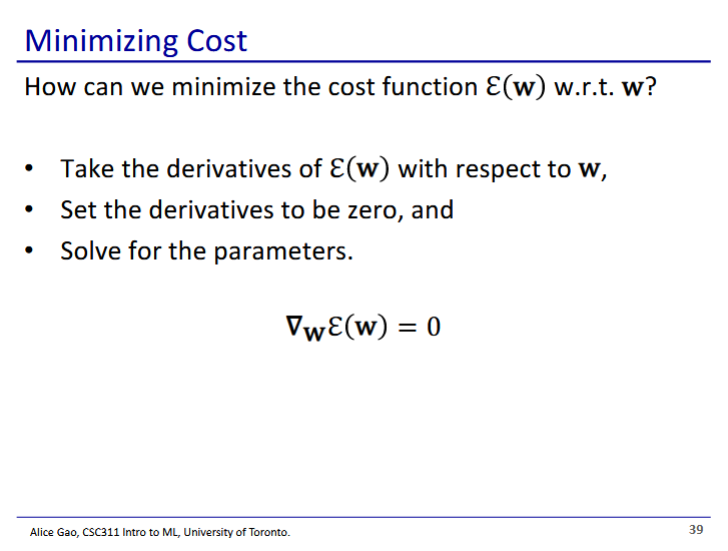
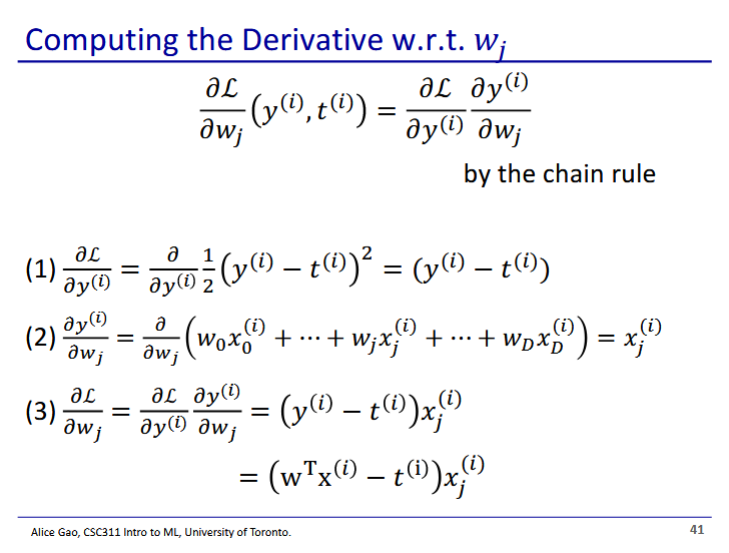
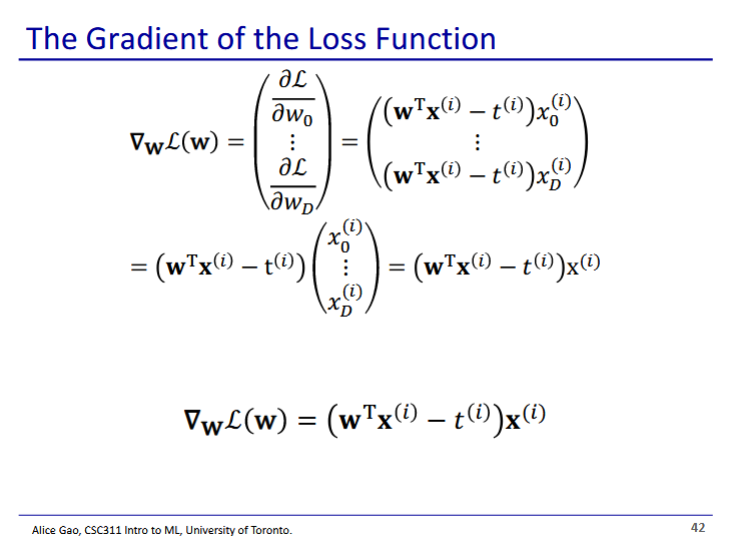
| **No lab this friday**  **Linear regression with multiple features cont’d**   * We sum together the gradient of the loss function to get the gradient of the cost function   + Loss function gradient:     - This is a vector of dimension D ( is a scalar)   + Cost function gradient:     - Non-vectorised:     - Vectorised: * **Computing optimal weights**   + Generalised equation:     - Derivation on slide 48   + Solving this is expensive (matrix inversion is   + Generalised equation **only** works for linear regression and mean squared error loss function     - Other combinations do not have such simple generalised equations   **Gradient descent**   * Gradient descent is a generalised method for optimising a function   + Easier to implement than direct solution - works for more than just linear regression with MSE loss function   + Iterative method with for each update     - Still faster than of generalised method for linear regression with MSE loss function   + We can plug in the cost function into the general update rule to get the rule to apply     - **This is a common test question, see slide 14** * **Process:**   + Start with a random point   + Repeatedly update the point using the update rule until stopping condition     - Update rule:     - Alternatively:       * Update each each in the weight vector     - Point is updated in the opposite sign as the gradient     - Update size is proportional to the gradient magnitude   + We stop when:     - w stops changing (theoretical)     - When the change in is small enough     - When we are tired of waiting and want a result * **Hyperparameter: learning rate**    + Learning rate scales the update size   + Could choose using a validation set, but can also choose based on what we want out of it   + Too small means function takes too long to converge   + Too large means the function may diverge (flip flops around the minimum)     - In this case you will see the value of the cost function increasing |
| --- |



