Common privacy attacks for NLP models

Some common attacks are membership inference, reconstruction, property inference and model extraction. A membership attack aims to discover whether a piece of information was present in the training data. Reconstruction attacks use partial knowledge to reconstruct the original training data and labels. Property inference attacks aim to extract some encoded knowledge from the training data that wasn't explicitly stated, such as the country of origin or gender of those in a dataset. Model extraction attacks aim to extract information from the model in an attempt to reconstruct and substitute it. This also includes methods that aim to extract hyperparameters or features of the architecture (such as learning rate, batch size, projection/hidden layer feature dimensions, etc.). LLMs can also be susceptible to jailbreaking and prompt injection attacks. E.g. LLMs can be prompted to reveal potential training data by a variety of methods like entering a <start> token. The system can then be prompted with the first section of a potentially leaked string with the remainder masked. An adversary can then calculate the probability of the model sharing the same potentially leaked data. This can be framed as analysing the model perplexity - a measure of "surprise". If the model is "not surprised" when predicting a string then it is more likely to be a member of the training set.

How might a malicious user exploit the ability to upload a set of weights to gain access to the sensitive data?

If an adversary had the ability to upload a set of weights then they could upload a model of the same dimensions but trained on poisoned data, allowing the extraction of training data. H. Yao et al. 2023 show that prompt generation can be used to generate a poison prompt set that can be used to train the back door task. While simultaneously utilising bi-level optimisation to train the task on both the malicious task as well as the original downstream task. This could have severe negative impacts on whoever used those weights as it could leave them open to membership inference attacks, sensitive data extraction, model output manipulation, trojan-ing the model or subtle sabotage. This highlights the importance of implementing robust security measures throughout the software engineering process.

How can different model hyperparameters affect convergence patterns and what might those convergence patterns reveal about the underlying data?

Hyperparameters shape the learning process's journey to an optimal minimum. Key parameters like the choice of optimiser, learning rate magnitude, learning rate warm-up, and batch size reveal how complex the optimisation surface is. For instance, learning rate warm-up can leverage prior successful experiments, but disclosing this might compromise IP.

Ideally a smooth, convex optimisation surface with one global minimum is desirable; but rarely achievable. In practice, observing a smooth convergence to a plateau suggests hitting the optimal configuration. A sharp initial loss drop followed by a gentler decline could indicate data that's superficially similar but requires more nuanced learning. Mini-batches can cause temporary deviations, while larger batch sizes smooth out noise or variability, hinting at dataset heterogeneity or sparsity.

The use of momentum in optimisers can accelerate convergence by ignoring occasional misguiding samples, revealing data or task complexity. However, over reliance can lead to overshooting. Advanced optimisers like AdamW, which adaptively adjust learning rates and include weight decay to improve convergence may indicate a more intricate dataset, compared to simpler methods like mini-batch SGD.

In the context of federated learning, momentum helps counteract slow convergence caused by client-specific data variations, effectively smoothing out the aggregated updates and guiding the model towards faster convergence. When a single client's model weight updates differ significantly from others due to non-IID data, it can push the aggregated global weights in a drastically different direction, leading to slower training and potential instability. It could reveal that the training data of that specific client has some features which differ from the rest or contain information that the others do not. The creators of FedClust also attempt to address this problem with weight driven clustering to improve overall model accuracy and convergence speed.

<u>What approaches could one take to reduce the amount of raw training data the model remembers?</u>

Methods to reduce the amount of raw training data the model remembers would include:

A. Text Anonymisation
B. Safety Training
C. Safety Model Editing
D. Differential Privacy in Federated Learning

Text anonymization has limited usefulness as it doesn't help remove inferred characteristics. E.g. If a redacted client is a "UK based jet turbine manufacturer" then the client must be Rolls Royce Ltd as they are the only UK based jet turbine manufacturer.

Meta's 77-page paper on how they trained llama by Touvron et al. described a three step approach to model safety training which yielded desirable results for Llama 2. 1. supervised fine-tuning on hand-crafted prompts. Followed by 2. Safety Reinforcement Learning with Human Feedback. Concluding with 3. safety context distillation to generate safer responses effectively internalising the safety context directly into its behaviour.

Safety Model Editing refers to techniques like Detect and Edit Privacy Neurons (DEPN) which utilises gradient integration to identify neurons associated with private information, then set their activation to zero. Essentially "switching off" the neurons associated with training data leakage.

Differential Privacy allows us to quantify how model outputs change with granular changes to the dataset (and resulting weights). By adding calibrated noise to gradient updates and clipping the norms, one limits data exposure and regularises the contribution that each client provides to the server. However this increases the computational overhead and slows down convergence.

Engineering & Security

To ensure privacy and security in a production setting, one must implement Role-Based Access Control and encryption. One good example of this is GCP's Key Management Service such as Workload Identity Federation. Utilising Kubernetes best practices such as defining network policies and using private clusters with the appropriate resources is paramount. One should be sure to secure model endpoints with an API Gateway, enforce authentication (OAuth 2.0 or JWT), and validate inference requests to prevent injection attacks or handling of malicious data. Monitoring and logging tools like OpenTelemetry can help capture requests between microservices to help track resource usage, detecting breaches and enable audit logging for traceability.