HAL
open science

A Neural Few-Shot Text Classification Reality Check

Thomas Dopierre, Christophe Gravier, Wilfried Logerais

To cite this version:

Thomas Dopierre, Christophe Gravier, Wilfried Logerais. A Neural Few-Shot Text Classification Reality Check. 16th Conference of the European Chapter of the Association for Computational Linguistics, European Chapter of the Association for Computational Linguistics (EACL), Apr 2021, Kyiv (virtual), Ukraine. pp.935-943, 10.18653/v1/2021.eacl-main.79 . ujm-03267869

HAL Id: ujm-03267869

https://ujm.hal.science/ujm-03267869v1

Submitted on 22 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Neural Few-Shot Text Classification Reality Check

Thomas Dopierre[12], Christophe Gravier[1], Wilfried Logerais[2]

[1]Laboratoire Hubert Curien
UMR CNRS 5516
Université Jean Monnet
Saint-Étienne, France
firstname.lastname@
univ-st-etienne.fr

[2]Meetic
Paris, France
{t.dopierre,w.logerais}@
meetic-corp.com

Abstract

Modern classification models tend to struggle when the amount of annotated data is scarce. To overcome this issue, several neural few-shot classification models have emerged, yielding significant progress over time, both in Computer Vision and Natural Language Processing. In the latter, such models used to rely on fixed word embeddings before the advent of transformers. Additionally, some models used in Computer Vision are yet to be tested in NLP applications. In this paper, we compare all these models, first adapting those made in the field of image processing to NLP, and second providing them access to transformers. We then test these models equipped with the same transformer-based encoder on the intent detection task, known for having a large number of classes. Our results reveal that while methods perform almost equally on the ARSC dataset, this is not the case for the Intent Detection task, where the most recent and supposedly best competitors perform worse than older and simpler ones (while all are given access to transformers). We also show that a simple baseline is surprisingly strong. All the new developed models, as well as the evaluation framework, are made publicly available[1].

1 Introduction

Text classification often requires a large number of mappings between texts and target classes, so that it is challenging to build few-shot text classification models (Geng et al., 2019). With the recent advances of transformer-based models (Devlin et al., 2018; Wolf et al., 2019) along with their fine-tuning techniques (Sun et al., 2019), text classification has significantly improved. In few-shot settings, methods based on these extracted text representations have been historically made of semi-supervision, especially thanks to pseudo-labeling (Blum and Mitchell, 1998; Mihalcea, 2004; Zhi-Hua Zhou and Ming Li, 2005), which aims at propagating known labels to unlabeled data points in the representational space. Such methods depend on the number of collected unlabeled data, which can also be costly to obtain (Charoenphakdee et al., 2019), and also suffer from the infamous pipeline effect in NLP (Tenney et al., 2019), as cascade processing tends to make errors accumulate. In order to address the hindrance of collecting unlabeled data, modern approaches include unsupervised data augmentation techniques (Xie et al., 2019). It consists of generating samples through well-established text augmentation techniques in Neural Machine Translation, such as backtranslation (Sennrich et al., 2015; Edunov et al., 2018), and then use a consistency loss, training the classifier to assign the same prediction to all variations of the same sample text. While collecting new pseudo-labels can therefore be overcome by manipulating the dataset (especially using data augmentation techniques), the pipelining error accumulation effect instead calls for new neural architectures supporting scarcity of labeled data in an end-to-end fashion. Such end-to-end few-shot neural architectures for few-shot classification were discovered in image processing – it includes Matching Networks (Vinyals et al., 2016), Prototypical Networks (Snell et al., 2017) plus a follow-up known as Prototypical Networks++ (Ren et al., 2018), and Relation Networks (Sung et al., 2018). Ultimately Induction Networks (Geng et al., 2019) is a meta-learning based method dedicated to few-shot text classification, supposedly the state-of-the-art. Since our contribution considers this family of models, we will further detail them in Section 2. Nonetheless, it is important to stress that most of these neural architectures were originally devised to integrate image feature extractors. Despite both text and image relying on features extractors, a paragraph or sentence of few words hardly convey

