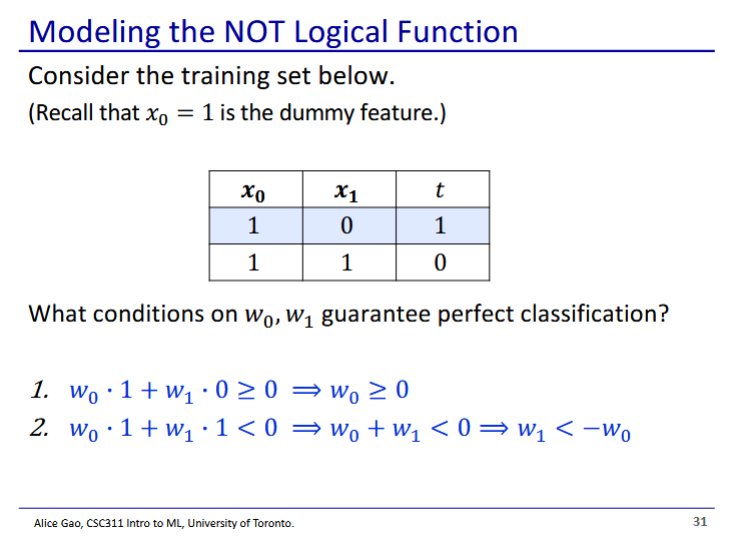
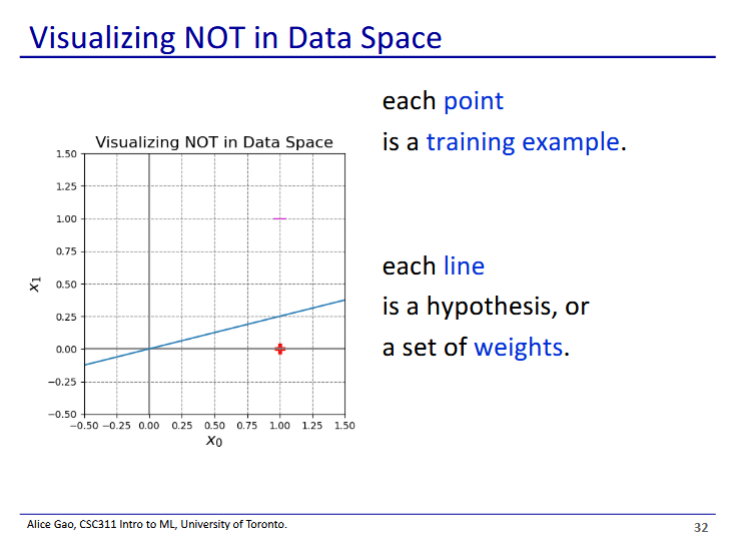
|  |
| --- |
| **Test stuff**   * Make a copy of your aid sheet - aid sheets will be collected after the test * ⅓ on each topic   + ⅓ KNN, ⅓ decision trees, ⅓ linear regression   + Linear regression will likely be hardest since we have not done it as much * Practice test available on piazza   **Binary linear classification recap:**   * **Visualisation in data space and weight space**   + Data space     - Each axis is a feature     - Each point on the graph is a data point     - Each line on the graph is a hypothesis       * Corresponds to a set of weights   + Weight space     - Each axis is a weight     - Each point on the graph is a set of weights (hypothesis)     - Each line on the graph is a training data point       * Lines put constraints on possible hypotheses   + See slides 32 and 33 of recap * We can only calculate a valid hypothesis if our data is linearly separable (we can draw a line that separates both groups perfectly)   + For non-linearly separable data this problem becomes more complicated as we would need to rethink our loss and activation functions   **Attempts to use binary linear classification on non-linearly separable data**   * **Attempt 1:** Linear mode, Threshold activation function, 0-1 loss function   + Loss function awards a 0 on a wrong output, and 1 on a correct output   + Problem: gradient is always 0 or undefined, impossible to do gradient descent   + See slides 8-13 * **Attempt 2:** Linear model, Threshold activation function, Squared loss function   + Loss function sets a target value for the linear model, and penalises the difference from target   + Problem: loss function penalises even when the model gives the correct final output     - Issue arises as possible values of z are unbounded   + See slides 14-16 * **Attempt 3:** Linear model, Logistic activation function, Squared loss function   + Logistic activation function: takes in z and returns a value along a sigmoid curve between 0 and 1, sort of like a curvier stepwise function   + We can now use our loss function to compare the output of the sigmoid curve against the target output   + Problem: the gradients for the loss function are near 0 at the extreme ends of the graph, which means gradient descent will not try to optimise in some extreme cases   + See slides 19-21   **Logistic regression**   * Components: linear model, logistic activation function, cross-entropy loss function * **Cross-entropy loss function**   + - Has 2 parts to calculate loss: first part disappears when t=0, second part disappears when t=1   + Avoids the problem that the squared loss function has   **Stochastic gradient descent and mini-batch gradient descent**   * Method for reducing the computational cost of calculating the gradient of the cost function   + **Stochastic** - we randomly choose a point each update and use its loss function gradient   + **Mini**-**batch** - we sum up the loss functions of a sample of points each update instead of the entire training set     - Typically we will break up the training data into batches and iterate sequentially through the batches     - Each pass over the entire training set is an epoch * **Advantages:**    + Much faster than full gradient descent, update runtime not dependent on N   + Useful if data arrives over time   + Will eventually get you to the same result * **Disadvantages:**    + Updates have more variance     - Variance increases with smaller batch size   + Operations cannot be vectorised (small efficiency loss) |

Linear regression recap:

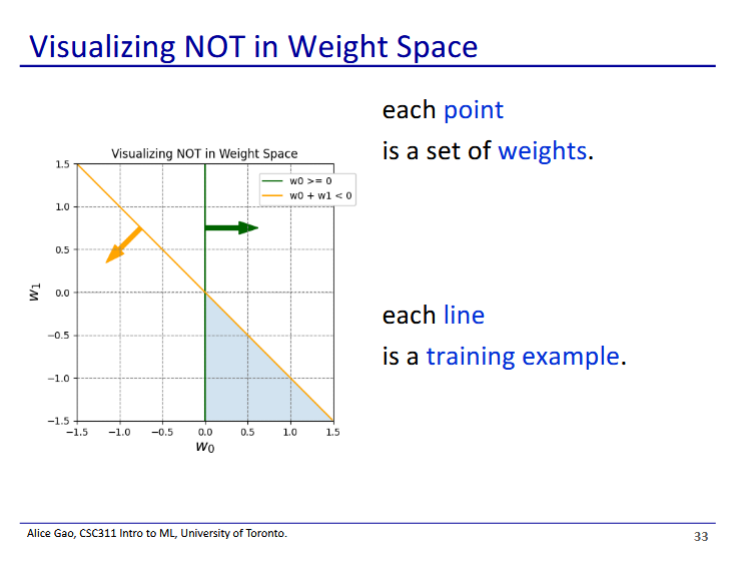
* Feature mapping
  + Using linear regression to learn non-linear functions
  + We remap the features such that the remapped features are nonlinear
* Regularizer
  + A way to prevent overfitting by picking simpler hypotheses
  + An alternative to tuning a hyperparameter
* Binary linear classification
  + Using linear regression to classify inputs into a binary category
  + We set a threshold value to decide the label



