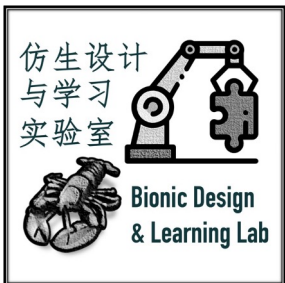


Lecture 05

Machine Learning II

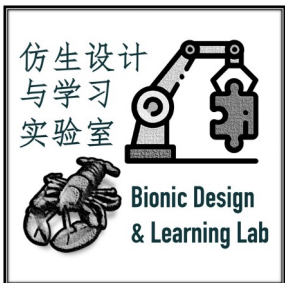


AncoraSIR.com

[Please refer to the course website for copyright credits]



Statistical Binary Classification



AncoraSIR.com



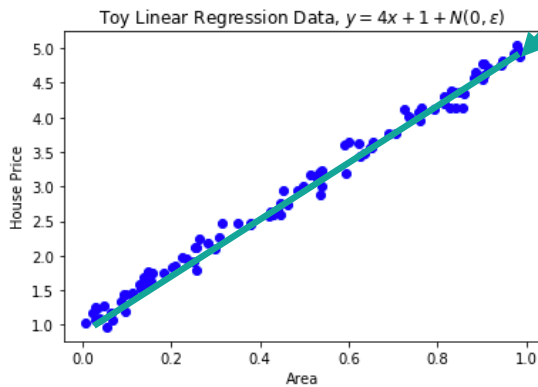
Statistical Binary Classification

To categorize new probabilistic observations into two predefined categories

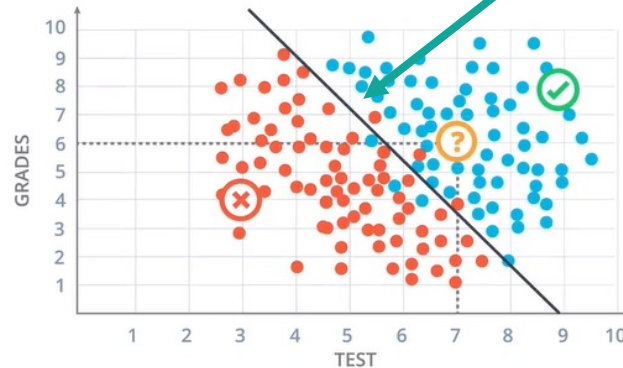
- Linear Regression

- A basic linear model for line-fitting
- $\hat{y} = f_{weightedSum}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$

Estimate a line to describe a relationship



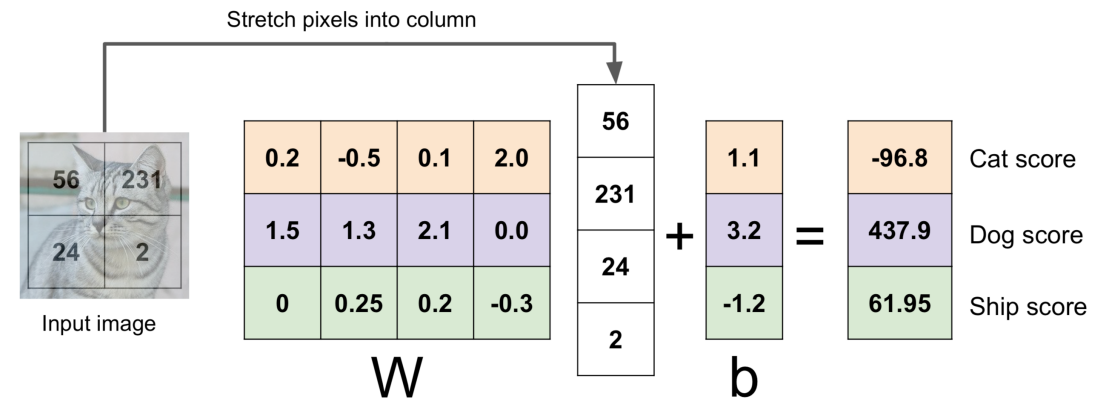
Predict University Acceptance based on Test and Grades (only two categories)



- Linear Classification

- Vectorized weights for two or multiple classes
- $\mathbf{s} = f_{weightedSum}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$

Is this picture a cat, a dog, or a ship?
(Can we make a decision based on the results on the right?)



- What if the problem becomes more complex?

$$\hat{y} = g_{Activation}(s) = \begin{cases} 1 & \text{if } s \geq 0 \\ 0 & \text{if } s < 0 \end{cases}$$

- Information lost about the distance to the cutoff value
- Uncertain about the final decision

Perceptron with an Activation Function

Non-linear extraction of information from the data to help making a decision

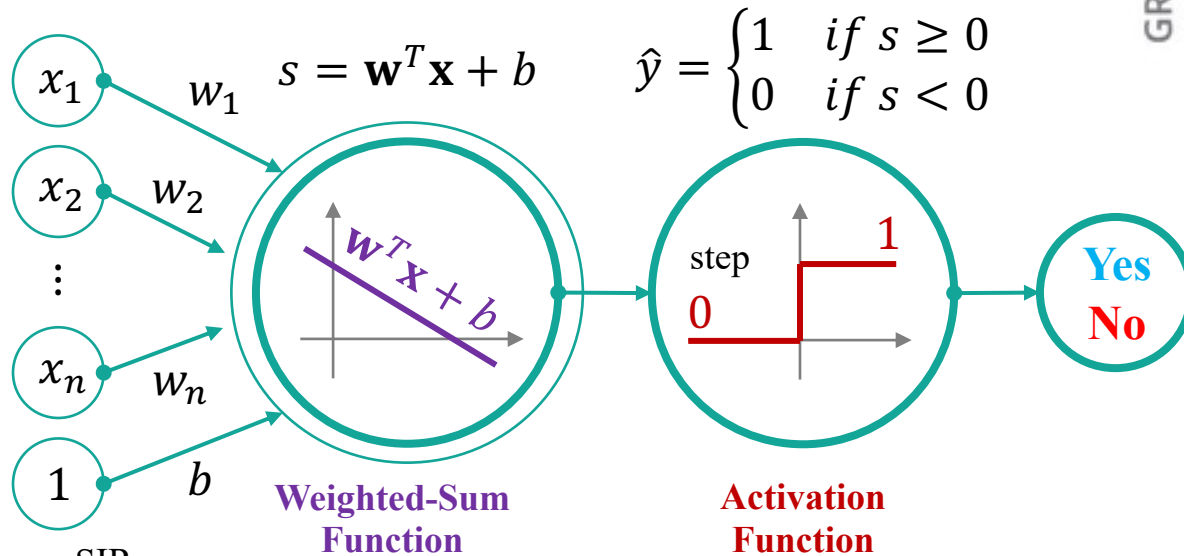
- An Artificial Neuron with two nodes

- **Weighted-sum node**

- Calculate a linear equation $s(\mathbf{x})$ with inputs on the weights plus bias

