MACHINE LEARNING

MEI/1

University of Beira Interior,
Department of Informatics

Hugo Pedro Proença,
hugomcp@di.ubi.pt, 2019/2020

Machine Learning

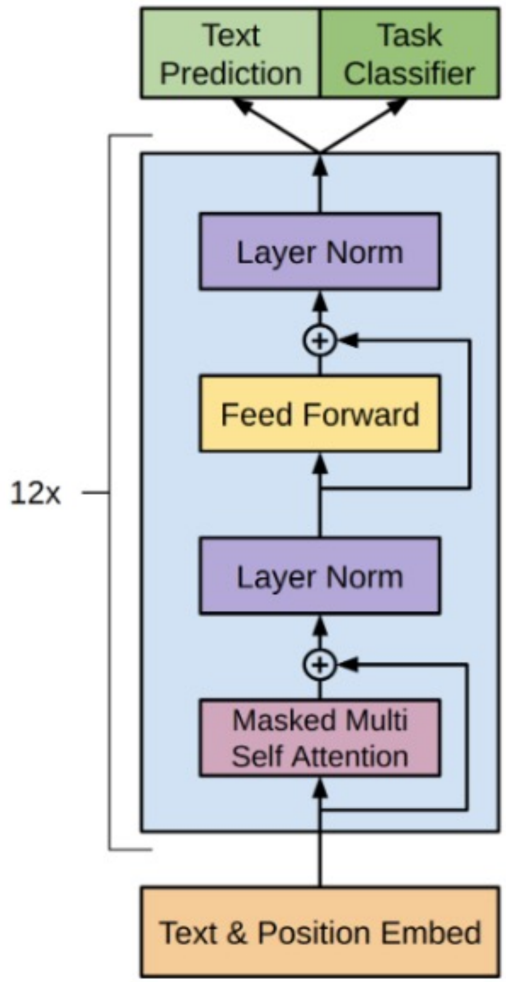
[02]

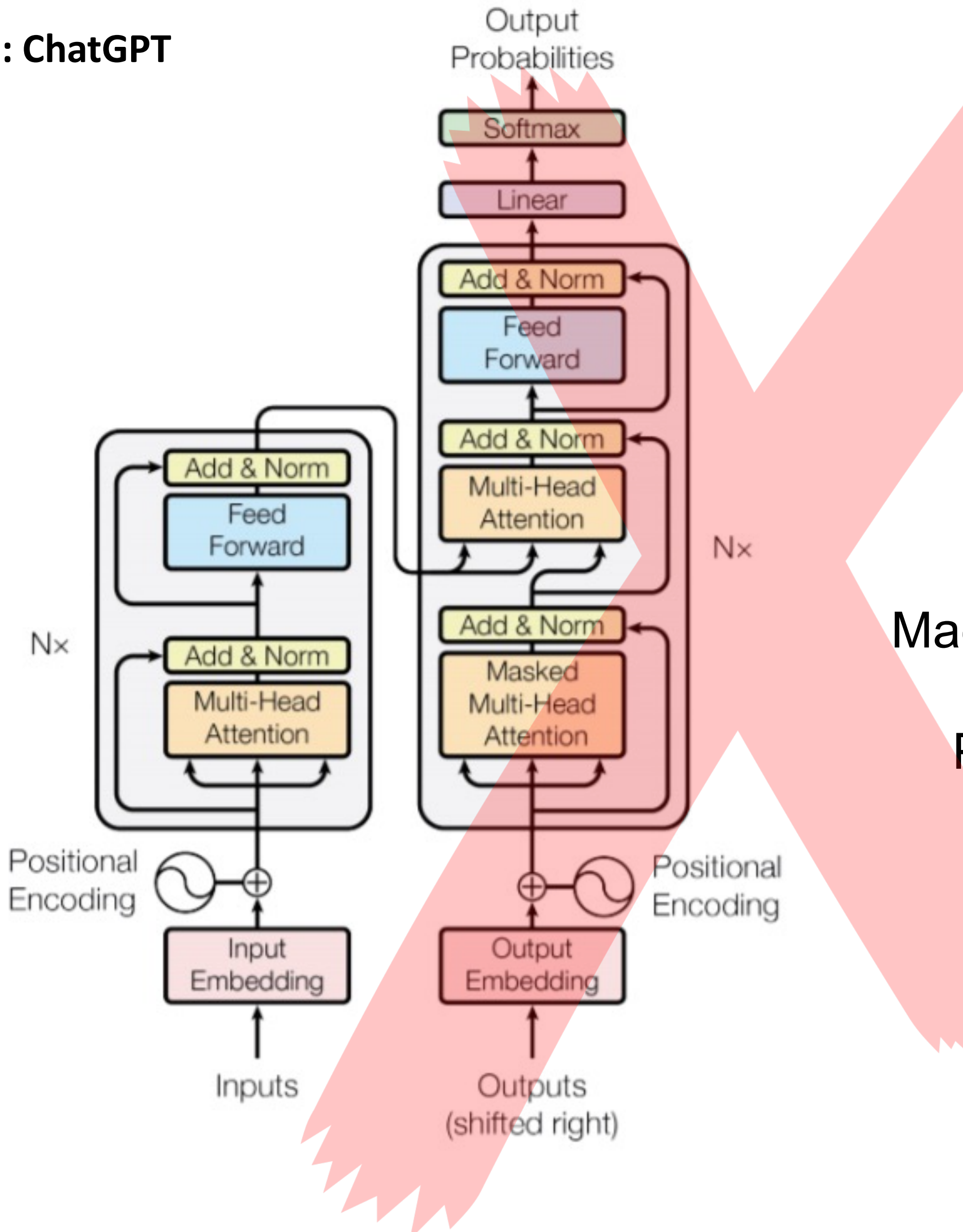
Syllabus

- Machine Learning Learning Paradigms
  - Linear Regression
  - Model Representation
  - Cost Function
  - Optimization
    - Closed-Form
    - Gradient Descent

Machine Learning Learning Paradigms

A Large Language Model (LLM): ChatGPT





Machine Learning on **User** Perspective!

