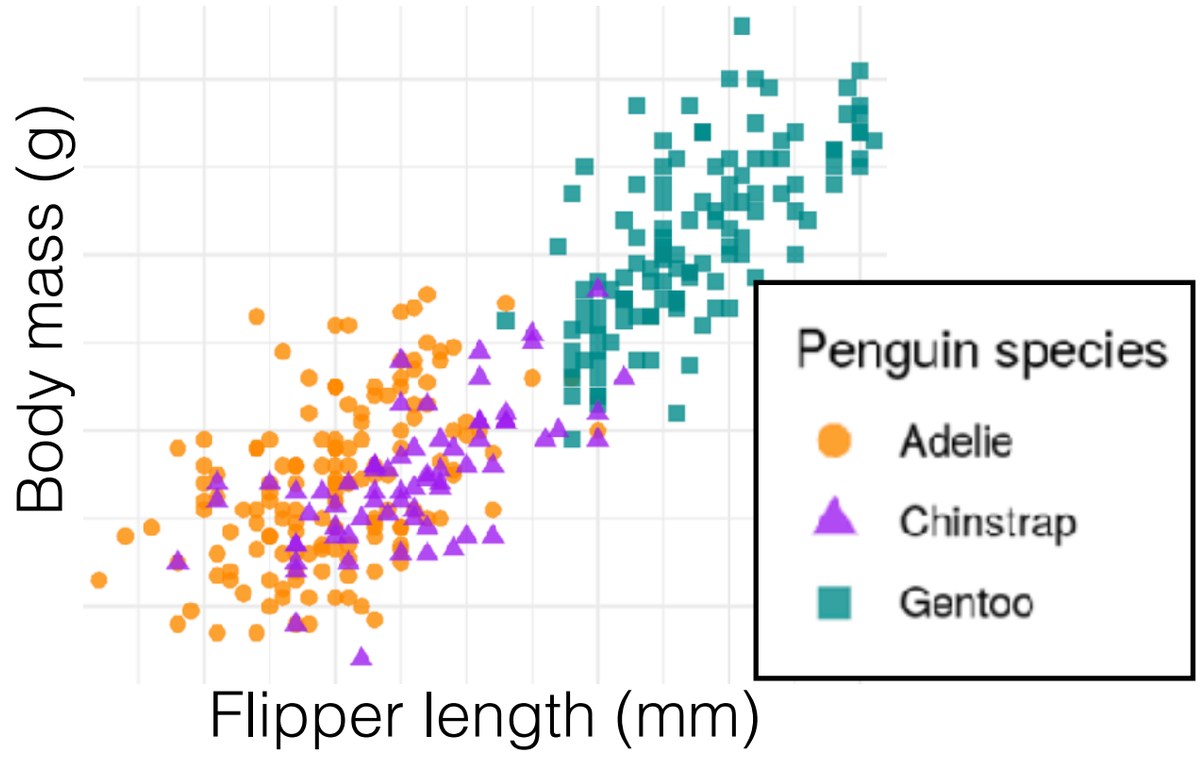
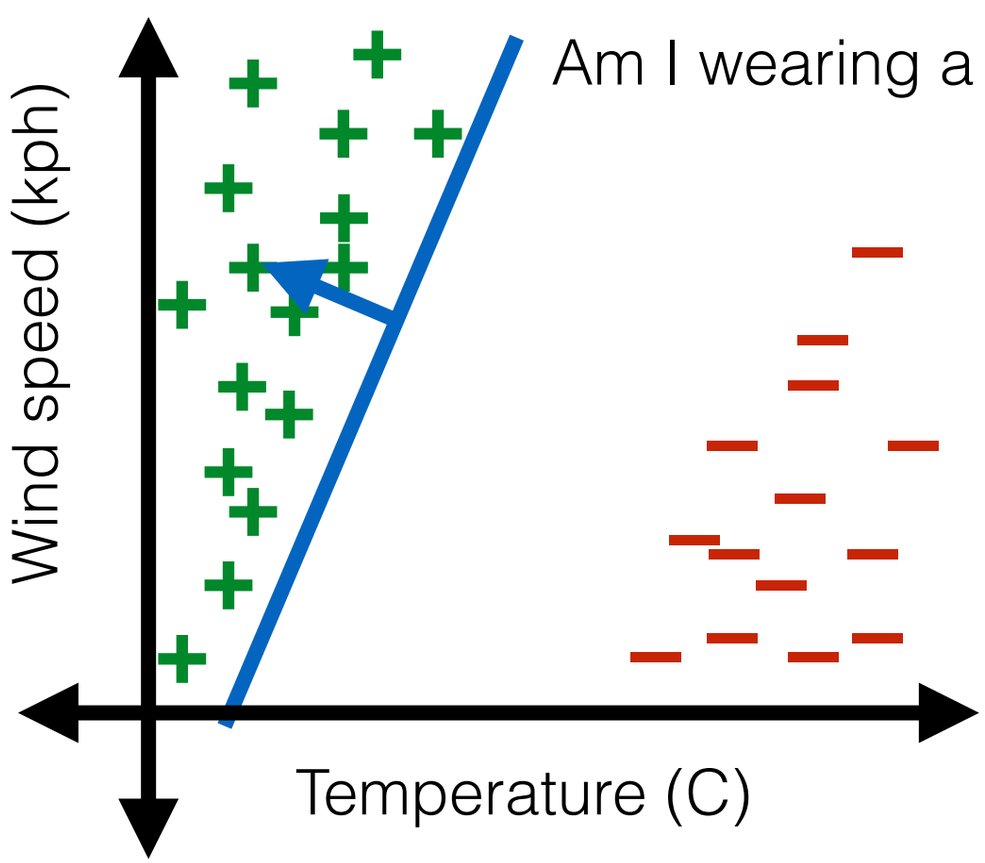
Recall

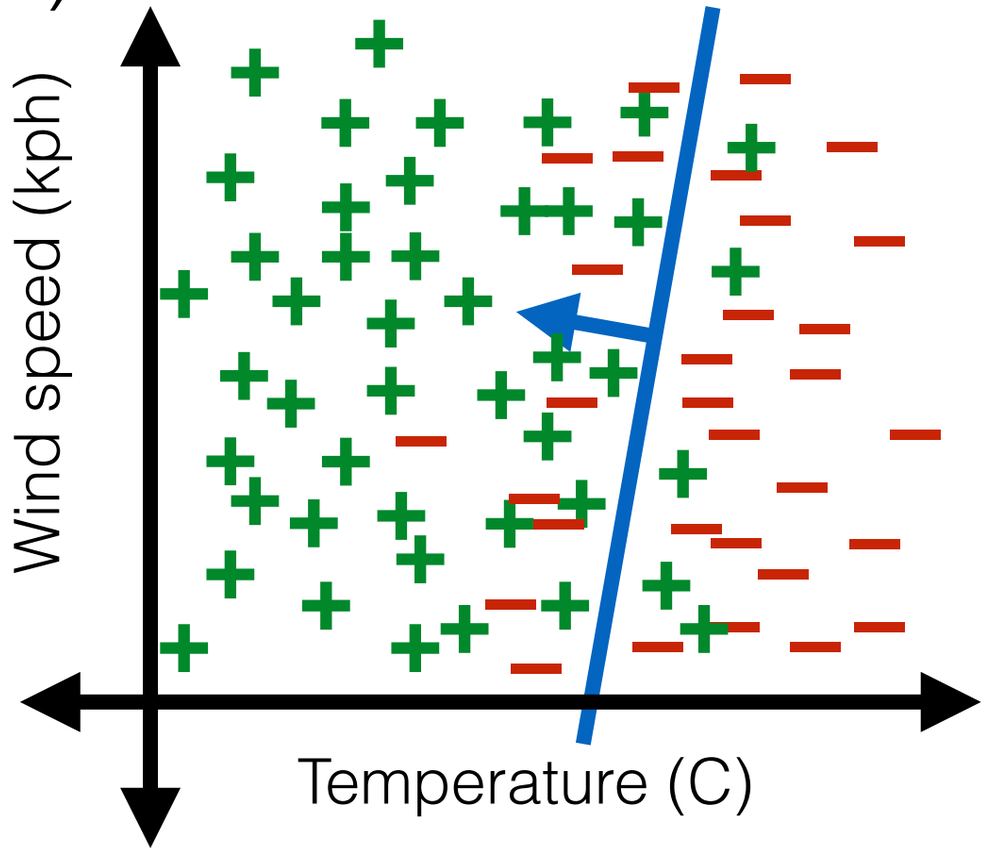
- Perceptron struggles with data that's not linearly separable



Notice

- Perceptron doesn't have a notion of uncertainty (how well do we know what we know?)

