Machine Learning
ITCS 6156/8156

Logistic Regression

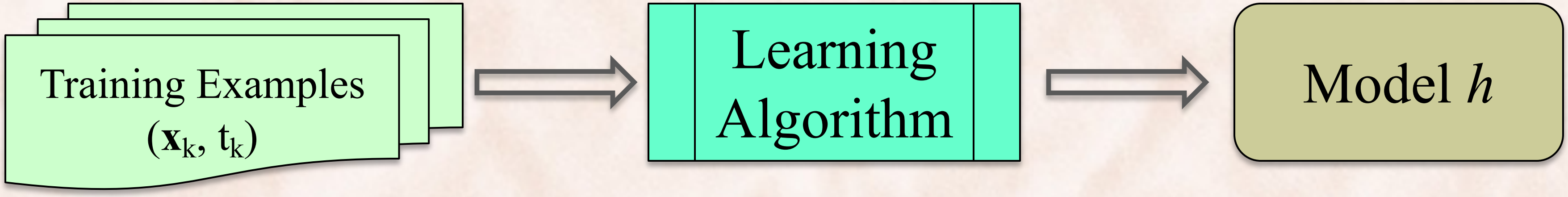
Razvan C. Bunescu

Department of Computer Science @ CCI

razvan.bunescu@uncc.edu

Supervised Learning

Training

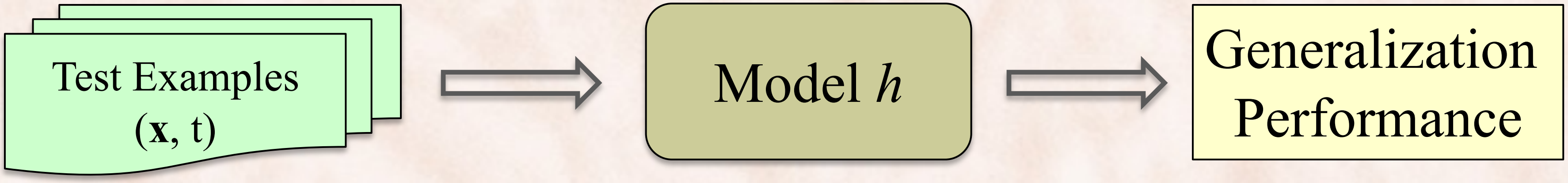


Training Examples ($\mathbf{x}_k$, t_k)

Learning Algorithm

Model h

Testing



Test Examples ($\mathbf{x}$, t)

Model h

Generalization Performance

Supervised Learning

- **Task** = learn an (unkown) function $t : X \rightarrow T$ that maps input instances $\mathbf{x} \in X$ to output targets $t(\mathbf{x}) \in T$:
  - **Classification**:
    - The output $t(\mathbf{x}) \in T$ is one of a finite set of discrete categories.
  - **Regression**:
    - The output $t(\mathbf{x}) \in T$ is continuous, or has a continuous component.

- Target function $t(\mathbf{x})$ is known (only) through (noisy) set of training examples:

  $(\mathbf{x}_1,t_1), (\mathbf{x}_2,t_2), \ldots (\mathbf{x}_n,t_n)$

Parametric Approaches to Supervised Learning

- **Task** = build a function $h(\mathbf{x})$ such that:
  - *h* matches *t* well on the training data:

    => *h* is able to fit data that it has seen.
  - *h* also matches *t* well on test data:

    => *h* is able to **generalize to unseen data**.

- **Task** = choose *h* from a "nice" *class of functions* that depend on a vector of parameters **w**:
  - $h(\mathbf{x}) \equiv h_{\mathbf{w}}(\mathbf{x}) \equiv h(\mathbf{w},\mathbf{x})$
  - **what classes of functions are "nice"?**

Three Parametric Approaches to Classification

1) Discriminant Functions: scoring function $f: X \rightarrow T$ that directly assigns a vector $\mathbf{x}$ to a specific class C_k.
   - Inference and decision combined into a single learning problem.
   - *Linear Discriminant*: the decision surface is a hyperplane in X:
     - Perceptron
     - Support Vector Machines
     - Fisher 's Linear Discriminant

Three Parametric Approaches to Classification

2) Probabilistic Discriminative Models: directly model the posterior class probabilities $p(C_k \mid \mathbf{x})$.

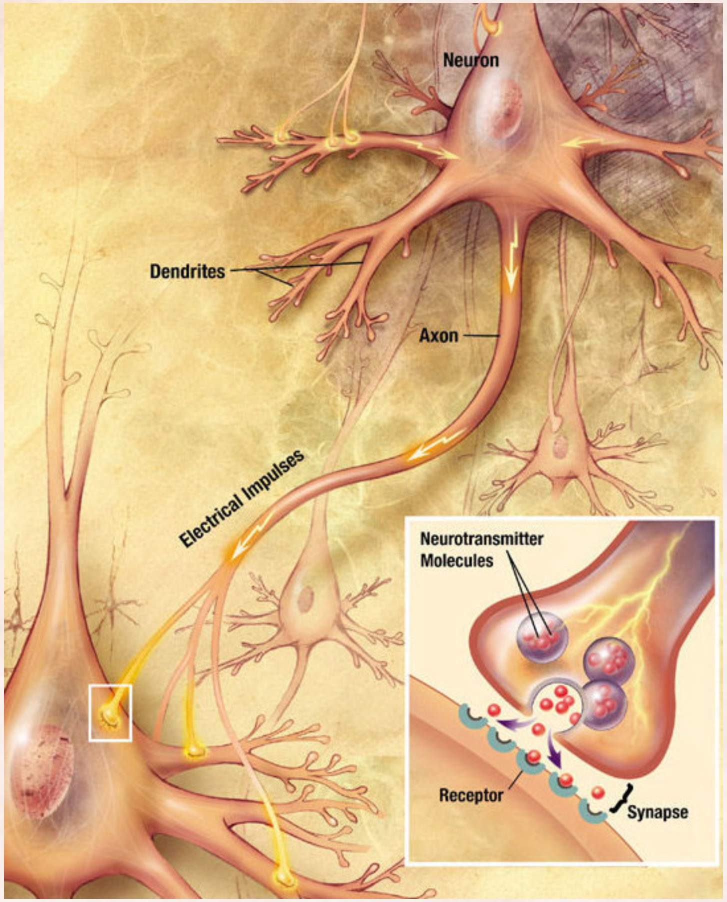
   - Inference and decision are separate.
   - Less data needed to estimate $p(C_k \mid \mathbf{x})$ than $p(\mathbf{x} \mid C_k)$.
   - Can accommodate many overlapping features.
     - Logistic Regression
     - Conditional Random Fields

Three Parametric Approaches to Classification

3) Probabilistic Generative Models:

- Model class-conditional $p(\mathbf{x} \mid C_k)$ as well as the priors $p(C_k)$, then use Bayes's theorem to find $p(C_k \mid \mathbf{x})$.
    - or model $p(\mathbf{x},C_k)$ directly, then marginalize to obtain the posterior probabilities $p(C_k \mid \mathbf{x})$.
- Inference and decision are separate.
- Can use $p(\mathbf{x})$ for *outlier* or *novelty detection*.
- Need to model dependencies between features.
    - Naïve Bayes.
    - Hidden Markov Models.