Machine Learning Learning Paradigms

Fine Tunning a LLM (ChatGPT)

Step 1: Prepare Data (JSON file)

```
{"messages": [ {"role": "system",
    "content": "You are an assistant that occasionally misspells words" },
  {"role": "user",
    "content": "Tell me a story." },
  {"role": "assistant",
    "content": "One day a student went to school." }]}
```

Step 2: Upload Data

```
!pip install -U openai
import openai
openai.api_key = "YOUR_OPENAI_API_KEY"
openai.File.create( file=open('/path/to/your/data.jsonl'), purpose='fine-tune',)
```

Step 3: Create Job

```
openai.FineTuningJob.create(training_file='your_file_id',model='gpt-3.5-turbo',)
```

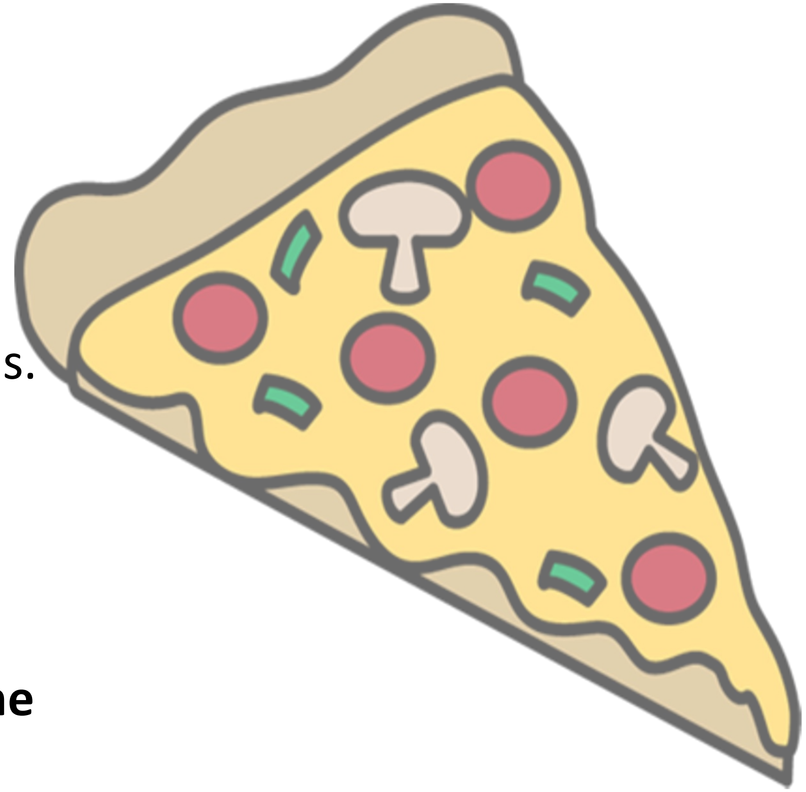
Step 4: Use Model

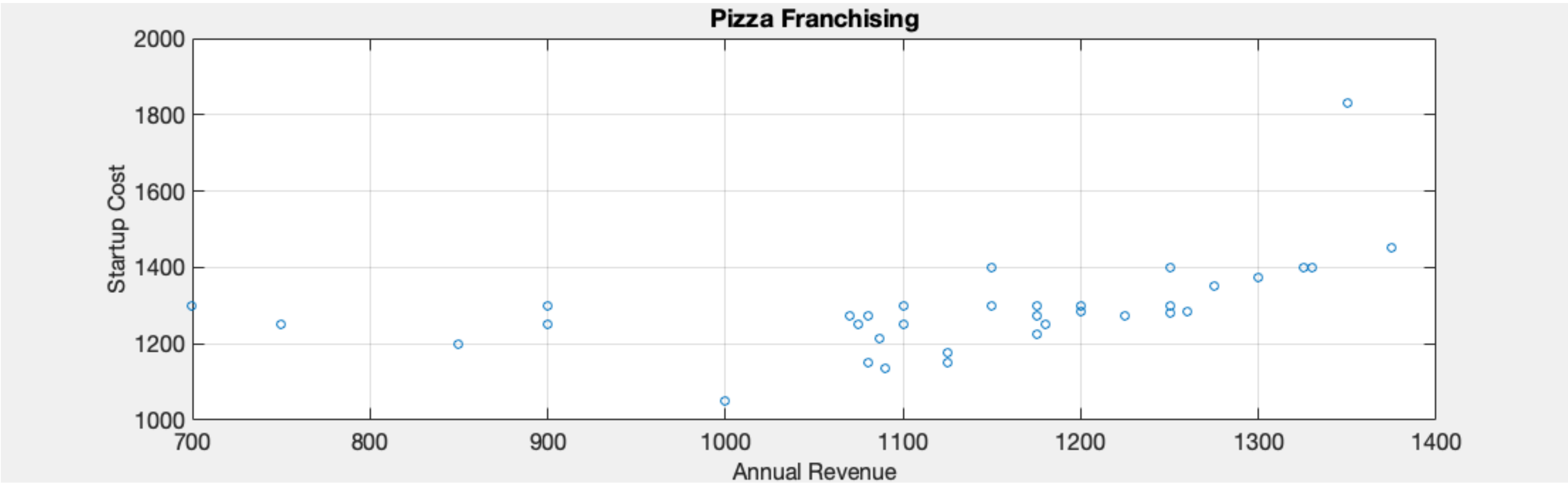
```
completion = openai.ChatCompletion.create( model='gpt-3.5-turbo',
  messages=[ {"role": "system", "content": "You are Zordon, leader of the Power Rangers."},
    {"role" "user", "content": "Zordon, the Red Ranger has been captured! What do we do?"}])
```

Machine Learning on **User** Perspective!

Pizza Franchising

- Pizza is a $45.1 billion industry in the United States.
- Suppose that one of the most well-known Pizza chain is interested in perceiving the relationship between the **average annual revenue** of its local stores and the corresponding **startup cost**.
- This data will be of maximum interest to **define the franchise fee for future openings**





Pizza Franchising

Independent Variable

Dependent Variable

- It appears that there is a **direct relation** between the annual income of one store, and the cost to start the store.
  - On average, larger stores sell more Pizza, but also they are more costly to set up:
  - Furniture, taxes, employees…

- In this problem we have 36 examples, typically designated as "**instances**"
  - N=36
- The independent variables are typically referred to as "**features**"
  - Are the input variables (x)
- The number of features determines the **dimensionality of the problem**
  - d=1
- The dependent variable is typically designated as the output, or "**target**"
  - The target distribution determines the type of supervised machine learning problem: classification or **regression** (in this case)

