Deep Learning with R

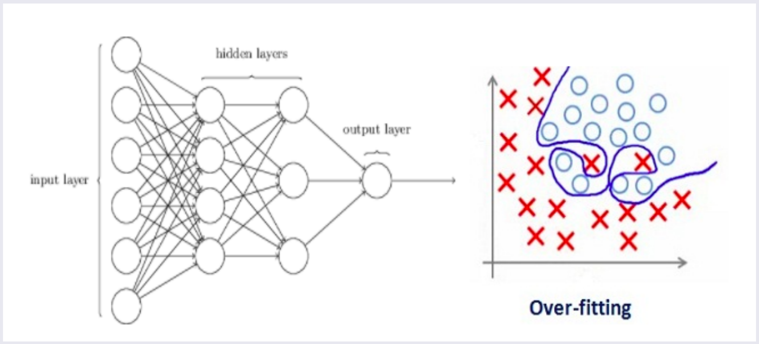
Managing overfitting

Mikhail Dozmorov

Virginia Commonwealth University

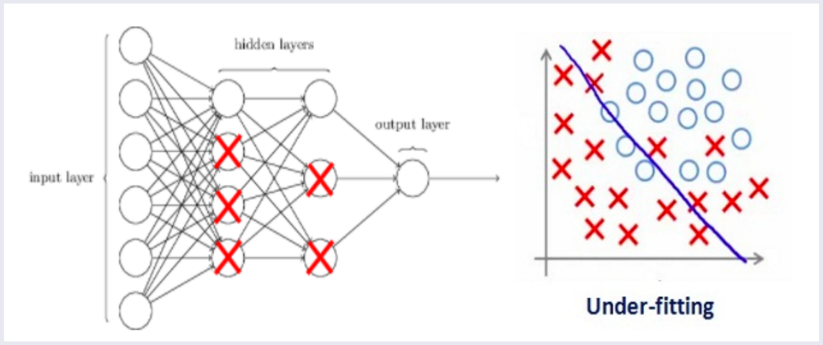
2020-06-10

Overfitting



- The performance of the model on the held-out validation data always peaks after a few epochs and then begins to degrade: the model quickly started to overfit to the training data
- Overfit model is not generalizable - it simply memorizes training data, including noise
- Get more data to avoid memorizing noise

Underfitting

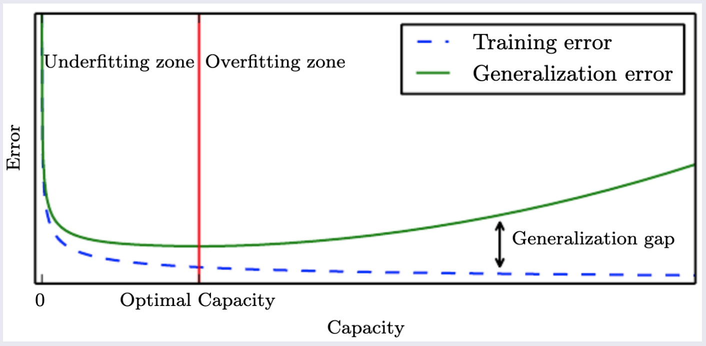


- Until the performance on the held-out validation data did not peak, the model remains underfit
- Less harmful than overfitting, but prevents to have the optimal model

https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

Generalization and model capacity

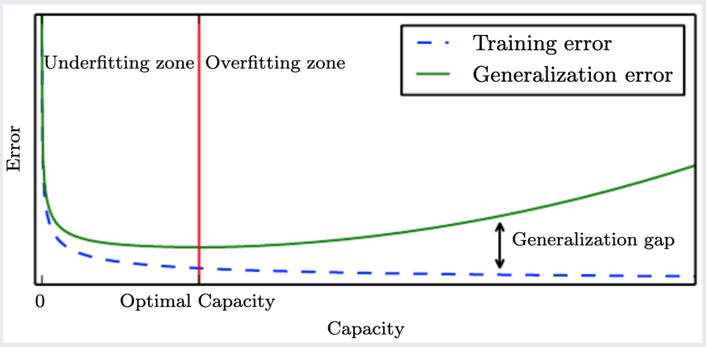
- Generalization is the ability to perform well on previously unobserved inputs
- Initially, the model has low capacity, and training and generalization errors are both high (underfitting)



https://www.deeplearningbook.org/contents/ml.html

Generalization and model capacity

- With training, capacity increases, but the gap between training and generalization error also increases
- Eventually, the size of this gap outweighs the decrease in training error (overfitting)



https://www.deeplearningbook.org/contents/ml.html

Regularization

- Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better
- In machine learning, regularization penalizes the model's coefficients. In deep learning, it penalizes the weight matrices of the nodes
- L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term. *Cost function = Loss (say, binary cross-entropy) + Regularization term*

