Slice Resource Allocation with distributed Deep Neural Networks for 5G+ Networks

Ali Ehsanian [1] Thrasyvoulos Spyropoulos [2]

[1]EURECOM, Sophia Antipolis, France [2]Technical University of Crete, Chania, Greece

Funded by the European Union

Abstract

End-to-end network slicing is a new concept for 5G+ networks, dividing the network into slices for distinct services. A key task is satisfying service level agreements (SLA) by forecasting how many resources to allocate to each slice. The increasing complexity of the network services makes resource allocation a daunting task for traditional methods. Hence, data-driven methods have been recently explored. Although such methods excel at the application level, their application to wireless resource allocation is challenging. Not only are the latencies required significantly lower, but also the cost of transferring raw data across the network to be processed by a central Deep Neural Network (DNN) can be prohibitive. For this reason Distributed DNN (DDNN) architectures have been considered, where a subset of DNN layers are executed at edge, to improve speed and communication overhead; if it is deemed that a "good enough" allocation have produced locally, the additional latency and communication is avoided; if not, intermediate features produced at the edge are sent to cloud layers. We propose a distributed DNN architecture for this task based on LSTM that excels at forecasting demands with long-term dependencies. We investigate (i) joint training (offline) of the local and remote layers, and (ii) optimizing the (online) decision mechanism for offloading samples either locally or remotely. We show that our architecture resolves nearly *50%* of decisions at the edge, with no additional SLA penalty (compared to centralized models).

Introduction

- **DDNN**: We propose a distributed LSTM architecture for the problem of balancing under-/over-provisioning of resources to different slices, and investigate how the methodology of DDNNs can be applied to such larger, more sophisticated architectures. Such an architecture contains a few LSTM units and a "local exit" (i.e. a prediction layer) at the edge, and a larger number of units and "remote exit" at the central cloud.
- **Offline Optimization**: We demonstrate the impact of properly tuning the joint training hyperparameters of local and remote "exits" (i.e., predicted allocations and related SLA costs).to achieve a good balance between: (i) making the local layers powerful enough to correctly make a large number of allocation decisions, while (ii) producing useful enough features that the remote layers could leverage, when improved allocation decisions are deemed necessary.
- **Online Optimization**: We propose a mechanism that measures the confidence in the local exit and predicts whether the remote exit (which requires additional latency and communication,) would improve the SLA costs enough to justify the extra overhead (i.e., a form of unsupervised learning).
- **Objective Function**:

$$f(\hat{y}_t, d_t) = \begin{cases} c_1 \cdot (\hat{y}_t - d_t)^2 & \text{if } (\hat{y}_t - d_t) \leq 0 \\ c_2 \cdot (\hat{y}_t - d_t) & \text{if } (\hat{y}_t - d_t) > 0 \end{cases}$$

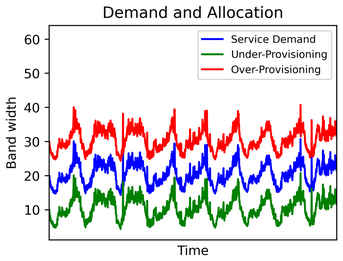

Figure 1. Over/Under-provisioning

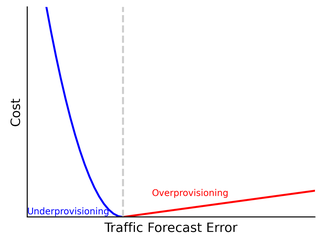

Figure 2. Objective Function

System Model

- Proposed DDNN Model:

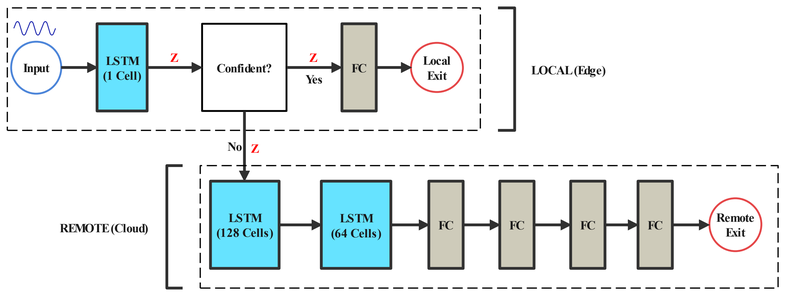

Figure 3. LSTM network is distributed over Edge and Cloud

- Local Exit: $\hat{y}^i_{L,t} = \mathcal{F}_L(\mathsf{d}^i_t; \boldsymbol{\theta}_L)$
- Remote Exit: $\hat{y}^i_{R,t} = \mathcal{F}_R(\mathsf{z}^i_t; \boldsymbol{\theta}_R)$
- Offline DDNN Training:

$$Loss_{DDNN} = \sum_{n=1}^{N} w_L \cdot f(\mathcal{F}_L(\mathsf{d}; \boldsymbol{\theta}_L), d_n) + w_R \cdot f(\mathcal{F}_R(\mathsf{z}; \boldsymbol{\theta}_R), d_n) = \sum_{n=1}^{N} w_L \cdot f(\hat{y}_{L,n}, d_n) + w_R \cdot f(\hat{y}_{R,n}, d_n)$$