[1]https://github.com/tdopierre/FewShotText

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pages 935–943
April 19 - 23, 2021. ©2021 Association for Computational Linguistics

as much information as a full-fledged three-canals 600×400 image ($720,000$ numerical values intrinsically). It is therefore of the utmost practical interest to validate and compare if what works best for end-to-end few-shot *image* classification is the same for end-to-end few-shot *text* classification. Moreover, when applying these end-to-end few-shot models to text, two main system components are into action: the text feature extractor itself and the downstream part of the neural network that provides a learning strategy over few shots. If we want to compare these systems, we need to plug the same feature extractor (hopefully the best one, that is transformer-based currently) into each end-to-end model. For the time being, the literature on end-to-end few-shot text classification compare aforementioned techniques using a different text extractor for each system, which is the one available when the technique was discovered – these text encoding varying greatly (Section 3.3). From that point-of-view, it is hardly possible to conclude if the improvement over time in few-shot text classification is due to new few-shot learning techniques or plainly to the significant advances made by text feature extractors. The same applies to vectors metrics: one method can use the cosine and another the euclidean distance, and that choice alone can impact conclusions made on the method being the state-of-the-art, although it could well rely only on the metric at work. Ultimately, experimental setups are usually restricted to one dataset, and evaluation schemes are heterogeneous among papers (Yu et al., 2018a),

- We revise different end-to-end neural architectures for few-shot text classification using the *same* transformer-based feature extractor,
- We investigate how these re-implemented state-of-the-art solutions compete with very simple baselines found to be yet very competitive for few-shot classification in the field of image-processing,
- We introduce an evaluation framework based on a number of intent detection datasets which is significantly bigger than what is usually used as evaluation in seminal papers transposing each of these architectures from image to text classification,
- The entire framework used in this paper, including all the re-implemented methods plugged with up-to-date transformers, is provided as an open-source repository for further research.

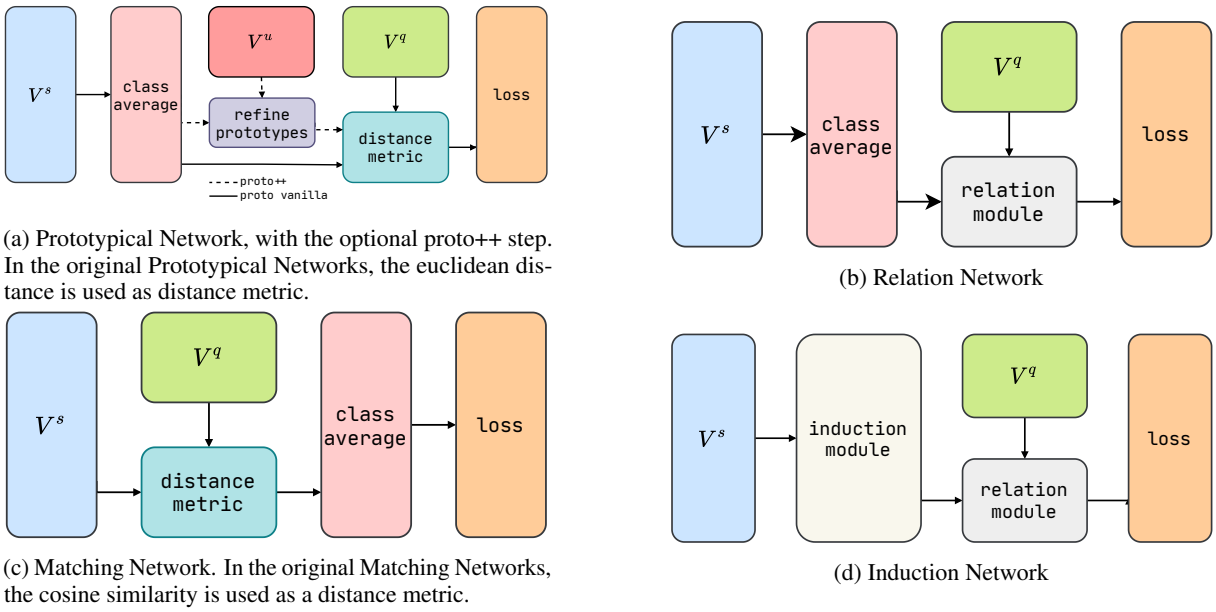
In a nutshell, we will demonstrate that providing a transformer-based encoder to a previously obsolete few-shot technique makes it the state-of-the-art again, that standard baselines are surprisingly strong, and that Induction Networks, while performing well for binary sentiment classification, struggles to perform correctly in the most common setups of few-shot text classification.