Annual Fee	Startup Cost
1000	1050
1125	1150
1087	1213
1070	1275
1100	1300
1150	1300
1250	1400
1150	1400
1100	1250
1350	1830
1275	1350
1375	1450
1175	1300
1200	1300
1175	1275
1300	1375
1260	1285
1330	1400
1325	1400
1200	1285
1225	1275
1090	1135
1075	1250
1080	1275
1080	1150
1180	1250
1225	1275
1175	1225
1250	1280
1250	1300
750	1250
1125	1175
700	1300
900	1250
900	1300
850	1200

Machine Learning I: Model Representation

- Suppose that the experts/administration/managers of the Pizza chain think that it might exist a roughly linear relationship between the annual revenue of one store and its startup cost:
  - This kind of “expertise” is always valuable to machine learning, as it simplifies the range of models that we can attempt to create
- Also, one of the Machine Learning’s foundation is the **Occam’s razor**:
- Known as the law of parsimony
  - Is a problem-solving principle that essentially states that "**simpler solutions are more likely to be correct than complex ones**".
  - When comparing competing hypotheses to solve a problem, one should select the solution with the fewest assumptions, i.e., **the simplest**
- The idea is attributed to English Franciscan friar William of Ockham (1287–1347), a scholastic philosopher and theologian.

Machine Learning I: Model Representation

- Linear Model
  - According to Occam's razor (and the administration also!), in the Pizza Franchising, we should start by consider a purely linear model to "**describe the pattern**" (i.e., describe the relationship) between the independent(s) and the dependent variables
- Formally, our model (**hypothesis**) is that:

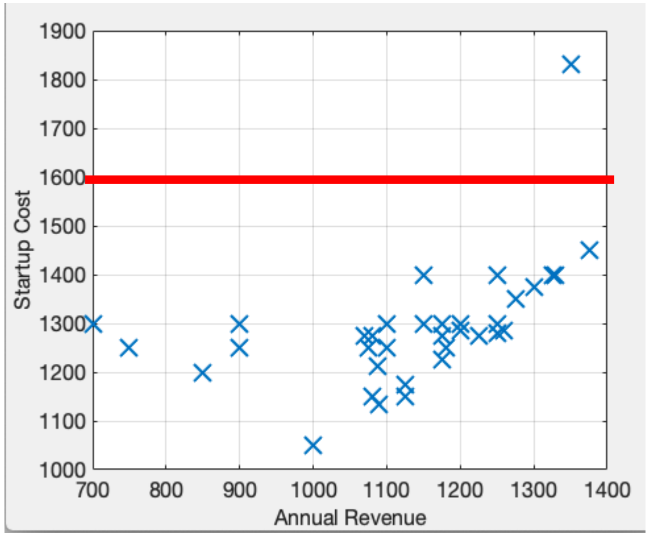
$$h_\theta(x) = \theta_1 . x + \theta_2$$

- The task of Machine Learning is to **find us the best possible model**, i.e., the one **that optimally expresses the relationship** between the independents and dependent variables
- This essentially involves to find the optimal (θ_1, θ_2) values
- After all, we end up with an **optimization problem in the R^2 space**

Machine Learning II: Cost Function

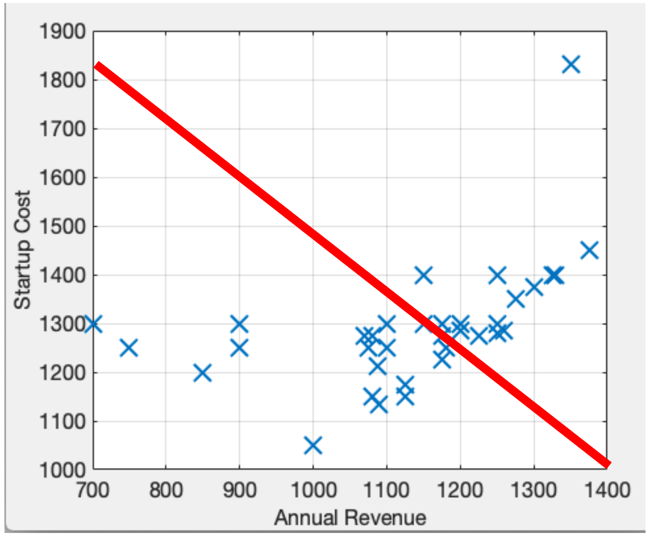
- Clearly, there will be models that are better than others:

$(\theta_1 = 0, \theta_2 = 1600)$

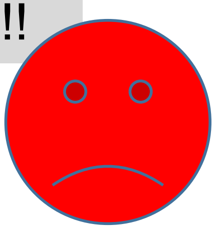


Bad.

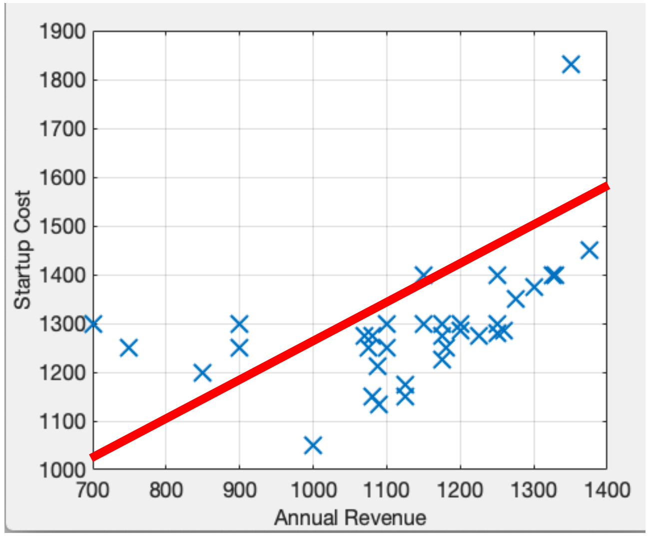
$(\theta_1 = -1.15, \theta_2 = 1005)$



Terrible!!