L1 regulatization

$$Cost\ function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

- λ is the regularization parameter. It is the hyperparameter whose value is optimized for better results.
- The cost added is proportional to the absolute value of the weight coefficients (the L1 norm of the weights). L1 reduces some weights to zero

L2 regularization

$$Cost\ function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

- L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero). The cost added is proportional to the square of the weight coefficients (the L2 norm of the weights)
- L2 is by far the most common and is known as ***weight decay*** in the context of neural nets

https://deeplizard.com/learn/video/iuJgyiS7BKM

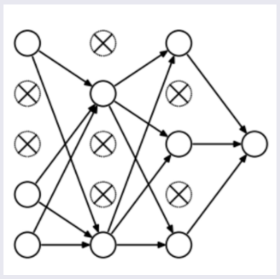
Why regularization helps to prevent overfitting

- Regularization helps to keep weights small
- The behavior of the network with small weights won't change too much if we change a few random inputs
- This makes the network robust against noisy changes, capturing instead patterns which are seen often across the training set
- The unregularized network can use large weights to learn a complex model that carries a lot of information about the noise in the training data

http://neuralnetworksanddeeplearning.com/chap3.html

Dropout

- Dropout was introduced as a regularization method to decrease overfitting in high capacity networks with many layers
- Dropout simulates the *ensembling of many models* by randomly disabling neurons with a probability p during each iteration, forcing the model to learn more robust features



Improving neural networks by preventing co-adaptation of feature detectors by Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2012)

Dropout example

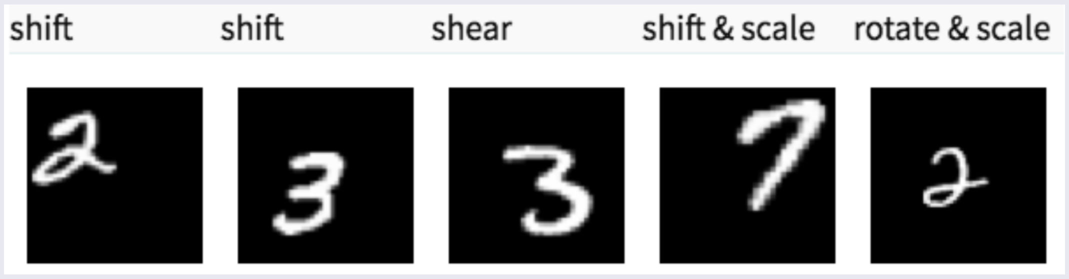
- Drop 50% of randomly selected neurons
- Run forward- and backpropagation through the modified network, update weights of the existing neurons
- Repeat the process, this time dropping another 50% of randomly selected neurons, repeating the process, and update weights for the current set of neurons
- Over several iterations, weights for all neurons will be learned
- When we actually run the full network, that means that twice as many hidden neurons will be active. To compensate for that, we halve the weights outgoing from the hidden neurons

http://neuralnetworksanddeeplearning.com/chap3.html

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929–1958.

Data augmentation

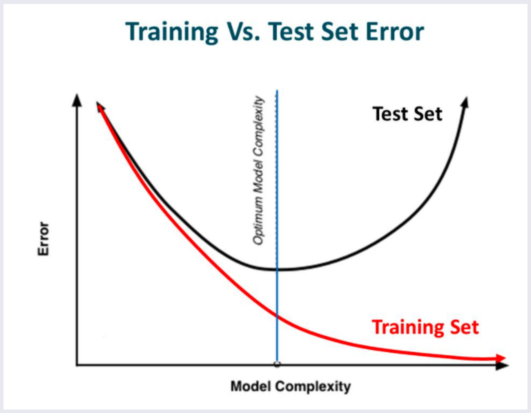
- Frequent operation in image data analysis
- Goal - slightly transform images to generate unseen examples. The model will be forced to learn the general properties of objects, not random fluctuations
- Common transformations: rotation, width/height shift, shear, zoom, flip, fill



https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

Early stopping

- Early stopping is a kind of cross-validation strategy where we keep one part of the training set as the validation set
- When we see that the performance on the validation set is getting worse, we immediately stop the training on the model



https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

Mini-batch training

- Use a small random subset of the training set at each optimization step rather than the full training set
  - Requires a constant amount of memory regardless of the data set size, which allows models to be trained on data sets much larger than the available memory
  - The random fluctuations between batches were demonstrated to improve the model performance by regularization

