Day 2 - Unsupervised learning

May 29, 2024

Machine learning in practice

29.05.2024

Course organization

Day 2: Unsupervised learning (Representations)

- Talk (45 minutes)
  1. Autoencoders (AEs)
  2. Bias and variance
- Hands-on tutorials
  1. Learning compact representations of single-cell data using AEs
  2. Using autoencoder representations for other tasks

Autoencoders

Autoencoders

- Autoencoders are artifical neural networks
- They learn representations of unlabelled data by means of reconstruction
- They are one of the most widely used artificial neural networks
- They are flexible and benefit from vast possible computational layers and data manipulation strategies available in deep learning
- They learn compact representations and can be used for feature engineering, data denoising, data generation etc.

Autoencoders

What are they?

The best way to see what an autoencoder is, and how it learns is to going through its training process. Consider,

- We want to build an autoencoder on our data consisting of 8000 samples and 3 features. We want to train it for 1 epoch.
- **Epoch**: Epoch is the number of times a neural network sees entire data (all 8000 samples)
- **Step**: For each epoch, we train the neural network in batches of data. Training on one batch of data refers to as a step.

$$\text{number of steps per epoch} = \frac{\text{dataset size}}{\text{batch size}},$$

e.g. For 8000 samples, there are are 10 steps per epoch with a batch size of 800.

A note on optimization

- In practice, backpropogation is done with the help of optimizers, which provide better ways to travel over loss functions and learning rates.

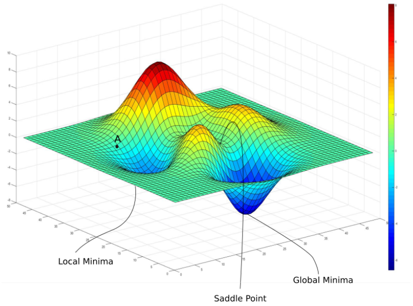

Figure: The surface of loss function. The network may get stuck in so-called local minima if we have constant learning rate and a simple gradient descent to optimize.

General network layers - I

- **Input Layer**
  The layer where the input data is fed into the network.
- **Dense (Fully Connected) Layer (The one we saw in the illustration)**
  $y = \sigma(Wx + b)$
  A layer where each neuron receives input from all neurons in the previous layer.
- **Convolutional Layer**
  $y = \sigma(W * x + b)$
  A layer that applies convolutional filters to local patches of the input.
- **Pooling Layer**
  $y = \text{pool}(x)$
  A layer that reduces the spatial dimensions of the input (e.g., Max Pooling, Average Pooling).

General network layers - II

- **Recurrent Layer**
  $h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b)$
  A layer designed for sequence data, where the output depends on previous computations (e.g., LSTM, GRU).
- **Batch Normalization Layer**
  $y = \frac{x-\mu}{\sqrt{^{2}+\epsilon}}\gamma + \beta$
  A layer that normalizes the activations of the previous layer.
- **Dropout Layer**
  $y = \text{dropout}(x, p)$
  A layer that randomly sets a fraction p of input units to zero at each update during training time to prevent overfitting.

General network layers - III

- **Activation Layer**
  $y = \sigma(x)$
  A layer that applies an activation function element-wise (e.g., ReLU, Sigmoid, Tanh).
- **Flatten Layer**
  A layer that flattens the input, usually used to transition from convolutional layers to dense layers.
- **Output Layer**
  The final layer of the network that produces the output, typically followed by an activation function (e.g., Softmax for classification).

Frameworks

- Building a network from scratch is hard
- A lot of early networks were in low level programming languages and automatic differentiation was hard
- Now, there are two most prominent frameworks: Tensorflow (From Google) and PyTorch (From Facebook).
- These frameworks offer:
    - High-level interfaces that simplify model creation and training.
    - GPU support for faster computation, making it feasible to train large models.
    - Extensive libraries of pre-built layers, models, and tools to facilitate deep learning development.
    - Strong community support and extensive documentation.
    - Tensorflow in combination with Keras (another framework by Google) is even easier to build
    - Weight sharing (reusing weights) is easy (We'll see in the lab session)

A note on Bias and Variance

- **Bias** refers to errors introduced by approximating a complex problem by a simplified model:
  - High bias leads to underfitting (*i.e.*, it can cause an algorithm to miss the relevant relations between features and target outputs).
- **Variance** refers to errors where model's output change with slight deviations in data distributions:
  - High variance leads to overfitting (*i.e.*, it can cause an algorithm to model the random noise in the training data).
- The **Bias-Variance Tradeoff**:
  - Low bias typically increases variance.
  - Low variance typically increases bias.
  - The goal is to find the right balance to minimize total error.

A note on Bias and Variance: Illustration

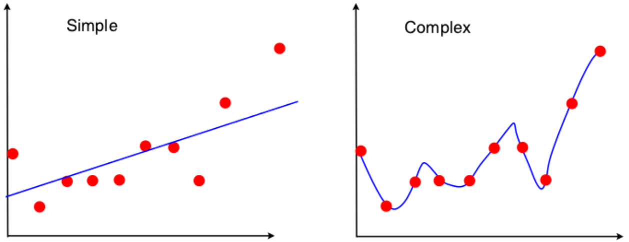

Figure: Left: high bias, right: high variance. Figure from Kaggle.

Approaches to mitigate the bias-variance tradeoff in autoencoders and deep neural networks in general

- **Regularization** (e.g., L2 regularization, dropout) to reduce overfitting.
- **Cross-validation** to ensure the model generalizes well to unseen data. (Hyperopt is a nice library to optimize neural networks with cross-validation among others `https://github.com/hyperopt/hyperopt`)
- **Data augmentation** to artificially increase the size of the training set.
- **Early stopping** to halt training when performance on a validation set starts to degrade. We'll use early stopping in our lab session.
- **Ensemble methods** (e.g., bagging, boosting) to combine the predictions of multiple models to reduce variance.

Autoencoders

Autoencoders

- **Feature engineering**: bottleneck representations
- **Denoising**: the reconstructed output has less noise as it is reconstructed from a compact representation with limited noise
- **Synthetic data generation**: With some modifications in the bottleneck. An example is variational autoencoders (A nice lecture from MIT https://www.youtube.com/watch?v=rZufA635dq4).