* Any set of weights that satisfy these 2 conditions will give perfect classification
* This model is best for separating data that is already linearly separable



* Note: x0 is a dummy feature, so its value is always 1
* Data space - each point corresponds to a training data point



Weight space - each point

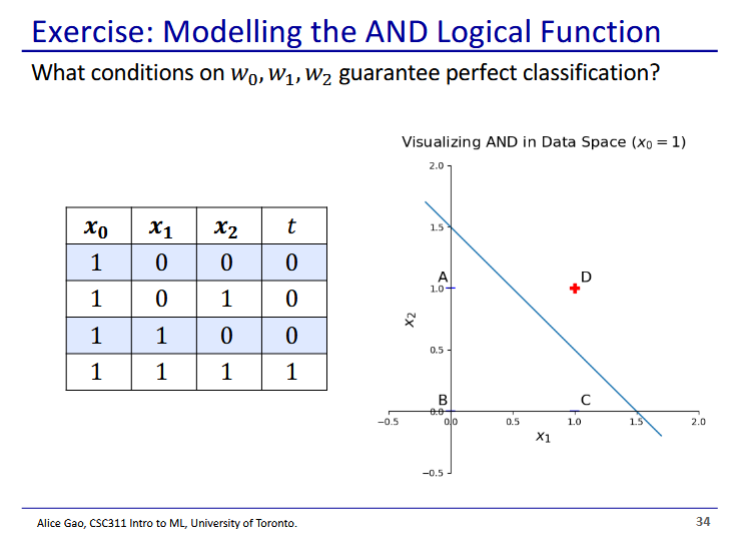
corresponds to a set of weights.

Each line, a training example.

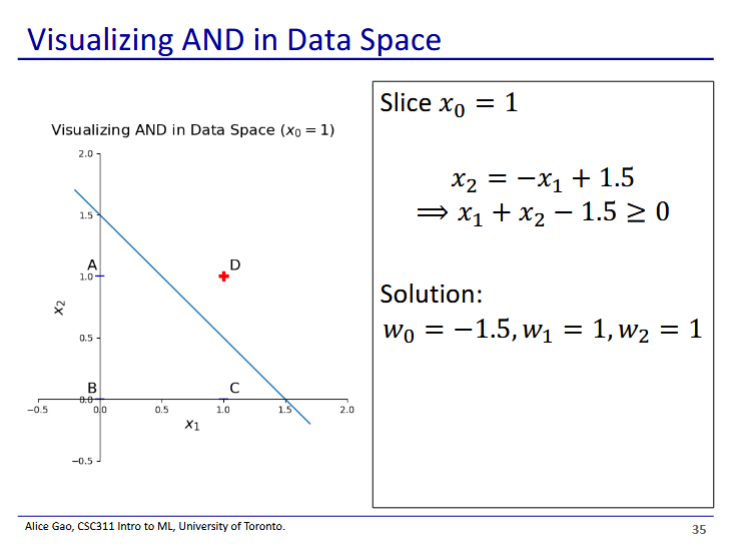
Points that fall into the shaded area

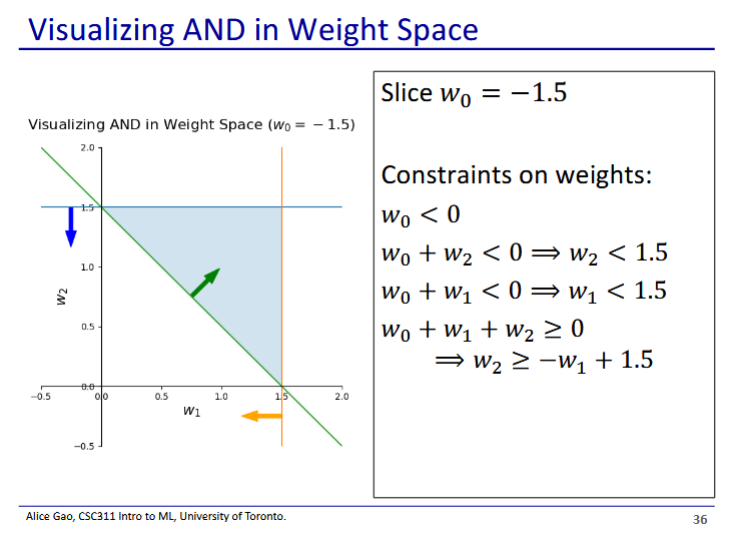
are sets of weights that work for the

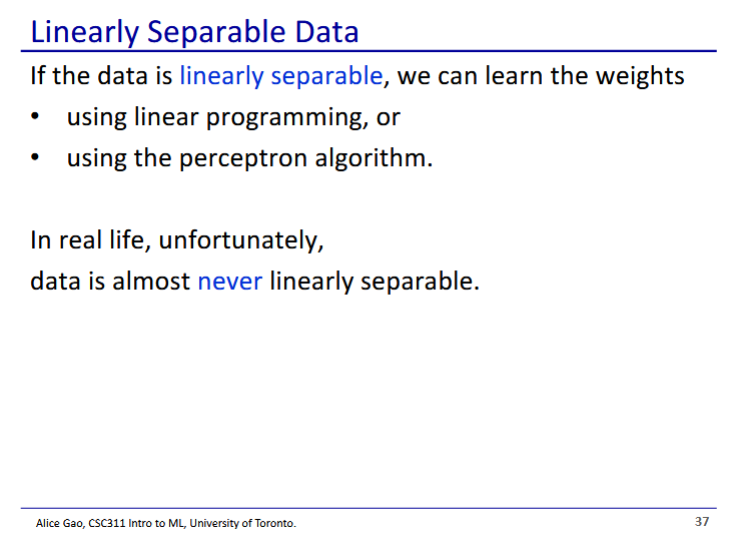
classifier



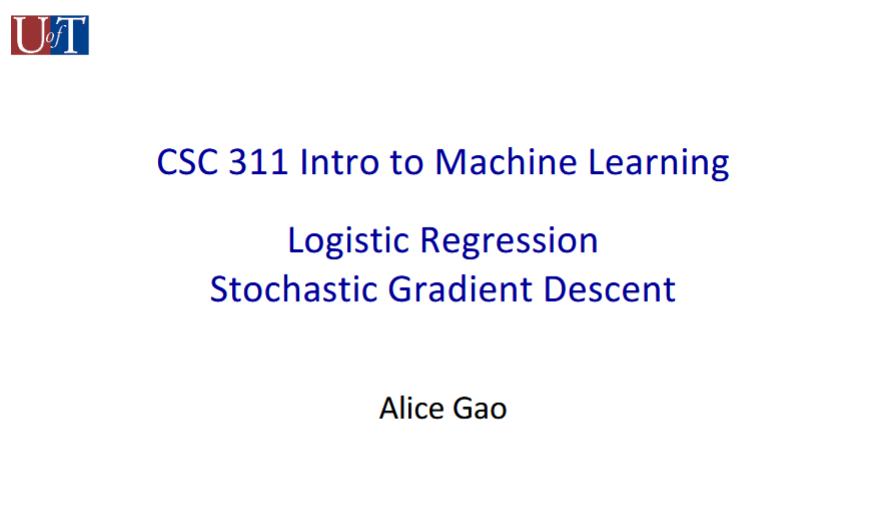
* Exercise: do this yourself
* Graph is the slice of the data space where x0=1 (since x0 is the dummy feature that is always 1)