2 Few-Shot Classification Methods

In this section, we will describe the few-shot learning methods. In the following section, sentence vectors derived from the sentence encoder are denoted v. V^s, V^q, and V^u represent vectors for support, query, and unlabeled points, respectively. The number of shots is denoted K, and the number of classes per episode is denoted C. The kth support vector of class c is denoted $v^s_{c,k}$. In the equations, $s^q_{i,j}$ (resp. $s^q_{i,c}$) will denote the similarity between the ith query vector and the jth support vector (resp. the cth class). Similarly, $s^u_{i,j}$ represents the similarity between the ith unlabeled sample and the jth support vector. When needed, the number of unlabeled data is denoted U. For each method relying on a given similarity or distance metric, we devise two experiments, using either the cosine similarity or the euclidean distance. Those additional experiments are crucial, as they allow us to compare methods directly, without introducing a metric choice bias. Architectures of the different few-shot approaches are illustrated in Figure 1. They are each detailed in Section 2.2 and onwards, yet we first introduce the common building blocks among all methods in what follows.

2.1 Common building blocks

Class average All Matching, Prototypical, and Relation Networks contain a *class average* block. This step is used to directly compare a query point to a given class in order to make a prediction for this query point. In both Matching and Prototypical Networks, this step averages embeddings of support points for each class (they are class `prototypes`), which are then compared to query points to output class probabilities. In Matching Networks, this block averages similarity scores class-wise. On the contrary, in Induction Networks,



(a) Prototypical Network, with the optional proto++ step. In the original Prototypical Networks, the euclidean distance is used as distance metric.

(b) Relation Network

(c) Matching Network. In the original Matching Networks, the cosine similarity is used as a distance metric.

(d) Induction Network

Figure 1: Few-Shot classification methods and variants used in our experiments.

this step is lacking as support points are converted into prototypes using an Induction Layer, which aims at finding a better way to aggregate such knowledge than using the average (Section 2.5).

Loss Matching and Induction Networks both use the mean squared error (MSE) loss. Other methods use cross-entropy (CE). We implemented both losses on Matching and Induction networks, and it leads to very similar results – and sometimes, slightly better using CE. We therefore report results for all models using CE due to space limitations. Note that both losses are available in the publicly available source code. The cosine similarity being bounded, it would not make sense to directly apply such a loss on cosine similarities. To overcome this issue, we multiply the cosine similarities by a constant factor of 5, allowing them to reach more extreme values, hence ensuring that probabilities obtained by softmax are sparse enough.

2.2 Matching Networks

Introduced by Vinyals et al. (2016), Matching Networks (Figure 1c) rely on the comparison between query and support vectors using the cosine similarity in the seminal paper. After similarities between a query point and all support points are computed, they are averaged for each class. The predicted label for a given query point is the one with the highest average cosine similarity. In our notation framework, this process is summed up in Equation 1.

$$s_{i,c}^{q,\text{matching}} = K^{-1} \sum_{k=1}^{K} \frac{(v_i^q)^T \, v_{c,k}^s}{\|v_i^q\|_2 \|v_{c,k}^s\|_2} \tag{1}$$

2.3 Prototypical Networks

Prototypical Networks (Figure 1a) were introduced by Snell et al. (2017) as an extension of Matching Networks. After obtaining support vectors from the encoder, a class-wise average operation is done, as in Equation 2. This results in C prototypes denoted $\{p_c, c \in [\![1, C]\!]\}$, each one being the representative of a class. Then, a distance metric compares all query points to all prototypes. For each query point, the predicted class is the one for which this distance is the smallest. In the original Prototypical Networks, the euclidean distance was used, as in Equation 3. We also add the cosine similarity-based distance in our experiments in order to measure the impact of selecting another distance metric.

$$p_c = K^{-1} \sum_{k=1}^{K} v_{c,k}^s \tag{2}$$

$$s_{i,c}^{q,\text{proto}} = \frac{\exp\left(-\|v_i^q - p_c\|_2^2\right)}{\sum_{c'=1}^{C} \exp\left(-\|v_i^q - p_{c'}\|_2^2\right)} \tag{3}$$

An extension to Prototypical Networks was proposed by Ren et al. (2018), where unlabeled data points are used along with support and query points. After computing each class's prototype, a soft k-means technique is applied to further refine those

prototypes using unlabeled data points. The refined prototypes, denoted $\tilde{p}_c$, are derived using Equation 4. This additional step aims at correcting the support points selection bias and making the method more robust.

$$\tilde{p}_c = \frac{\sum_{k=1}^{K} v_{c,k}^{s} + \sum_{i=1}^{U} v_i^u s_{i,c}^{u,\text{proto}}}{K + \sum_i s_{i,c}^{u,\text{proto}}} \quad (4)$$

