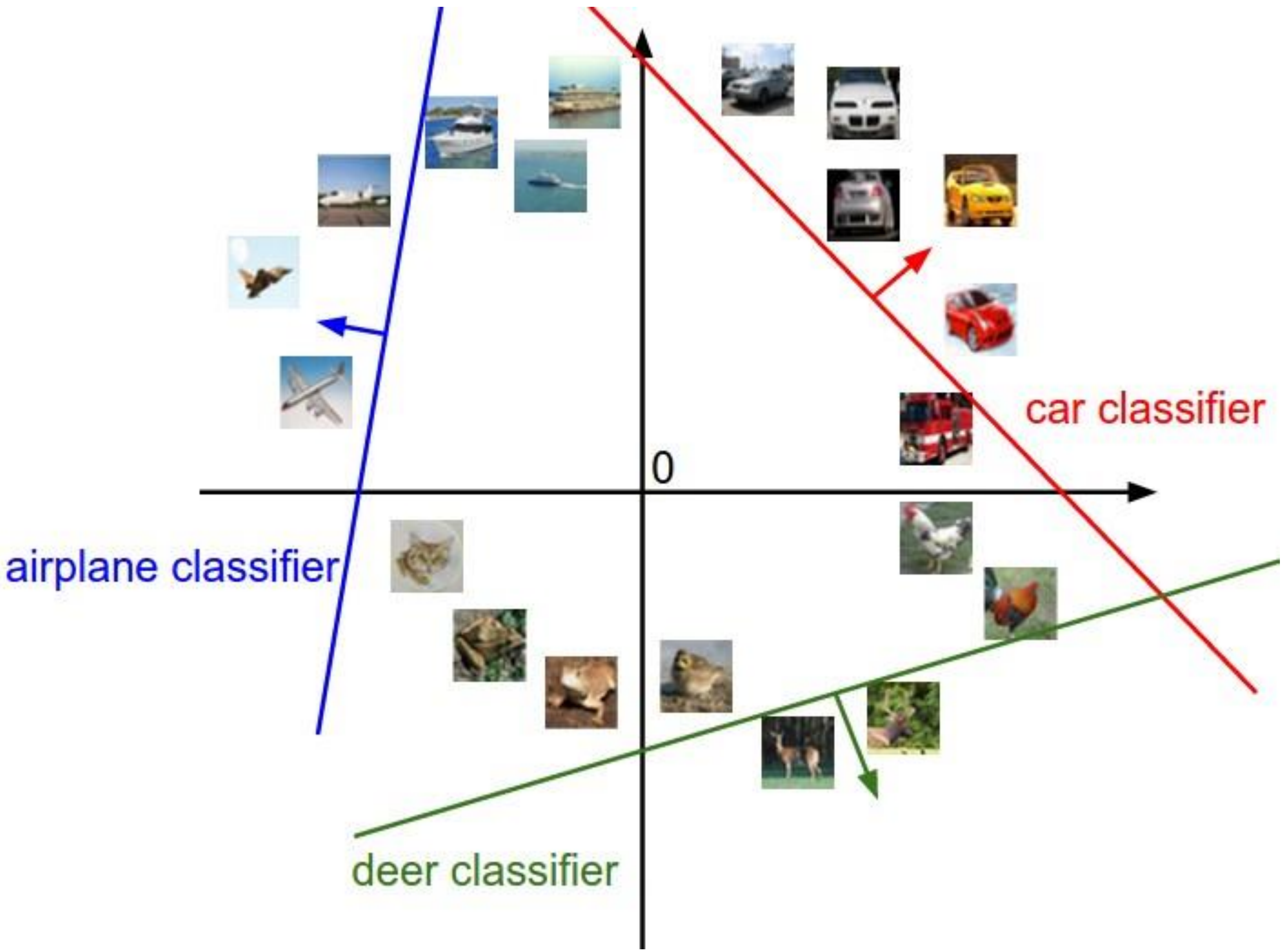
Multiclass Classification

Based on the Stanford course “CNN for visual recognition”

http://cs231n.github.io/linear-classify/#softmax

Example



Loss Function

Training data: $\{x_t, y_t\}, t = 1, \dots, n;$ $x_t \in R^d, y_t \in \{1,\dots,C\}$

Prediction: $f(x_t, \theta)$, θ- learned parameters

How do we compare the prediction $f(x_t, \theta)$ with the true label y_t ?

