Supplementary Material: Efficient On-Device Models using Neural Projections

Sujith Ravi [1]

1. ProjectionGraph

Going beyond deep learning, we also extend our framework to novel settings and train lightweight models in other types of learning scenarios. For example, the training paradigm can be changed to a semi-supervised or unsupervised setting. The *trainer* model itself can be modified to incorporate structured loss functions defined over a graph or probabilistic graphical model instead of a deep neural network. Figure 1 illustrates an end-to-end *projection graph* approach for learning lightweight models using a graph optimized loss function.

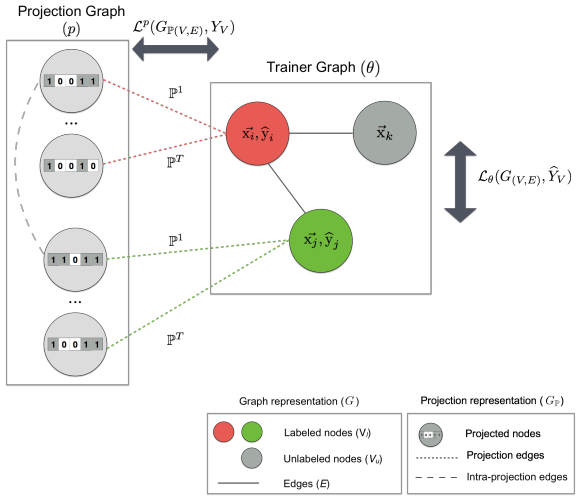


Figure 1. Illustration of a ProjectionGraph model trained using graph learning algorithms.

Notation for trainer graph: $\vec{x}_i$ represents an input node and $\vec{x}_j$, $\vec{x}_k$ represent its neighborhood $\mathcal{N}(\vec{x}_i)$ in the original graph $G = (V, E)$. E refers to edges and $V = V_l \cup V_u$ the nodes in this graph, where V_l indicates labeled nodes *(red, green)* and V_u the unlabeled nodes *(grey)*. $\hat{y}_i$ indicates the ground-truth corresponding to a labeled node $\vec{x}_i \in V_l$, with *red, green* colors indicating different output values for $\hat{y}_i$. If $V = V_l$, the graph is trained in a supervised manner whereas $V_l \subset V; V_u \neq \varnothing$ yields a semi-supervised graph learning formulation. *Notation for projection graph:* Each node $\vec{x}_i$ in the trainer graph G is connected via an edge to a corresponding projection node in $G_{\mathbb{P}}$. The projection nodes are discrete representations of trainer node $\vec{x}_i$ obtained by applying the projection functions $\mathbb{P}^1 ... \mathbb{P}^T$ to the feature vector associated with the trainer node. In addition, $G_{\mathbb{P}}$ may also contain intra-projection edges computed using a similarity metric applied to the projection vector representations. For example, we can use Hamming distance $\mathcal{H}(.)$ to define a similarity metric $1 - \frac{\mathcal{H}(.)}{d}$ between projection nodes represented as $d-$bit vectors. The training objective optimizes a combination of graph loss $\mathcal{L}_\theta(.)$ and projection loss $\mathcal{L}^p(.)$ that biases the projection graph to mimic and learn from the full trainer graph. $\mathcal{L}_\theta(.)$ optimizes the trainer's predicted output y_i against the ground-truth $\hat{y}_i$ whereas $\mathcal{L}^p(.)$ optimizes predictions from the projection graph y_i^p against the neigbhoring trainer predictions y_i.

The ProjectionGraph model can be trained efficiently using large-scale distributed graph algorithms (Ravi & Diao, 2016) or even neural graph approaches (Yang et al., 2016; Bui et al., 2018). The projection model training can also be further extended to scenarios involving distributed devices using complementary techniques (Konecný et al., 2016). We leave these explorations as future work.

