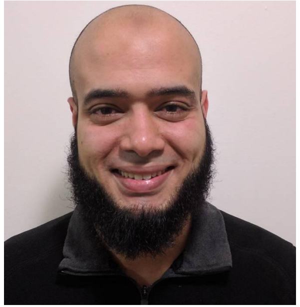
Machine Learning
Softmax Function

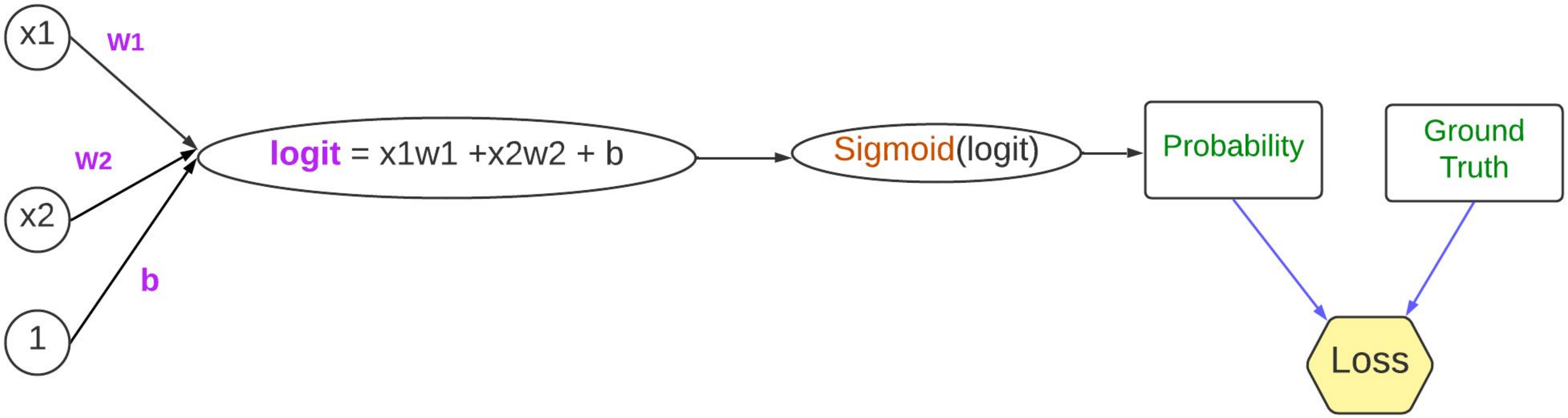
Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



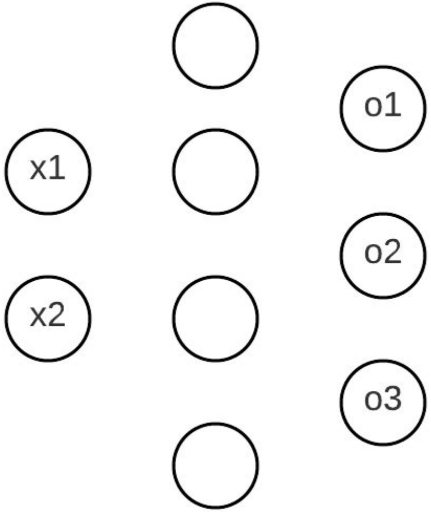
From A Single Node to Multi-Node

- When we have a single logit node, we just converted to a probability output with sigmoid and used logloss with the ground truth (either 0 or 1)
  - 4 critical components: **logits** ⇒ **Activation** to map probability ⇒ **GT** ⇒ **Loss** function
- What if we generated N logits for N classes!
  - We need to answer the 3 things: activation, gt and loss



Issue #1: Target Output

- Assume our network ends with 3 output nodes (**logit** values)
  - Recall logits are non-normalized values (+ve, -ve)
- We are learning 3 **independent** classes: dog, cat and horse for an input image
- In binary class, we learned a single ground truth 0 or 1
- What should be our target ground truth for the 3 values?



