CSC 561: Neural Networks and Deep Learning

Logistic Regression

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

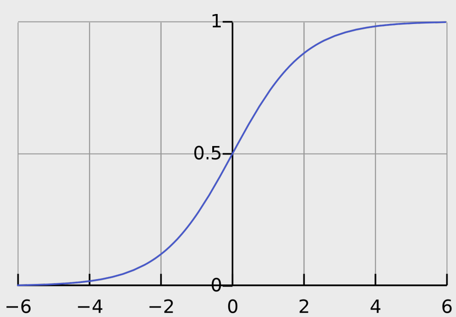
Spring 2024

THINK BIG WE DO℠

Logistic regression

Logistic function

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$



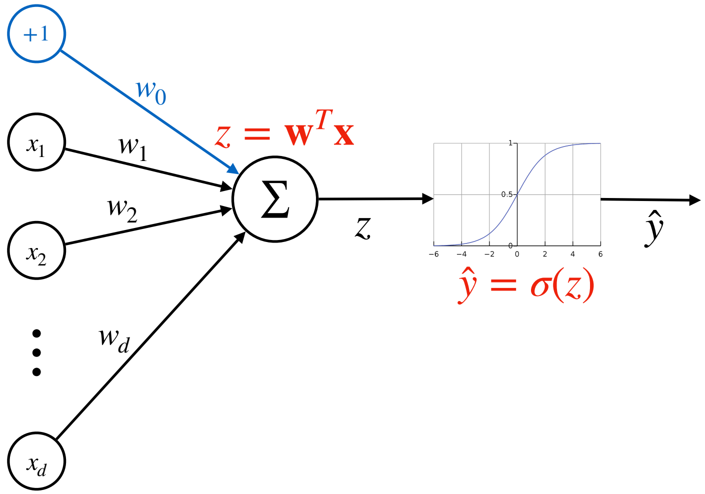

```
mapping ℝ to [0,1]

continuous and
differentiable
```

Logistic regression

- Binary classifier
  - models $P(y \mid \mathbf{x}), \mathbf{x} \in \mathbb{R}^d, y \in \{+1, -1\}$
    - a threshold (e.g., $\theta = 0.5$) may be used for a final binary classification
  - uses the logistic function
- A **linear classifier**
  - although the "activation function" is nonlinear

NN-like



Probabilistic interpretation

$P(y = -1 \mid \mathbf{x}; \mathbf{w}) = 1 - P(y = +1 \mid \mathbf{x}; \mathbf{w})$ $\quad$ $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$

(probability of class +1) $\quad P(y = +1 \mid \mathbf{x}; \mathbf{w}) = \dfrac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} = \sigma(\mathbf{w}^T\mathbf{x})$

(probability of class -1) $\quad P(y = -1 \mid \mathbf{x}; \mathbf{w}) = \dfrac{1}{1 + e^{\mathbf{w}^T\mathbf{x}}} = \sigma(-\mathbf{w}^T\mathbf{x})$

$$P(y \mid \mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-y\mathbf{w}^T\mathbf{x}}} = \sigma(y\mathbf{w}^T\mathbf{x})$$

`compact expression`

note that $P(y \mid \mathbf{x}) > 0.5$ when $y\mathbf{w}^T\mathbf{x} > 0$ (correct classifications)

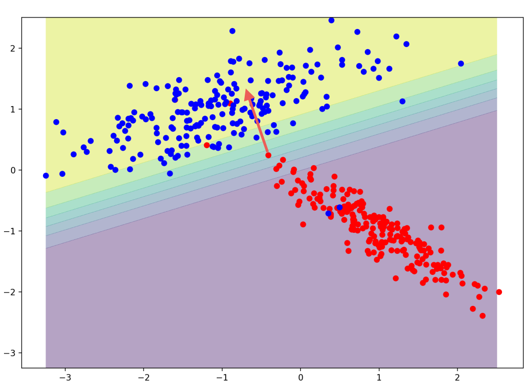
Linear decision boundary

$$P(y \mid \mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-y\mathbf{w}^T\mathbf{x}}} = \frac{1}{2}$$

