Deep learning

5.4. L_2 and L_1 penalties

François Fleuret

https://fleuret.org/dlc/

We have motivated the use of a loss with a Bayesian formulation combining the probability of the data given the model and the probability of the model

$$\log \mu_W(w \mid \mathscr{D} = \mathbf{d}) = \log \mu_{\mathscr{D}}(\mathbf{d} \mid W = w) + \log \mu_W(w) - \log Z.$$

If μ_W is a Gaussian density with a covariance matrix proportional to the identity, the log-prior $\log \mu_W(w)$ results in a quadratic penalty

$$\lambda \|w\|_2^2 = \lambda \sum_i w_i^2.$$

Since this penalty is convex, its sum with a convex functional is convex.

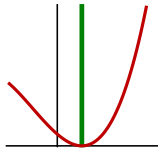
This is called the L_2 regularization, or "weight decay" in the artificial neural network community.

Notes

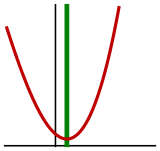
The main effect of L_2 regularization is to keep the parameters near zero and to make them less dependent on the data. Using such a penalty leads to higher training error but a lesser gap between training and test errors.

Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss alone.

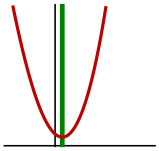
Since the derivative of $\|x\|_2^2$ is zero at zero, the optimal will never move there if it was not already there.



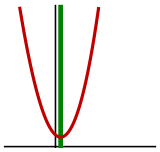
$(x-1)^2 + \frac{1}{6}(x-1)^3$



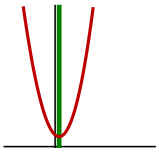
$(x-1)^2 + \frac{1}{6}(x-1)^3 + x^2$



$(x-1)^2 + \frac{1}{6}(x-1)^3 + 2x^2$



$(x-1)^2 + \frac{1}{6}(x-1)^3 + 3x^2$



$(x-1)^2 + \frac{1}{6}(x-1)^3 + 4x^2$

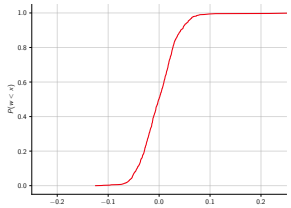
Notes

The red curve is a a function to illustrate the behavior of increasing the L_2 penalty term.
The green line shows the position of the minimum.
Increasing the L_2 penalty moves the optimum closer to zero, but it will never reach it.

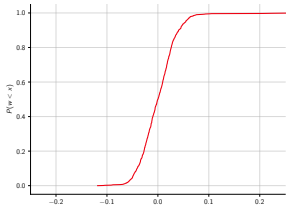
Convnet trained on MNIST with $1,000$ samples and a L_2 penalty.

	Error	
λ	Train	Test
0.000	0.000	0.064
0.001	0.000	0.063
0.002	0.000	0.064
0.004	0.005	0.065
0.010	0.022	0.075
0.020	0.048	0.101

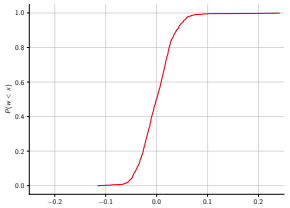
```
output = model(train_input[b:b+batch_size])
loss = criterion(output, train_target[b:b+batch_size])

for p in model.parameters():
    loss += lambda_l2 * p.pow(2).sum()

optimizer.zero_grad()
loss.backward()
optimizer.step()
```



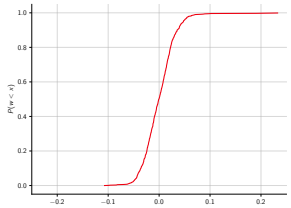
$\lambda = 0.000$



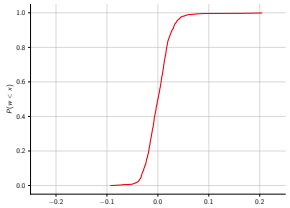
$\lambda = 0.001$



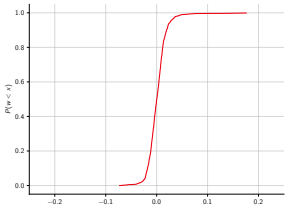
$\lambda = 0.002$



$\lambda = 0.004$



$\lambda = 0.010$



$\lambda = 0.020$

Notes

We train a LeNet on MNIST with a L_2 penalty weighted with various values λ.
As shown in the table, when λ increases, the gap between the training and the test error gets better, but when the regularization gets too important, both errors get large.
The red plots show the cumulative distributions of the weights. The larger λ, and the more the weights are concentrated around zero.