- **Activation node**

- Apply the step function to get the predicted result $\hat{y}(s)$



AncoraSIR.com

An example of acceptance at a University



$$\begin{aligned}
 \hat{y} &= g_{\text{Activation}}[f_{\text{WeightedSum}}(\mathbf{x})] \\
 &= \text{step}(s, 0) \\
 &= \text{step}(\mathbf{w}^T \mathbf{x} + b, 0)
 \end{aligned}$$

Perceptron Algorithm

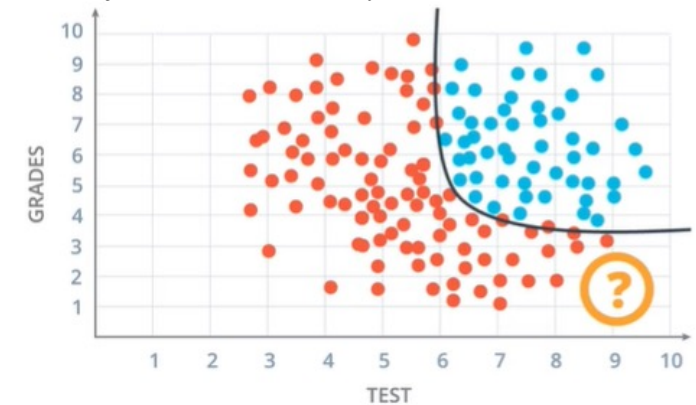
How to get the weights?

- Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$ and initialize t to 1.
 - Let's automatically scale all examples \mathbf{x} to have (Euclidean) length 1, since this doesn't affect which side of the plane they are on.
- Given an example \mathbf{x} , predict positive if and only if $\mathbf{w}_t \cdot \mathbf{x} > 0$.
 - One may consider the bias term b as a weight w_0 for $x_0 = 1$
- On a mistake, update as follows until convergence criteria reached:
 - If mistake on a positive \mathbf{x} , then $\mathbf{w}_t + 1 \leftarrow \mathbf{w}_t + \mathbf{x}$,
 - So $\mathbf{w}_{t+1} \cdot \mathbf{x} = (\mathbf{w}_t + \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} + 1$,
 - *We move closer by 1 to the value we wanted.*
 - If mistake on a negative \mathbf{x} , then $\mathbf{w}_t + 1 \leftarrow \mathbf{w}_t - \mathbf{x}$,
 - So $\mathbf{w}_{t+1} \cdot \mathbf{x} = (\mathbf{w}_t - \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} - 1$,
 - *We move closer by 1 to the value we wanted.*
- $t \leftarrow t + 1$.

If data is separable by a large margin, then Perceptron is a good algorithm to use.



What if the boundary line is non-linear?



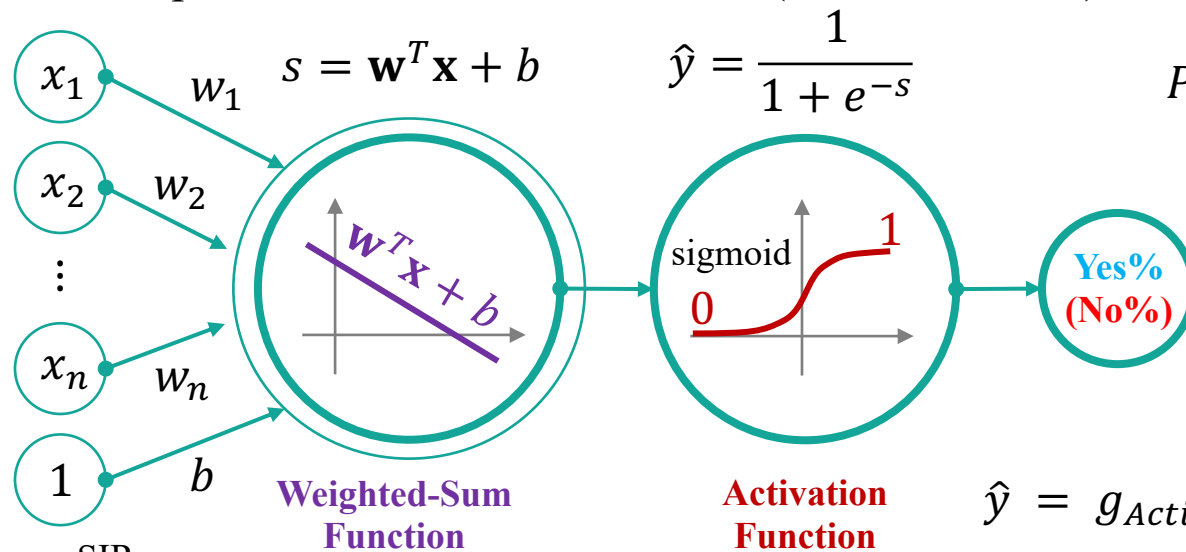
By what chances will I get accepted to a University?

Based on my Test and Grade scores ...