$$1 + e^{-y\mathbf{w}^T\mathbf{x}} = 2$$

$$e^{-y\mathbf{w}^T\mathbf{x}} = 1$$

$$\mathbf{w}^T\mathbf{x} = 0$$



Learning the parameters

MLE

- Maximum likelihood estimation
  - choose parameters $\mathbf{w}$ that maximize the **conditional data likelihood** $P(\mathbf{y} \mid X; \mathbf{w})$, i.e., the probability of the observed values conditioned on the feature values

`i.i.d. assumption`

$$P(\mathbf{y} \mid X; \mathbf{w}) = \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w})$$

  - objective function:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w})$$

MLE

maximize the likelihood

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w})$$

maximize $P\left(y^{(i)} = +1 \mid \mathbf{x}^{(i)}\right)$ for any $\mathbf{x}^{(i)}$ with a positive label, and maximize $P\left(y^{(i)} = -1 \mid \mathbf{x}^{(i)}\right)$ for any $\mathbf{x}^{(i)}$ with a negative label

$$= \arg\max_{\mathbf{w}} \frac{1}{n} \log\left(\prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w})\right)$$

$$= \arg\max_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \log\left(P(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w})\right) \qquad \frac{1}{1 + e^{-y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)}}}$$

$$= \arg\max_{\mathbf{w}} -\frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)}}\right)$$

minimize the negative log likelihood

$$= \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \boxed{\log\left(1 + e^{-y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)}}\right)}$$

per instance loss

Applying gradient descent

$$J(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)}}\right)$$

`cross-entropy loss: no closed-form solution, but loss is convex`

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \left[\frac{\partial J(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial J(\mathbf{w})}{\partial w_d}\right]$$

`gradient`

$f(z) = \log(1 + e^z)$

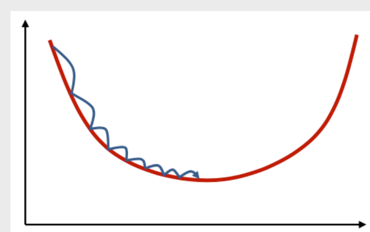
$f'(z) = \frac{e^z}{1 + e^z} = \sigma(z)$

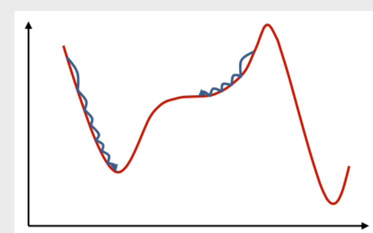
$$\frac{\partial J(\mathbf{w})}{\partial w_j} = -\frac{1}{n} \sum_{i=1}^{n} \sigma\left(-y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)}\right) y^{(i)} x_j^{(i)}$$

`partial derivative`

Gradient descent and convex functions

- Convex functions
  - for appropriate learning rates, GD will <u>always find the minimum</u>
- Non-convex functions
  - GD may find a local minimum (or inflection point)





Show me the code

```
def __forward(self, X):
    z = X @ self.weights
    y_pred = self.__sigmoid(z)
    return y_pred


def __loss(self, X, y):
    inv = -y * (X @ self.weights)
    loss = np.log(1 + np.exp(inv)).mean()
    dw = self.__sigmoid(inv) * -y * X
    dw = np.mean(dw, axis=0, keepdims=True).T
    return loss, dw
```


Understanding matrix form

$$\mathbf{y} = \mathbf{X}\mathbf{w}$$

input vector $\mathbf{x}^{(1)}$

$$\begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \approx \begin{bmatrix} \left(\mathbf{x}^{(1)}\right)^T \\ \vdots \\ \left(\mathbf{x}^{(n)}\right)^T \end{bmatrix} \cdot \begin{bmatrix} w_0 \\ \vdots \\ w_d \end{bmatrix}$$

