

Deep learning

Dr. Aissa Boulmerka
a.boulmerka@centre-univ-mila.dz

2023-2024

CHAPTER 7

MACHINE LEARNING STRATEGY

Carrying out error analysis

- Error analysis - process of manually examining mistakes that your algorithm is making. It can give you insights into what to do next. E.g.:
 - In the cat classification example, if you have 10% error on your dev set and you want to decrease the error.
 - You discovered that some of the mislabeled data are dog pictures that look like cats. Should you try to make your cat classifier do better on dogs (this could take some weeks)?
 - 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.

Carrying out error analysis

- Based on the last example, error analysis helps you to analyze the error before taking an action that could take a lot of time with no need.
- Sometimes, you can evaluate multiple error analysis ideas in parallel and choose the best idea. Create a spreadsheet to do that and decide.
- In the last example you will decide to work on great cats or blurry images to improve your performance.
- This quick counting procedure, which you can often do in, at most, small numbers of hours can really help you make much better prioritization decisions, and understand how promising different approaches are to work on.

Evaluate multiple ideas in parallel

- Ideas for cat detection:
 - Fix pictures of dogs being recognized as cats
 - Fix great cats (lions, panthers, etc..) being misrecognized
 - Improve performance on blurry images

Image	Dog	Great Cats	Blurry	Instagram filters	Comments
1	✓			✓	Pitbull
2	✓		✓	✓	
3					Rainy day at zoo
4		✓			
....					
% of total	8%	43%	61%	12%	

Cleaning up incorrectly labeled data

- DL algorithms are quite robust to random errors in the training set but less robust to systematic errors. But you can 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	Incorrectly Labeled	Comments
1	✓				
2	✓		✓	✓	Labeler missed cat in the background
3					
4		✓		✓	Drawing of a cat: not a real cat
% of total	8%	43%	61%	6%	

Cleaning up incorrectly labeled data

Image	Dog	Great Cats	Blurry	Incorrectly Labeled	Comments
1	✓				
2	✓		✓	✓	Labeler missed cat in the background
3					
4		✓		✓	Drawing of a cat: not a real cat
% of total	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.

Correcting incorrect dev/test set examples

- Consider these guidelines while correcting the dev/test mislabeled examples:
 - Apply the 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 done if you reached a good accuracy)
 - Train and (dev/test) data may now come from a slightly different distributions.
 - It's very important to have dev and test sets to come from the same distribution. But it could be OK for a train set to come from slightly other distribution.

Build your first system quickly, then iterate

- Noisy background
 - Café noise
 - Car noise
- Accented speech
- Far from microphone
- Young children's speech
- Stuttering
- ...

Guideline:

**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

- A lot of teams are working with deep learning applications that have training sets that are different from the dev/test sets due to the hunger of deep learning to data.
- There are some strategies to follow up when training set distribution differs from dev/test sets distribution.
 - **Option one (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.
 - **Option two:** 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.

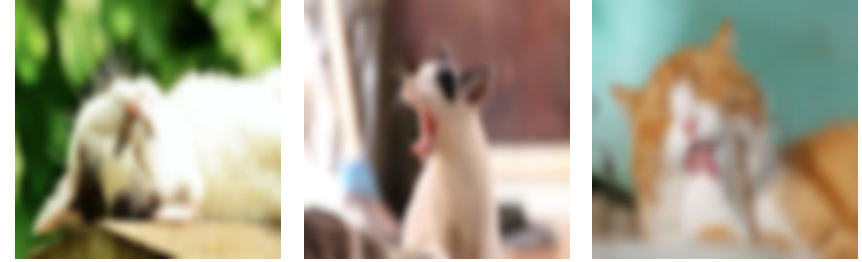
Cat app example

Data from webpages



≈ 200 000

Data from mobile app

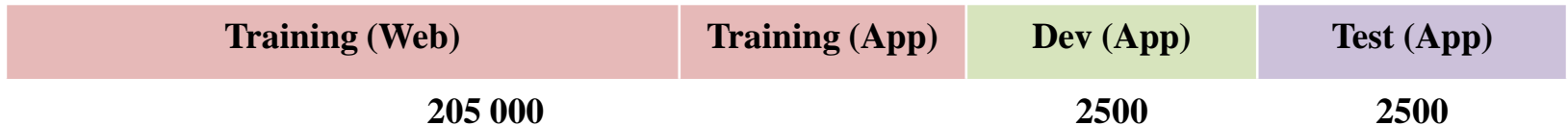


≈ 10 000

Option 1:



Option 2:



Speech recognition example



Training

Purchased data

Smart speaker control

Voice keyboard

... \approx 500 000

Dev/test

Speech activated
rearview mirror

\approx 20 000

Bias and Variance with mismatched data distributions

- Bias and Variance analysis changes when training and dev/test set is from the different distribution.
- **Example:** the cat classification example. Suppose you've worked in the example and reached this
 - **Human error: 0%**
 - **Train error: 1%**
 - **Dev error: 10%**
- In this example, 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**.

