Evaluation metric precision vs recall

Its better and faster to set a single number evaluation metric for your project before you start it.

Difference between precision and recall (in cat classification example):

Suppose we run the classifier on 10 images which are 5 cats and 5 non-cats.

Confusion matrix:

	Predicted cat	Predicted non-cat
Actual cat	3	2
Actual non-cat	1	4

- **Precision**: percentage of true cats in the recognized result: P = 3/4
- **Recall**: percentage of true recognition cat of the all cat predictions: R = 3/5
- **Accuracy**: 7/10

Note: Using a precision/recall for evaluation is good in a lot of cases, but separately they don't tell you which algothims is better.

Classifier	Precision	Recall
A	95%	90%
B	98%	85%

F1 score:

You can think of F1 score as an average of precision and recall $F_1 = \frac{2}{\frac{1}{P}+\frac{1}{R}}$.

Satisfying and Optimizing over some metric

multi number evaluation metric. Ex:

Classifier	F1	Running time
A	90%	80 ms
B	92%	95 ms
C	92%	1,500 ms

Satisfying both demands:

```
Maximize F1                     # optimizing metric
subject to running time < 100ms # satisficing metric
```

Train/dev/test distributions

- Dev and test sets have to come from the same distribution.
- Choose dev set and test set to reflect data you expect to get in the future and consider important to do well on.
- Setting up the dev set, as well as the validation metric is really defining what target you want to aim at.

Size of the dev and test sets

- examples ~ <100k 70% training/ 30% test or 60% training/20% dev/ 20% test.
- examples >1M 98% training/1% dev/1% test.

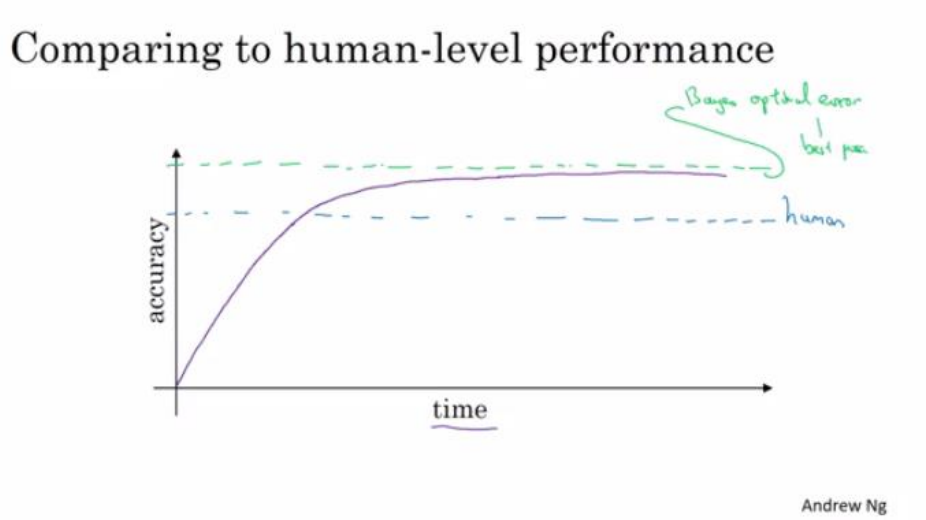
Human-level and Bayes error

- We compare to human-level performance because of two main reasons:
    i. Because of advances in deep learning, machine learning algorithms are suddenly working much better and so it has become much more feasible in a lot of application areas for machine learning

algorithms to actually become competitive with human-level performance.

ii. It turns out that the workflow of designing and building a machine learning system is much more efficient when you're trying to do something that humans can also do.

- After an algorithm reaches the human level performance the progress and accuracy slow
  down.



- You won't surpass an error that's called "Bayes optimal error".
- There isn't much error range between human-level error and Bayes optimal error.
- Humans are quite good at a lot of tasks. So as long as Machine learning is worse than humans, you can:
  - Get labeled data from humans.
  - Gain insight from manual error analysis: why did a person get it right?
  - Better analysis of bias/variance.

deep learning has surpassed human-level performance. Like:

- Online advertising.
- Product recommendation.
- Loan approval.

Improving your model performance

i. **avoidable bias(**training error- human level error) is large:
   - Train bigger model.
   - Train longer/better optimization algorithm (like Momentum, RMSprop, Adam).
   - Find better NN architecture/hyperparameters search.

ii. **Variance(**dev/test set error -training set error) is large:
   - Get more training data.
   - Regularization (L2, Dropout, data augumentation).
   - Find better NN architecture/hyperparameters search.

Carrying out error analysis

Error analysis approach:

- Get 100 mislabeled dev set examples at random.
- Count up how many are dogs.
- if 5 of 100 are dogs then training your classifier to do better on dogs will decrease your error up to 9.5% (called ceiling), which can be too little.
- if 50 of 100 are dogs then you could decrease your error up to 5%, which is reasonable and you should work on that.

Multi objective miss labeled example:

Image	Dog	Great Cats	blurry	Instagram filters	Comments
1	✓			✓	Pitbul
2	✓		✓	✓	
3					Rainy day at zoo
4		✓			
....					
% totals	**8%**	**43%**	**61%**	**12%**	

- In the last example you will decide to work on great cats or blurry images to improve your performance.

