf{w}$ to reduce the objective function (e.g., f_{MSE}).

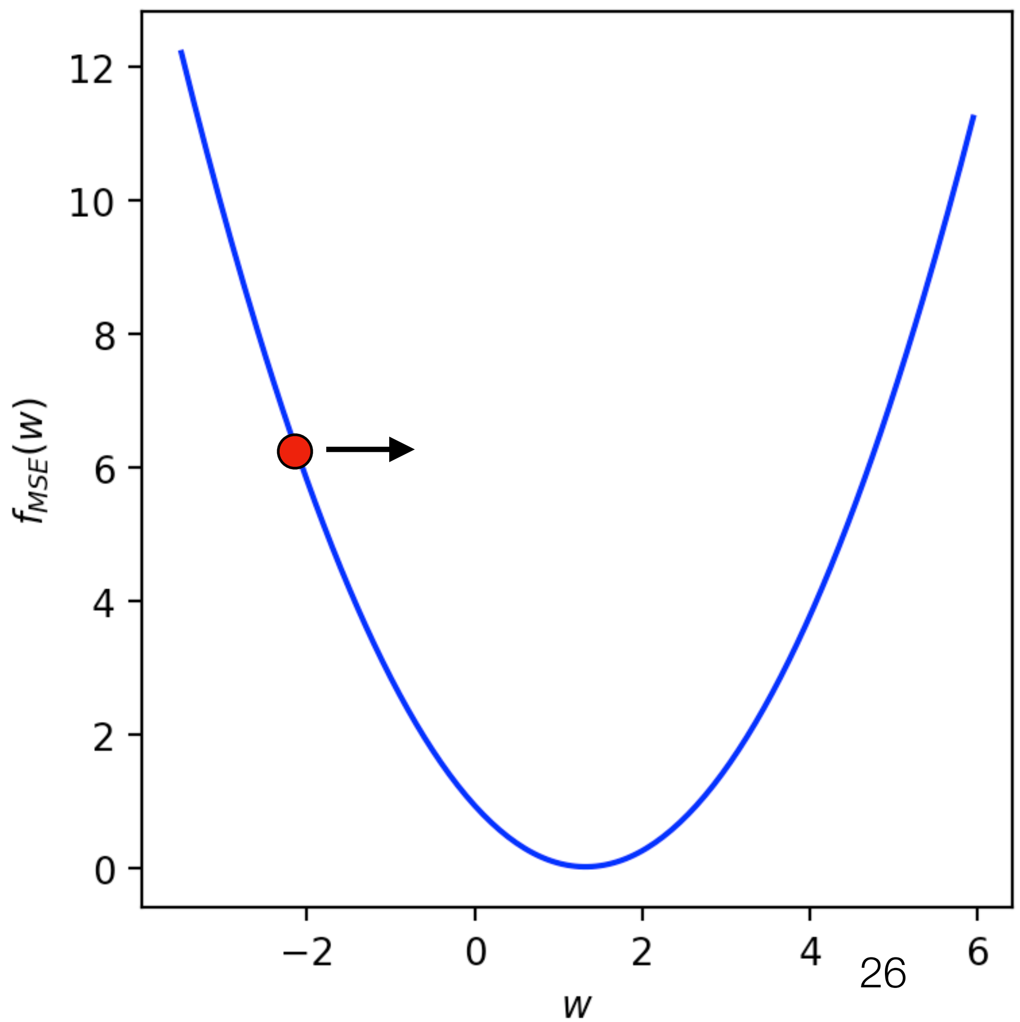
Gradient descent

- 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?



