Basic Deep Learning Regressor

COMP421 2020

Victoria University of Wellington

Overview

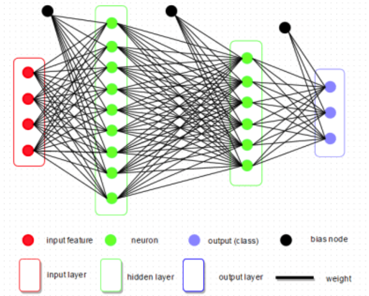
- Goal: to learn how to program basic feedforward network for regression task.
- Material mostly from chapter 6 in Goodfellow book; some from chapter 5.
- Use both deterministic and probabilistic formulation of training objective.
- Assignment is to implement regressor in TensorFlow 2 or PyTorch.

Feedforward network is a deterministic function

- Feedforward network attempts to model a function f^*.
  - We know f^* exists but we don't what it is, we only have data.
  - For example, x is image and y is bird species with $y = f^*(x)$.
  - Formal notation examples: $f^* : \mathbb{R}^{d_x} \to \{0, 1\}$ or $f^* : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$.
- Define network $f(x; \theta)$ (or write $f(x|\theta)$) where θ are network parameters:
  - The network is a universal approximator that we aim to fit to data.
  - We generally omit θ where unambiguous: then we write $f(x)$.
- Feedforward: the network is layered, information flows forward only:
  - $f^{(i)} : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ is layer i in the network.
  - Three-layer network is $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$.
  - Forward flow refers to inference, not to training.
- Internal layers are referred to as *hidden layers*.
- Internal variables are referred to as *latent variables*.
- Common to use y for output, z for latent variables, x for input.
  - For network $y = f^{(3)}(f^{(2)}(f^{(1)}(x)))$, we have $z^{(1)} = f^{(1)}(x)$ and $z^{(2)} = f^{(2)}(z^{(1)})$.
  - Notation varies with authors.

Layers consist of neurons

- Each layer of feedforward network consists of set of neurons, nodes, or units:
  - Number of units, the *width of the layer*, is set by programmer.
    - Typically based on *ablation studies*.
- Unit j of layer i does $f_j^{(i)} : \mathbb{R}^{d_{i-1}} \to \mathbb{R}$:
  - $f_j^{(i)}(z^{(i-1)}) = g(W_{j:}z^{(i-1)} + c_i)$.
    - g a nonlinear function, W a matrix, c_i a scalar.
  - Include bias in W (one more column):
    $f_j^{(i)}(z^{(i-1)}) = g(W_{j:}\begin{bmatrix} z^{(i-1)} \\ 1 \end{bmatrix})$.
- A layer: $f^{(i)}(z^{(i-1)}) = \begin{bmatrix} f_1^{(i)}(z^{(i-1)}) \\ \vdots \\ f_p^{(i)}(z^{(i-1)}) \end{bmatrix}$
- With an *element-wise operator* g we can write
  $f^{(i)}(z^{(i-1)}) = g(W\begin{bmatrix} z^{(i-1)} \\ 1 \end{bmatrix})$.



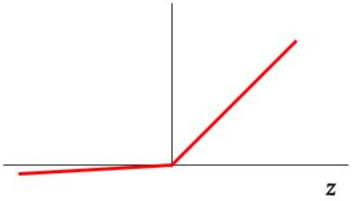
- Commonly the offset is not described but included in implementation.
- (These slides consistently use column vectors; book also uses row vectors.)

The weights W

- The weights W are network parameters. We could also have written θ, but θ tends to be used for more abstract notions.
- Given a g, the W matrix defines the particular relation between input and output.
  - There is a weight matrix for each layer. All together often also referred to as W.
- The number of weights in W typically is enormous. Hence the deep neural network can produce virtually any relationship.
- Example: a small network of three layers with five neurons each: $(5 * 5)^3 = 15625$ weight coefficients.
- Millions of weights is common.
- All elements of W must be learned from examples. Hence we need lots of data for *training*.
- Training of W can take days, even weeks.

The nonlinearity g

- The nonlinearity of g facilitates the matching of a nonlinear f^*.
- Essentially any nonlinearity can be used in principle, but not all work well for training:
  - Selection of g has been based on practical training performance, no high-flying theoretical arguments.
- Common choices in recent work:
  - ReLU (rectified linear unit): $g(Wz) = \max(0, Wz)$.
  - Leaky ReLU: $g(Wz) = \max(\alpha Wz, Wz)$, with $\alpha \ll 1$.
- Leaky ReLU perhaps most common.



