Deep learning

5.1. Cross-entropy loss

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ DE GENÈVE

The usual form of a classification training set is

$$(x_n, y_n) \in \mathbb{R}^D \times \{1, \dots, C\},\ n = 1, \dots, N.$$

We can train on such a data-set with a regression loss such as the MSE using a "one-hot vector" encoding: that converts labels into a tensor $z \in \mathbb{R}^{N \times C}$, with

$$\forall n,\ z_{n,m} = \begin{cases} 1 & \text{if } \ m = y_n \\ 0 & \text{otherwise.} \end{cases}$$

The usual form of a classification training set is

$$(x_n, y_n) \in \mathbb{R}^D \times \{1, \dots, C\}, \; n = 1, \dots, N.$$

We can train on such a data-set with a regression loss such as the MSE using a "one-hot vector" encoding: that converts labels into a tensor $z \in \mathbb{R}^{N \times C}$, with

$$\forall n, \; z_{n,m} = \begin{cases} 1 & \text{if } \; m = y_n \\ 0 & \text{otherwise}. \end{cases}$$

For instance, with $N = 5$ and $C = 3$, we would have

$$N \updownarrow \begin{pmatrix} 2 \\ 1 \\ 1 \\ 3 \\ 2 \end{pmatrix} \xrightarrow{\text{One-hot encoding}} \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}}_{\longleftrightarrow \; C}$$

The usual form of a classification training set is

$$(x_n, y_n) \in \mathbb{R}^D \times \{1, \dots, C\}, \; n = 1, \dots, N.$$

We can train on such a data-set with a regression loss such as the MSE using a "one-hot vector" encoding: that converts labels into a tensor $z \in \mathbb{R}^{N \times C}$, with

$$\forall n, \; z_{n,m} = \begin{cases} 1 & \text{if } \; m = y_n \\ 0 & \text{otherwise.} \end{cases}$$

For instance, with $N = 5$ and $C = 3$, we would have

$$N \updownarrow \begin{pmatrix} 2 \\ 1 \\ 1 \\ 3 \\ 2 \end{pmatrix} \xrightarrow{\text{One-hot encoding}} \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}}_{\longleftrightarrow \; C}$$

This can be done with `F.one_hot`.

However, MSE is justified with a Gaussian noise around a target value that makes sense geometrically. Beside being conceptually wrong for classification, in practice it penalizes responses "too strongly on the right side".

However, MSE is justified with a Gaussian noise around a target value that makes sense geometrically. Beside being conceptually wrong for classification, in practice it penalizes responses "too strongly on the right side".

Consider this example with correct class 1, and two outputs $\hat{y}$ and $\hat{y}'$.

$$y \qquad\qquad \hat{y} \qquad\qquad \hat{y}'$$

$$\begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 2 & -1 & -1 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$$

Both $\hat{y}$ and $\hat{y}'$ have a MSE of 1, even though the $\hat{y}$ leads to a perfect prediction, and the $\hat{y}'$ to a perfectly wrong one.

However, MSE is justified with a Gaussian noise around a target value that makes sense geometrically. Beside being conceptually wrong for classification, in practice it penalizes responses "too strongly on the right side".

Consider this example with correct class 1, and two outputs $\hat{y}$ and $\hat{y}'$.

$$y \qquad \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \qquad \hat{y} \qquad \begin{pmatrix} 2 & -1 & -1 \end{pmatrix} \qquad \hat{y}' \qquad \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$$

Both $\hat{y}$ and $\hat{y}'$ have a MSE of 1, even though the $\hat{y}$ leads to a perfect prediction, and the $\hat{y}'$ to a perfectly wrong one.

The criterion of choice for classification is the **cross-entropy**, which fixes these inconsistencies.

We can generalize the logistic regression to a multi-class setup with $f_1, \dots, f_C$ functionals that we interpret as **logits**

$$P(Y = y \mid X = x, W = w) = \frac{1}{Z} \exp f_y(x; w) = \frac{\exp f_y(x; w)}{\sum_k \exp f_k(x; w)}$$

We can generalize the logistic regression to a multi-class setup with $f_1, \dots, f_C$ functionals that we interpret as **logits**

$$P(Y = y \mid X = x, W = w) = \frac{1}{Z} \exp f_y(x; w) = \frac{\exp f_y(x; w)}{\sum_k \exp f_k(x; w)},$$

from which

$$\begin{aligned}
&\log \mu_W(w \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid W = w)\, \mu_W(w)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid W = w) + \log \mu_W(w) - \log Z \\
&\qquad = \sum_n \log \mu_{\mathscr{D}}(x_n, y_n \mid W = w) + \log \mu_W(w) - \log Z \\
&\qquad = \sum_n \log P(Y = y_n \mid X = x_n, W = w) + \log \mu_W(w) - \log Z' \\
&\qquad = \sum_n \log \left(\frac{\exp f_{y_n}(x; w)}{\sum_k \exp f_k(x; w)} \right) + \log \mu_W(w) - \log Z'.
\end{aligned}$$