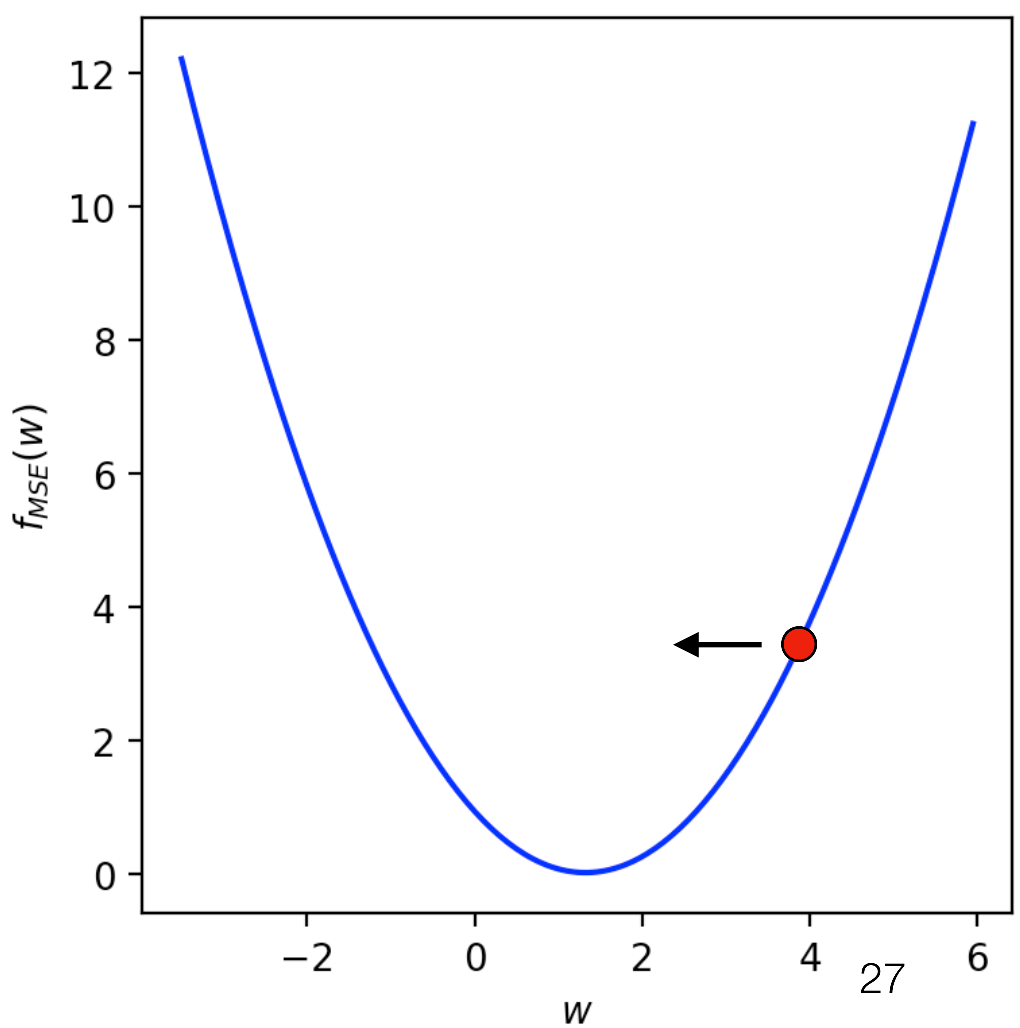
The slope at f_{MSE}(-2.1) is *negative*, i.e., we can *decrease* our cost by *increasing* w.

