CS182/282A Deep Neural Networks **April 14^{th}, Spring 2023**

Lecture 21: Catastrophic forgetting, knowledge distillation and meta-learning

Lecturer: Anant Sahai *Scribes: Johan Fredrik Agerup*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

Prof. Q and A

Red boxes are for questions from lecturer.

Class Q and A

Blue boxes are for questions from class.

Comment from lecturer.

Elaboration of comment (if needed).

21.1 Rest of the course preview

Preview of where we are going - want to achieve two things:

1. Focus on **generative models** and generative AI such that modern **diffusion models** are substantially treated.
2. Understand the road that was taken to get to the **generative pre-trained transformer (GPT)**.

21.2 Catastrophic forgetting (CF)

"When a model learns something new, it can forget something it already knows" [1,4].

This phenomenon can be viewed as both a feature and a bug. In this lecture, we will treat this phenomenon called *catastrophic forgetting* as a bug.

21.2.1 Treat CF as a bug

Instructive example:
A vanilla setup to explain when CF occurs is that we are given a multi-class classification problem for which we train the model on each of the classes sequentially. For example, if we use the MNIST datasets (images of numbers from 0-9) we train first only on the 1's, then on the 2's, and so on. After finishing the training on the 1's, both the training loss and the held-out loss is small for the 1's, but when the model starts to train on the 2's, the loss increases for both the 1's and the 2's.

Thus, if we view multi-class classification as many different binary tasks and train at one task at a time, it will result in bad performance on all classes [1].

Different approaches for training on multiple tasks

1. **Freezing the model:** If the pre-trained model is changed while working on a specific task, then forgetting is a risk (i.e. parameters are not trained).
2. **Linear probing strategy:** Freeze the pre-trained model by only training a task-specific head.
3. **Soft prompt before pre-trained model:** For language models or anything that can be prompted, it is possible to insert a soft prompt before the pre-trained model. In this way, the pre-trained model is not changed, thus forgetting is not a risk.
4. **Low-rank adapter:** Instead of changing all the weights in the pre-trained model, a low-rank adapter can be added to the pre-trained model to enable low-rank updates on the weights. The risk of forgetting is only present when doing full fine-tuning and does not occur (necessarily) when doing low-rank weight adjustments.

21.2.2 Strategies to combat CF when doing full fine-tuning

What should you have done in the multi-class classification problem instead of training on each class sequentially?
Gold standard strategy: In the multi-class classification problem, the data should have been shuffled before training.
So, what shall we do to avoid forgetting?
We want to see all the tasks during training - this is called *replay*.

21.2.2.1 Replay

Replay is a term for shuffling in data of "old" tasks during training on "new" tasks - older parts of the sequence are *replayed* to avoid forgetting.

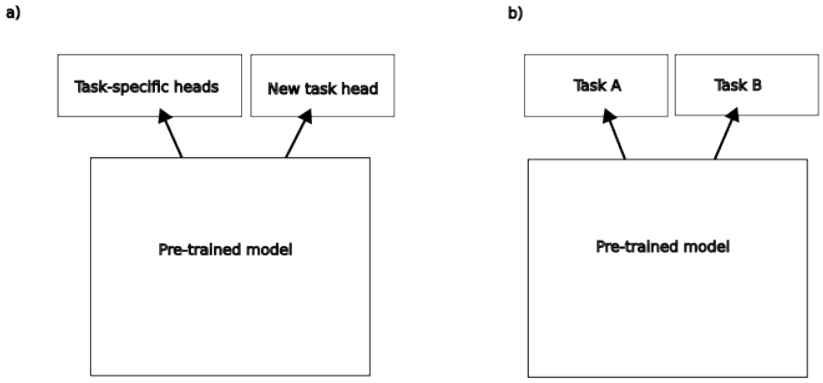

Figure 21.1: a) General problem setting for catastrophic forgetting. b) Example of learning a new task B given a model pre-trained on task A. (Source: Taken from lecture).

