L2 regularization and momentum

(Super-short and extremely simplified, see lectures for more.)

L2-regularization is adding $\lambda \sum w^2$ to loss (for some tiny $\lambda \in \mathbb{R}$).
For basic SGD: same as decreasing weights by λw_i at every SGD step.
This encourages weights to be small, which can prevent overfitting.
This is skipped for biases.

Momentum means replacing the update step

$$\bar{\theta} \leftarrow \bar{\theta} - \varepsilon(\nabla_{\bar{\theta}} \mathcal{L} + \lambda \bar{\theta})$$

with:

$$\bar{v}_{\bar{\theta}} \leftarrow \mu \bar{v}_{\bar{\theta}} + \nabla_{\bar{\theta}} \mathcal{L} + \lambda \bar{\theta}$$

$$\bar{\theta} \leftarrow \bar{\theta} - \varepsilon \bar{v}_{\bar{\theta}}$$

where e.g.: learning rate $\varepsilon = 10^{-3}$, momentum $\mu = 0.99$, weight decay $\lambda = 10^{-4}$ ($\lambda = 0$ for biases).

softmax

In the last layer, instead of

$$g_i = \sigma(f_i) = \frac{\exp(f_i)}{1 + \exp(f_i)}$$

we can use

$$\bar{g} = \mathrm{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

It gives a probability distribution ($\sum g_i = 1$, $g_i \geq 0$).

softmax

In the last layer, instead of

$$g_i = \sigma(f_i) = \frac{\exp(f_i)}{1 + \exp(f_i)}$$

we can use

$$\bar{g} = \mathrm{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

It gives a probability distribution ($\sum g_i = 1$, $g_i \geq 0$).

For $N = 2$ classes, $\mathrm{softmax}(\bar{f}) = (\sigma(d), 1 - \sigma(d))$ where $d = f_0 - f_1$.

softmax

In the last layer, instead of

$$g_i = \sigma(f_i) = \frac{\exp(f_i)}{1 + \exp(f_i)}$$

we can use

$$\bar{g} = \mathrm{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

It gives a probability distribution ($\sum g_i = 1$, $g_i \geq 0$).

When outputs are probabilities (sigmoid or softmax), instead of MSE, better to use *cross-entropy* (CE) or *negative log-likelihood* (NLL) loss.

NLL loss is $-\ln g_\ell$ (maximize probability for the target label ℓ).
CE loss is $-\sum_i y_i \ln g_i$ (same as NLL if $\bar{y}$ is the one-hot encoded label).
(Named because it is the cross-entropy $H(p, q) = -\sum p_i \ln q_i$.)

Why negative log loss?

NLL can be derived from the "maximum likelihood estimation" principle, an idea in statistical modeling.

Let's start with a different task: modeling a distribution X (e.g. coming from a hypothetical random process that outputs an image of a cat). We want a model that outputs the estimated probability $p_{\text{model}}(x)$ of any data point x (any image). So the model should tell us how likely it is to get x by sampling an image from X, or "is that a typical cat image, how typical?"

The model has parameters θ: weights and biases in a neural net. We want to optimize θ knowing a observed/training set of images $\{x^{(1)}, x^{(2)}, ...\}$ sampled from X independenty.

Maximum likelihood estimation

Model with parameters θ outputs $p_{\text{model}}(x)$, an estimation of $p(x)$.
We only have $\text{OBSERVS} = \{x^{(1)}, x^{(2)}, \dots\}$ sampled from $X \sim p(x)$.
How do we choose θ, what do we want to maximize?
The "maximum likelihood estimation" principle says:

maximize the *likelihood*.

Likelihood (in this context and in a lot of statistics) means the joint probability that the model assigns to the TRAIN set, as a function of θ. Since we sampled independently, this is

$$\text{likelihood}(\theta) = \prod_{x \in \text{OBSERVS}} p_{\text{model}}(x) = p_{\text{model}}(x^{(1)}) \cdot p_{\text{model}}(x^{(2)}) \cdot \dots$$

This is the same as choosing θ to minimize

$$\text{NLL} = -\log \text{likelihood} = \sum_{x \in \text{OBSERVS}} -\log p_{\text{model}}(x)$$

Why max likelihood?

You can just say you want to do "maximum likelihood estimation", take it as an axiom.
You could also say "I want to minimize $D_{KL}(p_{\text{obs}} \| p_{\text{model}})$".

$$= \mathbb{E}_{x \sim p_{\text{obs}}}[\log p_{\text{obs}} - \log p_{\text{model}}]$$

Since p_{obs} is constant w.r.t. θ, it's the same as minimizing NLL

$$= \mathbb{E}_{x \sim p_{\text{obs}}}[-\log p_{\text{model}}] = \frac{1}{N} \sum_{x \in \text{OBSERVED}} -\log p_{\text{model}}(x)$$

You *could* instead say "I want to minimize $MSE(p_{\text{model}}, p_{\text{obs}})$", if for you the difference between $p(x) = 0$ and 0.001 matters less than between 0.01 and 0.1. Sometimes that's what you want.
But often a model saying $p(x) = 0$ (I'm *infinitely* sure this is not a cat) is catastrophically bad (e.g. if you bet on that information).

