Deep Learning Specialization

3. Structuring Machine Learning Projects

- Orthogonalization

  - Independently adjusting one parameter at a time.

- Single no. Evaluation Metric

  - Precision: TP / TP + FP
  - Recall: TP / TP + FN “sensitivity”
  - F1 Score: 2 / (1/precision + 1/recall) “harmonic mean”

  - Optimizing vs Satisfying metrics
    - optimizing: max e.g: accuracy
    - satisfying: reach some threshold running time

- Some notes

  - choose train/test set from the same distribution; to reflect the data you expect to get in the future.
  - set your test set to be big enough to give confidence in the overall performance of your system.

  - In case evaluation metric is no longer preferred:

    Example: we want to give more weight to porn images in a classification problem.

    New Error Evaluation: $\frac{1}{\sum w^i} \sum w^i \mathbf{1}\{ y^i_{pred} \neq y^i \}$

    w^i : 1, if x^i is non porn. 10, otherwise.

  - Orthogonalization
    - defines one metric to evaluate classifiers. i.e. place target.
    - how to do well on this metric. i.e. aim to shoot.

- If you're model is doing well on your chosen metric, yet the dev/test set does not correspond to doing well in practice (in the application you're developing for example), then change your metric and/or dev/test set.

- Human-level Performance

    - Bayes Optimal Error: can not be exceeded (unless you're overfitting).

    -

Human	1%	7.5%
Train Set	8%	8%
Test Set	10%	10%
	focus on bias	focus on variance

    Avoidable Bias = Humans – Train Set, Variance = Train Set – Test Set

    - Sometimes performance surpass human-level.

    - How to tackle:
        - Avoidable Bias:
            - train bigger model
            - train longer / better optimization alg.
            - NN architecture / hyperparameter search
        - Variance:
            - add more data
            - regularization
            - NN architecture / hyperparameter search

- Error Analysis

1. get approximately 100 mislabelled dev set examples.
2. count up how many are labelled correctly.

	case #1	case #2
- Accuracy: 90% Error: 10%	5% of all labels were incorrectly labelled from 10% to 9.5%	50% from 10% to 5%

N.B. know which direction is the most promising.

- Evaluate multiple ideas in parallel.
  Goal: is to get better intuition how to proceed or in which direction.
  Example: in a spread sheet, where # causes of error stacked vertically and # of images stacked as rows.
    # causes could be blurry images, incorrectly labelled.

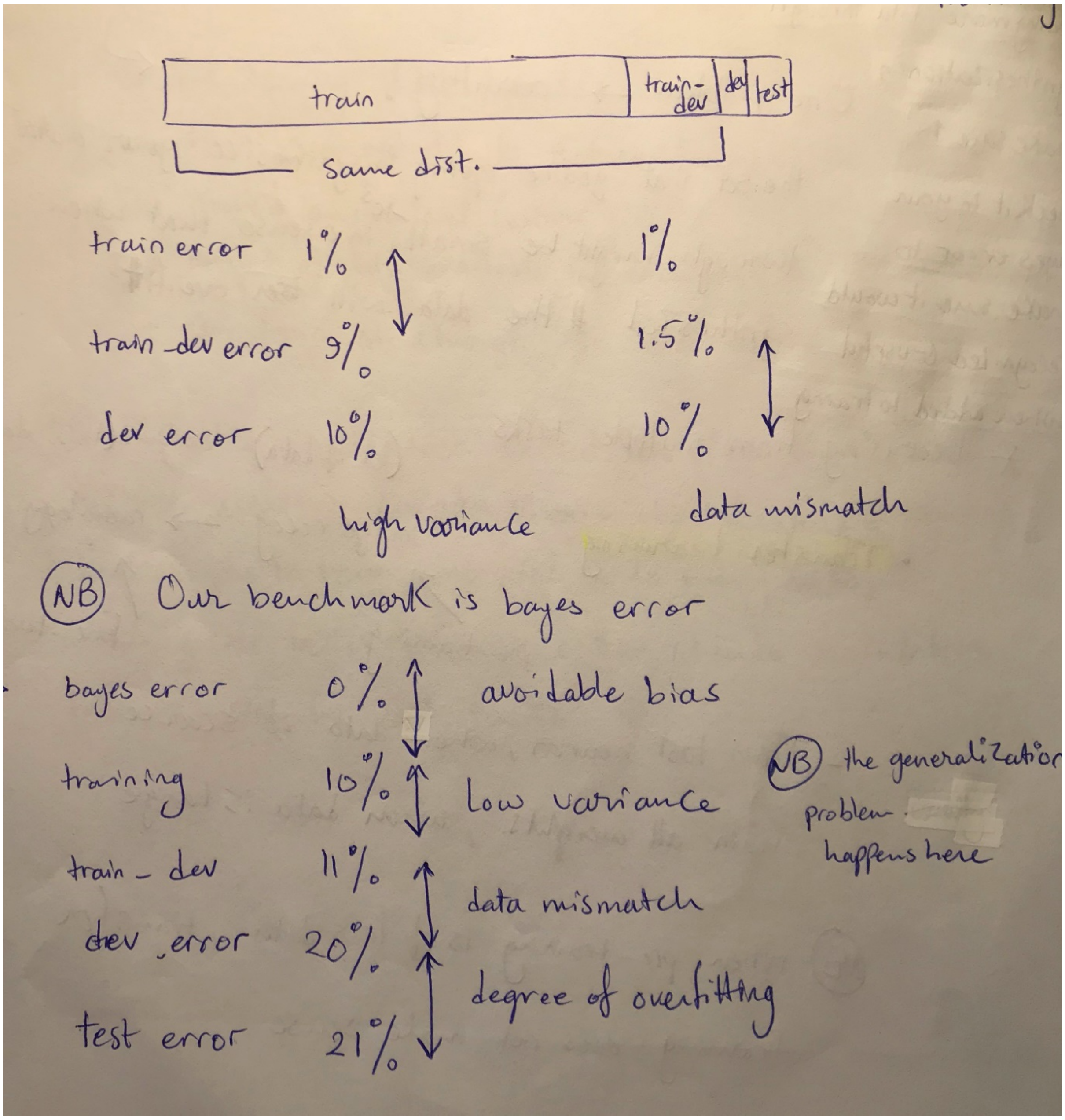
- Deep Learning algorithms are quite robust to random errors in training sets, however they are less robust to systematic errors in training sets.

