|  |
| --- |
| Introduction to ML Strategy |
| Setting up your goal |
| Comparing to human-level performance |
| Error Analysis |
| Mismatched training and dev/test set |
| Learning from multiple tasks |
| End-to-end deep learning |
|  |

|  |
| --- |
| **Introduction to ML Strategy**  Why we need good ML strategy for ML/DL projects.  Let way we are working on a cat detection application and we achieved 90% accuracy at test set. To improve the accuracy further there can n-number of options to try like   * Collect more data * Collect more diverse training set * Train algorithm longer with gradient descent * Try Adam instead of GD * Try bigger network * Try smaller network * Try dropout * Add L\_2 regularization * Network architecture change (activation functions, #hidden units …)   If we randomly try any option that does not seems very good approach which can improve the productivity of the project to achieve the desired result faster. So, a good strategy helps a lot for ML projects.  **Orthogonalization**  To understand the concept of orthogonalization let see the below example.  In old television system we had some dedicated nob to adjust display. Where each knob had some dedicated functions like widen/tilt/zoom etc. With this by adjusting different knob we can finally adjust the display of the television.  Similarly, consider driving a car. We have steering to take the direction and acceleration / brake for movement.  What if in case of TV we have only one nob which has functionality like 0.1\*zoom + 0.5 \* widen + 0.8\*tilt . Then it would be very difficult to adjust the display by adjusting only one knob. Likewise, let say we have a car where we have only ne action which can create 0.5\* direction change + 0.1 \* acceleration  Then, it would be very difficult to drive the car.  In machine learning project also, if we can have control of individual metrics that we want to change would be best option for having faster optimization.  Scientifically, 2 vectors are said to be orthogonal to each other if they are 90 deg apart from each other. |
|  |
| **Chain of assumption in ML**  When a supervised learning system is design, these are the 4 assumptions that needs to be true and orthogonal.  1. **Fit training set well in cost function**  - If it doesn’t fit well, the use of a bigger neural network or switching to a better optimization algorithm might help.  2. **Fit development set well on cost function**  - If it doesn’t fit well, regularization or using bigger training set might help.  3. **Fit test set well on cost function**  - If it doesn’t fit well, the use of a bigger development set might help  4. **Performs well in real world**  - If it doesn’t perform well, the development test set is not set correctly or the cost function is not evaluating the right thing. |
| **Using single number evaluation metric**  While designing deep learning system, we come across many performance metrics. It would be difficult to decide the direction of the project if we follow many metric. In practice if we can come up with single scaler evaluation metric for the system that helps to quickly iterate and decide the next action.  Let say we have precision and recall for one Deep learning model. Looking into below result it would be difficult to say which model is better.  Example:    In this case, if we come up with another metrics which represents both other 2 metrices then we can only follow this new metric. Like F1 score (harmonic mean)    So looking into this table now it is easy for us to say that Model A is better than B so we will try to improve model A  Another example, we are designing a cat detection app and we have below error for different geos. So in this case we should add another column average and follow the average instead of individual country-wise results |
| **Satisfying and optimizing metric:**  There are different metrics to evaluate the performance of a classifier, they are called evaluation matrices. They can be categorized as satisficing and optimizing matrices. It is important to note that these evaluation matrices must be evaluated on a training set, a development set or on the test set.  Example: Cat vs Non-cat          In this case, accuracy and running time are the evaluation matrices. Accuracy is the optimizing metric, because you want the classifier to correctly detect a cat image as accurately as possible. The running time which is set to be under 100 ms in this example, is the satisficing metric which mean that the metric must meet expectation set.  The general rule is:    𝑁𝑚𝑒𝑡𝑟𝑖𝑐: 1 𝑂𝑝𝑡𝑖𝑚𝑖𝑧𝑖𝑛𝑔 𝑚𝑒𝑡𝑟𝑖𝑐  𝑁𝑚𝑒𝑡𝑟𝑖𝑐 − 1 𝑆𝑎𝑡𝑖𝑠𝑓𝑖𝑐𝑖𝑛𝑔 𝑚𝑒𝑡𝑟𝑖𝑐 |
| **Training, development and test distributions**  Setting up the training, development and test sets have a huge impact on productivity. It is important to choose the development and test sets from the same distribution and it must be taken randomly from all the data.  Guideline  Choose a development set and test set to reflect data you expect to get in the future and consider important to do well. |
| **Size of the development and test sets**  Old way of splitting data  We had smaller data set therefore we had to use a greater percentage of data to develop and test ideas and models.    **Or**        Modern era – Big data Now, because a large amount of data is available, we don’t have to compromised as much and can use a greater portion to train the model.        Guidelines  • Set up the size of the test set to give a high confidence in the overall performance of the system.  • Test set helps evaluate the performance of the final classifier which could be less 30% of the whole data set.  • The development set must be big enough to evaluate different ideas. |
