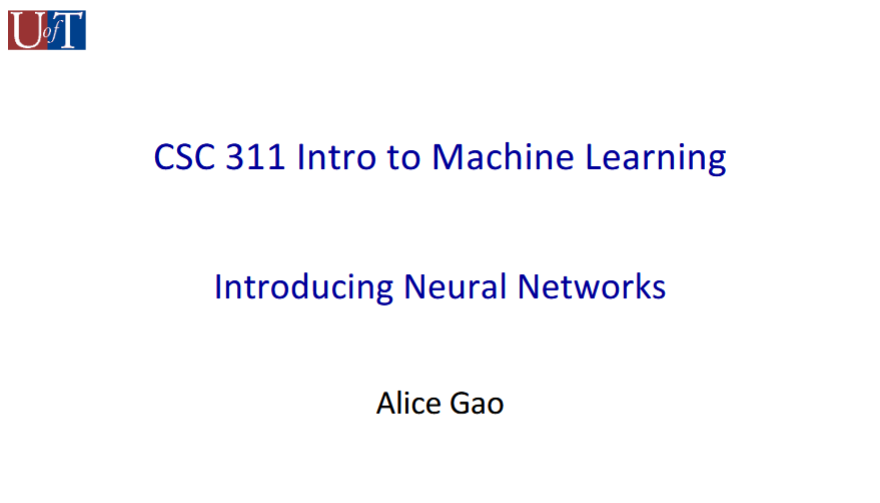
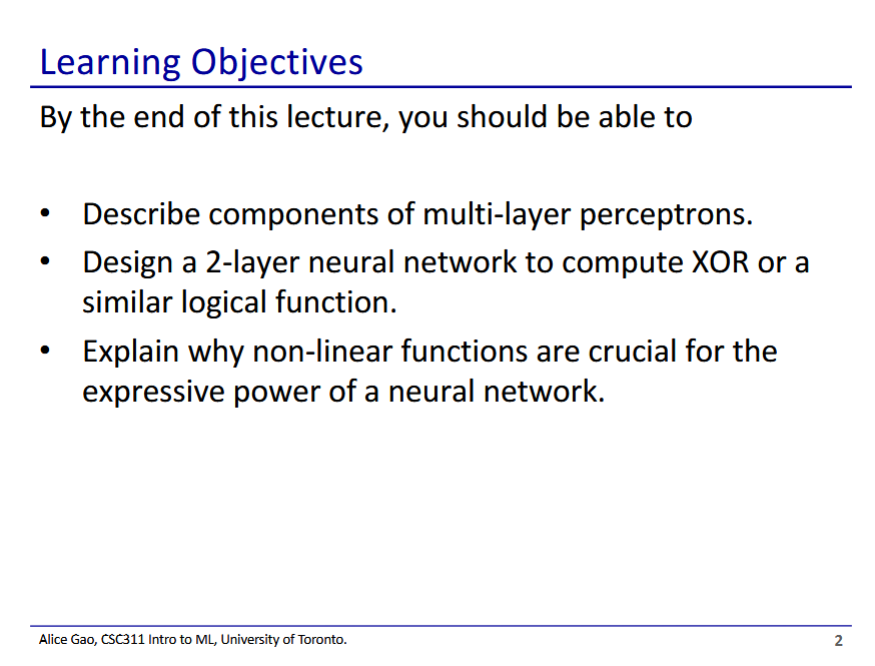
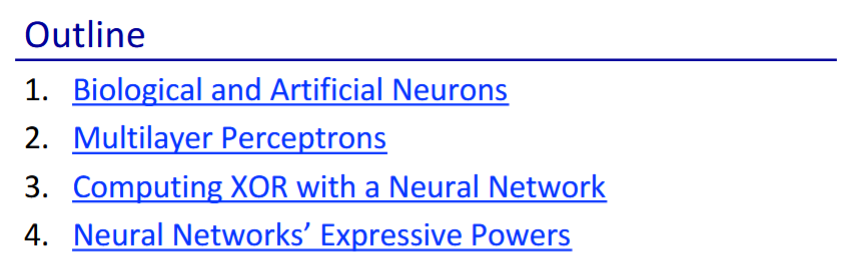
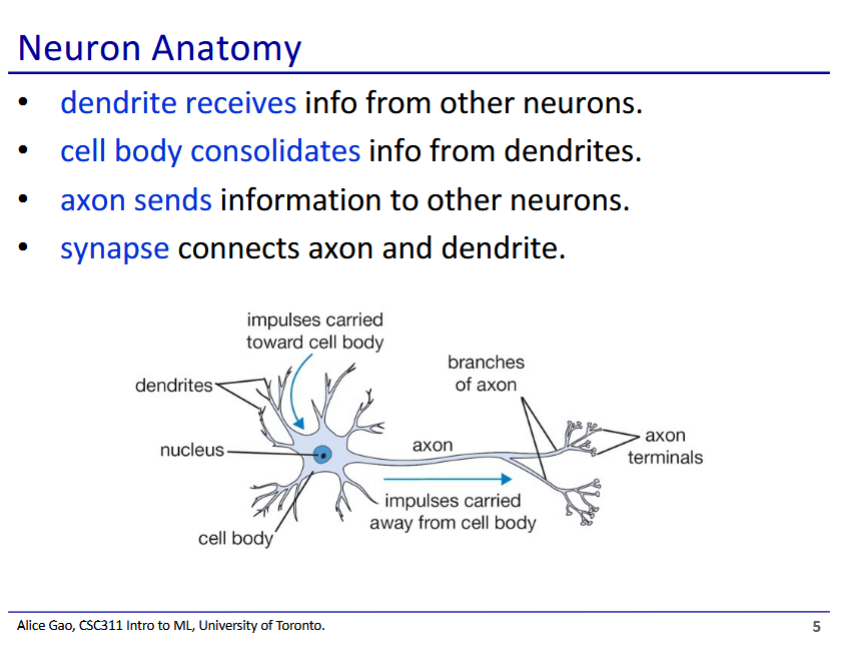
| **More admin things!**   * Lab 3 will be on this friday * Half of A2 is posted to give us more time to work on it   + Other 2 questions will be posted next week * A1 grading should be done by Thursday   **Artificial neurons**   * Each artificial neuron is a logistic regression model   + Each dendrite takes in a weighted feature of the input   + The cell body then sums the inputs and passes the value to the activation function (logistic activation or some other function)   + The output of the activation function is passed through the exon * A layer of artificial neurons forms a softmax regression model   + Softmax function used on the final outputs of the layer   **Multilayer perceptrons**   * Acyclic/feed-forward neural network with multiple layers   + Information flows down the neural network in one direction as one layer feeds into the next     - Weight matrix dimensions exercise on slide 15   + In contrast to recurrent neural networks, where feedback of information occurs * Layers   + Input layer     - Input features fed in here     - Not counted as one of the layers since no processing is done here   + Hidden layers     - Layers between the input and output layers     - We do not typically observe the values for these layers as they are not meant to be interpretable   + Output layer     - Shows the output   + How many layers, how many nodes in the layers, and which activation functions to use are hyperparameters * Activation functions   + Logistic     - Ranges from 0-1 (disadvantage since we sometimes want a negative output)     - Has gradients of 0 near extremes     - Not really used anymore in favour of tanh activation   + Tanh     - Shaped like the logistic function but steeper and ranging from -1 to 1     - Also has gradients of 0 near extremes   + ReLu (rectified linear unit)     - Returns 0 when z is negative, returns z when z is nonnegative     - Best choice in practice   **Building a neural network by hand (example: XOR)**   * We can break the not linearly separable function (XOR) into a combination of linearly separable functions (NOT, AND, OR) using **sum of minterms** * We can then use this representation to construct the neural network * Example on slides 24, 25 (probably incomplete) |
| --- |