2. On-Device Conversational Models

We demonstrate the effectiveness of the proposed projection-based learning architectures for powering on-device predictions for conversational applications on smartwatches and mobile devices. Smart Reply (Kannan et al., 2016) is a widely-used feature for automated reply suggestions in email and chat conversations. With Smart Reply, a user can easily send a quick reply (e.g., "yup") in response to a chat message (e.g., "Are you on your way?") with a single tap. These responses are especially useful on mobile or wearable devices where text input may be slower than on desktops.

The original Smart Reply feature relied on cloud-based models for both training and prediction, i.e., every prediction required sending the input message to a server which may

[1]Google Research, Mountain View, California, USA. Correspondence to: Sujith Ravi <sravi@google.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

require an additional server roundtrip with associated power costs. The cloud-based machine learning architectures are also computationally intensive, have significant memory requirements, and require an active internet connection. This makes it infeasible to make predictions on a users phone, smartwatch, or IoT device.

To solve this problem, we developed a fully on-device Smart Reply system that generates response predictions to incoming conversations (shown in Figure 2). The system builds and trains a projection-based model (described earlier) on the cloud but once trained, the model is downloaded to a mobile device and runs all predictions locally on device. This allows products to generate Smart Reply responses with low latency, even when the device is not connected to the internet. Additionally, this architecture increases user privacy by allowing predictions to be made without input messages ever having to leave the local device.

Figure 2. On-device smart reply predictions for incoming message notifications on a smartwatch. Replies are generated using Projection model.

Figure 3 gives an overview of the learning and inference steps for this system. Training is performed using a *projection*-based learning algorithm (ProjectionNet or ProjectionGraph). The projections capture representations of a message x obtained by applying projection functions P to the feature vector associated with x (example illustration in the bottom half of the right side of Figure 3). For conversational applications like Smart Reply, we use text features (e.g., skip-grams) extracted from the message content. Once trained, the *projection* model is downloaded and inference is performed directly on the mobile device. For inference, we apply the same steps as in training—pre-processing, feature extraction and projection. The learned weights from the projection model that are relevant to the particular message (or conversation) are then applied to produce the final ranked list of response suggestions (Figure 3). The overall on-device model size is only a few Megabytes, *tiny* (100-1000x smaller footprint) compared to Cloud-based LSTM models that are used for similar prediction tasks.

On-device Smart Reply models[1] have been deployed to wearable devices and enable messaging applications on smartwatches powered by Android Wear 2.0 (shown in Figure 2).[2] We have also released an on-device conversational model in *TensorFlow Lite* (tfl), an open-source library specifically designed for on-device machine learning.[3]

3. Computing Model Predictability in Neural Bits

We further study the notion of predictability of deep neural networks in the context of *neural projections*. More specifically, using the ProjectionNet framework described in Section 2.1, we formulate the questions: "How many neural bits are required to solve a given task?" "How many bits are required to capture the predictive power of a given deep neural network?"

These motivate a series of studies on the different datasets. We show empirical results to answer these by computing the number of bits encoded in the *projection* network (Section 2.1) used in each task. Since a very simple projection network architecture can be represented in bits (i.e., output of projection operations result in bit vectors) and we only use a single layer of projections, we can compute the total number of bits required to represent a specific ProjectionNet model. If we compare this to the accuracy achieved on a given task by applying the *projection* network alone for inference, this helps answers the first question.

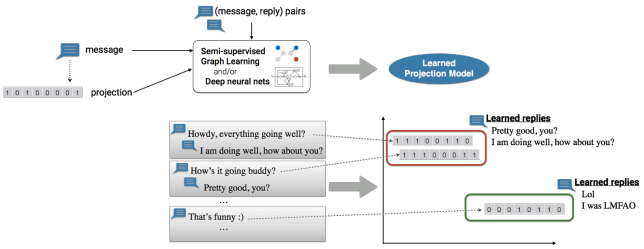
On the visual recognition task for MNIST (Section 3.1), 80-100 neural projection bits are enough to solve the task with 70-80% accuracy, and increasing this to 720 bits achieves 92.3% precision which is further improved by using a deeper projection network. For the language task of semantic classification (Section 3.2) involving sequence input, 720 neural projection bits are required to achieve a top-1 accuracy of 82.3%. Going deeper with projections and combining with additional non-linear operations results in further improvements, yielding 97.7% that outperforms even strong LSTM baselines.

