Course 3, week 2: ML Strategy (2)

Error Analysis:

Cleaning up Incorrectly Labeled Data

- What if we have data that is incorrectly classified/labeled? Is it worth while to go in and fix it?
- Deep learning algorithms are quite robust to random errors in the training set.
  - So long as the incorrectly labeled examples are not too far from random, then it's probably OK to leave the errors as they are and not spending too much time fixing them. The algorithm will be OK so long as the training set is big enough and the percentage of errors is not too high.
- Caveat: deep learning algorithms are less robust to systematic errors
  - If a labeler consistently classifies a white dog as a cat, that will cause some problems for the learning algorithm as it will learn that relation.
- What about incorrectly labeled examples in the dev or test sets?
  - Do some error analysis where we also count of the # of incorrectly labeled examples in the set.
    - Analyze the mislabeled examples by the classifier to understand which ones are truly mislabeled and which ones are incorrectly labeled

(By a human, therefore the label is not the correct one).

* If the incorrectly labeled data on the dev/test set impairs our ability to properly assess the performance of the model then maybe we should spend a bit of time fixing the labels.

  - To decide, we can look at the,
    - Overall dev set error: 10%
    - Error due to incorrect labels: 6% (incorrectly labeled percentage) • 10% = 0.6%
    - Error due to other causes: 9.4%

* As in this, it might not be worth while to look into the other sources of error from incorrect labels.

- When correcting incorrect dev/test set examples,
  - Apply the same process/correction to both the dev/test sets to make sure they continue coming from the same distribution.
  - Consider examining the examples that the algorithm got right as well.
    - There some examples that we might have gotten right that might also be worth fixing.
      - If we only fix the ones that we got wrong (By fixing the labels) that will lead to more biased estimate of the error.
  - The train and dev/test data may come from slightly different distributions.
    - This is OK as the algorithm is robust to the

training set.

Build your first model quickly, then iterate

- for almost any machine learning application, there could be 50 different directions we could go in that are reasonable. The challenge is how do we pick one to focus on.
  - The best methodology is to pick a first system quickly and then iterate.
    - Compare it to the ideal dev/test set and metrics selected.
    - Then use bias/variance and error analysis to decide where to focus next.

Error Analysis

Training and testing on different distributions

- Teams often combine data from different sources to enrich the train set. These additional data sets might not come from the same distribution as the dev/test sets.

Data from webpages

- Crawled from the web
- 200k examples

Data from mobile app

- Less from photos
  - Blurry
  - Less professional
  - Less well framed
- 10k examples
- We want the final system to do well on the

mobile app distribution

└ but the dataset is relatively small

- The dilemma is that we don't only want to use the 10K mobile pictures as it's too small, but the 200K crawled images don't come from the exact distribution that we're trying to predict.
- Option one: combine the datasets and randomly shuffle them into a train, dev, test set.
  - └ advantages: all our sets will now come from the same test distribution so the training process will be easier to manage
  - └ disadvantage (huge): the majority of the examples in the dev set will come from the web page distribution of images (rather than what we care about — the mobile app distribution).
    - └ only $\frac{10K}{210K}$ % of dev set data will come from the mobile app (for the dev set).
    - └ we would be spending too much time optimizing on the web set distribution of images in this case.
    - └ Hence, option one is not recommended
- Option two: have most of the training data be from the web crawler (200K web + 5K mobile) and then the dev and test sets should come from mobile app

exclusively (2.5% mobile calls).

- advantage: how we are aiming to the target accurately. The dev set data comes from the distribution that we care about and now we are optimizing our algorithm to that set of data
- disadvantage: train vs. dev/test distribution how differ
  - however, this way of setting will provide better performance over the long term.

Bias and variance with mismatched data distributions

- the way we analyze bias and variance changes when the test set distribution is different from the dev/test set.
- Cat classifier example
  - humans ≈ 0% error (Bayes optimal error is nearly 0%)
  - training error: 1%
  - dev error: 10%

If our sets came from the same distribution, we could say here that we have a large variance problem (our algorithm is not generalizing well from the training set). In the setting where our distributions come from different places, we can no longer draw this conclusion.

- this disparity might just be reflecting that the images in the dev set are harder to classify (they're of a lower resolution (or any other reason)).

- It's difficult to quantify how much of the additional error is from overfitting the training data and how much is caused by the difference in distribution.

- A way to trace out this difference is to create a new set (called the training-dev set) where the data has the same distribution as the train set, but hasn't been used for training.

- Example ①
  - Training error: 1%
  - Training-Dev error: 9%
  - Dev error: 10%

  ] We can conclude a variance problem (as the model has overfitted to the training data)

- Example ②
  - Training error: 1%
  - Training-Dev error: 1.5%
  - Dev error: 10%

  ] data mismatch problem.

* In data mismatch, we are learning to predict well on the training set distribution, while not doing so well in the dev set (which is composed of a slightly different distribution).

- Example ③
  - Human level: ~ 0%
  - Training error: 10%

  ] avoidable bias problem (underfitting).

  - Train-Dev error: 11%
  - Dev error: 12%

- Example ④ (two problems)

- human level: ~ 0%] avoidable bias
- training error: 10%
- train-dev error: 11%] data mismatch
- dev error: 20%

- General principles:
  - human level] avoidable bias
  - train error] variance (overfitting)
  - train-dev error] data mismatch
  - dev error] overfitting to the dev set (find a bigger dev set)
  - test error

addressing data mismatch

- If error analysis shows that we have a data mismatch problem, then what can we do?
- We can try to fix this by making the training data more similar to the dev set data or alternatively, we could collect more data that's similar to dev/test sets.

