Efficient On-Device Models using Neural Projections

Sujith Ravi [1]

Abstract

Many applications involving visual and language understanding can be effectively solved using deep neural networks. Even though these techniques achieve state-of-the-art results, it is very challenging to apply them on devices with limited memory and computational capacity such as mobile phones, smart watches and IoT. We propose a neural projection approach for training compact on-device neural networks. We introduce *projection* networks that use locality-sensitive projections to generate compact binary representations and learn small neural networks with computationally efficient operations. We design a joint optimization framework where the projection network can be trained from scratch or leverage existing larger neural networks such as feed-forward NNs, CNNs or RNNs. The trained neural projection network can be directly used for inference on device at low memory and computation cost. We demonstrate the effectiveness of this as a general-purpose approach for significantly shrinking memory requirements of different types of neural networks while preserving good accuracy on multiple visual and text classification tasks.

1. Introduction

Recent advances in deep neural networks have resulted in powerful models that demonstrate high predictive capabilities on a wide variety of tasks from image classification (Krizhevsky et al., 2012) to speech recognition (Hinton et al., 2012) to sequence-to-sequence learning (Sutskever et al., 2014) for natural language applications like language translation (Bahdanau et al., 2014), semantic conversational understanding (Kannan et al., 2016) and other tasks. These networks are typically large, comprising multiple layers involving many parameters, and trained on large amounts of data to learn useful representations that can be used to predict outputs at inference time. For efficiency reasons, training these networks is often performed with high-performance distributed computing involving several CPU cores or graphics processing units (GPUs).

In a similar vein, applications running on devices such as mobile phones, smart watches and other IoT devices are on the rise. Increasingly, machine learning models are used to perform real-time inference directly on these devices—e.g., speech recognition on mobile phones (Schuster, 2010), medical devices to provide real-time diagnoses (Lee & Verma, 2013) and Smart Reply on watches (AndroidWear), among others. However, unlike high-performance clusters running on the cloud, these devices operate at low-power consumption modes and have significant memory limitations. As a result, running state-of-the-art deep learning models for inference on these devices can be very challenging and often prohibitive due to the high computation cost and large model size requirements that exceed device memory capacity. Delegating the computation-intensive operations from device to the cloud is not a feasible strategy in many real-world scenarios due to connectivity issues (data cannot be sent to the server) or privacy reasons (certain types of data and processing needs to be restricted to a user's personal device). In such scenarios, one solution is to take an existing trained neural network model and then apply compression techniques like quantization (e.g., reducing floating point precision (Courbariaux et al., 2014)) to reduce model size. However, while these techniques are useful in some cases, applying them post-learning to a complex neural network tends to dilute the network's predictive quality and does not yield sufficiently high performance. An alternate strategy is to train small models for on-device prediction tasks, but these can lead to significant drop in accuracies (Chun & Maniatis, 2009) which limits the usability of such models for practical applications. In particular, feature or vocabulary pruning techniques commonly applied to limit parameters in models like recurrent neural networks, while yielding lower memory footprint, can affect the predicitive capacity of the network for language applications.

This motivates the learning of efficient on-device machine learning models with low memory footprint that can be run directly on device for inference at low computation cost. The main contributions of the paper are:

[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.

- Neural projection approach to learn lightweight neural network models for performing efficient inference on device with low memory and computation cost.
- Joint learning framework that can leverage any existing deep network (e.g., feed-forward, CNN or recurrent neural network) to teach a lightweight projected model in a joint optimization setup which is trained end-to-end using backpropagation.
- Parameterized projection functions that permit efficient computation; can be combined with other operations like convolutions to yield flexible projection neural network models of varying sizes that is configurable based on the task or device capacity.
- Experimental evaluation demonstrating the effectiveness of the projection approach in achieving significant reduction in model sizes while providing competitive performance on visual and language classification tasks (Section 3).
- Novel extensions of the framework for semi-supervised and graph learning settings; on-device projection models for real-world conversational application.

There have been a number of related works in the literature that attempt to learn efficient models under limited size or memory constraints. Some of these works employ techniques ranging from simple dictionary lookups to feature pruning (Stolcke, 2000) or hashing (Weinberger et al., 2009; Shi et al., 2009; Ganchev & Dredze, 2008) to neural network compression. In the past, researchers have proposed different methods to achieve compact representations for neural networks using reduced numerical precision (Courbariaux et al., 2014), vector quantization (Gong et al., 2014), binarization strategies for networks (Courbariaux & Bengio, 2016) or weight sharing (Denil et al., 2013; Chen et al., 2015). Most of these methods aim to exploit redundancy in the network weights by grouping connections using low-rank decomposition or hashing tricks. In contrast, our work proposes to learn a simple projection-based network that efficiently encodes intermediate network representations (i.e., hidden units) and operations involved, rather than the weights. We also introduce a new training paradigm for on-device models where the simple network is coupled and jointly trained to mimic an existing deep network that is flexible and can be customized by architecture or task. As we show in Section 2.1, the specifics of the training process can also include a choice to optimize towards soft targets as in model distillation approaches (Hinton et al., 2015). Dropouts (Srivastava et al., 2014) and other similar variants commonly used in practice for deep learning attempt to reduce parameters during neural network training by dropping unimportant neurons. However, they serve a different purpose, namely for better regularization. (Wang et al., 2015) offers a survey of binary hashing literature that is relevant to the projection functions used in our work. The coupled network training architecture proposed in this paper (described in Section 2.1) also resembles, conceptually at a high level, generative adversarial networks (GANs) (Goodfellow et al., 2014) which are used in unsupervised learning settings to reconstruct or synthesize data such photorealistic images.