We can apply the exact same scheme with a Laplace prior

$$\begin{aligned}\mu(w) &= \frac{1}{(2b)^D} \exp\left(-\frac{\|w\|_1}{b}\right) \\ &= \frac{1}{(2b)^D} \exp\left(-\frac{1}{b}\sum_{d=1}^{D}|w_d|\right),\end{aligned}$$

which results in a penalty term of the form

$$\lambda\|w\|_1 = \lambda\sum_i |w_i|.$$

This is the L_1 regularization. As for the L_2, this penalty is convex, and its sum with a convex functional is convex.

Notes

As for the L_2 penalty, the greater λ, the more the weights will be pushed to 0.

An important property of the L_1 penalty is that, if $\mathscr{L}$ is convex, and

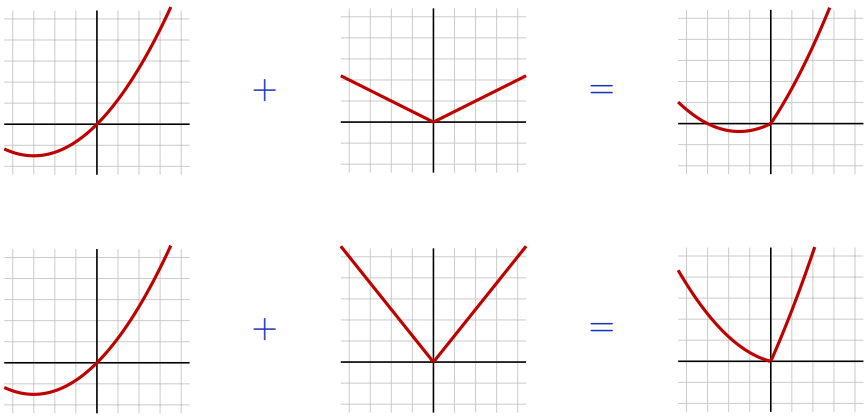
$$w^* = \operatorname*{argmin}_w \mathscr{L}(w) + \lambda \|w\|_1$$

then

$$\forall d, \ \left| \frac{\partial \mathscr{L}}{\partial w_d}(w^*) \right| < \lambda \ \Rightarrow \ w_d^* = 0.$$

In practice it means that this penalty pushes some of the variables to zero, but contrary to the L_2 penalty they actually move and remain there.

The λ parameter controls the sparsity of the solution.



Notes

These graphs help to get an intuition of why the optimal value can be moved to zero, and not "closer" to zero.
On the top row, when λ is such that the slope of the penalty (middle) is less steep than that of the original loss (left), the optimal of their sum (right) is not at zero.
On the bottom row, when λ is such that the slope of the penalty (middle) is steeper than that of the original loss (left), the optimal of their sum (right) is at zero.

With the L_1 penalty, the update rule becomes

$$w_{t+1} = w_t - \eta \left(g_t + \lambda \operatorname{sign}(w_t) \right),$$

where sign is applied per-component. This is almost identical to

$$w'_t = w_t - \eta g_t$$
$$w_{t+1} = w'_t - \eta \lambda \operatorname{sign}(w'_t).$$

This update may overshoot, and result in a component of w'_t strictly on one side of 0, while the same component in w_{t+1} is strictly on the other.

While this is not a problem in principle, since w_t will fluctuate around zero, it can be an issue if the zeroed weights are handled in a specific manner (e.g. sparse coding to reduce memory footprint or computation).

The **proximal operator** prevents parameters from "crossing zero", by adapting λ when it is too large

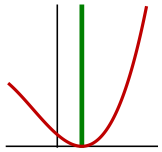
$$w'_t = w_t - \eta g_t$$
$$w_{t+1} = w'_t - \eta \min(\lambda, |w'_t|) \odot \text{sign}(w'_t).$$

where min is component-wise, and $\odot$ is the Hadamard component-wise product.

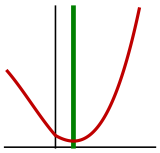
Notes

Moving by $\min(\lambda, |w'_t|)$ prevents oscillating around 0: when the [absolute value of the] gradient is less than λ, the step is adapted to get to 0 and not overshoot.