2.4 Relation Networks

Relation Networks (Sung et al., 2018) challenge the idea of using a pre-defined metric. The Relation Module takes as an input a query vector $v_i^q \in \mathbb{R}^d$, and the prototype of a class $p_c \in \mathbb{R}^d$, the latter being obtained the same way as in Prototypical Networks (Equation 2). The idea is to use a relation module, modeling the relationship between those two vectors, yielding a similarity score $s_{i,c} \in (0, 1)$. Instead of using a pre-defined distance metric like the euclidean or the cosine one, this approach allows such networks to learn this metric by themselves. Two different relation module architectures exist.

base The `base` relation module concatenates both v_i^q and p_c, and applies a small feed-forward neural network composed of two linear layers, with a ReLU activation function in between. The formula for this given relation module is described in Equation 5, where $C(\cdot, \cdot)$ denotes the concatenation operator, $f(\cdot)$ denotes the ReLU activation function, and w, M_1, M_2 are learnable parameters.

$$s_{i,c}^{q,\text{rel-base}} = \langle w, M_2 (f (M_1 (C (v_i^q, p_c)))) \rangle \quad (5)$$

NTL Introduced by Socher et al. (2013), the **N**eural **T**ensor **L**ayer relation module uses intermediate learnable matrices $M_k \in \mathbb{R}^{d,d}$ to model the relation between support vectors and prototypes. The similarity score for this relation module is obtained using Equation 6, where w is a learnable parameter. Following the work done by Geng et al. (2019), we fix the number h of intermediate matrices to 100 in all our experiments.

$$s_{i,c}^{q,\text{rel-ntl}} = \left\langle w, z_{i,c}^{\text{rel-ntl}} \right\rangle \quad , \quad w \in \mathbb{R}^h \quad (6)$$

$$z_{i,c,t}^{q,\text{rel-ntl}} = f\left((v_i^q)^T M_t p_c \right) \quad , \quad t \in [\![1, h]\!] \quad (7)$$

2.5 Induction Networks

Induction Networks (Geng et al., 2019) aims at finding a general representation of each class in the support set to compare to new queries. They are composed of both an induction module and a relation module. The main motivation for such networks is that representing the class by the average vector of its data points – what is done in Prototypical and Relation networks – is too restrictive. The first part, the induction module, leverages a dynamic routing (Sabour et al., 2017) algorithm. In their contribution, Geng et al. (2019) show that their method can better *induce* (hence their name) and generalize class-wise representations. For the second part, an NTL Relation Module is used: this is the same as the one introduced earlier (Section 2.4). Such networks are illustrated in Figure 1d.

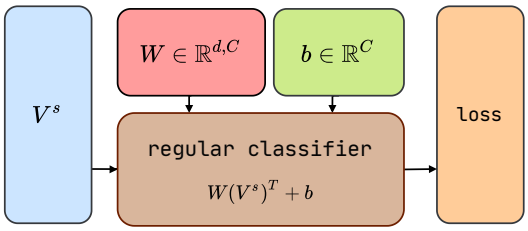
As in (Geng et al., 2019), we fix the number of routing iterations to 3, and the number of matrices in the NTL to 100.

2.6 Classifier Baselines

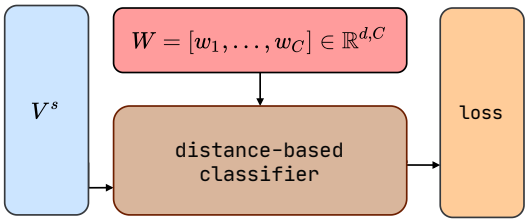
Few-shot learning algorithms are designed to overcome the data scarcity problem. With the tremendous shift in the architecture of sentence encoders using transformers, control baselines are needed to validate their ability to learn from few samples. For this reason, we include as a first `Baseline` model a traditional classifier, as described by Equation 8, added on top of BERT. Both W and b are learnable parameters, fine-tuned on the support vectors V^s. In our experiments, this method will henceforth be referred to as `Baseline`.

$$s_{i,j}^{q,\text{baseline}} = (W v_i^q + b)_j \quad ; \quad W \in \mathbb{R}^{C,d} \quad (8)$$

In addition to this `Baseline` model, we also implement a variant of it, which will henceforth be referred to as `Baseline++`. In that second baseline, the classifier design differs as follows: it measures similarities to a learnable vector instead of transforming vectors into logits using a linear layer. The matrix W used in the `Baseline` model can be writen as $[w_1, \ldots, w_C]$ where each $w_k \in \mathbb{R}^d$ is a weight vector corresponding to the kth class. To measure the similarity between class j and a query vector v_i^q, we compute the similarity scores in Equation 9. After all scores $s_{\cdot,\cdot}$ are computed, we then obtain a probability vector through



(a) Baseline Network.