2. Neural Projection Networks

In this section, we present Neural Projection Networks and a joint optimization framework for training projection networks with reduced model sizes. We first introduce the objective function using a coupled full+projection network architecture and then describe the projection mechanism used in our work, namely locality sensitive hashing (LSH) and how it is applied here.

2.1. ProjectionNets

Neural networks are a class of non-linear models that learn a mapping from inputs $\vec{x}_i$ to outputs y_i, where $\vec{x}_i$ represents an input feature vector or sequence (in the case of recursive neural networks) and y_i is an output category for classification tasks or a predicted sequence. Typically, these networks consist of multiple layers of hidden units or neurons with connections between a pair of layers. For example, in a fully-connected feed-forward neural network, the number of weighted connections or network parameters that are trained is $O(n^2)$, where n is the number of hidden units per layer.

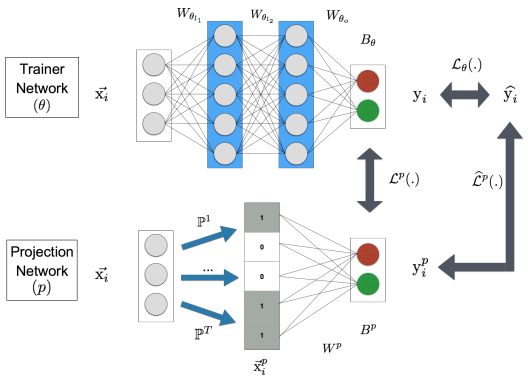

Figure 1. Projection Network trained using feed-forward trainer.

We propose a new projection approach and joint optimization framework for training compact on-device models for inference. The architecture uses a *trainer* network coupled with a *projection* network and trains them jointly. Figure 1 illustrates the Neural Projection Network architecture using a feed-forward NN for the trainer network. The coupled networks are jointly trained to optimize a combined loss

function:

$$\mathcal{L}(\theta, p) = \lambda_1 \cdot \mathcal{L}_\theta(.) + \lambda_2 \cdot \mathcal{L}^p(.) + \lambda_3 \cdot \widehat{\mathcal{L}}^p(.) \quad (1)$$

where $\mathcal{L}_\theta(.)$, $\mathcal{L}^p(.)$ and $\widehat{\mathcal{L}}^p(.)$ are the loss functions corresponding to the two networks as defined below.

$$\begin{aligned}\mathcal{L}_\theta(.) &= \sum_{i \in N} \mathcal{D}(h_\theta(\vec{x}_i), \widehat{y}_i) \\ \mathcal{L}^p(.) &= \sum_{i \in N} \mathcal{D}(h^p(\vec{x}_i), h_\theta(\vec{x}_i)) \\ \widehat{\mathcal{L}}^p(.) &= \sum_{i \in N} \mathcal{D}(h^p(\vec{x}_i), \widehat{y}_i) \end{aligned} \quad (2)$$

N indicates the number of training instances in the dataset, $\vec{x}_i$ represents the input feature vector in a feed-forward network or sequence input in an RNN, and $\widehat{y}_i$ refers to the ground-truth output classes used for network training. $h_\theta(\vec{x}_i)$ represents a parameterized representation of the hidden units in the *trainer* network that transforms $\vec{x}_i$ to an output prediction y_i. Similarly, $h^p(\vec{x}_i)$ represents the *projection* network parameters that transforms the input to corresponding predictions y^p_i. We apply softmax activation at the last layer of both networks to compute the predictions y_i and y^p_i.

Decomposable Loss Function: $\mathcal{D}$ denotes a distance function that measures the prediction error used in the loss functions. This is decomposed into three parts—trainer prediction error, projection simulation error and projection prediction error. Reducing the first leads to a better trainer network and decreasing the latter in turn learns a better projection network that is simpler but with approximately equivalent predictive capacity. In practice, we use cross-entropy for $\mathcal{D}(.)$ in all our experiments but this is configurable depending on the task and can be replaced with suitable choices (e.g., $\|\cdot\|_2$, KL divergence, etc.).

Optimization Choices: For the *projection* $\mathcal{L}^p$ in Equation 2, we follow a distillation-style approach (Hinton et al., 2015) to optimize $\mathcal{D}(.)$ since it has been shown to yield better generalization ability than a model trained on just the labels $\widehat{y}_i$. λ_1, λ_2 and λ_3 are hyperparameters that affect the trade-off between these different types of errors. These are tuned on a small heldout development set and in our experiments, we set them to $\lambda_1 = 1.0, \lambda_2 = 0.1, \lambda_3 = 1.0$. In some settings, it is beneficial to fix the teacher and train only the smaller network. Setting both λ_1 and λ_2 to 0 will train the small projection network from scratch. When there is sufficiently large training data, we observe that pre-training and fixing the teacher ($\lambda_1 = 0$) produces slight improvement in performance for the smaller model as well as faster convergence. We also observe that joint training improves teacher network when training data is limited and enables multi-sized training, i.e., two projection networks with different sizes.