Artificial data synthesis

Learning from multiple tasks

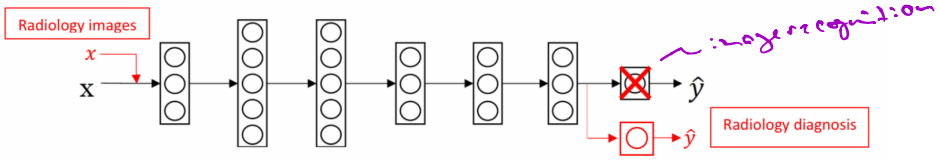
Transfer Learning

- one of the most powerful ideas from deep learning is that we can sometimes take knowledge that the network has learned

from one task and apply it to a separate task.

└ such as training a neural net to recognize cats and then using the knowledge for x-rays

- if we want to transfer a neural net from image recognition to radiology, we can do that by deleting the output layer of the network, and the weights and biases connecting to it, and then creating a new output layer with randomized weights and biases.



i. train the image recognition network as usual, training the usual parameters through forward passes and backprop.

ii. having trained the parameters, select the output layer (by initializing its weights)

iii. train on the desired data set to which we want to transfer the learned network (through radiology scans — for diagnosis).

└ if we have a small dataset (for radiology) we might want to train only the last layer (the output layer) (this is known as fine tuning)

└ if we have a lot of data we might want to re-train all the parameters of the network.

└ if we want to do this, fact is is known

of pre-training because we are using image recognition data pre-initialize the weights of the network.

* we are taking knowledge learned through image recognition and transferring it to radiology diagnosis.
  - this knowledge helps as we originally learn how to detect low level features, such as edges, lines, curves or positive objects (from a large image recognition dataset).
    - helping the diagnosis learn faster and/or with less data later on.

- Sometimes, instead of only creating a new output node, we can add a few extra layer at the end of our original network to solve a different kind of problem.
- When does transfer learning make sense?
  - when we have a lot of data for the data we are transferring from and relatively less data for the problem we are transferring to.
- transfer learning does not make sense in the cases where you have a few examples in the task you're transferring from and a relatively large # of examples in the task you're transferring to.
  - doing it will not hurt, but we will not gain any meaningful advantages from doing it in this way.

Multi-task Learning

- We start off simultaneously trying to have one neural network do several things at the same time and hope that the network can generalize to individual tasks (after being trained on different tasks)

- imagine we are trying to train a neural network that has an output $(4,1)$ for example (four labels in an image)
  - we have to define a loss function that can take into account these four outputs

$$\text{Loss} : \underset{(4,1)}{\hat{y}^{(i)}} = \frac{1}{m} \cdot \sum_{i=1}^{m} \sum_{j=1}^{4} \mathcal{L}\left(\hat{y}_j^{(i)}, y_j^{(i)}\right)$$

  - where $\mathcal{L}\left(\hat{y}_j^{(i)}, y_j^{(i)}\right) = -y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right) - (1-y_j^{(i)}) \log\left(1-\hat{y}_j^{(i)}\right)$

* the main difference between this and the original loss function is that we are now summing from $j=1 \ldots 4$ for the different outputs. Unlike soft max regression, which assigned a single label to a single example, this one example will have multiple labels (one of the classification vs. detection multiple objects in a image (autonomous vehicles)).
  - multiple objects can appear in the same image.

- By using this cost function, we are looking at a image and trying to solve four problems simultaneously
  - this could be replaced by four different neural

networks that can recognize one object in the image (the advantage of doing it in a multi-task learning manner is that the low-level features, in the early layers, are often shared by the multiple tasks we're trying to solve).

- By having more objects to recognize, we can end up having a better performance than if we were to train the networks separately.

- In multi-task learning, we can train the network even when some images have only a subset of the labels (the presence of a specific object in some examples of the dataset are not labelled at all). We can omit the object that aren't present from the sum of the loss function (tweaking the summation of onehots term from example to example when the presence of certain objects in an image is a question mark).

- When does multi-task learning make sense?
  - When we train on a set of tasks that could benefit from having shared lower-level features).
  - Usually: the amount of data we get for each task is similar.
  - Can train a big enough neural network to do well on all the tasks.

End to End Deep Learning

What is end-to-end deep learning?

- end-to-end deep learning encapsulates all the intermediate processing steps into a single neural network to solve a specific problem.
  - take for example traditional speech recognition
    - x (audio) → MFCC (signal processing (mel-frequency cepstrum)) → features → ML → phonemes (Cat = ka-ah-tu (basic units of words)) → words → transcript
  - in end to end deep learning, we skip all these intermediate steps and go directly from audio to the transcript.
    - end-to-end deep learning works in the cases where we have an abundance of data. for smaller datasets, the traditional methodology works better.

When to use end-to-end deep learning

- pros and cons of end-to-end deep learning
  - pros
    - let the data speak
      - the mapping from x → y will be figured out of the neural network. The model will reflect less human pre-conceptions.
    - less hand-designing of components needed
      - simplifying design workflow

- Cons
  - may need a large amount of data to learn the allowable mapping.
  - excludes potentially useful hand designed components
    - there's an inability to inject manual knowledge into the model in end-to-end deep learning.