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

Jacob Whitehill, WPI

Gradient descent

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

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

Gradient descent

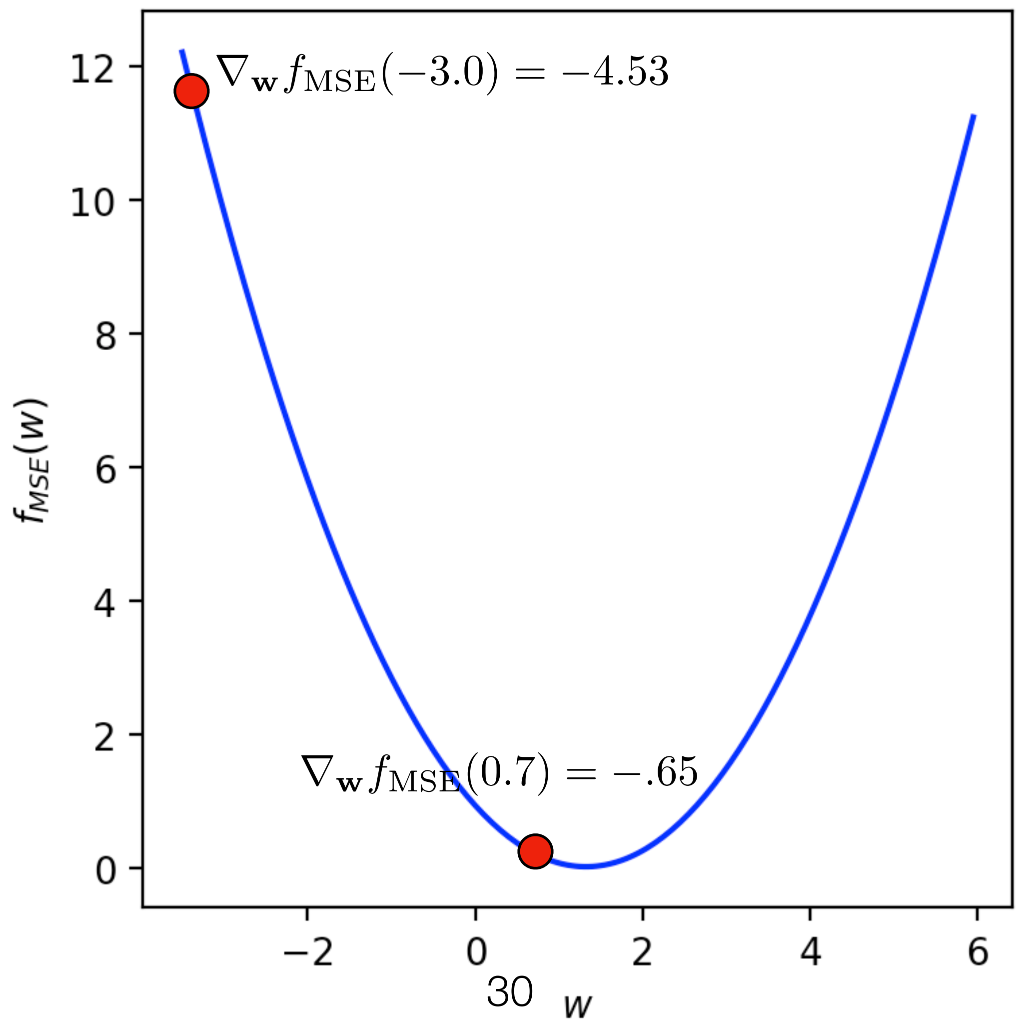
- How do we know the slope? Compute the **gradient** of f_{MSE} w.r.t. **w**:

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

- 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**.