Trainer Network (θ). The *trainer* model is a full neural network (feed-forward, RNN or CNN) whose choice is flexible and depends on the task. Figure 1 shows a trainer using feed-forward network but this can be swapped with CNNs (Section 2.3) or LSTM RNNs (Section 3.2) or other deep neural networks. For the network shown in the figure, the activations for $h_\theta(.)$ in layer l_{k+1} is computed as follows:

$$A_{\theta_{l_{k+1}}} = \sigma(W_{\theta_{l_{k+1}}} \cdot A_{\theta_{l_k}} + B_{\theta_{l_{k+1}}}) \quad (3)$$

where σ is the ReLU activation function (Nair & Hinton, 2010) applied at each layer except the last and A indicates the computed activation values for hidden units.

The number of weights/bias parameters W_θ, B_θ in this network can be arbitrarily large since this will only be used during training which can be effectively done using high-performance distributed computing with CPUs or GPUs.

Projection Network (p). The *projection* model is a simple network that encodes a set of efficient-to-compute operations which will be performed directly on device for inference. The model itself defines a set of efficient "projection" functions $\mathbb{P}(\vec{x}_i)$ that project each input instance $\vec{x}_i$ to a different space $\Omega_\mathbb{P}$ and then performs learning in this space to map it to corresponding outputs y^p_i. Figure 1 illustrates a simple *projection* network with few operations. The inputs $\vec{x}_i$ are transformed using a series of T projection functions $\mathbb{P}^1, ..., \mathbb{P}^T$, followed by a single layer of activations.

$$\vec{x}^p_i = [\, \mathbb{P}^1(\vec{x}_i), ..., \mathbb{P}^T(\vec{x}_i)\,];\ \vec{x}^p_i \in \{0, 1\}^{T \cdot d} \quad (4)$$

$$y^p_i = \text{softmax}(W^p \cdot \vec{x}^p_i + B^p) \quad (5)$$

The projection transformations use pre-computed parameterized functions, i.e., they are not trained during the learning process, and their outputs are concatenated to form the hidden units for subsequent operations. During training, the simpler *projection* network learns to choose and apply specific projection operations $\mathbb{P}^j$ (via activations) that are more predictive for a given task. It is possible to stack additional layers connected to the bit-layer in this network to achieve non-linear combinations of projections. Note that projections can also be coupled with other operations like convolutions, as we show later (Sections 2.3, 3.1). It is also possible to make the projection parameters trainable and add it to any layer in the network by replacing the binary functions discussed below with approximate, differentiable equivalents but we leave this for future work.

The *projection* model is jointly trained with the *trainer* and learns to mimic predictions made by the full trainer network which has far more parameters and hence more predictive capacity. Once learning is completed, the transform functions $\mathbb{P}(.)$ and corresponding trained weights W^p, B^p from the projection network are extracted to create a lightweight model that is pushed to device. At inference

time, the lightweight model and corresponding operations is applied to a given input $\vec{x}_i$ to generate predictions y_i^p.

The choice of the type of projection matrix $\mathbb{P}$ as well as representation of the projected space $\Omega_{\mathbb{P}}$ in our setup has a direct effect on the computation cost and model size. We propose to leverage an efficient randomized projection method using a modified version of locality sensitive hashing (LSH) to define $\mathbb{P}(.)$. In conjunction, we use a bit representation $\{0,1\}^d$ for $\Omega_{\mathbb{P}}$, i.e., the network's hidden units themselves are represented using projected d-bit vectors. This yields a drastically lower memory footprint compared to the full network both in terms of number and size of parameters. We highlight a few key properties of this approach below:

- There is no requirement for committing to a preset vocabulary or feature space unlike typical methods. For example, LSTM RNN models typically apply pruning and use smaller, fixed-size vocabularies in the input encoding step to reduce model complexity.
- The proposed learning method scales efficiently to large data sizes and high dimensional spaces. This is especially useful for natural language applications involving sparse high dimensional feature spaces. For dense feature spaces (e.g., image pixels), existing operations like fully-connected layers (or even convolutions) can be efficiently approximated for prediction without relying on a large number of parameters. Such operations can also be applied in conjunction with the projection functions to yield more complex *projection* networks while constraining the memory requirements.
- Computation of $\mathbb{P}(x_i)$ is independent of the training data size.
- We ensure that $\mathbb{P}(.)$ is efficient to compute on-the-fly for inference on device.

Next, we describe the projection method and associated operations in more detail.

2.2. Locality Sensitive Projection Network

The *projection* network described earlier relies on a set of transformation functions $\mathbb{P}$ that project the input $\vec{x}_i$ into hidden unit representations $\Omega_{\mathbb{P}}$. The projection operations outlined in Equation 4 can be performed using different types of functions. One possibility is to use feature embedding matrices pre-trained using word2vec (Mikolov et al., 2013) or similar techniques and model $\mathbb{P}$ as a embedding lookup for features in $\vec{x}_i$ followed by an aggregation operation such as vector averaging. However, this requires storing the embedding matrices which incurs additional memory complexity.

