Regularization

Alberto TONDA, Ph.D. (Senior permanent researcher, DR)

UMR 518 MIA-PS, INRAE, AgroParisTech, Université Paris-Saclay
UAR 3611, Institut des Systèmes Complexes de Paris Île-de-France

Outline

- Regularization
- L1/L2 regularization
- Dropout
- Batch normalization
- Regularization in pytorch

Regularization?

- An ally in the battle to fight against **overfitting**
  - Neural network researchers noticed a *correlation* with overfitting
  - Parameters or neuron outputs with large absolute values

- If parameter values represent “knowledge”
  - High values mean *hyperspecialized* neurons
  - If for some reason **they are not activated**, everything after fails
  - Ideally, we would like knowledge to be more spread out
  - “More paths”, redundancy

L1/L2 regularization

- We would like the network to fit the data, but also not parameter values much larger than others in same module
- What can we do?

L1/L2 regularization

- We would like the network to fit the data, but also not parameter values much larger than others in same module

- Modify the **loss function**
  - Add another term, penalize models with large sum of absolute values of weights (L1)
  - Or large sum of squared values of weights (L2)
  - Weighted sum with hyperparameter λ

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

Dropout

- Popular technique (for a while)
  - Active only during the training process
  - During forward pass, **randomly set** input tensor element to **zero**
  - This also impacts backward pass and gradient updates
  - Only parameter is a **probability**, applied to each tensor element
  - Can also be seen as **setting neuron output to zero**
  - Can be applied (or not) independently to each module

Batch normalization

- Growing in popularity in recent years
- Like Dropout, can be applied (or not) to each module
- Normalize output tensor of a module z, for each dimension i
  - $z_i' = \frac{z_i - \bar{z}_i}{\sqrt{Var(z_i) + \epsilon}} \cdot \gamma_i + \beta_i$
  - γ, β are learnable parameters of the module, optimized
- Avoids extremely large output values of the module
- In test/validation, γ and β are set to means of values seen during training

pytorch utils

- Pytorch implements regularization as additional **modules**
  - torch.nn.Dropout()
  - torch.nn.BatchNorm1d(), 2d(), 3d() *# horrible naming convention*
- **model.train()** and **.eval()** to activate/deactivate reg. modules
- L2 regularization is an **optimizer option**, weight_decay

SGD

CLASS torch.optim.SGD(*params*, *lr=0.001*, *momentum=0*, *dampening=0*, *weight_decay=0*, *nesterov=False*, ***, *maximize=False*, *foreach=None*, *differentiable=False*) [SOURCE]

Implements stochastic gradient descent (optionally with momentum).

- **weight_decay** (*float*, *optional*) – weight decay (L2 penalty) (default: 0)

Questions?

Bibliography

Images and videos: unless otherwise stated, I stole them from the Internet. I hope they are not copyrighted, or that their use falls under the Fair Use clause, and if not, I am sorry. Please don’t sue me.