- **Weighted-sum node**
 - Unchanged as the input data remains the same
- **Activation node**
 - Can be changed as we want a new expression of the output as a probability of prediction
 - Sigmoid function as a natural choice that transforms the output to a value between 0 and 1 (or within 100%)



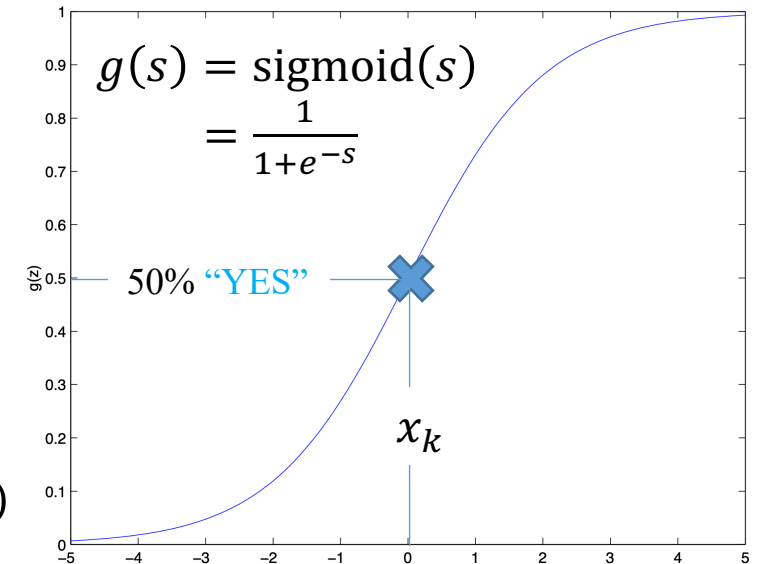
An example of acceptance at a University



$$P(\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}}$$

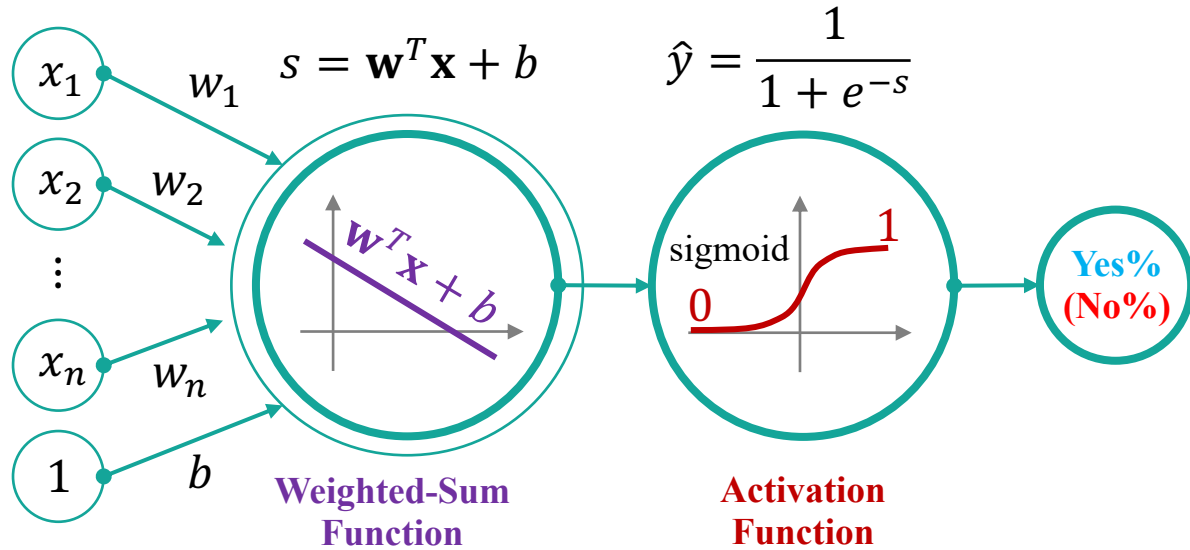
Can be viewed as a probability

$$\hat{y} = g_{Activation}(s) = \text{sigmoid}(s)$$



Logistic Regression

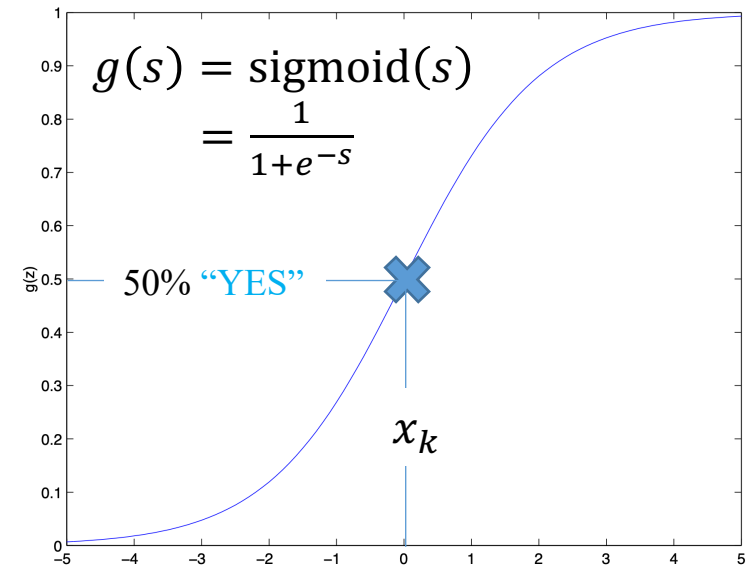
$$\hat{y} = g_{Activation}[f_{WeightedSum}(\mathbf{x})] = \text{sigmoid}(\mathbf{w}^T \mathbf{x} + b)$$



Problem statement

- Assume $\hat{y} = g_{Activation}[f_{WeightedSum}(\mathbf{x})] = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}}$
- How to minimize the **prediction error/loss** on a single training sample (with a maximum likelihood set of \mathbf{w}) ?