Instead, we employ an efficient randomized projection method for this step. We use locality sensitive hashing (LSH) (Charikar, 2002) to model the underlying projection operations. LSH is typically used as a dimensionality reduction technique for applications like clustering (Manning et al., 2008). Our motivation for using LSH within Projection Nets is that it allows us to project similar inputs $\vec{x}_i$ or intermediate network layers into hidden unit vectors that are nearby in metric space. This allows us to transform the inputs and learn an efficient and compact network representation that is only dependent on the inherent dimensionality (i.e., observed features) of the data rather than the number of instances or the dimensionality of the actual data vector (i.e., overall feature or vocabulary size). We achieve this with binary hash functions (Charikar, 2002) for $\mathbb{P}$.

Theorem 1 *For $\vec{x}_i, \vec{x}_j \in \mathbb{R}^n$ and vectors $\mathbb{P}_k$ drawn from a spherically symmetric distribution on $\mathbb{R}^n$ the relation between signs of inner products and the angle $\measuredangle(\vec{x}_i, \vec{x}_j)$ between vectors can be expressed as follows:*

$$\measuredangle(\vec{x}_i, \vec{x}_j) = \pi \ \Pr\{sgn[\langle \vec{x}_i, \mathbb{P}_k \rangle] \neq sgn[\langle \vec{x}_j, \mathbb{P}_k \rangle]\} \quad (6)$$

This property holds from simple geometry (Charikar, 2002), i.e., whenever a row vector from the projection matrix $\mathbb{P}$ falls inside the angle between the unit vectors in the directions of $\vec{x}_i$ and $\vec{x}_j$, they will result in opposite signs. Any projection vector that is orthogonal to the plane containing $\vec{x}_i \vec{x}_j$ will not have an effect. Since inner products can be used to determine parameter representations that are nearby, $\langle \vec{x}_i, \vec{x}_j \rangle = ||\vec{x}_i|| \cdot ||\vec{x}_j|| \cdot cos\measuredangle(\vec{x}_i, \vec{x}_j)$, therefore we can model and store the network hidden activation unit vectors in an efficient manner by using the signature of a vector in terms of its signs.

Computing Projections. Following the above property, we use binary hashing repeatedly and apply the projection vectors in $\mathbb{P}$ to transform the input $\vec{x}_i$ to a binary hash representation denoted by $\mathbb{P}_k(\vec{x}_i) \in \{0,1\}$, where $[\mathbb{P}_k(\vec{x}_i)] := sgn[\langle \vec{x}_i, \mathbb{P}_k \rangle]$. This results in a d-bit vector representation, one bit corresponding to each projection row $\mathbb{P}_{k=1 \ldots d}$.

The projection matrix $\mathbb{P}$ is fixed prior to training and inference. Note that we never need to explicitly store the random projection vector $\mathbb{P}_k$ since we can compute them on the fly using hash functions over feature indices with a fixed row seed rather than invoking a random number generator. In addition, this also permits us to perform projection operations that are linear in the *observed* feature size rather than the *overall* feature size which can be prohibitively large for high-dimensional data, thereby saving both memory and computation cost. In other words, the projection network can efficiently model high-dimensional sparse inputs and large vocabulary sizes common for text applications (Section 3.2) instead of relying on feature pruning or other

pre-processing heuristics employed to restrict input sizes in standard neural networks for feasible training. The binary representation is significant since this results in a significantly compact representation for the *projection* network parameters that in turn reduces the model size considerably compared to the *trainer* network.

Note that other techniques like quantization or weight sharing (Courbariaux et al., 2014) can be stacked on top of this method to provide small further gains in terms of memory reduction as we show in Section 3.2.

Projection Parameters. In practice, we employ T different projection functions $\mathbb{P}^{j=1...T}$ as shown in Figure 1, each resulting in a d-bit vector that is concatenated to form the projected activation units $\vec{x}_i^p$ in Equation 4. T and d vary depending on the *projection* network parameter configuration specified for $\mathbb{P}$ and can be tuned to trade-off between prediction quality and model size. Note that the choice of whether to use a single projection matrix of size $T \cdot d$ or T separate matrices of d columns depend on the type of projection employed (dense or sparse), described in Section 2.4.

2.3. ProjectionCNN and Deeper Projection Networks

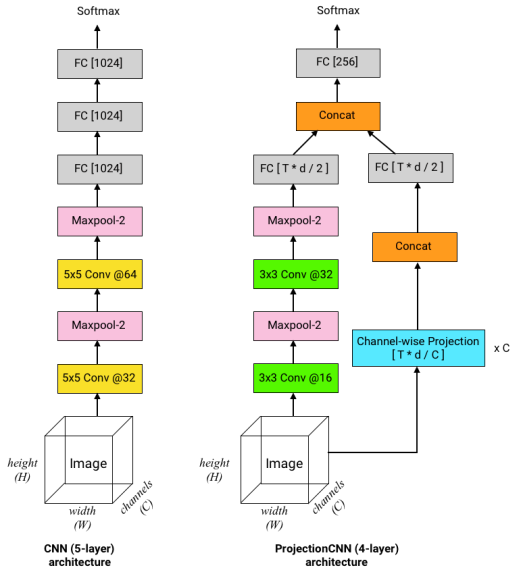

Figure 2. Model architectures for image classification. Left: Baseline CNN (5-layer) model. Right: ProjectionCNN (4-layer) model using channel-wise projections. Depth of the network and projection parameters ($T = 64, d = 8$) can be configured depending on task. Each Conv, Projection and FC layer is followed by batchnorm and ReLU in both models.