To answer the second question, we use a given deep neural network (e.g, feed-forward NN) with specified configuration to model the full *trainer* network in the framework described in Section 2. We then use this to train a corresponding neural *projection* network with hidden bit representations. Finally, we compute the number of neural projection bits used in

[1] https://ai.googleblog.com/2017/11/on-device-conversational-modeling-with.html

[2] https://developer.android.com/reference/android/app/ Notification.Action.html#getAllowGeneratedReplies()

[3] Download code & model here: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/models/smartreply/g3doc



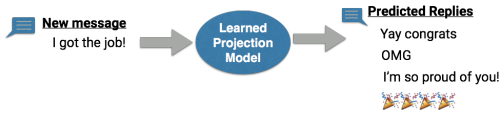


Figure 3. Left: Learning projection model for Smart Reply. *Right:* Inference using on-device Smart Reply model.

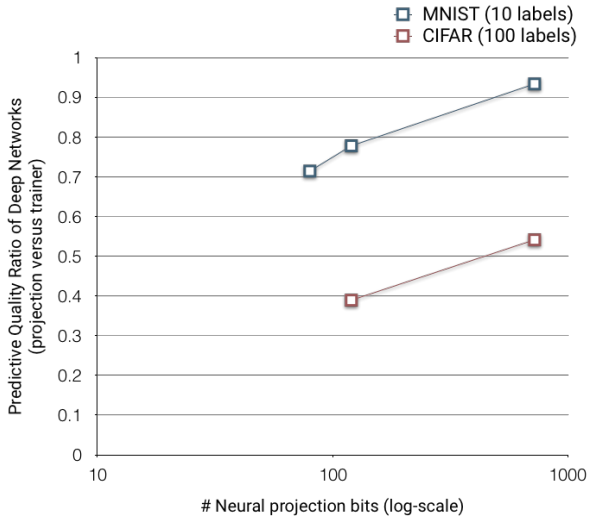


Figure 4. Comparing predictive power of deep neural networks using neural projection bits on different visual classification tasks.

the second network that simulates the *trainer* and plot this value against the *predictive quality ratio*, which is defined as the ratio of accuracies achieved by separately performing inference with simple versus full network on a given task. Figure 4 shows this plot for the MNIST and CIFAR-100 tasks. The plot shows that the predictive power of a 3-layer feed-forward network with 3-5M parameters can be succinctly captured to a high degree (ratio=$\sim$0.8) with a simple 100-bit ProjectionNet for MNIST classification and just 720 bits are required to recover more than 90% of the base deep network quality. On more complex image recognition tasks that involve larger output classes, a higher number of bits are required to represent a *trainer* deep network with the same architecture. For example, on CIFAR-100 task, we observe that a 3-layer feed-forward network can be projected onto 720 neural bits at a predictive ratio of 0.5. However, we also notice a steep increase in predictive ratio moving from 120 to 720 neural bits. We expect a similar trend at even higher bit sizes and on more complex tasks.

References

TensorFlow Lite. https://www.tensorflow.org/lite/.

Bui, T. D., Ravi, S., and Ramavajjala, V. Neural graph learning: Training neural networks using graphs. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018.

Kannan, A., Kurach, K., Ravi, S., Kaufmann, T., Tomkins, A., Miklos, B., Corrado, G., Lukacs, L., Ganea, M., Young, P., and Ramavajjala, V. Smart reply: Automated response suggestion for email. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

Konecný, J., McMahan, H. B., Ramage, D., and Richtárik, P. Federated optimization: Distributed machine learning for on-device intelligence. *CoRR*, abs/1610.02527, 2016. URL http://arxiv.org/abs/1610.02527.

Ravi, S. and Diao, Q. Large scale distributed semi-supervised learning using streaming approximation. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 519–528, 2016.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. *CoRR*, abs/1603.08861, 2016. URL http://arxiv.org/abs/1603.08861.