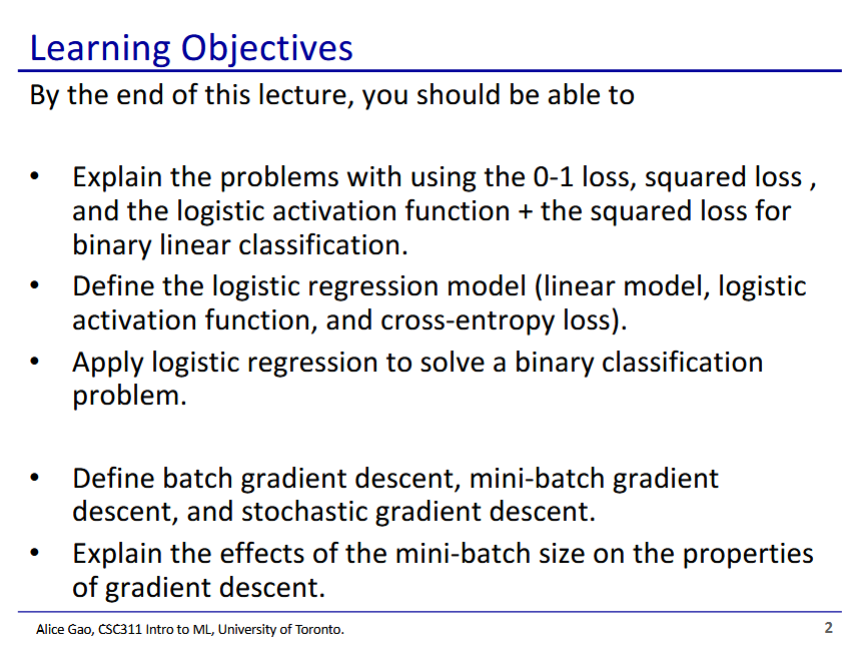






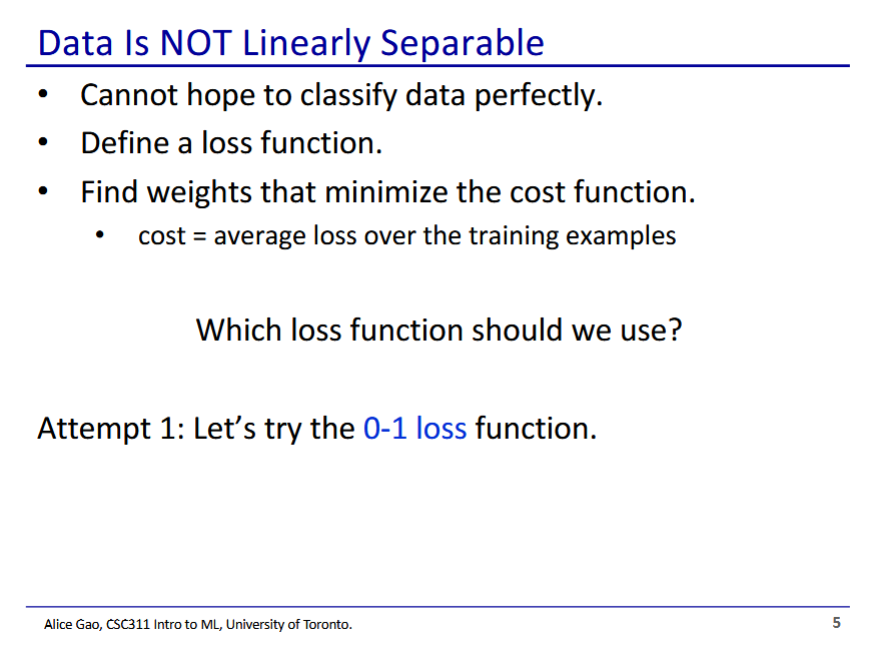
* Data is almost never linearly separable
  + Thus we need to use logistic regression