We can generalize the logistic regression to a multi-class setup with $f_1, \dots, f_C$ functionals that we interpret as **logits**

$$P(Y = y \mid X = x, W = w) = \frac{1}{Z} \exp f_y(x; w) = \frac{\exp f_y(x; w)}{\sum_k \exp f_k(x; w)},$$

from which

$$\begin{aligned}
\log \mu_W(w \mid \mathscr{D} = \mathbf{d}) \\
&= \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid W = w)\, \mu_W(w)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&= \log \mu_{\mathscr{D}}(\mathbf{d} \mid W = w) + \log \mu_W(w) - \log Z \\
&= \sum_n \log \mu_{\mathscr{D}}(x_n, y_n \mid W = w) + \log \mu_W(w) - \log Z \\
&= \sum_n \log P(Y = y_n \mid X = x_n, W = w) + \log \mu_W(w) - \log Z' \\
&= \underbrace{\sum_n \log \left(\frac{\exp f_{y_n}(x; w)}{\sum_k \exp f_k(x; w)} \right)}_{\text{Depends on the outputs}} + \log \mu_W(w) - \log Z'.
\end{aligned}$$

We can generalize the logistic regression to a multi-class setup with $f_1, \dots, f_C$ functionals that we interpret as **logits**

$$P(Y = y \mid X = x, W = w) = \frac{1}{Z} \exp f_y(x; w) = \frac{\exp f_y(x; w)}{\sum_k \exp f_k(x; w)},$$

from which

$$\begin{aligned}
&\log \mu_W(w \mid \mathscr{D} = \mathbf{d}) \\
&\quad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid W = w)\, \mu_W(w)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\quad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid W = w) + \log \mu_W(w) - \log Z \\
&\quad = \sum_n \log \mu_{\mathscr{D}}(x_n, y_n \mid W = w) + \log \mu_W(w) - \log Z \\
&\quad = \sum_n \log P(Y = y_n \mid X = x_n, W = w) + \log \mu_W(w) - \log Z' \\
&\quad = \underbrace{\sum_n \log \left(\frac{\exp f_{y_n}(x; w)}{\sum_k \exp f_k(x; w)} \right)}_{\text{Depends on the outputs}} + \underbrace{\log \mu_W(w)}_{\text{Depends on } w} - \log Z'.
\end{aligned}$$

If we ignore the penalty on w, it makes sense to minimize the average

$$\mathscr{L}(w) = -\frac{1}{N} \sum_{n=1}^{N} \log \underbrace{\left(\frac{\exp f_{y_n}(x_n; w)}{\sum_k \exp f_k(x_n; w)} \right)}_{\hat{P}_w(Y=y_n | X=x_n)}.$$

If we ignore the penalty on w, it makes sense to minimize the average

$$\mathscr{L}(w) = -\frac{1}{N}\sum_{n=1}^{N} \log \underbrace{\left(\frac{\exp f_{y_n}(x_n; w)}{\sum_k \exp f_k(x_n; w)}\right)}_{\hat{P}_w(Y=y_n|X=x_n)}.$$

Given two distributions p and q, their **cross-entropy** is defined as

$$\mathbb{H}(p, q) = -\mathbb{E}_p\left[\log q\right] = -\sum_k p(k) \log q(k),$$

with the convention that $0 \log 0 = 0$.

If we ignore the penalty on w, it makes sense to minimize the average

$$\mathscr{L}(w) = -\frac{1}{N}\sum_{n=1}^{N} \log \underbrace{\left(\frac{\exp f_{y_n}(x_n; w)}{\sum_k \exp f_k(x_n; w)} \right)}_{\hat{P}_w(Y=y_n|X=x_n)}.$$

Given two distributions p and q, their **cross-entropy** is defined as

$$\mathbb{H}(p, q) = -\mathbb{E}_p\left[\log q\right] = -\sum_k p(k) \log q(k),$$

with the convention that $0 \log 0 = 0$. So we can re-write

$$\begin{aligned} -\log\left(\frac{\exp f_{y_n}(x_n; w)}{\sum_k \exp f_k(x_n; w)} \right) &= -\log \hat{P}_w(Y = y_n \mid X = x_n) \\ &= -\sum_k \delta_{y_n}(k) \log \hat{P}_w(Y = k \mid X = x_n) \\ &= \mathbb{H}\left(\delta_{y_n}, \hat{P}_w(Y = \cdot \mid X = x_n)\right). \end{aligned}$$