- Hypothesis Function: $h_{\mathbf{w}}(x) = \text{sigmoid}(\mathbf{w}^T \mathbf{x} + b)$
- Model output with a probability: $P(y | x; \mathbf{w}) = [h_{\mathbf{w}}(x)]^y [1 - h_{\mathbf{w}}(x)]^{1-y}$
 - Yes%:** $P(y = 1 | x; \mathbf{w}) = h_{\mathbf{w}}(x)$
 - No%:** $P(y = 0 | x; \mathbf{w}) = 1 - h_{\mathbf{w}}(x)$



Loss Function for Logistic Regression

It measures how well you are doing on a single training example

- Assume that m training examples were generated independently $h_w(x) = \text{sigmoid}(w^T x + b)$
- We can write the likelihood of the parameters

$$\begin{aligned} L(w) &= p(\vec{y} | X; w) \\ &= \prod_{i=1}^m p(y^{(i)} | x^{(i)}; w) \\ &= \prod_{i=1}^m [h_w(x^{(i)})]^{y^{(i)}} [1 - h_w(x^{(i)})]^{1-y^{(i)}} \end{aligned}$$

$$\begin{aligned} g'_{\text{activation}}(s) &= \frac{d}{ds} \frac{1}{1+e^{-s}} \\ &= \frac{1}{(1+e^{-s})^2} \cdot e^{-s} \\ &= \frac{1}{(1+e^{-s})^2} \cdot \left(1 - \frac{1}{1+e^{-s}}\right) \\ &= g(s) \cdot (1 - g(s)) \end{aligned}$$

- Take the log expression, we have the **loss function**

$$\begin{aligned} \ell(w) &= \log L(w) \\ &= \sum_{i=1}^m y^{(i)} \log h_w(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_w(x^{(i)})) \end{aligned}$$

- Usually take a “-” sign to indicate loss

Stochastic Gradient Descent

Finding the maximum likelihood of estimation

- Rewrite the weight parameters in vectorized form
 - $w := w + \alpha \cdot \nabla_w \cdot \ell(w)$
 - + sign here to **maximize** likelihood
- When working with a single training example (x, y) ,
 - $\frac{\partial}{\partial w_j} \ell(w) = \left(y \frac{1}{g(w^T x)} - (1 - y) \frac{1}{1 - g(w^T x)} \right) \frac{\partial}{\partial w_j} g(w^T x) = (y - h_w(x)) x_j$
- Therefore, we can derive the stochastic gradient ascent rule
 - $w_j := w_j + \alpha \left(y^{(i)} - h_w(x^{(i)}) \right) x_j^{(i)}$

Cost Function

It measures how well you are doing on an entire training set

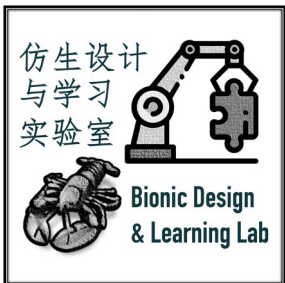
- We want the loss/error function to be as small as possible
 - If $y^{(i)} = 1$, then
 - $\text{LossFunc}(\hat{y}, y) = -\left[y^{(i)} \log h_w(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_w(x^{(i)}))\right] = -\log h_w(x^{(i)}) = -\log \hat{y}$
 - It means that we want $\log \hat{y}$ to be as big as possible, but remember that it is bounded by 1
 - If $y^{(i)} = 0$, then
 - $\text{LossFunc}(\hat{y}, y) = -\left[y^{(i)} \log h_w(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_w(x^{(i)}))\right] = -\log(1 - \hat{y})$
 - It means that we want $\log \hat{y}$ to be as small as possible, or close to 0
- Cost Function
 - The average of the loss functions of the entire training set, which is to be minimized

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Summary

	Linear Regression	Perceptron	Logistic Regression
Problem	Value Prediction	Binary Classification with a threshold	Binary Classification with a probability
Weighted-Sum	$w^T \mathbf{x} + b$	$w^T \mathbf{x} + b$	$w^T \mathbf{x} + b$
Activation Function	NA	Step Function	Sigmoid Function
Prediction Outputs	Continuous Value	Discrete Value $\{0, 1\}$	Continuous Probability $(0, 1)$
Loss	Squared Loss	Hinge Loss	Log-Loss

Multi-class Classification

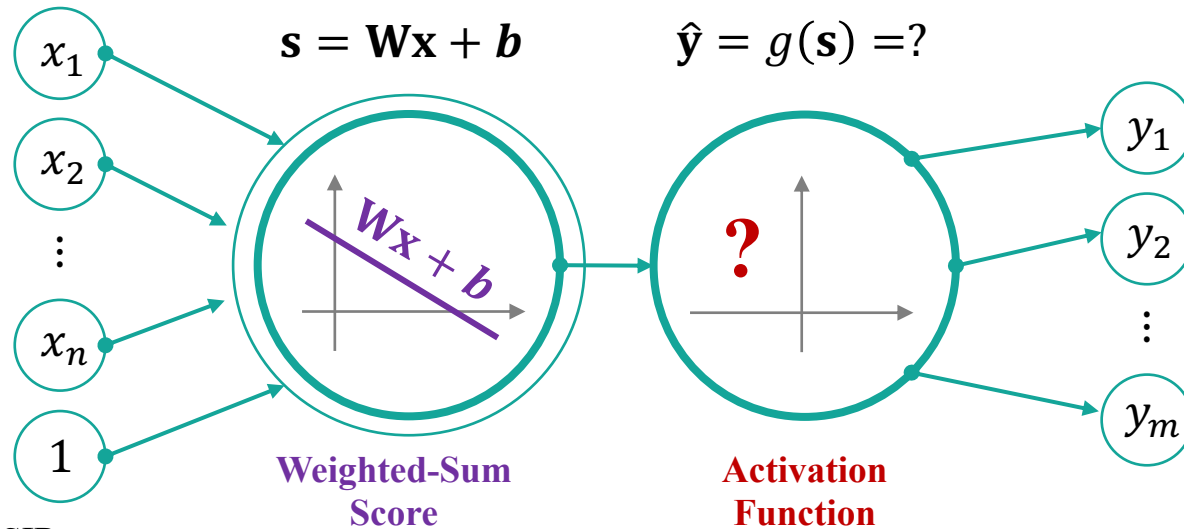
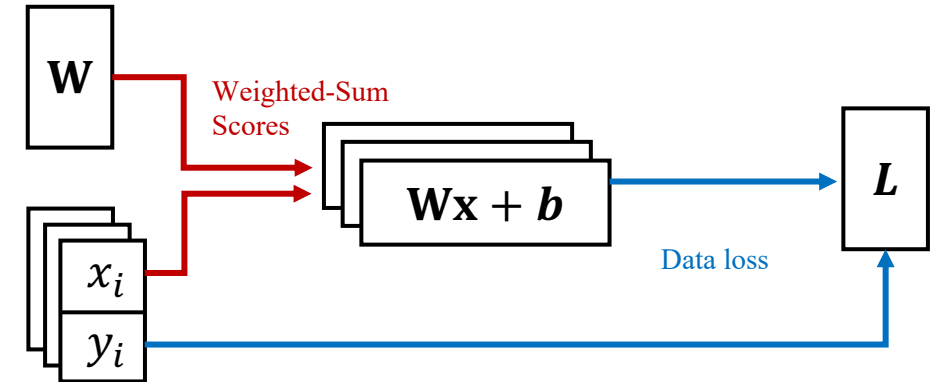
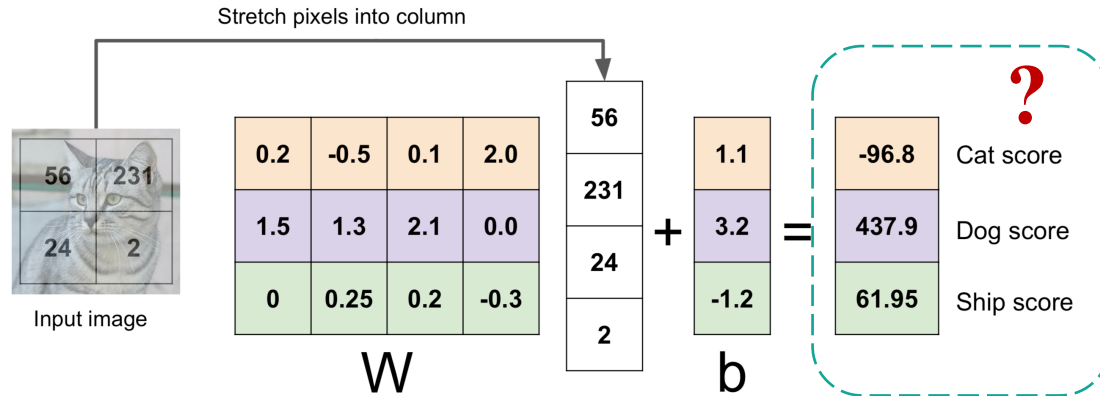