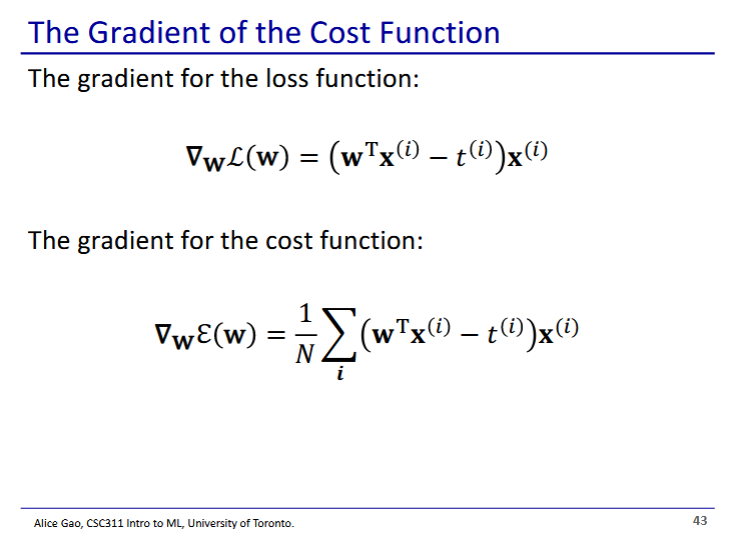
* We want the derivative to be 0 when derived by each weight

- 

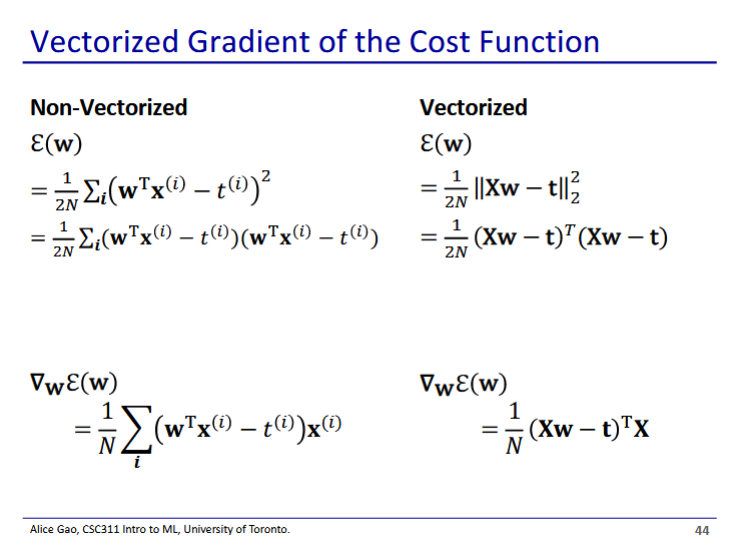
* Like with linear regression with 1 feature, we use chain rule to derive the loss function



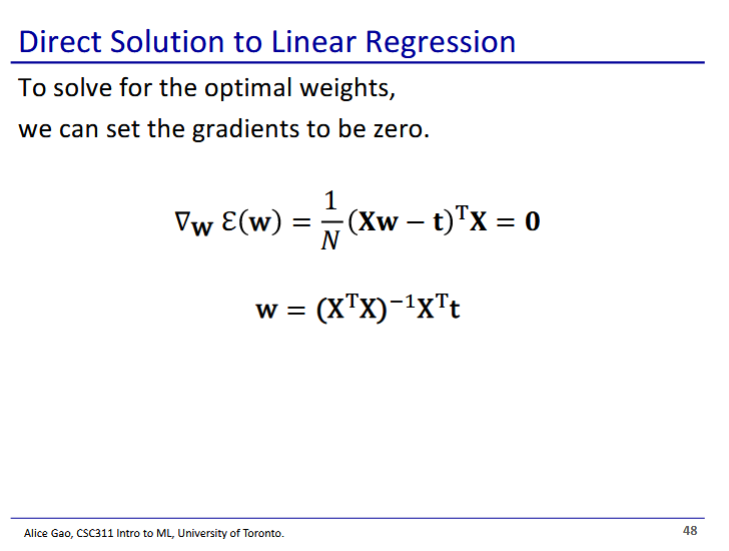
* We can then combine the derivatives of the loss function for each feature into a vector
* The is a scalar, so we can factor it out in the second line



