Course-3

Structuring Machine Learning Projects

⊙ WEEK 1 :

A strategy is needed to quickly figure out which of all the ideas are worth pursuing and which ones can be safely discarded to improve model accuracy.
So, we will study strategies to analyze a ML problem that will help point in the dir^n of the most promising things to try.

Orthogonalization - when effective ML engineers know very well what hyperparameter to tune to get a certain effect.
Orthogonal controls that are ideally aligned with the things you actually want to control, it makes it much easier to tune the hyperparameters we want to tune
· Chain of assumptions in ML :
- Fit training set well on cost function
- Fit dev set well on cost function
- Fit test set well on cost function
- Performs well in real world.

Now, if the algo is not fitting the training set well on the cost f^n, we want just one attempt to make sure that the algo is tuned to make it fit well on the training set. Those attempts could be using a bigger network or using better optimize algo.
If algo doesn't fit on dev set then the attempts could be regularization or getting bigger training set to generalize more on the dev set.
If algo doesn't fit test set then the attempt could be having a bigger dev set bcoz it might have happened due to overfitting.
If algo satisfies all but is not delivering in real world then the attempt would be probably to either change dev set or change cost f^n.
↓
dev set distribu^n might not be set correctly.

Note - Early stopping not only fit the training sets less well but it also is often done to improve dev set performance. So this attempt of tuning is less orthogonalized as it affects not just one but two things. It is not bad to use early stopping but when other orthogonal controls are available, it makes tuning of network much easier.

So, in ML it is nice if we can look at our system and say oh, this piece of it is wrong and then have exactly one hyperparameter or one attempt or a specific set of hyperparameters that helps to just solve that problem that is limiting the problem of ML system.

Q How to diagonose the bottleneck of system performance as well as identify the specific set of attempts to tune the system to improve that aspect of its performance?

Ans

* Single number evaluation metric : to quickly tell us if the new thing we just tried is working better or worse than the last idea. So, always set up a single number evaluation metric to evaluate the model.

Q. What is the problem with using a precision recall metric?

Ans There is a tradeoff b/w precision and recall and we want both of them to be good.

e.g.

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

, Here, how to decide which is the better classifier, A or B?

So, rather than using two values to pick a classifier, define a new evaluation metric called F_1 score that combines both precision and recall.

$\downarrow$ (Harmonic mean of Precision & recall)

So, having a well defined dev set which is how we are measuring our precision and recall, plus a single number evaluation metric (single row number) allows us to quickly tell which one is the better classifier.

Q How to set up optimizing as well as satisfying metrics?

Ans Its not easy to put up all the things we care into a single row number evaluation metrics

e.g. if we want an algo to have maximum accuracy and a running time < say 100 ms then, accuracy is the optimizing metric and running time is the satisficing metric. So, this is a reasonable way to put together accuracy and running time.

Note So if we have n metrics, we can keep 1 to be optimizing and the other n-1 to be satisficing i.e as long as they satisfy a particular threshold we don't care how much better it is in that threshold.

e.g. wake up devices (devices which wake up and get ready for the user on listening to some phrases like OK Google, Hey Siri, etc) need to have high accuracy like what is the likelihood that device will wake up on listening to these words. Also false +ve is a concern i.e how many times it randomly wakes up. So maximize accuracy subject to the condiⁿ that you have at most 1 false +ve every 24 hrs i.e device wakes up only once a day if noone is talking to it.

This split into satisficing and optimizing helps pick a classifier while keeping in mind all the different metrics.

* Dividing train, dev and test set decides the speed with which the team ML applicaⁿ is built.

1) Dev and test set should come from the same distribution. bcoz if

they lie in different distribution, what the model learnt keeping in mind the target to be the dev set will fail when the test set is tested bcoz the target has now been moved. So, choose a dev set and test set to reflect data you expect to get in the future and consider important to do well on.
The choice of training set decides how well we can actually hit the target.

Note If dev set is large enough that we don't think we will overfit, then it's not totally unreasonable to just have a train dev set and not a test set.