(b) Baseline++ Network. The distance-base classifier can either use cosine or euclidean distance

Figure 2: Few-Shot classification baselines used in our experiments.

normalization using the softmax function).

$$s_{i,j}^{q,\text{baseline++}} = \frac{w_j^T v_i^q}{\|w_j\|_2 \|v_i^q\|_2} \tag{9}$$

As in Prototypical Networks, the derived vectors $[w_1, \ldots, w_C]$ can be interpreted as class prototypes. For both baselines, at each training episode, the weights W and b are initialized, and the whole model is fine-tuned for a few iterations using support samples. This is important in practice, as it teaches the sentence encoder – a transformer, see Section 3.3 – how to produce good enough embeddings for the downstream classifier to learn efficiently. At test time, the same process is used – using test labels –, except that we freeze the encoder's weights and only fine-tune the classifier part. The baselines architectures are represented in Figure 2.

3 Experimental Setup

3.1 Few-Shot Evaluation Setup

Introduced by Vinyals et al. (2016), few-shot classification corresponds to the case when a classifier must adapt to new classes, denoted here as $\mathcal{C}_{test}$, unseen during training, and only given a few labeled examples of these new classes. To this end, the approaches assume that during training, a task-significant set of classes noted $\mathcal{C}_{train}$ is available, along with an accordingly task-significant number of labeled data for each class $c_{train_i} \in \mathcal{C}_{train}$. For each training episode, C classes are sampled from $\mathcal{C}_{train}, C \ll |\mathcal{C}_{train}|$. Then, K support examples and Q query examples are randomly drawn for each of these classes. The model is then iteratively trained using both query and support points.

At testing time, the same sampling strategy is made, this time drawing classes among $\mathcal{C}_{test}$, with $\mathcal{C}_{test} \cap \mathcal{C}_{train} = \emptyset$. The model is then evaluated on its ability to predict labels for the Q query samples, using the K support samples (unless otherwise stated, C, Q, and K values are the same at both testing and training time).

This training procedure is called C-way K-shot classification. In all our experiments, we used $K = Q = 5$. Concerning the value of C, it is fixed to 2 for ARSC, as this dataset is already composed of binary classification tasks. Regarding the intent detection datasets we introduce later (Section 3.2), in order to see the shift between ARSC binary tasks and the more common 5-way evaluation (Geng et al., 2019; Ren et al., 2018), we measured performances of the different models with C ranging from 2 to 5.

3.2 Datasets

In this section, we describe the datasets used in our evaluation framework. The first one is a popular sentiment classification dataset, while the others are intent detection datasets. All datasets are public and in English.

ARSC The **A**mazon **R**eview **S**entiment **C**lassification dataset (Blitzer et al., 2007) is composed of product reviews from 27 product categories. Each review belongs to one of the 27 domains, and contains a grade ranging from 1 to 5 stars. The usual setup (Yu et al., 2018b; Geng et al., 2019) to evaluate few-shot classification with this dataset is as follows: for each of $p \leq 27$ product category and $2 \leq t \leq 5$ score thresholds, $\mathcal{E}_{ARSC} = p \times t$ binary classification evaluation tasks are created. In each of these $p \times 4$ tasks, a competitor model must learn to classify negative ($< t$) and positive ($\geq t$) reviews. To build our test tasks, we consider the same product categories as previous works (Yu et al., 2018b; Geng et al., 2019), which are *Books*, *DVD*, *Electronics*, *Kitchen*, and $t = 3$ (thresholds are picked in the $\{2, 4, 5\}$ set) – hence 12 binary classification test tasks in our benchmark for this dataset.

Each of these twelve evaluation tasks comes with a number of support test samples ($K = Q = 5$ as stated previously). Nonetheless, in (Yu et al.,

2018b) the same 5 samples per testing class are fixed for all experiments[2], which leads to a significant selection bias towards these 5 randomly selected samples used throughout the evaluation. In order to get more consistent results, we ran additional experimental runs, each of them selecting randomly new support samples. In the ARSC result table (table 1), this corresponds to the last column (`BERT + Sample shots`).