Batch normalization

- The technique that improves convergence of deep neural networks
- Standardizes input features to each have a mean of zero and variance of one
- Helps to combat overfitting
- Implemented in `layer_batch_normalization()` layer type in Keras, used after a convolutional or densely connected layers
- Frequently used in many network architectures (ResNet50, Inception V3, Xception)

https://d2l.ai/chapter_convolutional-modern/batch-norm.html

Batch normalization

Batch normalization is applied to individual layers, after the affine transformation or convolution, and before the nonlinear activation function

- In each training iteration, for each layer, we first compute its activations as usual
- Then, we normalize the activations of each node by subtracting its mean and dividing by its standard deviation estimating both quantities based on the statistics of the current minibatch

$$BN(x) = \gamma \bigodot \frac{x - \hat{\mu}}{\hat{\sigma}} + \beta$$

- γ - coordinate-wise scaling coefficient, β - offset

https://d2l.ai/chapter_convolutional-modern/batch-norm.html

https://deeplizard.com/learn/video/dXB-KQYkzNU

Hyperparameter tuning

- Numerous choices in selecting network architecture
  - Number of layers, and their configuration (network topology)
  - How many units or filters in each layer
  - Which activation function(s) is the best for a given problem
  - Should you use batch normalization/dropout, or other regularization techniques
  - Which loss function to use
  - How to choose a (constant or variable) learning rate

Hyperparameters

- **Hyperparameters** - parameters controlling the complexity of machine learning algorithms, to achieve a best bias-variance tradeoff
  - **Grid search** - (systematically or random) search across many combinations of hyperparameters
  - **Early stopping** - stop a grid search once the reduction in the error stops marginally improving
  - Active research field

https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

https://towardsdatascience.com/simple-guide-to-hyperparameter-tuning-in-neural-networks-3fe03dad8594

Ensemble deep learning

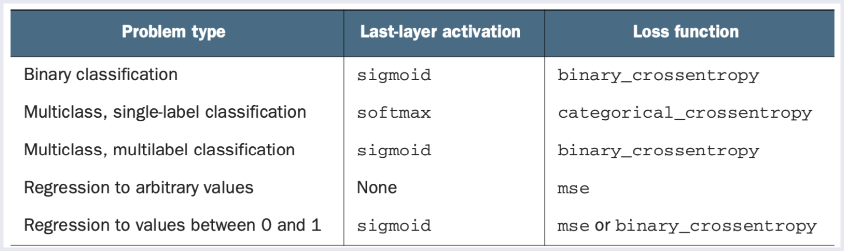
- As in machine learning, ensembling the models is a powerful technique to obtain the best possible results
- Pooling together the predictions from multiple models under the assumption that different good models trained independently are likely to capture various aspects of the data
- Select the best performing and the most different models to build an ensemble, discard the rest
- Ensembling is discouraged in publications as it inflates the performance

Universal Deep Learning workflow

- **Defining the problem and assembling a dataset**
  - (Binary/multiclass/multilabel) classification? (Scalar/vector) regression?
- **Choosing a measure of success**
  - Accuracy? AUROC? Class imbalance?
- **Deciding on an evaluation protocol**
  - (Hold-out/k-fold) cross-validation?
- **Preparing your data**
  - Data as tensors, staled to small values ([-1, 1] or [0, 1] range)

Universal Deep Learning workflow

- **Developing a model that does better than a baseline**
  - Activation function (last layer in particular), loss function, optimizer



Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

- **Scaling up: developing a model that overfits**
  - Add layers, neurons/units per layer, train for longer
- **Regularizing your model and tuning your hyperparameters**
  - Add dropout, L2/L1 regularization, tweak layers