| **Setting up your goal**  When to change development/test sets and metrics  Example: Cat vs Non-cat  A cat classifier tries to find a great amount of cat images to show to cat loving users. The evaluation metric used is a classification error.   |  |  | | --- | --- | | Algorithm | Classification error [%] | | A | 3% | | B | 5% |     It seems that Algorithm A is better than Algorithm B since there is only a 3% error, however for some reason, Algorithm A is letting through a lot of the pornographic images.  Algorithm B has 5% error thus it classifies fewer images but it doesn't have pornographic images. From a company's point of view, as well as from a user acceptance point of view, Algorithm B is actually a better algorithm. The evaluation metric fails to correctly rank order preferences between algorithms. The evaluation metric or the development set or test set should be changed.  The misclassification error metric can be written as a function as follow:  𝑚𝑑𝑒𝑣  𝐸𝑟𝑟𝑜𝑟  ∑ ℒ{(𝑦̂(𝑖) ≠ 𝑦(𝑖)}  𝑚𝑑𝑒𝑣  𝑖=1  This function counts up the number of misclassified examples.  The problem with this evaluation metric is that it treats pornographic vs non-pornographic images equally. On way to change this evaluation metric is to add the weight term 𝑤(𝑖).  (𝑖) = { 10 1 𝑖𝑓𝑖𝑓 𝑥𝑥((𝑖𝑖)) 𝑖𝑠𝑖𝑠 𝑝𝑜𝑟𝑛𝑜𝑔𝑟𝑎𝑝𝑛𝑜𝑛 − 𝑝𝑜𝑟𝑛𝑜𝑔𝑟𝑎𝑝ℎ𝑖𝑐 ℎ𝑖𝑐  𝑤  The function becomes:  𝑚𝑑𝑒𝑣  𝐸𝑟𝑟𝑜𝑟  𝑖=1  Guideline  1. Define correctly an evaluation metric that helps better rank order classifiers 2. Optimize the evaluation metric |
| **Comparing to human-level performance Comparing to human-level performance**  Why human-level performance?    Today, machine learning algorithms can compete with human-level performance since they are more productive and more feasible in a lot of application. Also, the workflow of designing and building a machine learning system, is much more efficient than before.  Moreover, some of the tasks that humans do are close to ‘’perfection’’, which is why machine learning tries to mimic human-level performance.  The graph below shows the performance of humans and machine learning over time.  The    Bayes optimal  error    Human  s    Machine  Learning    Machine learning progresses slowly when it surpasses human-level performance. One of the reason is that human-level performance can be close to Bayes optimal error, especially for natural perception problem.  Bayes optimal error is defined as the best possible error. In other words, it means that any functions mapping from x to y can’t surpass a certain level of accuracy.  Also, when the performance of machine learning is worse than the performance of humans, you can improve it with different tools. They are harder to use once its surpasses human-level performance.  These tools are:   * Get labelled data from humans * Gain insight from manual error analysis: Why did a person get this right? * Better analysis of bias/variance. |
| Avoidable bias  By knowing what the human-level performance is, it is possible to tell when a training set is performing well or not.  Example: Cat vs Non-Cat     |  |  |  | | --- | --- | --- | |  | Classification error (%) | | | Scenario A | Scenario B | | Humans | 1 | 7.5 | | Training error | 8 | 8 | | Development error | 10 | 10 |     In this case, the human level error as a proxy for Bayes error since humans are good to identify images. If you want to improve the performance of the training set but you can’t do better than the Bayes error otherwise the training set is overfitting. By knowing the Bayes error, it is easier to focus on whether bias or variance avoidance tactics will improve the performance of the model. Scenario A There is a 7% gap between the performance of the training set and the human level error. It means that the algorithm isn’t fitting well with the training set since the target is around 1%. To resolve the issue, we use bias reduction technique such as training a bigger neural network or running the training set longer. Scenario B The training set is doing good since there is only a 0.5% difference with the human level error. The difference between the training set and the human level error is called avoidable bias. The focus here is to reduce the variance since the difference between the training error and the development error is 2%. To resolve the issue, we use variance reduction technique such as regularization or have a bigger training set. |