Extension to multiple classes

From binary to C>2 classes

- Multinomial logistic regression
  - models $P(y = c \,|\, \mathbf{x}; W), \mathbf{x} \in \mathbb{R}^d, y \in \{0, \ldots, c-1\}$
  - uses the **softmax** function

softmax

$$P(y = c \mid \mathbf{x}; W) = \boxed{\frac{e^{\mathbf{w_c}^T\mathbf{x}}}{\sum_{k=1}^{C} e^{\mathbf{w_k}^T\mathbf{x}}}}$$

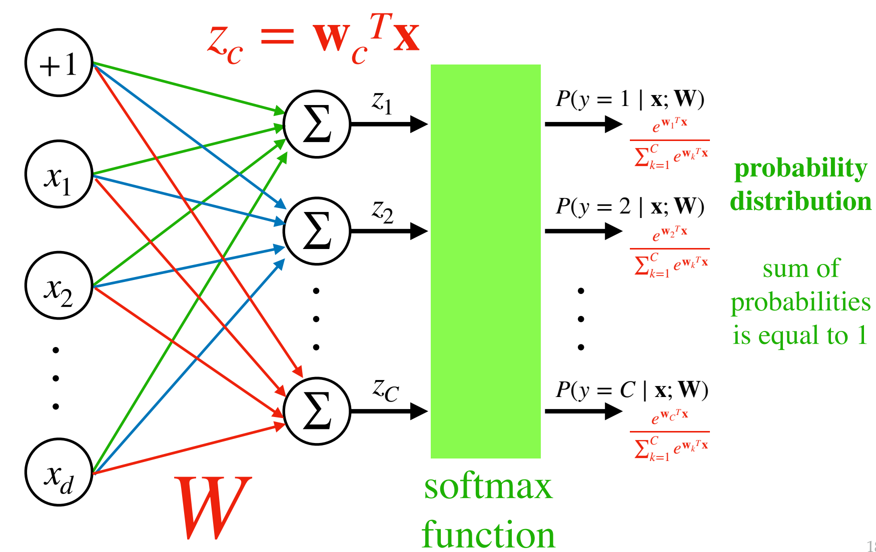
$W_{C\times(d+1)}$ is a matrix where rows are “per-class” weight vectors

One-hot encoding

	y
red	0
green	1
blue	2
blue	2
green	1
blue	2
yellow	3
blue	2
red	0
red	0

red	green	blue	yellow
1	0	0	0
0	1	0	0
0	0	1	0
0	0	1	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	1	0
1	0	0	0
1	0	0	0

NN-like



Multinomial logistic regression

- Predict the class with the highest probability

$$\hat{y} = \arg\max_{c} P(y = c \mid \mathbf{x}; \mathbf{W})$$

- How to learn the parameters?
  - use MLE to derive a **loss function** … then apply **gradient descent**

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \frac{1}{n} \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{W})$$

MLE

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \frac{1}{n} \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{W})$$

$$= \arg\max_{\mathbf{W}} \frac{1}{n} \prod_{i=1}^{n} \prod_{c=1}^{C} P\left(y^{(i)} = c \mid \mathbf{x}^{(i)}; \mathbf{W}\right)^{t_{i,c}}$$