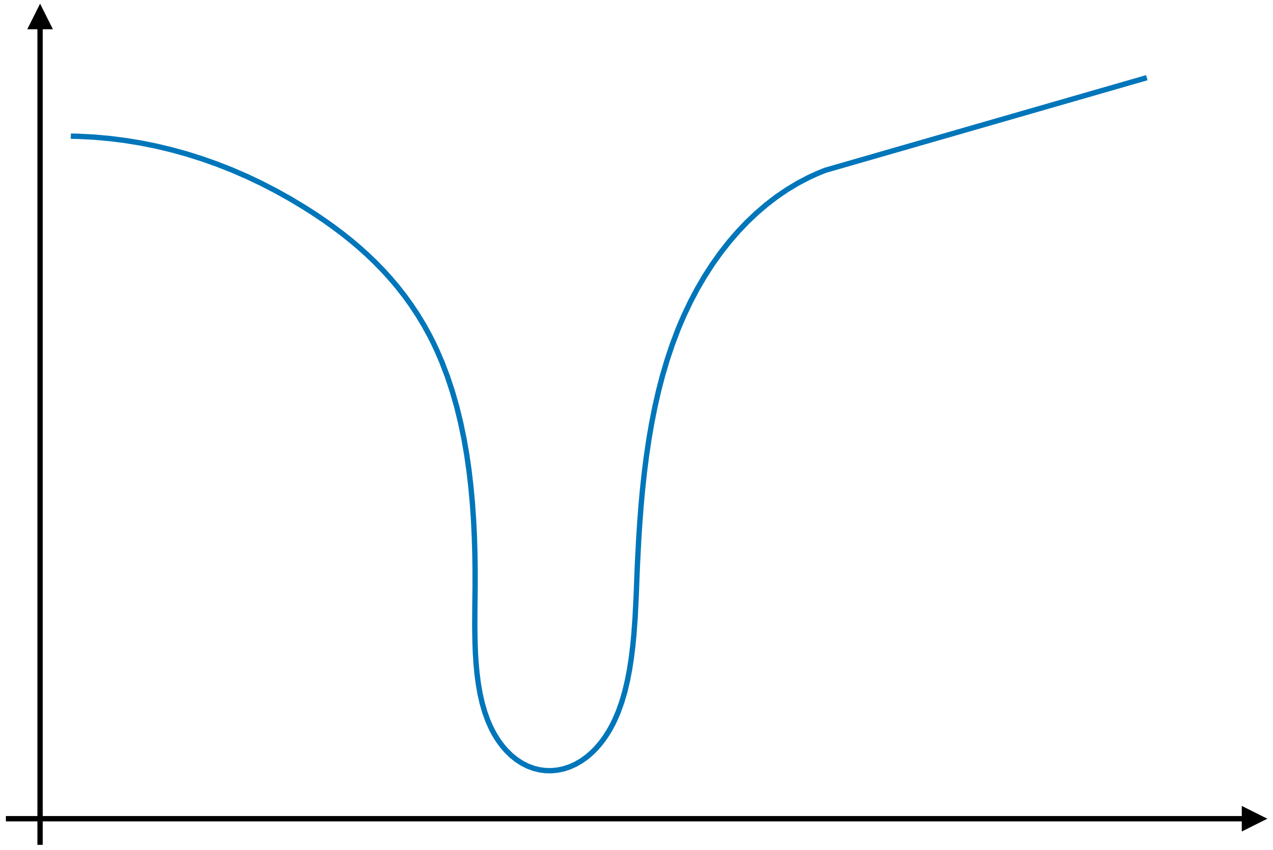
Jacob Whitehill, WPI

Exercise

- Draw on paper a function (with one local minimum) such that the magnitude of the gradient is NOT an indicator of how far to move w so as to reach the local minimum.