Bias and Variance with mismatched data distributions

- To solve this issue 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:
 - **Human error: 0%**
 - **Train error: 1%**
 - **Train-dev error: 9%**
 - **Dev error: 10%**
- Now we are sure that this is a **high variance problem**.

Bias and Variance with mismatched data distributions

- Suppose we have a different situation:
 - **Human error: 0%**
 - **Train error: 1%**
 - **Train-dev error: 1.5%**
 - **Dev error: 10%**
- In this case we have something called **Data mismatch problem**.

Bias and Variance with mismatched data distributions

Human error	0%	0%	0%	0%
Train set error	1%	1%	10%	10%
Train-dev error	9%	1.5%	11%	11%
Dev set error	10%	10%	12%	20%
	High variance	Data mismatch	Avoidable bias	Avoidable bias + Data mismatch

Bias and Variance conclusions

i. Human-level error (proxy for Bayes error)

ii. Train error

- Calculate **avoidable bias = training error - human level error**
- If the difference is big then its **Avoidable bias problem** then you should use a strategy for high bias.

iii. Train-dev error

- Calculate **variance = training-dev error - training error**
- If the difference is big then its high **variance problem** then you should use a strategy for solving it.

iv. Dev error

- Calculate **data mismatch = dev error - train_dev error**
- If difference is much bigger then its **Data mismatch problem.**

v. Test error

- Calculate degree of **overfitting to dev set = test error - dev error**
- If the difference is big (positive) then maybe you need to find a bigger dev set.
- Unfortunately, there aren't many systematic ways to deal with data mismatch. There are some things to try about this in the next section.