Q. When to change dev/test sets and metrics?
Ans Sometimes partway through a project we might realise that we set the target at the wrong place and need to shift and doing so is perfectly OK.
e.g. Metric : classification error
Algo A : 3% error (allows penetration of pornographic images)
Algo B : 5% error (no pornographic images penetrate)

for cat classification.

Now, A does better on evaluation metric + Dev set but the company and users prefer B because it doesn't penetrate pornographic content.
So, when the evaluation metric is no longer correctly rank ordering preferences between algorithms like in the above case where according to metric A is better but is actually not, then perhaps there is a need to change the evaluation metric or the dev-test set.

$$\text{Error} : \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}}$$ is an indicator that $y^{(i)}_{pred} \neq y^{(i)}$

// counts the no. of misclassified examples.

predicted value — actual value

We need to ensure that those misclassified examples are not the pornographic images bcoz that might upset the user.

So, $$\boxed{\text{changed metric} = \frac{1}{(m_{dev})'} \sum_{i=1}^{m_{dev}} \omega^{(i)} = \frac{1}{\Sigma \omega^{(i)}} \sum_{i=1}^{m_{dev}} \omega^{(i)}}$$

$$\text{where } \omega^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-porn} \\ 10 & \text{if } x^{(i)} \text{ is porn} \end{cases}$$

So, now if the classifier will make a mistake with porn image, error will be 10 times and so the overall error of the algo would go down if it repeatedly misclassifies porn examples as cat images, hence putting the algo much lower on the preference list.

23/8/18

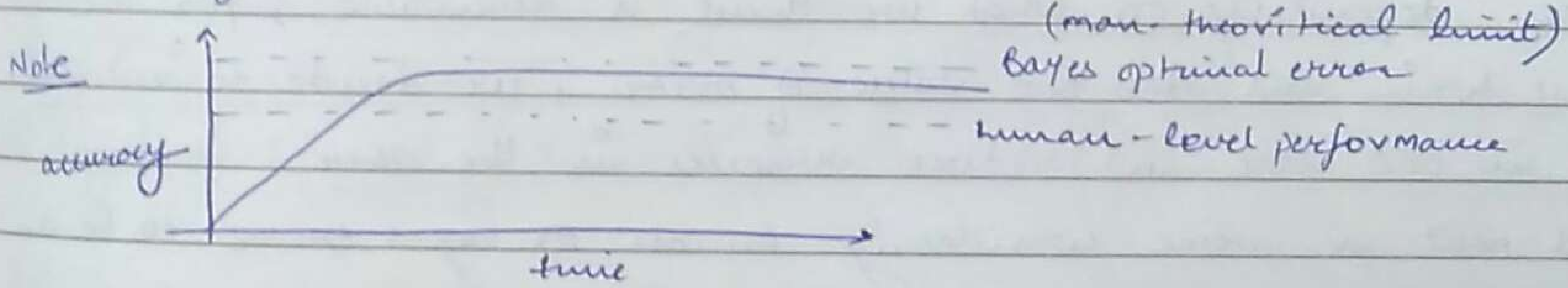
Orthogonalize of cat pictures : anti-porn

① define an evaluation metric (place the target)

② how to do well on this metric. (how to aim the target accurately)

② nd step can be done by optimizing the cost f^N J and incorporating the weights ω in it.

Note



Bayes error is the very best theoritical f^N for mapping from x to y, and this limit can never be surpassed by any model at any time. Progress is quite fast until we surpass human level performance but after that it slows down bcoz:

i) Human level error and bayes error are not too far from each other.

ii) Tools are easier to use when accuracy is below human level while its hard to use these tools to improve accuracy after reaching human level performance.

Manual error analysis i.e why did a human get this right? can help improve model performance. Better analysis of bias/ variance. All these tactics are easy to apply when

Note Human error being used as a proxy for the Bayesian error.

algo is doing bad than humans, but is hard to apply when it outperforms human

e.g. If in cat classifica example, human error = 1%.
train error = 8%
dev error = 10%
, then may be we want the algo to do better on the training set bcoz the figures show that our algo is not even fitting the training set well. So, we need to focus on reducing bias. Suppose in some other case

human error = 7.5%
train error = 8%.
dev error = 10%.