$(\theta_1 = 0.82, \theta_2 = 446)$



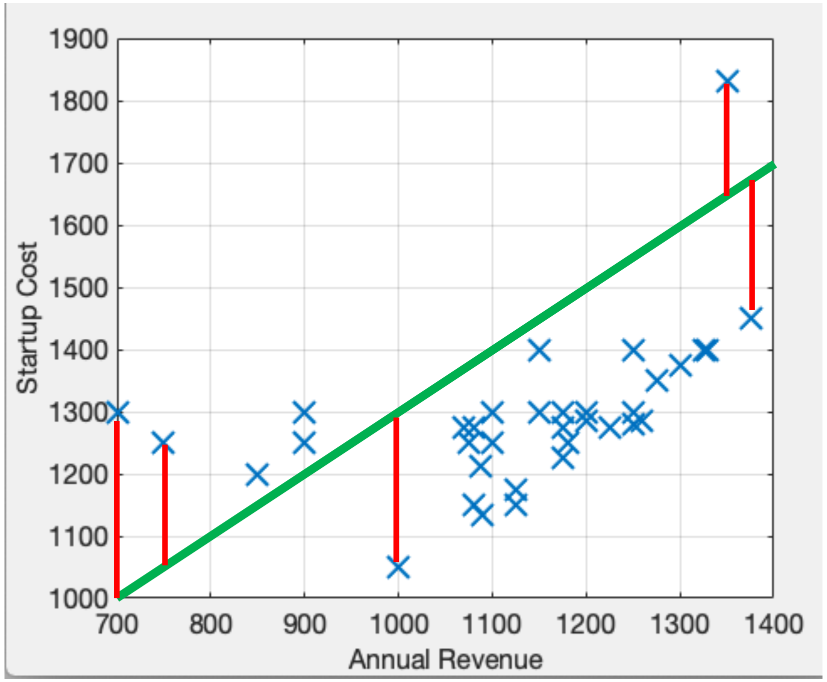
Good…

…but “***the best***”?

Machine Learning II: Cost Function

- The **Cost Function** should distinguish between two alternate hypotheses, i.e., it should be used to favor one hypothesis instead of other
- In practice, the cost function receives the **parameters of one model** and returns “***how good/bad the model is***”

- In this problem, we are interested in models that are as close as possible to the data points
- I.e., the “optimal model” will overlap exactly all the points we have in the dataset
  - Impossible, for the type of model chosen



Machine Learning II: Cost Function

- The **Cost Function** is typically expressed as **J()**
- **The cost function receives as input, the parameters of the model**
  - In this case, it receives two parameters: (θ_1, θ_2)
- Hence, the cost function is formally J: R^2 → R

$$J(\theta_1, \theta_2) = \frac{1}{2N} \sum_{i=1}^{N} \left(h_{\theta}(x^{(i)}) - y^{(i)}\right)^2$$

Why ??

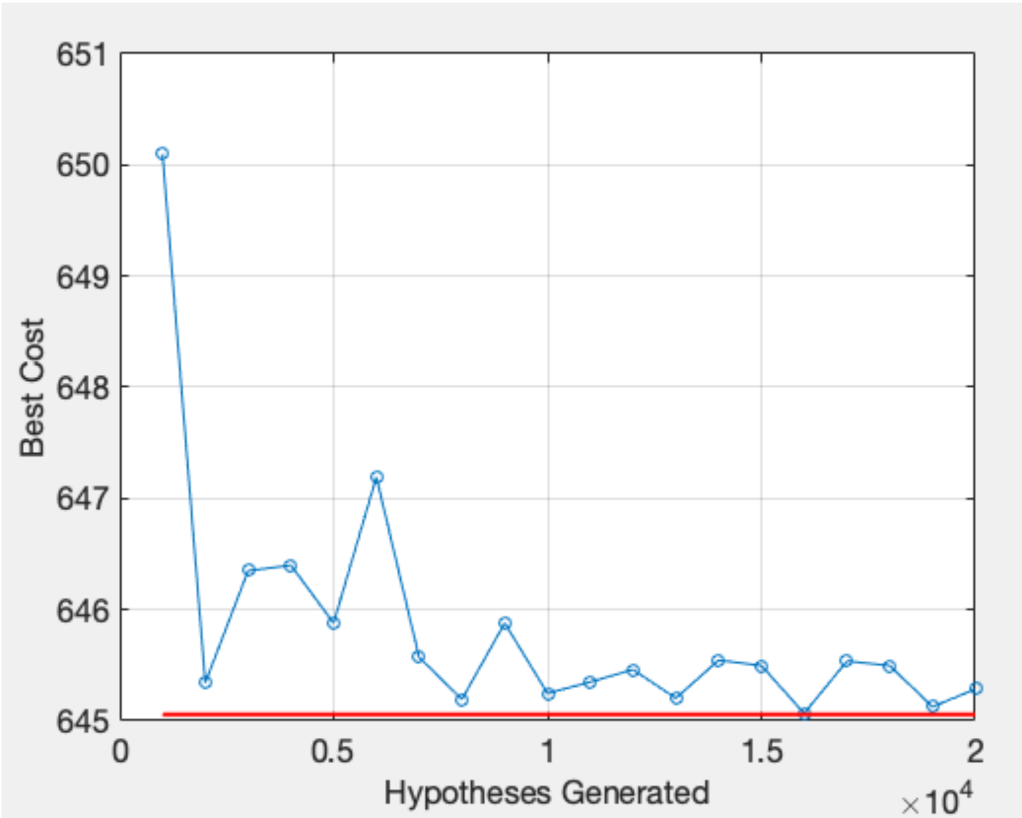
- In practice, this function sums up all the Euclidean distances between the targets (ground truth) in our dataset and the values given by the model at each point
  - Clearly, if one model is optimal $h_{\theta}(x^{(i)}) == y^{(i)}$ and **J=0**
- At the (almost) end of this story, Machine Learning is about minimizing **J()**

Machine Learning III: Optimization

- *“Computers are so fast these days, what if we simply generate millions of different hypotheses and pick the **best one?**”*
  - *This is the “brute-force” approach, that (only) in problems of reduced dimensionality might lead to reasonable results.*

- The plot given at right compares the best model obtained “*by chance*” (dependent variable), with respect to the numbers of models randomly created (independent variable).