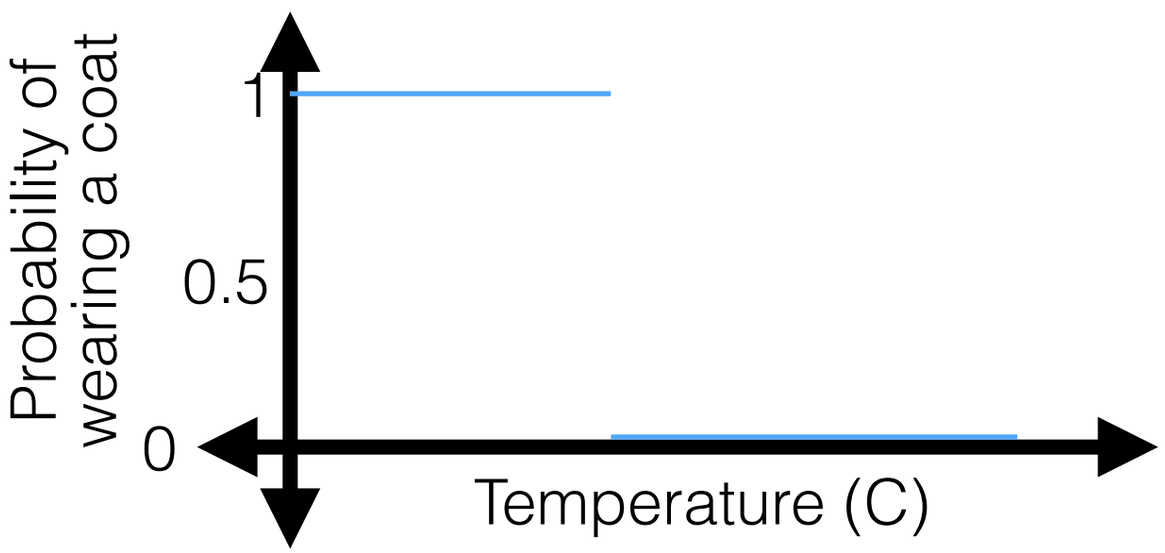


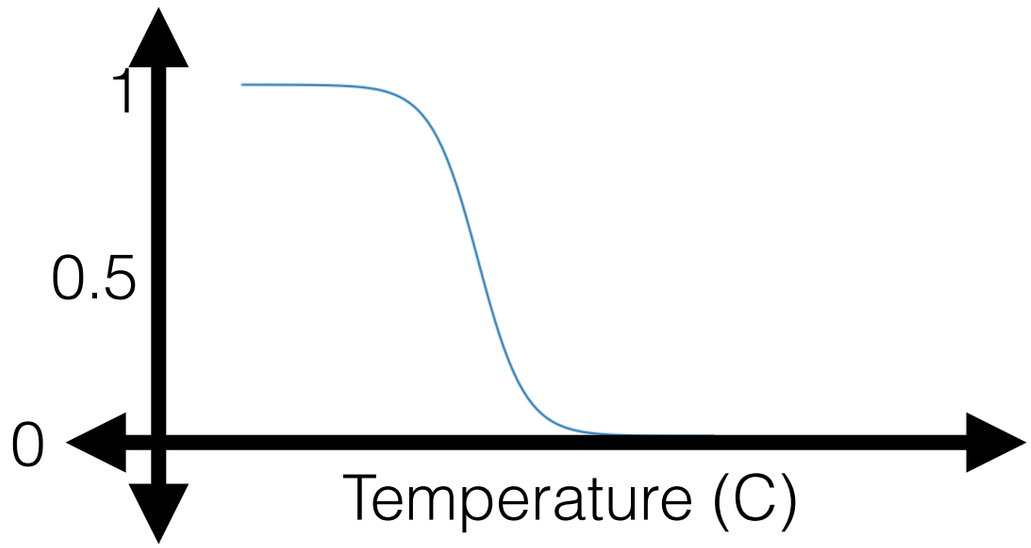


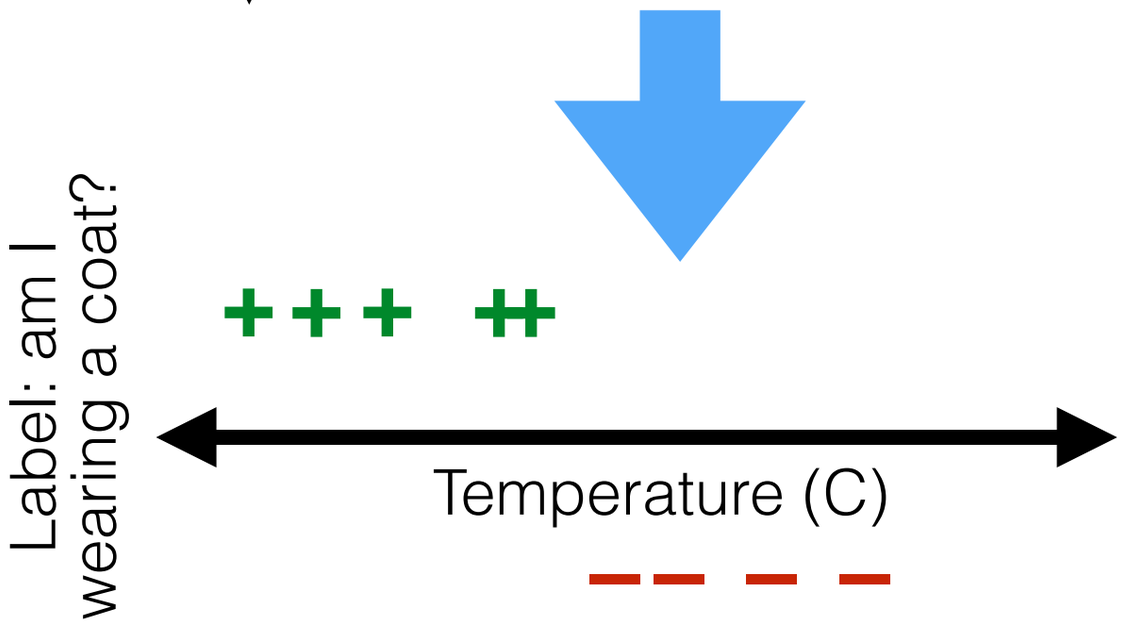
Lecture 4 * Perceptron doesn't tell us about uncertainty (Slide - 2 -)

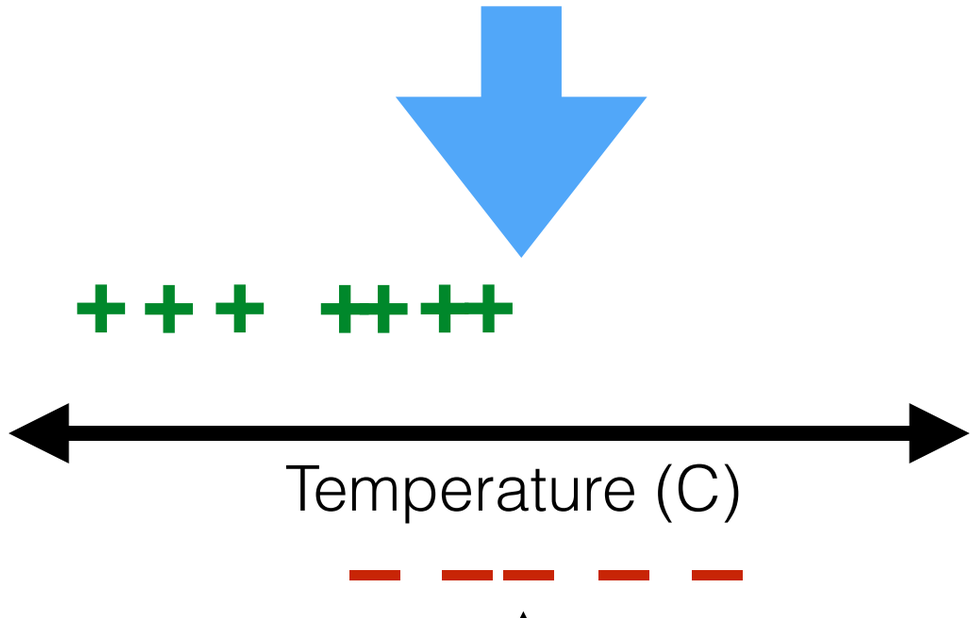
- Notice from the slides that there is a large gap between the labels. So, any classifier lying within that gap will be perfect.
- But since, there is this large gap, there is no uncertainty. For example, if you take a classifier very close to the positive labels in that gap. And for prediction for a new point, lying in that gap, we predict negative. That would be technically correct.
- But, since there is no data near that point (since it's in a gap) it would be logically ambiguous .... because there is no previous data.
- Perceptron just spits out a classifier. It doesn't tell us anything about this uncertainty.

Capturing uncertainty



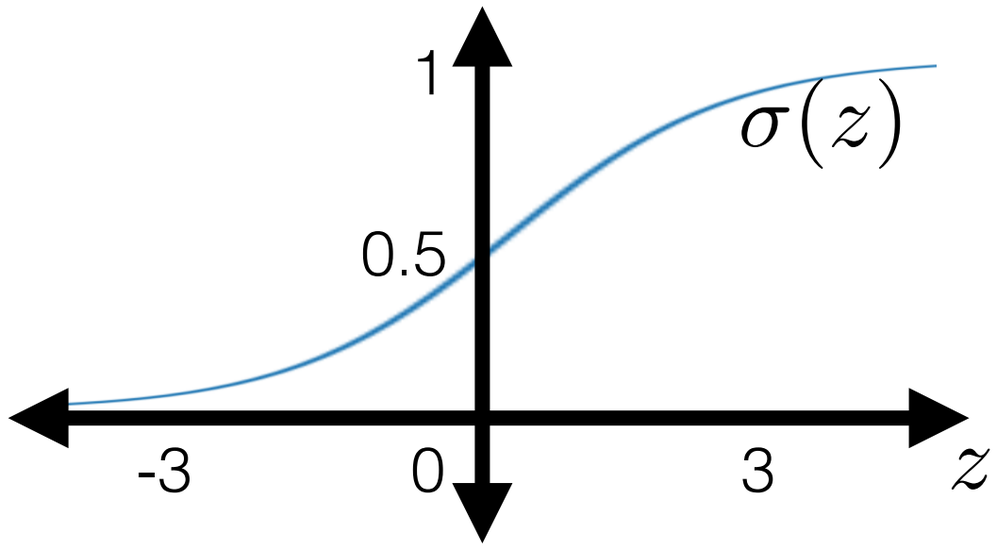






- How to make this shape?
  - Sigmoid/logistic function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



Capturing Intensity (Sigmoid Function) (slide - 3 -)

- Here, we can see that a different version of this sigmoid function can be used to describe the PDF of wearing a coat vs. temperature.
- We need this sigmoid function to be squished & more steep vertically. Thus, we multiply 'z' by a constant to increase its steepness.
- Now, we wanna flip the function horizontally. So, we multiply 'z' by (-1). But that can already be covered by the constant multiplier.

