Neural Networks

STAT3009 Recommender Systems

by Ben Dai (CUHK)
on August 26, 2022

» Recall LFM: NCF

Recall the basic Latent Factor Model:

$$\min_{\boldsymbol{P},\boldsymbol{Q}} \frac{1}{|\Omega|} \sum_{(u,i)\in\Omega} (r_{ui} - \boldsymbol{p}_u^\intercal \boldsymbol{q}_i)^2 + \lambda \Big(\sum_{u=1}^{n} \|\boldsymbol{p}_u\|_2^2 + \sum_{i=1}^{n} \|\boldsymbol{q}_i\|_2^2 \Big) \qquad (1)$$

* The **interaction** btw users and items are formulated as **inner production**.
* It can be extended to **high-order** nonlinear interaction.

» Nonlinear interaction: Neural networks

* For a general nonlinear function f, the predicted rating can be formulated as,

$$\widehat{r}_{ui} = f(\boldsymbol{p}_u, \boldsymbol{q}_i).$$

* Nonlinear methods: **polynomials**, **B-splines**, **kernel methods**.
* or $f(\cdot, \cdot)$ can be a **neural network**.

Before apply **neural networks** into recommender systems, we shall have a quick overview of ML models and neural networks.

» Recall ML overview

Data (feat, label) is a pair of input features and its outcome

$\rightarrow$Model f_θ: a parameterized function to map features to label

Loss $L(\cdot,\cdot)$: the measure of how good the predicted outcome compared with the true outcome

$\rightarrow$Opt The algorithm for solving the problem

$\rightarrow$: data, loss are all the same; we just design our model as a neural network, and find an opt algo to solve it

» Recall ML Overview

Let's Recall:

- →Step 1 Design your **model** (param & hp); Grid for hp
- →Step 2 Train param based on training set with different hp
- Step 3 Compute valid loss for each hp based on a valid set or *k*-fold CV; and select the optimal hp
- Step 4 Refit the model with optimal hp based on ALL data
- Step 5 Make **prediction** for test set

» Recall ML Overview

Let's Recall:

→Step 1 Design your **model** (param & hp); Grid for hp

→Step 2 Train param based on training set with different hp

Step 3 Compute valid loss for each hp based on a valid set or *k*-fold CV; and select the optimal hp

Step 4 Refit the model with optimal hp based on ALL data

Step 5 Make **prediction** for test set

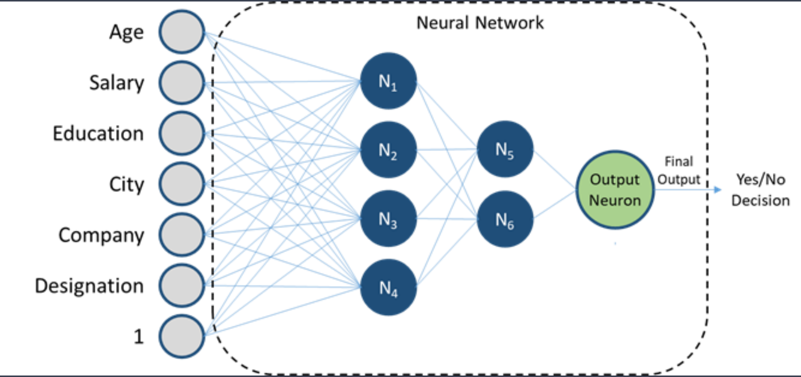
Q1 What's the parameters and hp for a neural network?

Q2 How to train a neural network?

» Neural networks

Model diagram:

input → hidden layer 1 → ⋯ hidden layer L → output

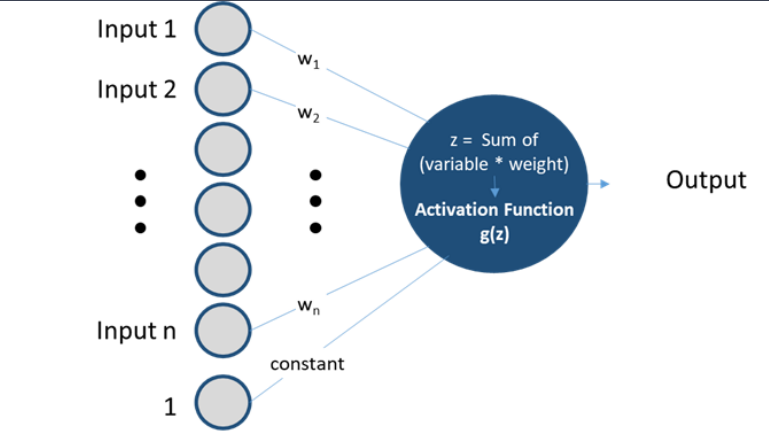


[1]