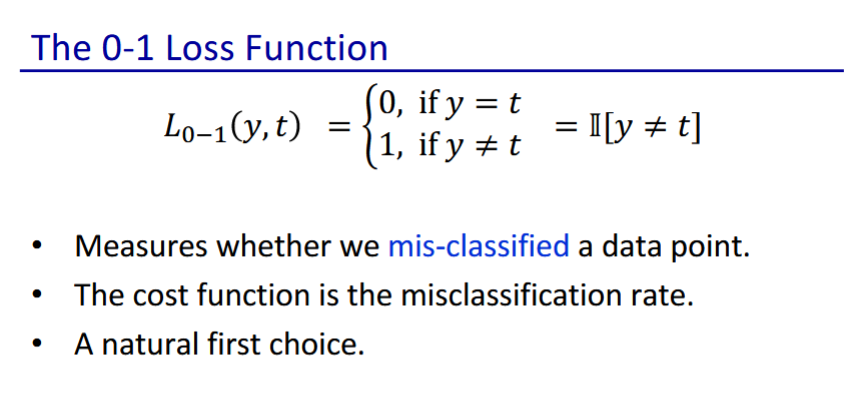




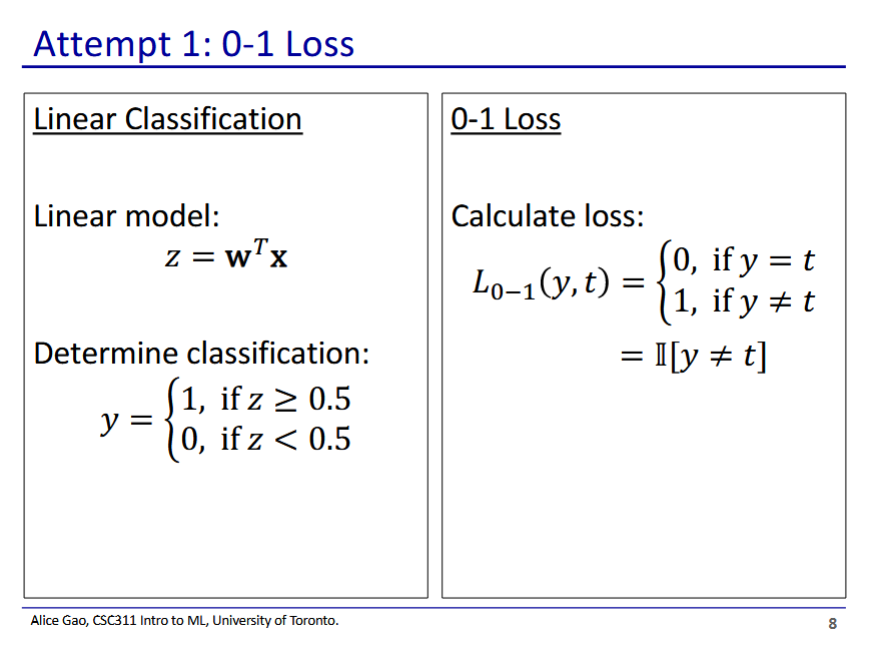




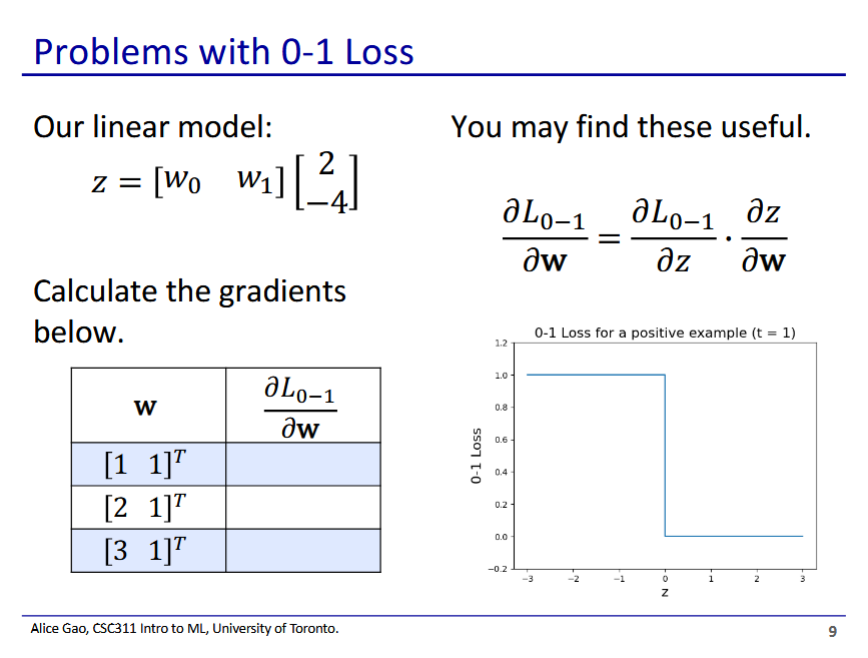




* 0-1 loss function
  + 1 if we got it right, 0 if we got it wrong

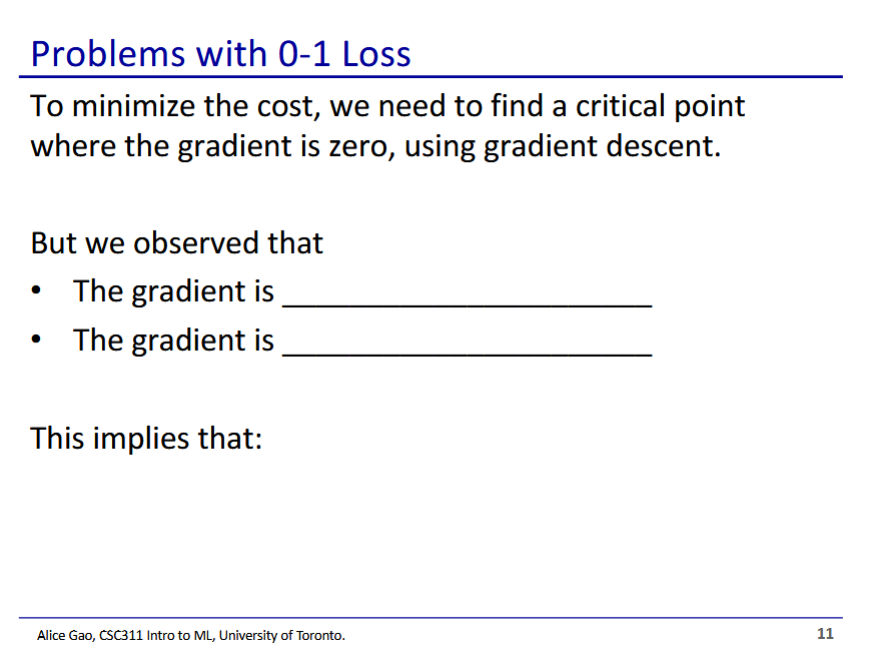


* We will try linear classification with the 0-1 loss function

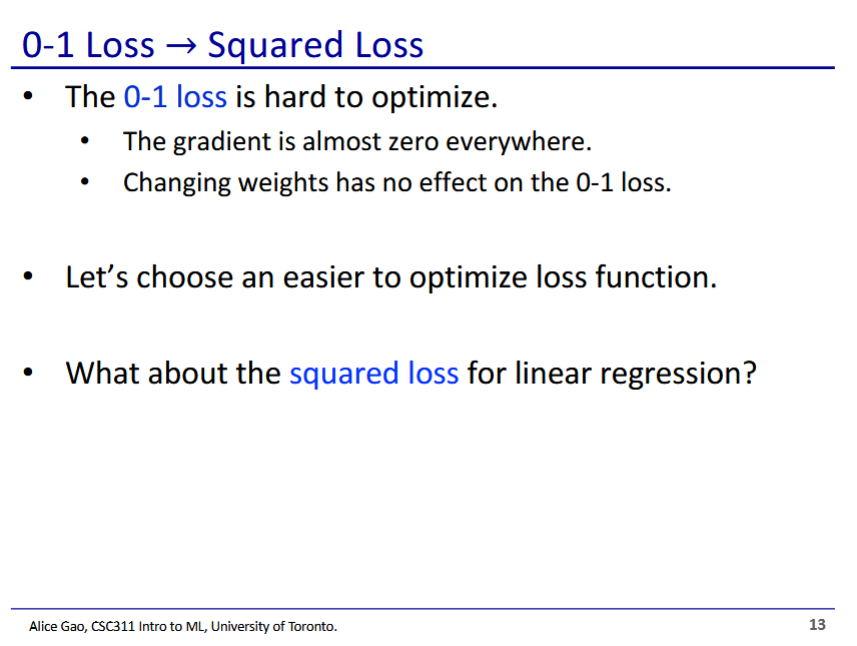


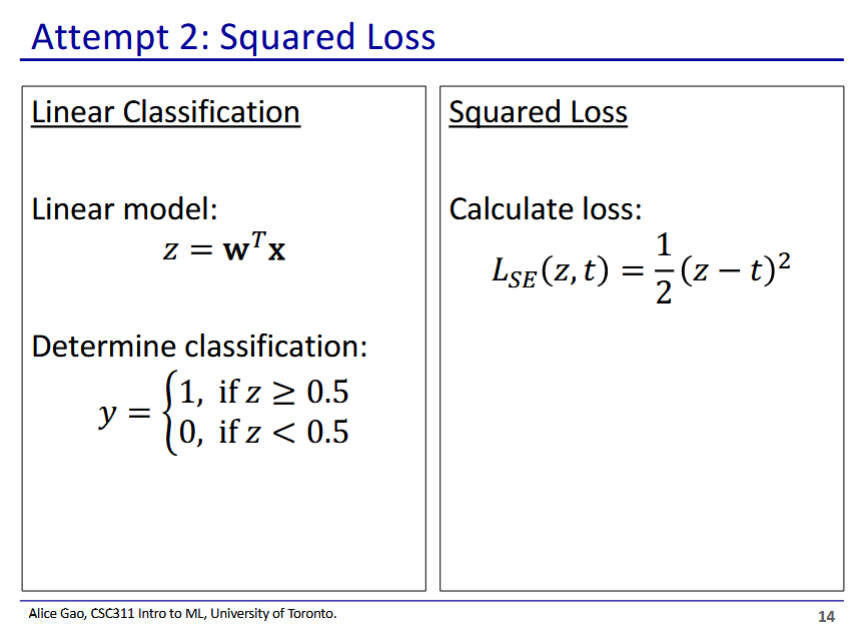
* First find z values for each of the weights by taking the dot product with [2 -4]T