Neurons



Soma is the central part of the neuron:

- *where the input signals are combined.*

Dendrites are cellular extensions:

- *where majority of the input occurs.*

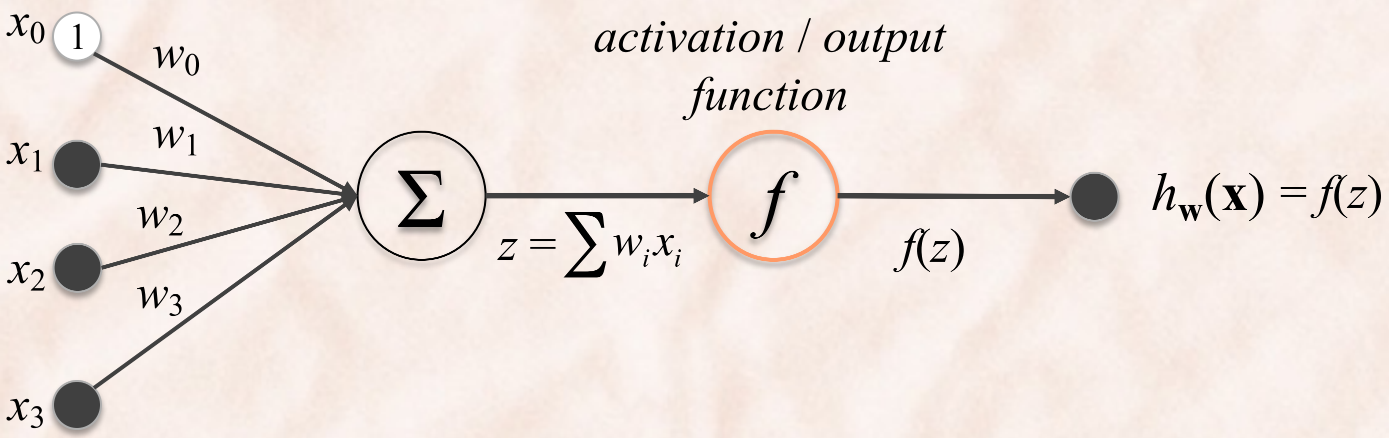
Axon is a fine, long projection:

- *carries nerve signals to other neurons.*

Synapses are molecular structures between axon terminals and other neurons:

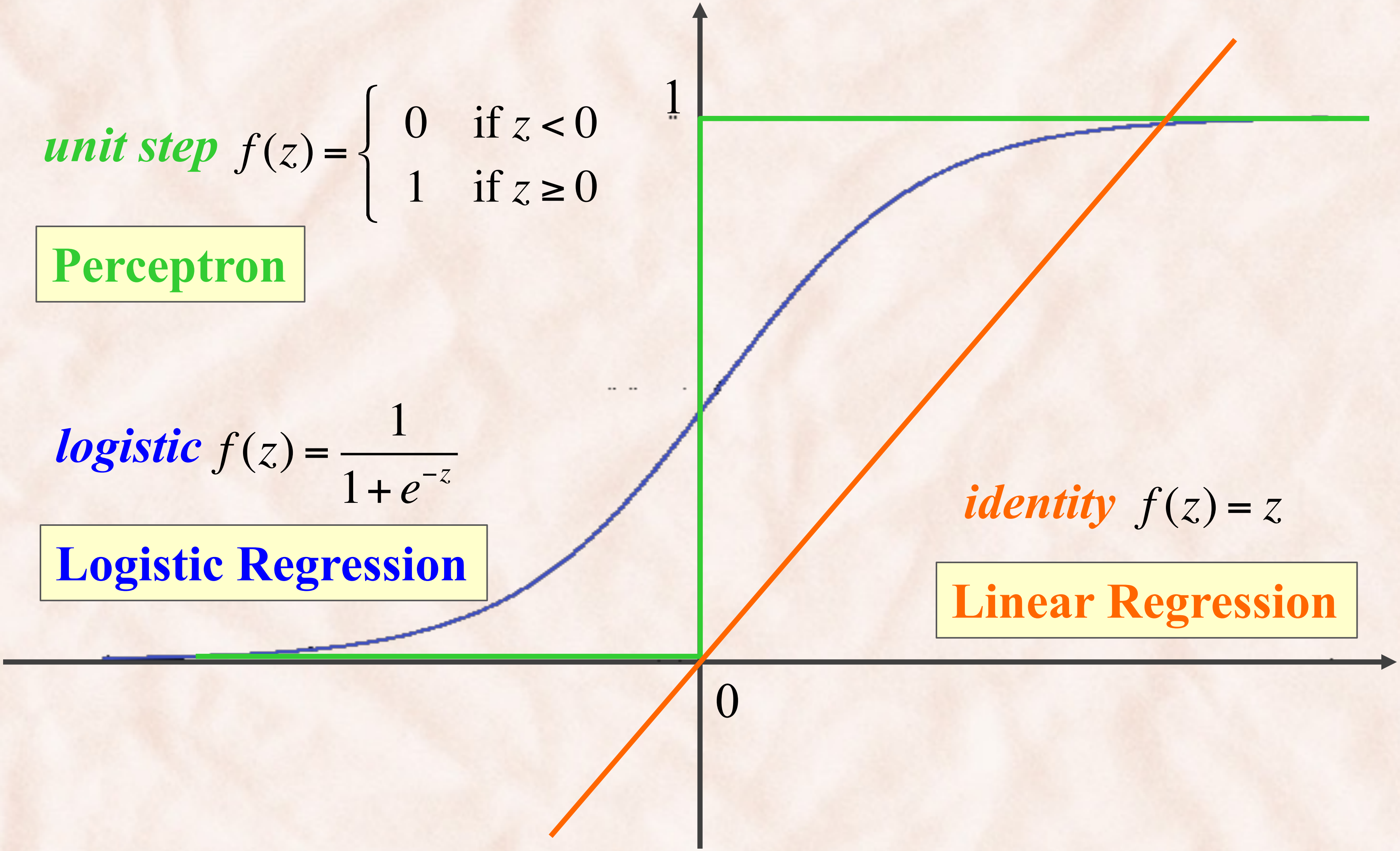
- *where the communication takes place.*