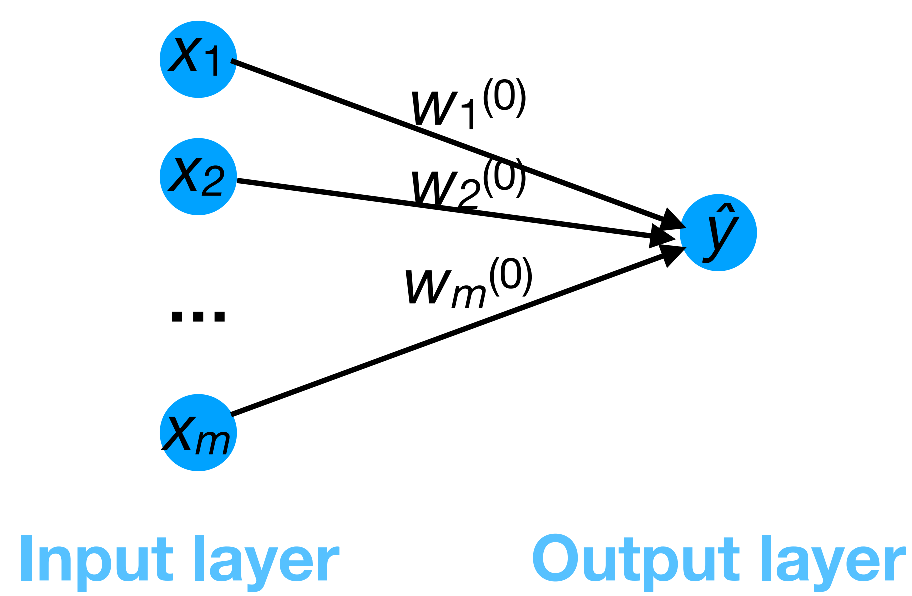
Exercise

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



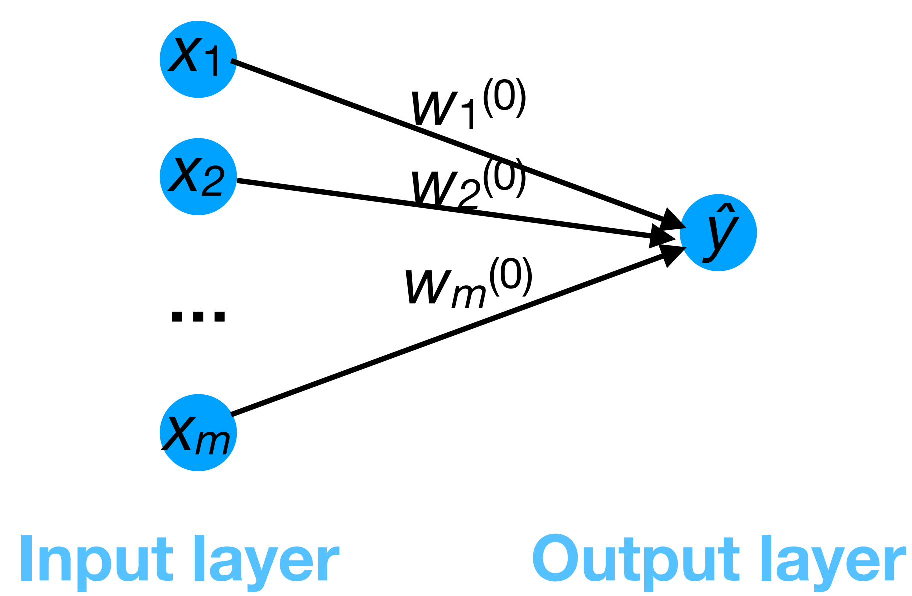
Gradient descent algorithm

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



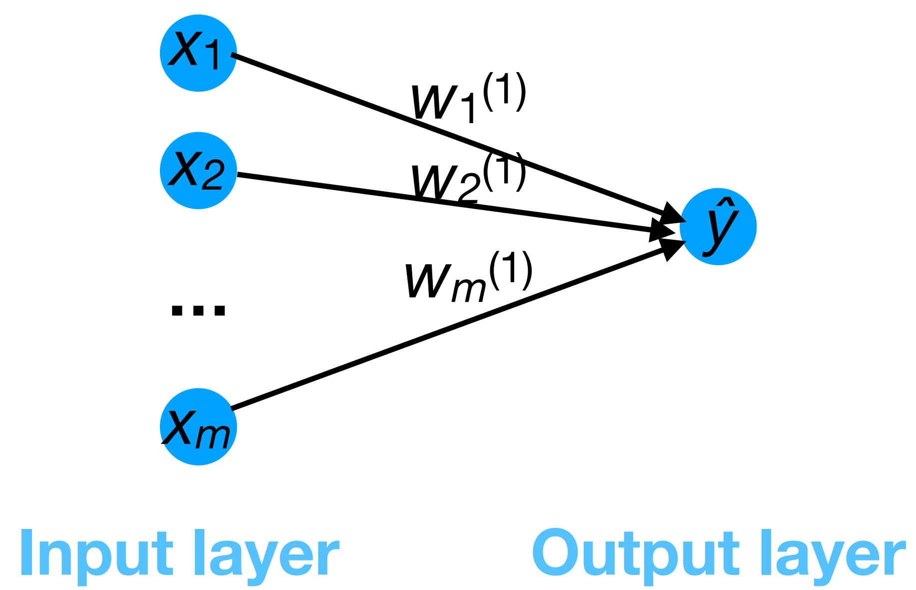
Gradient descent algorithm

- 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 **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 **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…

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

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

...

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

Gradient descent algorithm

- Set **w** to random values; call this initial choice **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…

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

...

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

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

Jacob Whitehill, WPI

Gradient descent demos

- Demo video.