The problem setting, illustrated in Fig. **??** a), is that we want to train a new task-specific head. The model already has other pre-trained task-specific heads, and we have data available for the new task. In practice, training a "new" task-specific head involves injecting data from the other task-specific heads during training such that weights associated with the "new" task are affected by the gradients of the other task-specific heads, not just the "new" task-specific head. Thus, the gradients associated with the "old" task-specific heads balance out the gradients from the "new" task-specific head that is pulling in the direction of forgetting the old task. However, this leads to the challenge of requiring the old data. If the model is supposed to be trained on many tasks, this implies working with an amount of data that prevents training the model due to memory and compute limits.

When you have many tasks, do you want to use separate pre-trained models for all the tasks or is the pre-trained model shared across tasks?
There is a risk of forgetting only when the model is shared across different tasks. We want to share weights across tasks, so we have to deal with the risk of forgetting.

The terms "old" and "new" tasks can be misleading as this leads us to think about doing these tasks in some sequential manner. It should rather be thought of as continual learning, where the "new" tasks are learned without forgetting the "old". By adding continual learning, the model will adapt to problem related changes. Thus, continual learning strengthens the robustness of real-world applications. Continual learning is also useful in situations where some new demand in the learning problem will come up. In these situations the term "new" is related to time (and not additional task).

Aside (important when we think about the road to GPT): What if we don't have access to task-specific data and heads for "old" tasks during fine-tuning for "new" tasks?

Related to the rhetorical question above. In practice, the setting is that you have a model and want to fine-tune it for the "new" task. You have data and task-specific heads for the "new" task but you do not have data or task-specific heads for the other tasks. **How do you combat catastrophic forgetting?**

Class: Introduce some additional data (data augmentation) to increase the potential number of tasks the model will train on by making the data less task-specific.
Prof: This is one way of doing it but another better way of acquiring other data is based on the pre-training idea where we have data that is not necessarily task-specific but it is representable for the data you might see in the task-specific cases. These data can be thought of as proxy tasks (aka surrogate tasks). In practice, this means that these pre-training tasks will be "shuffled in" during training. **Note** that this is not as good as having task-specific data but it is better than augmented data (and having no data).
Example: ChatGPT mixed in pre-training data when training the model to be better at interacting with people when doing next-token prediction (autocomplete). This made the model better at next-token prediction.

While doing fine-tuning and avoiding catastrophic forgetting do you need to be worried about adjusting learning rates for each task?
This has been experimented with. Loss terms for every task are summed in the general loss function - thus each loss term should have similar units. Acquiring similar units requires adjusting weights to compensate for the difference in units. Adjusting the weights is comparable to adjusting the learning rate for each task. It is hard to disentangle how the learning rate is implicitly adjusted by summing up different tasks in the loss function and explicitly adapting the learning rates for the different tasks. However, there are ideas about shaping the loss, e.g. there are different variants of entropy measures that involve raising the arguments to powers (Renyi, Renyi cross-entropy that in some limit goes to Shannon cross-entropy, etc.) instead of using a logarithmic function.

What if we had a task-specific head but no task-specific data?
Let a model have two task-specific heads A and B, as illustrated in Fig. **??** b), e.g. for classifying natural scenes and classifying medical images. We only have task B-specific data (x_i, y_i) where x_i are examples coming from task B distribution and y_i are task B labels.

Goal: We want to avoid CF on task A - but we are missing task A labels. The typical reason for not having task A labels is due to privacy but in this case the task A head is available. If you are serving a model to two customers you would want to use what the model has learned from the first customer on the second. So the model has to be trained on the second customer without degrading the model performance on the task associated with the first customer. The model is already pre-trained on task A, and since ML models are "data-hungry beasts that get stronger by eating data," the model is better off by training on task B when pre-trained on task A compared to random initialization of the weights.

So, how to acquire task A labels?
Task A labels can be generated from task B data using the task A-specific head. This leads to the concept of *knowledge distillation*.