OOS The **O**ut **O**f **S**cope dataset[3] (Larson et al., 2019) is an intent detection dataset containing 150 equally-distributed classes. While initially used for out-of-scope prediction, it was also motivated by a high number of classes, a low number of examples per class (150), and its chatbot life-like style. In our experiments, we discard the out-of-scope class, keeping the remaining 150 classes to work with.

Liu Introduced by Liu et al. (2019), this intent detection dataset consists in 54 classes. This dataset was collected on the Amazon Mechanical Turk platform, where workers were given an intent and had to formulate queries for this intent with their own words. It is highly imbalanced: the most common class (`query`) holds 5,920 samples while the least common one (`volume_other`) 24 samples.

TREC28 TREC[4] is an open-domain fact-based dataset for question classification. We use the 50 labels version of the dataset but remove the labels which have less than 40 samples. This filtering process yields a dataset with 28 classes, ranging from 40 to 962 samples per class.

3.3 Sentence Encoder

In previous works comparing few-shot text classification methods, sentence encoders were not always the same. For example, Yu et al. (2018b) use a CNN on top of word embeddings, while Geng et al. (2019) use a Bi-LSTM. Those differences make the results hard to compare since they do not use the same method to convert sentences into vectors. In our experiments, in order to reduce this selection bias, and since it is now the state-of-the-art in many applications, we use a BERT (Devlin et al., 2018) encoder, using models from the Hugging Face (Wolf et al., 2019) team.

For each dataset, instead of using an off-the-shelf pre-trained model, we fine-tune it on the masked language modeling task, as it greatly improves the quality of embeddings (Sun et al., 2019; Xie et al., 2019). This fine-tuned transformer is then used as input for all few-shot models.

4 Observations

We report results for the ARSC dataset in Table 1, and results for the Intent Detection tasks in Table 3.

4.1 Baselines are surprisingly strong

Few-shot learning methods were originally used to overcome data scarcity. In those situations, training a classifier on top of a small dataset – in our case, 5 samples per class – can be hard. However, our experiments on ARSC show that the `Baseline` and `Baseline++`, plain and simple classifiers, get surprisingly close to state-of-the-art results. Table 2 provides four correct and four incorrect classification examples for the `Baseline` model.

While it fails to predict the correct text label for some shots, it is also able to correctly classify sentences such as *What do I take home ?* among the 50 test classes of the OOS dataset. On the ARSC dataset, it is also important to note that the `Baseline++` model is significantly better than the `Baseline`, and is even on par with all other architectures, except `Prototypical Networks`.

4.2 Sample selection bias

The mean accuracy difference between the last and the second columns of Table 1 accounts for the difference of randomly selecting new support samples at each iteration (last column) as opposed to picking the same fixed pool of support samples as done previously (second to last column). We can see that this difference alone is in the range of the increments brought by each model over time (baselines aside, bringing from 1 point up to 2.6 points for Prototypical Networks). This huge gap shows the importance of using evaluation tricks like cross-validation, instead of evaluating only for one run over a fixed set of shots.

4.3 Impact of switching to transformers

One of the main contributions of our paper is to compare few-shot learning methods with the lowest bias possible (see Section 3.3). On the ARSC dataset, using transformers drastically changes the

[2] See labeled sampled in https://github.com/Gorov/DiverseFewShot_Amazon

[3] https://github.com/clinc/oos-eval

[4] https://trec.nist.gov/data/qa.html

Model	Configuration		Mean binary accuracy		
	Metric	Relation module	Original encoder †	BERT as encoder (↗ or ↘ w.r.t. original encoder)	BERT + Sample shots
Matching Network (Vinyals et al., 2016)	euclid.	N/A	–	81.2	82.9
	cosine	N/A	65.7	81.9 (↗)	83.3
Prototypical Network (Snell et al., 2017)	euclid.	N/A	68.2	80.0 (↗)	82.6
	cosine	N/A	–	81.7	83.5
Proto++ (Ren et al., 2018)	euclid.	N/A	📷	82.4	**84.0**
	cosine	N/A	📷	**82.6**	83.6
Relation Network (Sung et al., 2018)	N/A	base	–	81.0	82.9
	N/A	ntl	83.1	81.7 (↘)	83.3
Induction Network (Geng et al., 2019)	N/A	ntl	85.6	79.3 (↘)	80.3
Baseline	N/A	N/A	📷	80.7	79.8
Baseline++	euclid.	N/A	–	81.9	82.2
	cosine	N/A	📷	79.7	81.1

Table 1: Mean accuracy on the 12 ARSC binary classification test tasks. In column †, results are reproduced from the Induction Networks seminal paper (Geng et al., 2019) (where applies), a dash (−) means that results for that encoder/metric pair were not reported, and 📷 denotes models only tested on computer vision tasks (first time applied to text in our contribution). The *BERT* column is our implementation using the same 5 shots as the first column but using a BERT encoder for all methods. The last column is also using BERT, but results are averaged over five runs, sampling different shots for each run. In the Configuration column, N/A means that the configuration criteria does not apply to the model.