$\boxed{\text{avoidable bias}}$

, then though the train and dev error is same as in previous case, our performance is not that bad bcoz it is only a little worse than human. Maybe in this case our focus is on reducing the variance i.e gap b/w the train and dev set error.

In computer vision human error is very close to bayes error. Hence, depending on what we think is achievable; for the same train and dev set value of error, we decide to reduce bias in one case and reduce variance in the other.

Till now we were considering human or bayes error to be 0.

We can't do better than Bayes error unless we are overfitting.
In e.g.1 it is easier to reduce the avoidable bias while in e.g.2 it is easier to reduce the variance.

e.g. what if human error = 0.5%
train error = 0.7% ↕ 0.2%
dev error = 0.8% ↕ 0.1%

. In this case both avoidable bias and variance have comparable values. So which one would you address?

Ans Now, we need to try to do better on our training set. We know making progress in a ML problem gets harder as we approach human level performance through our model.

Note ⇒ A learning algo's performance can be better than human level but never than bayes level bcoz that is the maximum attainable.

Date	
Page	

Note Having an estimate of human level performance gives us an estimate of bayes error.

Bsup e.g.

Human ~~Bayesian~~ error : 0.5%
Train " : 0.3%] overfitted by 0.2%. {Bsup}

Dev error : 0.4%.

Here 0.5% is highest known human error i.e the best work humans could do but train error is coming out to be 0.3% which means Bayesian error could be 0.1, 0.2 or 0.3, that part of info is unavailable. So, the model is actually not overfitting but reaching the bayes level. Also, in such cases it is not explicit whether the focus should be on reducing bias or variance which in turn slows down the efficiency of progress. Also, now the humans can't be trusted in this case to know what else can be done with the algo to improve it further as it has already surpassed human level error. e.g. cases are :

① Online advertising ② Product recommendations
③ Logistics (predicting transit time) ④ Loan approvals

All of the above problems have learnt from structured data and are not natural perception problems (e.g computer vision) or speech or language processing task where humans excel unlike in the structured data problems where algos of ML have surpassed human level performance as these algos have looked at far more data than any human could. So much data automatically allows the algo to draw patterns & predict much better.

Still there are fields like speech and image recogniⁿ, medical diagnosis (ECG, cancer) and minute radio readings etc where machines have surpassed single human performance after

Date
Page

great efforts.

- The two fundamental assumptions of supervised learning are:
  i) Training set can be fitted pretty well, in other words low avoidable bias is achieved roughly saying.
  ii) The training set performance generalizes pretty well to the dev/test set, in other words saying that variance is not too bad.

Note Avoidable bias tells how much better we need to do on our train set and the difference b/w training error and dev error indicate the level of existing variance problem, i.e how much effort is needed in making the performance generalize from training to the dev set.

To solve avoidable bias
- Train bigger model
- Train longer with better optimize algos
- NN architecture / hyperparameter search to be reframed. (e.g no. of layers, hidden units, etc)

To solve variance
- Get more data (to generalize better)
- Regularize (L_2, dropout or data augmentation
- NN architecture / hyperparameter search

Q. You train a system and errors are: Train error : 4%
Dev error : 4.5%

Now, should the 4% training error be brought down by using a bigger network to train?

Ans Insufficient info to decide anyway.

If human error is taken to be = 0%, then the mentioned step should be taken as we have high bias in this case provided human error = 0%.

Q. What if my model has high accuracy but also many false negatives?

Ans Rethink the appropriate metric for the task completed by the model and use this new metric to drive all further development.

⊙ Week 2 : 24/8/18

→ Error Analysis : The process of manually examining the algorithm to find out why it is still unable to reach human level performance is called so.

e.g. If we have made a cat classifier with 10% dev error (error red bcoz it is misclassifying some dogs as cats), then maybe we would like to train with more dog pictures or maybe design features specific to dogs or something in order to make the cat classifier do better on dogs.

Q Should we start off with a dog project to solve the error? Would it be worth the effort?

Ans Instead of starting off with it just to find out in the end that it wasn't much helpful, do error analysis beforehand :

i) Get 100 mislabeled dev set examples.

ii) Count up how many of these 100 are actually dog images.

