Structuring Machine Learning Projects Course

Best Practice 1 - Orthogonalization: ML experts who design learning algorithms have in mind orthogonalization. In orthogonalization each parameter or setting can be tuned to achieve a single and specific effect. So, depending our problem, each time we can tune the appropriate parameter to solve it. A real-life example is the TV settings where we tune multiple independent parameters for setting the image of the display.

For example in a supervised learning problem:

- **To fit better the training set** (to have low bias) we can use a bigger network with enough capacity, use a better learning rate, use a modern optimizer (Adam, RMSprop), etc.

- **To fit better the validation set** (to have low variance) we can use regularization, use a bigger training set, etc.

- **To fit better the test set** (to have good unbiased estimate) we can use a bigger validation set.

- **To perform well to real world** we can use either a more appropriate cost function to measure correctly our error or more appropriate validation / test sets that are more realistic for the problem we want to solve.

Early stopping is less orthogonalized cause it affects both the training and validation error.

Best Practice 2 - Single real value evaluation metric: Using a single real value evaluation metric can really help to compare easily models that try new ideas. For example, in a classification problem instead of using the precision and recall metrics we can use the f1-score evaluation metric. Using a single evaluation metric helps to fast and automatically select the best from N candidate models. Another case as an evaluation metric could be to use the average performance of a model for data from different geographical locations.

Best Practice 3 – Use Optimization and Satisfying metrics when using multiple evaluation metrics: It is not always possible to combine all metrics we care into a single evaluation metric (e.g. precision+recall metrics as f1-score or average various metrics in a single value). When we have to use N evaluation metrics that cannot be combined into a single one then we have to select one of them as the **optimization metric** (the one that we actually care more and want to be as good as possible, e.g. f1-score) and the rest N-1 as **satisfying metrics** that must obey some constraints (just perform better than some threshold value, e.g. running time of the algorithm < 100 ms). With this technique we can still automatically select the best model from all the candidate models by finding first the models that satisfy the satisfying metrics and then select from them the best model based on the optimization metric.

Best Practice 4 - Use same validation/test data distributions and metrics: Never use different evaluation metrics or data distributions for validation / test sets. Having different data distributions or evaluation metrics is like trying to build a model to solve the wrong problem. Whenever we set an evaluation metric and a validation set we define a target to aim. After defining the target we can try to hit it by searching for a good model. However, when we use a data distribution for test set that is different from validation set or use a different evaluation metric then we actually change the aim target (the problem to solve). Both the validation / test distributions should come from the same distribution. We can shuffle a single dataset and sample randomly from it to create the validation and test sets. Furthermore, the training set doesn’t have to come from the same distribution of

validation/test sets but is important of how well we can hit the aim target and solve the problem. **The validation and test sets should come from the same distribution and should reflect the real data we are going to handle in the future.**

Best Practice 5 - Use reasonably split ratios for training/validation/test sets: In old traditional ML dataset sizes we use the 70%/30% (for training/validation sets) or 60%/20%/20% (for training/validation/test sets) split ratios. However, in deep learning areas because the datasets are very large we use most of the data for training set and a small fraction (smaller than 20% or 30%) for validation and/or test sets.

Best Practice 6 - Place the correct aim target to solve from the beginning or shift the problem if there is a need: When we set an evaluation metric and validation / test sets we actually define a target for a problem to solve. It is very possible for the problem to change due to the covariance shift problem (when data from the real world change distribution, e.g. users upload photo of lower resolution) and we need to adapt either the evaluation metric or the validation / test sets to represent more the correct data. For example, in case a cat photo detector classifies non-cat pornographic images as cats (false positives) we might think to redefine the evaluation metric and use a bigger weight to the pornographic images specifically. Also, sometimes we do not use good validation / test sets and we are solving a different problem from the very beginning. **So, if doing well on your evaluation metric + validation / test sets does not correspond to doing well on your application, we need to change our metric and/or validation / test sets.**

Best Practice 7 - Have a baseline error to compare with: All ML problems have a theoretical error threshold (called Bayesian) that cannot be reached. For many deep learning applications (computer vision, NLP, speech recognition) the human-level error is an estimate or proxy of the theoretical Bayesian error. While searching for a model we can make progress fast and easily reach or surpass the human-level error but it is way more difficult to superpass the Bayesian error. For any ML application it is important to have a level of baseline error (e.g. human-level error) to compare with. By analyzing the model's train/validation/test and baseline error we can decide on how to invest our time to improve the model (e.g. improve training error to reach the baseline error by reducing the avoidable bias or improve validation error by reducing variance). Fit well the training set using a complex model with enough capacity by reducing the model bias error (various techniques exist to achieve it). In case there is a baseline error try to reduce the avoidable bias. Also, fit well the validation set by reducing the variance error of the model (various techniques exist to achieve it). However, by the time we surpass the baseline error we no longer are able to know how to invest our time to improve the model.