McCulloch-Pitts Neuron Function



- Algebraic interpretation:
  - The output of the neuron is a **linear combination** of inputs from other neurons, **rescaled by** the synaptic **weights**.
    - weights w_i correspond to the synaptic weights (activating or inhibiting).
    - summation corresponds to combination of signals in the soma.
  - It is often transformed through an **activation / output function**.

Activation / Output Functions



unit step $f(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

Perceptron

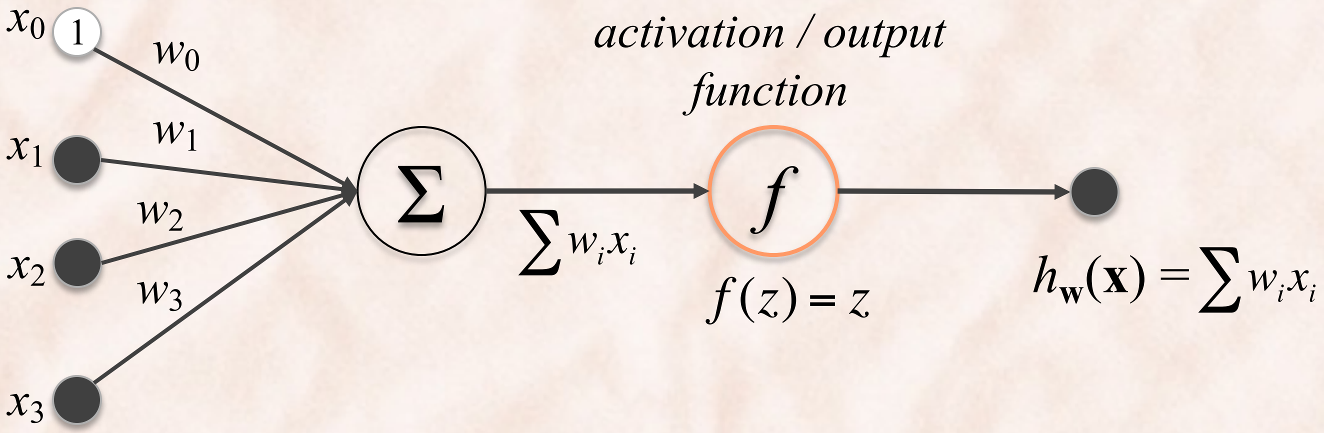
logistic $f(z) = \dfrac{1}{1+e^{-z}}$

Logistic Regression

identity $f(z) = z$

Linear Regression

Linear Regression

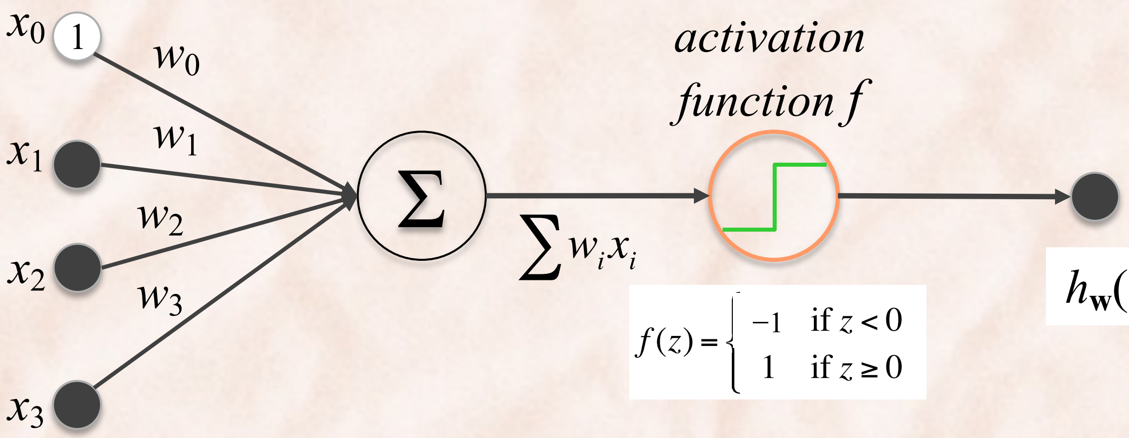


- Polynomial curve fitting is Linear Regression:

  $\mathbf{x} = \varphi(x) = [1, x, x^2, \ldots, x^M]^T$

  $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

Perceptron



$$f(z) = \begin{cases} -1 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

- Assume classes T = $\{c_1, c_2\} = \{1, -1\}$.
- Training set is $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots (\mathbf{x}_n, t_n)$.

$\mathbf{x} = [1, x_1, x_2, ..., x_k]^T$

$h(\mathbf{x}) = sgn(\mathbf{w}^T\mathbf{x}) = sgn(w_0 + w_1\, x_1 + \ldots + w_k\, x_k)$

a linear discriminant function

Linear Discriminant Functions

- Use a linear function of the input vector:

$$h(\mathbf{x}) = \mathbf{w}^T\varphi(\mathbf{x}) + w_0$$

weight vector

bias $= -$ *threshold*

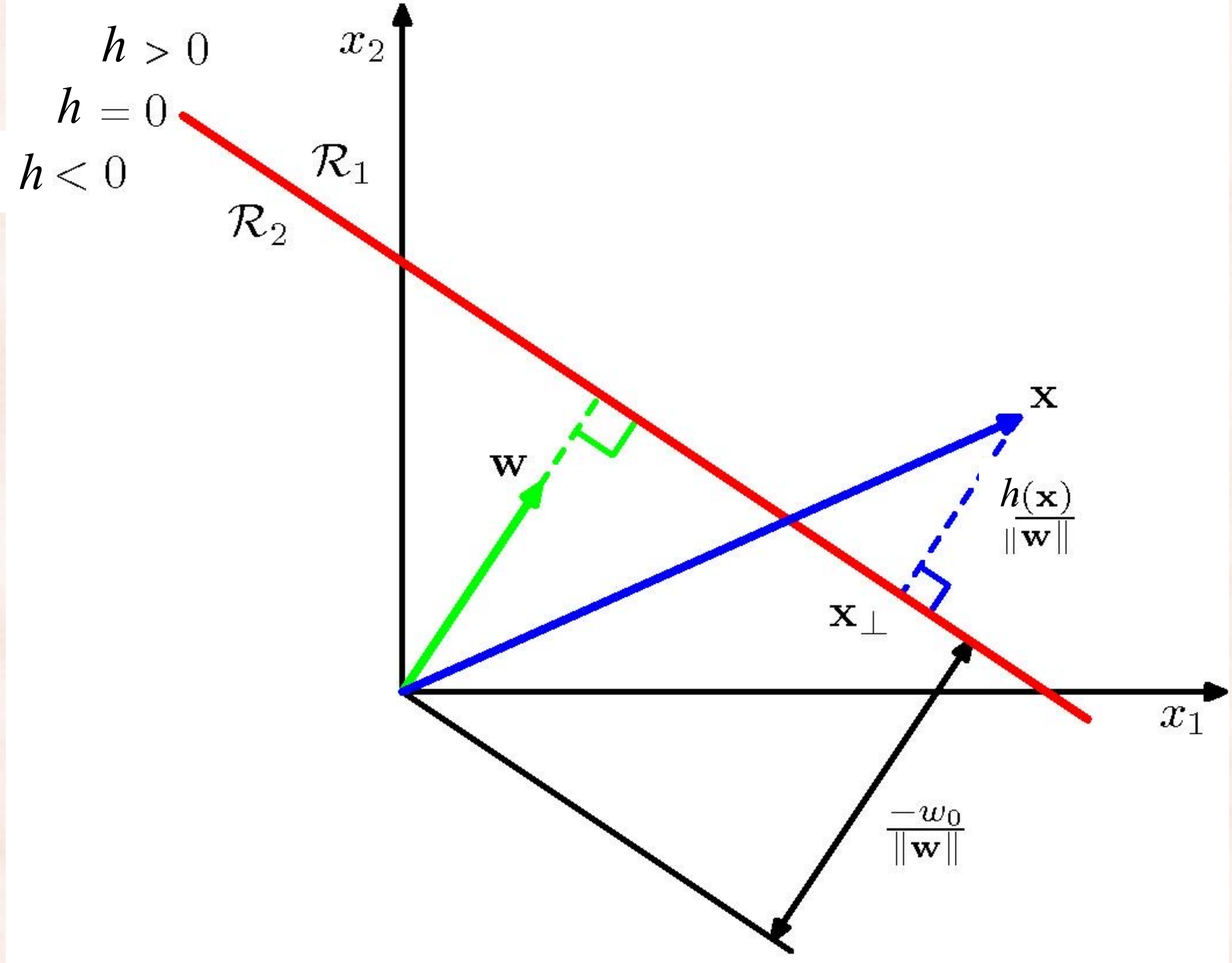
- Decision:

$\mathbf{x} \in C_1$ if $h(\mathbf{x}) \geq 0$, otherwise $\mathbf{x} \in C_2$.

$\Rightarrow$ decision boundary is hyperplane $h(\mathbf{x}) = 0$.

- Properties:
  - **w** is orthogonal to vectors lying within the decision surface.
  - w_0 controls the location of the decision hyperplane.

Geometric Interpretation



From Perceptron to Logistic Regression

- Features $\mathbf{x} = [1, x_1, x_2, x_3, x_4]$.
- Weights $\mathbf{w} = [\mathrm{w}_0, \mathrm{w}_1, \mathrm{w}_2, \mathrm{w}_3, \mathrm{w}_4]$

Discriminant function model

Perceptron

Training: Find **w** to fit training data.
Inference: Compute $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$
Decision:

- if $h(\mathbf{x}) \geq 0$ output label +1
- else output label -1

Probabilistic discriminative model

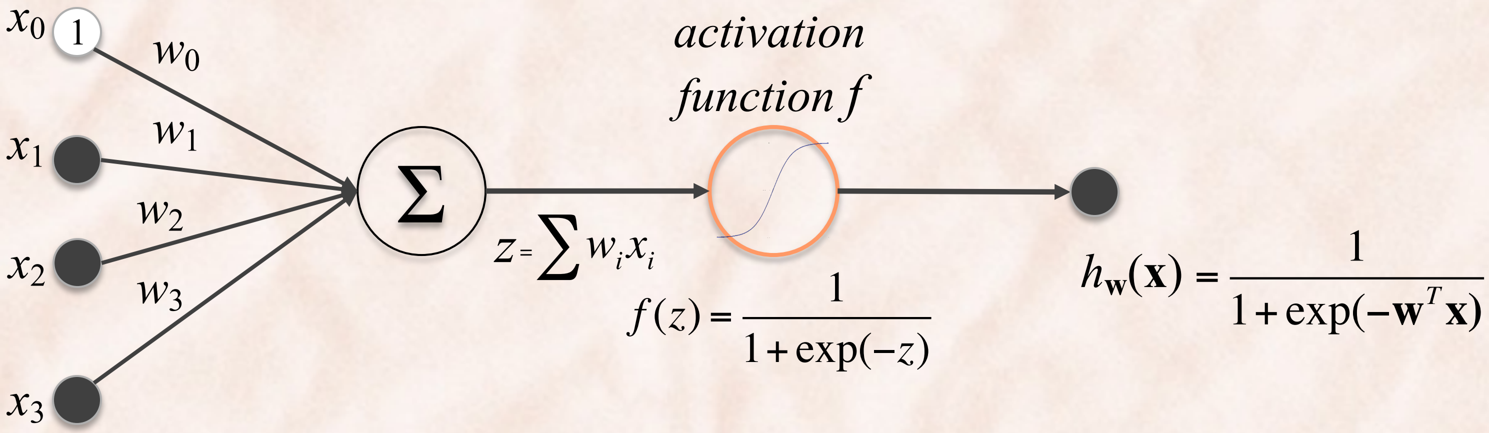
Logistic Regression

Training: Find **w** to fit training data.
Inference: Compute $z = \mathbf{w}^T\mathbf{x}$
Decision:

- if $z \geq 0$ output label +1
- else output label 0

Take logit z, compute probabilistic output $\mathrm{p}(+1|\mathbf{x}) = \sigma(\mathrm{z}) = \frac{1}{1+\exp(-z)}$