* (chain rule)
  + when z=-2, 2, and is undefined when z=0 (see graph)
* Thus, the gradients are:

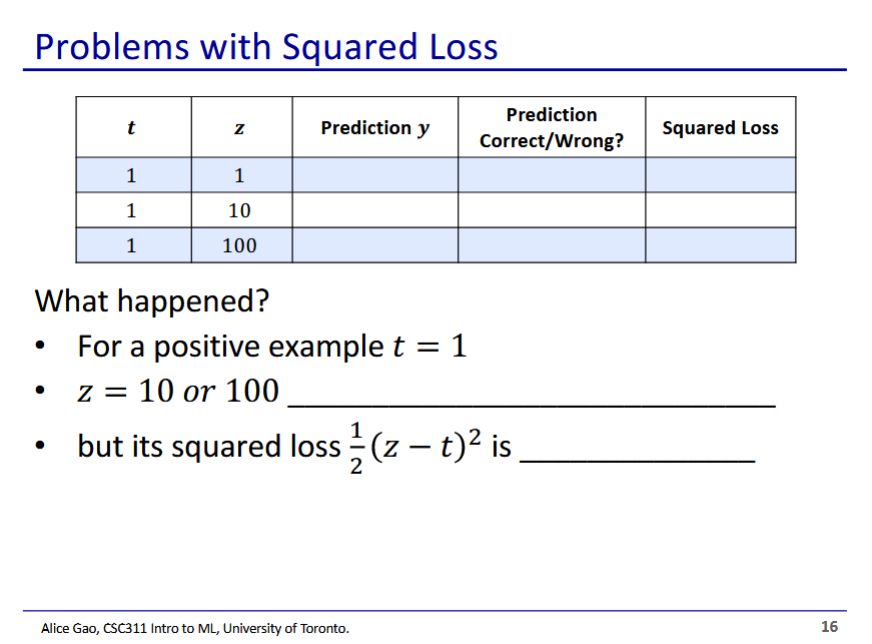
|  |  |  |
| --- | --- | --- |
| w | z |  |
| [1 1]T | -2 | 0 |
| [2 1]T | 0 | undefined |
| [3 1]T | 2 | 0 |



* The gradient is 0 almost everywhere, and undefined when z=0
* This implies that gradient descent won’t work, since there is no angled slope to guide each step towards the minimum