`Consider a matrix` $T_{n\times C}$ `where every row is a` **`one-hot encoding`** `of the target variable`

$$= \arg\max_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^{n} \sum_{c=1}^{C} t_{i,c} \log\left(P(y^{(i)} = c \mid \mathbf{x}^{(i)}; \mathbf{W})\right)$$

$$= \arg\min_{\mathbf{W}} -\frac{1}{n} \sum_{i=1}^{n} \sum_{c=1}^{C} t_{i,c} \log\left(P(y^{(i)} = c \mid \mathbf{x}^{(i)}; \mathbf{W})\right)$$

$$= \arg\min_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^{n} \boxed{\sum_{c=1}^{C} -t_{i,c} \log\left(\frac{e^{\mathbf{w}_c \mathbf{x}^{(i)}}}{\sum_{k=1}^{C} e^{\mathbf{w}_k \mathbf{x}^{(i)}}}\right)}$$

per instance cross-entropy loss

Softmax and the cross-entropy loss

Probabilities must be >= 0 | Probabilities must sum to 1 | $L_i = -\log P(Y = y_i | X = x_i)$

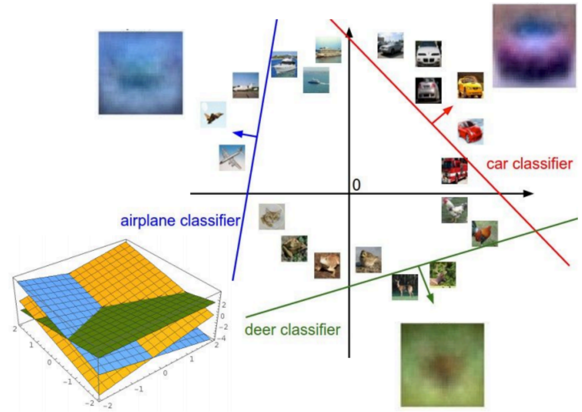
	Unnormalized log-probabilities / logits		unnormalized probabilities		probabilities	
cat	3.2	exp	24.5	normalize	0.13	$L_i = -\log(0.13) = 2.04$
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

Q	compare	P
0.13	*Kullback–Leibler divergence* $D_{KL}(P \| Q)$	1.00
0.87		0.00
0.00		0.00

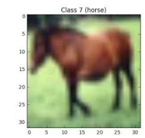
$$D_{KL}(P \| Q) = \sum P(y) \log \frac{P(y)}{Q(y)}$$

$$\underbrace{H(P, Q)}_{\text{cross-entropy}} = D_{KL}(P \| Q) - \underbrace{H(P)}_{\text{entropy}}$$

Geometric interpretation



Wx



Array of 32x32x3

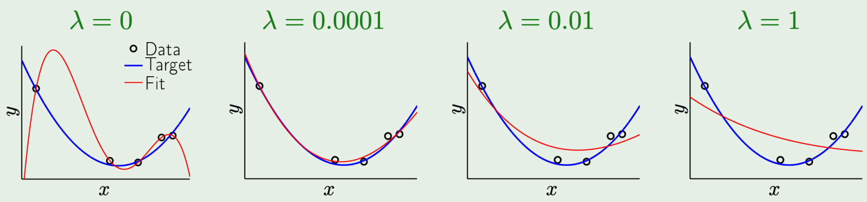
Regularization

Regularization

- Adding a **penalty** to the weights to control the complexity of the model
  - usually penalizing higher weights (**except intercept**)
  - results in **simpler** or **more sparse** solutions
- Impact of regularization can be controlled by a hyperparameter (*lambda*)

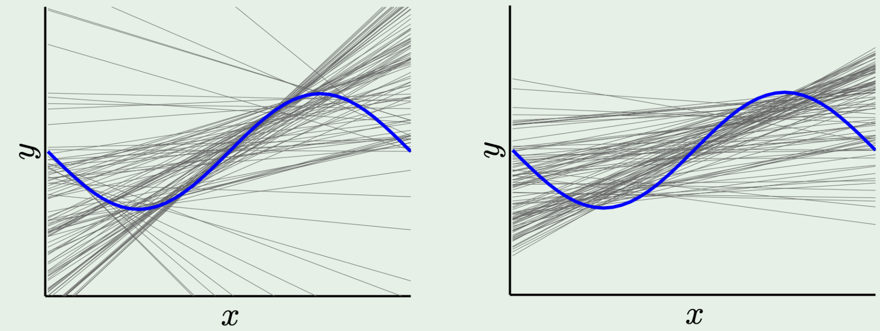
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \underbrace{l\left(h_w, x_i, y_i\right)}_{\text{better predictions}} \underbrace{+ \lambda R(\mathbf{w})}_{\text{control overfitting}}$$