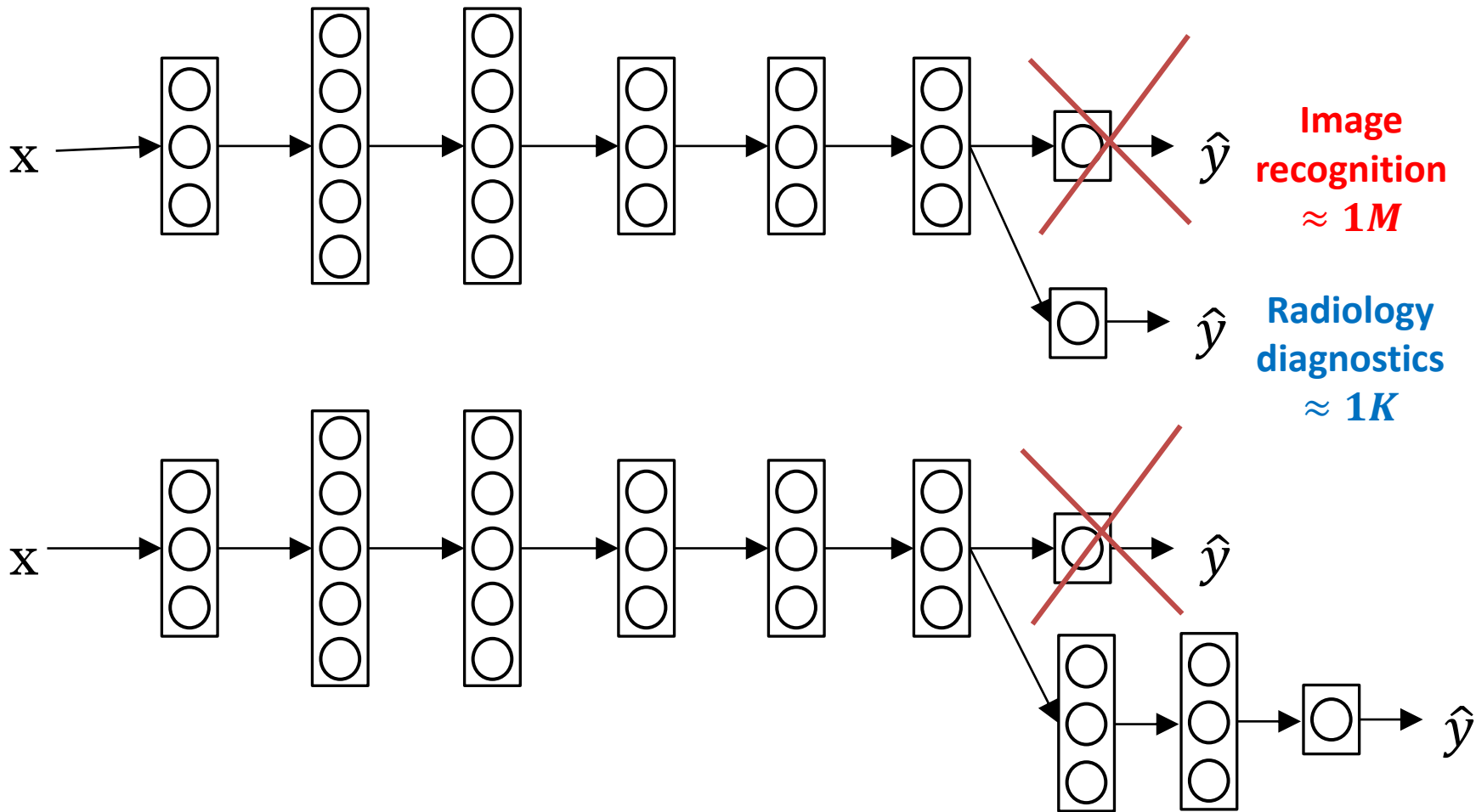
Addressing data mismatch

- There aren't completely systematic solutions to this, but there are 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.**

Artificial data synthesis

- If your goal is to make the training data more similar to your dev set 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 (video games).
- 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



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.
- To do transfer learning, delete the last layer of NN and it's weights and:
 - i. **Option 1:** if you have a small data set - keep all the other weights as a fixed weights.
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:** if you have enough data you can retrain all the weights.

Transfer learning

- Option 1 and 2 are called **fine-tuning** and training on task A called **pretraining**.
- 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 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).
 - Then Y shape will be **(4,m)** because we have **4 classes** and each one is a binary one.
 - Then

$$Cost = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^4 \mathcal{L}(\hat{y}_j^{(i)}, y_j^{(i)})$$

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

Multi-task learning

- 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.

- Multi-task learning will also work if Y isn't complete for some labels. For example:

$Y = [1 \ ? \ 1 \ \dots]$

$[0 \ 0 \ 1 \ \dots]$

$[? \ 1 \ ? \ \dots]$

- And in this case it will do good with the missing data, just the loss function will be different.

Multi-task learning

- Multi-task learning makes sense:
 - i. Training on a set of tasks that could benefit from having shared lower-level features.
 - ii. Usually, amount of data you have for each task is quite similar.
 - iii. 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.
- Today transfer learning is used more often than multi-task learning.

What is 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:**

Audio → Features → Phonemes → Words → Transcript : **non-end-to-end system**


Audio  Transcript : **end-to-end deep learning system**

- End-to-end deep learning gives data more freedom, it might not use phonemes when training!
- To build the end-to-end deep learning system that works well, we need a big dataset (more data than in non end-to end system). If we have a small dataset the ordinary implementation could work just fine.

What is end-to-end deep learning?


Example 2:

▪ **Face recognition system:**

- Image  Face recognition : **end-to-end deep learning system**
- Image → Face detection → Face recognition : **deep learning system - best approach.**
- In practice, the best approach is the second one for now.
- In the second implementation, it's a two steps approach where both parts are implemented using deep learning.
- Its working well because it's harder to get a lot of pictures with people in front of the camera than getting faces of people and compare them.
- In the second implementation at the last step, the NN takes two faces as an input and outputs if the two faces are the same person or not.

What is end-to-end deep learning?

Example 3:

- Machine translation system:
- English → Text analysis → ... → French : **non-end-to-end system**
- English  French : **end-to-end deep learning system - best approach**
- Here the **end-to-end deep learning system** works better because we have enough data to build it.

What is end-to-end deep learning?

Example 4:

Estimating child's age from the x-ray picture of a hand:

Image → Bones → Age : **non-end-to-end system - best approach for now**

Image \longrightarrow Age : **end-to-end system**

In this example **non-end-to-end system** works better because we don't have enough data to train end-to-end system.

Whether to use end-to-end deep learning

- **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).
- **Applying end-to-end deep learning:**
 - **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.
 - When applying supervised learning you should carefully choose what types of X to Y mappings you want to learn depending on what task you can get data for.

References

- Andrew Ng. Deep learning. Coursera.
- Geoffrey Hinton. Neural Networks for Machine Learning.
- Kevin P. Murphy. Probabilistic Machine Learning An Introduction. MIT Press, 2022.
- MIT Deep Learning 6.S191 (<http://introtodeeplearning.com/>)