For complex tasks, we design deeper projection network variants by combining projections stacked with other non-linear operations. We also extend the proposed framework to handle more than one type of *trainer* or *projection* network and even simultaneously train several models at multiple resolutions using this architecture. Figure 2 illustrates our design for a **ProjectionCNN** neural network architecture for image classification and compares it to a standard convolution model. The new model is constructed by applying channel-wise projections combined with simpler, faster convolutions (3x3 with fewer filters) and other operations to yield compact yet powerful projection networks.

2.4. Training and Inference

We use the compact bit units to represent the *projection* network as described earlier. During training, this network learns to move the gradients for points that are nearby to each other in the projected bit space $\Omega_{\mathbb{P}}$ in the same direction. The direction and magnitude of the gradient is determined by the *trainer* network which has access to a larger set of parameters and more complex architecture. The two networks are trained jointly using backpropagation. Despite the joint optimization objective, training can progress efficiently with stochastic gradient descent with distributed computing on high-performance CPUs or GPUs.

Once trained, the two networks are de-coupled and serve different purposes. The *trainer* model can be deployed anywhere a standard neural network is used. The simpler *projection* network model weights along with transform functions $\mathbb{P}(.)$ are extracted to create a lightweight model that is pushed to device. This model is used directly "on" device at inference time by applying the same operations in Equations 4, 5 (details described in Sections 2.1, 2.2) to a new input $\vec{x}_i$ and generate predictions y_i^p.

Complexity. The time complexity for projection during inference is $O(n \cdot T \cdot d)$, where n is the *observed* feature size (*not* overall vocabulary size) which is linear in input size, d is the number of LSH bits specified for each projection vector $\mathbb{P}_k$, and T is the number of projection functions used in $\mathbb{P}$. The model size (in terms of number of parameters) and memory storage required for the *projection* inference step is $O(T \cdot d \cdot C)$, where C is the number of outputs (e.g., classes) or hidden units in the next layer in a multi-layer projection network.

As an alternative to the bit vector representation $\Omega_{\mathbb{P}}$, the projection matrix $\mathbb{P}$ can instead be used to generate a sparse representation of hidden units in the *projection* network. Each d-bit block can be encoded as an integer instead of a bit vector. This results in a larger parameter space overall $O(T \cdot 2^d)$ but can still be beneficial to applications where the actual number of learned parameters is tiny and inference can be performed via efficient sparse lookup operations.

3. Experiments

In this section we demonstrate the effectiveness of the proposed approach with several experiments on different bench-

mark datasets and classification tasks involving visual recognition and language understanding. The experiments are run in TensorFlow (Abadi et al., 2015).

Evaluation. For each task, we compute the performance of each model in terms of precision@1, i.e., accuracy % of the top predicted output class. Models were trained using multiple runs, each experiment was run for a fixed number of (400k) time steps with a batch size of 200 for the visual tasks and 100 for the text classification task. The observed variance across runs wrt accuracy was small, around $\pm 0.1\%$.

We also compute the *Compression Ratio* achieved by various models, i.e., ratio of # parameters in the baseline deep network compared to the proposed model. The model size ratios reported here are based on number of free parameters and not wrt actual model size stored on disk.

3.1. Visual Classification from Images

Tasks. We apply our neural projection approach and compare against baseline systems on benchmark image classification datasets—MNIST, Fashion-MNIST (Xiao et al., 2017) and CIFAR-10.

We first compare the performance of different approaches using the original MNIST handwritten digit dataset (MNIST). The dataset contains 60k instances for training and 10k instances for testing. We hold out 5k instances from the training split as dev set for tuning system parameters. Fashion-MNIST is a recent real-world dataset with similar grayscale style images and splits as MNIST but it is a much harder task. CIFAR-10 dataset contains colour images with 50k for training, 10k for testing.

Baselines and Method. We compare projection neural networks (ProjectionNet, ProjectionCNN) at different model sizes with full-sized deep neural network counterparts for each task. The deep network architecture varies depending on the task type. For MNIST, we use a feed-forward NN architecture (3 layers, 1000 hidden units per layer) with L2-regularization as one of the baselines and *trainer* network for ProjectionNet. For Fashion-MNIST and CIFAR-10 tasks, we employ deeper convolution baselines and ProjectionCNN networks.

Results. Table 1 shows results of the baseline and comparison to ProjectionNet models with varying sizes (T, d). The results demonstrate that a small ProjectionNet with 13x fewer parameters comes close with 97.1% accuracy whereas a tiny ProjectionNet with a remarkably high compression ratio of 388x is able to achieve a high accuracy of 92.3% compared to 98.9% for the baseline that is significantly larger in memory footprint. Moreover, ProjectionNet models are able to achieve even further reduction in model size (upto 2000x-3500x) while yielding around 70-80% precision for top-1 and 90-94% precision among top-3 predictions.