If we ignore the penalty on w, it makes sense to minimize the average

$$\mathscr{L}(w) = -\frac{1}{N}\sum_{n=1}^{N} \log \underbrace{\left(\frac{\exp f_{y_n}(x_n; w)}{\sum_k \exp f_k(x_n; w)}\right)}_{\hat{P}_w(Y=y_n|X=x_n)}.$$

Given two distributions p and q, their **cross-entropy** is defined as

$$\mathbb{H}(p, q) = -\mathbb{E}_p\big[\log q\big] = -\sum_k p(k) \log q(k),$$

with the convention that $0 \log 0 = 0$. So we can re-write

$$\begin{aligned}
-\log\left(\frac{\exp f_{y_n}(x_n; w)}{\sum_k \exp f_k(x_n; w)}\right) &= -\log \hat{P}_w(Y = y_n \mid X = x_n) \\
&= -\sum_k \delta_{y_n}(k) \log \hat{P}_w(Y = k \mid X = x_n) \\
&= \mathbb{H}\left(\delta_{y_n}, \hat{P}_w(Y = \cdot \mid X = x_n)\right).
\end{aligned}$$

So $\mathscr{L}$ above is the average of the cross-entropy between the deterministic "true" posterior δ_{y_n} and the estimated $\hat{P}_w(Y = \cdot \mid X = x_n)$.

This is what `torch.nn.CrossEntropyLoss` computes.

This is what `torch.nn.CrossEntropyLoss` computes.

```
>>> f = torch.tensor([[-1., -3., 4.], [-3., 3., -1.]])
>>> target = torch.tensor([0, 1])
>>> criterion = torch.nn.CrossEntropyLoss()
>>> criterion(f, target)
tensor(2.5141)
```

and indeed

$$-\frac{1}{2}\left(\log \frac{e^{-1}}{e^{-1}+e^{-3}+e^{4}} + \log \frac{e^{3}}{e^{-3}+e^{3}+e^{-1}}\right) \simeq 2.5141.$$

This is what `torch.nn.CrossEntropyLoss` computes.

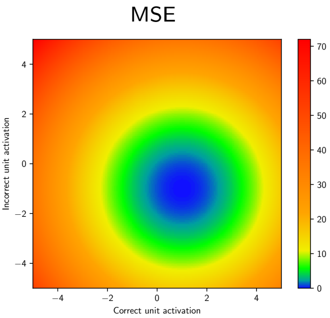
```
>>> f = torch.tensor([[-1., -3., 4.], [-3., 3., -1.]])
>>> target = torch.tensor([0, 1])
>>> criterion = torch.nn.CrossEntropyLoss()
>>> criterion(f, target)
tensor(2.5141)
```

and indeed

$$-\frac{1}{2}\left(\log\frac{e^{-1}}{e^{-1}+e^{-3}+e^{4}}+\log\frac{e^{3}}{e^{-3}+e^{3}+e^{-1}}\right)\simeq 2.5141.$$

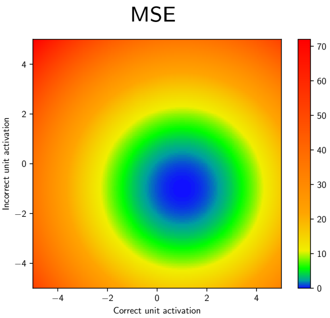
The range of values is 0 for perfectly classified samples, $\log(C)$ if the posterior is uniform, and up to $+\infty$ if the posterior distribution is "worse" than uniform.

Let's consider the loss for a single sample in a two-class problem, with a predictor with two output values.



$$\mathscr{L} = (x-1)^2 + (y+1)^2$$

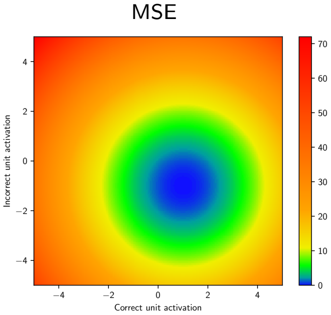
Let's consider the loss for a single sample in a two-class problem, with a predictor with two output values.