Best Practice 8 - Error analysis of misclassified examples: Manually examine misclassified examples (false positives / negatives) and try to understand where to invest time and effort so that the classifier will be improved drastically. For example, if we have a classifier with 10% misclassification error on validation set we can use a random sample of misclassified examples to group/count by category (e.g. blurry images / instagram filtered images / snapchat filtered images) and estimate the relative improvement in the misclassification error in case of investing time to improve each category of misclassified images (e.g. in a sample of 100 validation misclassified images if 5% are blurry and 95% are Instagram-like photos we should invest time to change the model for fixing the Instagram-like case cause it gives more value). Also, this technique is a good way to answer a question like "Will improve the classifier drastically if we classify more correctly the blurry images?"

What to do the examples that are incorrectly labeled in training and/or validation/test set:

- **For all sets:** We can **manually fix** the incorrectly labeled examples as long as we want to invest time and effort to better train or evaluate a classifier model.

- **For training set:** As long as the incorrect labeled examples for each class label are a small percentage of the total dataset and are **random errors** we can leave them as is because many deep learning algorithms are robust to input random noise and can still learn from noisy data. However, ML models are not robust to **systematic errors**. For example, for an animal classification problem if there is a single white dog incorrectly classified as bird we can assume that it is a random error (random human error) and leave it as is. However, if a noteworthy percentage of white dogs are classified as birds then this probably is a systematic error and will affect the classifier.

- **For validation/test set:** We can also perform **error analysis** using a sample of misclassified examples form validation set for the category of incorrectly labeled examples and evaluate if it is worthy to invest time to fix the labels of the examples. For example, if a very small fraction of misclassified images are images with wrong labels we might not want to invest time for fixing them. However, if the fraction is big then by fixing it we will see a major improvement in validation and/or test metrics.

Best Practice 9 - Build fast a model and iterate appropriate to improve it: Most teams overthink a problem rather than build something simple, quick and dirty. It is wiser to build fastly a prototype model and then iterate to improve it. We should first define the validation/test sets and validation metrics to place the aim target and then train a model with a training set to hit the target. After the first version we can get intuitions on how to invest time and how to iterate to improve it. For example, we can either perform error analysis on misclassified examples or bias/variance analysis to reduce bias and variance errors.

Best Practice 10 - Training and validation/test sets distributions: We can use a training set with same distribution as validation/test sets or different distribution which is comprised of data from different sources (e.g. cat photos from Instagram, Snapchat, WEB, movies, cartoons, professional photography, etc). Whatever the case, what we care most is to perform well on the validation/test sets. When we have a small real-world dataset (data from the real problem we are trying to solve) but we want to train a deep learning model we can use a training set with different distribution. For example, we can crawl the WEB to obtain 200K of training face images from different distributions in order to train a face detector. However, after the training we should evaluate the face detector using the 10K real-world face images from validation/test sets. We can use split ratios as: a) training=200K, validation=5K, test=5K or b) training=205K (we use some of the real-world data), validation=2.5K, test=2.5K.

Best Practice 11 – Use training/validation set to check if validation error is due to variance or data mismatch: When using different training and validation/test distributions it is possible the validation error to be much higher than the training error either due to data mismatch problem and/or variance problem. One way to discover the exact reason is to split the training set and create an extra training-validation set that will be used only to check if there is a variance problem. If the training-validation error is much higher than the training error then we are sure that we have a variance problem. However, if it is small and the validation set error remain big then we can be sure that it is due the data mismatch problem. Furthermore, it is possible to have both errors (variance and data mismatch).

