On Tuesday we saw the model where at each time $t$:

- The world / adversary produces a loss vector $l^t = (l^t_1, l^t_2, \ldots, l^t_N) \in [0,1]^N$

- We have to produce a probability vector $p^t = (p^t_1, p^t_2, \ldots, p^t_N) \in [0,1]^N$ with $\sum p^t_i = 1$

Of course both are done simultaneously without knowledge of each other, like separated by a curtain.

Call this set $\Delta_N = \{ x \in [0,1]^N : \sum x_i = 1 \}$ "probability simplex".

Then curtain opened and...

Our loss = $\langle l^t, p^t \rangle$

We can now "learn" from this info of $l^t$ (we see $l^t$) to decide on new probability vector $p^{t+1}$.

(Of course the adversary can also learn)

Fix $\epsilon > 0$

Now: want a strategy to generate these $p^t$ vectors such that

$$\sum_t \text{our loss @ } t \leq \sum_t l^t_i (1 + \epsilon) + \frac{O(\log N)}{\epsilon}.$$  

$$\sum_t \langle e^t, p^t \rangle \leq \sum_i l^t_i + \left( \epsilon T + \frac{\log N}{\epsilon} \right)_{\text{best in hindsight}} \ "\text{regret}".$$
And we saw an algorithm for this: Mult. Weights or Weighted Majority

- Initially weights $W_e = 1$ for $e$ experts/corespondents

  each time:

  $p_i^t = \frac{W_i}{\sum_{j=1}^{n} W_j}$

  "prediction"

  then we have $l^t = (l_1^t, l_2^t, \ldots, l_n^t)$

  set: $W_i^{\text{new}} = W_i^{\text{old}} \cdot (1 - \epsilon l_i^t)$

  to reduce weights of experts with large loss.

  In fact let's change it (remember $1 + x \geq e^x$ for $x$ small)

  $W_i^{\text{new}} = W_i^{\text{old}} \cdot e^{-\epsilon l_i^t}$

  "update"

Note: $W_i^{\text{after (k+1 steps)}} = W_i^{\text{after (k steps)}} \cdot e^{-\epsilon l_i^t}$

$= (W_i^{\text{after (k steps)}} \cdot e^{-\epsilon l_i^t}) \cdot e^{-\epsilon l_i^t}$

$= e^{-\epsilon (l_i^1 + l_i^2 + \ldots + l_i^k)}$

$= 1$, $e$
So the weight at time $t = \exp(-\varepsilon \sum_{i \in t} l_i^s)$

\[ p_t^i = \exp(-\varepsilon \sum_{i \in t} l_i^s) \]

weight exponentially small in total flow until now.

\[ \prod_{t} \exp(-\varepsilon \sum_{i \in t} l_i^s) \]

"exponential weights"

Two observations:

1. These kind of ideas useful for solving:
   - Max flows
   - Linear programs

Here's one example: Max flow.

Given $s-t$ flow network, instead of maintain residual graph, each time push flow along a shortest $s-t$ path.

Shortest edge lengths $= \frac{\text{current flow on edge}}{\text{capacity of edge}}$

Repeat this $O\left(\frac{F \log n}{\varepsilon^2}\right)$ times, sum up all the flow to some number, not important right now.
import networkx as nx

edges = [[0, 1], [1, 0], [0, 9], [0, 2], [2, 3], [3, 4], [4, 9], [1, 5], [5, 6], [6, 7], [7, 9]]
eps = 0.1
T = 2000

G = nx.DiGraph()
G.add_nodes_from(range(0, 10))
for e in edges:
    G.add_edge(e[0], e[1], weight=1, flow=0.0)

# Repeatedly find shortest paths
for t in range(0, T):
    SP = nx.dijkstra_path(G, 0, 9)
    for i in range(len(SP) - 1):
        G[SP[i]][SP[i+1]]['weight'] = G[SP[i]][SP[i+1]]['weight']*(1+eps)
        G[SP[i]][SP[i+1]]['flow'] = G[SP[i]][SP[i+1]]['flow']+1.0

# and rescale
maxi = 0
for u, v in G.edges():
    if maxi < G[u][v]['flow']:
        maxi = G[u][v]['flow']

for u, v in G.edges():
    G[u][v]['flow'] = G[u][v]['flow']/maxi

for u, v, f in G.edges(data='flow'):
    print("edge ", u, v, " flow: ", "%.2f" % f)
But back to business:

What if we did a simpler idea: Look at the past and choose the best action in hindsight (each time).