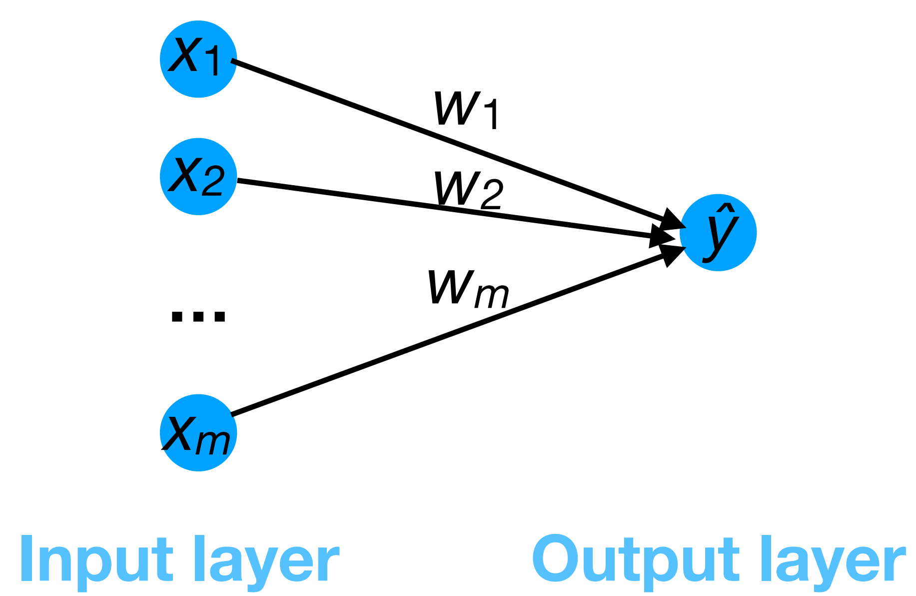
Exercise

Gradient descent

- For the 2-layer NN below, let m=2 and $\mathbf{w}^{(0)}$=[1 0]$^\top$.

- Compute the updated weight vector $\mathbf{w}^{(1)}$ after one iteration of gradient descent using (1/2) MSE loss, a single training example ($\mathbf{x}$,y)=([2, 3]$^\top$, 4), and learning rate $\boldsymbol{\epsilon}$=0.1.

- Recall: $\nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{w}) = \frac{1}{n}\mathbf{X}(\mathbf{X}^\top \mathbf{w} - \mathbf{y})$



Solution

$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{w}) &= \frac{1}{n}\mathbf{X}(\mathbf{X}^\top \mathbf{w} - \mathbf{y}) \\
\mathbf{w}^{(1)} &\leftarrow \mathbf{w}^{(0)} - \epsilon \nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{w}) \\
&= \left[\begin{array}{c} 1 \\ 0 \end{array}\right] - 0.1 \left[\begin{array}{c} 2 \\ 3 \end{array}\right] \left(\left[\begin{array}{c} 2 \\ 3 \end{array}\right]^\top \left[\begin{array}{c} 1 \\ 0 \end{array}\right] - \left[\begin{array}{c} 4 \end{array}\right] \right) \\
&= \left[\begin{array}{c} 1 + 0.1 * 2 * 2 \\ 0 + 0.1 * 3 * 2 \end{array}\right] \\
&= \left[\begin{array}{c} 1.4 \\ 0.6 \end{array}\right]
\end{aligned}
$$