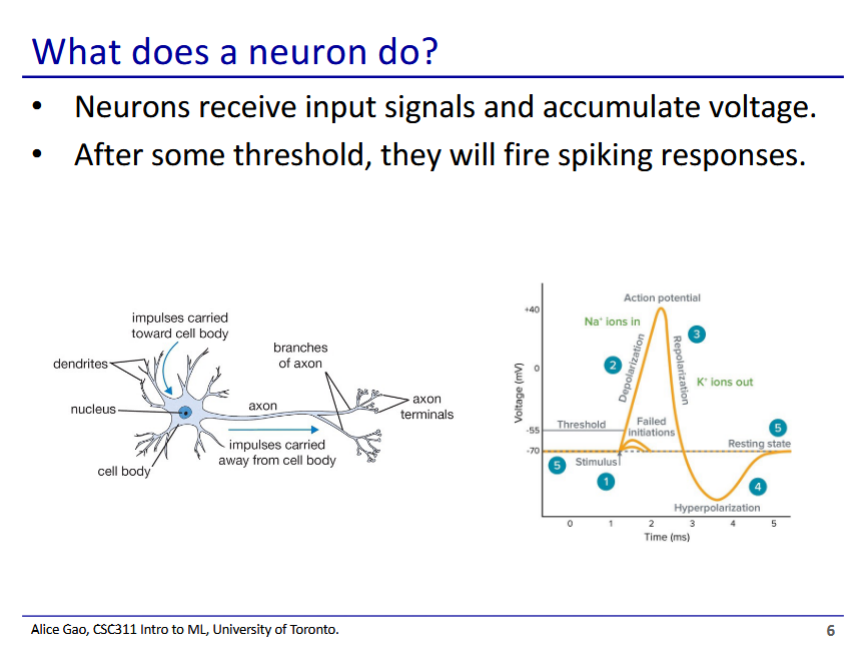




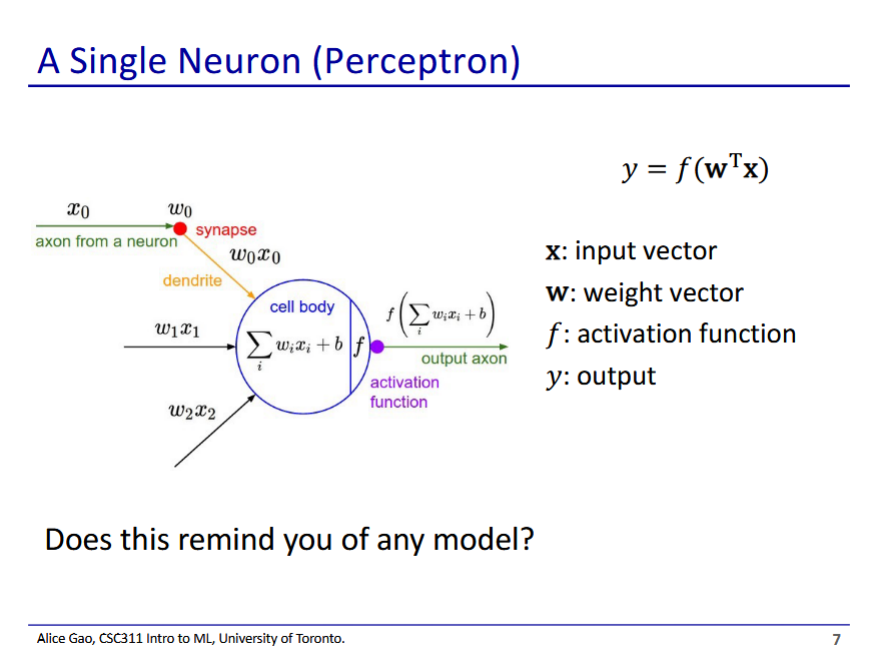




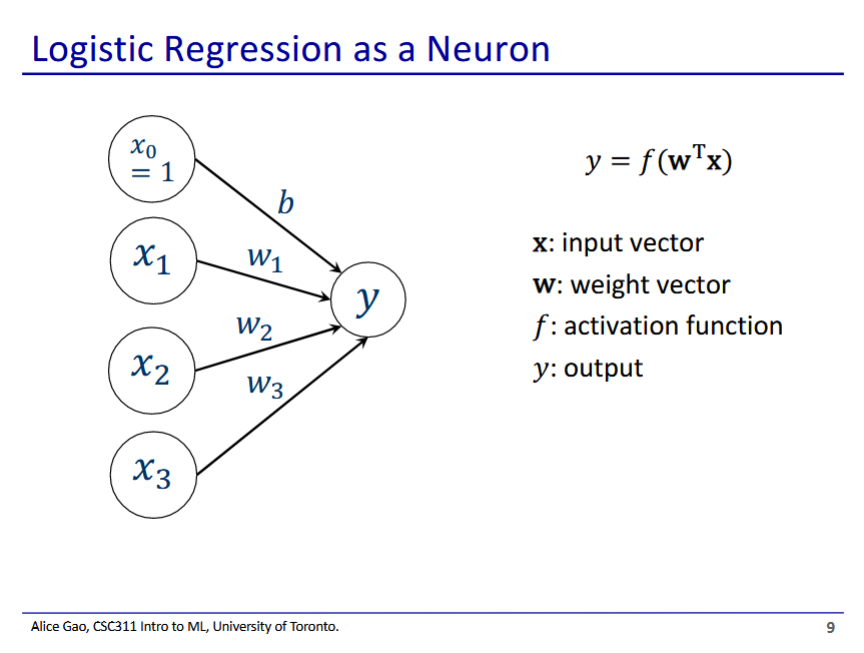
* This is a biological neuron, artificial neurons are sort of based on biological neurons



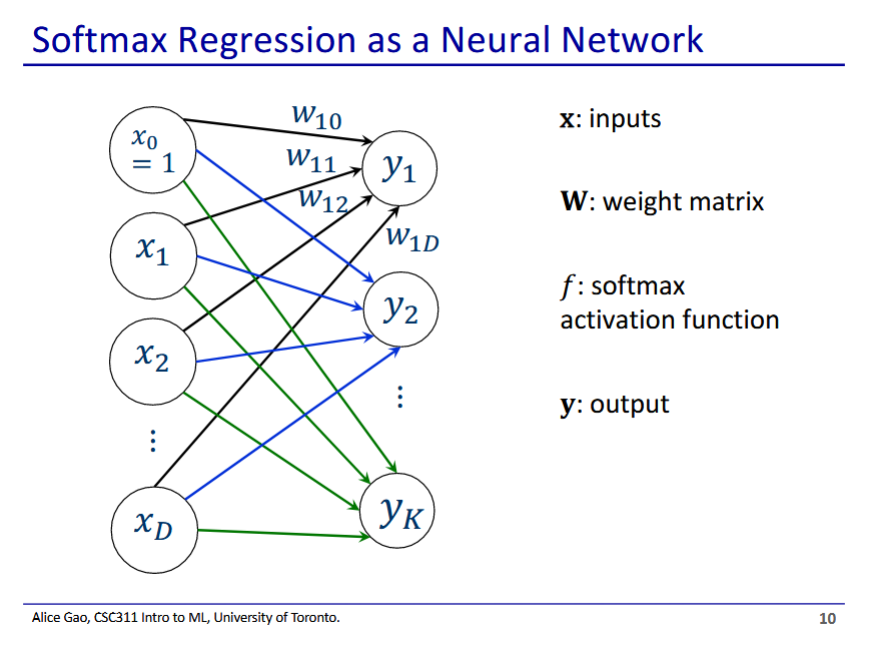
* The neuron makes a very basic calculation
  + Once input voltage reaches a certain threshold, the neuron fires