Supervised training of parameters θ for regression: MMSE

- Learning to predict a $y \in \mathbb{R}^{d_1}$ given an $x \in \mathbb{R}^{d_2}$ under common conditions.
- Remember that a basic feedforward network f is a *deterministic* function.
- We want to find $f(x;\theta)$ that best fits the observations $D = \{(x^{(n)}, y^{(n)})\}_{n=1,\cdots,N}$.
    - We design the architecture, then find the optimal θ, given that architecture.
    - (Then try again in our ablation study.)
- Natural to minimize average of a distance measure, for example, mean squared error over the database: $\sum_{n\in\mathcal{A}} \|y^{(n)} - f(x^{(n)};\theta)\|_2^2$.
    - This is referred to as L2 distortion measure; it does not like outliers (big errors).
    - L1 does not mind if the mapping has a few large errors.
- We then find the parameters as $\theta^* = \arg\min_\theta \sum_{n\in\mathcal{A}} \|y^{(n)} - f(x^{(n)};\theta)\|^2$.
- Machine-learning people like to formulate things as a probabilistic model, however.

Side step: log likelihood estimate of mean of normal distribution

- Normal distribution: $Y \sim \mathcal{N}(\mu, R)$ or $p(y|\theta) = c \exp\left(-\frac{1}{2}(y-\mu)^T \Sigma^{-1}(y-\mu)\right)$.
- Uniform iid case, that is $\Sigma = \sigma^2 I$:
  - $(y-\mu)^T \Sigma^{-1}(y-\mu) = \frac{1}{\sigma^2}(y-\mu)^T I (y-\mu) = \frac{1}{\sigma^2}\|y-\mu\|_2^2$.
  - Hence $p(y|\theta) = c \exp\left(-\frac{1}{2\sigma^2}\|y-\mu\|_2^2\right)$.
- Log likelihood of μ is $LL(\mu, \mathrm{data}) = \mathrm{constant} - \frac{\sigma^2}{2}\sum_{n=1}^{N}\|y^{(n)}-\mu\|_2^2$.
- Likelihood estimate of μ is $\mu^* = \arg\max_\mu LL(\mu, \mathrm{data}) = \arg\min_\mu \sum_{n=1}^{N}\|y^{(n)}-\mu\|_2^2$.
- Under these conditions maximum likelihood objective equivalent to MMSE objective!
- The value of σ^2 does not affect the outcome.
- This principle allows machine learning people to use max likelihood objective instead.
- But we are not quite there yet: in regression (machine learning) μ depends on the input data. Output consists of some mean μ, the observation includes additive noise (often modeled as normal, in part for reasoning above).

Supervised training of parameters θ for regression: log likelihood

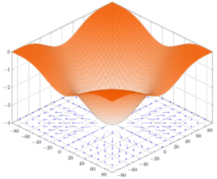
- Learning to predict a $y \in \mathbb{R}^{d_1}$ given an $x \in \mathbb{R}^{d_2}$ under common conditions.
- Remember that a basic feedforward network f is a *deterministic* function.
- In keeping with much of the literature, our model is *probabilistic*:
  $y = f^(x) + \eta$, where η is zero-mean iid Gaussian noise.
- Now formulate our regression problem:
  - Our model is $\eta = y - f^*(x) \sim \mathcal{N}(0, \sigma^2 I)$ or, equivalently, $y \sim \mathcal{N}(f(x), \sigma^2 I)$.
  - So the network outputs an estimated mean $\mu(x) = f(x)$ of the predictive distribution.
  - Find $f(x;\theta)$ that best fits the observations $D = \{(x^{(n)}, y^{(n)})\}_{n \in \mathcal{A}}$.
  - Fit $p(y|x,\theta) = c \exp\left(-\frac{1}{2\sigma^2}\|y - f(x;\theta)\|^2\right)$ using log likelihood.[1]
  - That is $\theta^* = \arg\max_\theta LL(\theta, D) = \arg\min_\theta \sum_{n \in \mathcal{A}} \|y^{(n)} - f(x^{(n)};\theta)\|^2$:
- We can now use a probabilistic formulation in our paper but do MSE in our gitHub implementation and confuse newcomers who did not attend COMP421.
- Moreover, at inference (when computing the predicted output) we generally don't add the output noise. That gives a better estimate of the desired output!

[1]Omitted norm subscript as is common for 2-norm.

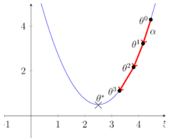
Supervised training of θ for regression: gradient descent

- The problem of predicting a $y \in \mathbb{R}$ given an $x \in \mathbb{R}$.
- Must solve: $\theta^* = \arg\max_\theta LL(\theta, D) = \arg\min_\theta \sum_{n\in\mathcal{A}} \|y^{(n)} - f(x^{(n)};\theta)\|^2$.
  - $D = \{(x^{(n)}, y^{(n)})\}_{n\in\mathcal{A}}$ is entire database.