21.3 Knowledge distillation

The underlying idea of generating task A labels from task B data using the task A-specific head is called *knowledge distillation*. The core idea in knowledge distillation is to use neural networks (NNs) to generate/augment labels and use the generated/augmented labels to train the model. The idea of knowledge distillation is the following:

1. Generate pseudo labels for task A on task B data by using task A net. Task B labels are one-hot encoded and pseudo labels are generated real-valued vectors.
2. We want a relevant "knowledge distillation loss" on task A pseudo labels during task B fine-tuning, e.g. turn scores of pseudo labels into probabilities using softmax and then use cross-entropy on these probability-like vectors. These vectors contain more useful information for training compared to differences in class predictions which are one-hot encoded. Taking the argmax over the probabilities throws away useful information. The probabilities are defined as:

$$\frac{\hat{y}(i)^{1/T}}{\sum_j \hat{y}(j)^{1/T}} \tag{21.1}$$

   where $\hat{y}(i)$ is the i'th logit produced from the pseudo labels and T is a number that increases the weight of smaller logit values when set to $T > 1$ (as suggested by [3]). In this case, the network is encouraged to better encode the similarities between classes [3, 4]. Task A and B might be very different, and in this case, it might be more sensible to use polynomial-type functions to get the probabilities as a higher T produces a softer probability distribution over classes [3]. In the literature, T is often called temperature, and as the Eq. **??** suggests the probabilities are equal in $\lim T \to \infty$. This is related to the fact that high temperature is related to randomness.

Can we use the similarity measure between tasks to tune the parameter T?
Yes, it is possible to use the "similarity" between tasks to tune T but the problem is one of context. To understand the similarity between two tasks you have to look at specific examples. Therefore, there is no definitive answer to this question.

When do we know if task A and B are similar "enough" that it makes sense to train task B on a model pre-trained on task A?
Deciding whether to use the model pre-trained on task A (fine-tuned on task A) on task B is based on judgement. However, if you are training in an online fashion (online learning) and have compute, then you have the resources to train the model on both tasks A and B. Since the resources to train on both tasks are available, held-out validation data can be used to decide whether it is reasonable to train on task B from a model pre-trained on task A. One alternative is to define a hyperparameter on whether to train on both tasks or not. Another alternative in the case when old and new have some sequential meaning and we are training in an online fashion is that hyperparameter-tuning can be done by using a mixture of experts/multiplicative weights bandit-style algorithms to make choices between training on both tasks or not.

The paper that introduced the problem addressed in the above question is called "Learning without forgetting" [4].

A general engineering idea is that you start without knowing how to solve a problem but you know what you need to know to solve the problem. In the spirit of deep learning (DL), the natural follow-up question

would then be if a neural net (NN) is available or if you can train a NN to solve for the missing elements in your problem. If an NN can be used, then the problem should be solvable. There is an ongoing transition in DL where we are using NNs to replace the missing elements in the problems we are trying to solve. An analogy to this tendency is that humans advanced significantly when they started to use tools to make new improved tools.

21.3.1 Why knowledge distillation?

Fig. **??** represents the heart of what we are doing in machine learning, which is trying to generalize (learning the underlying pattern). In this context the NN can be treated as the line in Fig. **??** representing the truth/underlying trend and query it to get the data. In some cases, measurements/examples might be missing for some parts of the domain as illustrated in Fig **??**. Revisiting the previous example of task A and B, the missing data points represent the labels from task A that is no longer available when we want to train on task B without forgetting what the model has learned for task A. In this situation, task B inputs can be used to generate pseudo labels for task A, which are mixed in when training on task B. This is a rough explanation of how forgetting can be avoided. To summarize, when learning "new" tasks we treat a trained NN like the line illustrated in Fig. **??**, and query the NN to get data (red points) that is missing (between the vertical dashed lines) for "old" tasks.

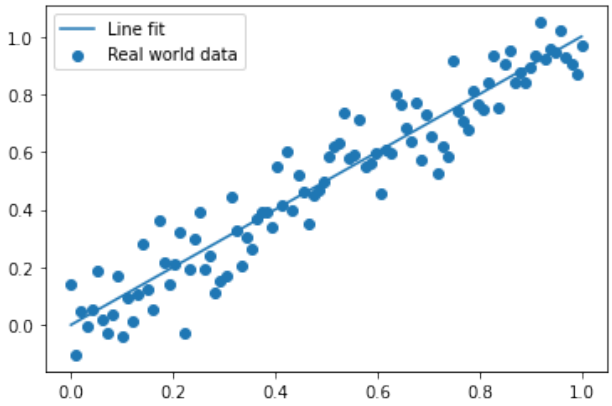

Figure 21.2: The figure represents the question of whether the line or the real-world data is "more real". Which one is closest to the truth depends on the point of view (POV). One POV is that real-world data is scattered as the points represent data from measurements associated with some uncertainty. Another POV is that the line represents an ideal world but not a realistic one, thus the scattered data points are closer to the true behaviour of whatever the data represents.(Source: Taken from lecture).