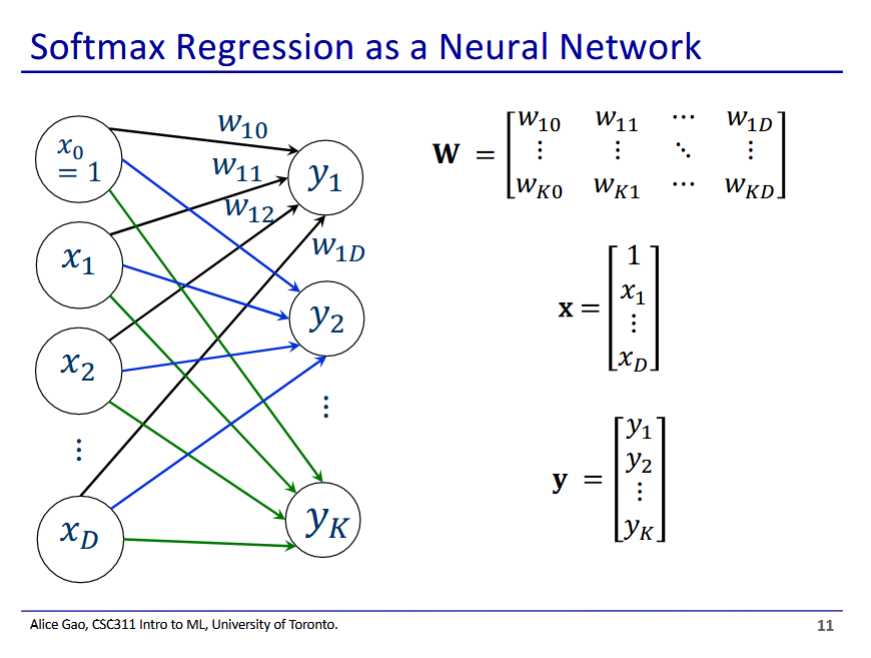
* Some familiar things
  + We have weights used to calculate a linear combination of inputs
  + We then have an activation function that makes an output
* This is logistic/softmax regression



* Logistic/softmax regression is essentially a 1-layer neural network
* This diagram shows a 1 layer neural network that represents logistic regression
  + Has 1 binary output

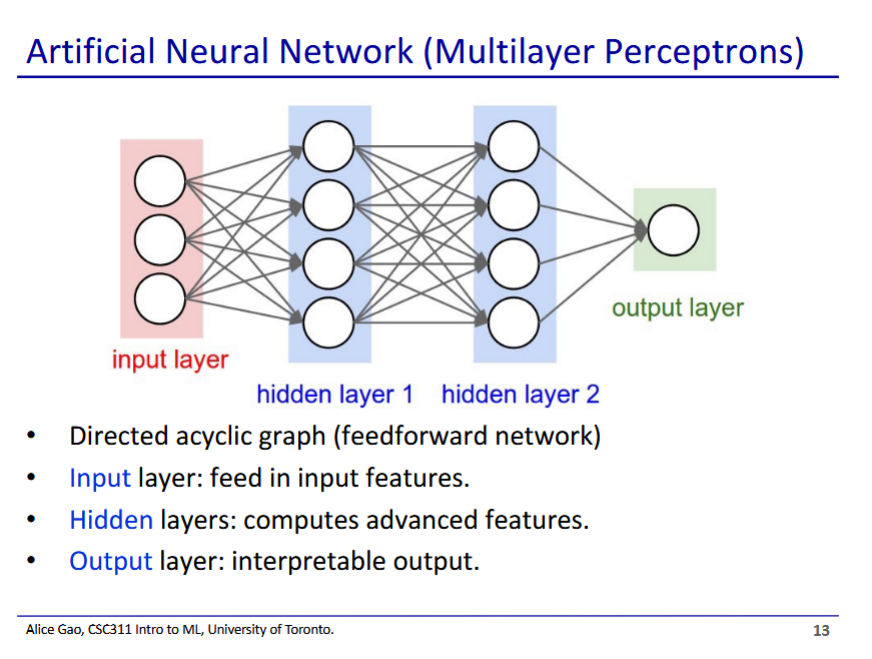


* This is a 1-layer neural network that represents softmax regression
  + Softmax regression used for categorical output
* Activation function is not shown in this diagram

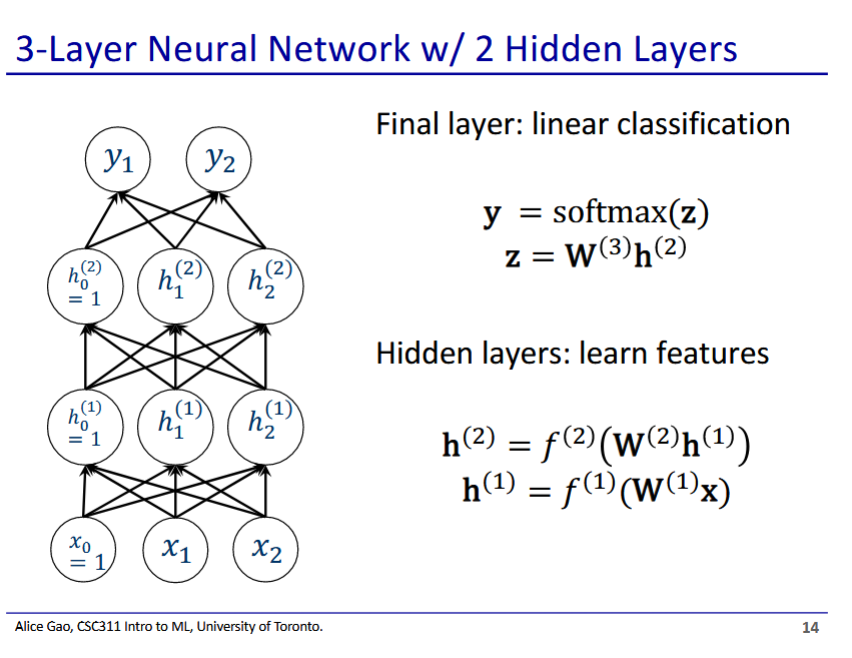


* Weights have 2 subscripts
  + First subscript corresponds to the output
  + Second subscript corresponds to the feature of input
  + W1x corresponds to the weight for the 1st output applied to the x’th input feature