| Understanding human-level performance  Human-level error gives an estimate of Bayes error.  Example 1: Medical image classification  This is an example of a medical image classification in which the input is a radiology image and the output is a diagnosis classification decision.   |  |  | | --- | --- | |  | Classification error (%) | | Typical human | 3.0 | | Typical doctor | 1.0 | | Experienced doctor | 0.7 | | Team of experienced doctors | 0.5 |     The definition of human-level error depends on the purpose of the analysis, in this case, the Bayes error is lower or equal to 0.5%.  Example 2: Error analysis   |  |  |  |  | | --- | --- | --- | --- | |  | Classification error (%) | | | | Scenario A | Scenario B | Scenario C | | Human (proxy for Bayes error) | 1 | 1 | 0.5 | | 0.7 | 0.7 | | 0.5 | 0.5 | | Training error | 5 | 1 | 0.7 | | Development error | 6 | 5 | 0.8 |  Scenario A In this case, the choice of human-level performance doesn’t have an impact. The avoidable bias is between 4%-4.5% and the variance is 1%. Therefore, the focus should be on bias reduction technique. Scenario B In this case, the choice of human-level performance doesn’t have an impact. The avoidable bias is between 0%-0.5% and the variance is 4%. Therefore, the focus should be on variance reduction technique. Scenario C In this case, the estimate for Bayes error must be 0.5% since you can’t go lower than the human-level performance otherwise the training set is overfitting. Also, the avoidable bias is 0.2% and the variance is 0.1%. Therefore, the focus should be on bias reduction technique.   Summary of bias/variance with human-level performance  * Human - level error – proxy for Bayes error * If the difference between human-level error and the training error is bigger than the difference between the training error and the development error. The focus should be on bias reduction technique   If the difference between training error and the development error is bigger than the difference between the human-level error and the training error. The focus should be on variance reduction technique |
| Surpassing human-level performance  Example1: Classification task     |  |  |  | | --- | --- | --- | |  | Classification error (%) | | | Scenario A | Scenario B | | Team of humans | 0.5 | 0.5 | | One human | 1.0 | 1 | | Training error | 0.6 | 0.3 | | Development error | 0.8 | 0.4 |   Scenario A  In this case, the Bayes error is 0.5%, therefore the available bias is 0.1% et the variance is 0.2%.  Scenario B  In this case, there is not enough information to know if bias reduction or variance reduction has to be done on the algorithm. It doesn’t mean that the model cannot be improve, it means that the conventional ways to know if bias reduction or variance reduction are not working in this case.    There are many problems where machine learning significantly surpasses human-level performance, especially with structured data:   * Online advertising * Product recommendations * Logistics (predicting transit time) * Loan approvals |
| Improving your model performance    The two fundamental assumptions of supervised learning    There are 2 fundamental assumptions of supervised learning. The first one is to have a low avoidable bias which means that the training set fits well. The second one is to have a low or acceptable variance which means that the training set performance generalizes well to the development set and test set.  If the difference between human-level error and the training error is bigger than the difference between the training error and the development error, the focus should be on bias reduction technique which are training a bigger model, training longer or change the neural networks architecture or try various hyperparameters search.  If the difference between training error and the development error is bigger than the difference between the human-level error and the training error, the focus should be on variance reduction technique which are bigger data set, regularization or change the neural networks architecture or try various hyperparameters search.   Summary   •  More data    •  Regularization    •  Neural Networks architecture/hyperparameters search    •    •  Train bigger model    •  Train longer, better optimization algorithms    •  Neural Networks architecture/hyperparameters sear  ch |
| **Error Analysis**  Let say we are working on a cat detector system where we are getting 90% accuracy or 10% error. We want to improve the performance further. So, what would be the efficient way to reduce the error  Instead of randomly following any technique if we analyse the error and then take decision that would improve efficiency of the team.  Like in this case we can follow the below approach   * Get ~ 100 misclassified dev set examples * Manually count how many are dogs   After evaluating if we see that only 5 out of 100 counts for dog misclassification.  Hence, if we resolve this problem then we can reduce the error max 0.5% - called **ceiling**    Another approach can be trying many ideas in parallel.   * Fix pictures of dogs being recognized as cats * Fix great cats (lions, panthers, etc..) being misrecognized * Improve performance on blurry images   To evaluate we can create a table as follows to see which category of the problem counts how much percent of the overall error    From this evaluation if we address Dog or Instagram category that would not help that much if we do for Great car/ Blurry issue |