[1] https://towardsdatascience.com/deep-learning-101-neural-networks-explained-9fee25e8ccd3

» Neural networks

Layer diagram: Look at one neuron in the next layer



2

2 https://towardsdatascience.com/deep-learning-101-neural-networks-explained-9fee25e8ccd3

» Neural networks

⚠ Math formulation:

* **Nonlinear** activation + **linear combination** of outputs from the previous layer
* From input $\mathbf{f}_0 = \mathbf{x}$ to output $\mathbf{f}_L(\mathbf{x})$:

$$\boldsymbol{f}_l(\boldsymbol{x}) = A(\boldsymbol{W}_l \boldsymbol{f}_{l-1}(\boldsymbol{x}) + \boldsymbol{b}_l), \quad l = 1, \cdots, L.$$

* $\mathbf{W}_l \in \mathbb{R}^{d_l \times d_{l-1}}$ - weight matrix in the l-th layer
* $\mathbf{b}_l \in \mathbb{R}^{d_l}$ - intercept terms in the l-th layer
* L - #Layers or depth of the neural network
* $A(\cdot)$ - activation function.
  * Activation function: logistic (sigmoid), ReLU, tanh, and more options[3]; why we need a nonlinear activation?
* $\mathbf{f}_l(\mathbf{x}) \in \mathbb{R}^{d_l}$ - #Neurons in the l-th layer

[3] https://en.wikipedia.org/wiki/Activation_function

» Neural networks: Param & hp

A1. Lay out parameters and hyperparameters

Params the collection of all weights and biases,

$$\theta = \{\boldsymbol{W}_0, \boldsymbol{b}_0, \cdots, \boldsymbol{W}_{L-1}, \boldsymbol{b}_{L-1}\}$$

- **weight**: $\boldsymbol{W}_l \in \mathbb{R}^{d_l \times d_{l-1}}$, **bias**: $\boldsymbol{b}_l \in \mathbb{R}^{d_l}$

» Neural networks: Param & hp

A1. Lay out parameters and hyperparameters

Params the collection of all weights and biases,

$$\theta = \{\boldsymbol{W}_0, \boldsymbol{b}_0, \cdots, \boldsymbol{W}_{L-1}, \boldsymbol{b}_{L-1}\}$$

* **weight**: $\boldsymbol{W}_l \in \mathbb{R}^{d_l \times d_{l-1}}$, **bias**: $\boldsymbol{b}_l \in \mathbb{R}^{d_l}$

hp the architecture of a neural network

* L - #Layers or depth of the neural network
* d_l - #Neurons in the l-th layer; $l = 1, \cdots, L$

Tradeoff $L, d_l \nearrow$

$\Longrightarrow$ model becomes more complicate

$\Longrightarrow$ model complexity $\nearrow$

$\Longrightarrow$ training error $\searrow$

A2. Train a neural network by SGD + backpropagation

SGD Recall. Compute stochastic gradients for all params

* Gradient:

$$\frac{\partial \text{Loss}}{\partial \theta} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L(y_i, \mathbf{f}_L(\mathbf{x}_i))}{\partial \theta}$$

* Approx by ONE sample:

$$\frac{\partial \text{Loss}}{\partial \theta} \leftarrow \frac{\partial L(y_i, \mathbf{f}_L(\mathbf{x}_i))}{\partial \theta}$$

* Approx by a BATCH of samples

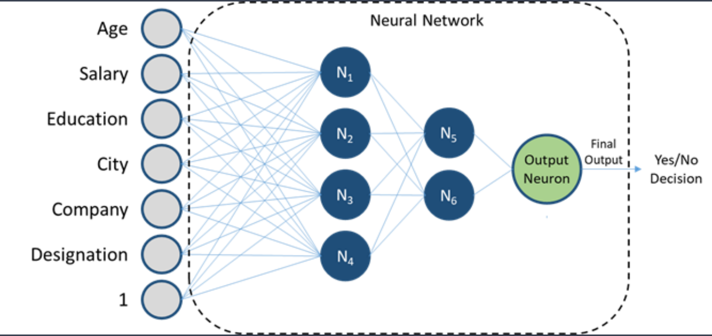
$$\frac{\partial \text{Loss}}{\partial \theta} \leftarrow \sum_{i \in Batch} \frac{\partial L(y_i, \mathbf{f}_L(\mathbf{x}_i))}{\partial \theta}$$