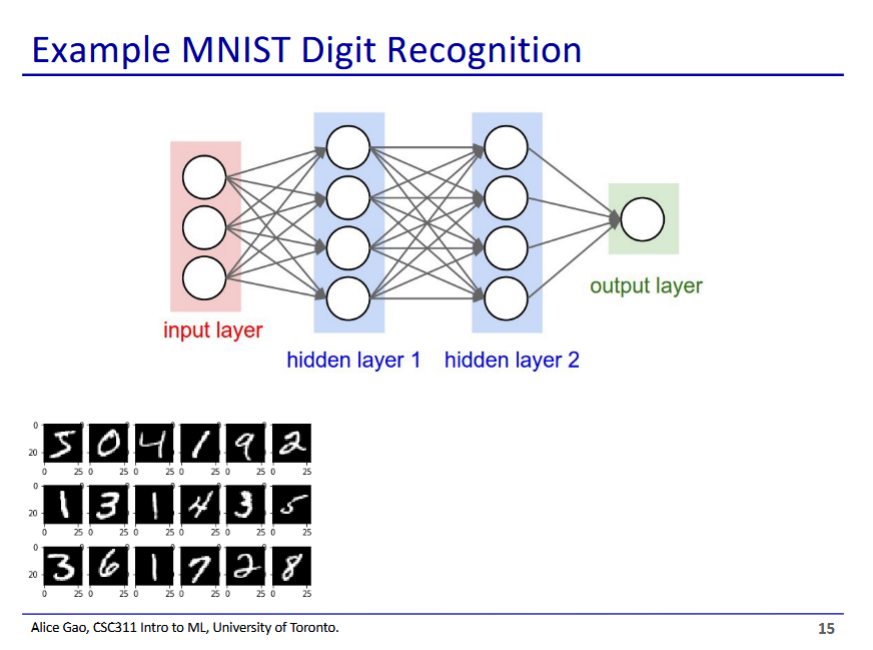




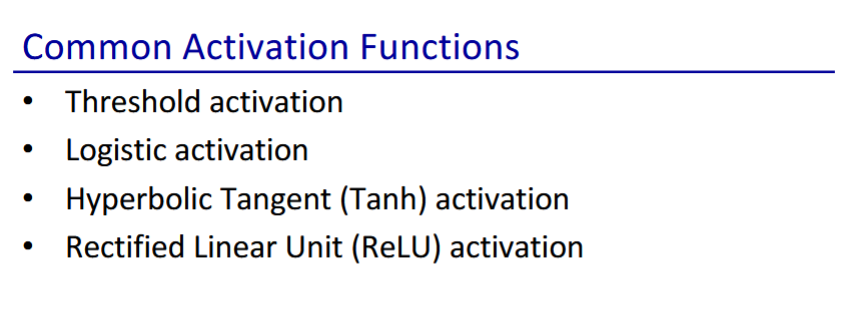
* Acyclic (feed-forward) neural network - information only flows in one direction down the neural network
  + Recurrent neural networks - information can be looped back into neural network
* Hidden layers
  + We do not typically observe the values for these layers
  + We don’t design the network such that hidden layers are interpretable
    - Input and output layers are designed to be interpretable

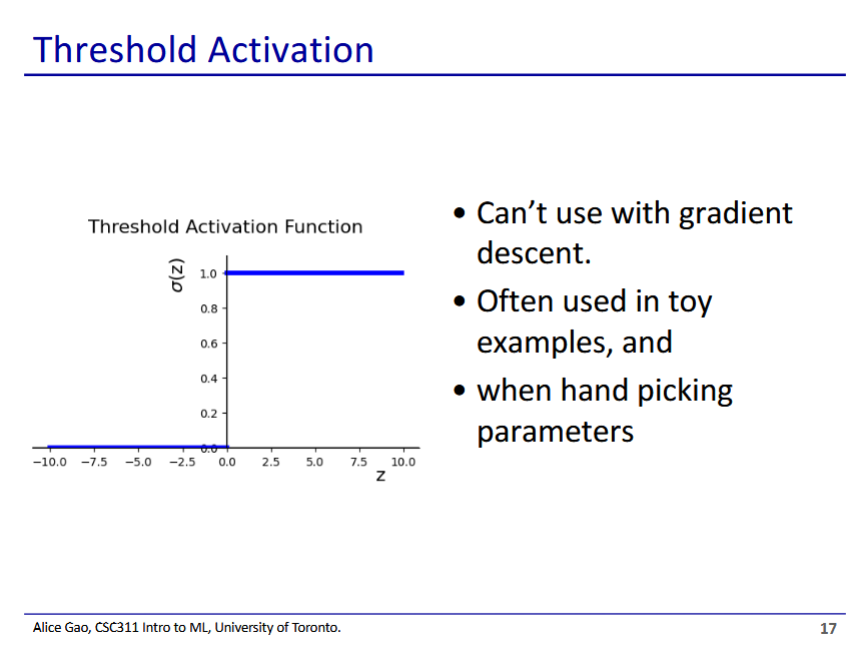


* 3 layers - input layer doesn’t count since it doesn’t do any computations
* Output of previous layer is fed into next layer as an input
* Why do we need a bias/dummy variable?
  + The bias shifts the hyperplane away from the origin
  + I.e. if we had a linear model without a bias, our regression line must pass through the origin