Correct classification examples	
S:	Do I have enough in my boa account for a new pair of skis ?
P:	**`balance`**
T:	**`balance`**
S:	What's 15% of 68 ?
P:	**`calculator`**
T:	**`calculator`**
S:	I need to know the nearest bank's location.
P:	**`directions`**
T:	**`directions`**
S:	What do I take home ?
P:	**`income`**
T:	**`income`**
Incorrect classification examples	
S:	On Tuesday you are supposed to have a meeting.
P:	**`meeting_schedule`**
T:	**`calendar`**
S:	What are my insurance rewards ?
P:	**`insurance`**
T:	**`redeem_rewards`**
S:	How much farther is Orlando from my location?
P:	**`current_location`**
T:	**`distance`**
S:	Stop talking please.
P:	**`change_speed`**
T:	**`cancel`**

Table 2: Examples of OOS query examples correctly and incorrectly predicted by the `Baseline` method using 5 shots. S (resp. P, T) is the sentence (resp. prediction and true label).

performances of all methods. When feeding the same transformer-based encoder to all few-shot methods, Prototypical Networks are now on top, whereas metric learning approaches (Induction & Relation Networks) tend to struggle, almost reaching the same performances as Matching Networks. Such metric learning approaches rely on various weight matrices and parameters, while more traditional approaches (`Matching`, `Proto`) do not use any parameter apart from the encoding step. This hints that the upstream transformer does most of the learning and is able to model the embedding space well enough such that no more additional metric learning is needed. The massive increase in embedding quality brought by the BERT encoder makes Prototypical Network approaches reclaim the state-of-the-art position.

4.4 The curious case of induction networks

When Geng et al. (2019) introduced Induction Networks, both the ARSC dataset and a private intent detection dataset were used for evaluation (publicly unavailable). Our experiments of this method on the ARSC dataset confirm those results in an acceptable range, even when trying to get more consistent results using multiple random seeds. Nonetheless, the performances of this method are underwhelming on all three intent detection datasets, even when matching the binary classification scenario using $C = 2$. Those poor performances were observed both on the test set

	Metric	Relation Module	Liu				OOS				TREC28			
			2	3	4	5	2	3	4	5	2	3	4	5
Matching	euclid.	-	96.6	93.7	91.1	89.1	99.2	98.7	98.1	97.7	89.4	81.6	76.6	69.6
	cosine	-	93.3	87.9	84.8	81.0	96.8	95.8	95.1	94.7	81.6	75.4	68.5	63.5
Proto	euclid.	-	97.4	95.3	93.4	91.8	**99.5**	99.0	98.7	98.4	**92.6**	**87.6**	82.0	**79.2**
	cosine	-	94.6	90.4	88.5	85.6	97.6	97.3	96.9	96.5	85.6	79.1	74.5	71.3
Proto++	euclid.	-	**97.7**	**95.7**	**93.7**	**92.2**	99.5	**99.1**	**98.8**	**98.5**	91.7	84.9	**82.0**	76.8
	cosine	-	94.0	90.9	87.9	85.4	97.5	97.3	97.0	96.5	83.8	78.1	71.0	65.9
Relation	-	base	88.2	76.5	71.8	65.1	91.1	86.0	79.9	77.9	80.8	66.3	61.7	51.8
	-	ntl	87.4	80.1	74.3	69.0	90.9	84.2	82.0	77.8	74.7	62.5	57.7	48.6
Induction	-	ntl	73.9	57.9	52.6	40.6	74.9	59.3	50.9	43.8	70.3	49.6	41.9	33.9
Baseline	-	-	94.3	89.0	84.1	79.8	99.1	98.5	97.7	97.2	90.5	83.6	79.3	75.7
Baseline++	euclid.	-	93.1	87.6	81.4	78.1	95.8	93.3	92.1	90.6	87.7	78.3	72.5	69.1
	cosine	-	93.1	86.8	81.0	75.1	98.9	97.9	96.8	96.1	86.7	78.2	72.1	70.0

Table 3: Mean accuracy of C-way 5-shot intent detection, with C ranging between 2 and 5. Each reported value is the average over five runs with different random seeds. For each column, the best method is highlighted in **bold**.

as well as the train set, discarding the over-fitting argument. Such a big performance gap between sentiment and intent classification tasks show that Induction Networks, while suited for the former, are not directly applicable to any type of task.

