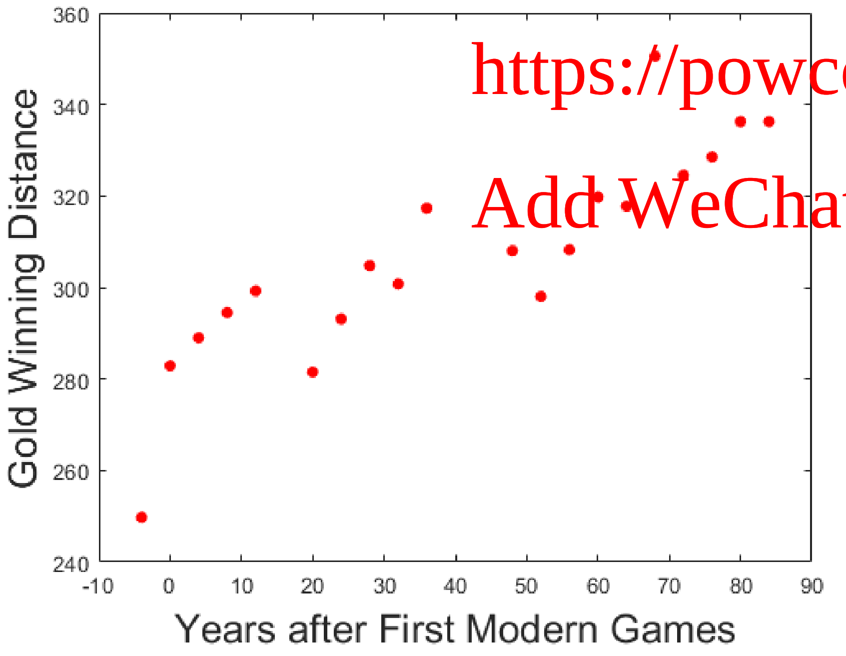
Linear Regression

Ata Kaban

Example of a regression problem

- Let’s look at some fun data. Can we predict the long-jump Gold winning distance 125 years after the first modern Olympic games?

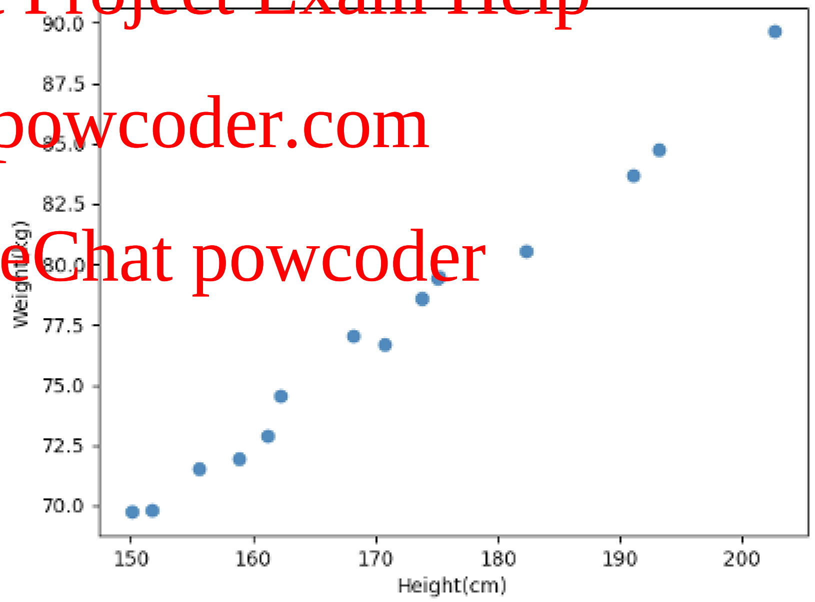


Data from: Rogers & Girolami. A First Course in Machine Learning. 2nd edition. Chapman & Hall/CRC, 2017

Example of a regression problem

- Let's look at more fun data. Can we predict people's weight from their height?

Height(cm)	Weight(kg)
150.00686	69.73847
151.64326	69.83261
155.54032	71.55730
158.80535	71.92875
161.17561	72.92118
⋮	⋮
175.15167	79.48533
182.32900	80.52182
191.11317	83.67998
193.21947	84.72086
202.68705	89.64049



https://becominghuman.ai/univariate-linear-regression-clearly-explained-with-example-4164e83ca2ee

Regression

- Regression means learning a function that captures the "trend" between input and output

- We then use this function to predict target values for new inputs

Univariate linear regression

- Visually, there appears to be a trend
- A reasonable ==model== seems to be the **class of linear functions (lines)**
- We have one input attribute (year) - hence the name **univariate**