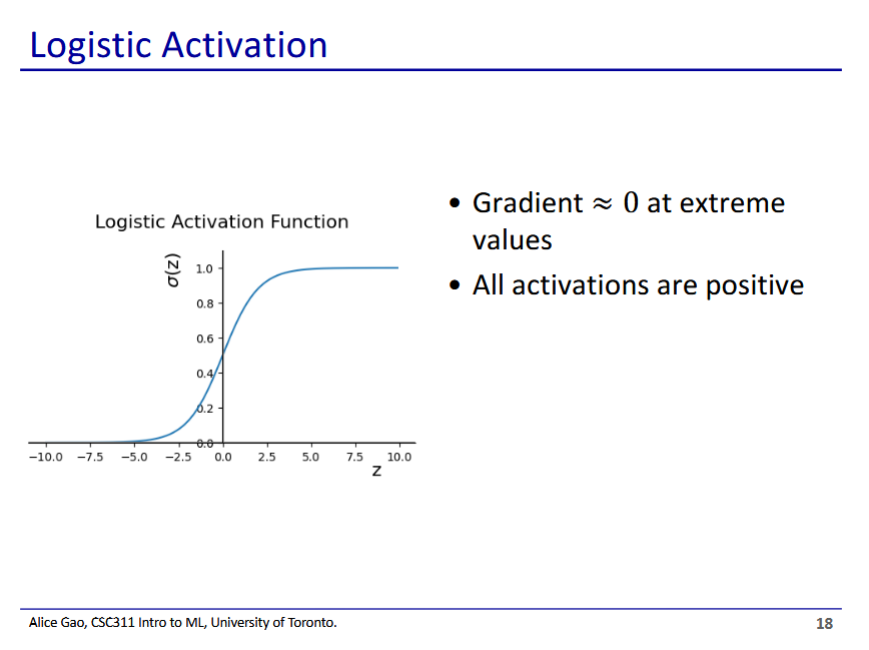


* Example: if we have 100 nodes in hidden layer 1 and 50 nodes in hidden layer 2, what are the dimensions of our input, weights, and target vector/matrices?
  + Input vector: 784x1
    - Each pixel is a feature
  + Target vector: 10x1
    - We have 10 digits to choose from
  + W1 (weights for hidden layer 1)
    - 100 x 784
  + W2
    - 50 x 100

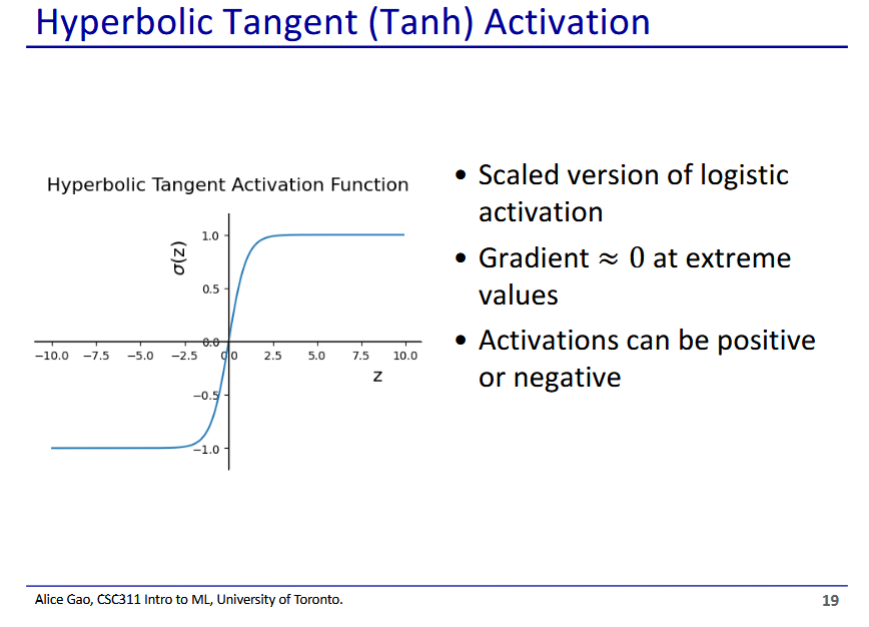




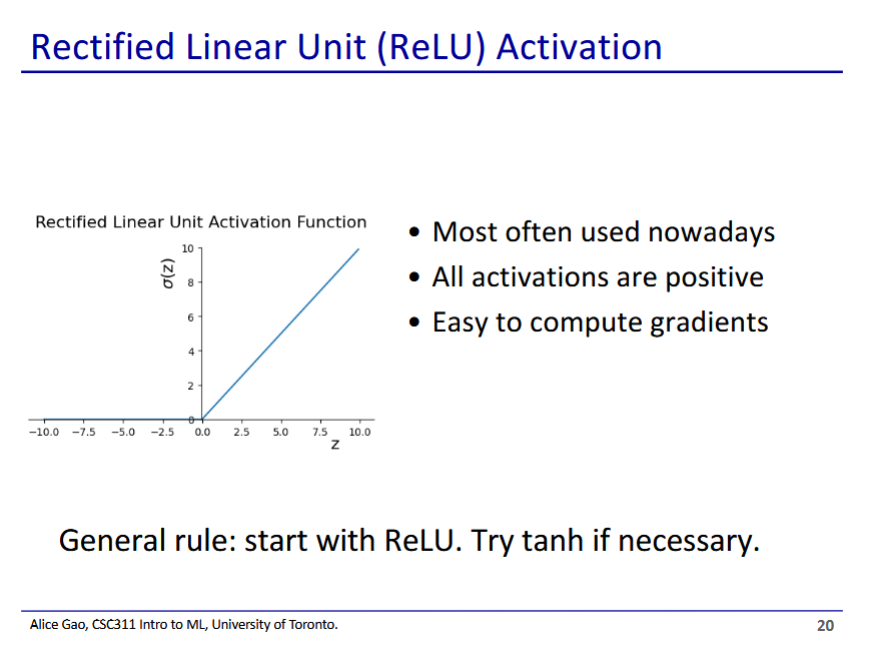
* Threshold is simple, but doesn’t work well in practice
  + Like with the 0-1 activation function, we can’t use linear regression
* Where will we see this?
  + Lecture, assignments, and tests (because it is simple)
  + But not really in real life