Going deeper with projections: Furthermore, going deeper with the projection network architecture (i.e., adding more layers) improves prediction performance even though it adds more parameters, overall size still remains significantly small compared to the baseline model. For example, a single-layer ProjectionNet with $T = 60, d = 10$ produces an accuracy of 91.1% whereas a 2-layer deep ProjectionNet with the same projection layer followed by a fully connected layer FC (128 hidden units) improves the accuracy considerably to 96.3%, yielding a gain of +5.2%. A stacked ProjectionNet with slightly more parameters further improves the accuracy to 97.1%, very close to the baseline performance at a 13x size reduction.

Combining convolutions with projections: We also trained a much larger baseline network with deep convolution layers (CNN). We used this network to learn a much smaller model using a new *ProjectionCNN* architecture, a convolutional variant of the projection model (Figure 2). The new model is constructed by applying convolution and projection operations in parallel to capture spatial and similarity information followed by fusing the corresponding outputs at each layer. This enables the network to jointly learn and combine convolutions and projections especially at the higher layers with non-linear activations. In Table 1, we show that ProjectionCNN yields a more powerful architecture, in fact it outperforms the NN baseline (99.4% vs 98.9%) at 4x model size reduction. Moreover it comes very close to the much bigger CNN baseline (99.6%) while yielding a 8x reduction in parameters. ProjectionCNN also outperforms previous best results from several other compression techniques (Chen et al., 2015) on the same dataset (0.6% error vs 1.22-2.19%).

Different training objectives: The training objective can be varied by removing specific loss components from Equation 1. We observe that using the joint architecture helps significantly and results in the best performance, especially for tiny models. The ProjectionNet $[T = 60, d = 12]$ from Table 1 achieves 92.3%. The same ProjectionNet trained *without* the full joint objective does worse, ∼91% when trained using $\widehat{\mathcal{L}}^p$ alone or using only L_θ + L^p. We also trained smaller baseline neural networks with fewer layers and parameters for comparison and observed that ProjectionNets achieve far higher compression ratios at similar performance levels. The same trend follows when comparing against other simpler regularized linear baseline methods which perform far worse and produce accuracies comparable or worse than projection networks trained in isolation without the joint architecture. We notice that on more complex problems involving large output space, pre-training the trainer network and then performing joint training helps the projection net converge faster to a good performance.

Comparison with other compression baselines: In addi-

Table 1. Classification Results (precision@1) for vision tasks using Neural Projection Nets and baselines.

Model		Compression Ratio *(wrt baseline)*	MNIST	Fashion MNIST	CIFAR-10
NN (3-layer)	*(Baseline: feed-forward)*	1	98.9	89.3	-
CNN (5-layer)	*(Baseline: convolutional)* (Figure 2, Left)	0.52*	99.6	93.1	83.7
Random Edge Removal	(Ciresan et al., 2011)	8	97.8	-	-
Low Rank Decomposition	(Denil et al., 2013)	8	98.1	-	-
Compressed NN (3-layer)	(Chen et al., 2015)	8	98.3	-	-
Compressed NN (5-layer)	(Chen et al., 2015)	8	98.7	-	-
Dark Knowledge	(Hinton et al., 2015; Ba & Caruana, 2014)	-	98.3	-	-
HashNet (best)	(Chen et al., 2015)	8	98.6	-	-
NASNet-A	*(7 cells, 400k steps)* *(Zoph et al., 2018)*	-	-	-	90.5
ProjectionNet	**(our approach)** Joint *(trainer = NN)*				
	[T=8,d=10]	3453	70.6		
	[T=10,d=12]	2312	76.9		
	[T=60,d=10]	466	91.1		
	[T=60,d=12]	388	92.3		
	[T=60,d=10] + FC [128]	36	96.3		
	[T=60,d=12] + FC [256]	15	96.9		
	[T=70,d=12] + FC [256]	13	97.1	86.6	
ProjectionCNN (4-layer)	**(our approach)** (Figure 2, Right) Joint *(trainer = CNN)*	8	99.4	92.7	78.4
ProjectionCNN (6-layer)	**(our approach)** *(Conv3-64, Conv3-128, Conv3-256, $\mathbb{P}$ [T=60, d=7], FC [128 x 256])*				
	Self *(trainer = None)*	4			82.3
	Joint *(trainer = NASNet)*	4			84.7

tion to the standard baseline models described above, we compare our approach against other strong compression baselines from the literature that use different techniques like weight sharing for compression (Chen et al., 2015). They use a similar baseline *(3-layer NN with 1k units per layer)* as the one listed in Table 1 (row 1). ProjectionCNN also outperforms previous best results from several other compression techniques (Chen et al., 2015) on the same dataset—0.6% error versus 1.22-2.19% achieved by their best-performing methods. Table 1 shows a detailed comparison of our approach against existing compression techniques on the same task.

Also, for tasks like semantic text classification (described in Section 3.2) involving LSTMs, even smaller neural network models require keeping vocabulary matrices $O(V \cdot d)$ with tens of thousands of words/phrases and >100 dimensions per row ($V \gg n$ by 1000x in Section 2.4).