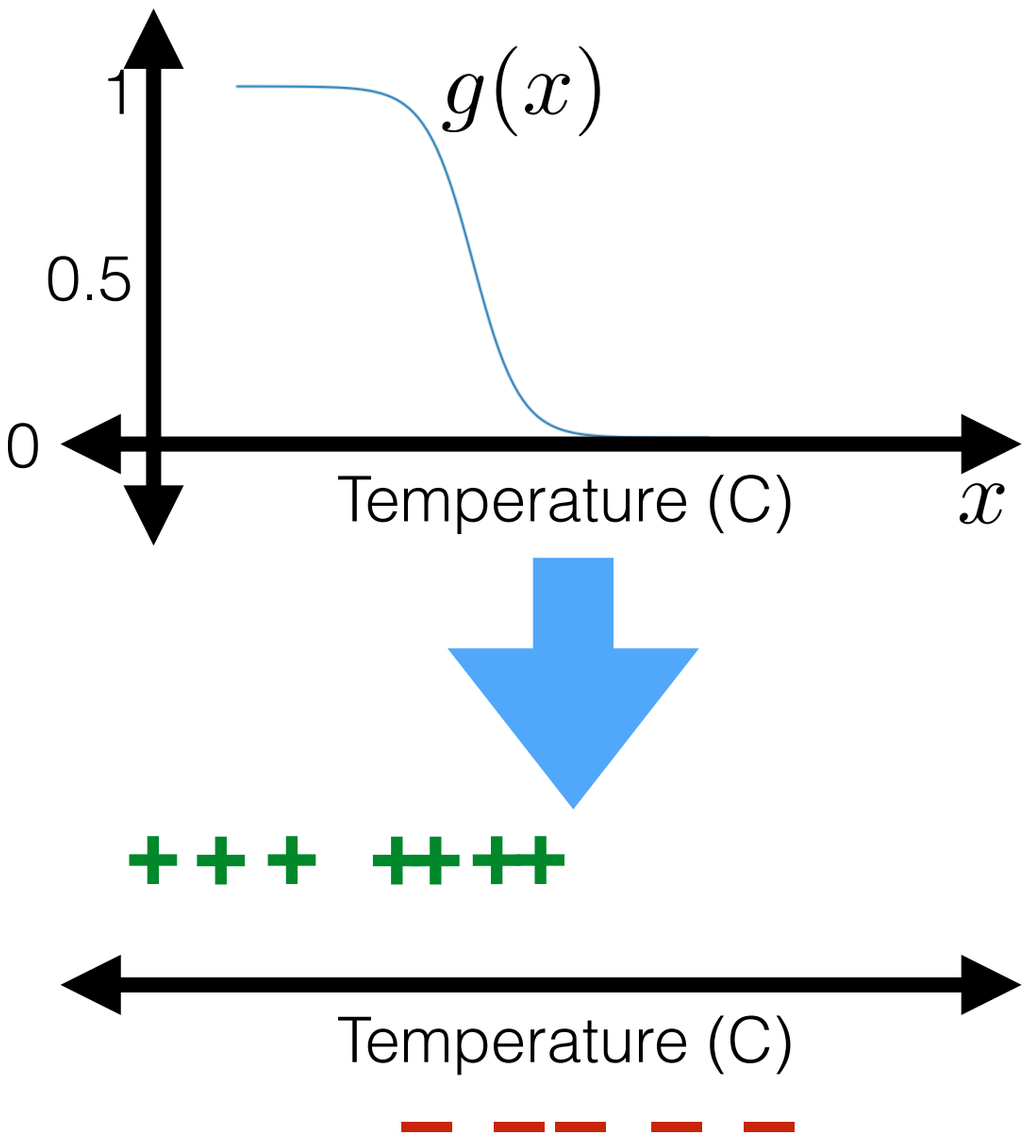
- Now, we wanna affect this function horizontally, so we also add a constant to "z".
- The ~~new form~~ upgraded sigmoid function can be seen in the next slide.

Capturing uncertainty

1 feature:

$$g(x) = \sigma(\theta x + \theta_0)$$

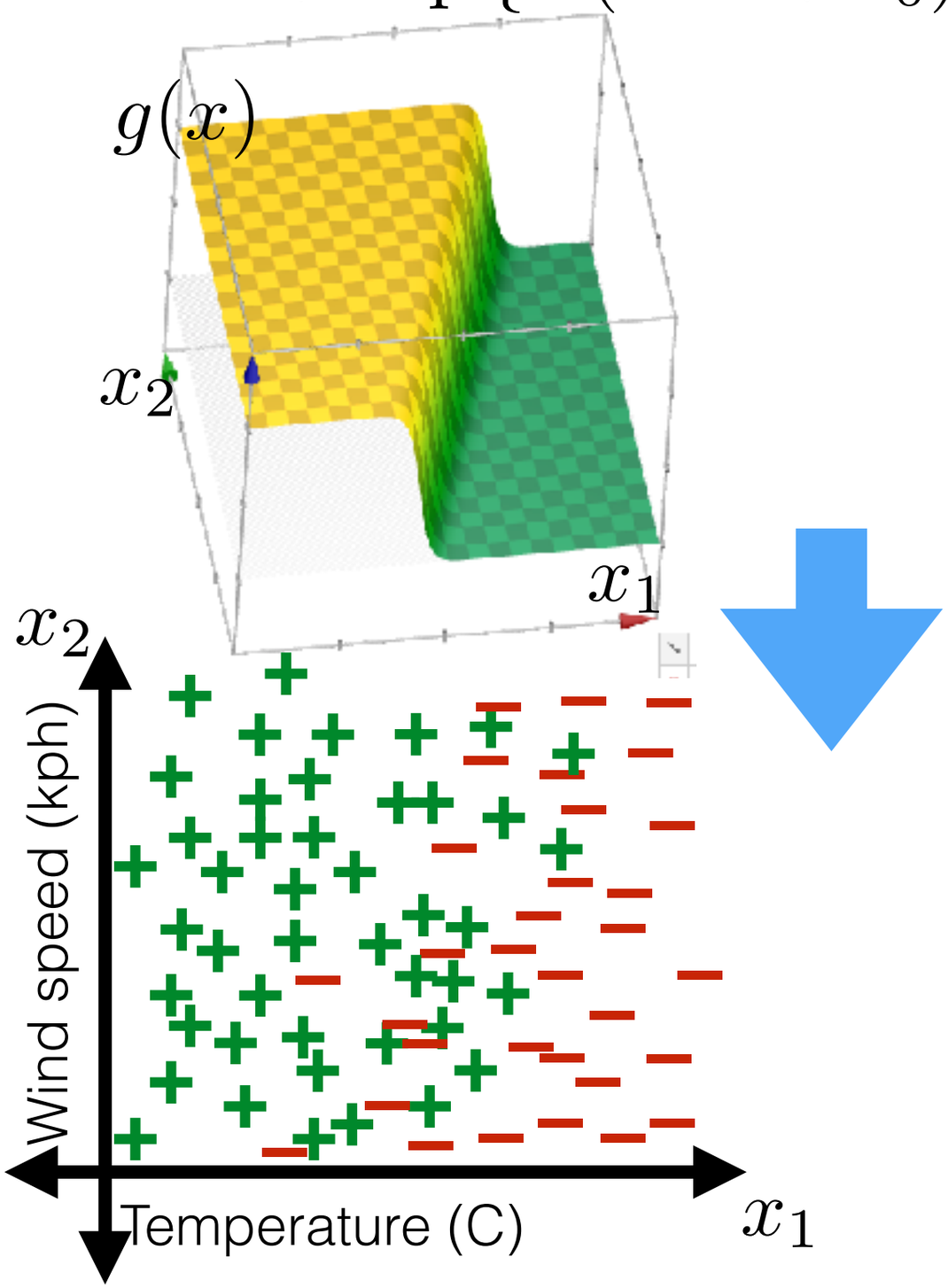
$$= \frac{1}{1 + \exp\{-(\theta x + \theta_0)\}}$$



2 features:

$$g(x) = \sigma(\theta^\top x + \theta_0)$$

$$= \frac{1}{1 + \exp\{-(\theta^\top x + \theta_0)\}}$$



★ Capturing uncertainty (Slide - 5 -)
(PDF to approximate labels)

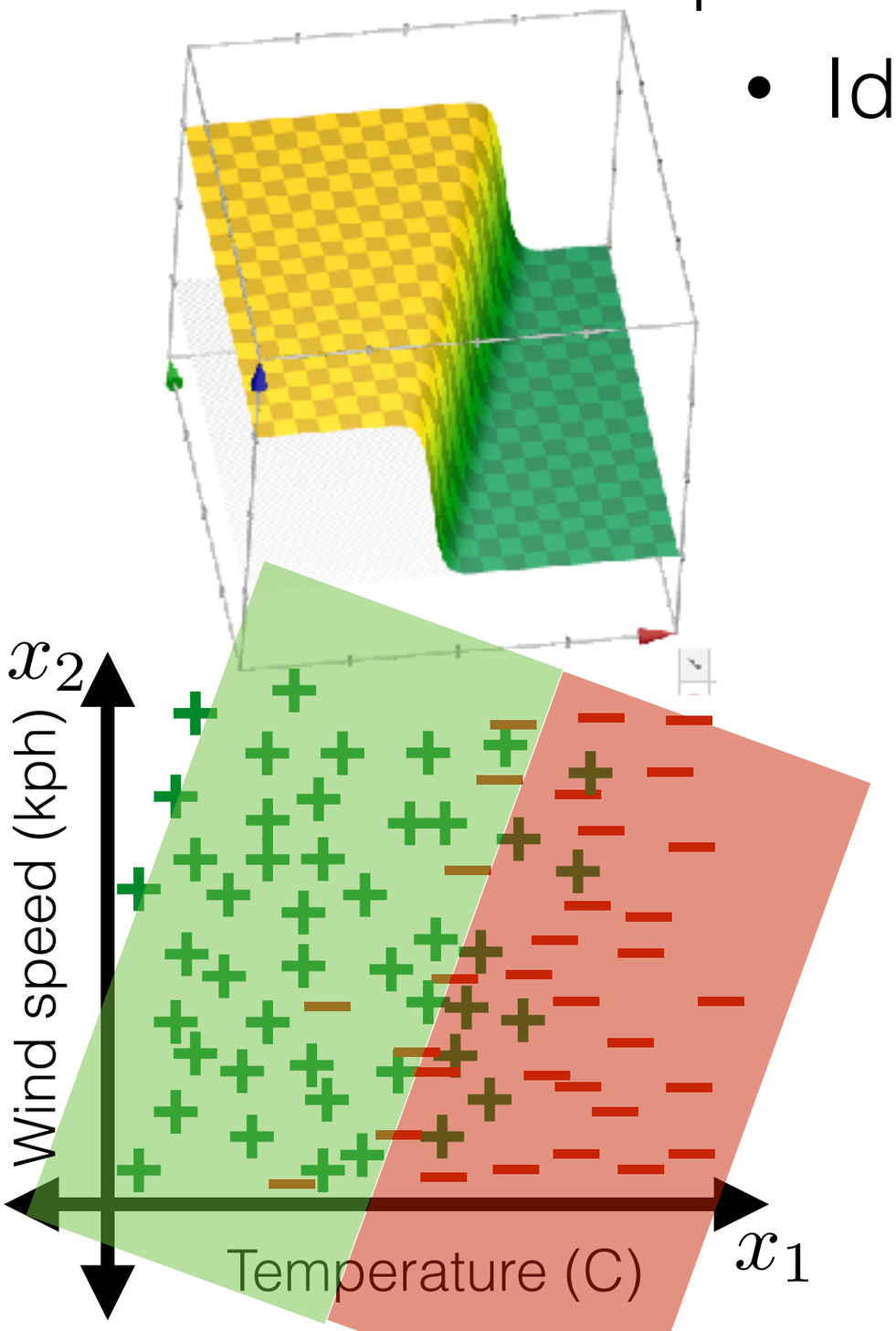
- You can sort of see that if we logically project the curve on to the feature plane & modify it, we can get our labels.
- All ~~part~~ parts where probability is non zero & non-unity you can see mixed labels & if you observe neatly enough, a sort of gradient in proportional ratio of positives to negatives according to probability.

