Machine Learning

Model Evaluation

Satishkumar L. Varma

Department of Information Technology
SVKM's Dwarkadas J. Sanghvi College of Engineering, Vile Parle, Mumbai.
ORCID | Scopus | Google Scholar | Google Site | Website

Model Evaluation: Outline

- Need of Model Evaluation
- Loss Function
- Regularization: Controlling the Learning Process
- Apply Regularization to Learning Models
- Model Evaluation
    - Underfitting,
    - Overfitting,
    - Lasso regularization,
    - Ridge regularization,
    - Elastic Net regularization.

Model Evaluation: Why Model Evaluation is Important

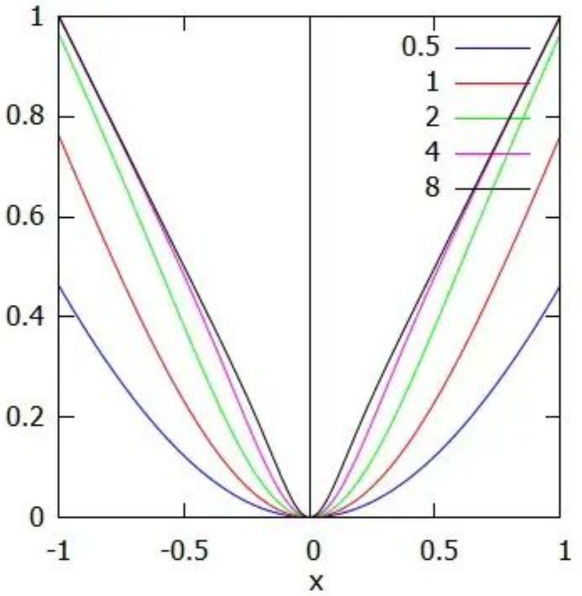
- Need of Model evaluation
    - It is important to assess the efficacy of a model during initial research phases, and
    - It also plays a role in model monitoring.
    - Before deploying your ML model, it is essential to evaluate its performance on real-world data.
    - If your model is trained on a clean dataset,
        - it is important to generate a dataset that simulates real-world data as closely as possible.
- Objectives
    - To understand the concept of overfitting and how regularization helps in reducing overfitting.
    - To understand if your model(s) is working well with new data, you can leverage a # of evaluation metrics.
    - To minimize an error function (loss function) or
    - To maximize the efficiency of production using Optimizers algorithms.

Model Evaluation: Loss function

- **LF** along with **optimization functions** are directly responsible for f**itting the model** to the given training data.
- Learning Objectives:
  - how loss functions are used in neural networks,
  - know different types of loss functions,
  - writing custom loss functions in TensorFlow, and
  - practical implementations of loss functions to process image/video training data
- NN processes the input data at each layer and eventually produces a predicted output value $\hat{y} = AF(X' W + b)$
- To training process makes the model maps the relationship between the training data and the outputs.
- To satisfy the equation $\hat{y} = AF(X' W + b)$, in this training process, the NN updates its **hyperparameters:**
  - W – weights
  - b – biases