Regularization prefers simpler models



overfitting ⟶ ⟶ ⟶ ⟶ underfitting

Restricting the hypothesis space



without regularization   with regularization

Regularization

- L1
  - tends to to drive weights to zero (**sparse solution**) effectively performing feature selection, which can be beneficial for interpretability and reducing model complexity
  - less sensitive to outliers compared to L2 regularization
  - absolute value function used in L1 regularization is not smooth, making the optimization process slightly more complex compared to L2

$$R(W) = \sum |W_{ij}|$$

- L2
  - encourages smaller coefficients by penalizing large ones more heavily
  - by shrinking weights, L2 regularization can improve the stability of the model, making it less sensitive to small changes in the data
  - can be more sensitive to outliers compared to L1 regularization
  - most common form of regularization (superior performance)

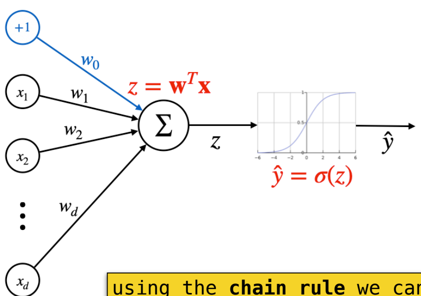
$$R(W) = \sum W_{ij}^2$$

- Other forms of regularization in neural networks
  - elastic net, dropout, batch normalization, early stopping, data augmentation

The importance of using differentiable functions

Differentiable activation functions

- Continuous activation functions
  - sigmoid, tanh, RELU, etc.
  - differentiable (almost) everywhere



$$\frac{d\hat{y}}{dz} = \sigma'(z)$$

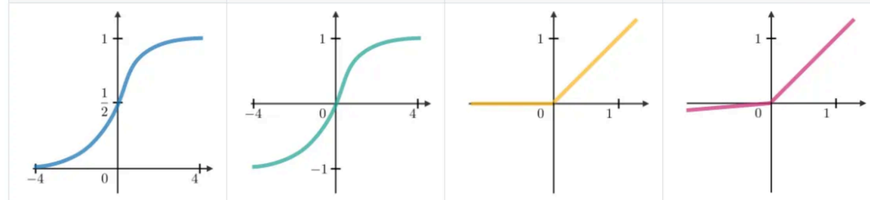
$$\frac{d\hat{y}}{dw_i} = \frac{d\hat{y}}{dz}\frac{dz}{dw_i} = \sigma'(z)x_i$$

$$\frac{d\hat{y}}{dx_i} = \frac{d\hat{y}}{dz}\frac{dz}{dx_i} = \sigma'(z)w_i$$

using the **chain rule** we can compute the change in the output for small changes to the input/weights

A few examples

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$

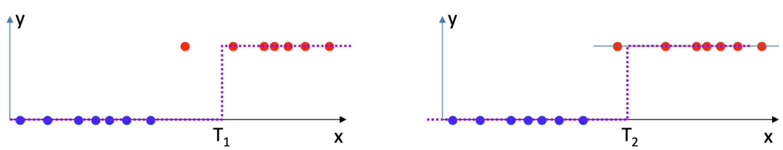


ReLU is a good default choice for most networks

https://medium.com/analytics-vidhya/understanding-activation-functions-data-science-for-the-rest-of-us-b652048a064f

Differentiable loss functions

- Threshold activation
  - shifting threshold does not change classification error



- Continuous loss and activation
  - can quantify a loss between continuous output and the desired target
    - the loss changes as the weights change, even if the classification error remains the same