Lecture 21

⇛ Before we get to neural networks, we'll need some tools in order to train the networks.

- **Loss Functions** ⇝ or "how do I penalize model errors?"

...lots of choices ⇛ y = label (data)
$f(x)$ = model prediction

* L2 loss = $[y - f(x)]^2$
* L1 loss = $|y - f(x)|$ (Huber)

⇛ the losses are aggregated over all data in order to make the COST FUNCTION

LOSS	COST
L2	MSE / χ^2
L1	MAE

↳ MSE / χ^2 is the most popular cost function
↳ helps for it to be easily differentiable.

⇛ Often attempt several cost functions when tuning a ML / NN architecture.

Gradient Descent

* What's the best way to find the valley floor?
  - start walking downwards ....
    i.e. move AGAINST THE GRADIENT

$$\Delta y = \frac{dy}{dx}\Delta x$$

$$y^{i+1} = y^{i} - \frac{dy}{dx}\Delta x$$

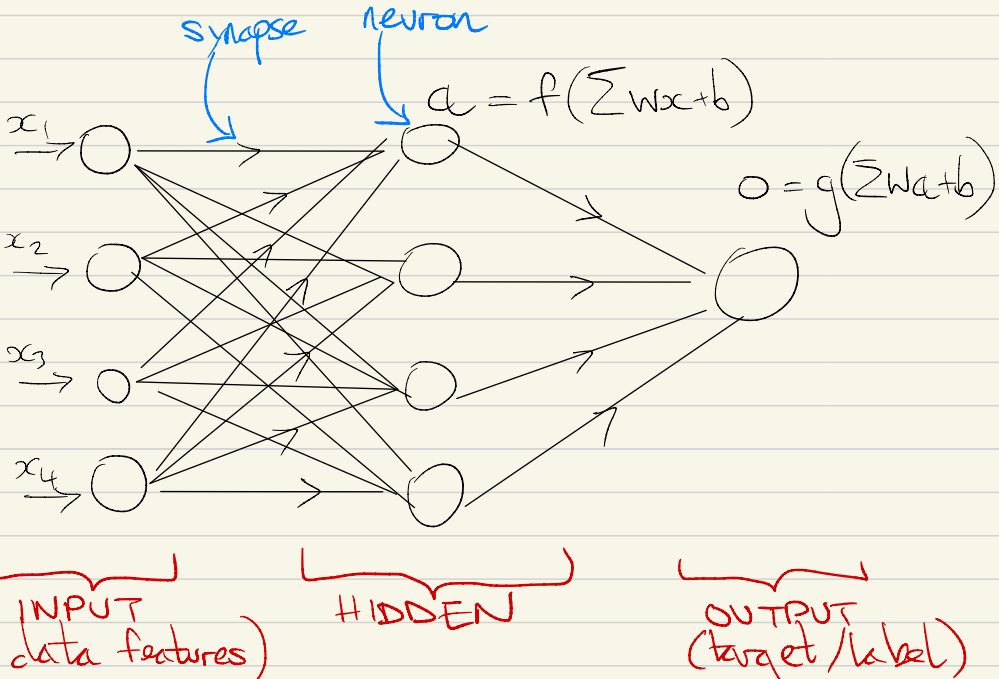
⇒ more generally .... $MSE(\theta) = \frac{1}{N}(y - X\theta)^T(y - X\theta)$

$$\nabla_\theta MSE(\theta) = \frac{2}{N} X^T(X\theta - y)$$

⇒ $\theta^{next} = \theta^{now} - \eta \nabla_\theta MSE(\theta)$

η: learning rate
--not too big/small

- **AdaBoost** ~ Adaptive Boosting
  - ‖→ Meta-estimator that uses gradient descent to tune classification performance.
- **Neural Networks** ~ ANNs are tools based on a metaphor with neurons.



* Synapses in hidden layers form a weighted sum of all inputs, add a bias, then pass to a neuron.

$$z = \sum_i wx_i + b$$

* Neurons sum this input, then pass it through a non-linear activation function

e.g. SIGMOID ACTIVATION FUNCTION

$$a = \frac{1}{1+e^{-z}}$$

* Backpropagation = clever way of finding gradient of cost function analytically.

* How many layers? ⇒ more layers gives more non-linearity.

* How many neurons? ⇒ roughly between $\sim 2 n_{input}$ and $n_{input} + n_{output}$.

* Which activation?
  - sigmoid = binary classify
  - ReLU = hidden layers
  - softmax = >2 classes classification

* Regularization can also help constrain the network architecture optimization.