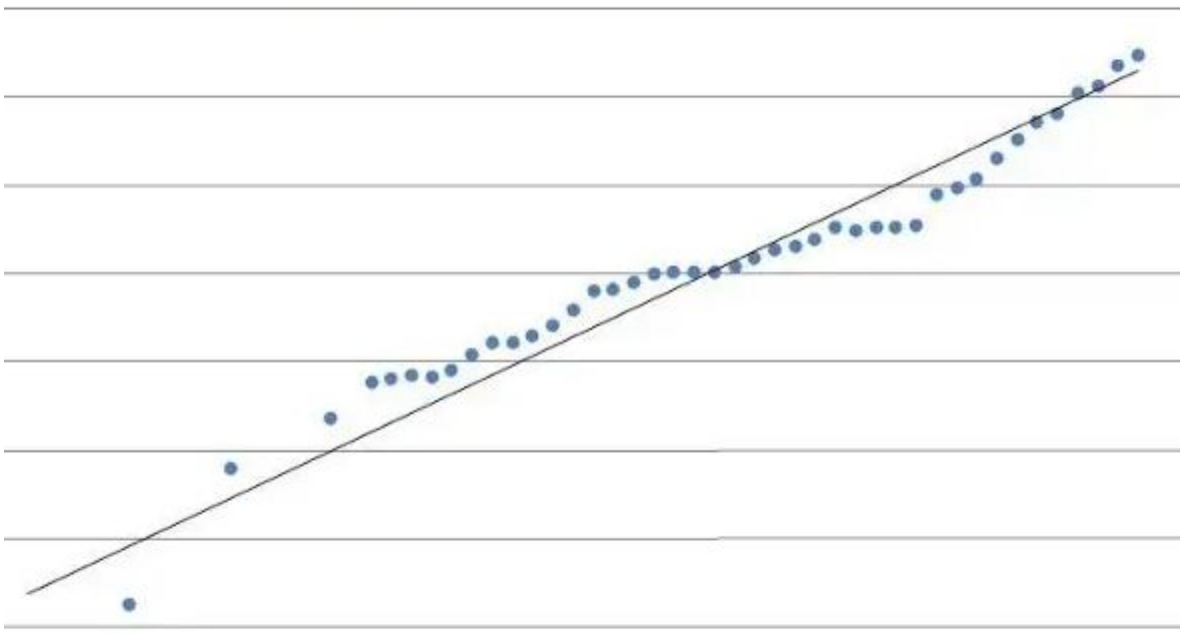
Model Evaluation: Loss function

- Each training input is loaded into the NN in a process called **forward propagation**.
- Once model produces o/p (predicted o/p) is compared against given target o/p in a process called **backpropagation**
- In **backpropagation** process, **hyperparameters** of model are then
  - adjusted to **minimize** (error=predicted-target) output
- This is where loss functions come in.
- A **loss function**
  - compares the target (y) and predicted (ŷ) output values;
  - measures how well the NN models the training data
- When training,
  - we aim to **minimize** this loss bet.ŷ and y
- The hyperparameters are adjusted to minimize the average loss
  - we find w and b, that minimize the value of J (average loss)



Model Evaluation: Loss function

- Loss function is akin to residuals, in statistics.
- Residuals measure the distance of the actual y values from the **regression line** (predicted values)
  - the goal being to minimize the net distance



Model Evaluation: Loss function

- Easy to demonstrate how loss functions are used in models.
  - Using Google's TensorFlow library to implement different loss functions
- In TensorFlow, the LF is specified as a parameter in **model.compile()**
  - This method trains the neural network eg. **model.compile(loss='mse', optimizer='sgd')**
- LF can be inputted either as
  - a String — as shown above — or
  - as a function object — either imported from TensorFlow or written as custom loss functions
  - eg. **from tensorflow.keras.losses import mean_squared_error**
  - **model.compile(loss=mean_squared_error, optimizer='sgd')**
- All loss functions in TensorFlow have a similar structure:
  - def loss_function (y_true, y_pred): return losses
- The **model.compile()** method expects only two input parameters for the loss attribute i.e y_true and y_pred

Model Evaluation: Types of Loss function

- **Regression Loss Functions** — used in regression neural networks;
  - given an input value, the model predicts a corresponding output value (rather than pre-selected labels);
  - Ex. Mean Squared Error, Mean Absolute Error
- **Classification Loss Functions** — used in classification neural networks;
  - given an input, the NN produces a vector of probabilities of the input belonging to various pre-set categories
  - can then select the category with the highest probability of belonging;
  - Ex. Binary Cross-Entropy, Categorical Cross-Entropy
- Loss function in Machine Learning / Deep Learning
- 1. Regression: MSE(Mean Squared Error), MAE(Mean Absolute Error), Huber loss
- 2. Classification: Binary cross-entropy, Categorical cross-entropy
- 3. AutoEncoder: KL Divergence
- 4. GAN: Discriminator loss, Minmax GAN loss

Model Evaluation: Loss function

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

```
def mse (y_true, y_pred):
    return tf.square (y_true - y_pred)
```

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y^{(i)} - \hat{y}^{(i)}|$$

```
def mae (y_true, y_pred):
    return tf.abs(y_true - y_pred)
```

$$CE\ Loss = -\frac{1}{n}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot log(p_{ij})$$

Binary cross-entropy is a special case of categorical cross-entropy (above), where M = 2 (the number of categories)

$$Huber\ Loss = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 & |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \frac{1}{n}\sum_{i=1}^{n}\delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta) & |y^{(i)} - \hat{y}^{(i)}| > \delta \end{cases}$$

```
def huber_loss_with_threshold (t = δ):
    def huber_loss (y_true, y_pred):
        error = y_true - y_pred
        within_threshold = tf.abs(error) <= t
        small_error = tf.square(error)
        large_error = t * (tf.abs(error) - (0.5*t))
        if within_threshold:
            return small_error
        else:
            return large_error
    return huber_loss
```

$$CE\ Loss = \frac{1}{n}\sum_{i=1}^{N} -(y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i))$$

```
def log_loss (y_true, y_pred):
    y_pred = tf.clip_by_value(y_pred, le-7, 1 - le-7)
    error = y_true * tf.log(y_pred + 1e-7) (1-y_true) * tf.log(1-
y_pred + 1e-7)
    return -error
```

Model Evaluation: What is Regularization?