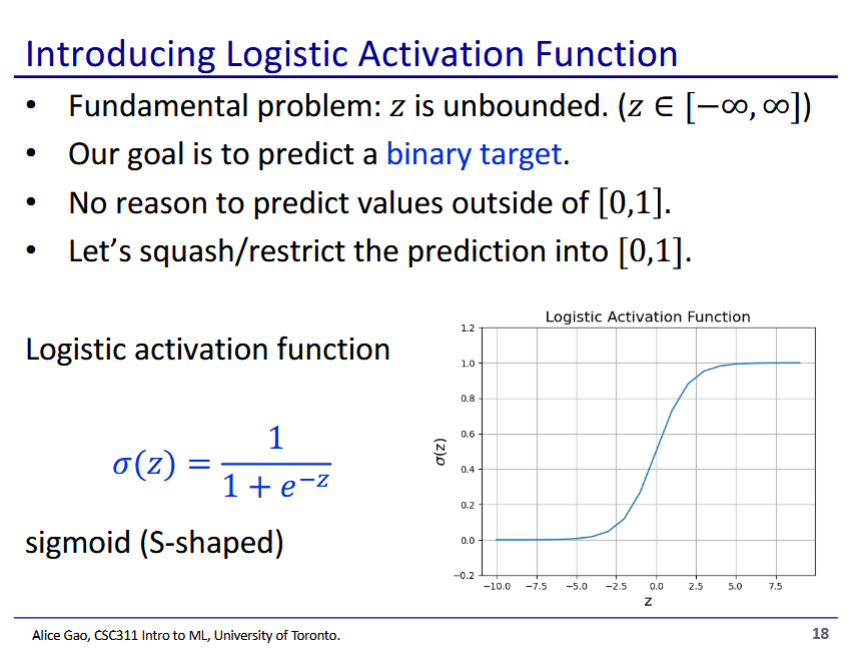




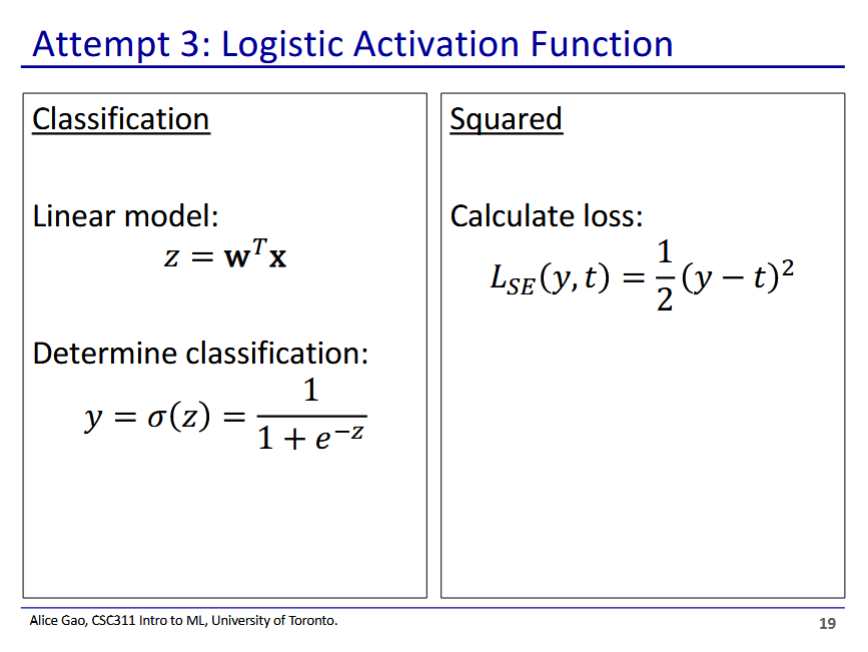


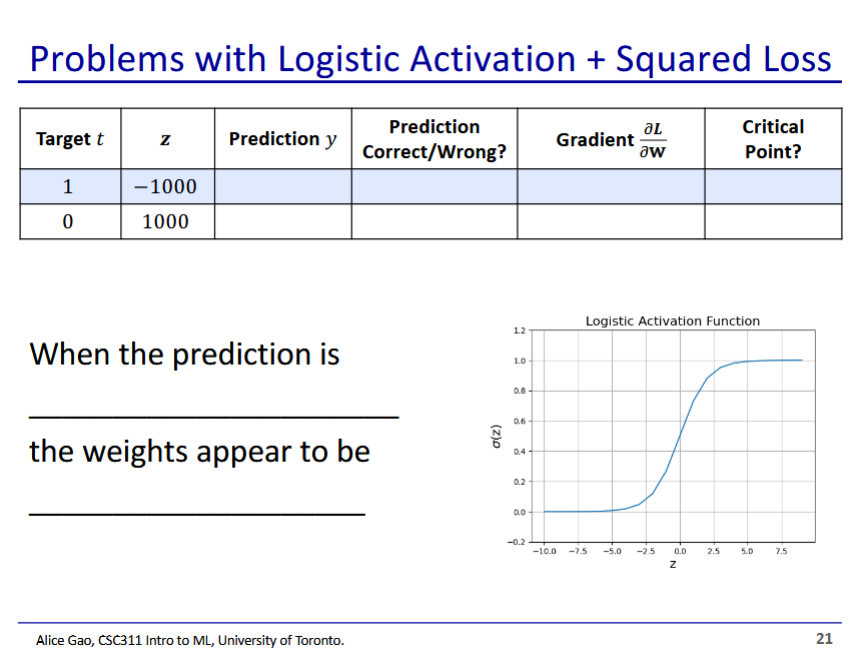
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| t | z | Prediction y | correct/incorrect | Squared loss |
| 1 | 1 | 1 | correct | 0 |
| 1 | 10 | 1 | correct |  |
| 1 | 100 | 1 | correct |  |

* Our model predicts correctly for each of these outputs, but we are getting very high loss when z=10 and z=100
  + Our loss should not be high when we are predicting correctly
* The problem arises as z is unbounded (linear), but t in our loss function is bounded
  + We can’t feed y into the loss function as that would just give us the 0-1 loss function



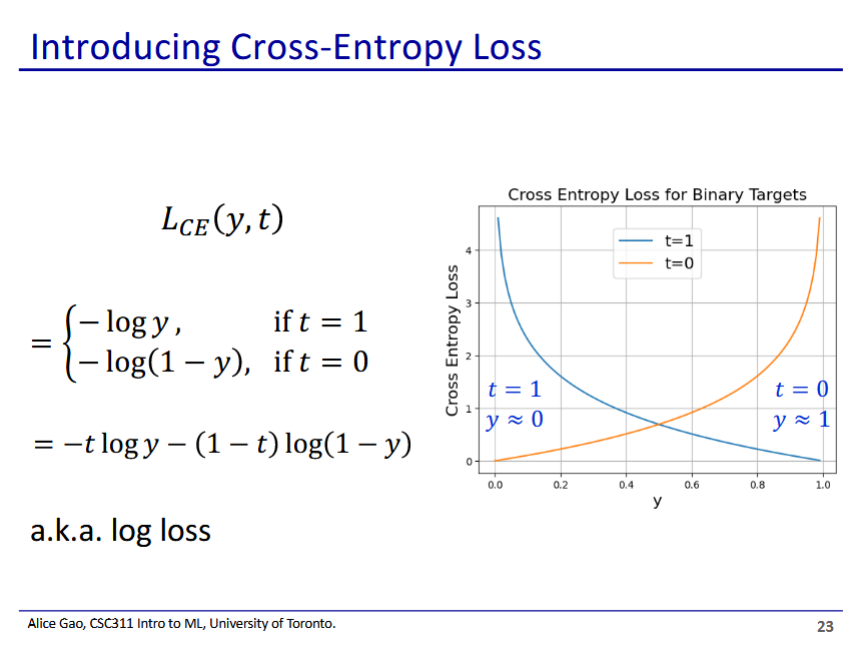
* We can use a function that is bounded to [0, 1], the sigmoid/logistic function
* This is a softer version of the step function
  + Like stepwise, but we have a slope that we can use with gradient descent