★ NOTE:- Earlier, we had a question about perceptrons & how they don't tell us about the uncertainty. Now these PDFs solve this problem. We not just ~~get~~ produce a classifier, we also produce a PDF from existing data so that we can better predict future data & describe the certainty or uncertainty of the predicted data.

Linear logistic classification

aka logistic regression

- How do we learn a classifier (i.e. learn θ, θ_0)?
- How do we make predictions?



- Idea: predict +1 if: probability > 0.5

$$\sigma(\theta^\top x + \theta_0) > 0.5$$

$$\frac{1}{1 + \exp\{-(\theta^\top x + \theta_0)\}} > 0.5$$

$$\exp\{-(\theta^\top x + \theta_0)\} < 1$$

$$\theta^\top x + \theta_0 > 0$$

- Same hypothesis class as before! But we will get:
  - Uncertainties
  - Quality guarantees when data not linearly separable

★ Linear logistic classification (slide- 6-) (logistic regression)

- Even though we ~~have~~ can use the same hypothesis class, the learning algorithm is going to be different.
- We will change it so that we ~~also~~ get ~~insights about uncertainty~~ a linear classifier even though the data isn't linearly separable by making use of PDFs

* Linear Logistic Regression (Slide - 6 - 107)

- Here, we propose a method that: we label a point based on the PDF. We generate a PDF based on the available data. i.e. we use a training algorithm on the sigmoid function so that it fits the data

Linear logistic classification

aka logistic regression

- How do we learn a classifier (i.e. learn θ, θ_0)?

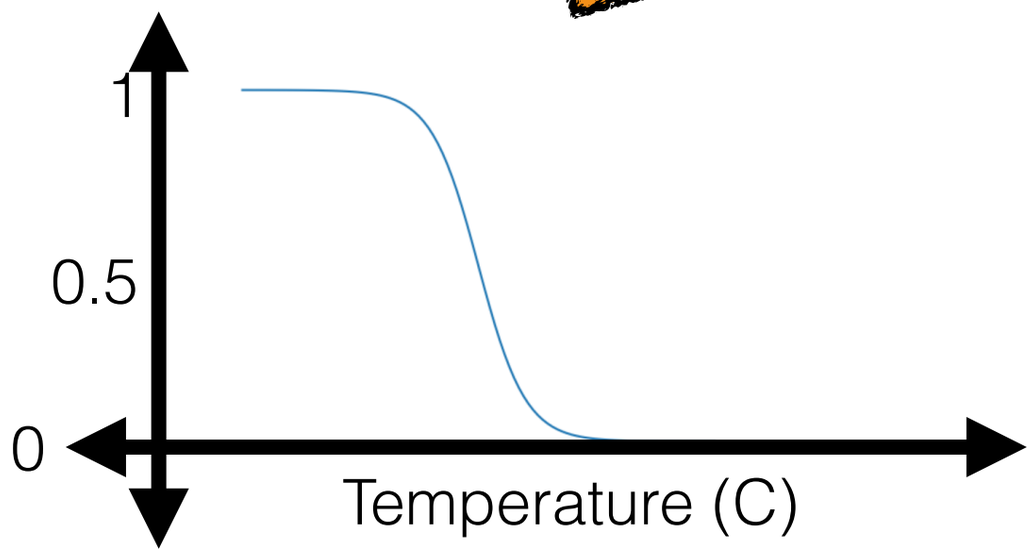
Probability(data)

$$= \prod_{i=1}^{n} \text{Probability}(\text{data point } i)$$

$$[\text{Let } g^{(i)} = \sigma(\theta^\top x^{(i)} + \theta_0)\,]$$

$$= \prod_{i=1}^{n} \begin{cases} g^{(i)} \text{ if } y^{(i)} = +1 \\ (1 - g^{(i)}) \text{ else} \end{cases}$$

$$= \prod_{i=1}^{n} (g^{(i)})^{\mathbf{1}\{y^{(i)}=+1\}} (1 - g^{(i)})^{\mathbf{1}\{y^{(i)}\neq+1\}}$$



Loss(data) = -(1/n) * log probability(data)

$$= \frac{1}{n} \sum_{i=1}^{n} -\left(\mathbf{1}\{y^{(i)} = +1\} \log g^{(i)} + \mathbf{1}\{y^{(i)} \neq +1\} \log(1 - g^{(i)})\right)$$

Negative log likelihood loss (*g* for guess, *a* for actual):

$$-L_{\text{nll}}(g, a) = (1\{a = +1\} \log g + 1\{a \neq +1\} \log(1 - g))$$

★ Linear Logistic Classification (Slide - 7 -)
(How do we learn θ, θ_0)

- Notice that $g^{(i)} = \sigma(\theta^T x^{(i)} + \theta_0)$ is the probability of you wearing at coat at $x^{(i)}$ data point.
- Now, while training, if the label is $+1$ i.e. you are wearing a coat, then you also want your probability fuction to be maximum. (we predict +1 for prob > 0.5. Hence, while training, for +1 label, we want maximum probability. We want the sigmoid function to be as good as possible)
- Now, while training, if the label is -1 i.e. you are not wearing a coat, then you also want your PDF to have minimum probability at that point. ~~But the minimum probability is zero~~ But, we don't wanna minimize something & maximize another at the same ~~point~~ time.
- Hence, for -1 label, instead of minimizing $g = \sigma(\theta^T x + \theta_0)$ we maximize: $(1 - g)$ which is the same thing, because they are complementary events.

- Now, in the 3rd equation, the power or the index ~~of~~ term of the probabilities have a certain meaning, $\mathbb{1}\{y^{(i)} = +1\}$ means that this term will take value 1 if $y^{(i)} = +1$ & 0 for anything else. i.e. $y^{(i)} = -1$

$1\{y^{(i)} \neq +1\}$ means that this term will take value 1 if $y^{(i)} \neq +1$ & 0 otherwise.
This notation isn't exclusive to this equation. This type of if-else notation is used everywhere & you need to be able to understand it.

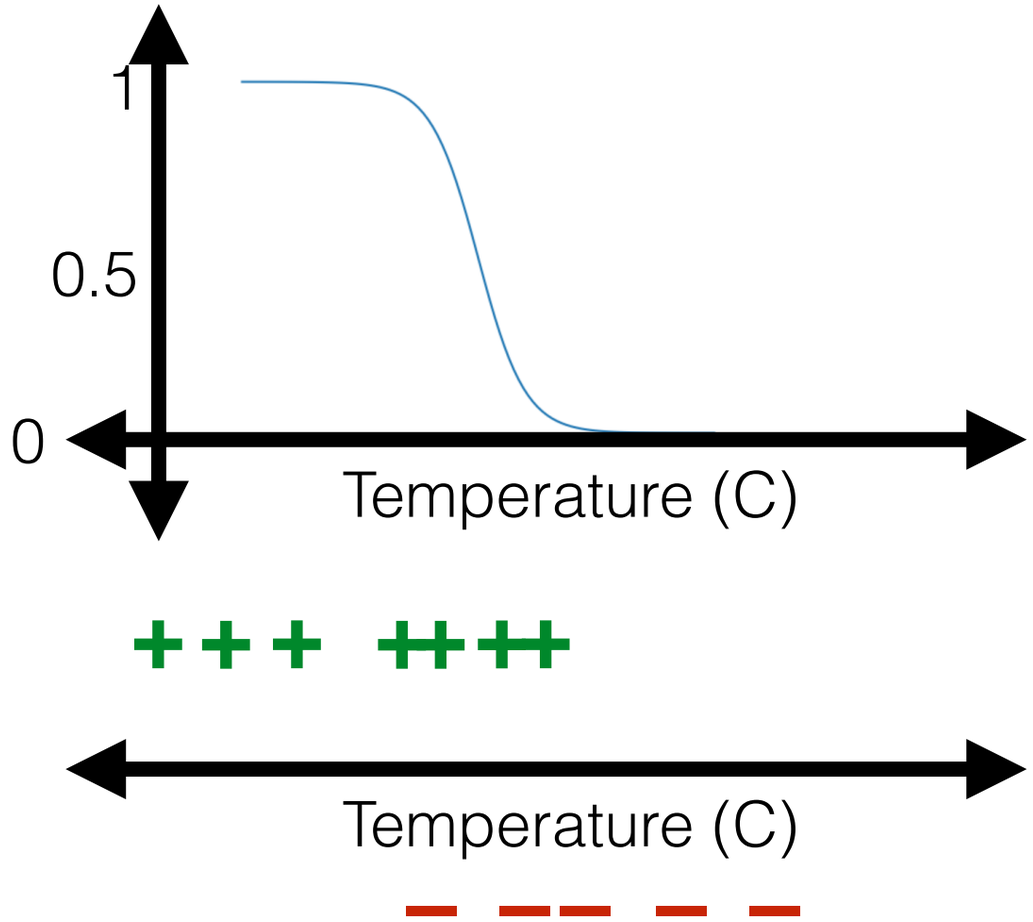
- Now, for calculation of loss or multiplication ~~of~~ of these probabilities, we use logarithm just ~~for~~ simplicity of calculation using computers.
- Since all probabilities < 1 & > 0, their product will be really small. ~~But their log will be > 1~~ But the magnitude of their log will be > 1. Hence, we use log.