Cross-entropy

When minimizing

$$NLL = \sum_{x \in \text{OBSERVS}} -\log p_{\text{model}}(x)$$

the frequency with which any x appears in OBSERVS matters.
Denoting the (empirical, sampled) frequency of x by $p_{\text{obs}}(x)$:

$$= \sum_{x \in \mathbb{R}^n} p_{\text{obs}}(x) \cdot (-\log p_{\text{model}}(x))$$

This is the *cross-entropy*, usually denoted $H(p_{\text{obs}}, p_{\text{model}})$.
Note it's not symmetric. It's equal to $D_{KL}(p_{\text{obs}} \| p_{\text{model}}) + H(p_{\text{obs}})$.
So minimizing NLL, CE, or $D_{KL}(p_{\text{obs}} \| p_{\text{model}})$ is equivalent here.

Max likelihood for a function

When modeling a function like image $\mapsto$ class, we observe a joint distribution of input-output pairs (x, y). The model should output conditional probabilities $p_{\text{model}}(y \mid x)$. The model usually takes input x and returns a softmax vector of probabilities for all y, ($model(x) = \bar{g}$ and $g_y = p_{\text{model}}(y \mid x)$).
Other than that, it's all similar to the previous case.

Likelihood is $\prod_{(x,y)\in \text{OBSERVS}} P(x, y) = \prod_{(x,y)} p_{\text{model}}(y|x)P(x) = \exp\left(\sum_{(x,y)} \log p_{\text{model}}(y|x) + \log P(x)\right)$
If we're not trying to model the distribution of inputs, then $P(x)$ is constant, so maximizing likelihood is the same as minimizing

$$NLL = \sum_{(x,y)\in \text{OBSERVS}} -\log p_{\text{model}}(y \mid x) = \sum_{(x,y)\in \text{OBSERVS}} -\log g_y$$

where g is the vector returned by softmax.

NLL and CE in torch

Names "negative log likelihood" and "cross-entropy" are usually used interchangeably.
In PyTorch, the difference between them is unrelated to names:

- NLLLoss takes input logarithms of probabilities (so logarithms of what softmax outputs, usually) and a target label,
- CrossEntropyLoss is the same as a LogSoftmax layer followed by NLLLoss: it takes logits (the last pre-activations, or what would be given to softmax) and a target: as a label or as a one-hot-encoded vector or as an arbitrary probability vector.

Usually models return logits (no softmax in $\mathrm{forward}()$, you need to call it yourself if you want probabilities!) and training uses CrossEntropyLoss.

Sometimes (e.g. in sci-kit) log likelihood is maximized, instead of minimizing NLL. But also often "negative" is skipped from the name just for convenience, even when negation *is* used.

Why NLL – final word

When you actually use the probabilities, you usually want their logarithms to be accurate, think: *information*, *ratios of probabilities* in Bayesian inference, amounts of certainty, etc.

Often though you really only care about e.g. accuracy (percentage of correct outputs). NLL is usually still a good choice for training (because it's differentiable, also maybe because you want the model to learn what's certain and what's uncertain). But you might want to monitor several test metrics.

log softmax

Usually you don't need probabilities but their logarithms:
as in NLL loss and statistical modeling.
You also want numerical precision for small probabilities.

Log of softmax is just a kind of normalization of log-probabilities:
you apply $\exp$ to $\bar{f}$, normalize it to sum to 1, then apply $\log$.

Note

$$\log \text{softmax}(\bar{g})_i = f_i - \text{const}, \qquad \text{where const} = \log(\textstyle\sum_j \exp(f_j))$$

So **logits** f_i tell you a lot: they have the same argmax and their differences are logarithms of probability ratios:
if $\text{softmax}(\bar{f}) = \bar{p}$, then $f_i - f_j = \log \frac{p_i}{p_j} = (\log p_i) - (\log p_j)$.
If $f_i - f_j = +1$ then i is e times more likely than j.

softmax gradients

For softmax outputs and CE / NLL loss, on a sample with true label ℓ:

$$\bar{y} = \mathbb{1}_\ell, \quad \bar{g} = \mathrm{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

$$\mathcal{L} = -\bar{y} \cdot \ln \bar{g} = -\ln g_\ell = -f_\ell + \ln\left(\sum_j \exp(f_j)\right)$$

softmax gradients

For softmax outputs and CE / NLL loss, on a sample with true label ℓ:

$$\bar{y} = \mathbb{1}_\ell, \quad \bar{g} = \mathrm{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

$$\mathcal{L} = -\bar{y} \cdot \ln \bar{g} = -\ln g_\ell = -f_\ell + \ln\left(\sum_j \exp(f_j)\right)$$

Trick: skip computing $\nabla_{\bar{g}^{(L)}}\mathcal{L}$, compute $\nabla_{\bar{f}^{(L)}}\mathcal{L}$ directly, avoiding the chain rule.