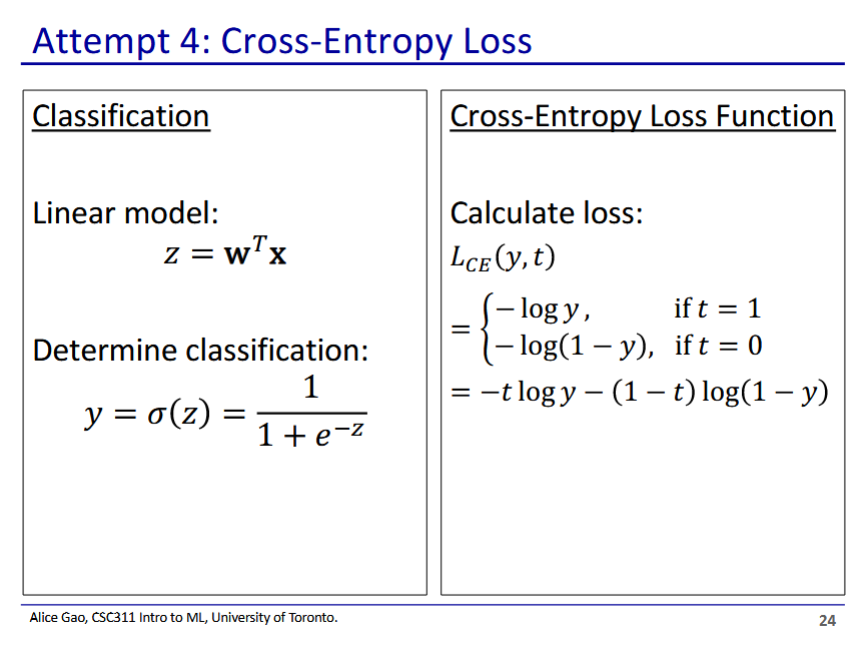


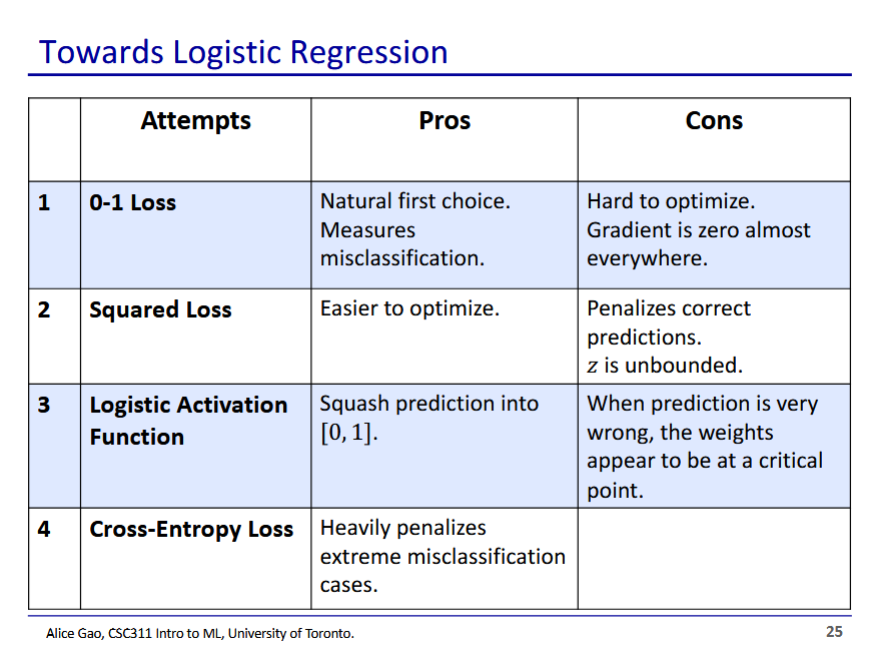
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| t | z | Prediction y | correct/incorrect | gradient | Critical point? |
| 1 | -1000 |  | incorrect |  | Yes |
| 0 | 1000 |  | incorrect |  | Yes |

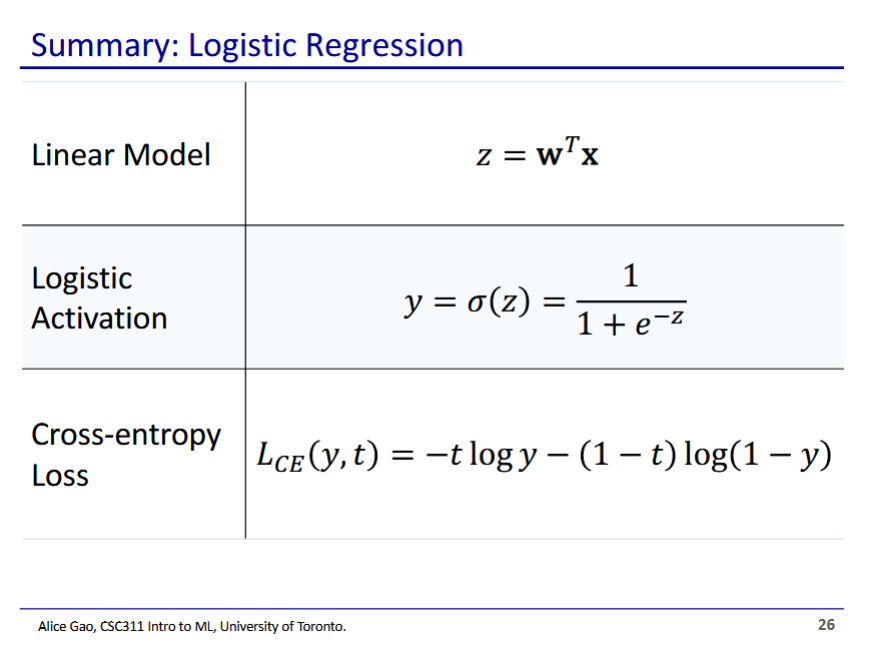
* The gradients for the loss function are all ~0 near the ends of the graph
* Our model is misclassifying, and yet our loss function gradient thinks we are at a critical point
  + This is bad, as gradient descent is not going to change this much

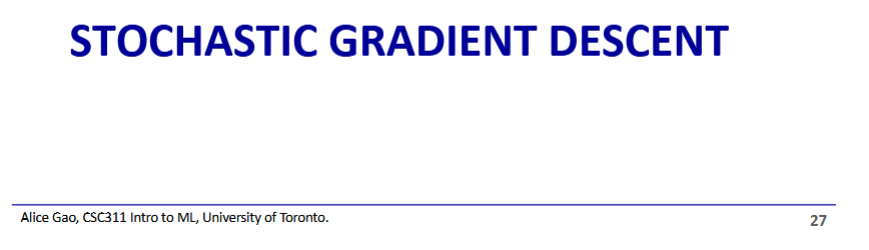


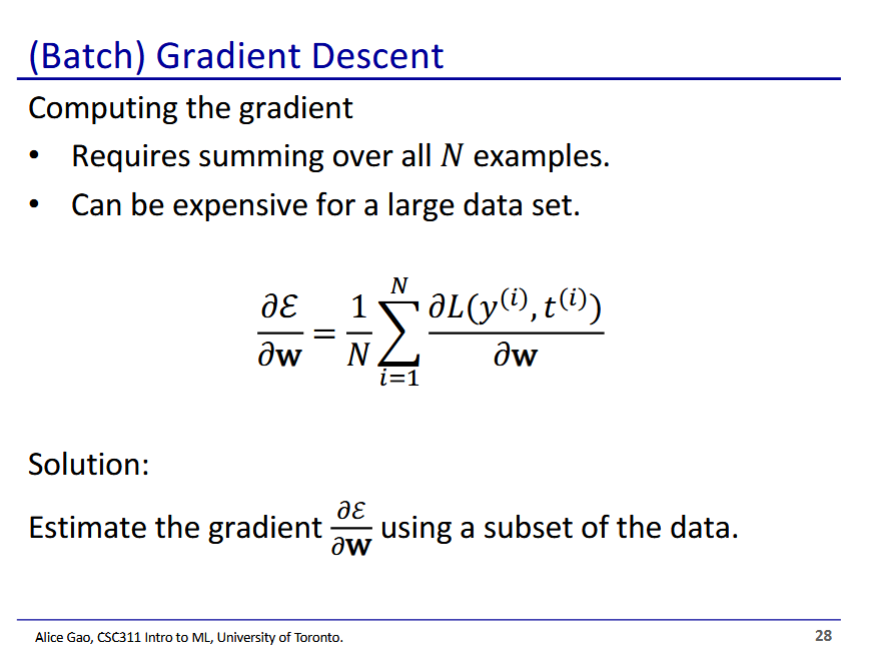
* These 2 are equivalent, since when t=0 the first part disappears, when t=1 the second part disappears