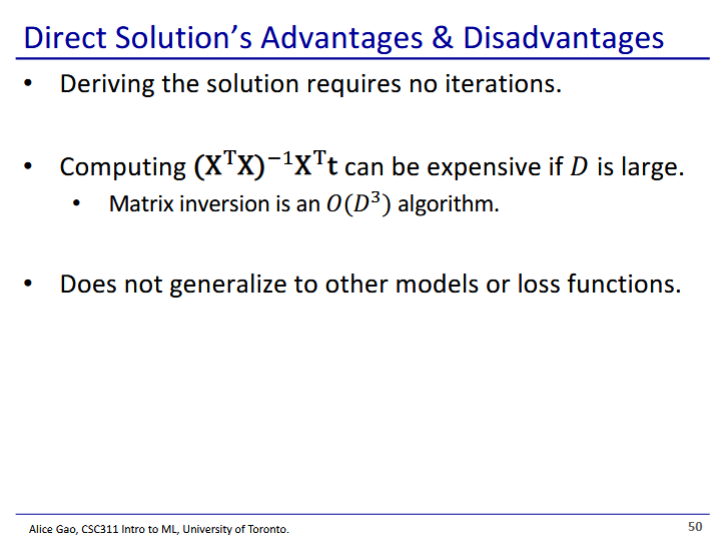
* Then to get the gradient of the cost function, we add together the gradient of the loss function for every training data point



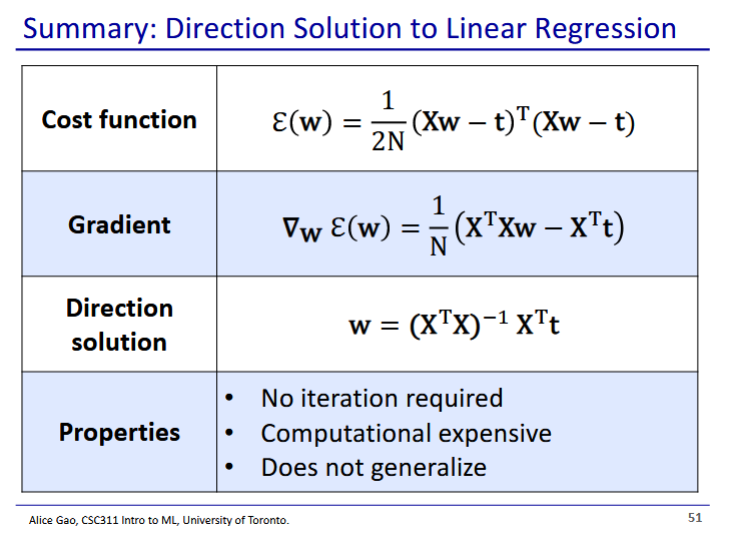
* Demonstration that the bottom left and bottom right are equivalent
* is a vector of size D
* is also a vector of size D
* The path we took in the previous slides was to find the non-vectorised gradient, then convert it to the vectorised version
  + There are other ways to derive it