Linear logistic classification

aka logistic regression

- How do we learn a classifier (i.e. learn θ, θ_0)?
- Want to find parameter values to minimize average (negative log likelihood) loss across the data

$$J_{\text{lr}}(\Theta) = J_{\text{lr}}(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} L_{\text{nll}}(\sigma(\theta^\top x^{(i)} + \theta_0), y^{(i)})$$



Gradient descent

- Gradient $\nabla_\Theta f = \left[\frac{\partial f}{\partial \Theta_1}, \ldots, \frac{\partial f}{\partial \Theta_m}\right]^\top$
  - with $\Theta \in \mathbb{R}^m$

`Gradient-Descent` $(\Theta_{\text{init}}, \eta, f, \nabla_\Theta f, \epsilon)$

`Initialize` $\Theta^{(0)} = \Theta_{\text{init}}$

`Initialize t = 0`

`repeat`

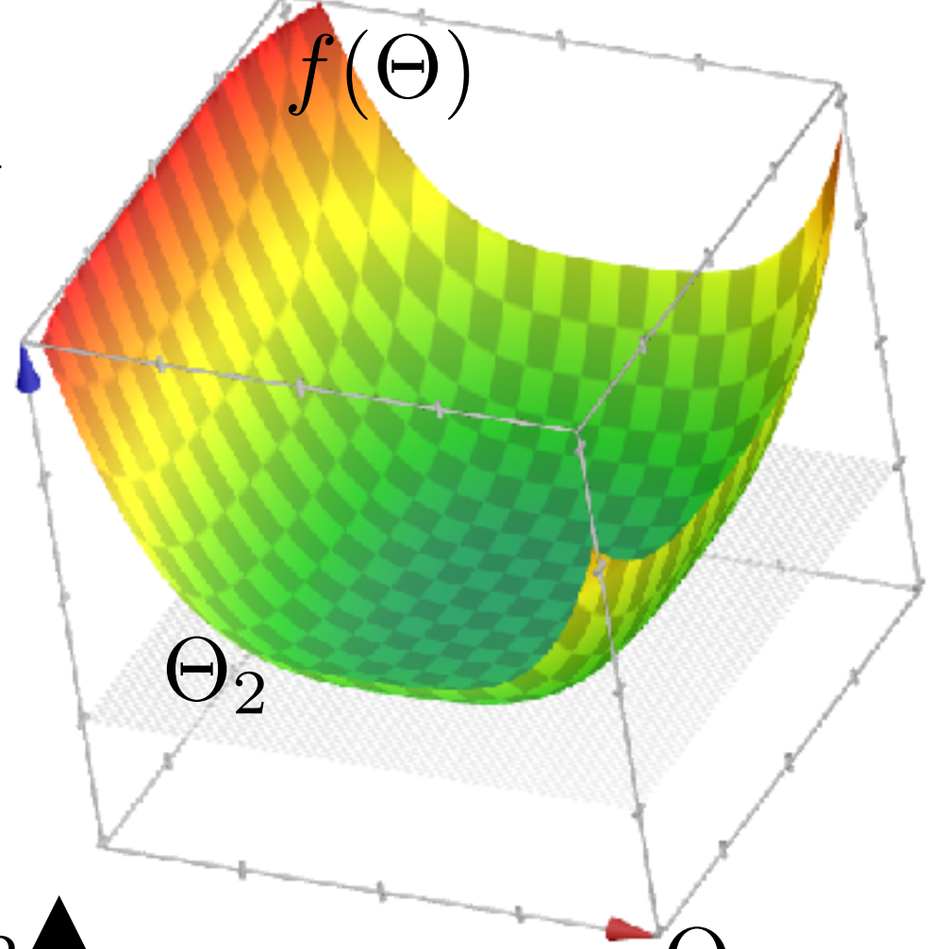
`t = t + 1`

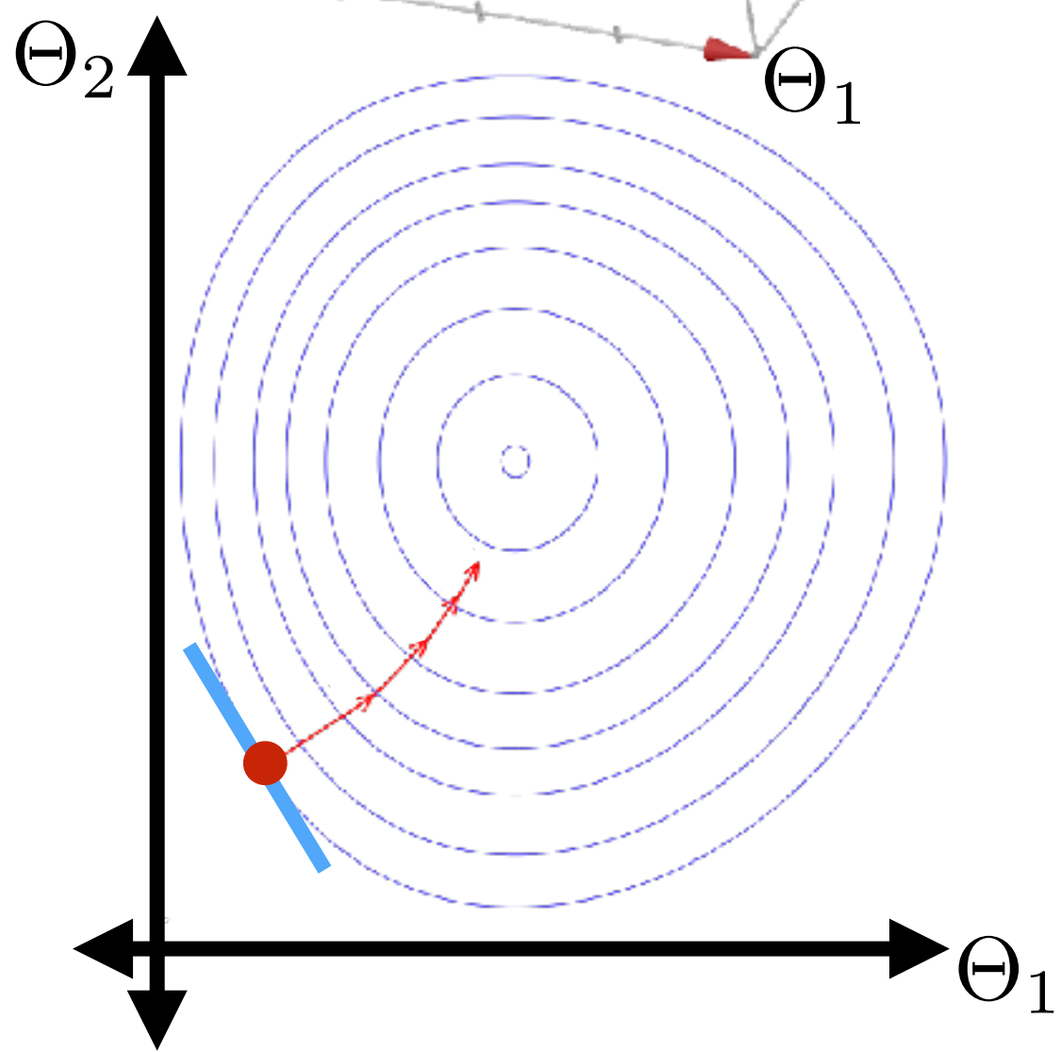
$\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$

`until` $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$

`Return` $\Theta^{(t)}$

- Other possible stopping criteria:
  - Max number of iterations *T*
  - $|\Theta^{(t)} - \Theta^{(t-1)}| < \epsilon$
  - $\|\nabla_\Theta f(\Theta^{(t)})\| < \epsilon$





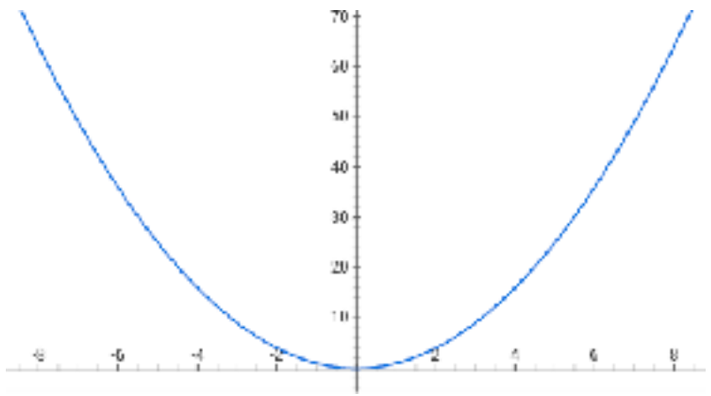
★ Gradient Descent (slide - 9 - 174)