Loss: $L(f(x_t, \theta), y_t)$

Intuitively, the loss will be high if we're doing a poor job of classifying the training data, and it will be low if we're doing well.

MSE Loss

- Samlpe loss

$$L_i = (f(x_i) - y_i)^2$$

- Full loss

$$L = \sum_i (f(x_i) - y_i)^2$$

- Have saw this before

Multiclass SVM Loss

- The correct class for each input should have a score higher than the incorrect classes by some fixed margin Δ.
- Assume that the score of the j-th class is $s_j = \mathrm{f}(x_i, \Theta)$,
- The Multiclass SVM loss for the i-th example is then formalized as:

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

Example: suppose that we have three classes and for some x_i we received the scores $s = [13, -7, 11]$ while $y_i = 1$ and that margin is set to Δ=10.

$$L_i = \underbrace{\max(0, -7 - 13 + 10)}_{\text{0 as incorrect class score (-7) is smaller than the correct class score (13) by at least margin 10}} + \underbrace{\max(0, 11 - 13 + 10)}_{\text{8, since the difference between the correct class score and the incorrect one is smaller than the margin}}$$

Multiclass Linear SVM Loss

$$\theta \triangleq W, \qquad s_j = f(x_i, W)_j$$

$$f(x_i, W) = W x_i$$

$$W = \begin{bmatrix} w_1 \\ \cdots \\ w_c \end{bmatrix} \quad w_j \triangleq \begin{bmatrix} w_j \\ b_j \end{bmatrix} \quad x \triangleq \begin{bmatrix} x \\ 1 \end{bmatrix}$$

$$\boxed{L_i = \sum_{j \neq y_i} max\,(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)}$$

Hinge Loss: $\max(0, -)$

Regularization

- Suppose that we have a dataset and a set of parameters $\boldsymbol{W}$ that correctly classify every example and $L_i = 0$ for all i.
- For any $k > 0$, kW will also produce zero loss.
- We wish to encode some preference for a certain set of weights $\boldsymbol{W}$ over others to remove this ambiguity.
- Extend the loss with a regularization penalty

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

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W)$$

The regularization forces large margin. The formulation is equivalent to the one we saw earlier for the binary case.

Multiclass Linear SVM Loss

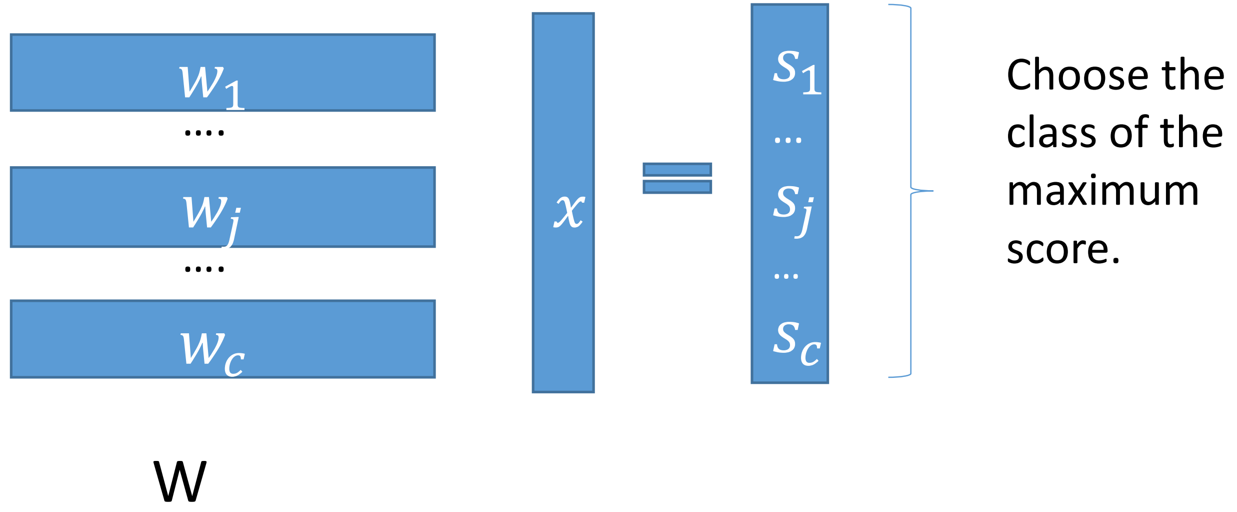
$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