Now, suppose there are only 5/100 images which are of dog but are classified as cat. Now if we work on the dog problem, we will be only able to correct 5 more pictures & thus reduce the error just by 5% i.e now its 9.5 instead of 10%. So, it isn't worth the time.

But suppose if out of 100, 50 of them were dog images, then it could be much more fruitful to spend time on dog

Note It also depends that what data is easy to collect and add in training data for better learning & performance. Maybe that 43% error is due to cats but collecting more cat images is difficult, but collecting others is easy, so ~~their~~ approach is changed.

Date
Page

problem as the error would now go down to 5% from being 10% which is worth the time spent.

Sometimes, we can also evaluate multiple ideas in ll during error analysis. E.g. Ideas for improving cat detection are:

i) Fix pictures of dogs being recognized as cats.
ii) Fix great cats (lions, panthers, etc..) being misrecognised.
iii) Improve performance on blurry images.

Misrecognized dev set eg.s	Dog	Great cats (tiger, etc)	Blurring	Insta (bcz of filters)	Comments.
1	✓			✓	Pitbull.
2			✓		
3				✓	
4		✓	✓		Rainy day
:		:			
:	:	:			
% of total	8%	43%	61%	12%	

So, we have found what % of misclassified data is bcoz of dogs, great cats, blurr images, etc. This e.g. shows that it would be good to work on blurry images 61% error lies there. Hence, this method helps us prioritize the approach we need to take to work on the error. This method also helps us know the various new reasons (e.g. Insta in this case) responsible for messing up the classifier.

27/3/18

Q What to do if our dataset has some mislabeled examples?

Ans

Note DL algos are quite robust to random errors in the training set. So, as long as the errors are random, there is not much need of spending time in the error fixation. The algos work fine even if there are some mislabelled data provided its (error) random and few, bcoz

DL algos are less robust to systematic errors.

In the table made earlier, an extra column (incorrectly labelled) can be added to know what % of data is misclassified bcoz of being initially mislabled in the dev set.

So, check what %age of dev set is misclassified bcoz of being mislabelled & see if its worth spending time on it to improve the overall accuracy.

eg.		Case I	Case II
Overall dev set error	:	10%	2%
error due to mislabelling	:	6% of 10%	30% of 2%
error due to other reasons	:	94% of 10%	70% of 2%

In case I, 0.6% error out of 10% is due to mislabelling which is very less while in case II, 0.6% error out of 2% is due to mislabelling which is quite significant and needs time and attention.

Note: So, this analysis is very important in chosing the right classifier. Directly the accuracy value might not give the correct measure of classifier accuracy. Digging deep into the 'reasons for that much accuracy' actually tells us which one is the better classifier.

Q. Points to remember while correcting incorrect dev/test examples.

Ans i) Apply same process to both dev and test to make sure they continue to come from the same distribution.

ii) Consider examining the examples our algo got right as well as the ones it got wrong. Bcoz it is possible that it got some examples right just by chance & not fixing this might lead to a bias problem. This step is a bit hard when accuracy are reasonable (eg. 98%) bcoz it is easy to examine the 2% of wrong data than to examine the 98% right data. Hence, this step is also not very often used but can be considered.

iii) Train and dev/test data may come from slightly different distributions.

Note Manual insights and a bit of hand engineering can actually help a lot in prioritizing where to go next.

28/5/18

If you are building some new ML application, just build something quick and dirty and then use that in prioritizing how to improve the system. Iterate the first simple model to get a right model instead of overthinking and making a complex model in the first attempt. Use the simple model made to do bias variance analysis and error analysis and decide in which direction to go so as to improve the model.

Note DL algos have great hunger for training data and many of the people are training their algos with a lot of data resulting in test/dev data having different distribu^n and train data coming from a different distribution. Some best practices when dealing with such data:

eg. Suppose we want to recognize mobile clicked images as cats or not. So, there are two ways of collecting the train, test and dev data.

① Collect web images and mobile clicked images and shuffle them well to have train and dev/test data coming from same distribu^n.

Disadv: Dev/test contain web images despite them not being the target images. Most of the models energy is wasted in optimizing for web image results and only a small fraction focusses on the mobile images (which actually is our target)

(2) Collect images from web and mobile, keep the train data to be a mix of both but the dev/test data should contain only the mobile images.