4.5 On metric choice

Prototypical Networks were originally designed to do better than Matching Networks. The two differences between them are the placement of the `class average` step, and the choice of the metric (cosine for Matching, euclidean for Prototypical). Our results show that metric choice yields a big gap in performances for both methods, this gap being larger than the gap caused by the model design. This hints that when using a pre-defined metric – excluding the case of metric learning –, choosing the right metric is of paramount importance. Moreover, while `Matching Networks` were designed to use the cosine distance, we found here that they perform significantly better when equipped with the Euclidean distance (on all datasets for all number of given test classes).

4.6 On architectural choices

Overall, Prototypical Networks come on top of every intent detection dataset. More importantly, their gap between other competing approaches is wider as the number of classes increases. This result is important, as in practice, the number of classes is likely to be higher than what is used in the literature. The extended variant, proto++, obtains mixed results. While this shows that using unlabeled data can have some benefits, we also observe that the proto++ way of integrating this external knowledge is perfectible. Ultimately, note that our results do not mirror Computer Vision results. Since few-shot learning methods are used on top of embeddings, we could emit the hypothesis that they can be applied to any embeddings, regardless of the field. However, while Relation Networks, for example, were performing well in Computer Vision classification tasks – the tasks which they were originally designed for – as well as text classification – back in the days when transformers did not exist –, this is not the case anymore. The drawback is that all methods are very sensitive to the feature extractor used in prior steps.

5 Conclusion

We provided a fair comparison for end-to-end neural few-shot text classification methods discovered over the last few years. When they are all equipped with a transformer-based text encoder, we show that Prototypical Networks become the state-of-the-art again. We also found that a traditional classifier trained on few shots yields very competitive results, especially when given shots are re-sampled at each iteration. Ultimately, we also demonstrated the significant impact of the vector metric, illustrated by Matching Networks strongly improving by only replacing the cosine by the euclidean distance. The complete source code with the re-implementation of all the tested methods and evaluation framework used in this study is publicly available[5] – we hope that it will help the community build upon consistent comparative experiments.

[5]https://github.com/tdopierre/FewShotText

References

John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 440–447.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100.

Nontawat Charoenphakdee, Jongyeong Lee, Yiping Jin, Dittaya Wanvarie, and Masashi Sugiyama. 2019. Learning only from relevant keywords and unlabeled documents. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*.

Ruiying Geng, Binhua Li, Yongbin Li, Xiaodan Zhu, Ping Jian, and Jian Sun. 2019. Induction networks for few-shot text classification. *arXiv preprint arXiv:1902.10482*.

Stefan Larson, Anish Mahendran, Joseph J Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K Kummerfeld, Kevin Leach, Michael A Laurenzano, Lingjia Tang, et al. 2019. An evaluation dataset for intent classification and out-of-scope prediction. *arXiv preprint arXiv:1909.02027*.

Xingkun Liu, Arash Eshghi, Pawel Swietojanski, and Verena Rieser. 2019. Benchmarking natural language understanding services for building conversational agents. *arXiv preprint arXiv:1903.05566*.

Rada Mihalcea. 2004. Co-training and self-training for word sense disambiguation. In *Proc. of 8th CoNLL-2004 (HLT-NAACL 2004)*.

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. 2018. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, and Timothy M. Hospedales. 2018. Learning to compare: Relation network for few-shot learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601. Association for Computational Linguistics.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. 2019. Unsupervised data augmentation for consistency training. *arXiv preprint arXiv:1904.12848*.

Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. 2018a. Diverse few-shot text classification with multiple metrics. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, New Orleans, Louisiana. Association for Computational Linguistics.

Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. 2018b. Diverse few-shot text classification with multiple metrics. *arXiv preprint arXiv:1805.07513*.

Zhi-Hua Zhou and Ming Li. 2005. Tri-training: exploiting unlabeled data using three classifiers. *IEEE TKDE*, 17(11):1529–1541.