- Model evaluation
  - Process of using different evaluation metrics to understand performance (strengths and weaknesses) of model
  - Assess the efficacy of a model during initial research phases, and it also plays a role in model monitoring.
- Regularization
  - Process of limiting (controlling) the learning process during training of a model
  - by adding another term to the loss (cost) function (trying to minimize).
    - Objective function $(\theta) = L(\theta) + \Omega(\theta)$
    - $L(\theta)$ is loss function
    - $\Omega(\theta)$ is regularization term
  - The main benefit of regularization is to mitigate overfitting.
  - Regularized models are able to generalize well on the unseen data.
  - A complex NN makes training model more prone to overfitting.
  - Regularization makes slight modifications to the learning algorithm such that the model generalizes better.

Model Evaluation: What is Regularization?

- Two ways to apply regularization to learning models:
- By adding another term to the loss function
  - We are trying to minimize
  - The objective function consists of two parts: loss function and regularization term:
    - Objective function $(\theta) = L(\theta) + \Omega(\theta)$
    - $L(\theta)$ is loss function
    - $\Omega(\theta)$ is regularization term
- By early stopping the learning process
  - during the training phase
  - The perfect example of this method is stopping the growth of a decision tree at an early stage.

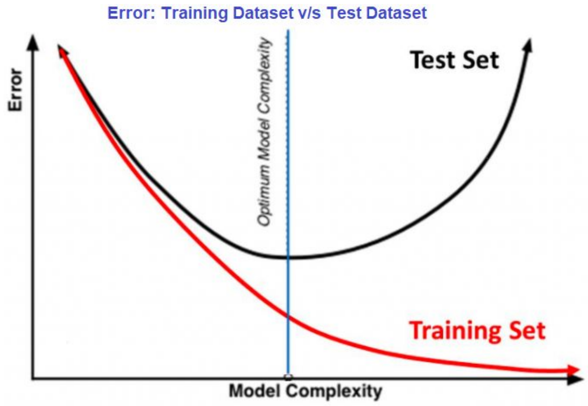
Model Evaluation: What is Regularization?

- Performing L2 regularization encourages the weight values towards zero (but not exactly zero)
- Performing L1 regularization encourages the weight values to be zero
- Intuitively speaking smaller weights reduce the impact of the hidden neurons.
- Less complex models typically avoid modeling noise in the data, and therefore, there is no overfitting.
- When choosing the regularization term α.
- The goal is to strike the right balance between low complexity of the model and accuracy
- If your alpha value is too high, your model will be simple, but you run the risk of underfitting your data.
- Your model won't learn enough about the training data to make useful predictions.
- If α value is too low, your model will be more complex, and you run the risk of overfitting your data.
- Your model will learn **too much** about the particularities of the training data, and
    - won't be able to generalize to new data.

Model Evaluation: What is Regularization?

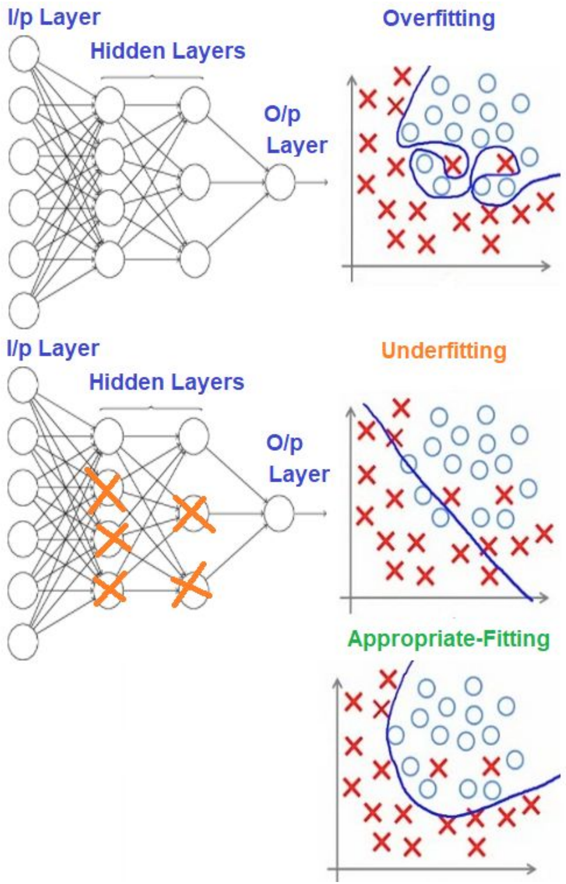
- As we move towards the right in this image our model tries to learn too well the details and
- the noise from the training data, which ultimately results in poor performance on the unseen data.