  - backpropagating the local exit performance to the local layers (i.e., $\boldsymbol{\theta}_L$) to ensure that the majority of local decisions $\hat{y}_L$ can be relied on, despite being based only a small/simple DNN module.
  - backpropagating the remote exit performance to *both* the remote layers (i.e., $\boldsymbol{\theta}_R$) to ensure they can provide improved inferences $\hat{y}_R$, when needed. It also extends to the local layers (i.e., $\boldsymbol{\theta}_L$) to ensure that they produce sufficiently useful intermediate features (i.e., z) for the additional remote layers.
  - w_L is the "local weight", and w_R is the "remote weight", which they regulate the impact of the local and remote exit on the overall loss of the DDNN during joint training.
- Online Inference and Offloading:
  - **Oracle-based Offloading**: The "Oracle" knows the potential value of remote processing for any given sample. This oracle serves as our reference. The loss difference is:

$$\mathcal{L}_n = f(\hat{y}_{L,n}, d_n) - f(\hat{y}_{R,n}, d_n)$$

    If we did have such an oracle, we could certainly always keep the local decisions that are better than the remote ones.
  - **Bayesian Confidence-based Offloading**: In practice, we do not have such an oracle, as the remote decision $\hat{y}_{R,m}$ and related cost, can not be known at the edge. In this method, the confidence block, includes a dropout layer with dropout probability p, followed by a linear FC block. The intermediate signal z is given to the confidence block, and it is forced to infer for each input sample J times. We calculate the **Uncertainty** as follows:

$$U = \frac{1}{K} \sum_{k=1}^{K} \sigma_k$$

    The confidence mechanism compares the measured uncertainty (U) value with a given confidence threshold (η).

$$\mathcal{I}_n = \begin{cases} U <= \eta & \textit{the model is confident about the local decision} \\ U > \eta & \textit{send the data to the cloud} \end{cases}$$

  - **Data-Driven Optimized Offloading**: The idea is to train a function (in the offline) which receives d and z and operate like a binary classifier. While training the DDNN, we can label the signals as locally or remotely according to the Oracle. Then use this data to train a binary classifier.

$$\begin{cases} \mathcal{L}_n <= 0 & \textit{the model is confident about the local decision} \\ \mathcal{L}_n > 0 & \textit{send the data to the cloud} \end{cases}$$

    If we can train such a classifier, then during inference time making decisions will be faster, and its overhead is much less compared to the Bayesian offloading.

- DDNN cost function:

$$C_{DDNN} = \sum_{n=1}^{N} \mathcal{I}_n \cdot f(\hat{y}_{L,n}, d_n) + (1 - \mathcal{I}_n) \cdot f(\hat{y}_{R,n}, d_n)$$

$$\mathcal{I}_n = \begin{cases} 1 & \textit{if the sample n exited locally} \\ 0 & else \end{cases}$$

Simulation Results

- $c_1 = 50$
- $c_2 = 1$
- $K = 16$
- $N = 144$
- $J = 10$
- $p = 0.4$

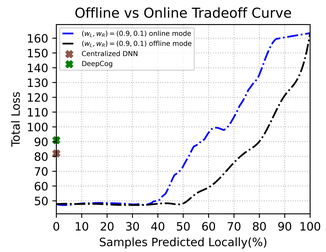

Figure 4. Offline vs Online trade-off curve

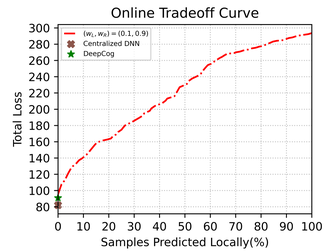

Figure 5. Online trade-off curve

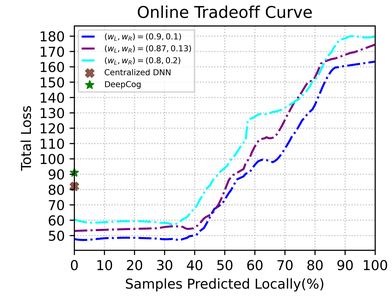

Figure 6. Online trade-off curve

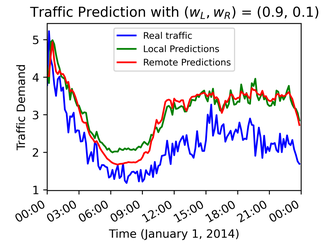

Figure 7. Traffic demand predictions

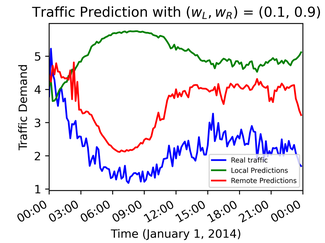

Figure 8. Traffic demand predictions

ali.ehsanian@eurecom.fr