$$y = f(x; \underbrace{w_0, w_1}_{\text{free parameters}}) = w_1 x + w_0$$

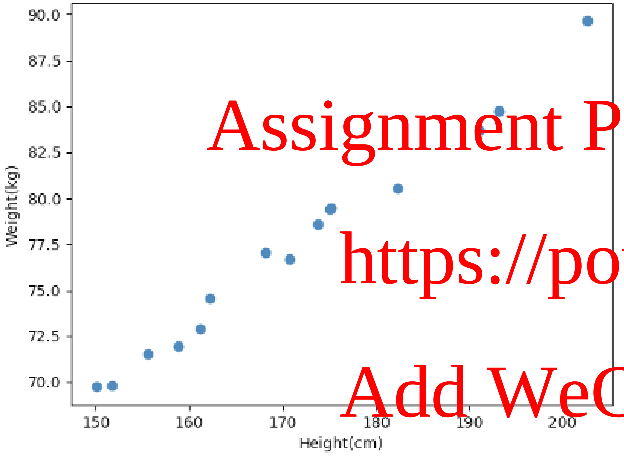
dependent variable

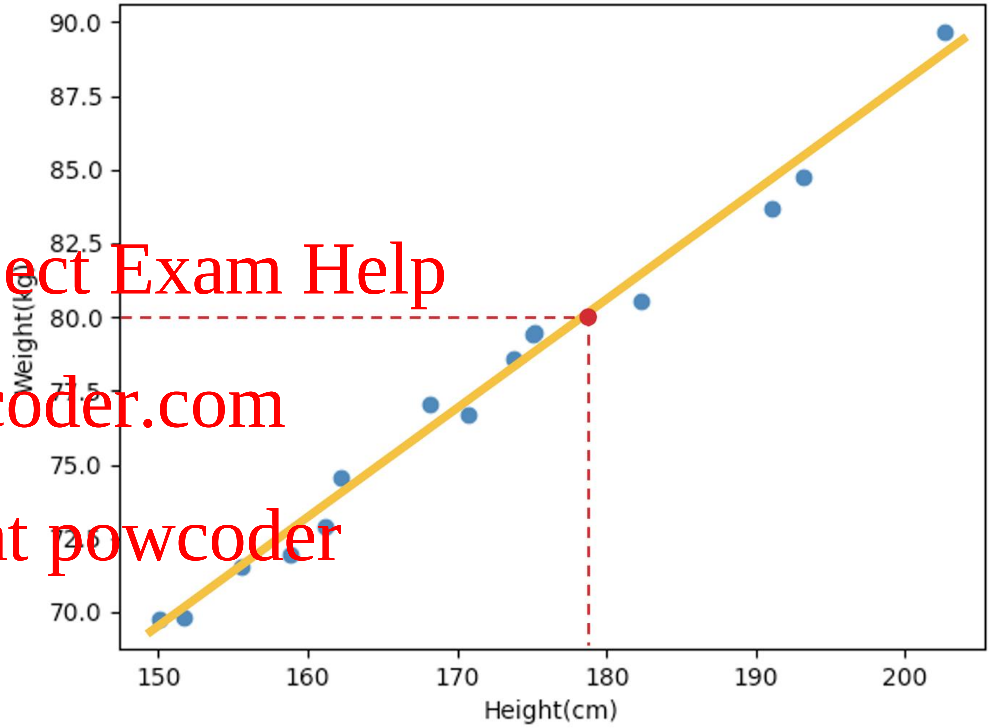
Independent variable

- Any line is described by this equation by specifying values for w_1, w_0.

Check your understanding

Height(cm)	Weight(kg)
150.00686	69.73347
151.64326	69.83261
155.54032	71.55730
158.80535	71.92875
161.17561	72.92118
⋮	
175.15167	79.48533
182.32900	80.52182
191.11317	83.67998
193.21947	84.72086
202.68705	89.64049





Suppose that from historical data someone already calculated the parameters of our linear model are $w_0 = 1.68,\ w_1 = 0.44$. A new person (James) has height x=178cm. Using our model, we can predict James' weight is 0.44 * 178 + 1.68 = 80kg.

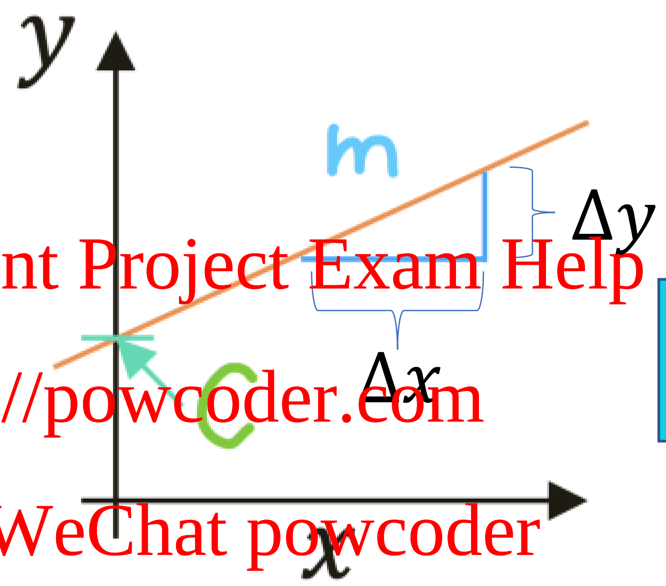
https://becominghuman.ai/univariate-linear-regression-clearly-explained-with-example-4164e83ca2ee

Play around with linear functions

- Go to https://www.desmos.com/calculator
- Type: $y = w_1 x + w_0$
- Plug in some values for the free parameters, or use the slider to see their effect
- $\textit{What is the role of the free parameters?}$
  - w_1 is the slope of the line
  - w_0 is the intercept with the y-axis
- Fixing concrete numbers for these parameters gives you specific lines

Equation of a straight line with slope *m* and intercept *c*.

$$f(x) = y = mx + c$$

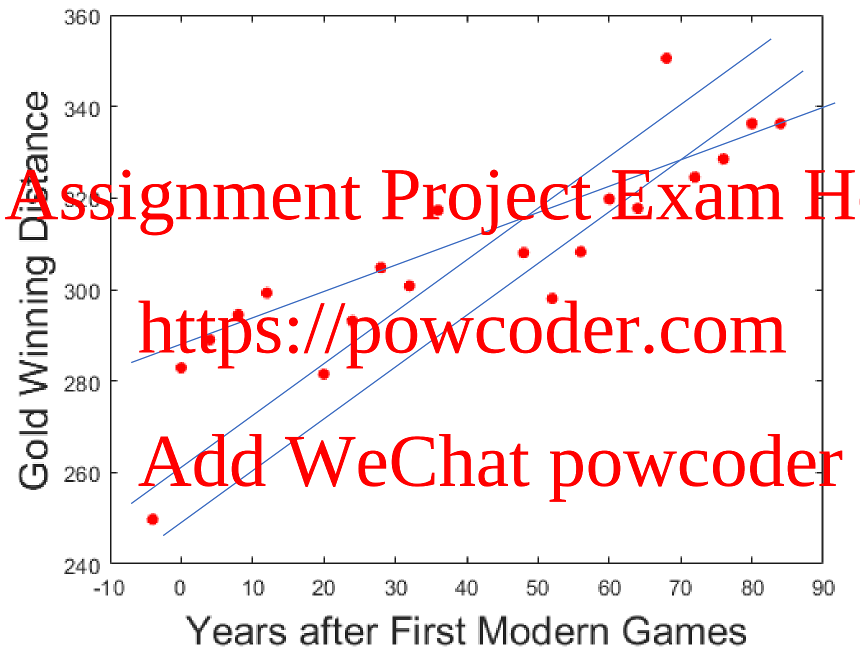


$$\boxed{m = \frac{\text{change in } y}{\text{change in } x} = \frac{\Delta y}{\Delta x},}$$

This is why: $f(x+\Delta x) = m(x+\Delta x) + c = mx + m\cdot\Delta x + c = f(x) + m\cdot\Delta x$

$$\Rightarrow m = \frac{f(x+\Delta x) - f(x)}{\Delta x} = \frac{\Delta y}{\Delta x}$$

Our goal: Find the “best” line



- Which is the ”best” line? That captures the trend in the data.
- Determine the “best” values for w_1, w_0.