Adv: Target images (dev/test) are only the mobile picture & so the model optimizes according to them only.

Disadv: Train and dev/test come from different distribution but still this condition is better than (1). This kind of split helps in long term.

Q. Should you always use all the data you have to train and test your ML model?

Ans Evaluation of bias and variance changes when the dev/test and train data come from different distributions.

e.g. cat classification

suppose humans ≈ Bayes error ≈ 0%.

training error = 1%

Dev error = 10%.

In this case, if train and dev data came from same distribution we could have concluded that we have large variance problem but now when they belong to diff. distribution, this is not a safe conclusion. May be the model is doing fine on dev set, the error may be bcoz of the fact that train data has high resolution images and so easy to learn while dev set has low quality images and so prone to error and difficult classification.

In this case going from train to dev, 2 things happened:

i) algo saw data in train set but not in dev set.

ii) algo didn't perform well and caused error.

Now, it's difficult to identify how much error is bcoz of what reason.

So, a piece of data called <u>training-dev set</u> is taken having same distribu^n as training set but not used for training. It is obtained by randomly shuffling the train set. So just as test and dev set are from the same distribution, similarly train & training-dev are from the same distribu^n.

Imp

<u>Let's say</u>: training error = 1%.
training dev error = 9%.
testset error = 10%. } 1%.

This shows that the error due to being from different distribu^n is just 1%. So, in this example we really have a variance problem.

<u>e.g.2</u>
training error = 1%.
train-dev error = 1.5%.
dev error = 10%.
} this example has a pretty low variance problem but a data mismatch problem.

<u>e.g.3</u>
bayes error = 0%.
train error = 10%. } bias problem
training-dev error = 11%.
dev error = 12%.

<u>e.g.4</u>
train error = 10% ⟶ high bias
training-dev error = 11% ⟶ low variance
dev error = 20%. } ⟶ high data mismatch.

→ <u>Bias/variance on mismatched training & dev/test sets</u>:
key quantities to look at are:

1) human level error 4% 2) Train error 7% 3) Train-dev error 10% 4) dev error 12%

$\underbrace{4\% \quad 7\%}_{\text{avoidable bias}} \quad \underbrace{7\% \quad 10\%}_{\text{degree of variance}} \quad \underbrace{10\% \quad 12\%}_{\text{mismatch problem}}$

5) test set error 12%.

5)-4) sells degree of overfitting to the dev set.

e.g. human error - 4 %.
train error - 10 %.
train-dev error - 12 %.
dev error - 6 %.
test error - 6 %.

going down of error is actually possible if the dev/test set easier than the train set.

27/8/18

Q. We have studied ways to address bias and variance but what about data mismatch?

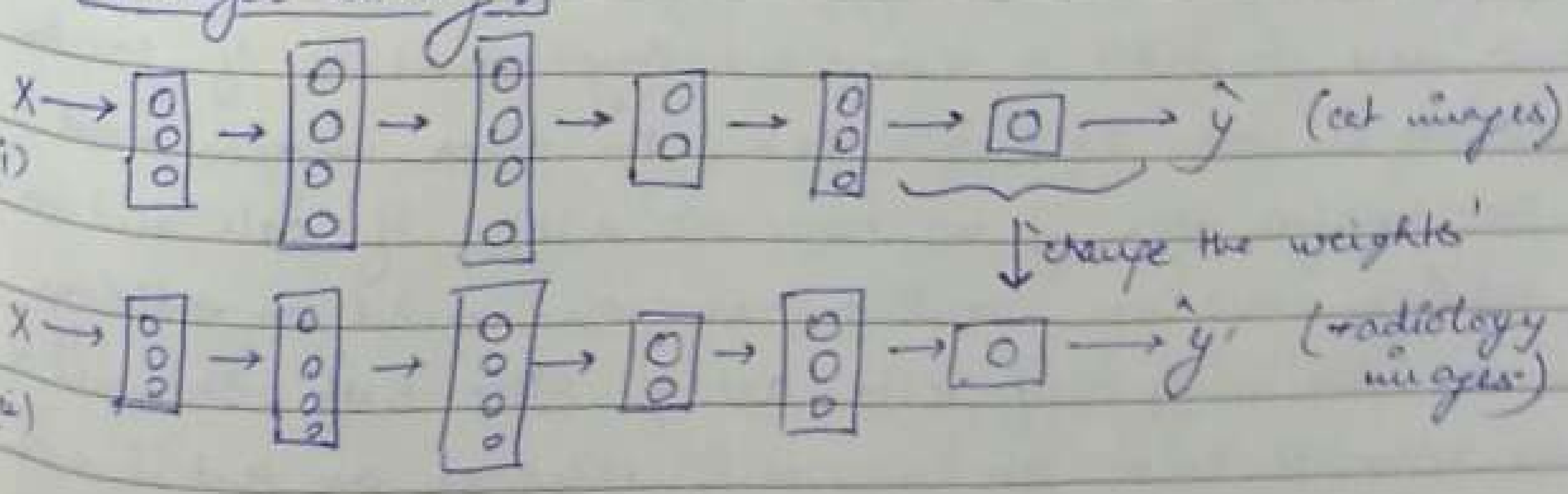
Ans. Instead of bias and variance as 2 potential problems, we now have data mismatch as the 3rd potential problem.
There are few things (though not too systematic) that can help address this problem.