| **Cleaning up incorrectly labelled data:**  Let say we have a cat detecting model where we see there are some mislabelled data in the dev set. If this mislabel is random in nature in training set on which the model has learnt, then there is not much point of worry. But if we have the data set where we have systematic error (like all white dogs are mislabelled as cat) can cause trouble for the learning of the model  Now, once we know that there are some error in the dev set due to mislabel then what we should do to correct this.   * We should create / update the table with one column having count of dev set error due to mislabel * We calculate percentage of mislabel error on overall dev set error * If that count for significant amount then may be it is a good option to correct the mislabel in the dev set otherwise we should look into other options first     Like for the above case, if we have   |  |  |  | | --- | --- | --- | |  | Scenario 1 | Scenario 2 | | Overall dev set error | 10% | 2% | | Errors due to incorrect label | 0.6% | 0.6% | | Errors due to other causes | 9.4% | 1.4% | | Conclusion | It is not good idea to start fixing label first | We can work on mislabel to improve further |   Guideline:   * Apply same process to your dev and test sets to make sure they continue to come from the same distribution * Consider examining examples your algorithm got right as well as ones it got wrong (not always practically followed) * Train and dev/test data may now come from slightly different distributions |
| Build system quickly, then iterate  Depending on the area of application, the guideline below will help you prioritize when you build your system. Guideline  1. Set up development/ test set and metrics    * Set up a target 2. Build an initial system quickly    * Train training set quickly: Fit the parameters    * Development set: Tune the parameters    * Test set: Assess the performance 3. Use Bias/Variance analysis & Error analysis to prioritize next steps |
| Training and testing on different distributions  Example: Cat vs Non-cat  In this example, we want to create a mobile application that will classify and recognize pictures of cats taken and uploaded by users.  There are two sources of data used to develop the mobile app. The first data distribution is small, 10 000 pictures uploaded from the mobile application. Since they are from amateur users, the pictures are not professionally shot, not well framed and blurrier. The second source is from the web, you downloaded 200 000 pictures where cat’s pictures are professionally framed and in high resolution.  The problem is that you have a different distribution:  1- small data set from pictures uploaded by users. This distribution is important for the mobile app. 2- bigger data set from the web.  The guideline used is that you have to choose a development set and test set to reflect data you expect to get in the future and consider important to do well.  The data is split as follow:            Test    set    Development    set    5    000    5    000      Training set    5    000      205 000    Web    App    App    The advantage of this way of splitting up is that the target is well defined.  The disadvantage is that the training distribution is different from the development and test set distributions. However, this way of splitting the data has a better performance in long term. |
| Bias and variance with mismatched data distributions Example: Cat classifier with mismatch data distribution When the training set is from a different distribution than the development and test sets, the method to analyse bias and variance changes.   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  | Classification error (%) | |  |  | | Scenario A | Scenario B | Scenario C | Scenario D | Scenario E | Scenario F | | Human (proxy for Bayes error) | 0 | 0 | 0 | 0 | 0 | 4 | | Training error | 1 | 1 | 1 | 10 | 10 | 7 | | Training-development error | - | 9 | 1.5 | 11 | 11 | 10 | | Development error | 10 | 10 | 10 | 12 | 20 | 6 | | Test error | - | - | - | - | - | 6 |    Scenario A If the development data comes from the same distribution as the training set, then there is a large variance problem and the algorithm is not generalizing well from the training set.  However, since the training data and the development data come from a different distribution, this conclusion cannot be drawn. There isn't necessarily a variance problem. The problem might be that the development set contains images that are more difficult to classify accurately.  When the training set, development and test sets distributions are different, two things change at the same time. First, the algorithm trained in the training set but not in the development set. Second, the distribution of data in the development set is different.    It's difficult to know which of these two changes what produces this 9% increase in error between the training set and the development set. To resolve this issue, we define a new subset called training-development set. This new subset has the same distribution as the training set, but it is not used for training the neural network.   Scenario B   The error between the training set and the training- development set is 8%. In this case, since the training set and training-development set come from the same distribution, the only difference between them is the neural network sorted the data in the training and not in the training development. The neural network is not generalizing well to data from the same distribution that it hadn't seen before Therefore, we have really a variance problem.  Scenario C  In this case, we have a mismatch data problem since the 2 data sets come from different distribution.  Scenario D  In this case, the avoidable bias is high since the difference between Bayes error and training error is 10 %. Scenario E In this case, there are 2 problems. The first one is that the avoidable bias is high since the difference between Bayes error and training error is 10 % and the second one is a data mismatched problem. Scenario F Development should never be done on the test set. However, the difference between the development set and the test set gives the degree of overfitting to the development set. General formulation         Bayes error    Training set error    Development set error    Test set error    Development  -    Training set error    Avoidable Bias    Variance    Data mismatch    Degree of  overfitting to    the development set |