- *N* is the number of training examples
- λ is a regularization parameter (There is no simple way of setting it; usually is determined by cross-validation).
- The parameters λ and Δ control the same tradeoff, thus we can safely set Δ=1.
- The magnitude of the W has direct effect on the scores and their difference: shrink W → increases the difference, increase W → decreases the difference.
- Therefore, the exact value of the margin between the scores is unimportant. The only real tradeoff is how large we allow the weights to grow (controlled by λ).

Multiclass Linear SVM Prediction

- Given the trained parameters W, we can classify an unseen input x as : $j^* = \underset{j}{\operatorname{argmax}}\, s_j$

where $s_j = f(x, W)_j$, $f(x, W) = Wx$



One vs All Multiclass SVM

- For each class *j* =1,…,C train a binary SVM, in which
  - the positive class (y=1) contains the training samples of class *j*
  - the negative class (y=-1) includes the samples of all other classes $i \neq j$.
- To classify an unseen input x, compute $s_j = w_j^T x$ for all j=1,…,C and predict the class as follows: $j^* = \underset{j}{\operatorname{argmax}}\, s_j$

All vs All trains binary classifiers for all pairs of classes – thus is computationally expensive and less popular

Softmax Function

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Squashes a vector z to a vector of values between zero and one that sum to one.

Softmax Classifier

- Multiclass SVM treats $f(x_i, W) = Wx_i$ as (uncalibrated and possibly difficult to interpret) scores for each class.
- **Softmax classifier** gives a slightly more intuitive output (normalized class probabilities) and has a probabilistic interpretation.
- The function mapping $f(x_i, W) = Wx_i$ is the same, but we interpret these scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a **cross-entropy loss** that has the form:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

or equivalently

$$L_i = -f_{y_i} + \log \sum_j e^{f_j}$$

where f_{y_i} corresponds to the j-th element of the vector of class scores f.

Full Loss:

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W)$$

Information-Theoretic View

- The *cross-entropy* between a "true" distribution *p* and an estimated distribution *q* is defined as

$$H(p, q) = -\sum_x p(x) \log q(x)$$

- The Softmax classifier minimizes the cross-entropy between the estimated class probabilities $q = e^{f_{y_i}} / \sum_j e^{f_j}$ and the "true" distribution, which in this interpretation is the distribution where all probability mass is on the correct class: $p = [0,0, \dots, 1,0,0]$ (contains a single 1 at the y_ith position)

Probabilistic Interpretation

$$P(y_i|x_i;W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

- can be interpreted as the probability assigned to the correct label y_i given the input x_i and parametrized by *W*.
- In the probabilistic interpretation, we are minimizing the negative log likelihood of the correct class, which can be interpreted as performing *Maximum Likelihood Estimation* (MLE).
- A nice feature of this view is that we can now also interpret the regularization term R(W) in the full loss function as coming from a Gaussian prior over the weight matrix *W* (we will show the derivation later on) – in that case it's *Maximum a posteriori* (MAP) estimation.

Numeric Stability

- In practice, the intermediate terms $e^{f_{y_i}}$ and $\sum_j e^{f_j}$ may be very large due to the exponentials.
- Dividing large numbers can be numerically unstable, so it is important to use a normalization trick.
- Notice that if we multiply the top and bottom of the fraction by a constant C it doesn't change the result:

$$\frac{C e^{f_{y_i}}}{C \sum_j e^{f_j}} = \frac{e^{f_{y_i} + \log C}}{\sum_j e^{f_j + \log C}}$$

A common choice for C is to set $\log C = -\max_j f_j$. This simply shifts the values inside the vector f so that the highest value is zero.