- Correcting the incorrect in dev/test sets:
  - apply the same process to both dev & 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.
  - train & dev/test data may come from slightly different distribution.

- Build your initial system quickly:
  - set up dev/test set and metric
  - build up the system fast
  - use bias/variance analysis and error analysis; to prioritize your next steps.

- Mismatched training & dev/test sets

  - when data is limited, make sure that the available targeted data (i.e. data from mobile, if you're developing an application) is used in dev/test set. In contrast, you can train your model from other data (i.e. available online).
  - Training-dev set: same distribution as training set, but not used for training.
    train



train
train-dev
dev
test
same dist.

train error	1%	1%
train-dev error	9%	1.5%
dev error	10%	10%
	high variance	data mismatch

NB Our benchmark is bayes error

bayes error	0%	avoidable bias
training	10%	Low variance
train - dev	11%	data mismatch
dev error	20%	degree of overfitting
test error	21%	

NB the generalization problem happens here

- One thought to consider: sometimes the training examples given are genuinely harder than those in test set.
- Addressing data mismatch:
  - carryout manual error analysis to try to understand more training & dev/test set.
  - make training data more similar, or collect more data similar to dev/test set "Artificial Data Synthesis".
- One problem might still persist here is overfitting:
  the set that you're going to synthesize your data (e.g. magnitude of 10^7) through might be too small (e.g. 10^3), in a sense that when synthesized the data will overfit.

- Transfer Learning

  - Example: image recognition (pre-training) and radiology diagnosis (fine-tuning).
  - The pre-training step represent the abundance of data, while the fine-tuning is of less data.

    N.B. when pre-training is of less data, transfer learning makes no sense.

  - Transfer Learning makes sense:
    - tasks A & B have the same input X.
    - A has more data than B.
    - low-level features from A could be helpful in B.

- Multi-Task Learning

  - It enables you to train one NN to do many tasks, which yields better performance than when do in isolation; assigning one NN for each task.
  - Example: one image has multiple labels: car, pedestrian, stop sign, traffic lights. we call such labels: tasks.

  - Multi-tasks learning makes sense:
    - training on a set of tasks that could benefit from having shared lower-level features.
    - amount of data available for each task is quite similar.
    - can train a big enough NN to do well on all tasks.

    N.B. in practice, multi-task is used less often than transfer learning, with one exception: computer vision and object detection.