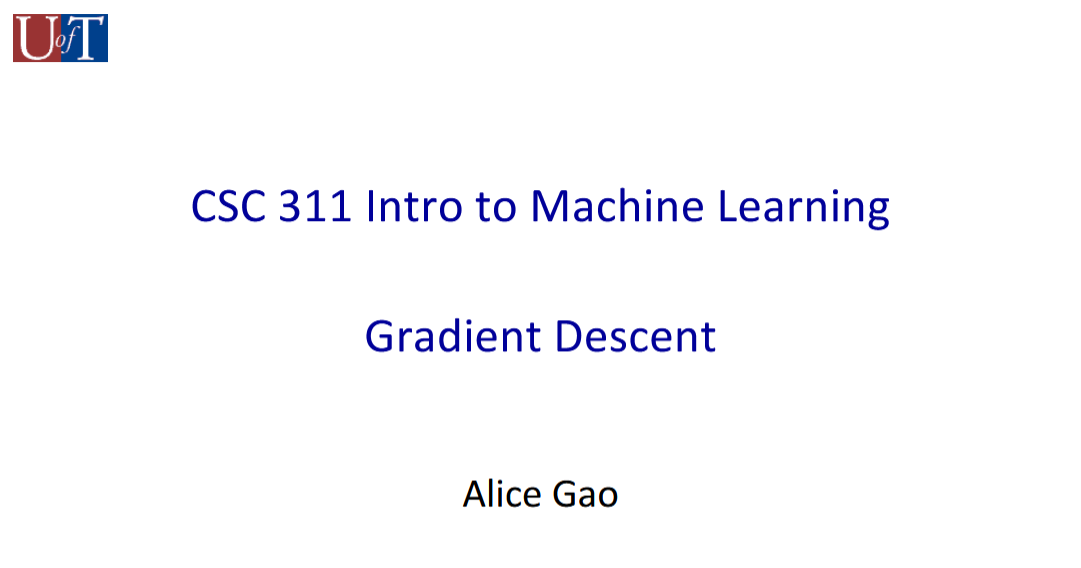


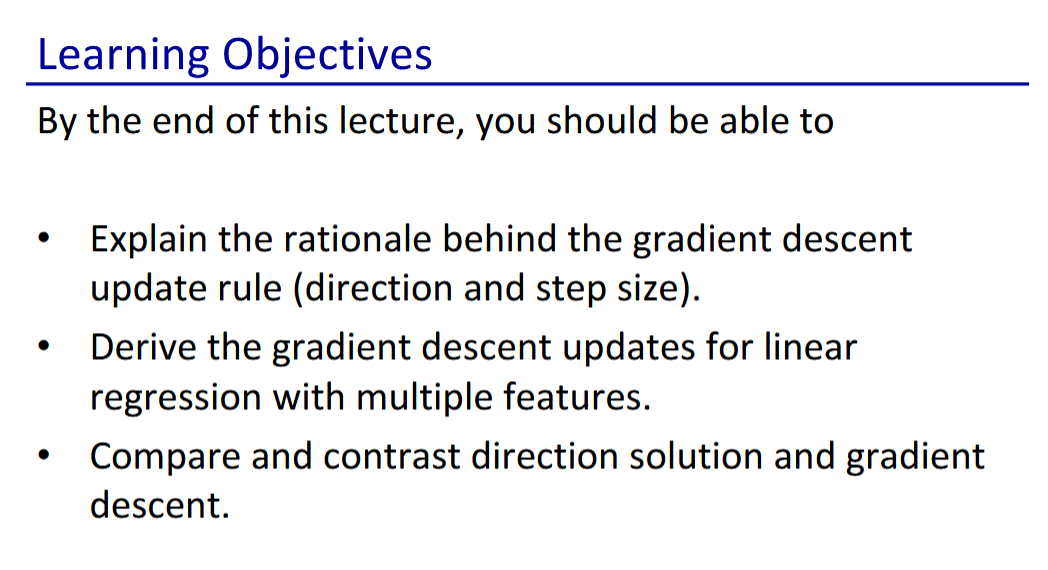
* Derivation:

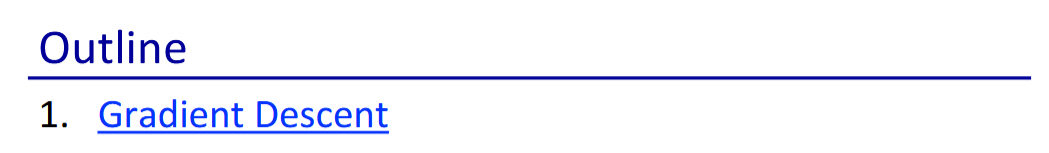


* Deriving this solution is difficult since matrix inversion () is in
* Other models or loss functions do not generalise this easily
* The solution is **gradient descent**