Relation to distillation and similar approaches: In Table 1, we also compare our method with other teacher-student (T-S) training approaches like distillation (Hinton et al., 2015) which build on top of earlier work (Ba & Caruana, 2014) and show that matching logits (soft targets) is a special case of distillation. Section 2.1 (Equation 2) describes how we follow a distillation-style approach to optimize the joint loss component $\mathcal{D}(.)$ but the student network is modeled using a novel projection architecture. More importantly, we compared our method against optimized variants of distilled models from the literature that were trained in T-S setup (*Dark Knowledge* in Table 1) where the student uses other compression techniques (baseline details in (Chen et al., 2015)) and our ProjectionCNN approach achieves the best results in terms of accuracy %.

Finally, we note that quantization (low-precision computing) and related techniques provide orthogonal benefits and can be combined with our method to achieve further optimization (i.e., to compress weight values in addition to reducing #parameters). For example, we can get an additional 4x size-reduction by using 8-bit integers instead of floating-point to store weight values as shown in Section 3.2.

More complex image tasks and huge architectures: We also performed experiments with larger convolutional nets for more complex image tasks. We observe the same trends using neural projection approach on Fashion-MNIST task. The task is much harder where the baseline NN model yields only 89.3% (compared to 98.9% on MNIST) and CNN (5-layer) model gets 93.1% (vs. 99.6% on MNIST). Our ProjectionCNN network achieves 92.7% significantly outperforming the NN baseline and very competitive with the deep CNN model but with the benefit of 8x compression.

On CIFAR-10 image classification task, a 5-layer CNN model yields 83.7% accuracy. Even though it is an unfair comparison, we also include a more complex deep convolutional network, NASNet (Zoph et al., 2018), that was designed using large-scale architecture search using 1000s of GPU hours and optimized on the same dataset, produces 90.5% after 400k training steps. In contrast, a much leaner and significantly faster ProjectionCNN network achieves 82.3% when trained from scratch by itself which improves to 84.7% with joint training and outperforms the baseline at 4x size-reduction. Projection architectures can yield similar benefits for other complex neural networks like Inception and ResNet variants.

3.2. Semantic Intent Classification from Text

Next, we compare the performance of our neural projection approach against baselines on text classification tasks.

Tasks. We evaluate our models and report precision@1 on multiple text classification tasks.

- **SmartReply Intent** is a real-world semantic intent classification task for automatically generating short email responses (Kannan et al., 2016). The goal is to discover and map short response messages to semantic intents. We use the same task and dataset as prior works (Bui et al., 2018)[1] with 20 intent classes, 5483 samples (3832 for training, 560 for validation and 1091 for testing). Each sample corresponds to a short response message text paired with a semantic intent category that was manually verified by human annotators. For example, *"That sounds awesome!"* and *"Sounds fabulous"* are labeled as *sounds_good* intent.

- **ATIS** is a benchmark corpus used in the speech and dialog community (Tur et al., 2010) for understanding different types of intents (18 classes) expressed during flight reservations.

Baselines and Method. We use an RNN sequence model with multilayer LSTM architecture (2 layers, 100 dimensions) as the baseline for the *Smart Reply Intent* task. For this task, we compare against previous works (Bui et al., 2018; Kannan et al., 2016) that use the same baselines as well as Smart Reply model (LSTM) used in real-world applications. The only difference is (Kannan et al., 2016) uses LSTM for end-to-end generation and response ranking whereas our task is classification into response intents. Other systems include—*Random* baseline ranks the intent categories randomly and *Frequency* baseline ranks them in order of their frequency in the training corpus. For the *ATIS* task, we compare our approach with a recent attention-based RNN model (Liu & Lane, 2016). We use unigram and bigram text features in our approach to build projection models. For these language tasks, we observed that projection networks achieved high performance even without a trainer model, so we set $\lambda_1 = \lambda_2 = 0$ during training.

Results. We show in Table 2 that ProjectionNet achieves very high precision@1, significantly outperforming baseline systems. On both language tasks, the model even outperforms RNNs but with significant reduction in memory footprint (*compression ratio* > 10) and computation compared to LSTM unrolling steps.

[1]For details regarding SmartReply and how the semantic intent clusters are generated, refer (Kannan et al., 2016).

Table 2. Classification Results (precision@1) for language tasks using Neural Projection Nets and baselines.

Model	**Compression** *(wrt RNN)*	**Smart Reply Intent**	**ATIS**
Random (Kannan et al., 2016)	-	5.2	-
Frequency (Kannan et al., 2016)	-	9.2	72.2
LSTM (Kannan et al., 2016)	1	96.8	-
Attention RNN (Liu & Lane, 2016)	1	-	91.1
ProjectionNet **(our approach)** *[T=70,d=14]→FC [256 x 128]*	>10	**97.7**	**91.3**

Quantized Projection Network. We also learn mobile-optimized versions of ProjectionNet models that can run inference on TensorFlow Lite (TFLite) open-source library. We apply quantized training with 8-bit operations similar to (Jacob et al., 2017) to learn a compressed ProjectionNet with an additional 4x reduction in model size and improved latency. On ATIS, quantized ProjectionNet reduces the size from 1.1M to a tiny 285KB footprint and still yields 91.0%. We also measured the average computation latency for this model—on a Pixel phone, it requires < 5 milliseconds.

4. Beyond Neural Projections

A few possible future extensions to the framework are discussed at the end of Section 2. Recent works have also demonstrated the effectiveness of tailoring projection-based approaches to solve other natural language tasks (Ravi & Kozareva, 2018; Sankar et al., 2019).