i) Carry out manual error analysis to try to understand difference b/w training and dev/test sets.

ii) Make training data more similar or collect more data similar to dev/test sets on the dimensions that matter and help solve the problem. This can be done using artificial data synthesis becoz it isn't necessary that similar kind of more data is available. But be aware that the synthetic images generated are not accidentally simulating data only from a tiny subset of the space of all possible examples.

Q. How to learn from multiple types of data at the same time?

Ans. Transfer learning:



If the available radiology data is less, then only the last weight (w) can be randomly assigned and then the model can be retrained on the radiology data. If data is bit more then some more layers can be trained as well. The already available trained model is referred 'pre-training' and the change and training of weights according to other data is 'hyperparameter tuning / fine tuning'. This is adopting an existing NN to a different task. It is like learning from one dataset and applying it to another. This can be helpful bcoz a lot of low level features like detecting edges, detecting curves, detecting positive objects are learnt which might help us in learning the other dataset of images bcoz the model already knows what needs to be learnt in order to learn images.

Note: Not only a single layer in place of the last layer, but many layers can be added to the pretrained model in order to learn the new dataset better.

Q. When does transfer learning make sense?

Ans It makes sense when you have a lot of data for the problem you are transferring from and relatively usually less data for the problem you are transferring to. This is because the model (transferred) already knows the intricate features of the data. It only requires to learn the bigger features of the dataset which is less in quantity.

Transfer learning won't make sense if the opposite is true. In the opposite case, the radiology images would be required much more in order to learn and they would be of greater importance as well. In such a case, the other images won't be that helpful bcoz if we have 100 images of cats and 100 of radiology and we want to train our model on radiology, then the 100 images of cats are of little help. It would be rather better to have 200 images of radiology as that is our

Note: Softmax activaⁿ is a good choice for o/p layer if it has a single label o/p. For multi-task learning it doesn't work.

forget and train our model solely on radiology images rather than pretraining it on cat images first.

So, transfer learning makes sense when :

i) Task A and B have same i/p x (like either images or audio or video.)

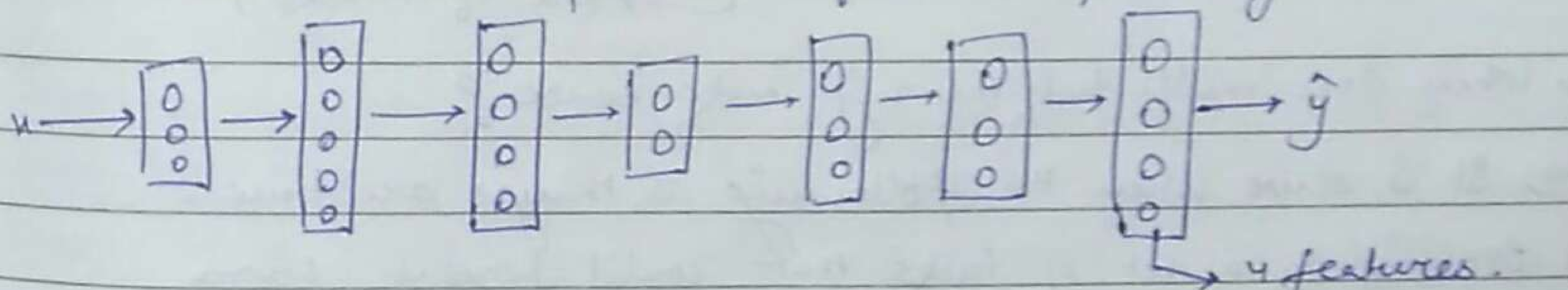
ii) You have a lot more data for Task A than Task B