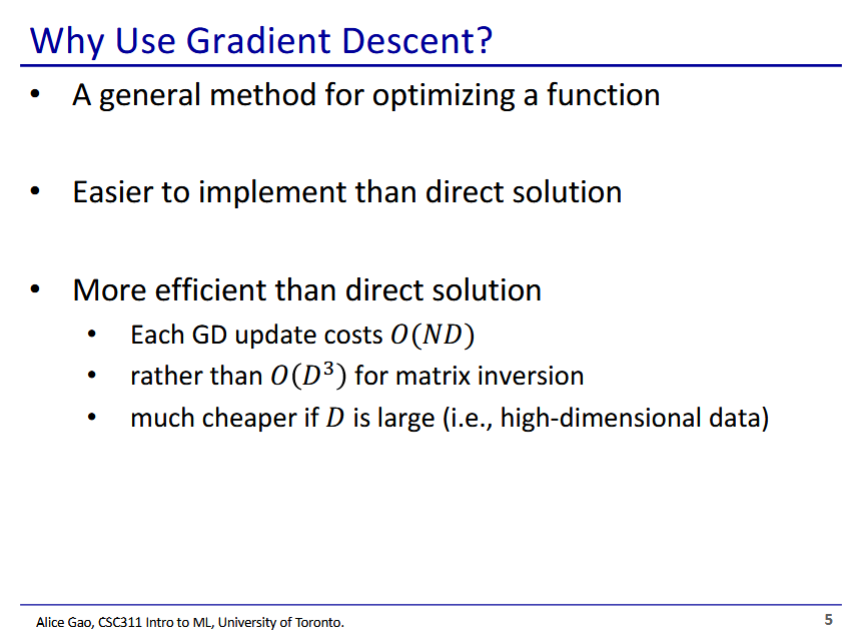




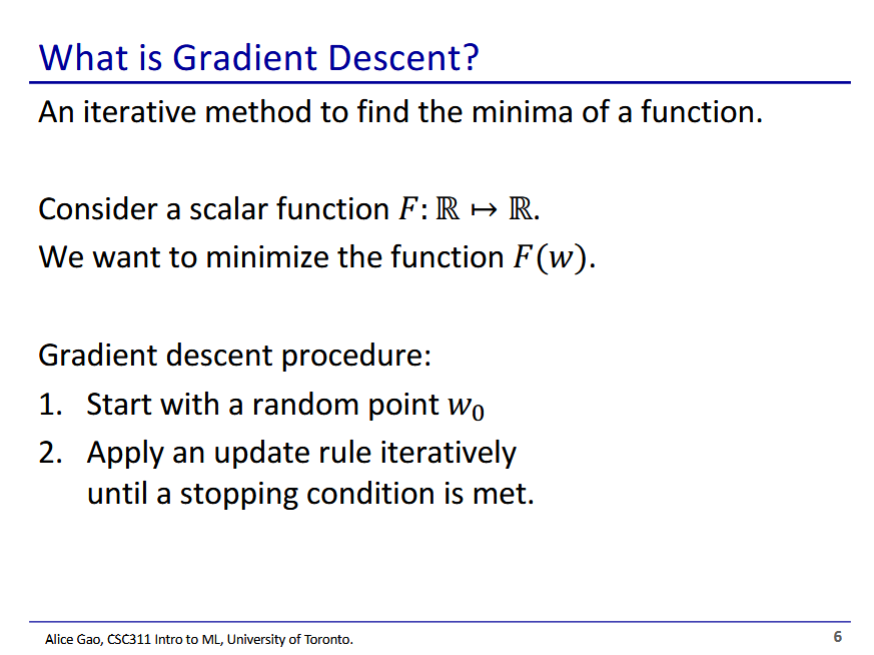




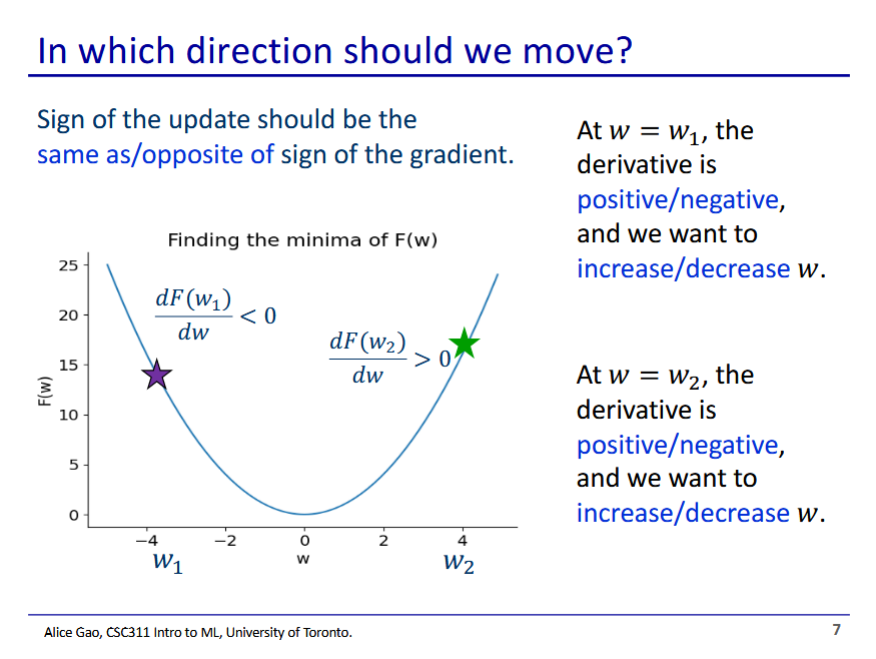




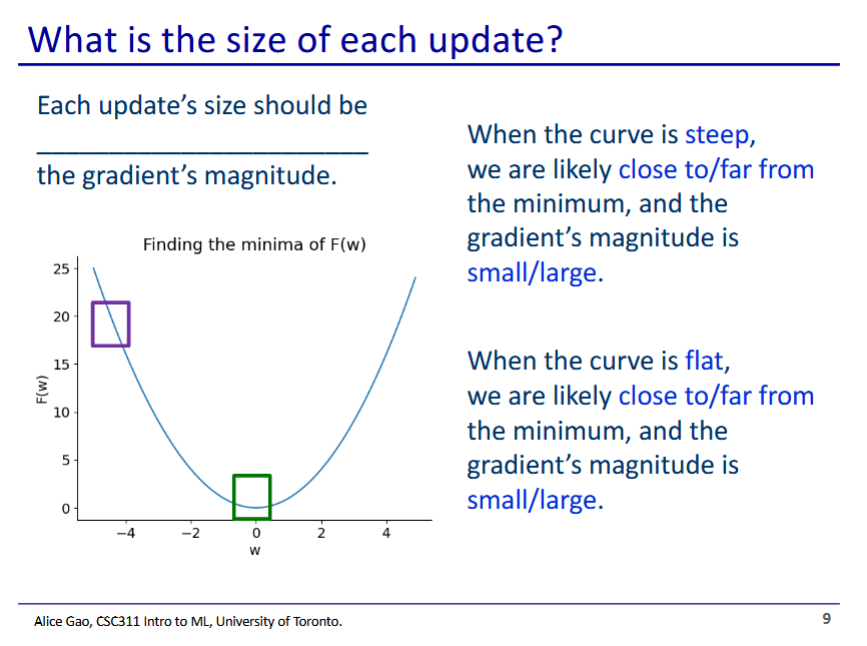
* Advantages of gradient descent
  + General optimisation algorithm for different models and loss functions
  + Simpler to implement and faster than direct solutions



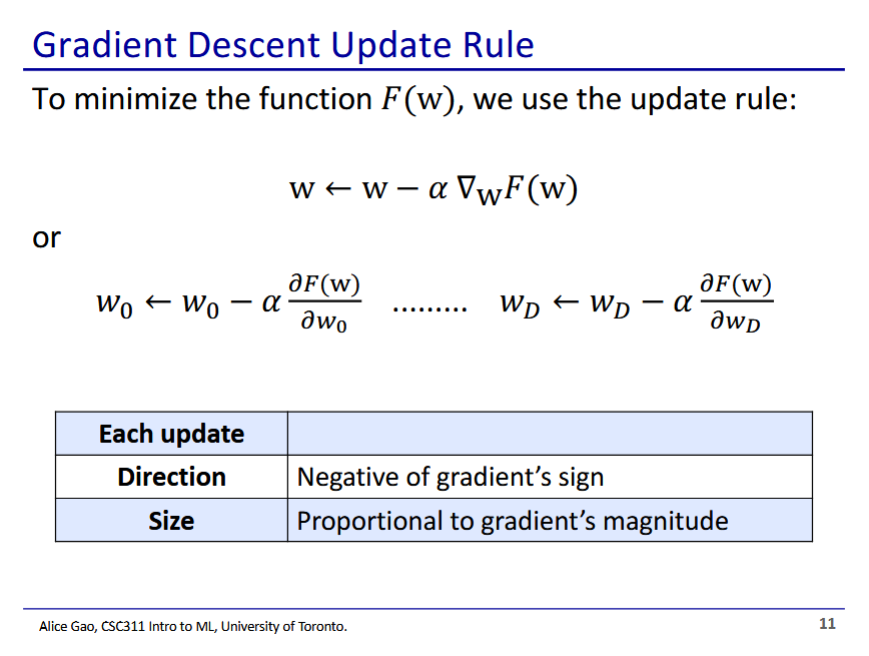
* Gradient descent is an iterative function