Kind of errors we can investigate with bias/variance error analysis:

1. Baseline error (e.g. human-level error)

Between difference error due to avoidable bias
2. Training error
Between difference error due to model's variance
3. Training-validation error
Between difference error due to data mismatch
4. Validation error
Between difference error due to validation set overfitting
5. Test error

Best Practice 12 - How to solve the problem of data mismatch: When there is a big validation error due to training/validation data mismatch problem we can reduce the error by making the training set distribution more similar to the validation set distribution. One way to achieve it is to make error analysis in the validation set and try to find categories of examples that it would be nice to exist in the training set so that the model can learn and generalize well to the validation set. The training set can be enhanced with new examples using artificial data synthesis techniques.

Transfer learning & Fine-tuning

In many applications – e.g. computer vision – we cannot train a complex and deep model for two reasons: a) we do not have enough computational power for training large no. of weights and b) the model will overfit when the dataset is small. One way to solve this problem is to use a pre-trained and well-generalized model that is already trained on a similar task, change the output layer for the new task, and try to either train the model for the new task either by **transfer the pre-learned knowledge as fixed or by fine tuning**.

Transfer knowledge: When we replace the last layer(s) of a pre-trained model with new randomly initialized layer(s) and then train the model for the new task by keeping only the earlier layers fixed (transfer knowledge). We usually do this when we have a small dataset.

Fine-tuning: When we replace the last layer(s) of a pre-trained model with new randomly initialized layer(s) and then train the model for the new task by training (fine-tuning) all the layers. We usually do this when we have a big dataset.

Example: Using an already pre-trained face classifier that was trained on thousands of celebrity images (task A) and which has learned how to detect lines, dots, curves, edges or small parts of facial characteristics (has learned to see faces) can really help to learn faster or learn a different task of face recognition (task B) that uses a smaller dataset for a specific target of people. We usually train first a face classifier with lots of face images as a general baseline for transfer learning and then we use it to create more targeted applications with datasets with smaller set of images. We do this because training a complex deep model using a small dataset will overfit for sure. If we have a big dataset of face images for the task B it has less meaning to do transfer learning.

Multi-task learning (e.g. object detection in computer vision): When we train a NN to map a single input example to multiple output labels, thus it is like **solving multiple tasks simultaneously** (e.g. detect if single image contains at the same time a dog, a cat, a mouse, etc). It is different from transfer learning which is a sequential process where we start by solving first the task A (e.g. detect dog) and then transfer to a task B (e.g. detect cat) and then combine all these different models to solve the multi-task problem. Multi-task learning is used when we want to predict from a single example multiple labels at the same time. So, the $y^{(i)}$ vector of an example $x^{(i)}$ **is not encoded with the one-hot encoding but can have multiple 1's** for different labels (e.g. y = [1 0 0 1 1 0]). Also, despite the fact that softmax activation function squeezes inputs to [0-1] range we don't use it cause all output probabilities sum to 1. **In place of softmax is used the simple sigmoid activation**

function. Also, although it is possible to train different NNs for each separate label and use them afterward to solve the multi-task problem, we can train a single NN to solve it using multi-task learning. The advantage of multi-task learning: When we have multiple labels with small number of examples, instead of building **different medium-performance overfitted NNs** for classifying the different labels we can build a **single high-performance complex NN** that solves the whole multi-task problem. Trying to learn one task can help learning also the other tasks. For example, it is possible for **low-level learned features from the earlier layers of the NN to be used to the next layers and help for the classification of various labels** (e.g. common features such as lines, curves, edges can be used to help the detection of dog, cat and mouse labels). Multi-task learning can be used even with examples that have some labels but we should skip from the loss the labels with unknown value.

End-to-end deep learning: As long as we have **enough data** we can map from input X to output Y directly using a **deep complex** NN, instead of using a pipeline that uses various intermediate preprocessing stages (e.g. hand-crafted feature extractors, filters, etc). One example of end-to-end learning is CNNs which are able to discover convolutional feature extractors automatically while training and can replace older hand-crafted feature extractors. **Because end-to-end learning uses deep learning it needs large no. of examples to work well. If this isn't possible and we have a small dataset the traditional approach of pipeline systems should be preferred.** One example, is using a face recognition system that is composed using a face detector and a face classifier where we first need to detect the faces in the captured image and then try to classify them. If we need to replace this pipeline using an NN using end-to-end learning we need a large no. of examples to mapping directly input to output.