Regularization and Special Layers in Neural Networks

Johanni Brea

Introduction to Machine Learning

EPFL

Table of Contents

1. Regularization for Neural Nets

2. Special Layers

3. Case Study: Rental Bikes

L1/L2 regularization

```python
# Define the training function with regularization
def train_model(model, criterion, optimizer, train_loader, regularization_type=None,
lambda_reg=0.01, epochs=20):
    epoch_losses = []

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0

        for inputs, targets in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs.squeeze(), targets)

            # Apply L1 regularization
            if regularization_type == 'L1':
                l1_norm = sum(p.abs().sum() for p in model.parameters())
                loss += lambda_reg * l1_norm

            # Apply L2 regularization
            elif regularization_type == 'L2':
                l2_norm = sum(p.pow(2).sum() for p in model.parameters())
                loss += lambda_reg * l2_norm

            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)

        epoch_loss = running_loss / len(train_loader.dataset)
        epoch_losses.append(epoch_loss)
        print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}")

    return epoch_losses
```

https://www.geeksforgeeks.org/machine-learning/l1l2-regularization-in-pytorch/

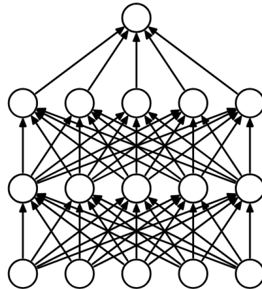
- total loss

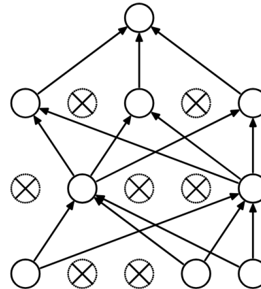
$$L_{\text{reg}-p}(\theta) = L(\theta) + \lambda \|\theta\|_p^p$$

where $\theta = (w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots)$

- Implementation for L2: "weight decay"
  $-\nabla\|\theta\|_2^2 = -2\theta$
  $-\nabla L_{\text{reg}-2}(\theta) = -\nabla L(\theta) - 2\lambda\theta.$
- In principle one could choose different regularization strengths λ for different layers.
- Tuning λ can be costly.

Dropout


(a) Standard Neural Net


(b) After applying dropout.

- Dropout is a regularization technique for neural networks.
- Randomly ”drops” (sets to zero) a subset of neurons during training.
- Prevents co-adaptation and reduces overfitting.
- At test time, all neurons are used with scaled weights.

https://jmlr.org/papers/v15/srivastava14a.html

Dropout: Implementation and Mathematical Details

- **Implementation**
  - During training, for each mini-batch, each neuron is randomly "dropped" (set to zero) with probability p; only the remaining ("active") neurons contribute to the computation of the output (sub-network).
  - At test time, all neurons are used, but their outputs are scaled by $(1 - p)$ to account for the missing units during training (averaging sub-networks).
- **Why does it prevent overfitting?**
  - Sub-networks are less flexible than the full network.
  - Model averaging is a well-established method for reducing variance.
  - Dropout can be interpreted as a regularizer that learns robust features because of noise during training.

Table of Contents

1. Regularization for Neural Nets

2. Special Layers

3. Case Study: Rental Bikes

Any Function Can Be a Layer!

$$y = b^{(4)} + w^{(4)}\mathrm{relu}\left(b^{(3)} + w^{(3)}\mathrm{relu}\left(b^{(2)} + w^{(2)}\mathrm{relu}(b^{(1)} + w^{(1)}x)\right)\right)$$

$$y = \mathrm{lin}_{b^4,w^4}(\mathrm{relu}(\mathrm{lin}_{b^3,w^3}(\mathrm{relu}(\mathrm{lin}_{b^2,w^2}(\mathrm{relu}(\mathrm{lin}_{b^1,w^1}(x)))))))$$

$$y = (\mathrm{lin}_{b^4,w^4} \circ \mathrm{relu} \circ \mathrm{lin}_{b^3,w^3} \circ \mathrm{relu} \circ \mathrm{lin}_{b^2,w^2} \circ \mathrm{relu} \circ \mathrm{lin}_{b^1,w^1})(x)$$

Instead of just $\mathrm{lin}_{w,b}$ we could use other functions to compose the neural network, e.g. dropout (non-parametric) or batch-normalization $\mathrm{bn}_{\beta,\gamma}$ (parametric).

Batch Normalization

- Batch normalization normalizes activations.
- Allows higher learning rates and reduces dependence on initialization.
- Can be inserted before or after the activation function in neural network layers.
- If used immediately after a linear layer, omit the bias in the linear layer (redundant with β).

For a mini-batch $\{a_i^{(l)}(x_1), \ldots, a_i^{(l)}(x_m)\}$, a batch normalization layer $\mathbf{bn}_{\beta,\gamma}$ computes:

$$\mu_i^{(l)} = \frac{1}{m}\sum_{i=k}^{m} a_i^{(l)}(x_k), \quad \sigma_i^{(l)} = \frac{1}{m}\sum_{i=k}^{m}(a_i^{(l)}(x_k) - \mu_i^{(l)})^2$$

where ϵ is a small constant for numerical stability. After normalization, learnable parameters $\gamma_i^{(l)}$ (scale) and $\beta_i^{(l)}$ (shift) are applied:

$$\hat{a}_i^{(l)}(x_k) = \gamma_i^{(l)} \frac{a_i^{(l)}(x_k) - \mu_i^{(l)}}{\sqrt{\sigma_i^{(l)} + \epsilon}} + \beta_i^{(l)}$$

Table of Contents

1. Regularization for Neural Nets

2. Special Layers

3. Case Study: Rental Bikes

A recipe for supervised learning

1. Collect (a lot of) data.
2. Look at the raw data; clean it if necessary.
3. Select relevant features from the raw data,
   i.e. choose a suitable representation of the raw data.
4. Select a machine learning method.
5. Fit the data and tune hyperparameters, e.g. with cross-validation.
6. - training loss: high, test loss: high (underfitting?): select a more flexible method.
   - training loss: low, test loss: high (overfitting?): select a less flexible method.
7. Repeat 4-6 until the lowest test loss is found.
8. If unhappy with the lowest test loss, repeat 2-7 or collect more data.
9. Fit the best model on all available data for best performance on unseen data.