* Simplest case: scalar function
  + We start at a random weight
  + We update the weight according to a rule repeatedly until we meet our stopping condition



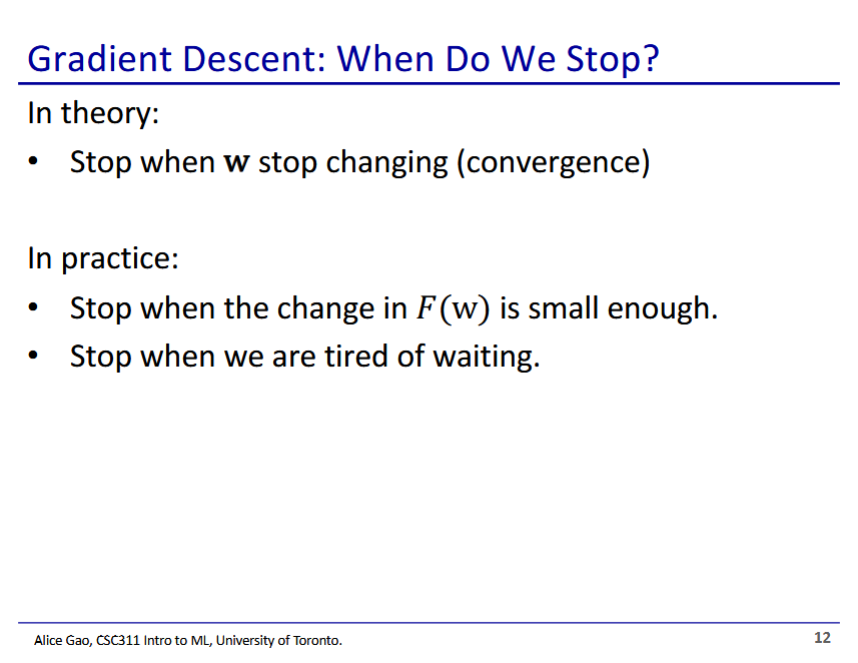
* Sign of the update should be **opposite** the sign of the gradient
* At the derivative is **negative**, and we want to **increase** w
* At the derivative is **positive** and we want to **decrease** w