iii) if low level features from A could be helpful for learning B.

All these points are true considering we want our model to do really well on task B. Also bcoz of this thing, each image of B is of much more importance than each image of A.

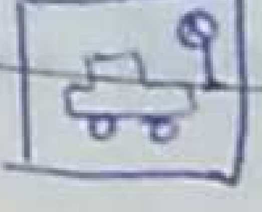
Multitask learning : Instead of learning from multiple datas sequentially as in transfer learning, we can also learn from multiple datas simultaneously and that is called multitask learning.

Note: Softmax regression assigns one label per example while in real life we can also have multiple labels per example. e.g.



like if x is a image of vehicle in traffic, 4 features or labels in y can be if there is traffic light, or pedestrian or sign board or car or bike. Hence, one image can have multiple labels.

$$\begin{matrix} y_1^{(i)} \\ y_2^{(i)} \\ y_3^{(i)} \\ y_4^{(i)} \end{matrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$ denoting that

the input image contains feature y_3 and y_4 but not y_1 and y_2 like  has a car and signboard but doesn't have a pedestrian and traffic light.

In this case $J = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{4} L(\hat{y}_j^{(i)}, y_j^{(i)})$

↳ usual logistic loss

So, while minimizing this J, we are carrying out multi-task learning, bcoz we are building a single NN that is looking at each image and basically solving 4 problems and is trying to tell if each image has each of these 4 objects (y_1, y_2, y_3, y_4) in it.

We could have also trained 4 NN to do 4 things and then combine the result but in that case we would have lost the advantage of having shared features b/w the NN thus leading to a less better performance.

Note Even if some images have only a subset of the labels and others are sorts of question marks or don't cares, we can still train our learning algo to do 4 tasks at the same time.

e.g. if $X = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(m)} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ & \vdots & & \\ 0 & 1 & ? & ? \end{bmatrix}$, then also NN can be trained

→ (case of 4 tasks)

Q. When does multi task learning make sense?

Ans It is sense when the following 3 things are true:

i) Training on a set of tasks that could benefit from having shared lower level features.

ii) Usually, amount of data we have for each task is quite similar. E.g. in the road images discussed before, task of determining true of pedestrian or car or stop sign or traffic light, all need same kind of images to get trained. Suppose we have 100 tasks and we have 10K images for each task, then every task can be helped by the knowledge gained from the other 99 tasks (i.e 990K images) just like what happens in transfer learning where the pretrained model is

trained by suppose 1M images and for the target we have just 1K images which actually get helped by the knowledge the model has gained from training with 1M images.
iii) when we can train a big enough NN to do well on all the tasks.

Note Only time when multitask learning hurts compared to separate NN for each task is if your NN isn't big enough but if we can train a big enough NN to properly learn all the tasks, then multitask learn should not or should very rarely hurt performance as compared to separate NN.

Transfer learning is used more often than multi task learning. Applicaⁿ of multitask lies in e.g. CV as we discussed the traffic e.g. earlier to detect various objects.
So, if we want to solve a data with relatively small size, transfer learning is the soluⁿ where a similar kind of large dataset is fed into the network and then the weights are fine tuned according to the desired problem.

Transfer multitask learning also exists. Multitask learning is not much seen because it is generally not so feasible to have several tasks to be trained on a single NN, object detecⁿ problems being an exception.

