Model Training Steps

Tensor Flow

	logistic regression	neural network
① specify how to compute output given input x and parameters w,b (define model) $f_{\vec{w},b}(\vec{x}) = ?$	`z = np.dot(w,x)+ b` `f_x = 1/(1+np.exp(-z))`	`model = Sequential([ Dense(...) Dense(...) Dense(...) ])`
② specify loss and cost $L(f_{\vec{w},b}(\vec{x}), y)$ 1 example $J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$	logistic loss `loss = -y * np.log(f_x) -(1-y) * np.log(1-f_x)`	binary cross entropy `model.compile( loss=BinaryCrossentropy())`
③ Train on data to minimize $J(\vec{w}, b)$	`w = w - alpha * dj_dw` `b = b - alpha * dj_db`	`model.fit(X,y,epochs=100)`

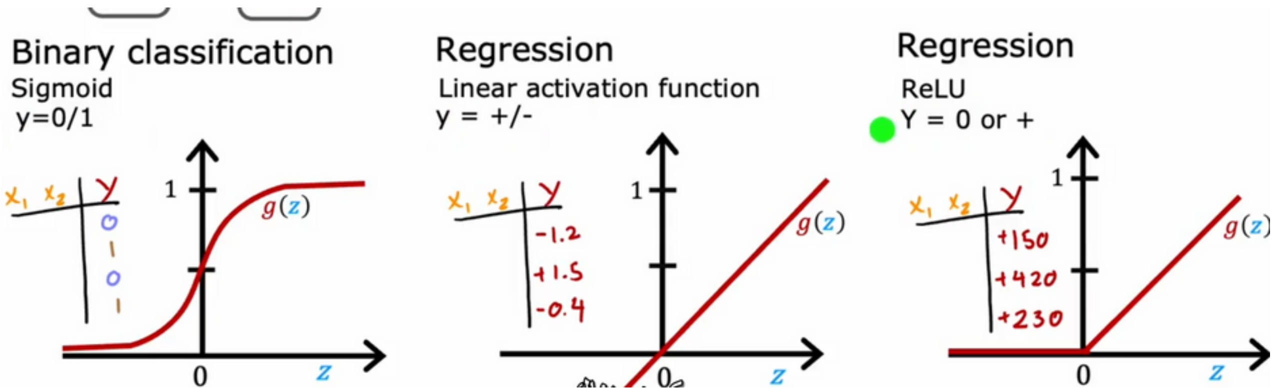
Neural network libraries

Use code libraries instead of coding "from scratch"

TensorFlow PyTorch

Choosing activation functions

1. **Sigmoid Function (Logistic):**
   - Range: (0, 1)
   - Advantages: Smooth gradient, output bound between 0 and 1, historically used in older architectures.
   - Disadvantages: Vanishing gradient problem (gradients approaching zero for extreme inputs), not zero-centered.
2. **Hyperbolic Tangent (tanh):**
   - Range: (-1, 1)
   - Advantages: Similar to sigmoid but zero-centered, sometimes used in hidden layers.
   - Disadvantages: Still suffers from vanishing gradient problem.
3. **Rectified Linear Unit (ReLU):**
   - Range: [0, +∞)
   - Advantages: Fast convergence due to non-vanishing gradient for positive inputs, computationally efficient.
   - Disadvantages: "Dying ReLU" problem (neurons can get stuck during training if their output is always zero).
4. **Leaky ReLU:**
   - Range: (-∞, +∞)
   - Advantages: Addresses the "dying ReLU" problem by allowing a small gradient for negative inputs.
   - Disadvantages: Gradient might still be too small for extremely negative inputs.



Choosing $g(z)$ for hidden layer

Softmax regression (4 possible outputs) $y = 1, 2, 3, 4$

× $z_1 = \vec{w}_1 \cdot \vec{x} + b_1$

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

× ○ □ △

$$= P(y = 1|\vec{x})$$

○ $z_2 = \vec{w}_2 \cdot \vec{x} + b_2$

$$a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

$$= P(y = 2|\vec{x})$$

□ $z_3 = \vec{w}_3 \cdot \vec{x} + b_3$

$$a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

$$= P(y = 3|\vec{x})$$

△ $z_4 = \vec{w}_4 \cdot \vec{x} + b_4$

$$a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

$$= P(y = 4|\vec{x})$$

Softmax regression (N possible outputs) $y=1,2,3,\ldots,N$

$$z_j = \vec{w}_j \cdot \vec{x} + b_j \quad j = 1, \ldots, N$$

parameters $w_1, w_2, \ldots, w_N$
$b_1, b_2, \ldots, b_N$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y = j|\vec{x})$$

note: $a_1 + a_2 + \ldots + a_N = 1$

Cost

Logistic regression

$$z = \vec{w} \cdot \vec{x} + b$$

$$a_1 = g(z) = \frac{1}{1+e^{-z}} \quad = P(y=1|\vec{x})$$

$$a_2 = \underbrace{1 - a_1} \quad = P(y=0|\vec{x})$$

$$loss = -y \underbrace{\log a_1}_{\text{if } y=1} - (1-y)\underbrace{\log(1-a_1)}_{\text{if } y=0}$$

a_2

$$J(\vec{w}, b) = \text{average loss}$$

Softmax regression

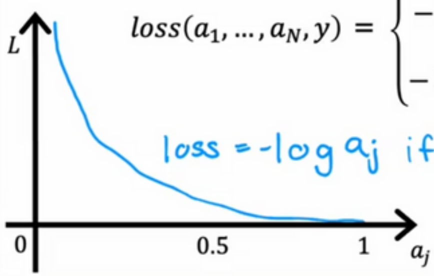
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} \quad = P(y=1|\vec{x})$$

$$\vdots$$

$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} \quad = P(y=N|\vec{x})$$

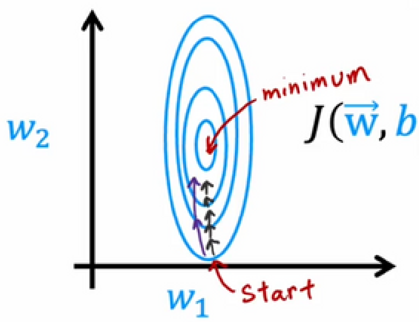
Crossentropy loss

$$loss(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y = 1 \\ -\log a_2 & \text{if } y = 2 \\ \vdots \\ -\log a_N & \text{if } y = N \end{cases}$$

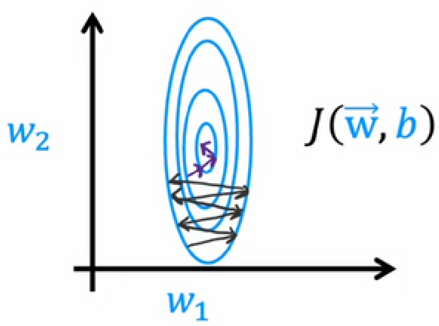


loss $= -\log a_j$ if $y = j$

Adam Algorithm Intuition



If w_j (or b) keeps moving in same direction, increase α_j.



If w_j (or b) keeps oscillating, reduce α_j.