AncoraSIR.com



Multi-class Classification

$$\hat{y} = g_{Activation} [f_{WeightedSum}(\mathbf{x})] = g_{Activation}(\mathbf{W}\mathbf{x} + \mathbf{b})$$



1. **Define a loss function** that quantifies our unhappiness with the scores across the training data.
2. Come up with a way of efficiently finding the parameters that minimize the loss function. (*optimization*)

Define a Loss Function


Quantify how good our current classifier is

3 training samples

$\{(x_i, y_i)\}_{i=1}^3$

Ground Truth

Labelled Prediction \hat{y}_i



x_i image

y_i label

Loss over the dataset is a sum of loss over examples

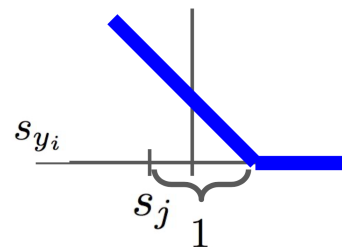
$$L = \frac{1}{N} \sum_i L_i(\hat{y}_i, y_i)$$

Denote Weighted-Sum score vector as $\mathbf{s} = f_{WeightedSum}(\mathbf{x})$

Let's try with the hinge loss:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



3 classes	Cat	3.2	1.3	2.2
	Car	5.1	4.9	2.5
	Frog	-1.7	2.0	-3.1

Define a Loss Function

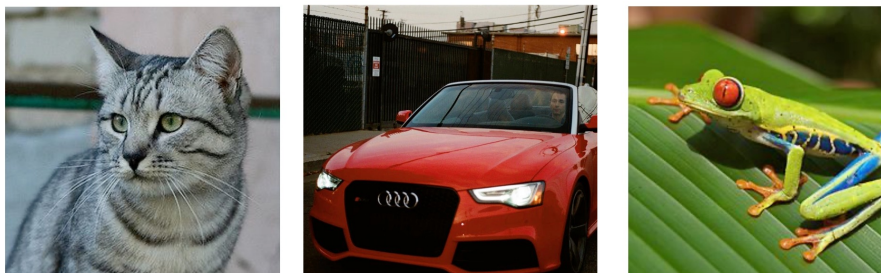
Quantify how good our current classifier is

3 training samples

$\{(x_i, y_i)\}_{i=1}^3$

Ground Truth

Labelled Prediction



3 classes

Cat	3.2	1.3	2.2
Car	5.1	4.9	2.5
Frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$\begin{aligned} L_1 &= \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 + 0 = 2.9 \end{aligned}$$

$$\begin{aligned} L_2 &= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1) \\ &= \max(0, -2.6) + \max(0, -1.9) \\ &= 0 + 0 = 0 \end{aligned}$$

$$\begin{aligned} L_3 &= \max(0, 2.2 + 3.1 - 1) + \max(0, 2.5 + 3.1 - 1) \\ &= \max(0, 6.3) + \max(0, 6.6) \\ &= 6.3 + 6.6 = 12.9 \end{aligned}$$

Define a Loss Function

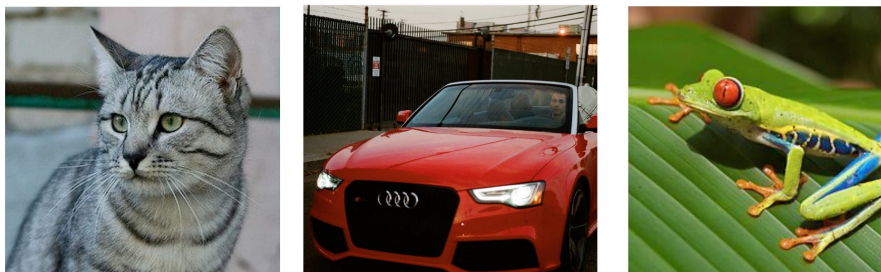
Quantify how good our current classifier is

3 training samples

$\{(x_i, y_i)\}_{i=1}^3$

Ground Truth

Labelled Prediction



3 classes

Cat	3.2	1.3	2.2
Car	5.1	4.9	2.5
Frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

Loss over full dataset is average:

$$\begin{aligned} L &= \frac{1}{N} \sum_i L_i(\hat{y}_i, y_i) \\ &= \frac{1}{3} (2.9 + 0 + 12.9) \\ &= 5.27 \end{aligned}$$