$$\frac{\partial \mathcal{L}}{\partial f_i} = \frac{\partial}{\partial f_i}\left(-f_\ell + \ln\left(\sum_j \exp(f_j)\right)\right) =$$

softmax gradients

For softmax outputs and CE / NLL loss, on a sample with true label ℓ:

$$\bar{y} = \mathbb{1}_\ell, \quad \bar{g} = \mathrm{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

$$\mathcal{L} = -\bar{y} \cdot \ln \bar{g} = -\ln g_\ell = -f_\ell + \ln\left(\sum_j \exp(f_j)\right)$$

Trick: skip computing $\nabla_{\bar{g}^{(L)}}\mathcal{L}$, compute $\nabla_{\bar{f}^{(L)}}\mathcal{L}$ directly, avoiding the chain rule.

$$\frac{\partial \mathcal{L}}{\partial f_i} = \frac{\partial}{\partial f_i}\left(-f_\ell + \ln\left(\sum_j \exp(f_j)\right)\right) =$$

$$= -\mathbb{1}[i = \ell] +$$

softmax gradients

For softmax outputs and CE / NLL loss, on a sample with true label ℓ:

$$\bar{y} = \mathbb{1}_\ell, \quad \bar{g} = \mathrm{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

$$\mathcal{L} = -\bar{y} \cdot \ln \bar{g} = -\ln g_\ell = -f_\ell + \ln\left(\sum_j \exp(f_j)\right)$$

Trick: skip computing $\nabla_{\bar{g}^{(L)}}\mathcal{L}$, compute $\nabla_{\bar{f}^{(L)}}\mathcal{L}$ directly, avoiding the chain rule.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial f_i} &= \frac{\partial}{\partial f_i}\left(-f_\ell + \ln\left(\sum_j \exp(f_j)\right)\right) = \\ &= -\mathbb{1}[i=\ell] + \frac{\frac{\partial}{\partial f_i}\left(\sum_j \exp(f_j)\right)}{\sum_j \exp(f_j)} =\end{aligned}$$

softmax gradients

For softmax outputs and CE / NLL loss, on a sample with true label ℓ:

$$\bar{y} = \mathbb{1}_\ell, \quad \bar{g} = \mathrm{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

$$\mathcal{L} = -\bar{y} \cdot \ln \bar{g} = -\ln g_\ell = -f_\ell + \ln\left(\sum_j \exp(f_j)\right)$$

Trick: skip computing $\nabla_{\bar{g}^{(L)}}\mathcal{L}$, compute $\nabla_{\bar{f}^{(L)}}\mathcal{L}$ directly, avoiding the chain rule.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial f_i} &= \frac{\partial}{\partial f_i}\left(-f_\ell + \ln\left(\sum_j \exp(f_j)\right)\right) = \\ &= -\mathbb{1}[i=\ell] + \frac{\frac{\partial}{\partial f_i}\left(\sum_j \exp(f_j)\right)}{\sum_j \exp(f_j)} = -y_i + g_i\end{aligned}$$

$$\nabla_{\bar{f}^{(L)}}\mathcal{L} = \bar{g}^{(L)} - \bar{y}, \quad \text{simple and no saturation}$$

softmax numerically

$$\bar{g} = \text{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

Computing $\text{np.exp}(1000)$ won't work.

softmax numerically

$$\bar{g} = \text{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

Computing $\text{np.exp}(1000)$ won't work.

Replacing $\bar{f}$ with $\bar{f} - c$ for any c gives the same result.

softmax numerically

$$\bar{g} = \text{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

Computing $\text{np.exp}(1000)$ won't work.

Replacing $\bar{f}$ with $\bar{f} - c$ for any c gives the same result.

Use $c = \max \bar{f}$.

Instead of dividing by sums of exponentials:

$$g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)} =$$

softmax numerically

$$\bar{g} = \text{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

Computing $\text{np.exp}(1000)$ won't work.

Replacing $\bar{f}$ with $\bar{f} - c$ for any c gives the same result.

Use $c = \max \bar{f}$.

Instead of dividing by sums of exponentials:

$$g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)} = \exp\Big(\qquad\qquad \Big)$$

softmax numerically

$$\bar{g} = \text{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

Computing $\text{np.exp}(1000)$ won't work.

Replacing $\bar{f}$ with $\bar{f} - c$ for any c gives the same result.

Use $c = \max \bar{f}$.

Instead of dividing by sums of exponentials:

$$g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)} = \exp\left(f_i - \ln\left(\sum_j \exp(f_j)\right)\right)$$

softmax numerically

$$\bar{g} = \mathrm{softmax}(\bar{f}), \quad g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}$$

Computing $\mathrm{np.exp}(1000)$ won't work.

Replacing $\bar{f}$ with $\bar{f} - c$ for any c gives the same result.

Use $c = \max \bar{f}$.

Instead of dividing by sums of exponentials:

$$g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)} = \exp\left(f_i - \ln\left(\sum_j \exp(f_j)\right)\right)$$

Altogether:

$$g_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)} = \exp\left(f_i - c - \ln\left(\sum_j \exp(f_j - c)\right)\right)$$