Loss functions (or cost functions)

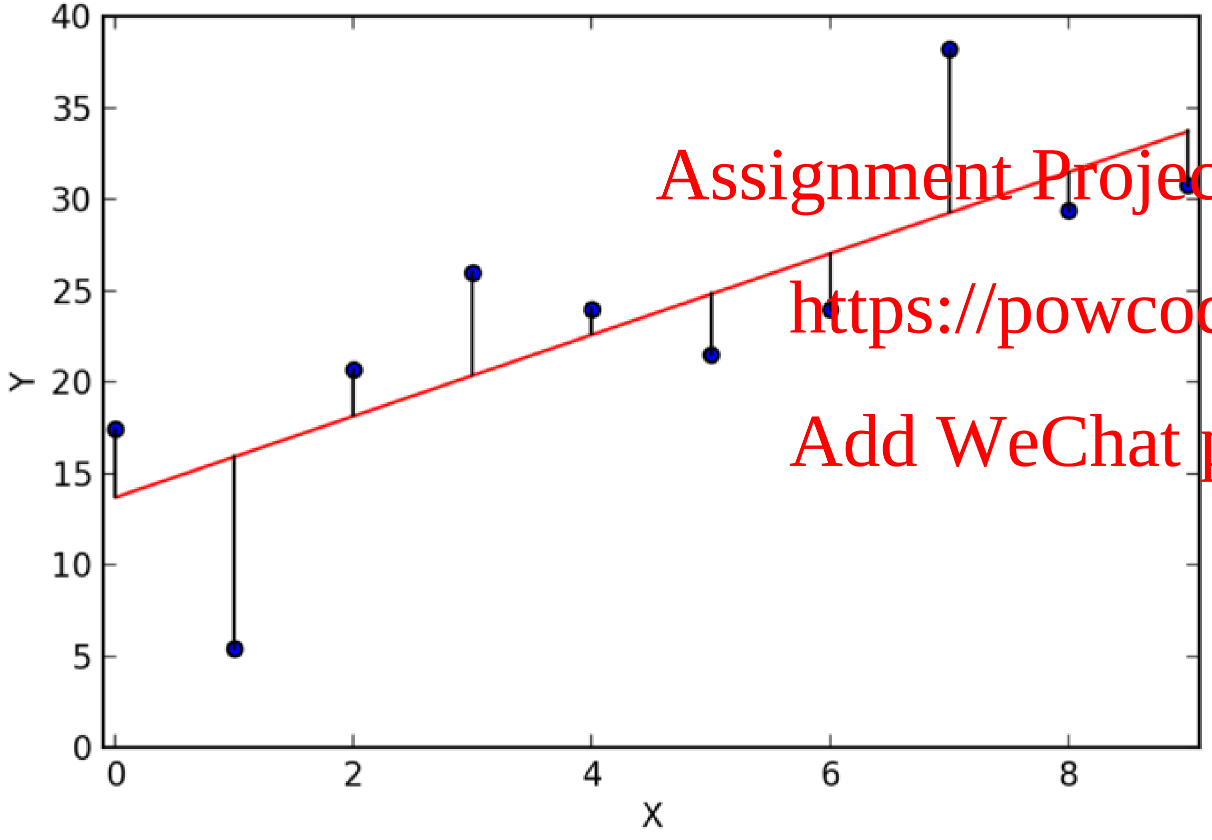
- We need a criterion that, given the data, for any given line will tell us how bad is that line.

- Such criterion is called a loss function. It is a function of the free parameters!

Terminology

- Loss function = cost function = loss = cost = error function

We average the losses on all training examples



For each training example (point) $n = 1, \ldots, N$,

- The loss on the n-th point is the mismatch between the output of the model for this point $f\left(x^{(n)}; w_0, w_1\right)$ and the observed target $y^{(n)}$.

Average these losses.

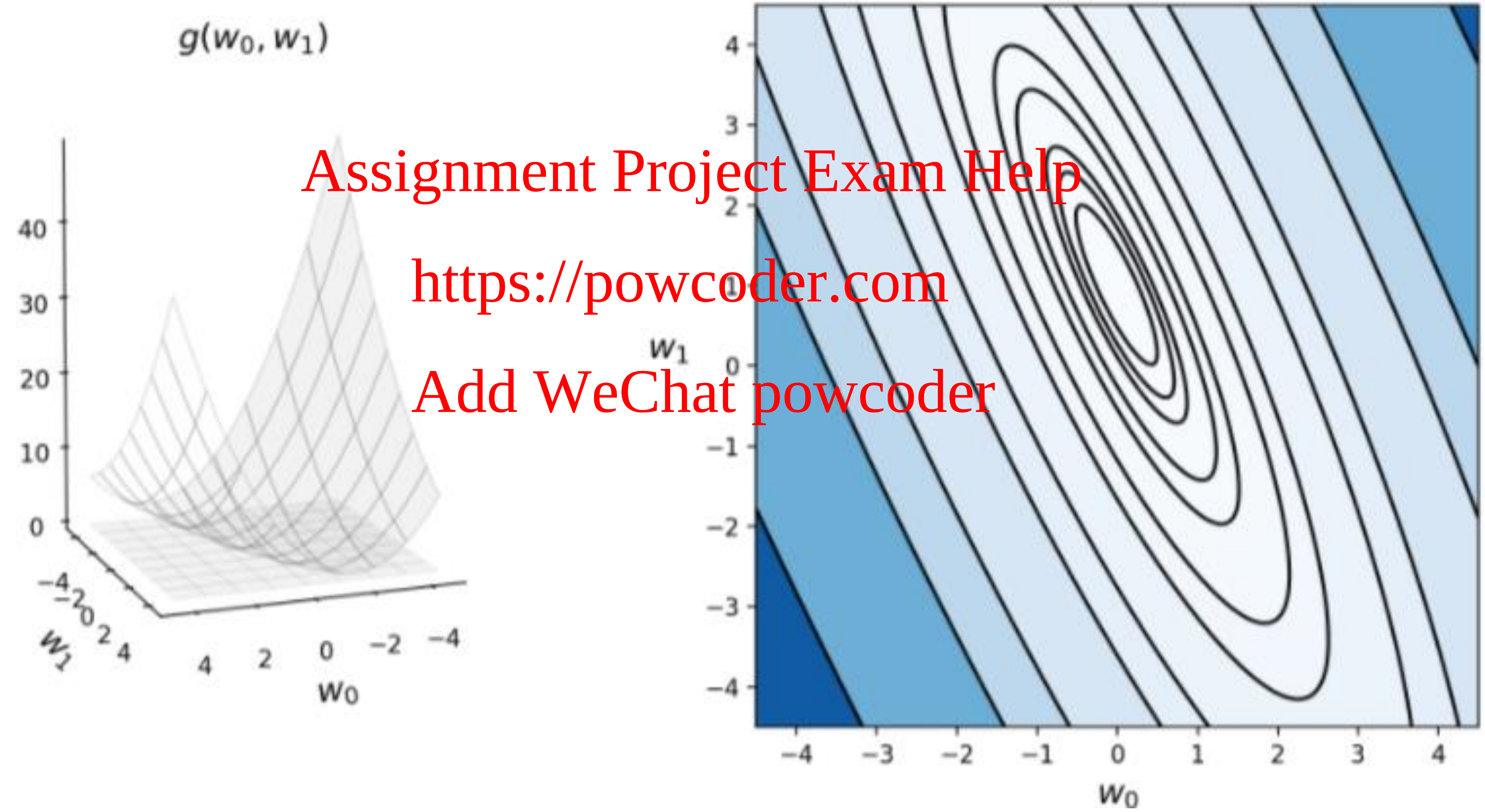
Square loss (L2 loss)