Logistic Regression for Binary Classification

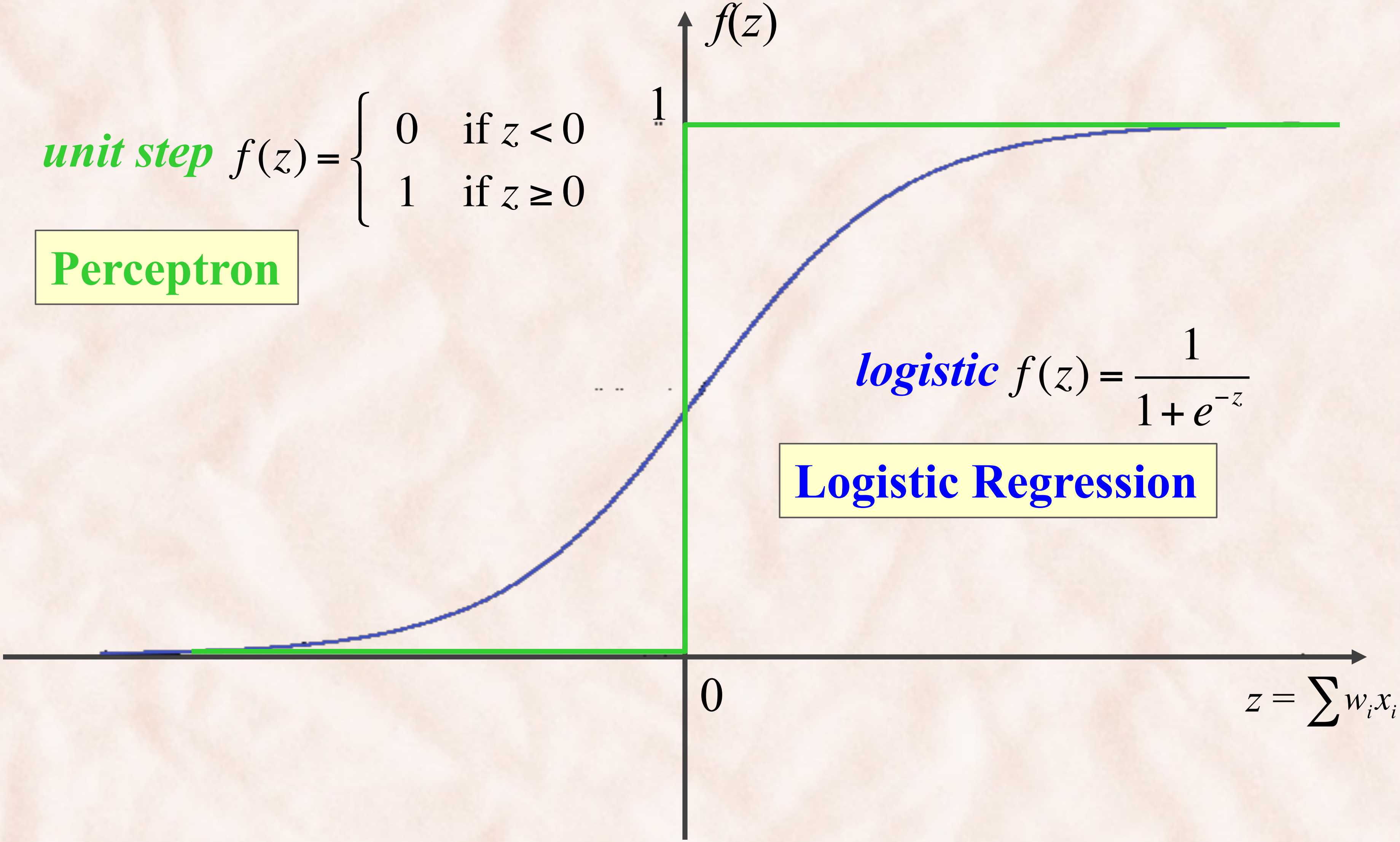


x_0
1
w_0
x_1
w_1
w_2
x_2
w_3
x_3
Σ
$z = \sum w_i x_i$
activation function f
$f(z) = \frac{1}{1+\exp(-z)}$
$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$

- Used for binary **classification**:
  - Labels T = $\{C_1, C_2\} = \{1, 0\}$
  - Output C_1 iff $h(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) > 0.5$
- Training set is $(\mathbf{x}_1,t_1), (\mathbf{x}_2,t_2), \ldots (\mathbf{x}_n,t_n)$.

  $\mathbf{x} = [1, x_1, x_2, \ldots, x_k]^T$

Activation / Output Functions f



unit step $f(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \ge 0 \end{cases}$

Perceptron

logistic $f(z) = \dfrac{1}{1 + e^{-z}}$

Logistic Regression

Logistic Regression for Binary Classification

- Model output can be interpreted as **posterior class probabilities**:

$$p(C_1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x}))}$$

$$p(C_2 \mid \mathbf{x}) = 1 - \sigma(\mathbf{w}^T \mathbf{x}) = \frac{\exp(-\mathbf{w}^T \mathbf{x})}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

Linear *decision boundary*

- Inference:
  - Output C_1 if p(C_1|x) ≥ 0.5, else output C_2.
    - assuming uniform misclassification costs …

Logistic Regression Learning

- Learning = finding the "right" parameters $\mathbf{w}^T = [w_0, w_1, \ldots, w_k]$
  - Find **w** that minimizes an *error function* $E(\mathbf{w})$ which measures the misfit between $h(\mathbf{x}_n, \mathbf{w})$ and t_n.
  - Expect that $h(\mathbf{x},\mathbf{w})$ performing well on training examples $\mathbf{x}_n \Rightarrow$ $h(\mathbf{x},\mathbf{w})$ will perform well on arbitrary test examples $\mathbf{x} \in X$.

- **Least Squares** error function?

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{h(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

  - Differentiable => can use gradient descent ✓
  - Non-convex => not guaranteed to find the global optimum ✗

Maximum Likelihood

Training set is D = $\{\langle \mathbf{x}_n, t_n\rangle \mid t_n \in \{0,1\}, n \in 1\ldots\mathrm{N}\}$

Let $h_n = p(C_1 | \mathbf{x}_n) \Leftrightarrow h_n = p(t_n = 1 | \mathbf{x}_n) = \sigma(\mathbf{w}^T \mathbf{x}_n)$

Maximum Likelihood (ML) principle: find parameters that maximize the likelihood of the labels.