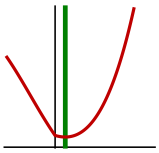
Increasing the λ parameter moves the optimal closer to 0, and away from the optimal for the loss without penalty.



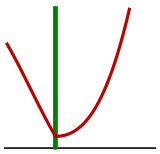
$(x-1)^2 + \frac{1}{6}(x-1)^3$



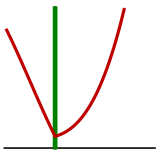
$(x-1)^2 + \frac{1}{6}(x-1)^3 + \frac{1}{2}|x|$



$(x-1)^2 + \frac{1}{6}(x-1)^3 + |x|$



$(x-1)^2 + \frac{1}{6}(x-1)^3 + \frac{3}{2}|x|$



$(x-1)^2 + \frac{1}{6}(x-1)^3 + 2|x|$

Notes

Each graph, shows in red for a certain λ the sum of the loss

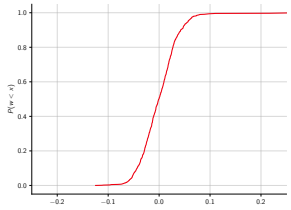
$$f : x \mapsto (x-1)^2 + \frac{1}{6}(x-1)^3$$

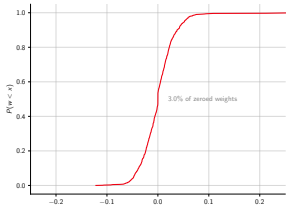
and the L_1 penalty, and with a green line its minimum.
Increasing the weight λ of the L_1 penalty moves the optimum closer to zero, and when $\lambda > |f'(0)| = 3/2$, the penalty term is big enough, and the minimum is at zero.

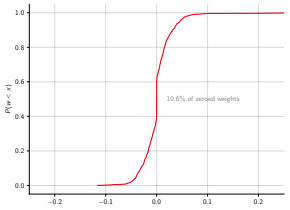
Convnet trained on MNIST with $1,000$ samples and a L_1 penalty.

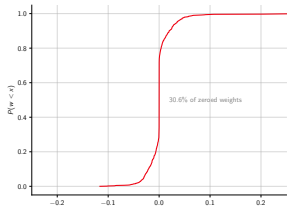
	Error	
λ	Train	Test
0.00000	0.000	0.064
0.00001	0.000	0.063
0.00002	0.000	0.067
0.00005	0.004	0.068
0.00010	0.087	0.128
0.00020	0.057	0.101
0.00050	0.496	0.516

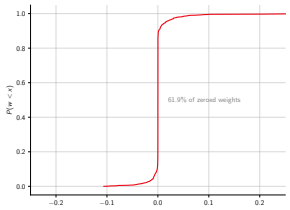
```
output = model(train_input[b:b+batch_size])
loss = criterion(output, train_target[b:b+batch_size])

optimizer.zero_grad()
loss.backward()
optimizer.step()

with torch.no_grad():
    for p in model.parameters():
        p.sub_(p.sign() * p.abs().clamp(max = lambda_l1))
```

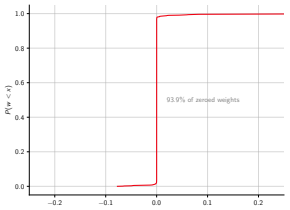

$\lambda = 0.00000$


$\lambda = 0.00002$


$\lambda = 0.00005$


$\lambda = 0.0001$


$\lambda = 0.0002$


$\lambda = 0.0005$

Notes

We train a LeNet on MNIST with a L_1 penalty weighted with various values λ.
Here we implement separately the update for the penalty with the clamping of the proximal operator `abs().clamp(max = lambda_l1)`.
As shown in the table, when λ increases, the gap between the training and the test error gets better, but when the regularization gets too important, both errors get large.
The red plots show the cumulative distributions of the weights. The larger λ, and the more the weights are concentrated around zero, with some of them exactly set to zero. With a penalty of $\lambda = 5e-4$, 91% of the weights are null.

Penalties on the weights may be useful when dealing with small models and small data-sets and are still standard when data is scarce.

While they have a limited impact for large-scale deep learning, they may still provide the little push needed to beat baselines.