- Also a probability distribution like a one-hot encoding
  - Assume o1 for dog, o2 for cat and o3 for horse
  - Assume the actual class is cat
  - Then ground truth are: [0, 1, 0]
    - Notice, it **sums to one**

Issue #2: Activation for Probability Distribution

- For simplicity, assume our logits are >= 0
  - For example: [5, 7, 8] is our logits
- Propose a simple function that can convert the logits to a probability distribution (e.g. sum = 1)
  - Intuitively, higher logits should corresponds to higher probability, as in sigmoid

- Just divide each number by the sum of the values
- Can you recognize an issue with the below 2 examples?
- What about negative logits?

```
def sum_activate_v1(x):
    return x / x.sum()

sum_activate_v1(np.array([0, 2, 3]))      # [0.   0.4 0.6]
sum_activate_v1(np.array([5, 7, 8]))      # [0.25 0.35 0.4 ]
```

Sum Function for Transformation

- One clear issue, the gaps between the values are the same. It is more initiative actually to generate the same output!
  - [5, 7, 8] - 5 = [0, 2, 3]
- Another issue: it fails for negative values!
- Find a modification that can solve both problems!

- Just subtract the minimum!
- Can you identify training problems?

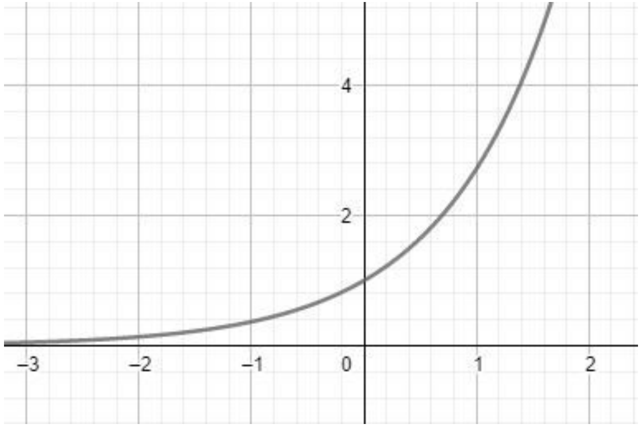
```python
def sum_activate_v2(x):
    x = x - x.min()
    return x / x.sum()


sum_activate_v2(np.array([0, 2, 3]))     # [0.  0.4 0.6]
sum_activate_v2(np.array([5, 7, 8]))     # [0.  0.4 0.6]
sum_activate_v2(np.array([-2, 0, 1]))    # [0.  0.4 0.6]
```

Sum Function for Transformation

- Not fully differentiable function due to the min function.
  - However, we may handle it with e.g. sub-gradient
- Logits insensitivity
  - Compare to strong non-linear transformations, the function could be **less sensitive** to differences between in logits: E.g. [5, 7, 8] vs [5, 7, 8.2]
    - From one side, this may not be logically desirable
    - From backpropagation perspective, gradients may be less informative leading to slower or less stable convergence

Softmax Function

- Sotmax is a *generalization of sigmoid* where:
  - Input: vector of real values (e.g. logits for N **independent** classes)
  - Output: probability (*categorical*) distribution for N values
- It is the same as the sum function, however, replace each Xi with exp(Xi)
  - Clearly in **range [0-1]**, regardless logits range and also **smooth** / **differentiable**



$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Softmax Function

- Observe that the function is already invariant to constant shifts

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

```python
def softmax(x):
    return np.exp(x) / np.sum(np.exp(x))

softmax(np.array([0, 2, 3]))      # [0.03511903 0.25949646 0.70538451]
softmax(np.array([5, 7, 8]))      # [0.03511903 0.25949646 0.70538451]
```

Softmax Function

X[i]	exp(X[i])	sum: exp(X)	Softmax[i]
1.5	4.48168907	197.953654	0.022
5.2	181.272241		0.915
2.3	9.97418245		0.050
0.8	2.22554093		0.011