- While going towards the right
  - Complexity of the model increases such that the training error reduces but the testing error doesn't.



Model Evaluation: Regularization

- How does Regularization help reduce Overfitting?
- NN which is overfitting on the training data (TD) as shown (top)
- In ML, Regularization penalizes the coefficients
- In DL, it actually penalizes the weight matrices of the nodes.
- Assume that our regularization coefficient is so high
    - that some of the weight matrices are nearly equal to zero.
- It result in a much simpler linear NN & slight underfitting of TD (middle)
- Such a large value of the regularization coefficient is not that useful
- We need to **optimize** the value of regularization coefficient in order to obtain a well-fitted model as shown (bottom)



Model Evaluation: Regularization Techniques

- Different techniques in order to apply regularization in learning for reducing overfitting
  - Ridge (apply L2 regularization)
  - Lasso (apply L1 regularization)
  - Elastic Net (Apply both L1 and L2 regularization)
  - Dropout regularization
  - Data augmentation
  - Early stopping
- What does Regularization achieve?
- Performing L2 regularization encourages the weight values towards zero (but not exactly zero)
- Performing L1 regularization encourages the weight values to be zero
  - Intuitively speaking smaller weights reduce the impact of the hidden neurons.
  - These **hidden neurons become neglectable (dropouts)** and
  - Overall complexity of the neural network gets reduced

Model Evaluation: Regularization Techniques

- **L1 and L2 Regularization** update the general **cost function** by adding another term known as regularization term
  - Cost function = Loss (say, binary cross entropy) + Regularization term
- Due to the addition of this regularization term
  - Values of wt. matrices decrease because it assumes NN with smaller wt. matrices leads to simpler models
  - Therefore, it will also reduce overfitting to quite an extent.
- However, this regularization term differs in L1 and L2
- **L2 regularization:**
  - Lambda is the regularization parameter
  - It is the hyperparameter whose value is optimized for better results.
  - It is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero)
- **L1 regularization:**
  - We penalize the absolute value of the weights
  - Unlike L2, the weights may be reduced to zero here.
  - Hence, it is very useful when we are trying to compress our model.
  - Otherwise, we usually prefer L2 over it

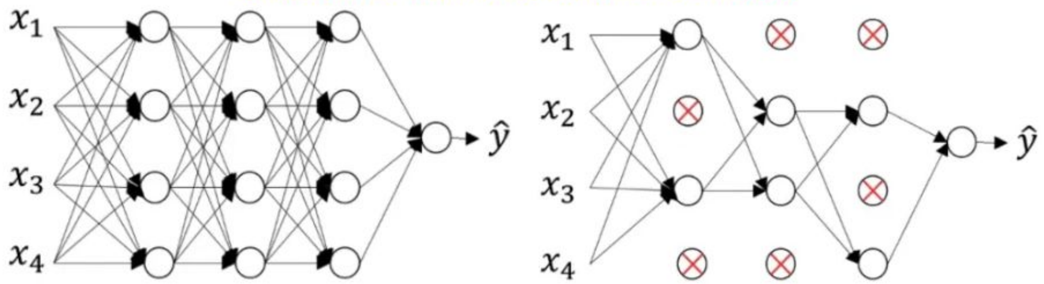
Model Evaluation: Regularization Techniques

- How to apply regularization to any layer in keras
- Sample code to apply L2 regularization to a Dense layer.
  - from keras import regularizers
  - model.add(Dense(64, input_dim=64, kernel_regularizer=regularizers.l2(0.01)
- Note:
  - Here, value of regularization parameter, i.e., lambda = 0.01, which we need to optimize further
  - We can optimize it using the grid-search method.
- Similarly, we can also apply L1 regularization

Model Evaluation: Dropout Regularization

- Dropout Regularization is simple and famous and powerful regularization technique
- Dropout means that with some probability P a neuron of the NN gets turned off **during training** (see figure)
    - Left side of Fig.: Assume we have a feedforward neural network with no dropout
    - Right side of Fig.: Using dropout with say a probability of P=0.5 that a random neuron gets turned off
- Observe that approximately half of the neurons are not active and are not considered as a part of the NN
- Now NN is no more complex rather observe that the NN becomes simpler.
- A simpler version of the NN results in less complexity that can reduce overfitting.
- Deactivation of neurons with a certain probability P is applied at each forward propagation and weight update step.