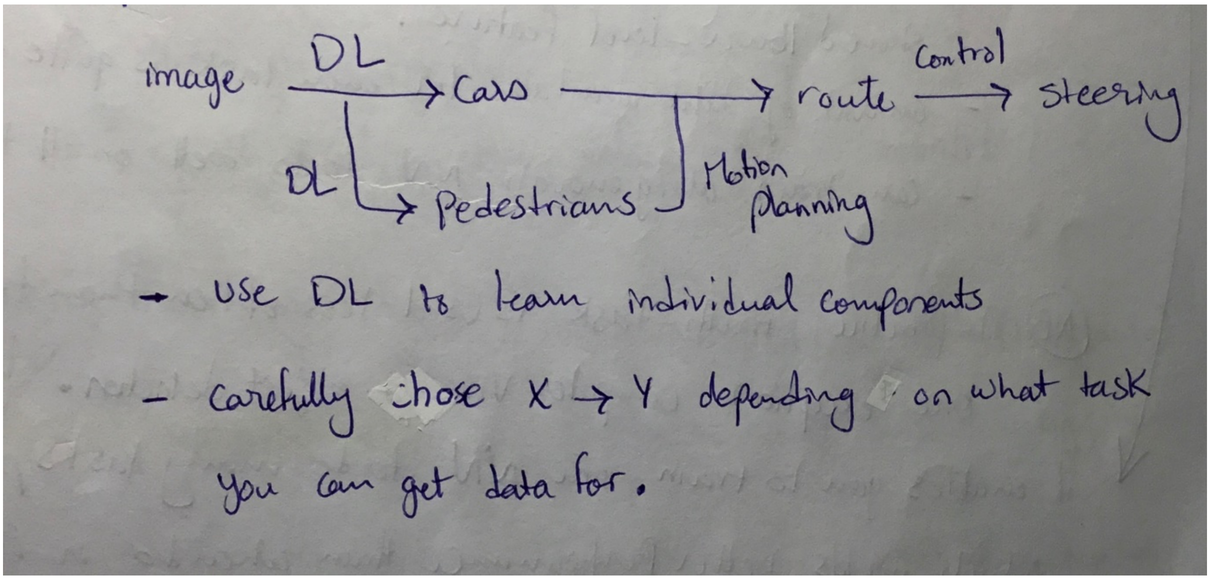
- End-To-End (E2E)

  - it usually works with very large datasets.
    Traditional Pipeline Approach in contrast deals with relatively small data.

  - Pros:
    - let the data speak.
    - less hand-designing of components needed.
  - Cons:
    - may need large amounts of data.
    - excludes potentially useful hand-designed components.

N.B. the amount of data is inversely proportional to hand-designing.

- Example:



→ use DL to learn individual components

– carefully chose X → Y depending on what task you can get data for.

- Batch Normalization

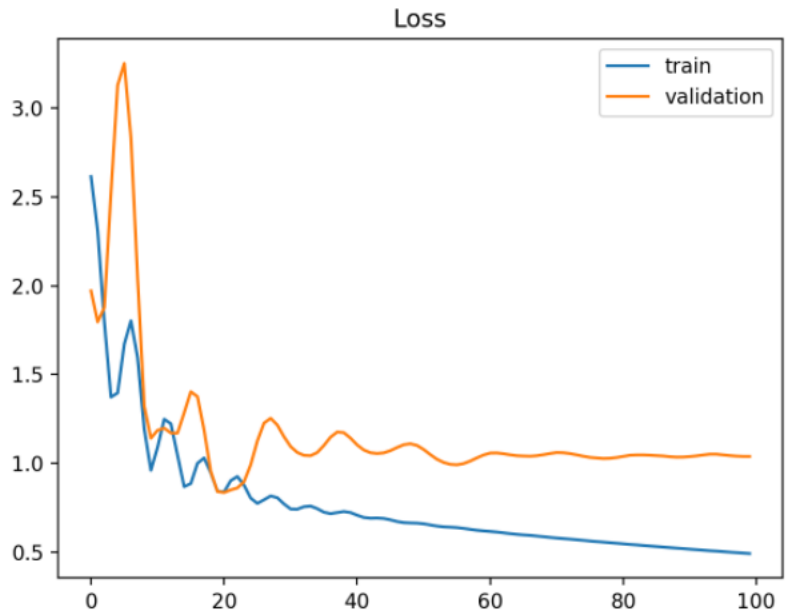
  - it has learnable parameters.
  - it uses 2 trainable parameters that allow the network to undo the normalization effect of this layer if needed.
  - it makes gradients more stable so that we can train deeper networks.
  - at test time, it uses a mean and variance computed on training samples to normalize data.

- General Notes

  - Making your network deeper by adding more parameterized layers will always slow down training and inference speed.
  - For a multi-task learning, if one example has missing entries $\begin{matrix}0\\?\\1\end{matrix}$, the algorithm will still be able to use it, cost can still be computed.
  - In case the algorithm did better one on distribution over the other, it could be because it was trained on that distribution or simply because the distribution might be easier. To get better sense, measure the human-level error separately on both distributions.
  - When adding synthesized data to your training, just make sure that it's useful enough! i.e. how? measure it against bayes.