- In some cases, the best random model was “close” to the optimal model:
  - Cost 645.05
  - $(\theta_1, \theta_2) = (0.376, 867.6)$



Machine Learning III: Optimization

- How to obtain the best possible model?
- Find the (θ_1, θ_2) parameters that minimize J()
- Formally:

$$\boldsymbol{\theta}^{*} = \arg\min_{\boldsymbol{\theta}} J(\theta_1, \theta_2)$$

- In practice, this is an optimization problem in 2D space, that requires to find the derivative of J() with respect to $\boldsymbol{\theta}$.
- Recall from single variable calculus that (assuming a function **f** is diferentiable) the minimum $x^{*}$of **f** has the property that the derivative df/dx is zero at $x=x^{*}$
  - An analogous result holds in the multivariate case:

$$\frac{\int}{\int \boldsymbol{\theta}} J(\boldsymbol{\theta}) \quad \rightarrow \quad \frac{\int}{\int \theta_1} J(\boldsymbol{\theta}), \quad \frac{\int}{\int \theta_2} J(\boldsymbol{\theta})$$

Partial Derivatives

Machine Learning Optimization: Closed-Form

- Minimizing J() is equivalent to minimize:

$$\sum_{i=1}^{N}(\theta_1 x^{(i)} + \theta_2 - y^{(i)})^2$$

$$\mathbf{X} = \begin{bmatrix} x_1 & \mathbf{1} \\ x_2 & \mathbf{1} \end{bmatrix}$$

Bias!!

- *Using matrix algebra, we know that*

$$\sum_{i=1}^{N}(\theta_1 x^{(i)} + \theta_2 - y^{(i)})^2 = (\boldsymbol{X\theta} - \mathbf{y})^{\mathrm{T}} (\boldsymbol{X\theta} - \mathbf{y})$$

- *So, we are interested in minimizing the above expression, i.e.,*

$$\frac{\int}{\int \boldsymbol{\theta}} (\boldsymbol{X\theta} - \mathbf{y})^{\mathrm{T}} (\boldsymbol{X\theta} - \mathbf{y}) = 0$$

- *Applying the distributive property. Also:*

$$(\mathbf{AB})^{\mathrm{T}} = \boldsymbol{A}^T \boldsymbol{B}^T$$

$$\frac{\int}{\int \boldsymbol{\theta}} \boldsymbol{X}^T\boldsymbol{\theta}^T\boldsymbol{X\theta} - \boldsymbol{X}^T\boldsymbol{\theta}^T\mathbf{y} - \mathbf{y}^T\boldsymbol{X\theta} + \mathbf{y}^{\mathrm{T}}\mathbf{y}) = 0$$

r scalar, $r^{\mathrm{T}} = r$
$\mathbf{y}^T\boldsymbol{X\theta}$ is scalar.

$$\mathbf{y}^T\boldsymbol{X\theta} = (\mathbf{y}^T\boldsymbol{X\theta})^T = \mathbf{y}\boldsymbol{X}^T\boldsymbol{\theta}^T$$

Machine Learning Optimization: Closed-Form

- Simplifying:

$$\frac{\int}{\int \boldsymbol{\theta}} \boldsymbol{X}^T\boldsymbol{\theta}^T\boldsymbol{X}\boldsymbol{\theta} - 2 * \boldsymbol{X}^T\boldsymbol{\theta}^T\mathbf{y} + \mathbf{y}^T\mathbf{y} = 0$$

Matrix Derivatives:

$$\frac{\int(\boldsymbol{AX})}{\int \boldsymbol{X}} = \boldsymbol{A}^T$$

$$\frac{\int(\boldsymbol{X}^T\boldsymbol{X})}{\int \boldsymbol{X}} = \mathbf{2}\boldsymbol{X}$$

$$\frac{\int(\boldsymbol{X}^T\boldsymbol{AX})}{\int \boldsymbol{X}} = \boldsymbol{AX} + \boldsymbol{AT}\,\boldsymbol{X}$$

- *Applying the derivatives rules:*

$$2\,\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} - 2 * \boldsymbol{X}^T\mathbf{y} = 0$$

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{X}^T\mathbf{y} = 0$$
$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} = \boldsymbol{X}^T\mathbf{y}$$
$$(\boldsymbol{X}^T\boldsymbol{X})^{-1}(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{\theta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\mathbf{y}$$

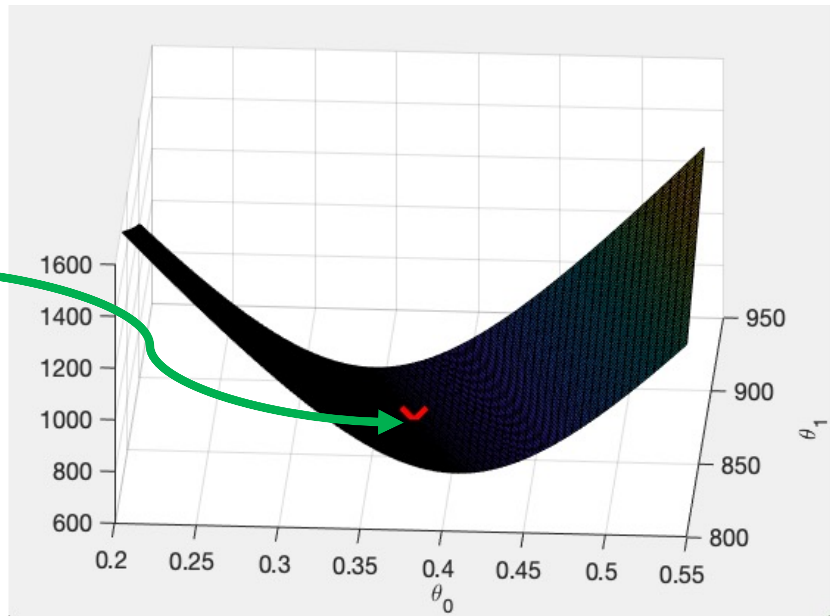
- *Solving with respect to* $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Machine Learning Optimization: Closed-Form

- The closed-form solution should be preferred for "smaller" datasets
  - When computing the matrix inverse is not a concern.
- For very large datasets, obtaining $(\mathbf{X}^T\mathbf{X})^{-1}$ can be extremely costly
  - $\mathbf{X}$ has N x (d+1) dimensions
- Also, there are cases where the $(\mathbf{X}^T\mathbf{X})^{-1}$ not exists
  - e.g., the matrix is non-invertible (singular) in case of perfect multicollinearity

If succeeded, the Closed-Form enables us to obtain the optimal configuration of the hypothesis $\theta*$ in a single step



Machine Learning Optimization: Partial Derivatives

- As we have seen, the goal is to obtain the $\boldsymbol{\theta}$ parameterization that minimizes J():

$$\mathrm{J}(\theta_1, \theta_2) = \frac{1}{2N} \sum_{i=1}^{N} \left(\theta_1 x^{(i)} + \theta_2 - y^{(i)}\right)^2$$