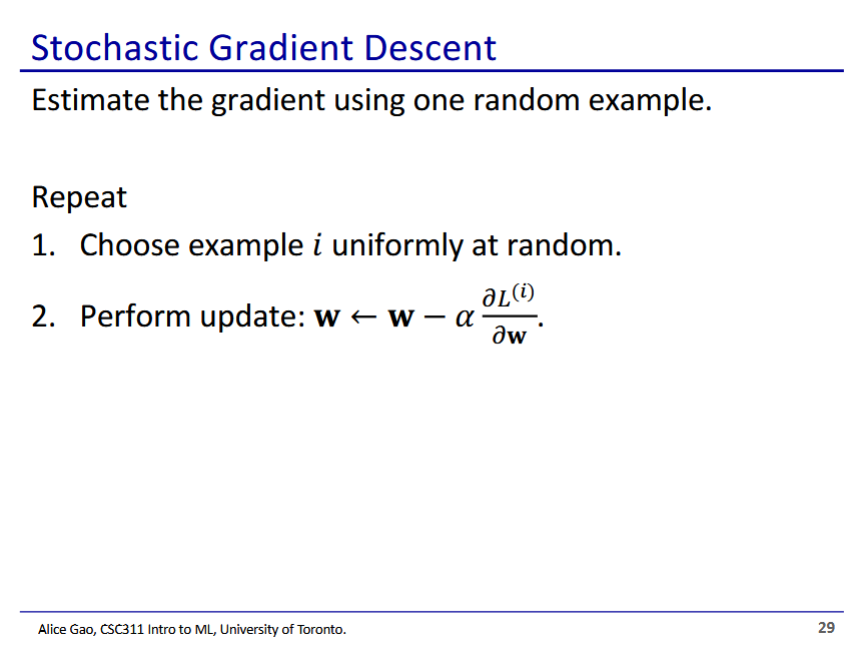




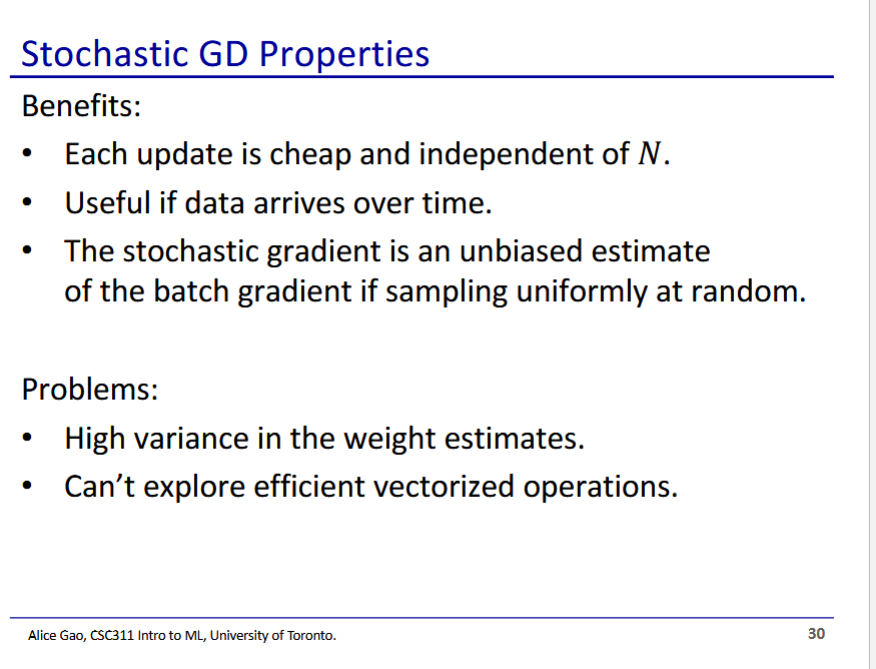




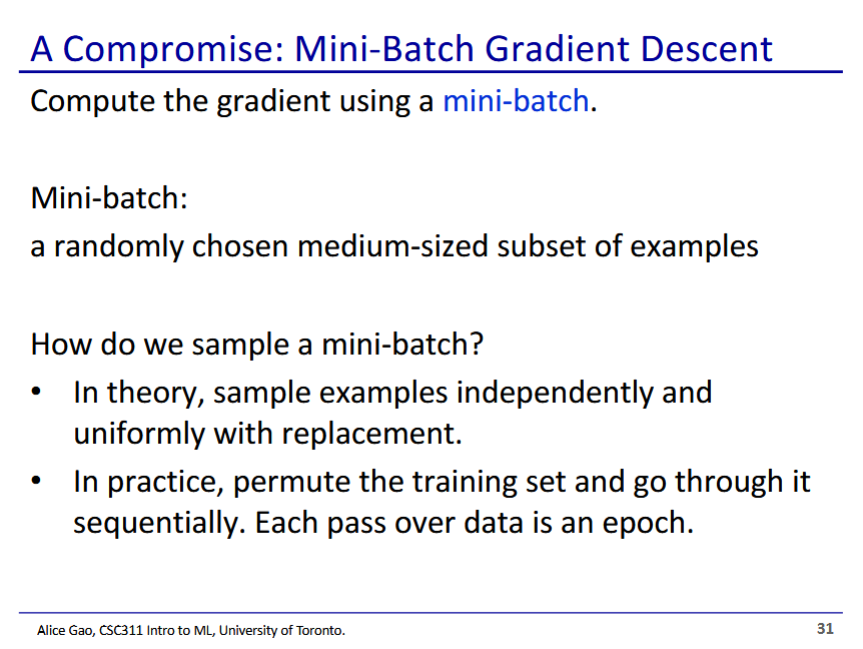
* We make an estimate of the gradient by selecting only some of the data



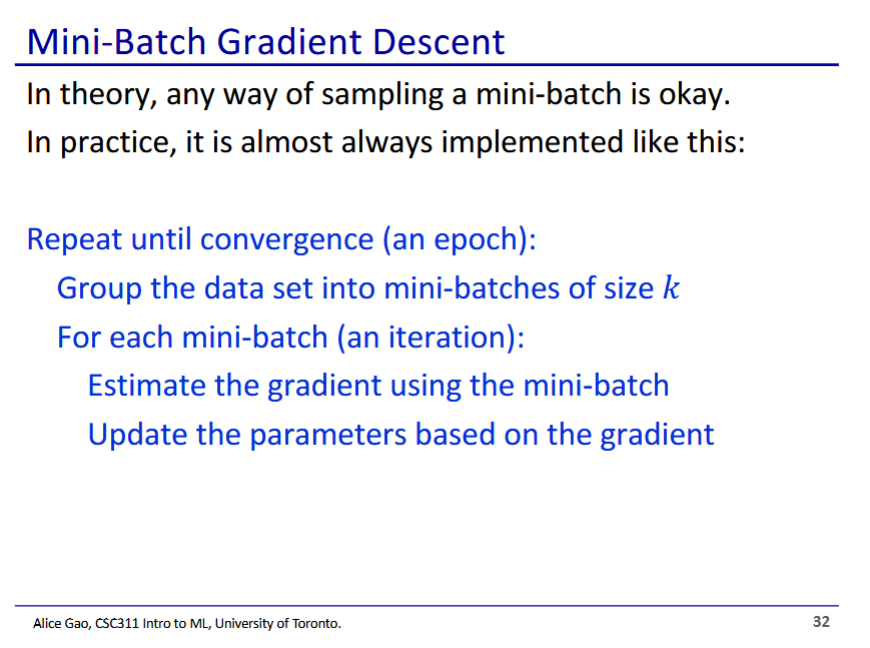
* Stochastic gradient descent - the most extreme, we choose our gradient based on only 1 example
  + Very fast, each update we only look at 1 data point
    - Also each update takes (1) time, is not related to the number of data points
  + Very noisy updates, sometimes updates will not go in the right direction
    - But will average out in the right direction



* Each update is noisy (high variance), but overall goes in the right direction (unbiased estimate)



* Instead we will compromise using mini-batch - a randomly chosen small subset of data



* We typically just group the data into mini-batches and iterate through mini-batches rather than randomly sample each time