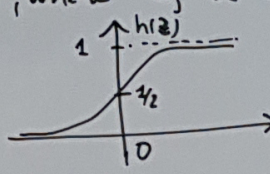
I. Logistic Regression = Regression + Sigmoid

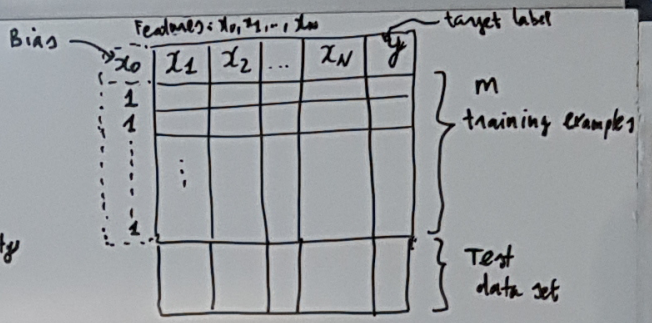
1.1) Regression: $z = w_0 x_0 + w_1 x_2 + \cdots + w_N x_N$, where w_j is weight

1.2) Sigmoid: output $= h(z) = \frac{1}{1+e^{-z}}$

Call "z" is logits



$h(z)$ is output $[0,1]$ with input $(-\infty, +\infty)$ $\Rightarrow h(z)$ is probability



2. Cost Function

$$J(w) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log\left(h(z(w^{(i)}))\right) + \left(1-y^{(i)}\right)\log\left(1-h(z(w^{(i)}))\right)$$

where:
- m: training examples
- $y^{(i)}$: label of "i" training example
- $w^{(i)}$: weight of ↑ ↑ ↑
- $h(z(w^{(i)}))$: prediction of ↑ ↑ ↑

Ex.

Real label $y^{(i)}$	Prediction $h(z(w^{(i)}))$	$\log h(z \mid w^{(i)})$	Loss Function $-1 \times \left(y^{(i)} \log h(z(w^{(i)})) + (1-y^{(i)})(1-\log(1-h(z(w^{(i)})))\right)$	
1	~ 1	0	0	GOOD
1	~ 0	large negative	large positive	BAD
0	~ 0	0	0	GOOD
0	~ 1	large negative	large positive	BAD

3. Update weights (w: $w_0, w_1, \cdots, w_j, \cdots, w_N$)

- $\nabla_{w_j} J(w) = \frac{1}{m}\sum_{i=1}^{m}\left(h^{(i)} - y^{(i)}\right) x_j$

  where x_j: feature corresponding to weight w_j
- $w_j = w_j - \alpha \times \nabla_{w_j} J(w)$

  where α: learning rate

  $\nabla_{w_j} J(w)$: gradient
- Implement Gradient Descent Function

For num_iter in num_iters: # num_iter run for all training examples

- Calculate Cost Function $J(w)$ for all training examples and all features
- Update weights in colum vector to speed up → $w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_N \end{bmatrix}_{(N+1,1)}$ ← Bias term and $x_0 = 1$ for all training examples
- $z^{(i)} = w_0^{(i)} x_0 + w_1^{(i)} x_1 + \cdots + w_N^{(i)} x_N$ (for one training examples)

  → $z = X_{(m,N+1)} \; w_{(N+1,1)} \Rightarrow z_{(m,1)}$

  Note $Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}_{(m \times 1)}$ ... training example 1
- $h(z) = \text{sigmoid}(z) \Rightarrow h(z)_{(m,1)}$
- $J(w) = -\frac{1}{m} \times \left(Y^T \cdot \log(h(z)) + (1-Y)^T \cdot \log(1-h(z))\right)$ ← a number
- $W = W - \frac{\alpha}{m} \times \left(X^T \cdot (h(z) - Y)\right)$

Python: def train(X, y, W, α, num_iters): return W, J ← a.s.k.a epoches

II. Sentiment Analysis: Tweet has 2 label (positive or negative)

1) one tweet → process (tokenize, stopwords, stemming) → GET words (a list of words)
2) freqs dictionary: {("happy", 1): 999, # 1 is for positive, ("happy", 0): 88, # 0 ↑ ↑ negative} (DB has been built)
3) Extract features for 1 training example (1 tweet)

(i) $X =$

x_0	x_1	x_2
1	↑	

→ "negative"

Fixed 1 for bias — number of times, the tweet words are positive

```
x = np.zeros((1,3))
x[0,0] = 1
for word in words:
    x[0,1] += freqs.get((word, 1), 0)   # if true get number from freqs, else get 0
    x[0,2] += freqs.get((word, 0), 0)
```

Training:

Tweets	y
T_1	0
T_2	1
T_3	1
T_4	1
T_5	0
...	...

→ From Tweet → make freqs dictionary → then extract features

$X_{train} =$

x_0	x_1	x_2	From
1	99	200	T_1
1	200	10	T_2
1	600	300	T_3
...	...	...	...

W: need initialize

α: a number. ex: 10^{-8}

num_iters: a number. ex: 1500

Python: def predict_tweet(tweet, freqs, W):
- tweet → get features X
- $z = x \cdot W$
- $y_{pred} = h(z)$: if $y_{pred} > 0.5$: positive else: negative