- Gradient $\nabla_\theta LL(\theta)$ is vector pointing in steepest direction:
  - $\nabla_\theta LL(\theta) = \begin{bmatrix} \frac{\partial LL(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial LL(\theta)}{\partial \theta_M} \end{bmatrix}$

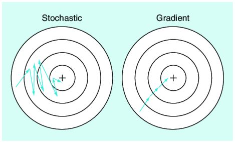


- Gradient descent walks loss function downhill in the steepest direction: $\theta^{k+1} = \theta^k + \epsilon \nabla_\theta LL(\theta, D)$:
  - Confusing: "loss function" is minus LL.
  - Until stuck in minimum of loss function.
  - Might not be the only/deepest minimum, of course.
  - The scalar ϵ is the learning rate.
    - Too fast, you overshoot; too slow, you never get there.
  - Works even if analytic solution does not exist.
  - Slow as gradient computed over entire data base.



Supervised training of θ for regression: stochastic gradient descent

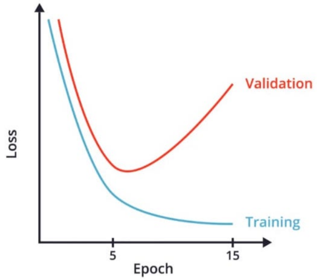
- Practical and better: stochastic gradient descent:
  $\theta^{k+1} = \theta^k + \epsilon \nabla_\theta LL(\theta, \mathbb{B}^k)$
  - $\mathbb{B}^k$ is minibatch k: the k'th random subset of database D.
  - Subset of fixed size, typically about 100 samples.
  - Update after each minibatch.
  - Gradient is noisy; hence less likely to get stuck in local minimum.
  - Faster.
- Note: originally (and still to some) *stochastic* referred to minibatches of just one sample.



Training, validation, and testing databases; when to stop training

- Training for a long time with lots of parameters leads to *overfitting*.

- Databases are split into testing and training databases.
- The training database is often split in a proper training database and a validation database.
- The validation database is used to check if overfitting is starting to occur.
- Referred to as *early stopping*.



Regression algorithm

- Train network f (find θ) to perform regression $\hat{y} = f(x; \theta)$.
- Data are of form $x \in \mathbb{R}^{dx}$ and $y \in \mathbb{R}^{dy}$.
- Note: frequent validation test inefficient.

```
Require: Training data $T = \{x^{(n)}, y^{(n)}\}_{n \in \mathcal{T}}$ and validation data $V = \{x^{(n)}, y^{(n)}\}_{n \in \mathcal{V}}$
Require: Learning rate $\epsilon$ and threshold $\alpha$
Require: Network $f : \mathbb{R}^{dx} \to \mathbb{R}^{dy}$ with parameters $\theta$
  $k = 1$
  $\nu^0 = \infty$
  $\nu^1 = \sum_{n \in V} \|y^{(n)}\|^2$
  $\theta \sim \mathcal{N}(0, 0.05I)$        # normal initialization
  while $\nu^k < \alpha \nu^{k-1}$ do
    $k \leftarrow k + 1$
    select minibatch $\mathcal{B}^k$ from $T$
    $L = \sum_{n \in \mathcal{B}^k} \|y^{(n)} - f(x^{(n)}; \theta)\|^2$
    $\theta \leftarrow \theta - \epsilon \nabla_\theta L$
    $\nu^k = \sum_{n \in V} \|y^{(n)} - f(x^{(n)}; \theta)\|^2$
  end while
  return $\theta$
```

Problem

1. Using TensorFlow 2 or PyTorch, write a regression algorithm that projects 3D zero-mean unit variance normal-distributed (Gaussian) data onto a unit-radius sphere (the surface of a ball). Natural to use Python and NumPy as basis.
2. You can choose your own platform for your work. For example, the project can be done in the cloud using Google's colab, which includes both TF 2 and PyTorch.
3. More detailed description:
   3.1 Write a function that creates normal-distributed input data x and the corresponding desired-output data y that live on the unit sphere (surface of ball of radius 1).
   3.2 Split training, validation, and test data.
   3.3 Create a 3D plot of the training data output on the sphere.
   3.4 Create a neural network model (e.g., using Keras). Use four fully-connected (dense) layers that are 3, 10, 10, and 3 wide, and ReLU.
   3.5 Train the model with reasonable minibatch size, epoch number, database size, initalization (that is, try some).
   3.6 Obtain training validation performance as a function of epoch.
   3.7 Verify with a reasonable measure that prediction (inference) using test data gives reasonable performance.
   3.8 Write report of two pages or less consisting of 1 intro, 2 theory/concept, 3 results/conclusion (with figures), using template provided on assignment web page. The report is graded based on depth of understanding, results, and clarity, not length. Provide link to code, or place it in appendix (which can then extend beyond the two pages).