- The loss expresses an error, so it must be always non-negative
- Square loss is a sensible choice to measure mismatch for regression
- *Mean Square Error* (MSE)

$$g(w_0, w_1) = \frac{1}{N}\sum_{n=1}^{N} \underbrace{\left(f\left(x^{(n)}; w_0, w_1\right) - y^{(n)}\right)^2}_{\text{loss for the n-th training example}}$$

and recall that, for any x, we have $f(x; w_0, w_1) = w_1 x + w_0$

Cost function depends on the free parameters



Check your understanding

- Suppose a linear function with parameters $w_0 = 0.5$, $w_1 = 0.5$
- Compute the loss function value for this line at the training example: $(1,3)$.

- $f(x^{(1)};0.5,0.5) = 0.5 * 1 + 0.5 = 1$ (output of the model)
- $y^{(1)} = 3$ (actual target)
- Square loss for this point: $(1-3)^2 = 4$.
- Cost = 4.

Univariate linear regression – what we want to do

- Given training data

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})$$

- Fit the model

$$y = f(x; w_0, w_1) = w_1 x + w_0$$

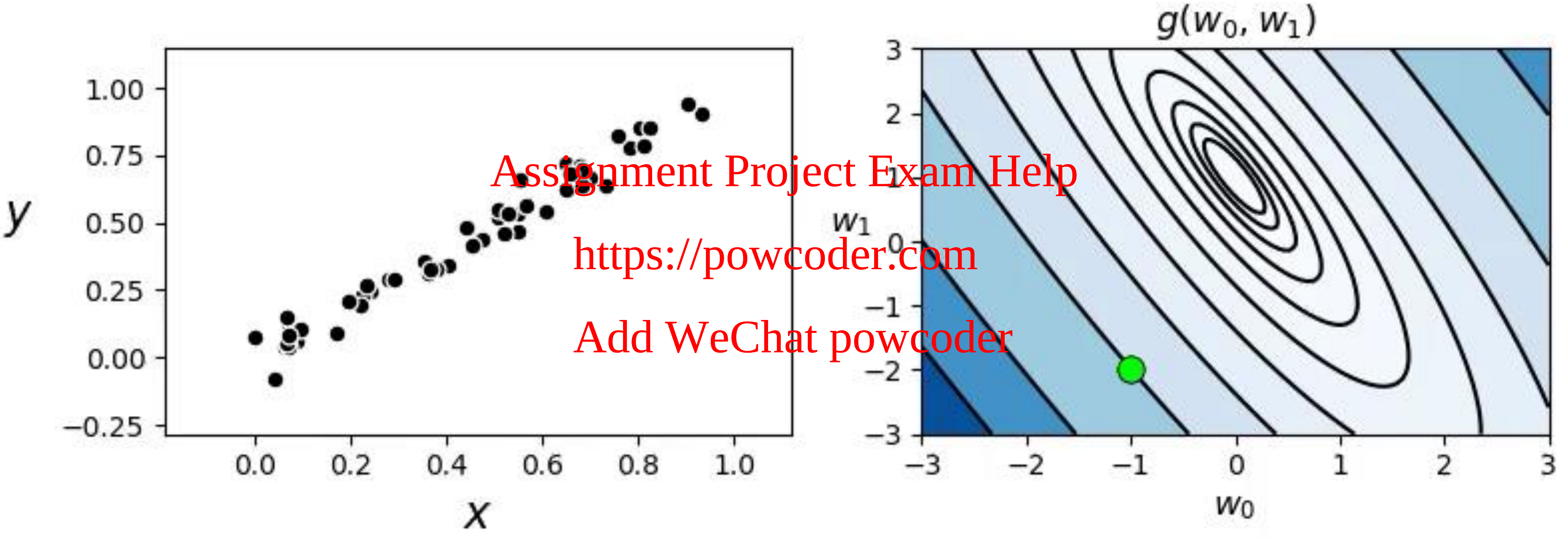
- By minimising the cost function

$$g(w_0, w_1) = \frac{1}{N} \sum_{n=1}^{N} \left(\left(w_1 x^{(n)} + w_0 \right) - y^{(n)} \right)^2$$

Univariate linear regression – what we want to do

- Every combination of w_0 and w_1 has an associated cost.