» Neural networks: training

A2. Train a neural network by SGD + backpropagation

SGD How to compute SG of $L(y_i, \mathbf{f}_L(\mathbf{x}_i))$ wrt all params

* From easiest (last) to hardest (first)
* So-called backpropagation
* Ref: How the backpropagation algorithm works



» Backpropagation: chain rule

Check the gradients for the params from different layers

Last layer $$\frac{\partial L(y_i,\mathbf{f}_L(\mathbf{x}_i))}{\partial \mathbf{W}_L} = \frac{\partial L(y_i,\mathbf{f}_L(\mathbf{x}_i))}{\partial \mathbf{f}_L(\mathbf{x}_i)} \frac{\partial \mathbf{f}_L(\mathbf{x}_i)}{\partial \mathbf{W}_L}$$

L-1 layer $$\frac{\partial L(y_i,\mathbf{f}_L(\mathbf{x}_i))}{\partial \mathbf{W}_{L-1}} = \frac{\partial L(y_i,\mathbf{f}_L(\mathbf{x}_i))}{\partial \mathbf{f}_L(\mathbf{x}_i)} \frac{\partial \mathbf{f}_L(\mathbf{x}_i)}{\partial \mathbf{f}_{L-1}(\mathbf{x}_i)} \frac{\partial \mathbf{f}_{L-1}(\mathbf{x}_i)}{\partial \mathbf{W}_{L-1}}$$

L-2 layer $$\frac{\partial L(y_i,\mathbf{f}_L(\mathbf{x}_i))}{\partial \mathbf{W}_{L-2}} = \frac{\partial L(y_i,\mathbf{f}_L(\mathbf{x}_i))}{\partial \mathbf{f}_L(\mathbf{x}_i)} \frac{\partial \mathbf{f}_L(\mathbf{x}_i)}{\partial \mathbf{f}_{L-1}(\mathbf{x}_i)} \frac{\partial \mathbf{f}_{L-1}(\mathbf{x}_i)}{\partial \mathbf{f}_{L-2}(\mathbf{x}_i)} \frac{\partial \mathbf{f}_{L-2}(\mathbf{x}_i)}{\partial \mathbf{W}_{L-2}}$$

$$\ldots$$

Chain rule!

» Neural networks: training

SGD/B-SGD has additional fitting parameters (fp)

$$\theta^{\text{new}} \leftarrow \theta^{\text{old}} - \text{learning rate} \times \sum_{i \in Batch} \frac{\partial L(y_i, \mathbf{f}_L(\mathbf{x}_i))}{\partial \theta}\Big|_{\theta^{\text{old}}}$$

* learning rate - step size per gradient update
* batch_size - Number of samples per gradient update
* epochs - Number of epochs to train the model. An epoch is an iteration over the entire training data provided

* **Goodness: flexible computing platforms**, such as TensorFlow, Keras and Pytorch, are available for implementing a custom neural network.
* What we will do in practice?
    * **Model.** Define your **own model** $\boldsymbol{f}(\boldsymbol{x})$
    * **Loss.** and **metric.** Specify **loss function** and **metrics** for the problem.
    * **Algo.** tf.keras.optimizer.SGD will **automatically compute the gradient** via backpropagation[4]
    * Feed **data** to train the defined model.

[4] http://neuralnetworksanddeeplearning.com/chap2.html

» Example: Data, Loss, Algo, and Metric

- **InClass demo: Implementation based on tf.keras in colab**
- **Binary Classification** - Credit card fraud detection: *credit card transactions labeled as fraudulent or genuine*

$$\operatorname*{argmin}_{\theta} \frac{1}{n}\sum_{i=1}^{n} L\big(y_i, \boldsymbol{f}(\boldsymbol{x}_i)\big)$$

- **Data.** $\boldsymbol{x}_i \in \mathbb{R}^d \to y_i \in \{0, 1\}$;
- **Model.** $\boldsymbol{f}(\boldsymbol{x}) \in [0, 1] \to \mathbb{P}(Y = 1 | \boldsymbol{x})$;
- **Loss.** Logistic loss (or Binary cross entropy);

$$L(y_i, f(\boldsymbol{x}_i)) = -\Big(y_i \log\big(f(\boldsymbol{x}_i)\big) + (1 - y_i)\log\big(1 - f(\boldsymbol{x}_i)\big)\Big).$$