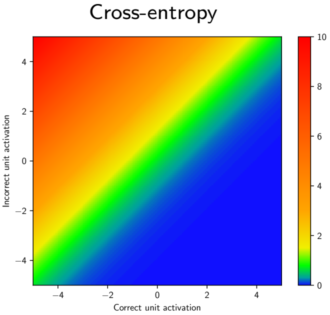


$$\mathscr{L} = (x-1)^2 + (y+1)^2$$

MSE incorrectly penalizes outputs which are perfectly valid for prediction

Let's consider the loss for a single sample in a two-class problem, with a predictor with two output values.





$$\mathscr{L} = (x-1)^2 + (y+1)^2$$

$$\mathscr{L} = -\log\left(\frac{e^x}{e^x+e^y}\right)$$

MSE incorrectly penalizes outputs which are perfectly valid for prediction, contrary to cross-entropy.

The cross-entropy loss can be seen as the composition of a “log soft-max” to normalize the [logit] scores into logs of probabilities

$$(\alpha_1, \dots, \alpha_C) \mapsto \left(\log \frac{\exp \alpha_1}{\sum_k \exp \alpha_k}, \dots, \log \frac{\exp \alpha_C}{\sum_k \exp \alpha_k} \right),$$

which can be done with the `torch.nn.LogSoftmax` module

The cross-entropy loss can be seen as the composition of a “log soft-max” to normalize the [logit] scores into logs of probabilities

$$(\alpha_1, \dots, \alpha_C) \mapsto \left(\log \frac{\exp \alpha_1}{\sum_k \exp \alpha_k}, \dots, \log \frac{\exp \alpha_C}{\sum_k \exp \alpha_k} \right),$$

which can be done with the `torch.nn.LogSoftmax` module, and a read-out of the normalized score of the correct class

$$\mathscr{L}(w) = -\frac{1}{N} \sum_{n=1}^{N} f_{y_n}(x_n; w),$$

which is implemented by the `torch.nn.NLLLoss` criterion.

The cross-entropy loss can be seen as the composition of a “log soft-max” to normalize the [logit] scores into logs of probabilities

$$(\alpha_1, \dots, \alpha_C) \mapsto \left(\log \frac{\exp \alpha_1}{\sum_k \exp \alpha_k}, \dots, \log \frac{\exp \alpha_C}{\sum_k \exp \alpha_k}\right),$$

which can be done with the `torch.nn.LogSoftmax` module, and a read-out of the normalized score of the correct class

$$\mathscr{L}(w) = -\frac{1}{N}\sum_{n=1}^{N} f_{y_n}(x_n; w),$$

which is implemented by the `torch.nn.NLLLoss` criterion.

```
>>> f = torch.tensor([[-1., -3., 4.], [-3., 3., -1.]])
>>> target = torch.tensor([0, 1])
>>> model = nn.LogSoftmax(dim = 1)
>>> criterion = torch.nn.NLLLoss()
>>> criterion(model(f), target)
tensor(2.5141)
```

Hence, if a network should compute log-probabilities, it may have a `torch.nn.LogSoftmax` final layer, and be trained with `torch.nn.NLLLoss`.

The mapping

$$(\alpha_1, \dots, \alpha_C) \mapsto \left(\frac{\exp \alpha_1}{\sum_k \exp \alpha_k}, \dots, \frac{\exp \alpha_C}{\sum_k \exp \alpha_k} \right)$$

is called soft-max since it computes a "soft arg-max Boolean label."

The mapping

$$(\alpha_1, \dots, \alpha_C) \mapsto \left(\frac{\exp \alpha_1}{\sum_k \exp \alpha_k}, \dots, \frac{\exp \alpha_C}{\sum_k \exp \alpha_k} \right)$$

is called soft-max since it computes a “soft arg-max Boolean label.”

```
>>> y = torch.tensor([[-10., -10., 10., -5. ],
...                   [  3.,   0.,  0.,  0. ],
...                   [  1.,   2.,  3.,  4. ]])
>>> f = torch.nn.Softmax(1)
>>> f(y)
tensor([[ 2.0612e-09,  2.0612e-09,  1.0000e+00,  3.0590e-07],
        [ 8.7005e-01,  4.3317e-02,  4.3317e-02,  4.3317e-02],
        [ 3.2059e-02,  8.7144e-02,  2.3688e-01,  6.4391e-01]])
```

PyTorch provides many other criteria, among which

- `torch.nn.MSELoss`
- `torch.nn.CrossEntropyLoss`
- `torch.nn.NLLLoss`
- `torch.nn.L1Loss`
- `torch.nn.NLLLoss2d`
- `torch.nn.MultiMarginLoss`

The end