Softmax Function: Logits Sensitivity

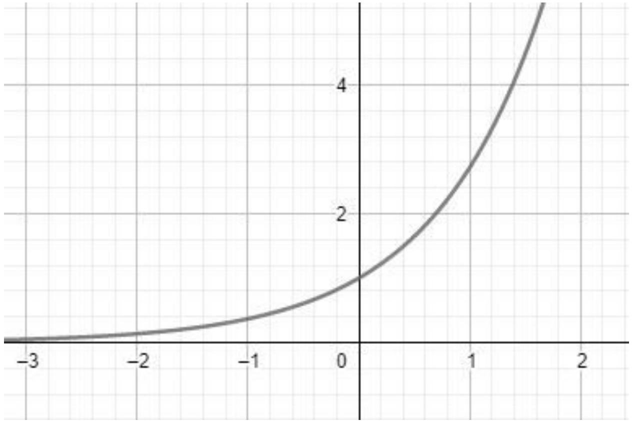
- Given that our target distribution is like one-hot-encoding, softmax cause **amplification of differences** that emphasizes larger values over smaller ones and makes the model more confident about its predictions toward the target class

X	Softmax(X)
[0, 2, 3]	[0.035 0.259 **0.70**5]
[0, 2, 3.2]	[0.0303 0.224 **0.74**5]
[0, 2, 3.5]	[0.0240 0.178 **0.79**7]
[0, 2, 3.8]	[0.0188 0.1391 **0.84**1]
[0, 2, 5]	[0.006 0.0471 **0.94**6]
[0, 2, 6]	[0.002 0.017 **0.97**9]
[0, 2, 7]	[0 0 0.**99**]

The Max Trick

- The "max trick" is a **numerical stability** trick to compute softmax function
  - exp(x) will overflow for large values
- Simply, subtract x.max() first from all values
  - Hence max logit = 0 and other values are < 0
- Prove: we can prove adding any constant to all values of X doesn't change the softmax output
  - Write the proof

```python
def softmax(x):
    x = x - x.max()
    return np.exp(x) / np.sum(np.exp(x))
```



Softmax Function Cons

- **Sensitive to Outliers**: exp(x) will enlarge large logits values
- **Lack of Calibration**: Predicted probabilities doesn't reflect true distribution due to the amplification of differences (ending close to one-hot-encoding)
- **Computational Overhead**: Exponentiation and division
- **Suboptimal for Imbalanced Classes**: It assumes equal weights for classes

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Softmax Derivative

- The input of softmax is N values (x1, x2, …. xn)
- The output is N values (s1, s2, ….sn)
  - Each output depends on all the input values (as the denominator sums over all inputs)
- How many partial derivatives that we need between input and output?
- N^2: one partial derivative between an input symbol and output symbol
- When we arrange them in a matrix, we call it Jacobian matrix
  - In general, it is an nxm matrix of all **first-order partial** derivatives (n functions and m variables)
  - J[i][j] = ∂ S[i] / ∂ X[j]   where S[i] is the ith output from the softmax output vector
- For example for N = 3, J is

∂S1/ ∂X1	∂S1/ ∂X2	∂S1/ ∂X3
∂S2/ ∂X1	∂S2/ ∂X2	∂S2/ ∂X3
∂S3/ ∂X1	∂S3/ ∂X2	∂S3/ ∂X3

Softmax Derivative

- With some math, we can compute the partial derivative ∂S[i] / ∂X[j]
- We will end with 2 cases: one for the diagonal and one for non-diagonal
  - Can you see the relationship to **sigmoid derivative**?!

$$\frac{\partial s(x)_i}{\partial x_j} = \begin{cases} s(x)_i(1 - s(x)_i) & \text{if } i = j \\ -s(x)_i s(x)_j & \text{if } i \neq j \end{cases}$$

- Hence the Jacobian matrix is:

S1 x (1-S1)	-S1 x S2	-S1 x S3
-S2 x S1	S2 x (1-S2)	-S2 x S3
-S3 x S1	-S3 x S2	S3 x (1-S3)

Hard and Soft Functions

- Hard functions are not fully differentiable
  - max(x) return the maximum value
  - argmax(x) return the index of the maximum value
  - min(x) return the minimum value
- To make training more stable, we tend to find a soft version that is differentiable and can approximate the answer
  - So soft implies smooth and differentiable
- Hence
  - Softmax: return the soft maximum value of an array
  - Soft-argmax: return the soft (expected) index of the maximum value of an array
- What is confusing now?!