- *(a+b)' = a' + b'*

$$\frac{\int}{\int \theta_1} \mathrm{J}(\boldsymbol{\theta}) = \frac{1}{\cancel{2}N} \sum_{i=1}^{N} \cancel{2} \left(\theta_1 x^{(i)} + \theta_2 - y^{(i)}\right) x^{(i)}$$

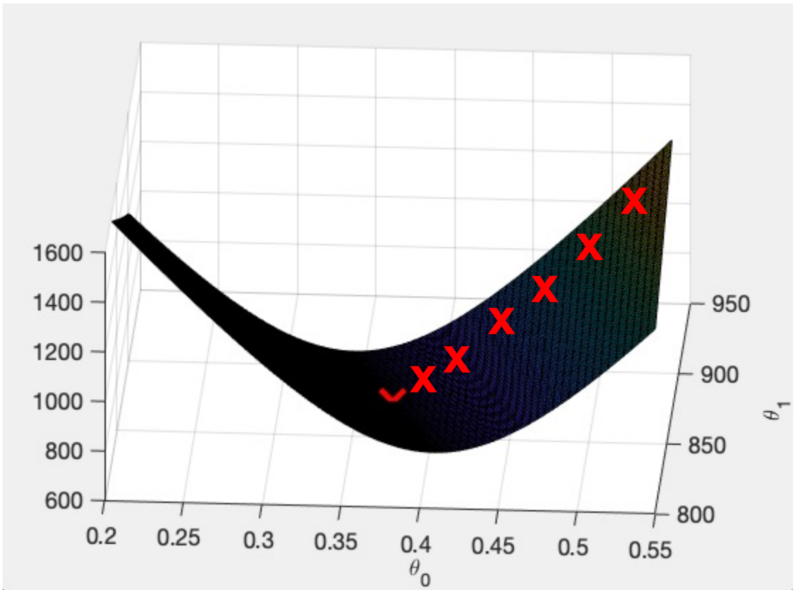
That's why!

$$\frac{\int}{\int \theta_2} \mathrm{J}(\boldsymbol{\theta}) = \frac{1}{\cancel{2}N} \sum_{i=1}^{N} \cancel{2} \left(\theta_1 x^{(i)} + \theta_2 - y^{(i)}\right)$$

Machine Learning Optimization: Gradient Descent

- In most practical cases, the Closed-Form is hard to obtain, and the solution is to use the "**Gradient Descent**" optimization version:
- Algorithm:
  1. Start with some random $\boldsymbol{\theta}$ configuration. $\boldsymbol{\theta}^{(0)}$
  2. Change iteratively (and slightly) $\boldsymbol{\theta}$, to reduce J($\boldsymbol{\theta}$)
     1. $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \Delta \quad \frac{\int}{\int \boldsymbol{\theta}} J(\boldsymbol{\theta})$
  3. (Hopefully) end up in a minimum

The rationale is to iteratively move in the steepest descend direction, in order to reach the (eventually local) minimum

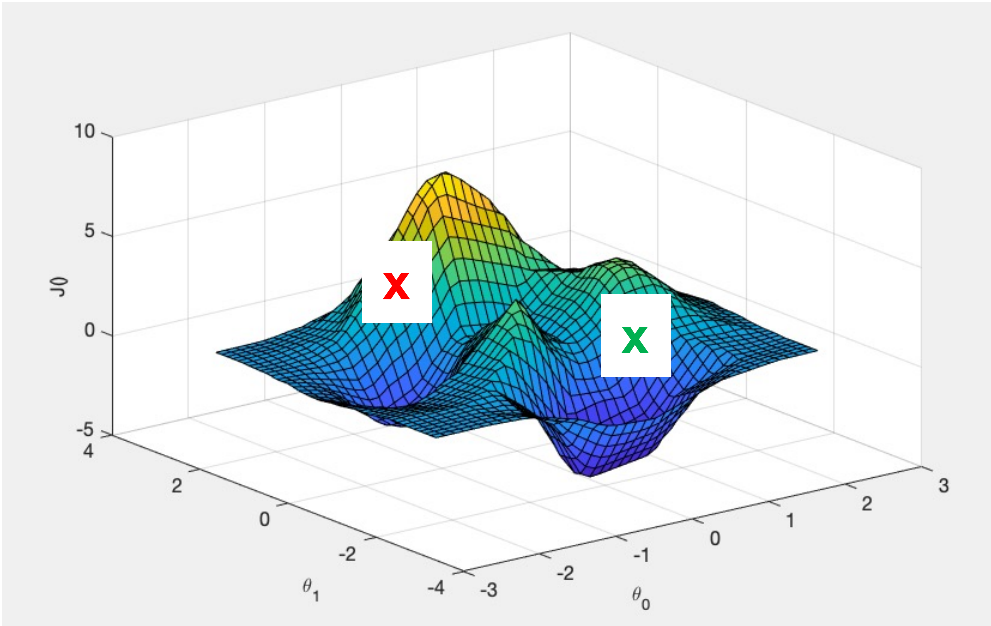


Machine Learning Optimization: Gradient Descent

$$\theta_0 = \theta_0 - \Delta \frac{1}{N} \sum_{i=1}^{N} \left(\theta_1 x^{(i)} + \theta_2 - y^{(i)}\right) x^{(i)}$$

$$\theta_1 = \theta_1 - \Delta \frac{1}{N} \sum_{i=1}^{N} \left(\theta_1 x^{(i)} + \theta_2 - y^{(i)}\right)$$

Main assumption in Gradient Descent: **Convexity**!