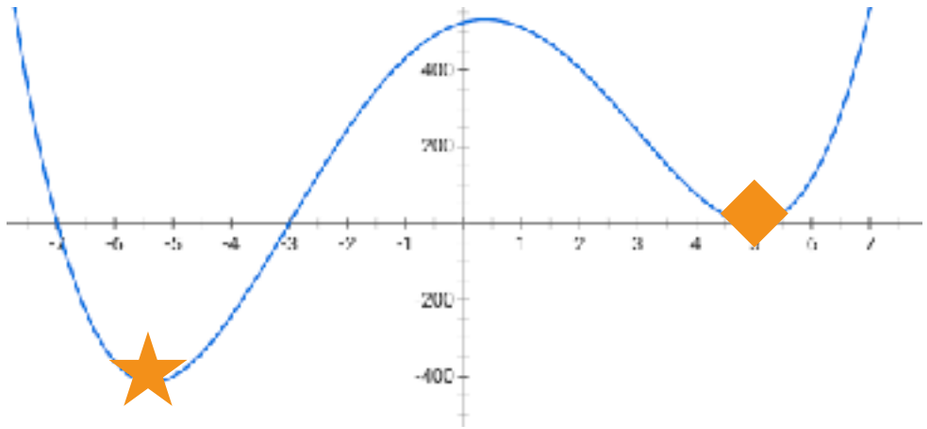
- Gradient Descent is basically an algorithm that locally minimizes ~~an~~ a general function. i.e. finds local minimum. Gradient Ascent would be maximizing the function.
- The gradient in this case is the rate of change of a function w.r.t. its parameters. It uses partial derivatives.
- For the algorithm, (which is an iterative method for finding solutions) we must define 'η' which is the step size by which the function should iterate. As always, lower the step size, more accurate the results & more need of computational power.
- Since it's an iterative method, we must also define 'ε' which is the change factor or tolerance between two updates.

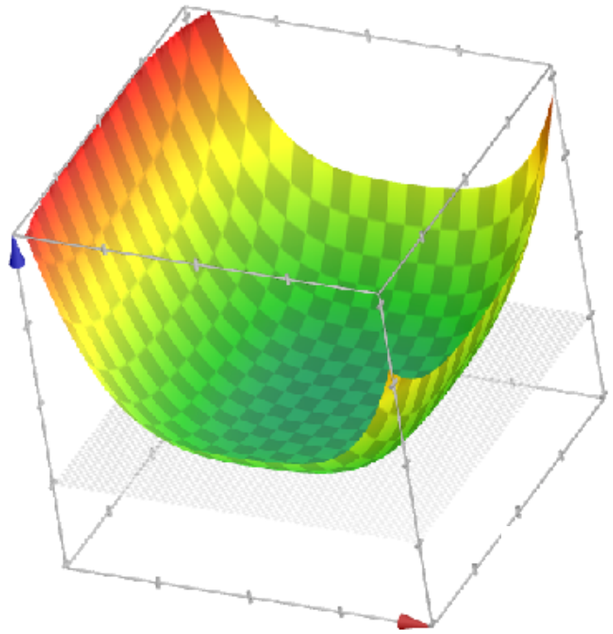
★ NOTE:- In this case, we are using this gradient descent on the loss function since its continuous, differentiable, we want to minimize this loss.

Gradient descent properties

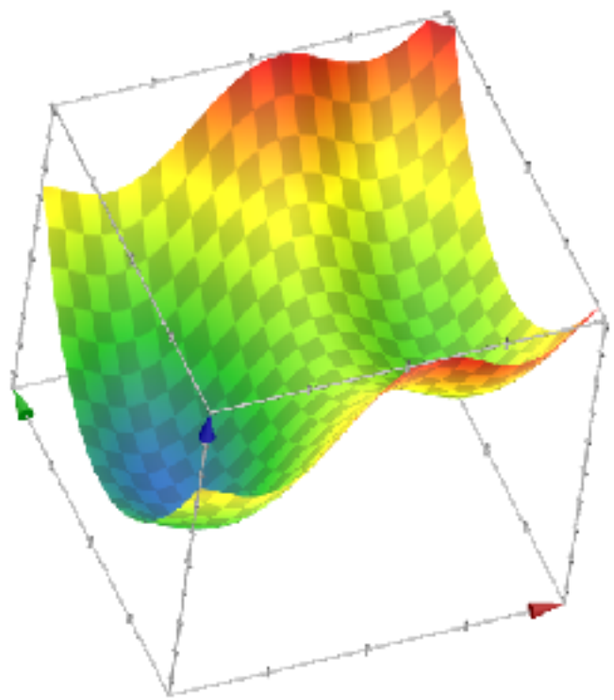
- A function *f* on $\mathbb{R}^m$ is convex if any line segment connecting two points of the graph of *f* lies above or on the graph







- **Theorem**: Gradient descent performance
  - **Assumptions**: (Choose any $\tilde{\epsilon} > 0$)
    - *f* is sufficiently "smooth" and convex
    - *f* has at least one global optimum
    - η is sufficiently small
  - **Conclusion**: If run long enough, gradient descent will return a value within $\tilde{\epsilon}$ of a global optimum Θ



* Gradient descent properties (Slide - 10 - 201)

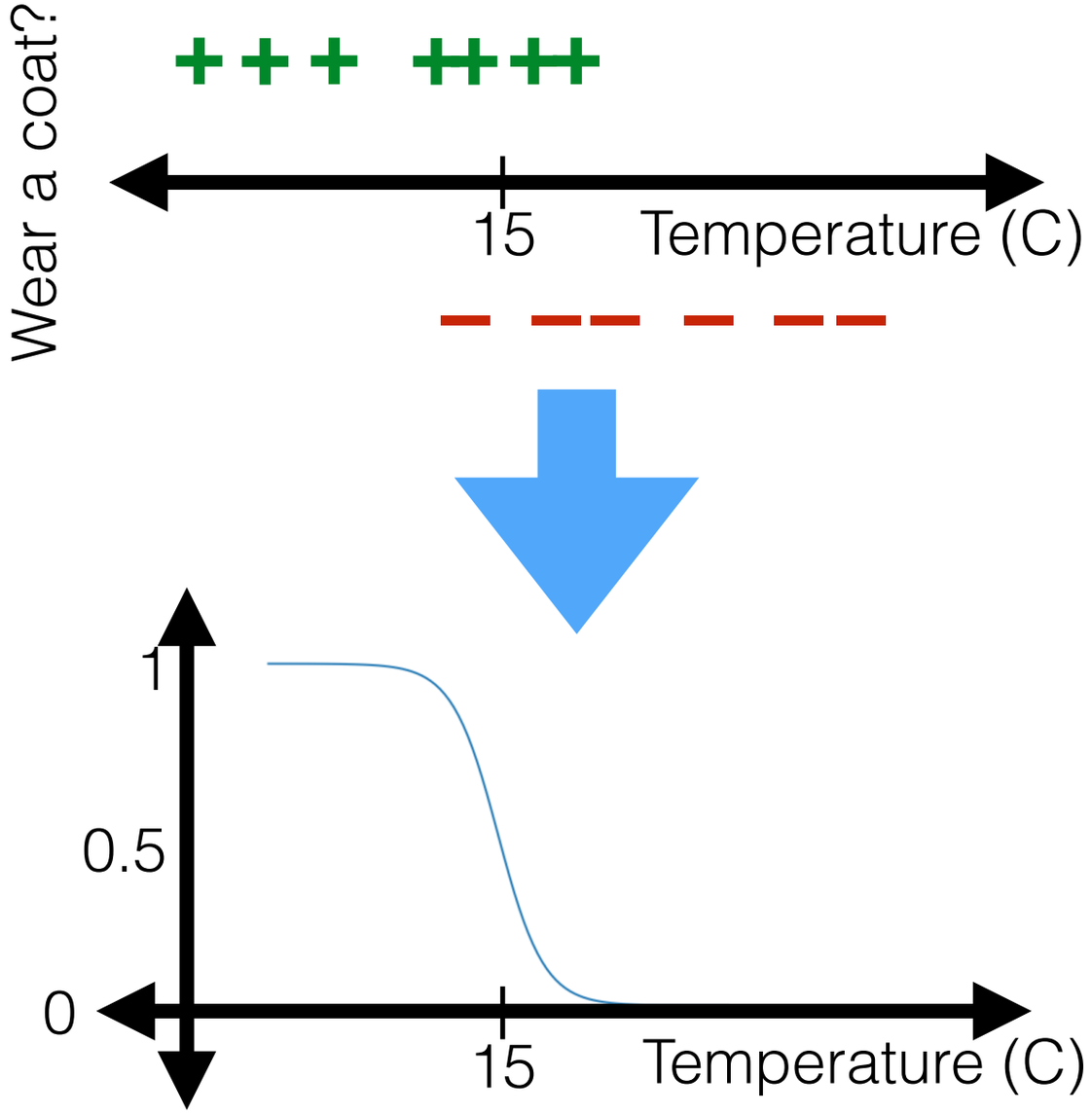
- In the assumptions, we assume that function 'f' is "smooth" which basically means continuous & differentiable till the local minima.
- Since we have also assumed that 'f' is also convex, ~~there will be only 1 local minima~~ the local minima is also the global minima.