Hard and Soft Functions

- Our softmax function actually returns a one-hot-like encoding where the index of the maximum value is close to 1
- This makes it more useful for soft-argmax not softmax
- As a result, people consider the name softmax a misleading one
- However, it is a common convention nowadays
- A challenge: implement soft_argmax(x) that returns the soft index of the maximum value
    - Common usage in deep learning where we need the argmax itself

Soft-ArgMax

- Below is a simple potential implementation for soft-argmax
  - Assume the input is as follows: [2, 4, 18, 3]
  - Which has softmax as follows: [0 0 0.99 0]
  - Imagine we computed indices sequence: [0, 1, 2, 3]
  - Now multiply both: the zeros will just cancel all non-max indices
  - The remaining index will be multiplied with value closer to 1 (in the best case)

```python
def soft_argmax(x):
    return np.sum(softmax(x) * range(x.size))


float_idx = soft_argmax(np.array([2, 4, 18, 3]))
print(int(round(float_idx)))    # 2
```

$$\mathrm{SoftArgMax}(x) = \sum_{i}^{N} i \times \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

Log Softmax

- There is something very interesting about the log of softmax

$$S(\mathbf{x})_j = \frac{\exp(x_j)}{\sum_{k=1}^{C} \exp(x_k)} \quad \text{for } j = 1, \ldots, C$$

$$\log(S(\mathbf{x}))_j = \log\left(\frac{\exp(x_j)}{\sum_{k=1}^{C} \exp(x_k)}\right)$$

$$\log(S(\mathbf{x}))_j = \log(\exp(x_j)) - \log\left(\sum_{k=1}^{C} \exp(x_k)\right) \quad = x_j - \log\left(\sum_{k=1}^{C} \exp(x_k)\right)$$

Log Softmax

$$= x_j - \log\left(\sum_{k=1}^{C} \exp(x_k)\right)$$

- What is that 2nd term?
  - This is what historically/properly called a softmax
    - But the term nodays used for the activation function!
  - It is a way get the maximum value in a softway (hence differentiable)
  - Hence, this term approximates max(x)
- Why max?
  - When one $\exp(X_k)$ is **much larger** than the others, $\exp(X_k)$ becomes the dominant term in the sum.
  - When taking the **logarithm**, the effect of the **smaller** terms becomes **negligible**, making the log summation close to max(x)

Log Softmax

- So the equation in fact can be approximated with **x - max(x)**
  - No exp(). No log()
  - But it is an approximation (good if max(x) is far from others)
  - Tradeoff: approximation (for numerical stability / efficiency) vs precision
- Tip: log probabilities are favoured in ML than probabilities
  - E.g. Addition vs multiplication in maximum likelihood estimation

$$= x_j - \log\left(\sum_{k=1}^{C} \exp(x_k)\right)$$

Log Softmax

```python
def logsoftmax1(x):
    x = x - x.max()
    return np.log(np.exp(x) / np.sum(np.exp(x)))

def logsoftmax2(x):
    return x - x.max()

if __name__ == '__main__':
    x = np.array([1, 2, 3, 10])
    # [-9.0014 -8.0014 -7.0014 -0.0014]
    logsoftmax1(x)
    # [-9        -8        -7        0]
    logsoftmax2(x)
```

Next!

- We learned a lot of wonderful things about softmax function
- To understand why it is that **deeply** used in the **deep** learning community, we need to consider the last piece: **the loss function**!
- Note
  - In the homework, you will prove several properties we mentioned today

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”