Hyperparameter tuning

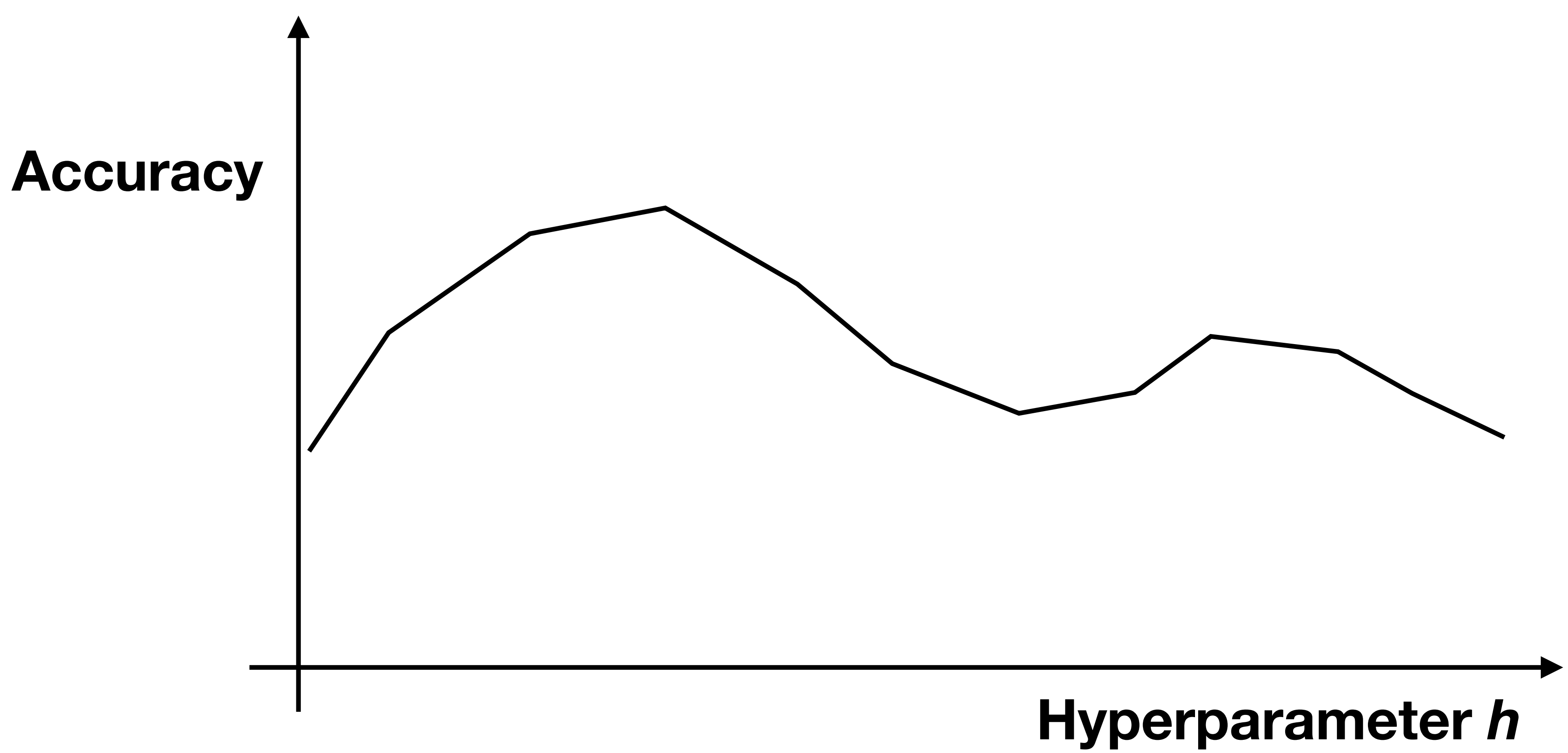
Hyperparameter tuning

- The values we **optimize** when training a machine learning model — e.g., **w** and *b* for linear regression — are the **parameters** of the model.

- There are also values related to the training process itself — e.g., learning rate ε, batch size $\tilde{n}$, regularization strength α — which are the **hyperparameters** of training.

Hyperparameter tuning

- Both the parameters and hyperparameters can have a huge impact on model performance on test data.
- Ideally, we would hope that the accuracy of the system varies smoothly with each hyper parameter value, e.g.:



Hyperparameter tuning

- However, in the real world, the hyperparameter landscape can be quite erratic, e.g.:

Hyperparameter tuning

- If you choose hyperparameters on the test set, you are likely deceiving yourself about how good your model is.
- **This is a subtle but very dangerous form of ML cheating.**

Hyperparameter tuning

- Instead, you should use a separate dataset that is not part of the test set to choose hyperparameters.
- Two commonly used (and rigorous) approaches:
    - Training/validation/testing sets
    - Double cross-validation

Training/validation/testing sets

- In an application domain with a large dataset (e.g., 100K examples), it is common to partition it into three subsets:
  - Training (typically 70-80%): optimization of parameters
  - Validation (typically 5-10%): tuning of hyperparameters
  - Testing (typically 5-10%): evaluation of the final model
- For comparison with other researchers' methods, this partition should be fixed.

Training/validation/testing sets

- Hyperparameter tuning works as follows:

1. For each hyperparameter configuration *h*:
   - Train the parameters on the **training** set using *h*.
   - Evaluate the model on the **validation** set.
   - If performance is better than what we got with the best *h* so far (h^*), then save *h* as h^*.
2. Train a model with h^*, and evaluate its accuracy *A* on the **testing** set. (You can train either on training data, or on training+validation data).

Exercise (from d2l.ai)

- Your manager gives you a difficult dataset on which your current algorithm doesn't perform so well. How would you justify to them that you need more data? Hint: you cannot increase the data but you can decrease it.