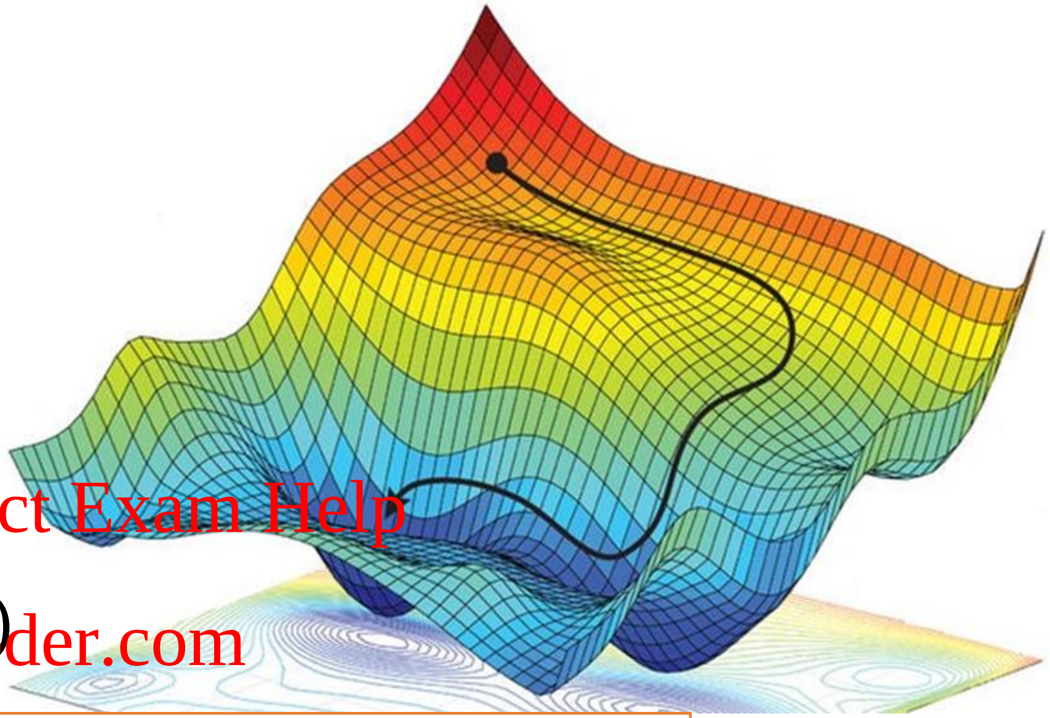
- To find the 'best fit' we need to find values for w_0 and w_1 such that the cost is minimum.



https://github.com/jermwatt/machine_learning_refined/tree/gh-pages/notes/5_Linear_regression/videos/animation_1.mp4

Gradient Descent

Gradient Descent



- A general strategy to minimise cost functions.

Goal: Minimise cost function $g(w_0, w_1)$

Start at say $w_0 := 0, w_1 := 0$

Repeat until no change occurs
    Update w_0, w_1 by taking
    a <u>small step</u> in the <u>direction of the steepest descent</u>
Return w_0, w_1

Gradient descent – the general algorithm

- Goal: Minimise cost function $g(\boldsymbol{w})$, where $\boldsymbol{w} = (w_0, w_1, , \ldots)$

Input: α>0

Initialise $\boldsymbol{w}$ // at 0 or some random value

Repeat until convergence

$$\boldsymbol{w} := \boldsymbol{w} - \alpha \, \nabla g(\boldsymbol{w})$$

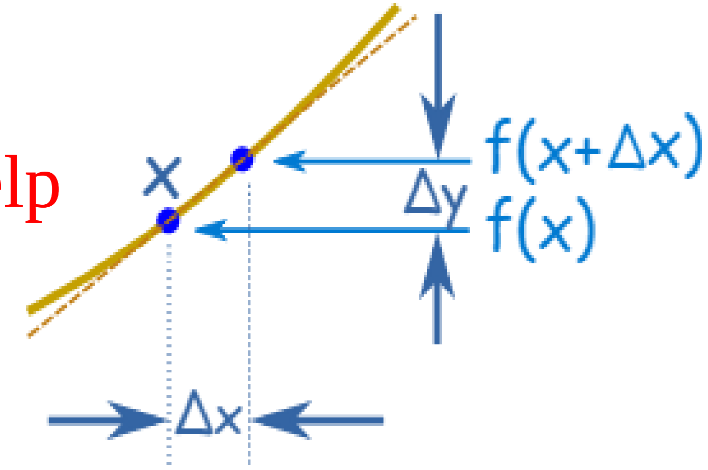
Return w

step size direction

α is called "learning rate"= "step size", for instance 0.01

How to find the best direction?

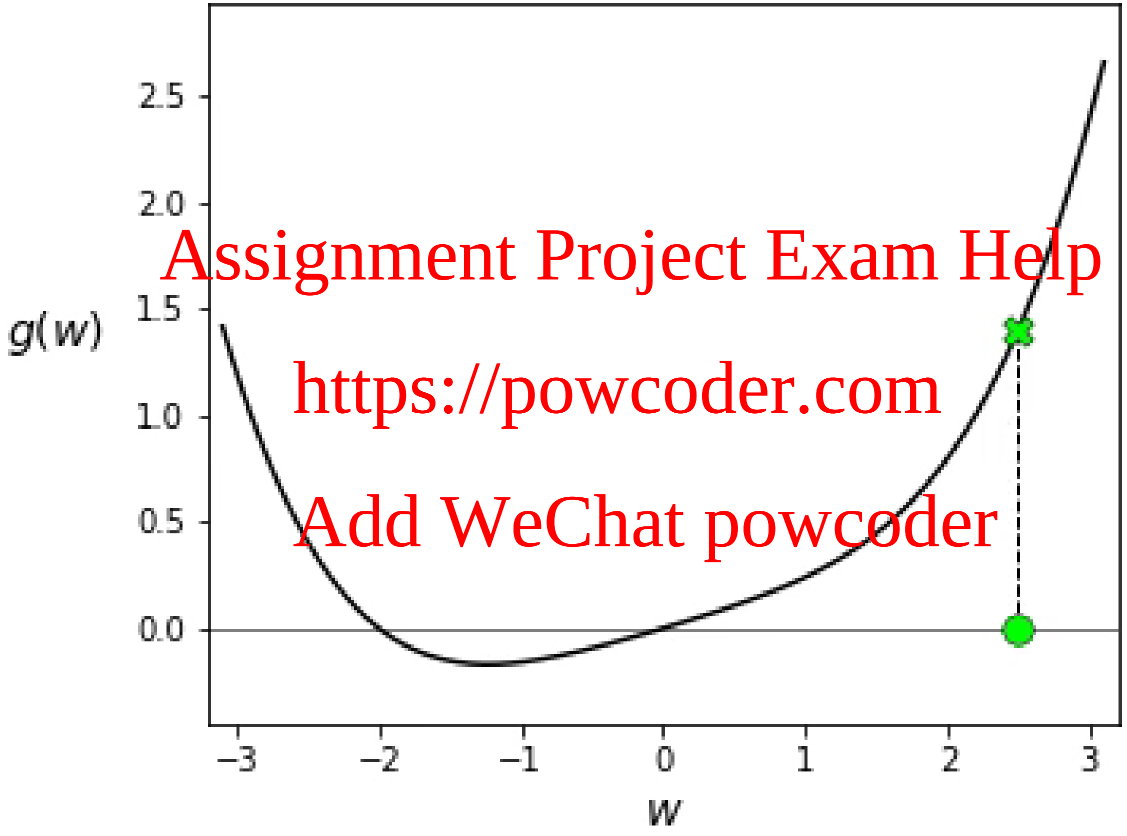
- First, recall from calculus that the derivative of a function is the change in function value as the argument of the function changes by a minimal amount.
- The derivative evaluated at a given location gives us the slope of the tangent line at that point.
- The negative of the slope points towards the minimum point. Check!