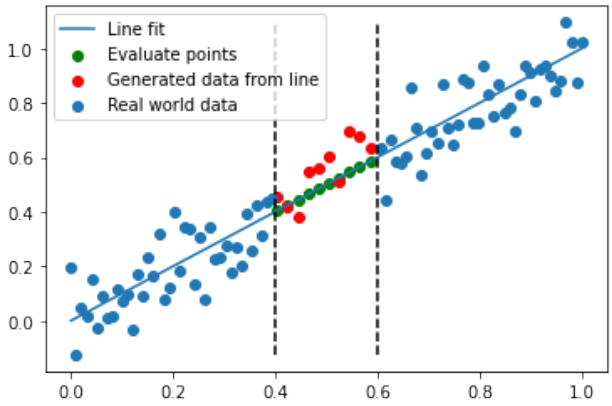

Figure 21.3: Data is missing for a part of the domain. In this plot, it is missing data between the vertical dashed black lines. The missing data can be mimicked by evaluating the NN representing the line (green points) and adding noise to resemble the real-world data. The red points represent the data generated by evaluating the NN (representing the line) with noise added to it. (Source: Taken from lecture).

Use cases of knowledge distillation:

1. Make a smaller model leveraging a large one. This means that a large model is approximated to fit in a smaller model and is done by using the output from the large model as input for the smaller model.
2. Self-distillation, which is used to improve the training of the model. In practice, the model is trained and gives the counterpart of a line (Fig. **??** and **??**). Then, the trained model is used to train another model with the same architecture (size) using data and output from the trained model. This process is repeated to iteratively improve the performance of the model.

Demystification of meta-learning

How do you generate the data (red points in Fig. ??) for task A if you do not have the input data of task A?
We have the NN trained on task A, so we have the line associated with task A. Then, we use task B as input data to generate data points for task A.
What happens then in the case where task A and B are very different, e.g. the example of task A being classification on natural scenes and task B being classification on medical images?
If the tasks are very different in the sense of their data distribution and you are querying a model classifying natural scenes on medical data, you are asking the model to extrapolate. The resulting extrapolation might be total gibberish. However, this leads to the question of whether the query (question you are asking the model) is gibberish. If the question has no meaningful answer then the fact that the answer is gibberish might not be a problem. You still have to worry because we are trying to avoid perturbing distinctions that are relevant for task A ("old" tasks) when training on task B ("new" tasks). The danger when extrapolating is that if the data for task B is such that it does not even excite modes that are relevant for task A in the pre-trained model this will lead to an unstoppable collapse along those modes. Thus, we can only hope that task B input data will excite modes in the pre-trained model relevant to task A.

So, can you combat this by a clever choice of knowledge distillation loss?
No, there is a saturation point for the richness of task B data in terms of having distinct components in some directions compared to task A, and this will be the limitation of choosing a clever knowledge distillation loss. No knowledge distillation strategy will give observability on the shared part of the model. There are cases where we have some degree of observability of where the training collapses for task A. In that case, the knowledge distillation loss has to be chosen such that the force of the gradients is strong enough to stop this collapse.

21.4 Meta-learning

In *meta-learning*, fine-tuning is central ("on a task you can see") as meta-learning means **to learn how to learn**. In practice, this means **"we want to be trained so that we are good at being fine-tunable"**.

Note: this is a distinction from "post-train" fine-tuning where the aim is to modify parameters after the main training is done. Now, the fine-tuning happens as we train since we know we are going to fine-tune anyways. This saves time and will in most cases produce better models.

Historical background: The importance of meta-learning became clear when people started using Image-Net/models trained on Image-Net to be fine-tuned for other tasks. In the context of self-supervision, we want to find a proxy task that will help fine-tune the model. The meta-learning perspective is that we know we want to be good at fine-tuning as opposed to hoping the model will be good at being fine-tuned. The distinction here is that latter perspective treats fine-tuning simply as an interesting emergent property, while the meta-learning perspective considers optimizing fine-tuning as we train the model. So, this leads to the question of how to do this in practice.