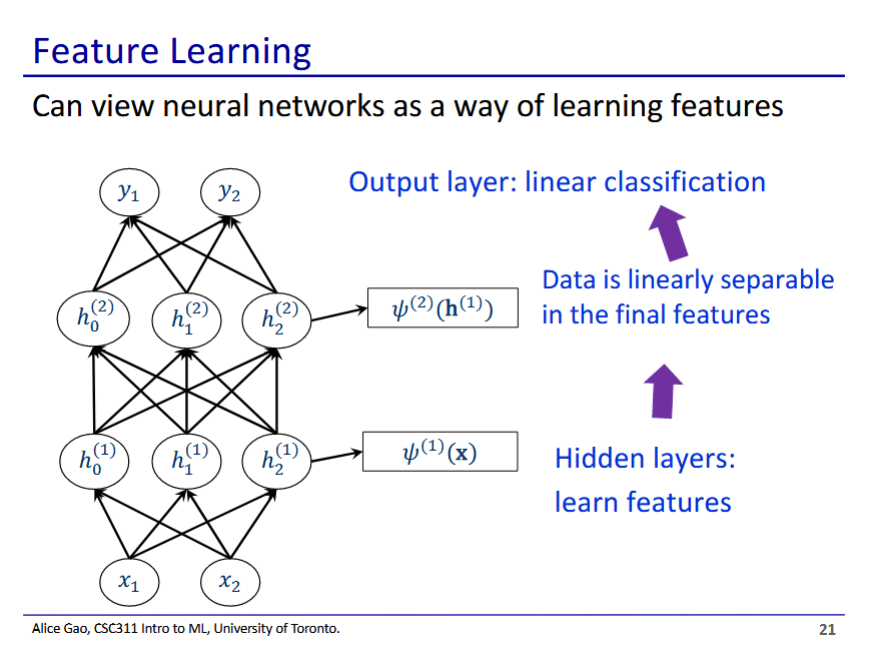
* Logistic function approximates the threshold function but works with gradient descent
* Has some problems
  + Gradient is ~0 near the extreme values, which causes some problems for gradient descent
  + All activations are positive - some models have both positive and negative weights
* Logistic and threshold activation functions only really exist for historical reasons
  + Not the top choices nowadays



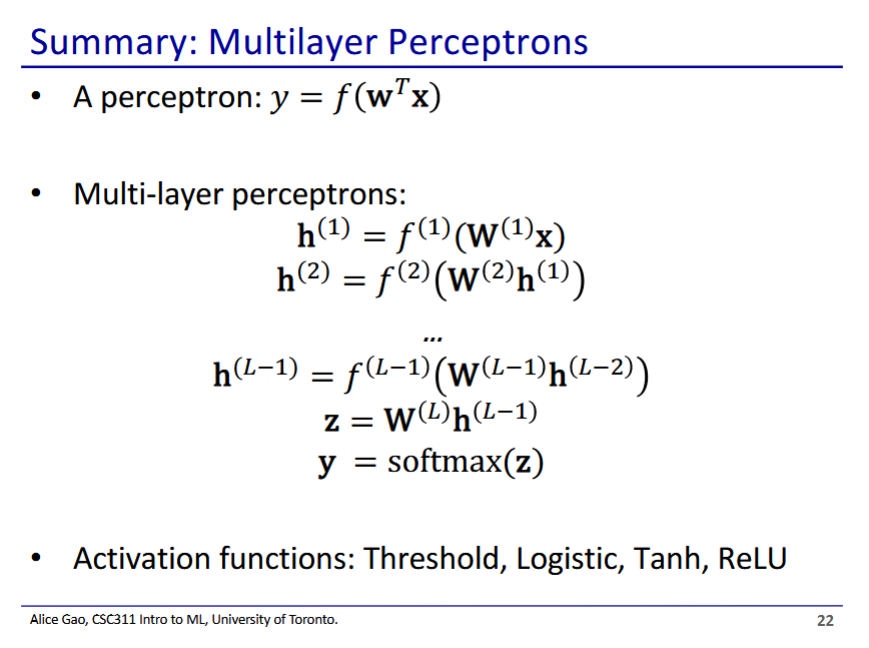
* Hyperbolic tangent - like logistic, but scaled from -1 to 1 rather than 0 to 1
  + The middle part is also more steep, which can help with gradient descent too
* Still has the problem that gradient ~0 at extreme values