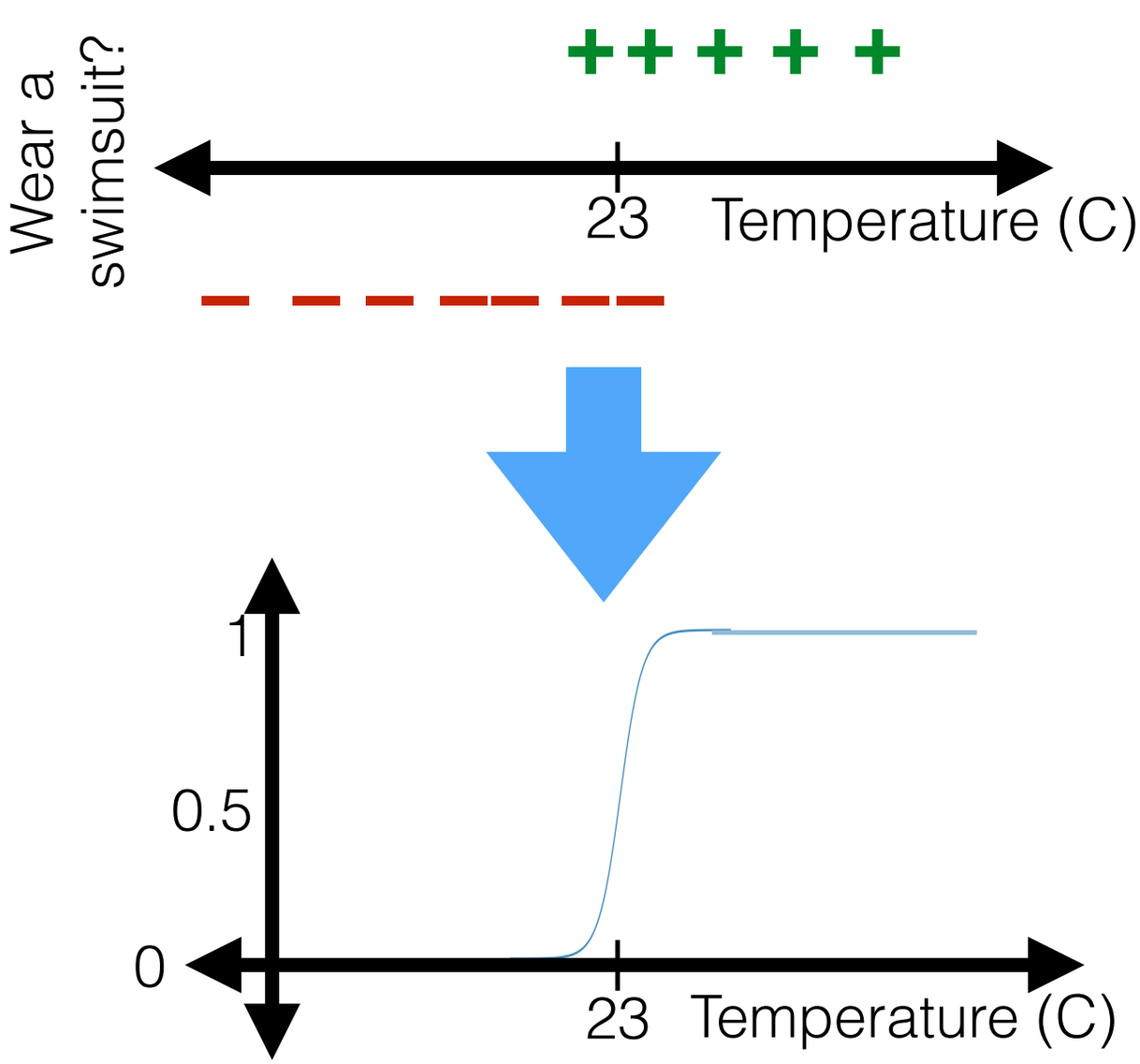
* Gradient descent for logistic regression (Slide-10-209) (Example)

- Notice that in the second example, the sigmoid function is steeper than the 1st one. That implies that θ for 2nd example will be greater than θ for 1st example.
- Keep that in mind for next slide example

Gradient descent for logistic regression

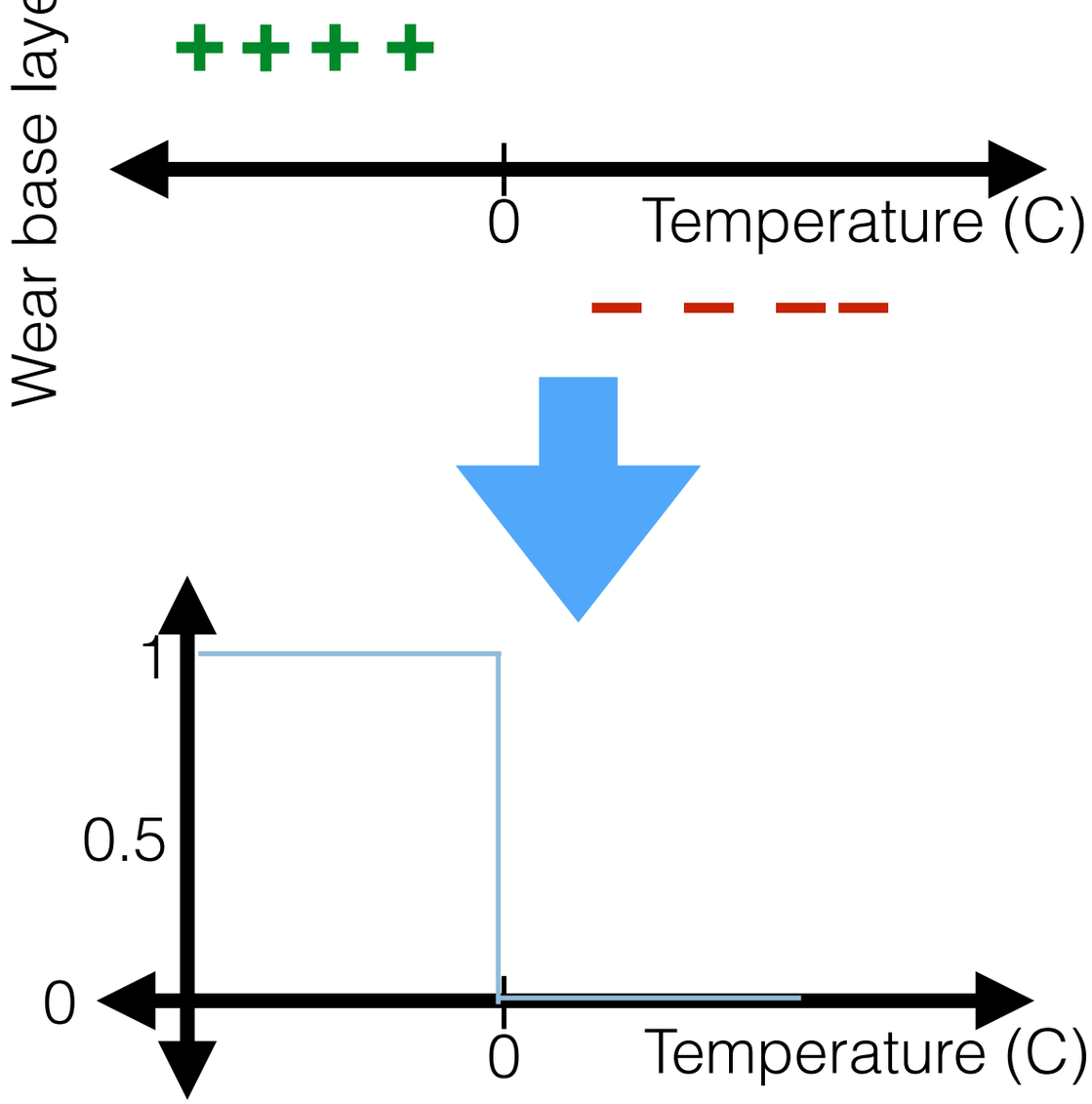
- Loss $J_{\text{lr}}(\Theta) = J_{\text{lr}}(\theta, \theta_0)$ is differentiable & convex
- Run `Gradient-Descent` ($\Theta_{\text{init}}, \eta, J_{lr}, \nabla_\Theta J_{lr}, \epsilon$)

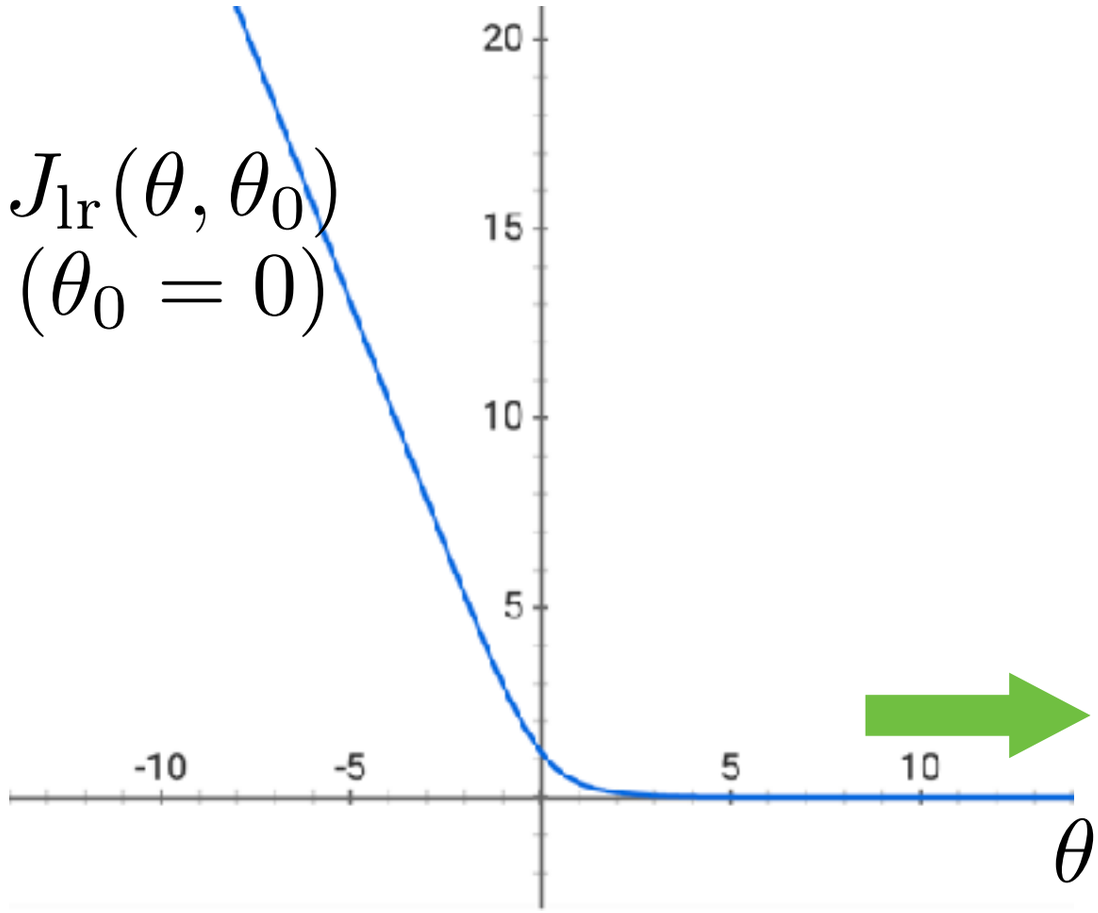




Gradient descent for logistic regression

- Loss $J_{\text{lr}}(\Theta) = J_{\text{lr}}(\theta, \theta_0)$ is differentiable & convex
- Run `Gradient-Descent(` $\Theta_{\text{init}}, \eta, J_{lr}, \nabla_\Theta J_{lr}, \epsilon$)