Recall that our goal is to find a set of \mathbf{W} with minimum loss over full dataset, i.e. the cost = 0

- Suppose that we found a \mathbf{W} such that $L = 0$. Is this \mathbf{W} unique?
 - L is still 0 with $2\mathbf{W}$
- **How do we choose between \mathbf{W} and $2\mathbf{W}$?**
 - Let's try regularization

Regularization

Prevent the model from doing too well on training data

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(\hat{y}_i, y_i) + \lambda R(W)$$

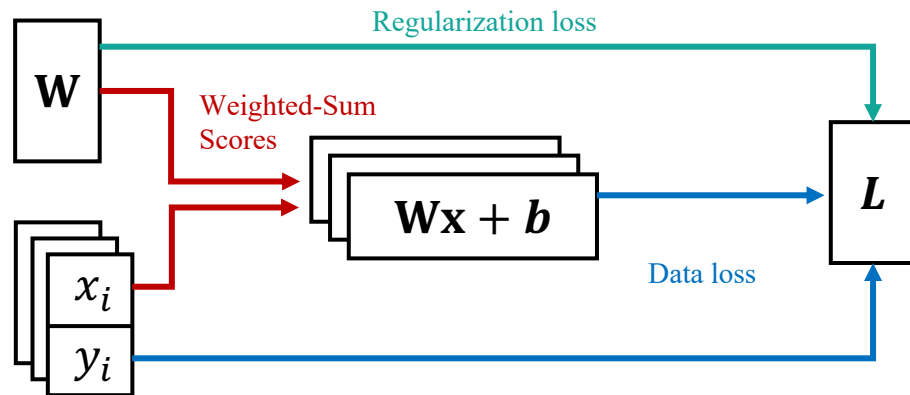
λ as strength of
Regularization
(hyperparameter)

Data loss

Regularization

Model predictions should
match training data

Prevent the model from doing
too well on training data



Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout

Batch normalization

Stochastic depth, fractional pooling, etc

Why regularize?

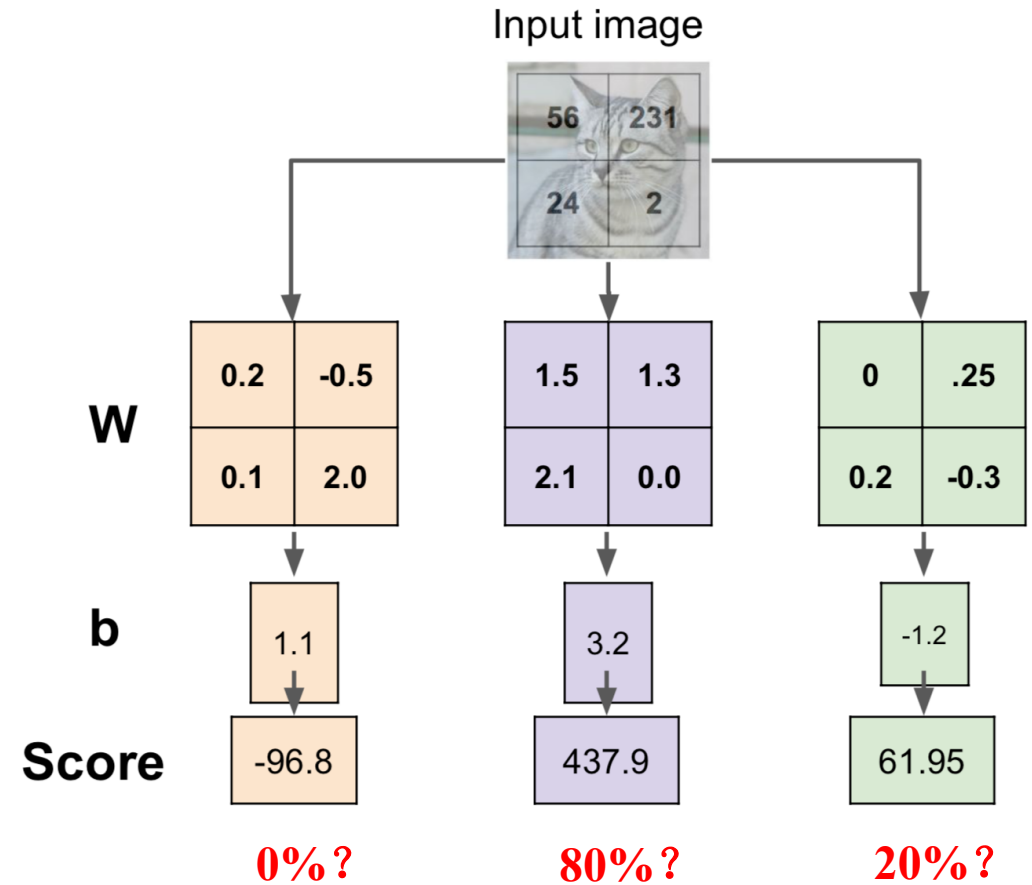
- Express preferences over weights
- Make the model simple so it works on test data
- Improve optimization by adding curvature

Softmax Operation

Interpret the outputs of our model as probabilities

$$\hat{y}_i = \text{softmax}(o_i) = \frac{\exp(o_i)}{\sum_j \exp(o_j)} \times 100\%$$