| Addressing data mismatch    This is a general guideline to address data mismatch:   * Perform manual error analysis to understand the error differences between training, development/test sets. Development should never be done on test set to avoid overfitting.      * Make training data or collect data like development and test sets. To make the training data more like your development set, you can use is artificial data synthesis. However, it is possible that if you might be accidentally simulating data only from a tiny subset of the space of all possible examples. |
| Transfer Learning  Transfer learning refers to using the neural network knowledge for another application. When to use transfer learning  * Task A and B have the same input 𝑥 * A lot more data for Task A than Task B * Low level features from Task A could be helpful for Task B  Example 1: Cat recognition - radiology diagnosis The following neural network is trained for cat recognition, but we want to adapt it for radiology diagnosis. The neural network will learn about the structure and the nature of images. This initial phase of training on image recognition is called pre-training, since it will pre-initialize the weights of the neural network. Updating all the weights afterwards is called fine-tuning.  For cat recognition  Radiology diagnosis  Input 𝑥: Radiology images – CT Scan, X-rays  Output 𝑦 : Radiology diagnosis – 1: tumour malign, 0: tumour benign    Guideline   * Delete last layer of neural network * Delete weights feeding into the last output layer of the neural network * Create a new set of randomly initialized weights for the last layer only • New data set (𝑥, 𝑦) |
| Multi-task learning  Multi-task learning refers to having one neural network do simultaneously several tasks. When to use multi-task learning  * Training on a set of tasks that could benefit from having shared lower-level features * Usually: Amount of data you have for each task is quite similar * Can train a big enough neural network to do well on all tasks  Example: Simplified autonomous vehicle The vehicle must detect simultaneously several things: pedestrians, cars, road signs, traffic lights, cyclists, etc. We could have trained four separate neural networks, instead of train one to do four tasks. However, in this case, the performance of the system is better when one neural network is trained to do four tasks than training four separate neural networks since some of the earlier features in the neural network could be shared between the different types of objects.  The input 𝑥(𝑖) is the image with multiple labels  The output 𝑦(𝑖) has 4 labels which are represents:   1. Pedestrians   𝑦(𝑖) = [1] Cars   1. Road signs - Stop   0 Traffic lights      | | | | 𝑌 = (4, 𝑚)  𝑌 = [𝑦(1) 𝑦(2) 𝑦(3) 𝑦(4) ]  𝑌 = (4,1)  | | | |    Neural Network architecture      To train  this neural network, loss function is  defined as follow:    Pedestrians    Car  s    Traffic lights    Road signs  -    Stop    1  −  𝑚  ∑  ∑  (  𝑦  𝑗  (  𝑖  )  log  (  𝑦  ̂  𝑗  (  𝑖  )  )  +  (  1  −    𝑦  𝑗  (  𝑖  )  )  log  (  1  −  𝑦  ̂  𝑗  (  𝑖  )  )  )  4  𝑗  =  1  𝑚  𝑖  =  1      Also, the cost can be computed such as it is not influenced by the fact that some entries are not labelled. Example:   |  |  |  |  | | --- | --- | --- | --- | | 1  0  𝑌 = [  0  ? | 0  1  1  0 | ?  ?  ?  1 | ?  0  ]  1  0 | |
| What is end-to-end deep learning  End-to-end deep learning is the simplification of a processing or learning systems into one neural network.  Example - Speech recognition model The traditional way - small data set Extract Features    Phonemes    Words    Audio    Transcript     The hybrid way - medium data set Phonemes    Words    Audio    Transcript       The End-to-End deep learning way – large data set     Audio    Transcript      End-to-end deep learning cannot be used for every problem since it needs a lot of labeled data. It is used mainly in audio transcripts, image captures, image synthesis, machine translation, steering in self-driving cars, etc. |
| Whether to use end-to-end deep learning    Before applying end-to-end deep learning, you need to ask yourself the following question: Do you have enough data to learn a function of the complexity needed to map x and y?  Pro:   * Let the data speak   + By having a pure machine learning approach, the neural network will learn from x to y. It will be able to find which statistics are in the data, rather than being forced to reflect human preconceptions.      * Less hand-designing of components needed - It simplifies the design work flow.   Cons:   * Large amount of labeled data   + It cannot be used for every problem as it needs a lot of labeled data.      * Excludes potentially useful hand-designed component   Data and any hand-design’s components or features are the 2 main sources of knowledge for a learning algorithm. If the data set is small than a hand-design system is a way to give manual knowledge into the algorithm. |