Machine Learning Optimization: Gradient Descent

- Learning Rate
  - Too large values lead to divergence
    - The optimal value of J() is not achieved, i.e., the best $\boldsymbol{\theta}$ configuration is not found
  - Too small values slow down the learning process.
- Remark
  - The update of parameters should be done simultaneously:

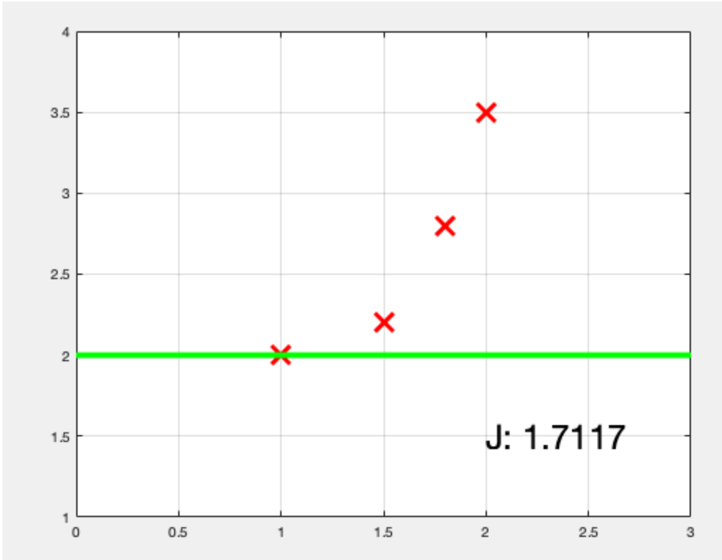
- $\boldsymbol{\theta_1}^{(t+1)} = \boldsymbol{\theta_1}^{(t)} - \Delta \frac{\int}{\int \boldsymbol{\theta_1}} J(\boldsymbol{\theta})$
- $\boldsymbol{\theta}_2^{(t+1)} = \boldsymbol{\theta_2}^{(t)} - \Delta \frac{\int}{\int \boldsymbol{\theta_2}} J(\boldsymbol{\theta})$

- $aux_1 = \boldsymbol{\theta_1}^{(t)} - \Delta \frac{\int}{\int \boldsymbol{\theta_1}} J(\boldsymbol{\theta})$
- $aux_2 = \boldsymbol{\theta_2}^{(t)} - \Delta \frac{\int}{\int \boldsymbol{\theta_2}} J(\boldsymbol{\theta})$
- $\boldsymbol{\theta_1}^{(t+1)} = aux_1$
- $\boldsymbol{\theta_2}^{(t+1)} = aux_2$

Gradient Descent Exercise

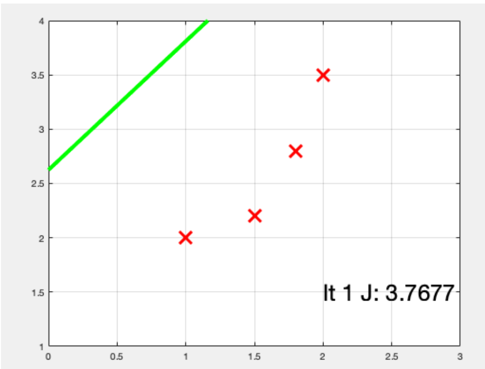
- Consider the following tiny dataset. Use the gradient descent algorithm to obtain the optimal linear regression hypothesis:
  - Start with $\boldsymbol{\theta_1}, \boldsymbol{\theta_2}$ = (0,2)
  - Use Δ=0.1

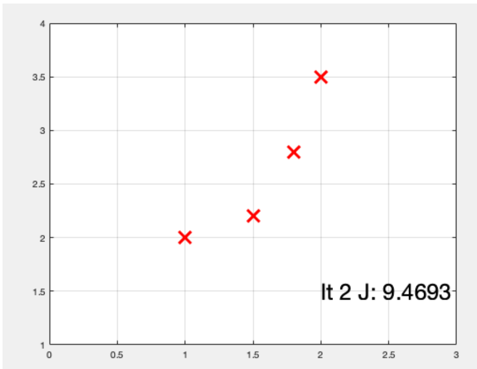
X	Y
1	2
1.5	2.2
1.8	2.8
2	3.5

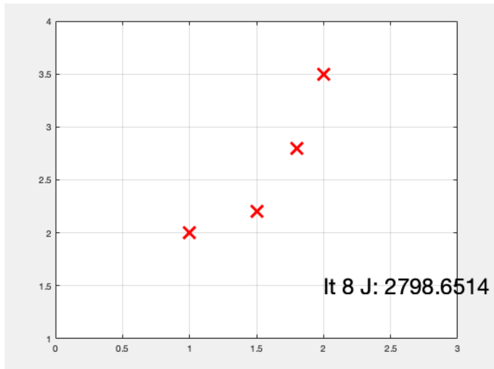


Gradient Descent Exercise

- Consider the following tiny dataset. Use the gradient descent algorithm to obtain the optimal linear regression hypothesis:
  - Start with $\boldsymbol{\theta_1}, \boldsymbol{\theta_2}$ = (0,2)
    - Use Δ=1
    - Use Δ=0.1
    - Use Δ=0.5
- Δ=1



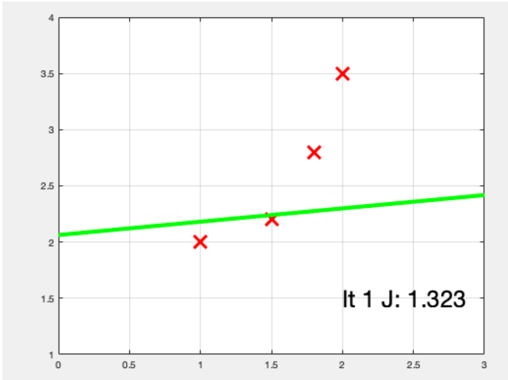


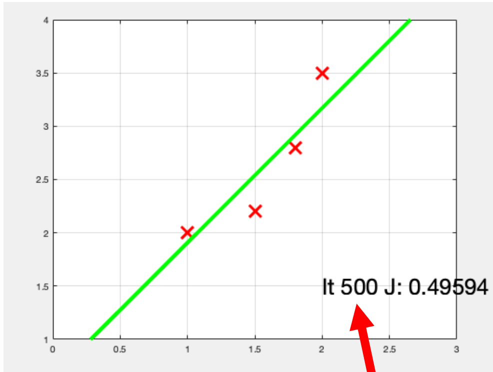


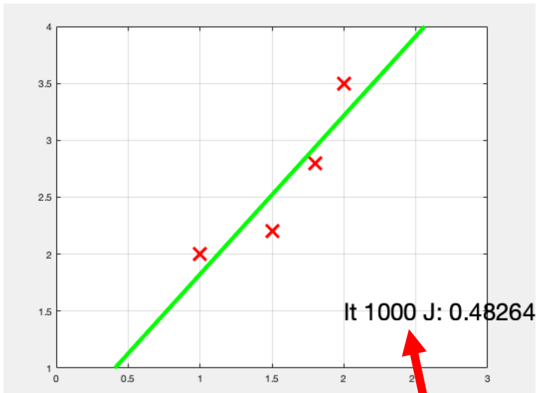
Diverged!!