After all, the best in hindsight (= best expert) is our final yardstick. So maybe best in hindsight at each time is not a bad idea.

**Problem:** this best in hindsight may change each time

E.g. suppose 2 experts,

\[ p' = \left( \frac{1}{2}, \frac{1}{2} \right) \quad \text{and} \quad l' = \left( 0, \frac{1}{2} \right) \]

So we play \[ b = (1, 0) \quad \text{and} \quad l^2 = (1, 0) \]

now we want to play best response to \[ l' + l^2 = (1, \frac{1}{2}) \]

\[ b^3 = (0, 1) \quad \text{and} \quad l^3 = (0, 1) \]

now we want to play best response to \[ l' + l^2 + l^3 = (1, \frac{1}{2}) \]

\[ b^4 = (1, 0) \quad \text{and} \quad l^4 = (1, 0) \]

etc.

We pay 1 each time, but each third expert i pays \[ l_i = 1 \] half the time.
We're "overfitting" to the data at each time.

When at time $t$, the losses would be $\approx \left( \frac{t}{2}, \frac{t}{2} + \frac{1}{2} \right)$ cumulative or $(\frac{t}{2} + \frac{1}{2}, \frac{t}{2})$.

But we choose either left or right.

\[ \xleftarrow{t+2} \quad \text{we flip flop.} \]

\[ \xleftarrow{t-1} \]

\[ \xleftarrow{t} \quad \text{cost fn} \]

Too strong a decision when "data is not conclusive."

**One solution**: Regularization.

Want to throw in a penalty function that prevents us from this hasty movement.

**Suppose**: we try to "maximize our own uncertainty subject to the data. One way is this —

Entropy of a distribution $p \in \Delta_n$ is

\[ H(p) = \sum_{i=1}^{N} p_i \log_2 \left( \frac{1}{p_i} \right) \]  

[Shannon, 48]
**Fact:** \( H(p) \geq 0 \quad \forall p \in \Delta_n \\

(2) For 2dim vector \( p = (a, 1-a) \), the entropy looks like

![Graph showing the entropy function]

\( H((a, 1-a)) \)

\( 0 \quad 1 \quad a \)

**Indeed:**

(3) \( H(p) \leq \log_2 N \)

and this is when \( p = \left( \frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N} \right) \)

"maximum uncertainty about where we are if we choose one coordinate i.e \( \frac{1}{N} \) or \( \frac{1}{N} \)"

(4) Motivation for this definition comes from coding/compression.

- Suppose sending one of \( N \) symbols across a channel.

- Each symbol has a unique encoding \( f(s) \) \( \in \) bits

- Frequency of symbol \( i \in [N] = p_i \)

\[ \Rightarrow \text{Expected length} = \sum p_i f(i) \text{ bits} \]

How to minimize this?
Want infrequent elements to have longer codew.

Can use Huffman codes e.g. to get that elements of frequency $p_i$ have encodings of length $n_i \approx \left\lceil \log \frac{1}{p_i} \right\rceil$

And this is optimal

$\Rightarrow E[\text{length of message in bits}] = \sum_i p_i \log \frac{1}{p_i}$

Coming back to the matter at hand.

Want to choose $\beta^t$ such that

1. good response to history $(l^1, l^2, \ldots, l^{t-1})$

2. maximizes uncertainty.

**Proposal:**

$\beta^t \leftarrow \arg \min_{\beta \in \Delta_N} \left( \sum_{t=1}^{t-1} \left< l^t, \beta \right> + \frac{1}{2^t} H(\beta) \right)$

\[ H(\beta) \equiv \max \text{ negative entropy} \]

Follow the "regularized" leader (FTRL)

$H(\beta) = \sum_i (p_i \ln \frac{1}{p_i} - p_i)$
Theorem: Playing FTRL gives low regret

\[ p_t \leftarrow \text{argmin}_{p \in \Delta_N} \left( \sum_{s \leq t} e_s^s, p \right) - \frac{1}{\varepsilon} H(p) \]