Meta-learning approaches: Assume that data is available for multiple tasks at training time. For Task $1, 2, \ldots, L$, we have datasets (x_{ij}, y_{ij}) where x_{ij} and y_{ij} are the examples and labels associated with the i'th task and j'th sample. We want to generalize being fine-tunable, meaning that the fine-tuning works across different tasks (is not task-specific). If we successfully generalize, then the model should be able to perform

well on a new task. So, the goal is given a new task $(x_j^{\text{test}}, y_j^{\text{test}})$ - the model performs well on held-out data $(x_{\text{test,test}}, y_{\text{test,test}})$. Doing well means fine-tuning on the new data $(x_j^{\text{test}}, y_j^{\text{test}})$ while performing well on the held-out data $(x_{\text{test,test}}, y_{\text{test,test}})$.

Do we care about forgetting the initial task in meta-learning?
It depends, but the initial perspective is that we only care about being fine-tunable on the new task and do not care about if that causes problems for other tasks. There is an intellectual connection between forgetting and learning fast. The goal is that the model should be able to learn the new task with a small amount of data. The idea is that the problem environment is changing rapidly and we want the model to quickly adapt to these changes. Additionally, in the context of robotics, training data is very expensive to acquire.

So, what is the connection between forgetting and fine-tuning? It is helpful to think about it in the context of having a small amount of data. Let us focus our intuition on a situation where we have little data per task (less data points than parameters in the model). Then, we can think that the model we have and the initialization are such that there are only a few important directions that we can train along considering the SVD of a locally linearized model. The SVD informs us along which directions the training will move associated with singular vectors corresponding to a few large singular values. This can be thought of as the behaviour of the "locally linearized model" (around the training data). Thus, we expect that the model will forget during training if these singular vectors points along similar directions (projections of one onto the other has a large magnitude) across several tasks. This is because the weights will be updated along the same components, so maximal forgetting occurs for parallel singular vectors. On the other hand, if the models are different in the neighbourhoods of the data across tasks, the singular vectors will point in different directions, and in the limit of maximal difference these vectors are orthogonal. In this case, training on the new task induces insignificant perturbations along the directions of the singular vectors associated with the other tasks and forgetting will not be an issue.

21.4.1 Model agnostic meta-learning (MAML)

In *model agnostic meta-learning*, the aim is to optimize meta-learning. Model agnostic refers to the fact that meta-learning is compatible regardless of the details of the model. The natural strategy is to do well on **defining** tasks that are as "orthogonal" as possible in the sense of the singular vectors associated to the SVD of a model locally linearized around training data. MAML is the baseline approach to combat the problem associated with "similar" tasks meaning MAML wants to do SGD on the problem of ensuring good performance given a new task using a small number of training examples [5]. The intention is to achieve rapid adaptation to new tasks.

21.5 What I wish this lecture also had to make things clearer?

While I do understand the level of abstraction the lecturer has chosen to not "get lost" in the details, details on the SVD of a locally linearized model in the context of forgetting in meta-learning would be helpful for clarification. To me, it is not clear exactly what is decomposed and what is meant by a locally linearized model (Linearized around what?). I think this could be fixed by specifying what is decomposed, and explaining how to locally linearized the model around which data. Furthermore, two examples can be specified in class where this leads to similar and different singular vectors associated with the large singular values of the SVD. The explanation given in class was understandable on a high level as you do not want the singular vectors associated with large singular values of a locally linearized model of one task to be similar to

another task as this will lead to forgetting when doing gradient descent updates. However, it is still difficult to concretize and see how one can check if forgetting is a risk or not. If the time limitation is too tight, I can suggest just mentioning a reference that explains and clarifies this (maybe even come up with some specific examples).

References

[1] McCloskey, M. and Cohen, N.J., 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In Psychology of learning and motivation (Vol. 24, pp. 109-165). Academic Press.

[2] Goodfellow, I.J., Mirza, M., Xiao, D., Courville, A. and Bengio, Y., 2013. An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211.

[3] Hinton, G., Vinyals, O. and Dean, J., 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

[4] Li, Z. and Hoiem, D., 2017. Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence, 40(12), pp.2935-2947.

[5] Finn, C., Abbeel, P. and Levine, S., 2017, July. Model-agnostic meta-learning for fast adaptation of deep networks. In International conference on machine learning (pp. 1126-1135). PMLR.