- The likelihood function is: $p(\mathbf{t}|\mathbf{w}, \mathrm{X}) = \prod_{n=1}^{N} p(t_n|\mathbf{w}, x_n)$
- The negative log-likelihood (cross entropy) error function:

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N} \ln p(t_n|x_n)$$

Maximum Likelihood

Training set is D = $\{\langle \mathbf{x}_n, t_n\rangle \mid t_n \in \{0,1\}, n \in 1 \ldots \mathrm{N}\}$

Let $h_n = p(C_1 \mid \mathbf{x}_n) \Leftrightarrow h_n = p(t_n = 1 \mid \mathbf{x}_n) = \sigma(\mathbf{w}^T \mathbf{x}_n)$

Maximum Likelihood (ML) principle: find parameters that maximize the likelihood of the labels.

- The likelihood function is $p(\mathbf{t} \mid \mathbf{w}) = \prod_{n=1}^{N} h_n^{t_n} (1 - h_n)^{(1-t_n)}$
- The negative log-likelihood (cross entropy) error function:

$$E(\mathbf{w}) = -\ln p(\mathbf{t} \mid \mathbf{x}) = -\sum_{n=1}^{N} \left\{ t_n \ln h_n + (1 - t_n) \ln(1 - h_n) \right\}$$

Maximum Likelihood Learning for Logistic Regression

- The ML solution is:

convex in $\mathbf{w}$

$$\mathbf{w}_{ML} = \arg\max_{\mathbf{w}} p(\mathbf{t} \mid \mathbf{w}) = \arg\min_{\mathbf{w}} \boxed{E(\mathbf{w})}$$

- ML solution is given by $\nabla E(\mathbf{w}) = 0$.
  - Cannot solve analytically => solve numerically with gradient based methods: (stochastic) gradient descent, conjugate gradient, L-BFGS, etc.
  - Gradient is (prove it):

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (h_n - t_n)\mathbf{x}_n$$

  - If we separate bias b from $\mathbf{w}$, what is $\nabla E(b)$?

Regularized Logistic Regression

- Use a Gaussian prior over the parameters:

$\mathbf{w} = [w_0, w_1, \dots , w_M]^{\mathrm{T}}$

$$p(\mathbf{w}) = N(\mathbf{0}, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left\{-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right\}$$

- Bayes' Theorem:

$$p(\mathbf{w} \mid \mathbf{t}) = \frac{p(\mathbf{t} \mid \mathbf{w})\, p(\mathbf{w})}{p(\mathbf{t})} \propto p(\mathbf{t} \mid \mathbf{w})\, p(\mathbf{w})$$

- MAP solution:

$$\mathbf{w}_{MAP} = \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{t})$$

Regularized Logistic Regression

- MAP solution:

$$
\begin{aligned}
\mathbf{w}_{MAP} &= \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{t}) \quad = \arg\max_{\mathbf{w}} p(\mathbf{t} \mid \mathbf{w}) p(\mathbf{w}) \\
&= \arg\min_{\mathbf{w}} -\ln p(\mathbf{t} \mid \mathbf{w}) p(\mathbf{w}) \\
&= \arg\min_{\mathbf{w}} -\ln p(\mathbf{t} \mid \mathbf{w}) - \ln p(\mathbf{w}) \\
&= \arg\min_{\mathbf{w}} E_D(\mathbf{w}) - \ln p(\mathbf{w}) \\
&= \arg\min_{\mathbf{w}} E_D(\mathbf{w}) + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} \quad \boxed{= \arg\min_{\mathbf{w}} E_D(\mathbf{w}) + E_{\mathbf{w}}(\mathbf{w})}
\end{aligned}
$$

$$
E_D(\mathbf{w}) = -\sum_{n=1}^{N} \{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \} \times \frac{1}{N}
$$

data term
(we also average)

$$
E_{\mathbf{w}}(\mathbf{w}) = \frac{\alpha}{2} \mathbf{w}^T \mathbf{w}
$$

regularization term

Regularized Logistic Regression

- MAP solution:

$$\mathbf{w}_{MAP} = \arg\min_{\mathbf{w}} \boxed{E_D(\mathbf{w}) + E_{\mathbf{w}}(\mathbf{w})}$$

still convex in **w**

- ML solution is given by $\nabla E(\mathbf{w}) = 0$.

α is also called ***decay***

$$\nabla E(\mathbf{w}) = \nabla E_D(\mathbf{w}) + \nabla E_{\mathbf{w}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(h_n - t_n)\mathbf{x}_n + \alpha\mathbf{w}$$

where $h_n = \sigma(\mathbf{w}^T\mathbf{x}_n)$

- Cannot solve analytically => solve numerically:
  - (stochastic) **gradient descent** [PRML 3.1.3], Newton Raphson iterative optimization [PRML 4.3.3], conjugate gradient, LBFGS.

Implementation: Vectorization of LR

- **Version 1**: Compute gradient component-wise.

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (h_n - t_n)\mathbf{x}_n \times \frac{1}{N}$$

  - Assume example $\mathbf{x}_n$ is stored in column X[:,n] in data matrix X.

```
grad = np.zeros(K)
for n in range(N):
    h = sigmoid(w.dot(X[:,n])
    temp = h − t[n]
    for k in range(K):
      grad[k] = grad[k] + temp * X[k,n] / N
```

```
def sigmoid(x):
   return 1 / (1 + np.exp(−x))
```

Implementation: Vectorization of LR

- **Version 2**: Compute gradient, partially vectorized.

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (h_n - t_n)\mathbf{x}_n \times \frac{1}{N}$$

```
grad = np.zeros(K)
for n in range(N):
    grad = grad + (sigmoid(w.dot(X[:,n])) − t[n]) * X[:,n] / N
```

```
def sigmoid(x):
    return 1 / (1 + np.exp(−x))
```

Implementation: Vectorization of LR

- **Version 3**: Compute gradient, vectorized.