- **Algo.** SGD;
- **Metric.** Acc, and other metrics for classification

» Neural networks: Cross-validation

Cross-validation... Given training, validation, testing sets

Step 1 Design your **neural network**; candidate hp

- param: weight matrix, intercept vector
- hp: depth, #neurons, types of layers

Step 2 Train param based on training set with different hp

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L\big(y_i, \mathbf{f}_L(\mathbf{x}_i)\big)$$

Step 3 Compute valid loss for each hp based on a valid set or k-fold CV; and select the optimal architecture

Step 4 Refit the model with optimal hp based on ALL data

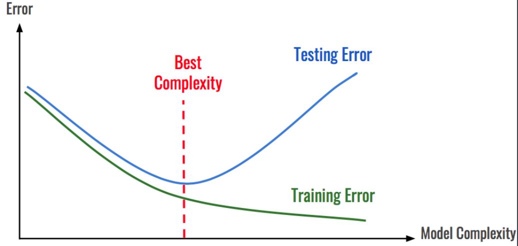
Step 5 Make **prediction** for test set

» Neural networks: Cross-validation

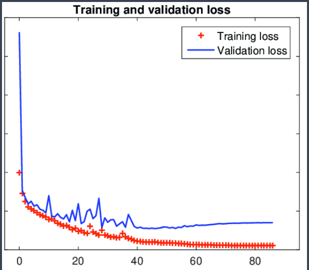
* CV in the [Previous Page] is entirely correct
* BUT hardly ever used in practice
* Training a neural network is not easy...
  * Too many **hps**
  * a CNN on 16 vCPU: **200 epochs took us 5 days to run**.
  * Source
* Solution: Monitor on valid set + Early-stopping: Stop training when a monitored valid metric has stopped improving.

» Neural networks: bias-variance trade-off

ML: x-axis: Model complexity VS y-axis: Error



DL: x-axis: #iteration VS y-axis: Error



» Neural networks: EarlyStopping

If we can stop before the overfitting ...

Monitor + Early-stopping:

```
Epoch 1/30
112/112 - 6s - loss: 1.1414e-09 - acc: 1.0000
- val_loss: 2.6730e-10 - val_acc: 1.0000
...
Epoch 4/30
112/112 - 4s - loss: 1.0782e-10 - acc: 1.0000
- val_loss: 4.6980e-11 - val_acc: 1.0000
Epoch 5/30
112/112 - 4s - loss: 7.6734e-11 - acc: 1.0000
- val_loss: 3.4825e-11 - val_acc: 1.0000
Epoch 6/30
Restoring model weights from the end of the best epoch: 1.
112/112 - 5s - loss: 5.8153e-11 - acc: 1.0000
- val_loss: 2.7316e-11 - val_acc: 1.0000
Epoch 6: early stopping
```

» Rules of Thumb: Neural Networks

Designing a neural network is a bit TOO flexible

* Problem $\rightarrow$ activation for output layer + Loss + metric
* Number of nodes of hidden layers - The first hidden layer should be around half of the number of input features. The next layer size as half of the previous. For example: 128, 64, 32, ...
* Activation - Usually ReLU is good
* Epochs - Start with 20 to see if the model fitting shows decreasing loss and any improvement in accuracy. If there is no minimal success with 20 epochs, increase epochs. If you get some minimal success, make epoch as 100. Combine some CV techniques
* Batch size - Select the batch size from the geometric progression of 2 starting with 16. For unbalanced datasets have larger value, like 128.

» Appendix: Universal approximation theorem

Theorem (Universal approximation theorem)

For any function[a] $f : \mathbb{R}^d \to \mathbb{R}^K$ *and any* $\varepsilon > 0$, *there exists a fully-connected ReLU network* g *of width exactly* $\max(d+1, K)$, *such that*

$$\|f - g\|_p^p \leq \varepsilon.$$

[a] Bochner-Lebesgue p-integrable function

* Universal approximation theorem implies that a deep neural network can approximate an arbitrary function.

» Universal approximation theorem: example

Example: a neural network $\rightarrow$ sin function

* Ground Truth: $f^*(x) = sin(x)$
* Network: a two-layer neural network using 100 neurons per layer
* The results of fitting: Source