Gradient Descent Exercise

- Δ=0.1



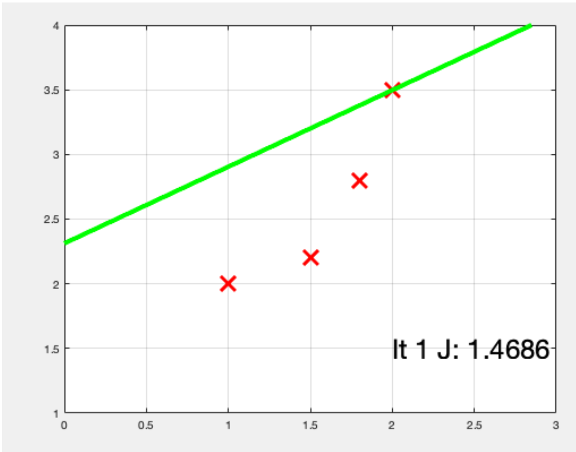


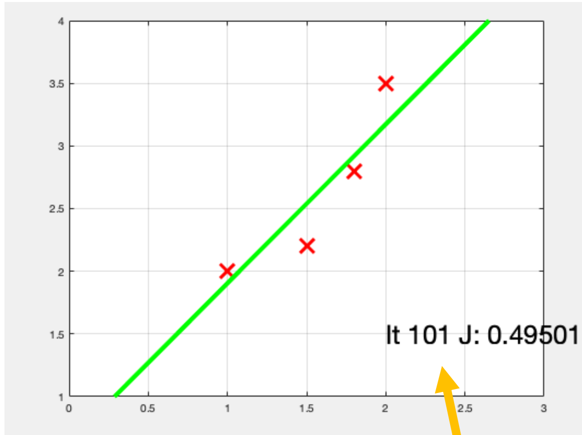


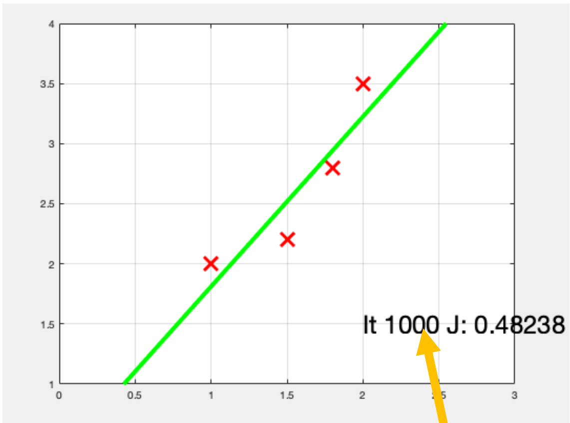
Too slow…

Gradient Descent Exercise

- Δ=0.5



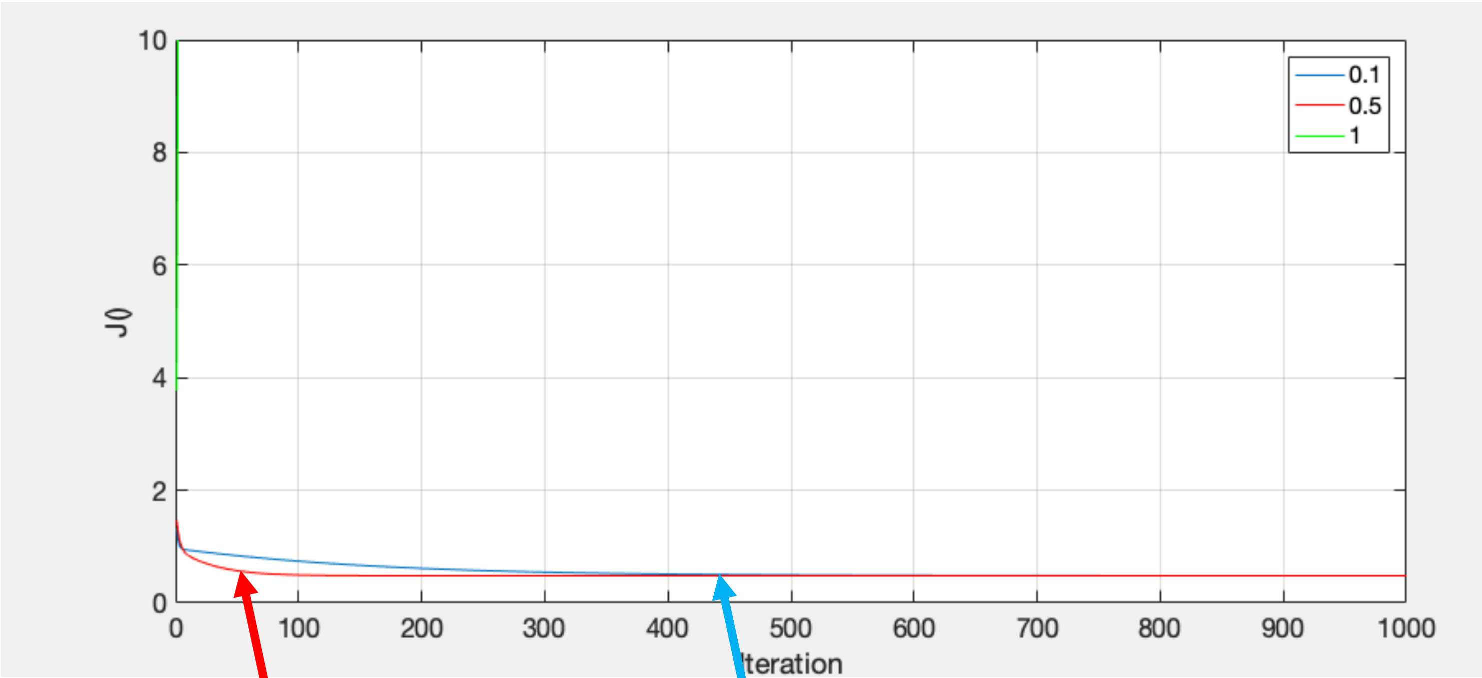




Better…

Gradient Descent Exercise

- Δ=1 vs. Δ=0.1 vs. Δ=0.5



Acceptable solution (Δ= 0.5) Acceptable solution (Δ= 0.1)

- Stop Criteria:
  - “T” iterations
  - While it stops to improv (i.e., $\boldsymbol{J}^{(t+1)} - \boldsymbol{J}^{(t)} < \varepsilon$)