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (h_n - t_n)\mathbf{x}_n \times \frac{1}{N}$$

grad = X.dot(sigmoid(**w**.dot(X)) − **t**) / N

```
def sigmoid(x):
    return 1 / (1 + np.exp(−x))
```

Vectorization of LR with Separate Bias

- Separate the bias b from the weight vector **w**.
- Compute gradient separately with respect to **w** and b:
  - Gradient with respect to **w** is:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (h_n - t_n)\mathbf{x}_n \times \frac{1}{N}$$

$$h_n = \sigma(\mathbf{w}^T \mathbf{x}_n + b)$$

grad = X.dot(sigmoid(**w**.dot(X) + b) − **t**) / N

  - Gradient with respect to bias b is:

$$\Delta b = -\frac{1}{N} \sum_{n=1}^{N} (h_n - t_n)$$

```
def sigmoid(x):
   return 1 / (1 + np.exp(−x))
```

Vectorization of LR with Regularization

- Only the gradient with respect to **w** changes:
  - never use L2 regularization on bias.

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (h_n - t_n)\mathbf{x}_n \times \frac{1}{N} + \alpha\mathbf{w}$$

$$\boldsymbol{grad} = \text{X.dot(sigmoid(}\mathbf{w}\text{.dot(X)} + b) - \mathbf{t}) \,/\, \text{N} + \alpha\mathbf{w}$$

Softmax Regression = Logistic Regression for Multiclass Classification

- Multiclass classification:

  T = {C$_1$, C$_2$, ..., C$_K$} = {1, 2, ..., K}.

- Training set is ($\mathbf{x}_1$,t$_1$), ($\mathbf{x}_2$,t$_2$), … ($\mathbf{x}_n$,t$_n$).

  $\mathbf{x} = [1, x_1, x_2, ..., x_M]$

  t$_1$, t$_2$, … t$_n$ $\in$ {1, 2, ..., K}

- One weight vector per class [PRML 4.3.4]:

$$p(C_k \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}))}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x})}$$

bias parameter inside each $\mathbf{w}_j$

$$p(C_k|\mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}_n + b_k)}{\sum_{j=1..K} \exp(\mathbf{w}_j^T \mathbf{x}_n + b_j)}$$

separate bias parameter b_j

Softmax Regression (K ≥ 2)

- Inference:

$$
\begin{aligned}
C_* &= \arg\max_{C_k} p(C_k \mid \mathbf{x}) \\
&= \arg\max_{C_k} \frac{\exp(\mathbf{w}_k^T \mathbf{x})}{\boxed{\sum_j \exp(\mathbf{w}_j^T \mathbf{x})}} \\
&= \arg\max_{C_k} \exp(\mathbf{w}_k^T \mathbf{x}) \\
&= \arg\max_{C_k} \mathbf{w}_k^T \mathbf{x}
\end{aligned}
$$

$Z(\mathbf{x})$ *a normalization constant*

- Training using:
  - Maximum Likelihood (ML)
  - Maximum A Posteriori (MAP) with a Gaussian prior on **w**.

Softmax Regression

- The **negative log-likelihood** error function is:

$$E_D(\mathbf{w}) = -\frac{1}{N}\ln\prod_{n=1}^{N} p(t_n \mid \mathbf{x}_n) = \boxed{-\frac{1}{N}\sum_{n=1}^{N}\ln\frac{\exp(\mathbf{w}_{t_n}^T\mathbf{x}_n)}{Z(\mathbf{x}_n)}} \dashrightarrow \boxed{\textit{convex in } \mathbf{w}}$$

- The Maximum Likelihood solution is:

$$\mathbf{w}_{ML} = \arg\min_{\mathbf{w}} E_D(\mathbf{w})$$

- The **gradient** is (prove it):

$$\nabla_{\mathbf{w}_k} E_D(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\big(\delta_k(t_n) - p(C_k \mid \mathbf{x}_n)\big)\mathbf{x}_n$$

where $\delta_t(x) = \begin{cases} 1 & x = t \\ 0 & x \neq t \end{cases}$ is the *Kronecker delta* function.

Regularized Softmax Regression

- The new **cost** function is:

$$E(\mathbf{w}) = E_D(\mathbf{w}) + E_{\mathbf{w}}(\mathbf{w})$$

$$= -\frac{1}{N}\sum_{n=1}^{N} \ln \frac{\exp\left(\mathbf{w}_{t_n}^T \mathbf{x}_n\right)}{Z(\mathbf{x}_n)} + \frac{\alpha}{2}\|\mathbf{W}\|^2$$

- The new **gradient** is (prove it):

$$\boldsymbol{grad}_k = \nabla_{\mathbf{w}_k} E(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\left(\delta_k(t_n) - p(C_k \mid \mathbf{x}_n)\right)\mathbf{x}_n + \alpha \mathbf{w}_k$$

Softmax Regression

- ML solution is given by $\nabla E_D(\mathbf{w}) = 0$.
  - Cannot solve analytically.
  - Solve numerically, by pluging [*cost*, *gradient*] = $[E(\mathbf{w}), \nabla E(\mathbf{w})]$ values into general convex solvers:
    - L-BFGS
    - Newton methods
    - conjugate gradient
    - (stochastic / minibatch) gradient-based methods.
      - gradient descent (with / without momentum).
      - AdaGrad, AdaDelta
      - RMSProp
      - ADAM, ...

Implementation

- Need to compute [*cost*, ***grad***]:
  - *cost* $= -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\delta_k(t_n)\ln p(C_k \mid \mathbf{x}_n) + \frac{\alpha}{2}\sum_{k=1}^{K}\mathbf{w}_k^T\mathbf{w}_k$
  - ***grad***$_k$ $= -\frac{1}{N}\sum_{n=1}^{N}\left(\delta_k(t_n) - p(C_k \mid \mathbf{x}_n)\right)\mathbf{x}_n + \alpha\mathbf{w}_k$

=> need to compute, for *k* = 1, ..., K:

- *output* $p(C_k \mid \mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T\mathbf{x}_n))}{\sum_j \exp(\mathbf{w}_j^T\mathbf{x}_n)}$

Overflow when $\mathbf{w}_k^T\mathbf{x}_n$ are too large.

Implementation: Preventing Overflows

- Subtract from each product $\mathbf{w}_k{}^{\mathrm{T}}\mathbf{x}_n$ the maximum product:

$$c_n = \max_{1 \le k \le K} \mathbf{w}_k^T \mathbf{x}_n$$

$$p(C_k \mid \mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}_n - c_n))}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x}_n - c_n)}$$

- When using separate bias b_k, replace $\mathbf{w}_k^T \mathbf{x}_n$ everywhere with $\mathbf{w}_k^T \mathbf{x}_n + b_k$.

