loss functions and optimization

① A loss function tells us how good our current classifier is. Given a dataset of examples

$$\{\underbrace{x_i}_{\text{image (pixels)}}, \underbrace{y_i}_{\text{label (category 1-10)}}\}_{i=1}^{N}$$

loss over the dataset is given by

$$\boxed{L = \frac{1}{N}\sum_i L_i(\underbrace{f(x_i, W)}_{\text{predicted value (label)}}, \underbrace{y_i}_{\text{True value (label)}})}$$

② multi-class SVM loss (Support Vector machines). Usually if we have 2 classes we could've used binary loss, but since here we have 10 classes, we will use multi-class SVM loss.

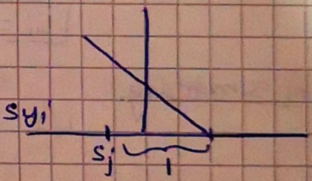
given an example (x_i, y_i) and using the shorthand for the score vector $s = f(x_i, W)$

the SVM loss has the form.

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } \underbrace{s_{y_i}}_{\text{correct score (true class)}} \geq \underbrace{s_j}_{\text{incorrect score}} + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

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

"Hinge loss"



eg.

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

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$
$$= \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1)$$
$$= \max(0, 2.9) + \max(0, -3.9)$$
$$= 2.9 + 0$$
$$= 2.9$$

→ Kind of a quantitative measure of how much our classifier screwed up in this training example.

loss over entire dataset

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i$$
$$= (2.9 + 0 + 12.9)/3$$
$$= 5.27$$

③ choosing W

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\overset{\text{hyperparameter}}{\lambda} R(W)}_{\text{Regularization loss}}$$

Data loss: model predictions should match the training data

Regularization loss: model should be simple, so that it works on test data.

Generally, we have the following Regularization types

① L1 Regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

② L2 " " $R(W) = \sum_k \sum_l W^2_{k,l}$ ← General.

③ Elastic net $(L_1 + L_2)$ $R(W) = \sum_k \sum_l \beta W^2_{k,l} + |W_{k,l}|$

④ max norm regularization.

⑤ Dropout

⑥ Fancier:

④ <u>Softmax loss</u> (multinomial logistic Regression)

scores = unnormalised log probabilities of the classes

$$\boxed{P(Y=k \mid X=x_i) = \underbrace{\frac{e^{s_k}}{\sum_j e^{s_j}}}_{\text{softmax function}}} \qquad \boxed{s = f(x_i; W)}$$

we want to maximise the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class.

$$\boxed{L_i = -\log P(Y = y_i \mid X = x_i)}$$

in summary:

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

Eg.	unnormalised log probabilities		unnormalised probabilities		probabilities	
Cat	3.2	$\xrightarrow{exp}$	24.5	$\xrightarrow{\text{normalise}}$	0.13	$L_i = -\log(0.13) = \underline{0.89}$
Car	5.1		164.0		0.87	
Frog	-1.7		0.18		0.00	

Recap

① we have some dataset of (x, y)

② we have a score function

③ we have a loss function.

full loss → $$\boxed{L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)}$$

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

(softmax) (SVM)

Q/ How do we find "W" which minimises the loss?
→ optimization.

→ following the slope #.

slope → In 1-d, the derivative of a function

$$\frac{df(x)}{dx} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

→ In multiple dimensions, the gradient is the vector of (partial-derivatives) along each dimension.

→ The slope in any direction is the dot product of the direction with the gradient. The direction of the steepest descent is the negative gradient.

2) Numerical gradient → approximate, slow, easy to write.
Analytical gradient → exact, fast, error-prone.

Hint → Always use analytical gradient, but check implementation with numerical gradient. This is called "gradient check".

3) Gradient Descent → algorithm where we use the gradient at every timestep to determine where to step next.

Stochastic gradient Descent → when calculating the loss, or the gradient descent, the full sum is expensive when N becomes very large, like:

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

instead, we approximate the sum using "minibatch" of examples 32/64/12

$$\boxed{\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)}$$