⇒ End to End Deep Learning: there have been some data processing systems or learning systems that require multiple stages of processing and end to end deep learning takes all those multiple stages and replace it usually with just a single NN.
e.g. (x)
audio $\xrightarrow{MFCC}$ features $\xrightarrow{ML}$ phenomes ⟶ words ⟶ transcript (y)

pipeline of events to get transcript

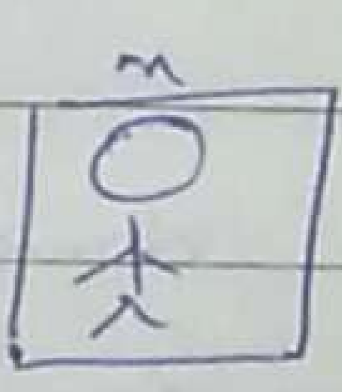
end to end DL changes this to audio ⟶ transcript
⟵ single NN ⟶
bypassing all the intermediate steps.

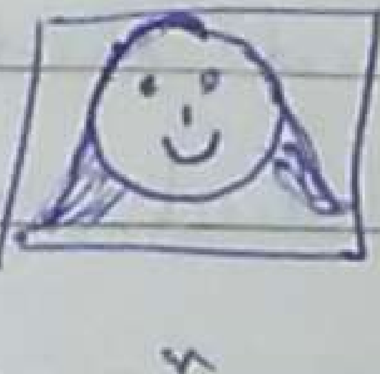
end to end DL needs a lot of data to work well and map x to y to learn its features.

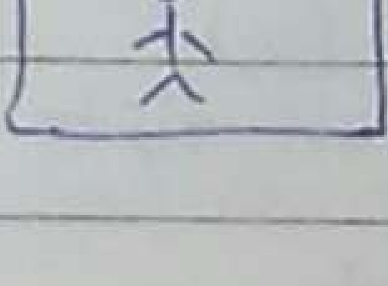
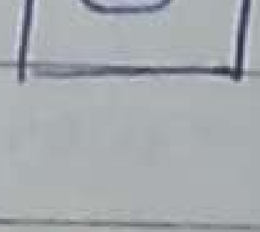
If we have 3000 hr of data, pipeline approach works well but if we have 10,000 hr to 100,000 hr of data end to end DL starts working well. If we have medium amount of data, intermediate approaches are taken i.e halfway pipeline and halfway end to end DL.

e.g. A person is captured in an office camera and his identity needs to be figured out. This problem can be subdivided into 2 problems:
i) zooming in the captured image to just crop the person's face.
ii) identifying the face and figuring out the person's identity.

The end to end DL approach doesn't work very well on the complete problem but on the two subproblems. This is because a lot of data is available of (x, y) form where x is a person's image and y is the loc^n of face in the image and also of form (x', y') where x' is the face and y' is the identity. But hardly any data is available of (x, y) where x is the captured image and y is the identity.

x

⟶ identity y (not available)

x

⟶ identity (data available)

x

⟶
y

(data available)

Note End to end DL is called so bcoz a direct mapping is done from one end of the system all the way to the other end of the system.

Date
Page

Q. Whether end to end DL should be used or not?

Ans Pros of end to end DL:

i) It let's the data speak: if we have a lot of (x, y) data, then the NN can figures out the mapping fn, however complex it may be. Also it figures out its own logics & fn & not depend on human preconcepn.

ii) less hand designing of components needed.

Cons:

i) Needs a large amount of data.

ii) It excludes potentially useful hand designed components (a way to inject manual knowledge into the algo in case data is not enough to draw complete insights.)

Two main sources of knowledge for an algorithm
- data
- Hand designed components/ features or other things.

Note hand designed components is a double edged sword bcoz it can be harmful also as it restricts the model to think in a certain way and not allow it to devise its own methods & conclusions.

e.g. in speech recognition system, phonems are a human conception to understand speech. The model need not necessarily interpret the speech in this manner and may find some other (which can be better also) way of dealing with the speech.

Note If you have sufficient data to learn a fn of of the complexity needed to map x to y, then end to end deep learning approach is worth applying.

e.g. (x, y) = (posn of objects on road, steering dirn of car) can be too much complex problem to be solved end to end.

Hence, autonomous car driving is not a problem to be solved by end to end DL considering the (x,y) type data available and the types of things we can learn with NN today.