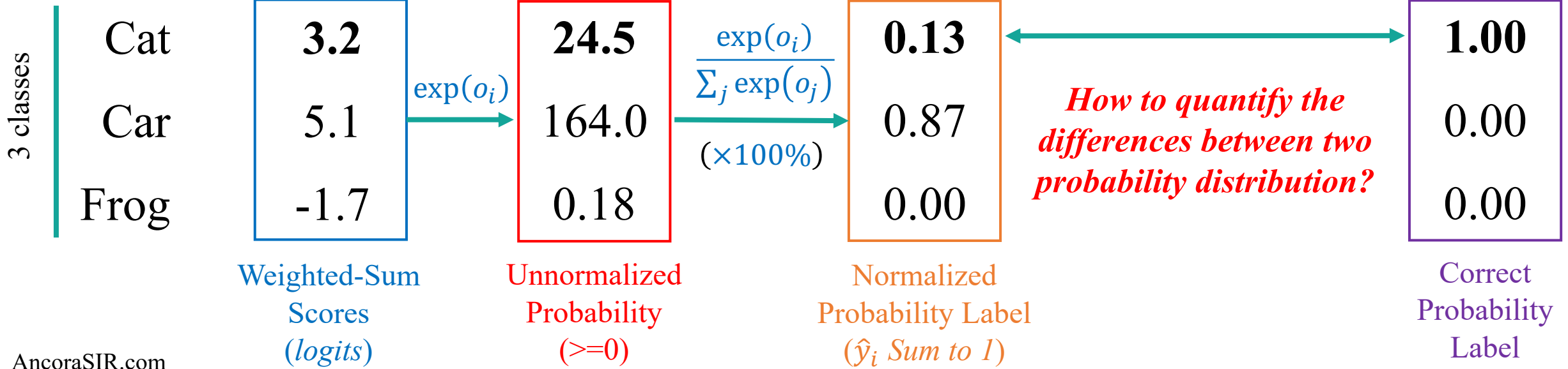
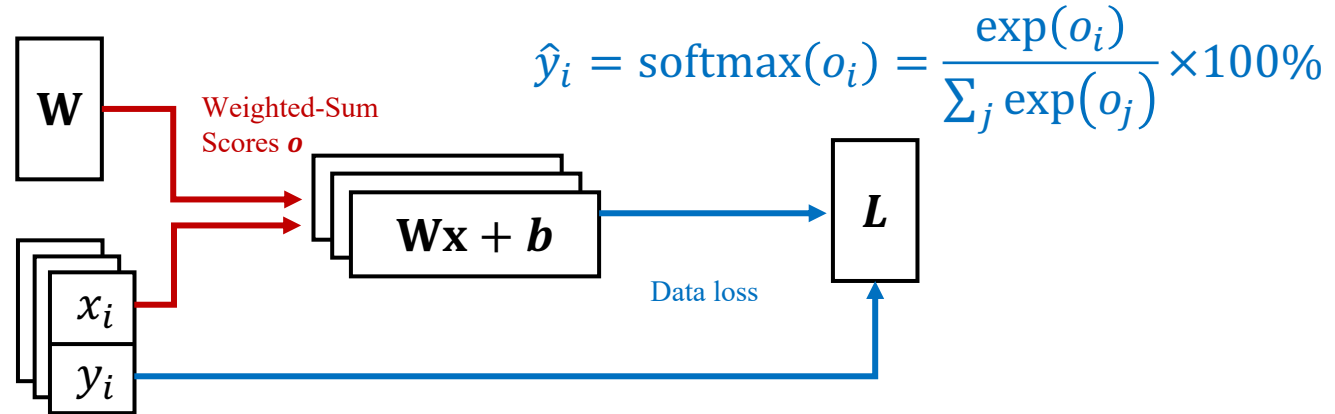
- One can interpret outputs \hat{y}_i as the probability that a given item belongs to class i .
- Then we can choose the class with the largest output value as our prediction
 - Why using o_i directly, instead of a probability?
 - What if the sum of probability is not 100%?
 - What if when o_i becomes negative?



Softmax Classifier

Want to interpret raw classifier scores as probabilities

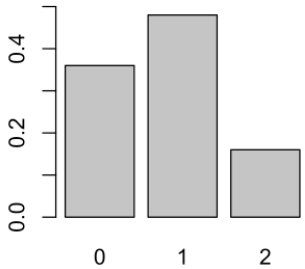
(x_1, y_1)
Ground Truth
Labelled Prediction



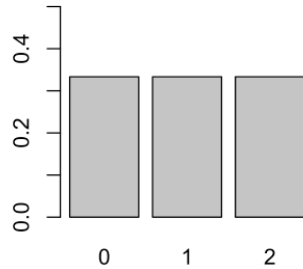
Kullback–Leibler Divergence

How to quantify the differences between two probability distribution?

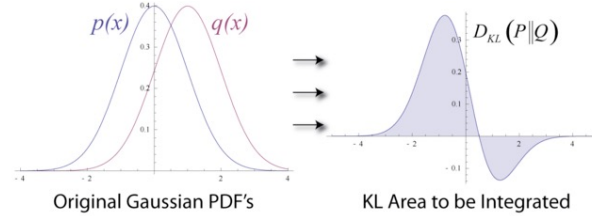
Distribution P
Binomial with $p = 0.4$, $N = 2$



Distribution Q
Uniform with $p = 1/3$



x	0	1	2
Distribution $P(x)$	0.36	0.48	0.16
Distribution $Q(x)$	0.333	0.333	0.333



$$\begin{aligned}
 D_{KL}(P \parallel Q) &= \sum_{y \in \mathcal{Y}} P(y) \log \frac{P(y)}{Q(y)} \\
 &= \sum_{y \in \mathcal{Y}} P(y) \log P(y) - \sum_{y \in \mathcal{Y}} P(y) \log Q(y) \\
 &= \left[- \sum_{y \in \mathcal{Y}} P(y) \log Q(y) \right] - \left[- \sum_{y \in \mathcal{Y}} P(y) \log P(y) \right] \\
 &= H(P, Q) - H(P)
 \end{aligned}$$

A good candidate of loss function for softmax

Can be minimized to update the weights

$$\begin{aligned}
 D_{KL}(P \parallel Q) &= \sum_{x \in \mathcal{X}} P(x) \ln \left(\frac{P(x)}{Q(x)} \right) \\
 &= 0.36 \ln \left(\frac{0.36}{0.333} \right) + 0.48 \ln \left(\frac{0.48}{0.333} \right) + 0.16 \ln \left(\frac{0.16}{0.333} \right) \\
 &= 0.0852996
 \end{aligned}$$

$$H(P, Q) = - \sum_{y \in \mathcal{Y}} P(y) \log Q(y) \quad H(P) = - \sum_{y \in \mathcal{Y}} P(y) \log P(y)$$

the cross-entropy of P and Q

the cross-entropy of P with itself (or the entropy of P)

Loss Function

Log-Likelihood expressed in cross-entropy

- The **likelihood** of the actual classes according to our model is

$$P(Y | X) = \prod_{i=1}^n P(y^{(i)} | x^{(i)}) \quad \longrightarrow \quad -\log P(Y | X) = \sum_{i=1}^n \boxed{-\log P(y^{(i)} | x^{(i)})}$$

- Maximizing the likelihood is equivalent to minimizing the log-likelihood.
- **Cross-entropy** loss for a single example (dropped superscript i)

$$l = -\log P(y | x) = -\sum_j y_j \log \hat{y}_j$$

- As \hat{y} is a discrete probability distribution and y is a one-hot vector, the sum over all j vanishes for all but one term.