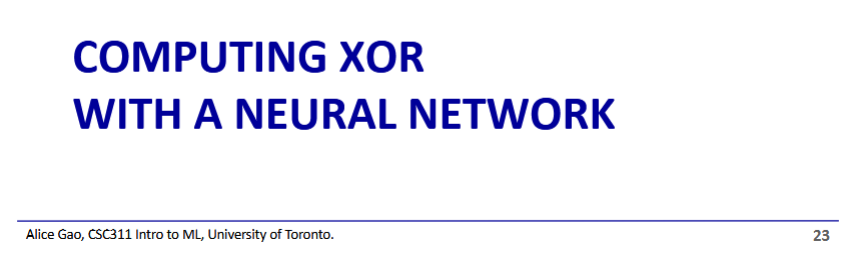
* Very popular and typically best choice in practice
* Explanation
  + If we have a negative z, return 0
  + If we have a positive z, return z

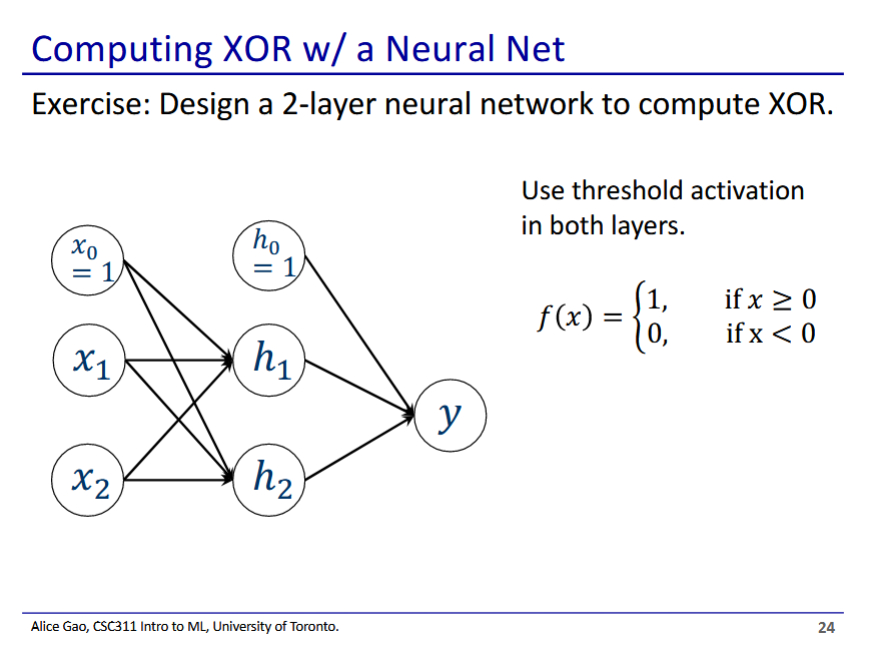


* We can use neural networks to learn features
  + Each hidden layer is essentially a feature map
    - After several hidden layers, we end up with data that is linearly separable
  + Good alternative to deciding feature maps by hand (hard!)

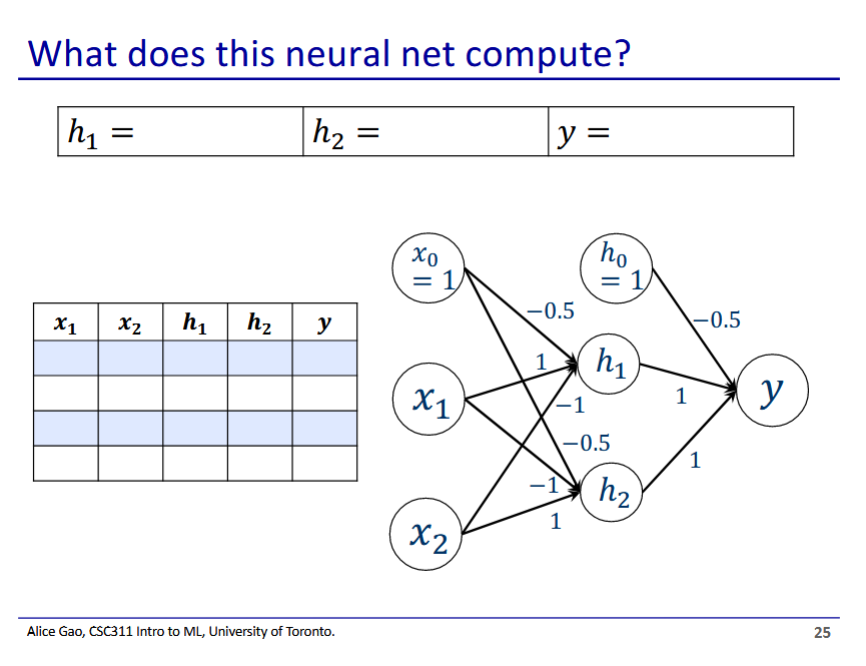


* How to design the hidden layers?
  + How many layers, how many nodes in each layer, what activation function to use are all hyperparameters





* This is a neural network that works by hand to compute XOR
  + Using the threshold activation function, since we are doing this by hand
* How do we approach making a neural network like this by hand?
  + XOR returns 1 if the inputs are different and 0 if they are the same
  + We can split XOR as
    - Alternatively as
  + NOT, AND, and OR are all linearly separable



| x\_1 | x\_2 | h\_1 | h\_2 | y |
| --- | --- | --- | --- | --- |
| 0 | 0 |  |  | 0 |
| 0 | 1 |  |  | 1 |
| 1 | 0 |  |  | 1 |
| 1 | 1 |  |  | 0 |

* h\_1 calculates
* h\_2 calculates
* y calculates