$$\frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$$\Delta x \rightarrow 0$$

Demo example for gradient descent algorithm



https://github.com/jermwatt/machine_learning_refined/tree/gh-pages/notes/3_First_order_methods/videos/animation_6.mp4

Gradient

- **Partial derivative** with respect to w_0 is $\frac{\delta g(w_0,w_1)}{\delta w_0}$. It means the derivative function of $g(w_0, w_1)$ when w_1 is treated as constant.
- **Partial derivative** with respect to w_1 is $\frac{\delta g(w_0,w_1)}{\delta w_1}$. It means the derivative function of $g(w_0, w_1)$ when w_0 is treated as constant.

- The vector of partial derivatives is called the **gradient**.

$$\nabla g(w) = \begin{pmatrix} \frac{\delta g(w_0,w_1)}{\delta w_0} \\ \frac{\delta g(w_0,w_1)}{\delta w_1} \end{pmatrix} \quad \text{where } w = \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$$

- The negative of the gradient evaluated at a location $(\widehat{w_0}, \widehat{w_1})$ gives us the direction of the **steepest descent** from that location.
- We take a small step in that direction.

Gradient Descent applied to solving Univariate Linear Regression

Computing the gradient for our L2 loss

- Recall the cost function g(w_0, w_1) = $\frac{1}{N}\sum_{n=1}^{N}\left(\left(w_1 x^{(n)} + w_0\right) - y^{(n)}\right)^2$
- *Using the chain rule, we have*:*

$$\frac{\delta g(w_0,w_1)}{\delta w_0} = \frac{2}{N}\sum_{n=1}^{N}\left(w_1 x^{(n)} + w_0\right) - y^{(n)}$$

$$\frac{\delta g(w_0,w_1)}{\delta w_1} = \frac{2}{N}\sum_{n=1}^{N}\left(\left(w_1 x^{(n)} + w_0\right) - y^{(n)}\right) x^{(n)}$$

---

*For a very detailed explanations of all steps watch: https://www.youtube.com/watch?v=sDv4f4s2SB8

Algorithm for univariate linear regression using GD

- Goal: Minimise $g(w_0, w_1) = \frac{1}{N}\sum_{n=1}^{N}\left(\left(w_1 x^{(n)} + w_0\right) - y^{(n)}\right)^2$

Input: α>0, training set {($x^{(n)}, y^{(n)}$):n=1,…,N}

Initialise $w_0 := 0, w_1 := 0$

Repeat

For n=1,…,N // more efficient to update after each data point

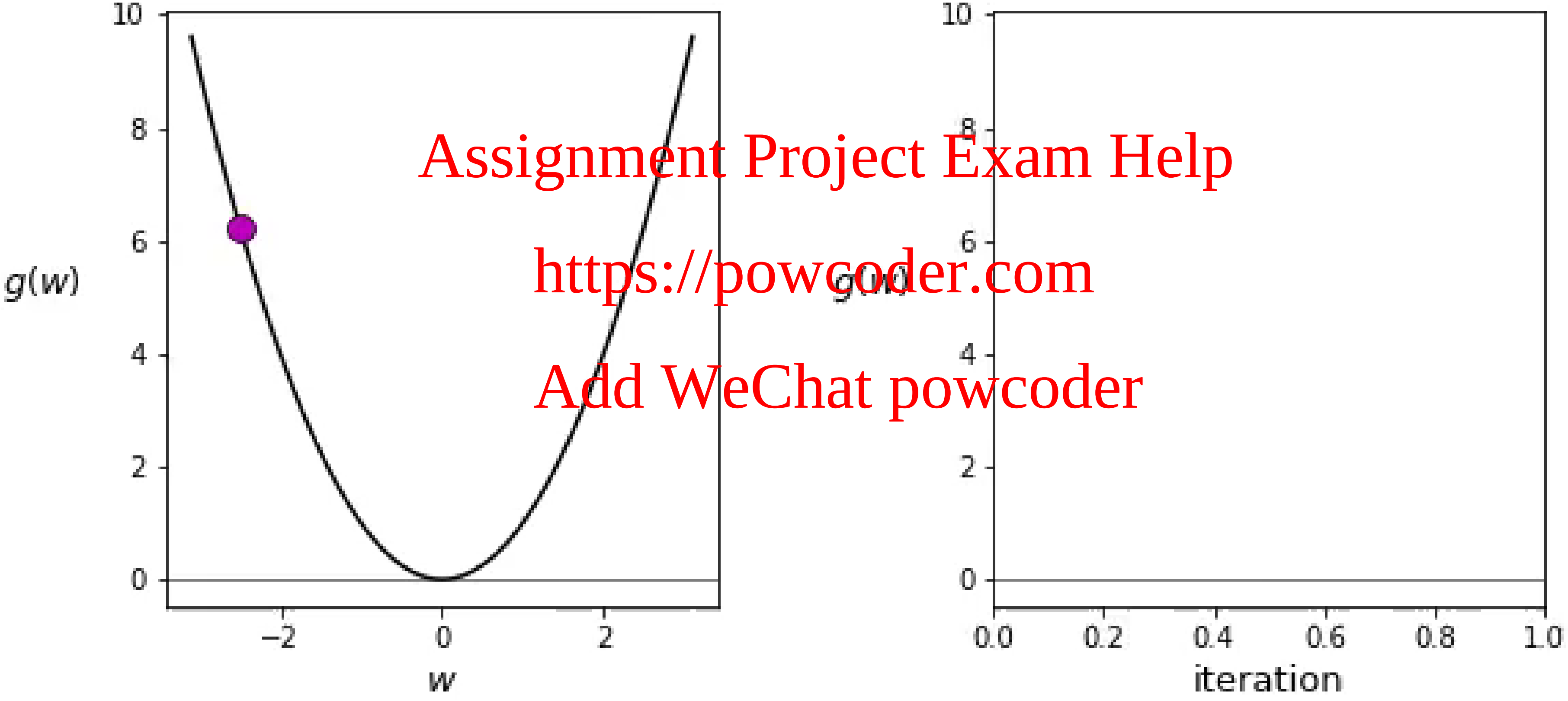
$$w_0 := w_0 - \alpha \cdot \left(\left(w_1 x^{(n)} + w_0\right) - y^{(n)}\right)$$

$$w_1 := w_1 - \alpha \cdot \left(\left(w_1 x^{(n)} + w_0\right) - y^{(n)}\right) x^{(n)}$$