Vectorization of Softmax with Separate Bias

- Separate the bias b_k from the weight vector $\mathbf{w}_k$.
- Compute gradient separately with respect to $\mathbf{w}_k$ and b_k :
  - Gradient with respect to $\mathbf{w}_k$ is:

$$\mathbf{grad}_k = -\frac{1}{N}\sum_{n=1}^{N}\left(\delta_k(t_n) - p(C_k \mid \mathbf{x}_n)\right)\mathbf{x}_n + \alpha\mathbf{w}_k$$

Gradient matrix is [$\mathbf{grad}_1$ | $\mathbf{grad}_2$ | … | $\mathbf{grad}_K$]

  - Gradient with respect to b_k is:

$$\Delta b_k = -\frac{1}{N}\sum_{n=1}^{N}(\delta_k(t_n) - p(C_k|\mathbf{x}_n))$$

Gradient vector is Δb =[Δb_1 | Δb_2 | … | Δb_K]

$$p(C_k|\mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T\mathbf{x}_n + b_k)}{\sum_{j=1..K}\exp(\mathbf{w}_j^T\mathbf{x}_n + b_j)}$$

$$\delta_k(t_n) = \begin{cases} 1\,, if\ t_n = k \\ 0\,, if\ t_n \neq k \end{cases}$$

Vectorization of Softmax

- Need to compute [*cost*, ***grad***, Δb]:

$$p(C_k|\mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T\mathbf{x}_n + b_k)}{\sum_{j=1..K}\exp(\mathbf{w}_j^T\mathbf{x}_n + b_j)}$$

  - *cost* $= -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\delta_k(t_n)\ln p(C_k \mid \mathbf{x}_n) + \frac{\alpha}{2}\sum_{k=1}^{K}\mathbf{w}_k^T\mathbf{w}_k$
  - ***grad***$_k$ $= -\frac{1}{N}\sum_{n=1}^{N}\left(\delta_k(t_n) - p(C_k \mid \mathbf{x}_n)\right)\mathbf{x}_n + \alpha\mathbf{w}_k$

=> compute ground truth matrix G such that G[k,n] = $\delta_k(t_n)$

$$\delta_k(t_n) = \begin{cases} 1\,, if\ t_n = k \\ 0\,, if\ t_n \neq k \end{cases}$$

```
from scipy.sparse import coo_matrix
groundTruth = coo_matrix((np.ones(N, dtype = np.uint8),
                          (labels, np.arange(N)))).toarray()
```

Vectorization of Softmax

- Compute $cost = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\delta_k(t_n)\ln p(C_k \mid \mathbf{x}_n) + \frac{\alpha}{2}\sum_{k=1}^{K}\mathbf{w}_k^T\mathbf{w}_k$
  - Compute matrix of $\mathbf{w}_k^T\mathbf{x}_n + b_k$.
  - Compute matrix of $\mathbf{w}_k^T\mathbf{x}_n + b_k - c_n$.
  - Compute matrix of $\exp(\mathbf{w}_k^T\mathbf{x}_n + b_k - c_n)$.
  - Compute matrix of $\ln p(C_k|\mathbf{x}_n)$.
  - Compute log-likelihood cost using all the above.

$$p(C_k|\mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T\mathbf{x}_n + b_k)}{\sum_{j=1..K}\exp(\mathbf{w}_j^T\mathbf{x}_n + b_j)}$$

$$\delta_k(t_n) = \begin{cases}1\,, if\ t_n = k\\ 0\,, if\ t_n \neq k\end{cases}$$

$$c_n = \max_{1\le k\le K}\mathbf{w}_k^T\mathbf{x}_n + b_k$$

$$\ln p(C_k|\mathbf{x}_n) = \mathbf{w}_k^T\mathbf{x}_n + b_k - \ln(\sum_{j=1..K}\exp(\mathbf{w}_j^T\mathbf{x}_n + b_j))$$

Vectorization of Softmax

- Compute $\mathbf{grad}_k = -\frac{1}{N}\sum_{n=1}^{N}\left(\delta_k(t_n) - p(C_k \mid \mathbf{x}_n)\right)\mathbf{x}_n + \alpha\mathbf{w}_k$

  - **Gradient matrix** = [**grad**$_1$ | **grad**$_2$ | … | **grad**$_\mathrm{K}$]

    - Compute matrix of $p(C_k|\mathbf{x}_n)$.

    - Compute matrix of gradient of data term.

    - Compute matrix of gradient of regularization term.

    - Compute ground truth matrix G such that G[k,n] = $\delta_k(t_n)$

$$p(C_k|\mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T\mathbf{x}_n + b_k)}{\sum_{j=1..K}\exp(\mathbf{w}_j^T\mathbf{x}_n + b_j)}$$

$$\delta_k(t_n) = \begin{cases} 1\,, if\ t_n = k \\ 0\,, if\ t_n \neq k \end{cases}$$

Vectorization of Softmax

- Useful Numpy functions:
  - np.dot()
  - np.amax()
  - np.argmax()
  - np.exp()
  - np.sum()
  - np.log()
  - np.mean()

Implementation: Gradient Checking

- Want to minimize $J(\theta)$, where θ is a scalar.
- Mathematical definition of derivative:

$$\frac{d}{d\theta}J(\theta) = \lim_{\varepsilon \to 0} \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon}$$

- Numerical approximation of derivative:

$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon}$$

where ε = 0.0001

Implementation: Gradient Checking

- If $\boldsymbol{\theta}$ is a vector of parameters θ_i,
  - Compute numerical derivative with respect to each θ_i.
    - Create a vector $\mathbf{v}$ that is ε in position *i* and 0 everywhere else:
      - *How do you do this without a for loop in NumPy?*
    - Compute $G_{\text{num}}(\theta_i) = (\mathrm{J}(\boldsymbol{\theta} + \mathbf{v}) - \mathrm{J}(\boldsymbol{\theta} - \mathbf{v})) / 2\varepsilon$
  - Aggregate all derivatives $G_{\text{num}}(\theta_i)$ into numerical gradient $G_{\text{num}}(\boldsymbol{\theta})$.

- Compare numerical gradient $G_{\text{num}}(\boldsymbol{\theta})$ with implementation of gradient $G_{\text{imp}}(\boldsymbol{\theta})$:

$$\frac{\left\| G_{num}(\boldsymbol{\theta}) - G_{imp}(\boldsymbol{\theta}) \right\|}{\left\| G_{num}(\boldsymbol{\theta}) + G_{imp}(\boldsymbol{\theta}) \right\|} \leq 10^{-6}$$