Cross-Entropy Loss and its Derivative

Also called softmax loss

- Plugging \mathbf{o} into the definition of the cross-entropy loss, we obtain:

$$l = - \sum_j y_j \log \hat{y}_j = \sum_j y_j \log \sum_k \exp(o_k) - \sum_j y_j o_j = \log \sum_k \exp(o_k) - \sum_j y_j o_j$$

- The derivative with respect to \mathbf{o} is

$$\partial_{o_j} l = \frac{\exp(o_j)}{\sum_k \exp(o_k)} - y_j = \text{softmax}(\mathbf{o})_j - y_j = P(y = j | x) - y_j$$

- The gradient is $P(y = j | x) - y_j$
 - The difference between the probability predicted by our model $P(y = j | x)$ and the true label y .
- Similar to regression where the gradient is $\hat{y} - y$
 - The difference between the true label y and the estimation \hat{y}

Vectorization for Minibatches

We typically carry out vector calculations for minibatches of data for efficiency

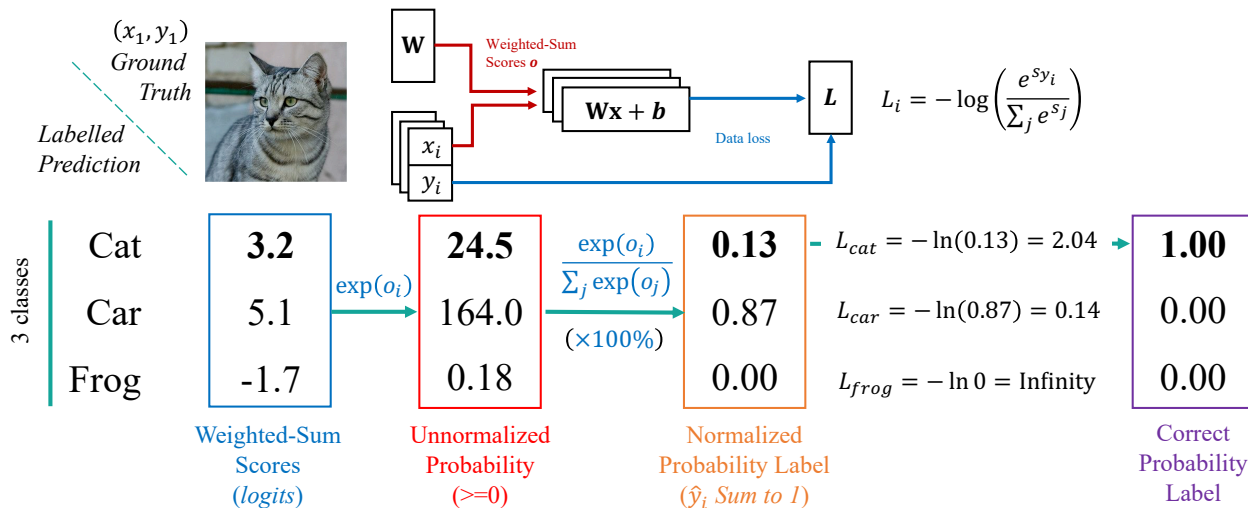
$$\hat{y}_i = \text{softmax}(o_i) = \frac{\exp(o_i)}{\sum_j \exp(o_j)} \times 100\%$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{o}) \text{ where } \hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)} \times 100\%$$

minibatch features \mathbf{X} are in $\mathbb{R}^{n \times d}$,
weights $\mathbf{W} \in \mathbb{R}^{d \times q}$, and
the bias satisfies $\mathbf{b} \in \mathbb{R}^q$.

$$\mathbf{O} = \mathbf{XW} + \mathbf{b},$$

$$\hat{\mathbf{Y}} = \text{softmax}(\mathbf{O})$$



A minibatch \mathbf{X} of examples

- dimensionality d and batch size n

Assume that we have q categories (outputs)

More efficient matrix-matrix computation \mathbf{XW}
Exponentiating all entries in \mathbf{O} then sum

Understanding of Softmax Regression

- When there are two classes, softmax regression reduces to logistic regression.

Softmax

Binary Classes

Logistic

$$\hat{y}_j = \frac{\exp(o_j)}{\sum_j \exp(o_j)}$$

Activation

$$\hat{y} = \frac{\exp(o)}{\exp(o) + 1}$$

$$-\frac{1}{n} \sum_{i=1}^n y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Loss

$$-\sum_{i=1}^n \sum_j y_j^{(i)} \log \hat{y}_j^{(i)}$$

- Softmax when $j=2$
$$\hat{y}_0 = \frac{\exp(o_0)}{\exp(o_0) + \exp(o_1)}$$
$$= \frac{\exp(o_0 - o_1)}{\exp(o_0 - o_1) + 1}$$

- The cross-entropy classification can be thought in two ways
 - As maximizing the likelihood of the observed data.
 - As minimizing out surprise required to communicate the labels.

Summary & Comparison

Linear Neural Network

	Linear Regression	Perceptron	Logistic Regression	Softmax Regression
Problem	Value Prediction	Binary Classification	Binary Classification	Multi-Class Classification
Weights	$wx + b$	$wx + b$	$wx + b$	$Wx + B$
Activation Function	NA	Step Function	Sigmoid Function	Softmax
Prediction Outputs	Continuous Value	Discrete Value 0, 1	Continuous Probability in (0,1)	A vector of Continuous Probabilities
Loss	Squared Loss	Hinge Loss	Log Loss (Binary cross entropy)	Cross Entropy
Decision Boundary			Linear	

Thank you~

songcy@sustech.edu.cn



AncoraSIR.com



SUSTech
Southern University
of Science and Technology