Cleaning up mislabeled data

- DL algorithms are quite robust to random errors in the training set but less robust to systematic errors. But it's OK to go and fix these labels if you can.
- If you want to check for mislabeled data in dev/test set, you should also try error analysis with the mislabeled column. Ex:

Image	Dog	Great Cats	blurry	Mislabeled	Comments
1	✓				
2	✓		✓		
3					
4		✓			
....					
% totals	**8%**	**43%**	**61%**	**6%**	

Then:

- If overall dev set error: 10%
  - Then errors due to incorrect data: 0.6%
  - Then errors due to other causes: 9.4%
- Then you should focus on the 9.4% error rather than the incorrect data.

Build your first system quickly, then iterate

- The steps you take to make your deep learning project:
  - Setup dev/test set and metric
  - Build initial system quickly
  - Use Bias/Variance analysis & Error analysis to prioritize next steps.

Training and testing on different distributions

There are some strategies to follow up when training set distribution differs from dev/test sets distribution.

1. (not recommended): shuffle all the data together and extract randomly training and dev/test sets.
    - Advantages: all the sets now come from the same distribution.
    - Disadvantages: the other (real world) distribution that was in the dev/test sets will occur less in the new dev/test sets and that might be not what you want to achieve.
2. take some of the dev/test set examples and add them to the training set.
    - Advantages: the distribution you care about is your target now.
    - Disadvantage: the distributions in training and dev/test sets are now different. But you will get a better performance over a long time.

Bias and Variance with mismatched data distributions

Bias and Variance analysis changes when training and Dev/test set is from the different distribution. you'll think that this is a variance problem, but because the distributions aren't the same you can't tell for sure. Because it could be that train set was easy to train on, but the dev set was more difficult.

To solve-

we create a new set called train-dev set as a random subset of the training set (so it has the same distribution) and we get:

situation 1:

- Human error: 0%
- Train error: 1%
- Train-dev error: 9%
- Dev error: 10%

Now we are sure that this is a **high variance** problem.

situation 2:

- Human error: 0%
- Train error: 1%
- Train-dev error: 1.5%
- Dev error: 10%

In this case we have something called ***Data mismatch*** problem.

Addressing data mismatch

There aren't completely systematic solutions to this, but there some things you could try.

1. Carry out manual error analysis to try to understand the difference between training and dev/test sets.
2. Make training data more similar, or collect more data similar to dev/test sets.

Make training data more similar:

one of the techniques you can use **Artificial data synthesis** that can help you make more training data.

- Combine some of your training data with something that can convert it to the dev/test set distribution.
  - Examples:
    a. Combine normal audio with car noise to get audio with car noise example.
    b. Generate cars using 3D graphics in a car classification example.
- Be cautious and bear in mind whether or not you might be accidentally simulating data only from a tiny subset of the space of all possible examples because your **NN might overfit** these generated data (like particular car noise or a particular design of 3D graphics cars).

Transfer learning

Apply the knowledge you took in a task A and apply it in another task B.

For example, you have trained a cat classifier with a lot of data, you can use the part of the trained NN it to solve x-ray classification problem.

How to do transfer learning?

To do transfer learning, delete the last layer of NN and it's weights and:

i. Option 1: small data set - keep all the weights fixed. Add a new last layer(-s) and initialize the new layer weights and feed the new data to the NN and learn the new weights.
ii. Option 2: enough data retrain all the weights

When transfer learning make sense?

- Task A and B have the same input X (e.g. image, audio).
- You have a lot of data for the task A you are transferring from and relatively less data for the task B your transferring to.
- Low level features from task A (like lines in image processing) could be helpful for learning task B.

Multi-task learning

Whereas in transfer learning, you have a sequential process where you learn from task A and then transfer that to task B. In multi-task learning, you start off simultaneously, trying to have one neural network do several things at the same time. And then each of these tasks helps hopefully all of the other tasks.

Example:

- You want to build an object recognition system that detects pedestrians, cars, stop signs, and traffic lights (image has multiple labels).

In the last example you could have trained 4 neural networks separately but if some of the earlier features in neural network can be shared between these different types of objects, then you find that training one neural network to do four things results in better performance than training 4 completely separate neural networks to do the four tasks separately.

When Multi-task learning makes sense?

ii. Training on a set of tasks that could benefit from having shared lower-level features.
iii. Amount of data you have for each task is quite similar.
iv. Can train a big enough network to do well on all the tasks.

- **If you can train a big enough NN, the performance of the multi-task learning compared to splitting the tasks is better.**

End-to-end deep learning

Some systems have multiple stages to implement. An end-to-end deep learning system implements all these stages with a single NN.

Example 1 Speech recognition system:

non-end-to-end system:

```
Audio ---> Features --> Phonemes --> Words --> Transcript    #
```

end-to-end system:

```
Audio ----------------------------------------> Transcript
```

End-to-end deep learning gives data more freedom, it might not use phonemes when training!

Pros of end-to-end deep learning:

- Let the data speak. By having a pure machine learning approach, your NN learning input from X to Y may be more able to capture whatever statistics are in the data, rather than being forced to reflect human preconceptions.
- Less hand-designing of components needed.

Cons of end-to-end deep learning:

- May need a large amount of data.
- Excludes potentially useful hand-design components (it helps more on the smaller dataset).

When Applying end-to-end deep learning is possible:

- Key question: Do you have sufficient data to learn a function of the **complexity** needed to map x to y?
- Use ML/DL to learn some individual components.