Until change remains below a very small threshold

Return w_0, w_1

Effect of the learning rate

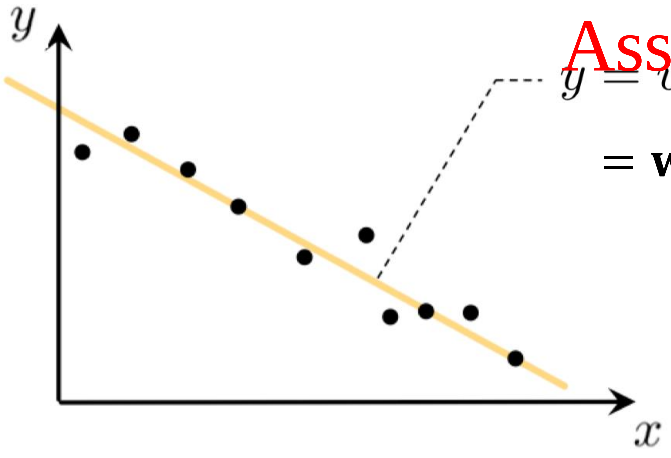


https://github.com/jermwatt/machine_learning_refined/tree/gh-pages/notes/3_First_order_methods/videos/animation_7.mp4

Extensions & variants of regression problems

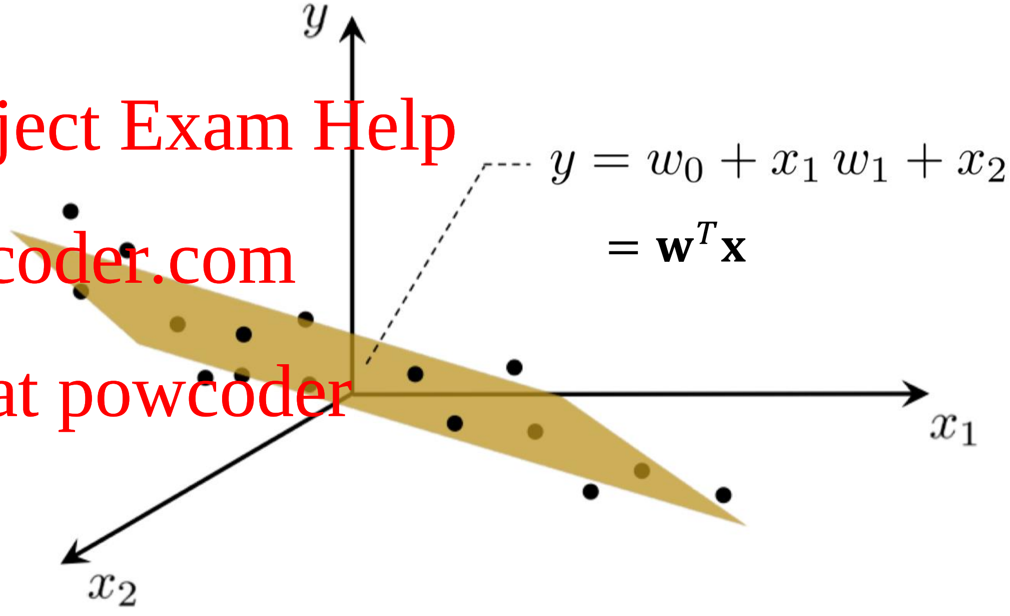
- We change the model
- The loss and cost function remains the same

Multivariate linear regression



$$y = w_0 + x\, w_1$$
$$= \mathbf{w}^T \mathbf{x}$$

with $\mathbf{w} := \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$ $\quad \mathbf{x} := \begin{pmatrix} 1 \\ x \end{pmatrix}$