- Unrepresentative Training Set

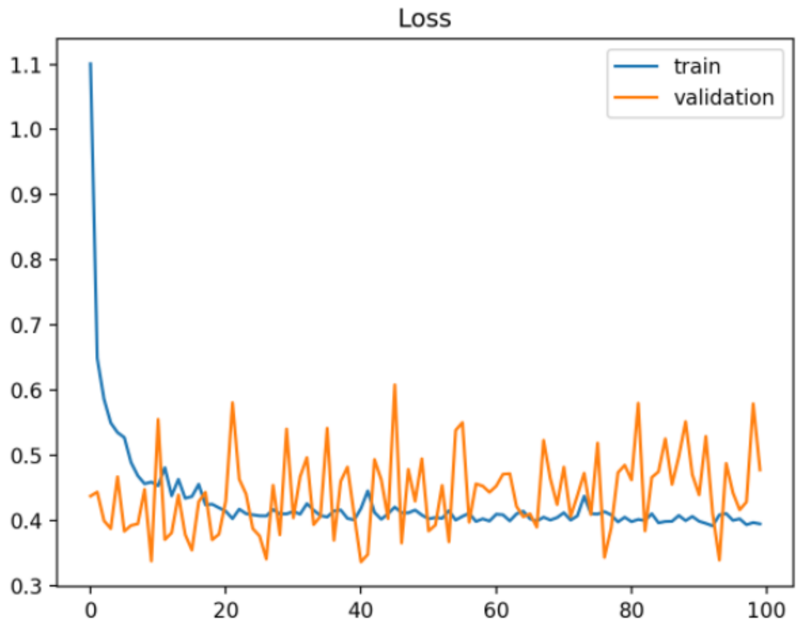
  - It means that the training dataset does not provide sufficient information to learn the problem, relative to the validation dataset used to evaluate it.
  - This may occur if the training dataset has too few examples as compared to the validation dataset.
  - This situation can be identified by a learning curve for training loss that shows improvement and similarly a learning curve for validation loss that shows improvement, but a large gap remains between both curves.



Example of Train and Validation Learning Curves Showing a Training Dataset That May Be too Small Relative to the Validation Dataset

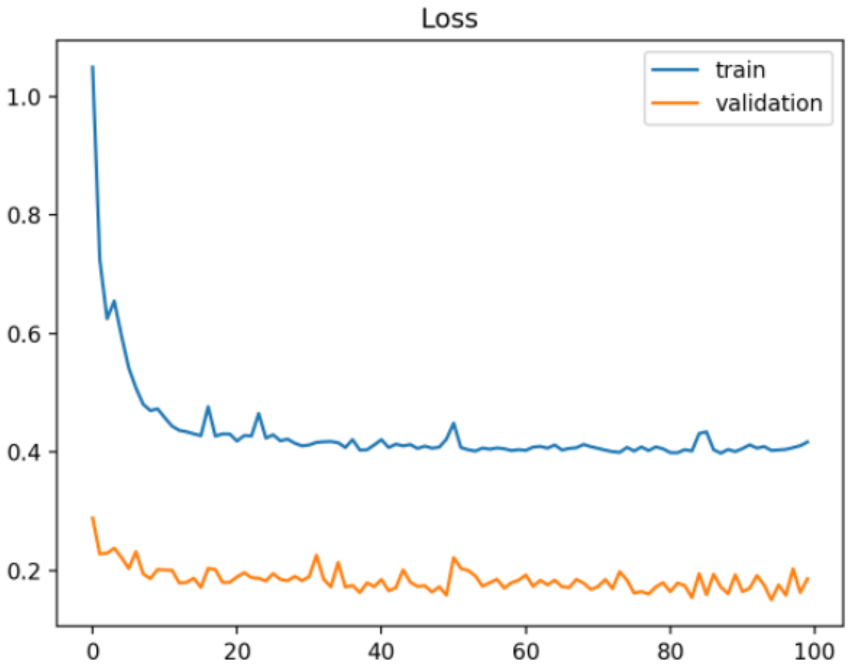
- Unrepresentative Validation Set

  - It means that the validation dataset does not provide sufficient information to evaluate the ability of the model to generalize.
  - This may occur if the validation dataset has too few examples as compared to the training dataset.
  - This case can be identified by a learning curve for training loss that looks like a good fit (or other fits) and a learning curve for validation loss that shows noisy movements around the training loss.



Example of Train and Validation Learning Curves Showing a Validation Dataset That May Be too Small Relative to the Training Dataset

- It may also be identified by a validation loss that is lower than the training loss. In this case, it indicates that the validation dataset may be easier for the model to predict than the training dataset.



Example of Train and Validation Learning Curves Showing a Validation Dataset That Is Easier to Predict Than the Training Dataset