Going beyond deep learning, we extend this framework to train lightweight models in *semi-supervised* or *unsupervised* learning scenarios and **ProjectionGraphs** with structured loss functions defined using a graph or probabilistic graphical model instead of a deep network.

The proposed projection-based learning architectures have been used to power **on-device conversational models** (Ravi, 2017) for real-world applications such as Smart Reply (Kannan et al., 2016) on smartwatches and mobile devices.

5. Conclusion

We introduced a new Neural Projection approach to train lightweight neural network models for performing efficient inference on device at low computation and memory cost. We demonstrated the flexibility of this approach to variations in model sizes and deep network architectures. Experimental results on visual and language classification tasks show the effectiveness of this method in achieving significant model size reductions and efficient inference while providing competitive performance. The projection-based machine learning models have already been applied to and proven useful for powering real-world on-device applications such as smart messaging.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

AndroidWear. Android Wear 2.0: Make the most of every minute. https://blog.google/products/android-wear/android-wear-20-make-most-every-minute/.

Ba, L. J. and Caruana, R. Do deep nets really need to be deep? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, pp. 2654–2662, 2014.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

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.

Charikar, M. S. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, pp. 380–388, 2002. URL http://doi.acm.org/10.1145/509907.509965.

Chen, W., Wilson, J. T., Tyree, S., Weinberger, K. Q., and Chen, Y. Compressing neural networks with the hashing trick. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, volume 37, pp. 2285–2294. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045361.

Chun, B.-G. and Maniatis, P. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS, pp. 8–8, 2009. URL http://dl.acm.org/citation.cfm?id=1855568.1855576.

Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. High-performance neural networks for visual object classification. *CoRR*, 2011. URL http://arxiv.org/abs/1102.0183.

Courbariaux, M. and Bengio, Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016. URL http://arxiv.org/abs/1602.02830.

Courbariaux, M., Bengio, Y., and David, J. Low precision arithmetic for deep learning. *CoRR*, abs/1412.7024, 2014. URL http://arxiv.org/abs/1412.7024.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and de Freitas, N. Predicting parameters in deep learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS)*, pp. 2148–2156, 2013. URL http://dl.acm.org/citation.cfm?id=2999792.2999852.

Ganchev, K. and Dredze, M. Small Statistical Models by Random Feature Mixing. In *Proceedings of the ACL-08: HLT Workshop on Mobile Language Processing*, 2008. URL http://clair.eecs.umich.edu/aan/paper.php?paper_id=W08-0804.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. D. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014. URL http://arxiv.org/abs/1412.6115.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative Adversarial Networks. *ArXiv e-prints*, 2014.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Hinton, G., Vinyals, O., and Dean, J. Distilling the Knowledge in a Neural Network. *ArXiv e-prints*, 2015.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A. G., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR*, abs/1712.05877, 2017. URL http://arxiv.org/abs/1712.05877.

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.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

Lee, K. H. and Verma, N. A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals. *IEEE Journal of Solid-State Circuits*, 48(7):1625–1637, 2013.

Liu, B. and Lane, I. Attention-based recurrent neural network models for joint intent detection and slot filling. *Proceedings of The 17th Annual Meeting of the International Speech Communication Association (INTERSPEECH)*, 2016.

Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL http://arxiv.org/abs/1301.3781.

MNIST. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pp. 807–814. Omnipress, 2010. URL http://www.icml2010.org/papers/432.pdf.

Ravi, S. On-device conversational modeling with tensorflow lite. https://ai.googleblog.com/2017/11/on-device-conversational-modeling-with.html, 2017.

Ravi, S. and Kozareva, Z. Self-governing neural networks for on-device short text classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 887–893, 2018. URL https://www.aclweb.org/anthology/D18-1105.

Sankar, C., Ravi, S., and Kozareva, Z. Transferable neural projection representations. In *Proceedings of the 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

Schuster, M. Speech recognition for mobile devices at google. In *Proceedings of the 11th Pacific Rim International Conference on Trends in Artificial Intelligence*, pp. 8–10, 2010. URL http://dl.acm.org/citation.cfm?id=1884293.1884297.

Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A., and Vishwanathan, S. Hash kernels for structured data. *J. Mach. Learn. Res.*, 10:2615–2637, 2009. URL http://dl.acm.org/citation.cfm?id=1577069.1755873.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, January 2014. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.

Stolcke, A. Entropy-based pruning of backoff language models. *CoRR*, cs.CL/0006025, 2000. URL http://arxiv.org/abs/cs.CL/0006025.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

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

Tur, G., Hakkani-Tur, D., and Heck, L. P. What is left to be understood in ATIS? In *Proceedings of 2010 IEEE Spoken Language Technology Workshop (SLT)*, pp. 19–24, 2010.

Wang, J., Liu, W., Kumar, S., and Chang, S. Learning to hash for indexing big data - A survey. *CoRR*, abs/1509.05472, 2015. URL http://arxiv.org/abs/1509.05472.

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1113–1120, 2009. URL http://doi.acm.org/10.1145/1553374.1553516.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *CVPR*, pp. 8697–8710. IEEE Computer Society, 2018.