Fact: \( H(p) \) is concave function of \( p \)
\( f(p) \) is linear \( \Rightarrow f(p) - \frac{1}{\varepsilon} H(p) \) is convex

\[ \Rightarrow f(p) - \frac{1}{\varepsilon} H(p) \text{ at point where derivative } = 0. \]
\text{(in higher dims, where gradient } = 0) \]

\[ \nabla f(p) = \sum_{s \leq t} e_s^s, \quad \nabla \langle a, x \rangle = a \]

\[ \nabla H(p) = \left( \frac{\delta H(p)}{\delta p_1}, \ldots, \frac{\delta H(p)}{\delta p_n} \right) \]

\[ \frac{\delta H(p)}{\delta p_i} = \sum_{j} \left( \frac{p_j \ln \frac{p_j}{p_i}}{p_i} - p_i \right) \frac{d}{dp_i} \left( \frac{p_j}{p_i} \right) - \frac{d}{dp_i} p_i \]

\[ \frac{d}{dx} (-x \ln x - x) = -\ln x + x \left( \frac{1}{2} - 1 \right) = -\ln x \]
\[ \Rightarrow \text{if we want} \quad \nabla f(p) = \frac{1}{\varepsilon} \nabla H(p) = 0. \]

\[ \Rightarrow \left( \sum_{s\leq t} l^s \right) + \frac{1}{\varepsilon} \langle \ln p_1, \ln p_2, \ldots, \ln p_n \rangle = 0. \]

\[ \Rightarrow \frac{1}{\varepsilon} \ln p_i = -\sum_{s\leq t} l^s \]

\[ \Rightarrow p_i = e^{-\varepsilon \sum_{s\leq t} l^s} \quad \text{weights in exponential weights!!} \]

\[ \text{Actually if you throw in the constraints that } p \in \Delta_n, \]

get that

\[ p_i = \frac{\exp(-\varepsilon \sum_{s\leq t} l^s)}{\sum_{j} \exp(-\varepsilon \sum_{s\leq t} l^s)} \quad \text{exact same as MW.} \]

\[ \text{just renormalizing the weights to sum to 1.} \]

---

\[ \text{Moral of the story} \]

\[ \text{Follow the Regularized Leader} = \text{Multi. Weights.} \]

\[ \text{Goal: Another use of regularization (this time to spread the mass out) same } L_1 \text{-regularization in Compressive Sensing (to concentrate the mass)} \]
But wait, there is more. 😊

- Why entropy, why not other regularizers?

Say we look at $||p||_2$ - the Euclidean length.

- This is small when $p$ is spread out $||\left(\frac{1}{n}, \cdots, \frac{1}{n}\right)|| = \frac{1}{n} \cdot \sqrt{n} = \frac{1}{\sqrt{n}}$

- And large when $p$ is concentrated $||\left(1, 0, \cdots, 0\right)|| = 1.$

So maybe

$s_t \leftarrow \arg \min_{\mathfrak{p} \in \Delta_n} \frac{1}{2} \sum_{s \in \mathcal{S}} \langle \mathfrak{p}, \mathfrak{s} \rangle + \frac{1}{3} ||\mathfrak{p}||_2^2$  

Note: Want to minimize both terms now.

Now cannot just drop this constraint like before.

So what's to be done?

- Do we really need to keep all of history around?

- What is really happening?

In order to go down this road, it will be helpful to abstract things a bit more.
Online Convex Optimization: -

At each time $t$,

- nature/adversary chooses convex function $f_t : \mathbb{R}^n \to \mathbb{R}$
  
  we don't see it at this time

- we choose $x_t \in \mathbb{R}^n$ inside some body $K \subseteq \mathbb{R}^n$
  
  then both are revealed to each other.

Our loss = $f_t(x_t)$

Want: $\exists x^* \in K$

$$\sum_t f_t(x_t) \leq \sum_t f_t(x^*) + \text{small}$$

↑ our loss  ↓ loss for a generic point  ↑ regret.

When $f_t(x_t) = \langle e_t, x_t \rangle$ and $x_t \in \Delta_n$

⇒ get previous problem.

To come next: can solve this using FTRL/Max Weights

and actually gradient descent!