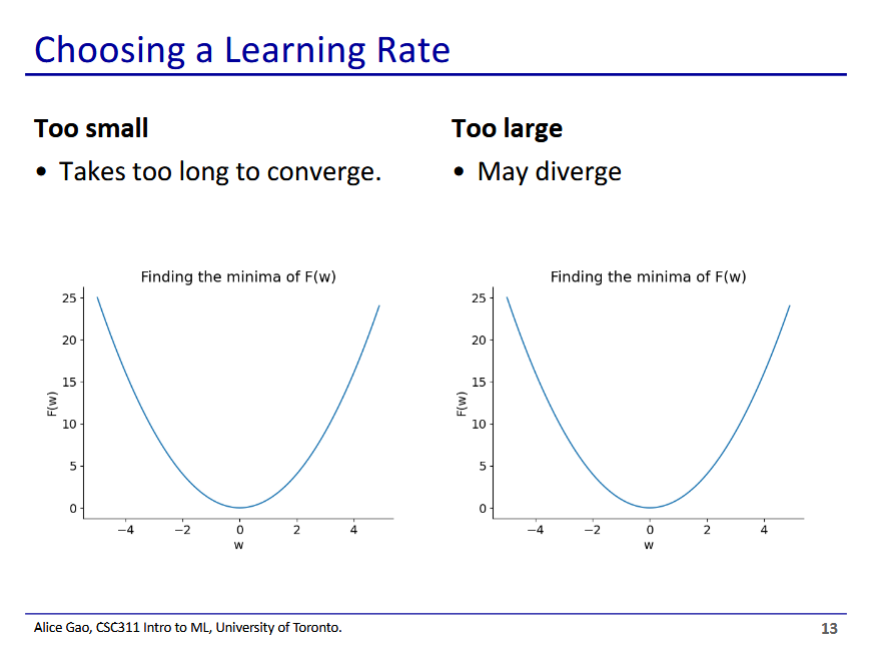
* The update’s size should be **proportional** to the gradient’s magnitude
* When the curve is steep, we are likely to be **far from** the minimum and the gradient’s magnitude is **large**
* When the curve is flat, we are likely **close to** the minimum and the gradients magnitude is **small**

****

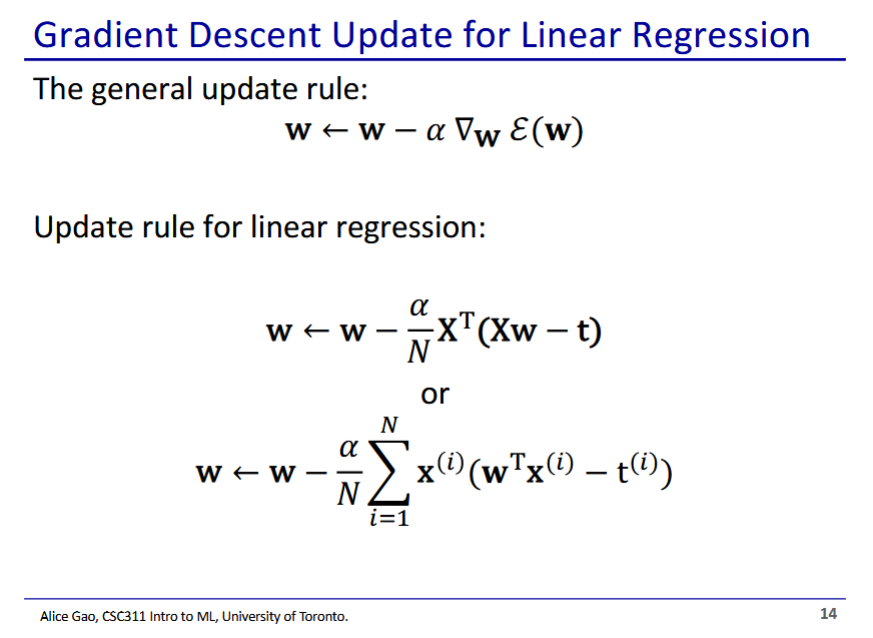
* We update the weight vector with
  + Alpha is the learning rate