Gradient descent for logistic regression (Slide-12-)
(Data with gap example)

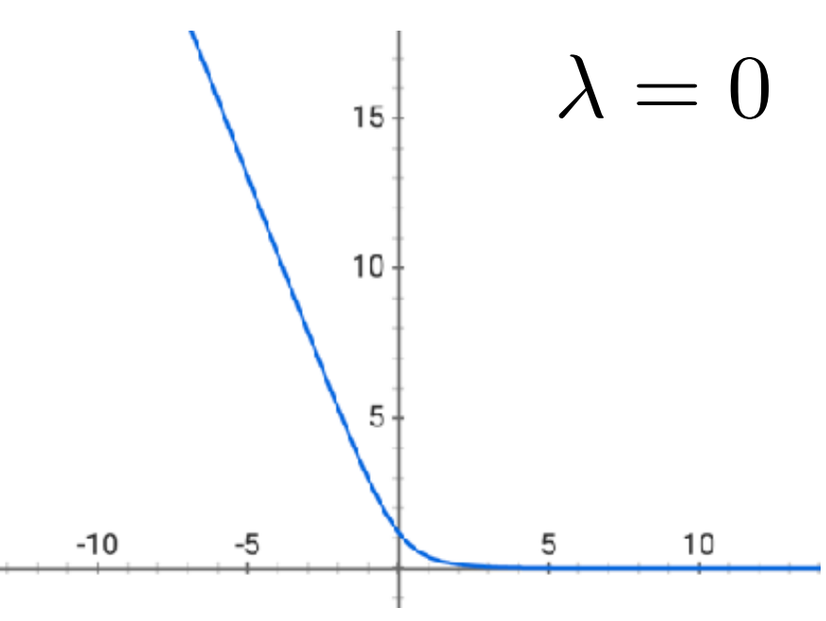
- Notice that in the original loss function, we just want to maximize the probabilities & there is no rule or constraint on how much should the parameters ($\theta_1 \& \theta_0$) should be tweaked for that to happen.
- As a result, when grdient descent is ran on such a function with gap in the data, the algorithm will keep on increasing θ till the steep slope is almost straight. i.e. $\theta \to \infty$

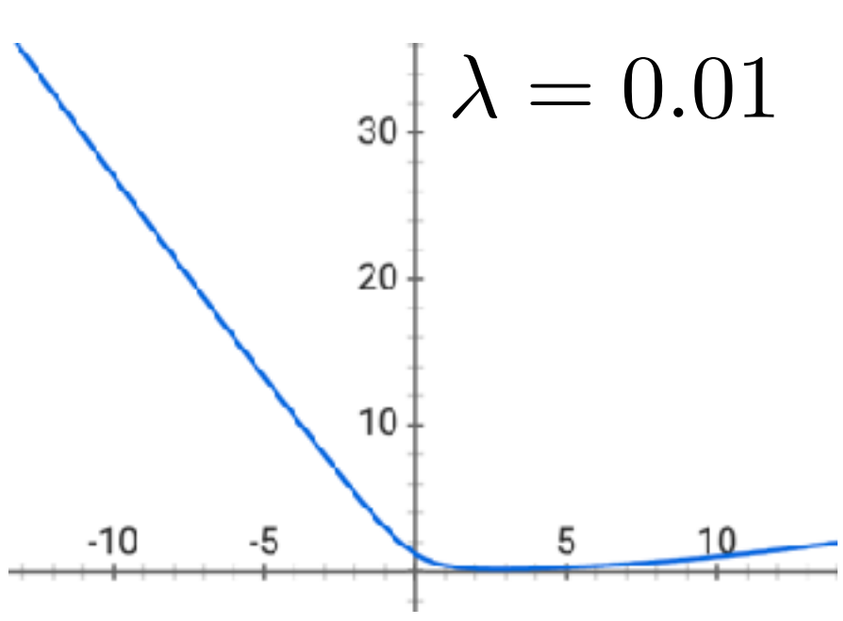
- But as you can imagine, this is unrealistic. If you see the loss function, there is no minima. The loss approaches 0 as θ increases. i.e. Loss $\to 0$ for $\theta \to \infty$
- This can be solved/improved by using a 'regularizer' that we include in our loss function.

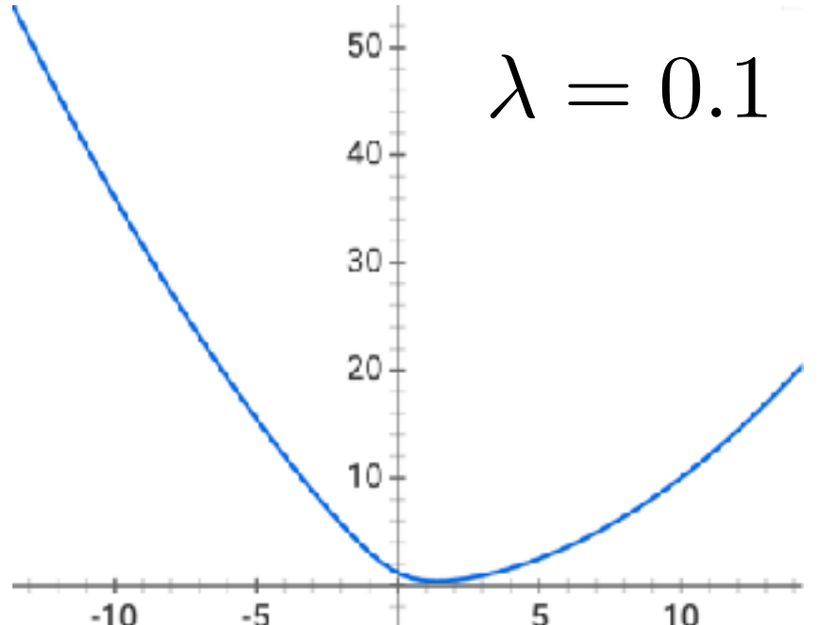
Logistic regression loss revisited

$$J_{\text{lr}}(\Theta) = J_{\text{lr}}(\theta, \theta_0)$$
$$= \frac{1}{n} \sum_{i=1}^{n} L_{\text{nll}}(\sigma(\theta^\top x^{(i)} + \theta_0), y^{(i)}) + \lambda \|\theta\|^2 \quad (\lambda \geq 0)$$

- A "regularizer" or "penalty" $R(\theta) = \lambda \|\theta\|^2$
- Penalizes being overly certain
- Objective is still differentiable & convex (gradient descent)







- How to choose hyperparameters? One option: consider a handful of possible values and compare via CV

* Logistic regression loss revisited (Regularizer or penalty) (Slide - 13 -)

- A regularizer is just a component that we add to the loss to express some information or insert a information that we can't directly incorporate into the loss function.
- In this example, the regularizer is used to express the loss that large θ will have on the function because very large θ will be illogical.
- In this case, this regularizer used is an L-2 regularizer. You can just imagine that we had been using the loss function with $\lambda = 0$ for the regularizer.
- You can also imagine the regularizer as just penalty for the loss function for very large θ.

- We have decreed here that very large θ is bad. The reason is that as we increase the θ, the slope of the sigmoid gets steeper & more horizontally compact. That means the area or the part where there is uncertainty i.e. ~~prob $\neq$ {1}~~ prob $\notin \{1, 0\}$ is narrowing. Which would mean we are very certain i.e. probabilities are going extreme.
- Just like overfitting, we can be overly certain, overconfident even if there is a gap in the original data.

- As you notice, as we induce a non-zero lambda, we start to get global minima.
- But as with all hyperparameters, you need to carefully & systematically choose value for λ. Too ~~much~~ big λ would overshadow the loss & too small λ will result to overcertainty (very large θ)
- One way to choose the hyperparameter value would be to consider a few possible / acceptable values & ~~comp~~ compare via cross-validation.

Logistic regression learning algorithm

LR-Gradient-Descent($\theta_{\text{init}}, \theta_{0,\text{init}}, \eta, \epsilon$)

Initialize $\theta^{(0)} = \theta_{\text{init}}$

Initialize $\theta_0^{(0)} = \theta_{0,\text{init}}$

Initialize t = 0

repeat

t = t + 1

$$\theta^{(t)} = \theta^{(t-1)} - \eta\left\{ \frac{1}{n} \sum_{i=1}^{n} \left[\sigma(\theta^{(t-1)\top} x^{(i)} + \theta_0^{(t-1)}) - y^{(i)}\right] x^{(i)} + 2\lambda\theta^{(t-1)} \right\}$$

$$\theta_0^{(t)} = \theta_0^{(t-1)} - \eta\left\{ \frac{1}{n} \sum_{i=1}^{n} \left[\sigma(\theta^{(t-1)\top} x^{(i)} + \theta_0^{(t-1)}) - y^{(i)}\right] \right\}$$

until $|J_{\text{lr}}(\theta^{(t)}, \theta_0^{(t)}) - J_{\text{lr}}(\theta^{(t-1)}, \theta_0^{(t-1)})| < \epsilon$

Return $\theta^{(t)}, \theta_0^{(t)}$

Exactly gradient descent with *f* given by logistic regression objective