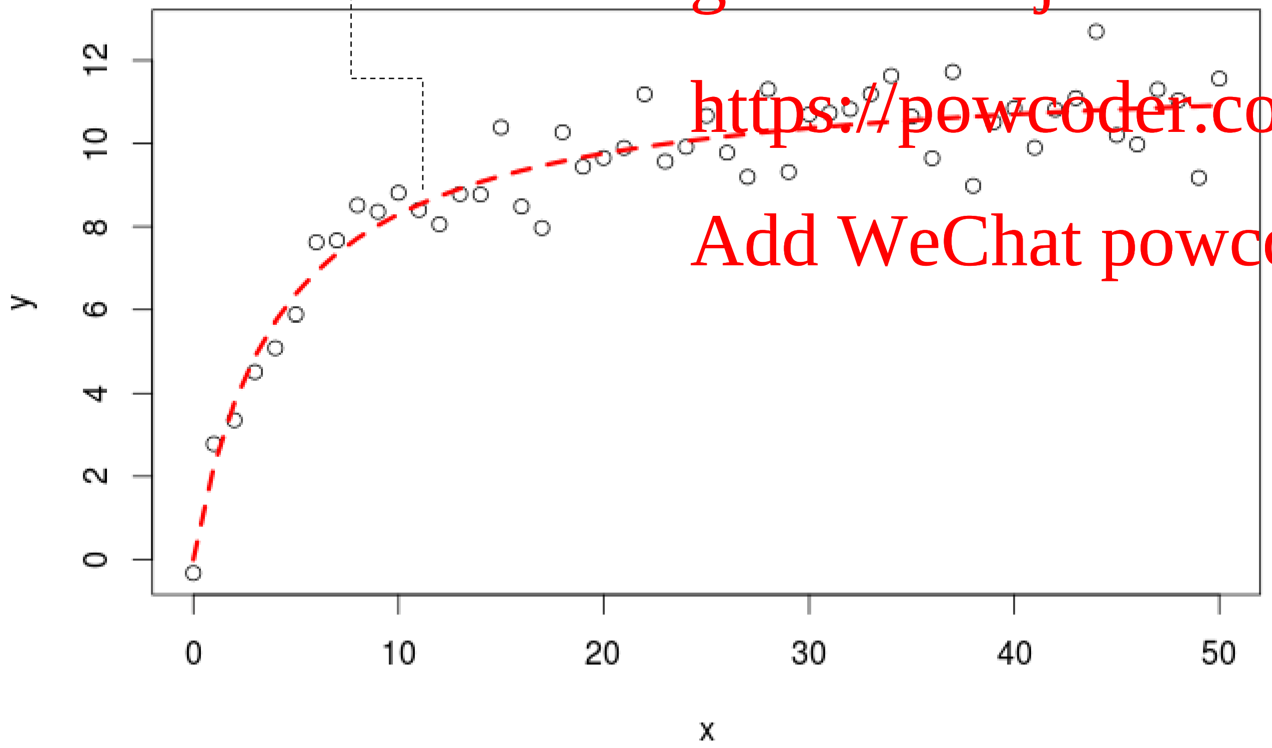


$$y = w_0 + x_1\, w_1 + x_2\, w_2$$
$$= \mathbf{w}^T \mathbf{x}$$

with $\mathbf{w} := \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$ $\quad \mathbf{x} := \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$

Univariate nonlinear regression

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_m x^m = \mathbf{w}^T \mathbf{x} \quad \text{with}$$



$$\mathbf{w} := \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \ldots \\ w_m \end{pmatrix} \qquad \mathbf{x} := \begin{pmatrix} 1 \\ x \\ x^2 \\ \ldots \\ x^m \end{pmatrix}$$

This is an m-th order polynomial regression model.

Figure from: https://www.r-bloggers.com/first-steps-with-non-linear-regression-in-r

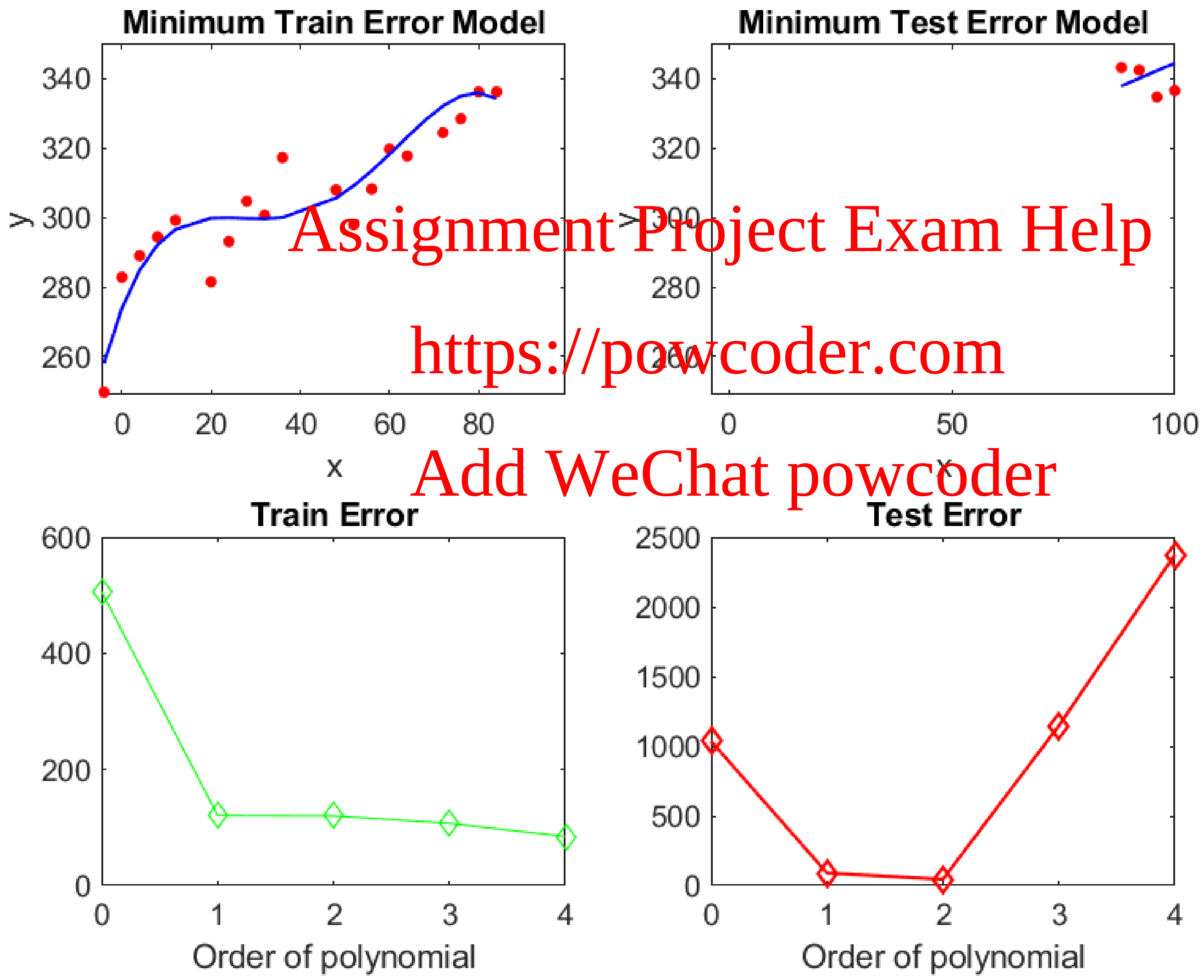
Advantages of vector notation

- Vector notation in concise
- With the vectors $\boldsymbol{w}$ and $\mathbf{x}$ populated appropriately (and differently in each case, as on the previous 2 slides), these models are still linear in the parameter vector.
- The cost function is the L2 as before
- So the gradient in both cases is:

$$\nabla g(\mathbf{w}) = 2(\mathbf{w}^T\mathbf{x}^{(n)} - y^{(n)})\mathbf{x}^{(n)}$$

- Ready to be plugged into the general gradient descent algorithm

Don’t get too carried away with nonlinearity



Reference Acknowledgement

Several figures and animations on these slides are taken from:

- Jeremy Watt et al. Machine Learning Refined. Cambridge University Press, 2020. https://github.com/jermwatt/machine_learning_refined