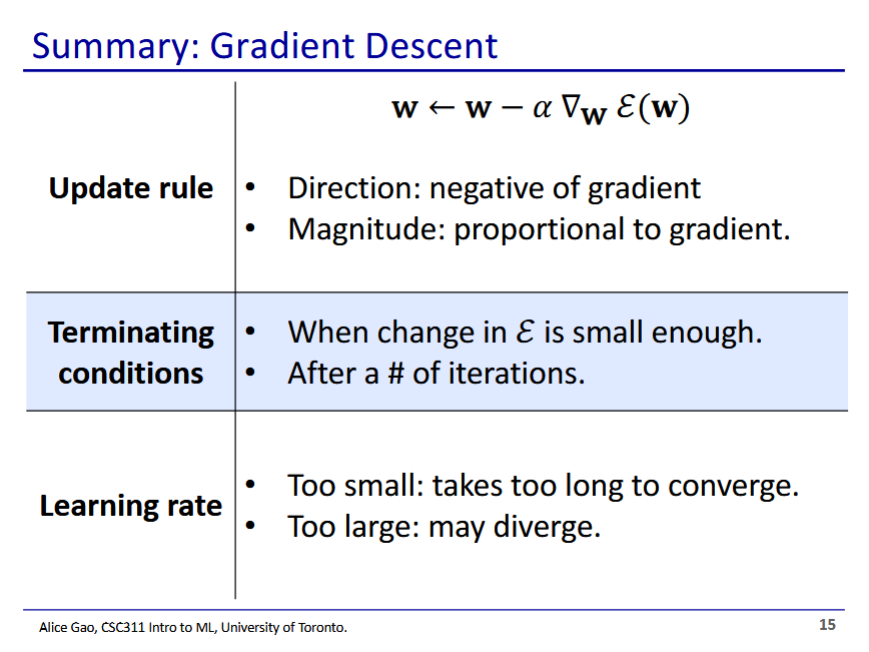
* In theory our descent should converge if we picked the right learning rate
* We stop when the change is small enough, or if we are tired of waiting (resource constraints)

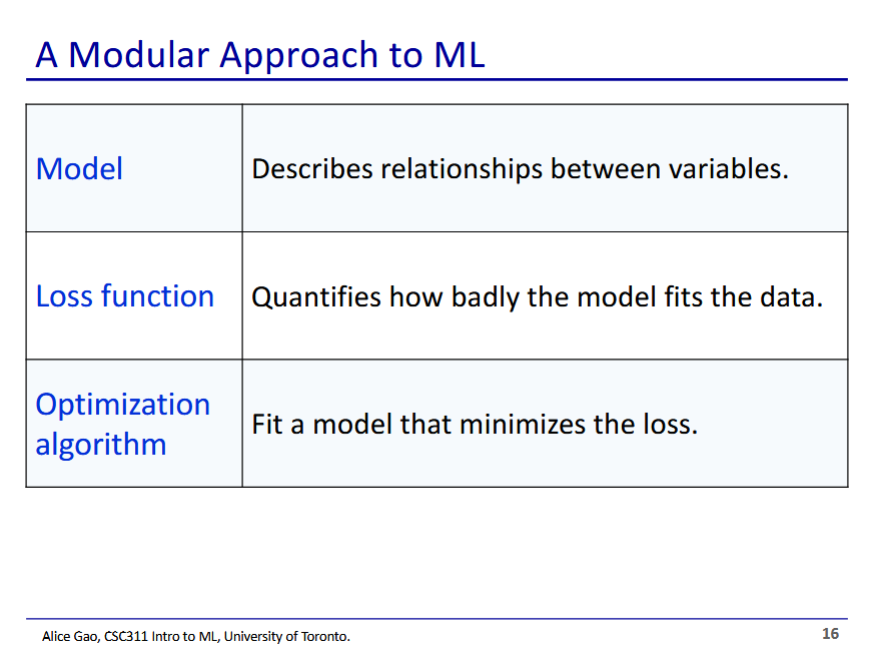


* Learning rate is an important part of the model, it is a hyperparameter
* We could choose the learning rate using a validation set
* But we can also choose it based on the properties of the learning rate
  + Too small means function takes too long to converge
  + Too large means the function may diverge (flip flops around the minimum)
    - In this case you will see the value of the cost function increasing



* To get the update rule for linear regression, we plug in the vectorised or non-vectorised cost function into general update rule
* Popular question on tests: take a model, derive its cost and plug into the general update rule





* We make 1 choice for each component and put them together to get an ML algorithm
* Machine learning is taking a learning problem and turning it into an optimisation problem