Neural Network without (left) and with dropout (right)



Model Evaluation: Dropout Regularization

- At every iteration
  - Dropout operation randomly selects few nodes & removes it along with in and outgoing wt. connections
- So each iteration has a different set of nodes and this results in a different set of outputs.
- It can also be thought of as an ensemble technique in machine learning.
- Ensemble models usually perform better than a single model as they capture more randomness.
- Similarly, dropout also performs better than a normal NN model.
- This probability of choosing how many nodes should be dropped is the hyperparameter of the dropout function.
- Dropout can be applied to both the hidden layers as well as the input layers.
- Due to these reasons, dropout is usually preferred when we have a large/complex NN structure
- The probability (say .25) of dropping can decided for tune the model
- We can tune it further for better results using the grid search method.
- In keras, we can implement dropout using the keras core layer.

Model Evaluation: Regularization

- **Data Augmentation**
- The simplest way to reduce overfitting is to increase the size of the training data
- In machine learning, we were not able to increase the size of training data as the labeled data was too costly.
- In case images, there are a few ways of increasing the size (some transformation) of the training data
    - rotating the image, flipping, scaling, shifting, etc.
- This technique of applying transformation is known as data augmentation
- This usually provides a big leap in improving the accuracy of the model
- It can be considered as a mandatory trick in order to improve our predictions.
- In keras, we can perform all of these transformations using ImageDataGenerator.
    - It has a big list of arguments which you you can use to pre-process your training data.

Model Evaluation: Regularization

- **Early stopping**
- If performance on validation set is getting worse, we stop training immediately. This is **known as early stopping**.
- It is a kind of cross-validation strategy where we keep one part of the training set as the validation set
- Stop training at the dotted line since after that our model will start overfitting on the training data.
- In keras, we can apply early stopping using the callbacks function.
  - from keras.callbacks import EarlyStopping
  - EarlyStopping(monitor='val_err', patience=5)
- Here, monitor denotes the quantity that needs to be monitored and ‘val_err’ denotes the validation error.
- Patience denotes the number of epochs with no further improvement after which the training will be stopped
- After the dotted line, each epoch will result in a higher value of validation error.
- So, patience (5 epochs) after the dotted line, our model will stop because no further improvement is seen.
- Note: We need to take extra care while tuning this hyperparameter
  - As after 5 epochs, the model may starts improving again and the validation error starts decreasing as well.

Summary

- Overfitting occurs in more complex neural network models (many layers, many neurons)
- Complexity of the neural network can be reduced by using L1 and L2 regularization as well as dropout
- L1 regularization
  - forces the weight parameters to become zero
- L2 regularization
  - forces the weight parameters towards zero (but never exactly zero)
- Smaller weight parameters make some neurons neglectable → NN becomes less complex → less overfitting
- During dropout
  - Some neurons get deactivated with a random probability P → NN becomes less complex → less overfitting

Model Improvement (Ensemble Learning)

- Refer slide
  - vsat2k_ML_Ch1c Model Improvement (Ensemble Learning) [PDF]

References

Text books:

1. Ethem Alpaydin, “Introduction to Machine Learning , 4th Edition, The MIT Press, 2020.
2. Peter Harrington, “Machine Learning in Action”, 1st Edition, Dreamtech Press, 2012."
3. Tom Mitchell, “Machine Learning”, 1st Edition, McGraw Hill, 2017.
4. Andreas C, Müller and Sarah Guido, “Introduction to Machine Learning with Python: A Guide for Data Scientists”, 1ed, O'reilly, 2016.
5. Kevin P. Murphy, “Machine Learning: A Probabilistic Perspective”, 1st Edition, MIT Press, 2012."

Reference Books:

6. Aurélien Géron, “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow”, 2nd Edition, Shroff/O'Reilly, 2019.
7. Witten Ian H., Eibe Frank, Mark A. Hall, and Christopher J. Pal., “Data Mining: Practical machine learning tools and techniques”, 1st Edition, Morgan Kaufmann, 2016.
8. Han, Kamber, “Data Mining Concepts and Techniques”, 3rd Edition, Morgan Kaufmann, 2012.
9. Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar, “Foundations of Machine Learning”, 1ed, MIT Press, 2012.
10. H. Dunham, “Data Mining: Introductory and Advanced Topics”, 1st Edition, Pearson Education, 2006.

Thank You.