Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT

You can find the code to reproduce the training of DistilBERT along with pre-trained weights for DistilBERT here.

In the last 18 months, transfer learning from large-scale language models has significantly improved upon the state-of-the-art on pretty much every Natural Language Processing task.

Usually based on the Transformer architecture of Vaswani et al., these pre-trained language models keep getting larger and larger and being trained on bigger datasets. The latest model from Nvidia has 8.3 billion parameters: 24 times larger than BERT-large, 5 times larger than GPT-2, while RoBERTa, the latest work from Facebook AI, was trained on 160GB of text 😍
Some people in the community question the relevance of keeping on training larger and larger Transformer especially when you take into account the financial and environmental cost of training. Here's are some of the latest large models and their size in millions of parameters.

At Hugging Face, we experienced first-hand the growing popularity of these models as our NLP library — which encapsulates most of them — got installed more than 400,000 times in just a few months.

However, as these models were reaching a larger NLP community, an important and challenging question started to emerge. **How should we put these monsters in production? How can we use such large models under low latency constraints?** Do we need (costly) GPU servers to serve at scale?

For many researchers and developers, these can be deal-breaking issues 🤔

To build more privacy-respecting systems, we noticed an increasing need to have machine learning systems operate **on the edge** rather than calling a cloud API and sending possibly private data to servers. Running models on
devices like your smartphone also requires light-weight, responsive and energy-efficient models!

Last but not least, we are more and more concerned about the environmental cost of scaling exponentially computing requirements of these models.

So, how can we reduce the size of these monster models?!

There are many techniques available to tackle the previous questions. The most common tools include quantization (approximating the weights of a network with a smaller precision) and weights pruning (removing some connections in the network). For these technics, you can have a look at the excellent blog post of Rasa on quantizing BERT.

We decided to focus on distillation: a technique you can use to compress a large model, called the teacher, into a smaller model, called the student.

Knowledge Distillation — Transferring generalization capabilities

Knowledge distillation (sometimes also referred to as teacher-student learning) is a compression technique in which a small model is trained to reproduce the behavior of a larger model (or an ensemble of models). It was introduced by Bucila et al. and generalized by Hinton et al. a few years later. We will follow the latter method.

In supervised learning, a classification model is generally trained to predict a gold class by maximizing its probability (softmax of logits) using the log-likelihood signal. In many cases, a good performance model will predict an output distribution with the correct class having a high probability, leaving other classes with probabilities near zero.

But, some of these “almost-zero” probabilities are larger than the others, and this reflects, in part, the
generalization capabilities of the model.

For instance, a `desk chair` might be mistaken with an `armchair` but should usually not be mistaken with a `mushroom`. This uncertainty is sometimes referred to as the “dark knowledge” 😁.

Another way to understand distillation is that it prevents the model to be too sure about its prediction (similarly to label smoothing).

Here is an example to see this idea in practice. In language modeling, we can easily observe this uncertainty by looking at the distribution over the vocabulary. Here are the top 20 guesses by Bert for completing this famous quote from the *Casablanca* movie:

![Top 20 guesses from BERT (base) for the masked token. The Language model identified two highly probable tokens (day & life) followed by a long tail of valid tokens.](image)

How can we copy this dark knowledge?

In the **teacher-student training**, we train a student network to mimic the **full output distribution** of the teacher network (its knowledge).

We are training the student to generalize the same way as the teacher by matching the output distribution.

Rather than training with a cross-entropy over the hard targets (one-hot encoding of the gold class), we transfer the knowledge from the teacher to the student with a cross-entropy over the soft targets (probabilities of the teacher). Our training loss thus becomes:
\[ L = - \sum_i t_i \cdot \log(s_i) \]

With \( t \) the logits from the teacher and \( s \) the logits of the student.

This loss is a richer training signal since a single example enforces much more constraint than a single hard target.

To further expose the mass of the distribution over the classes, Hinton et al. introduce a **softmax-temperature**:

\[ p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \]

\( T \) is the temperature parameter.

When \( T \to 0 \), the distribution becomes a Kronecker (and is equivalent to the one-hot target vector), when \( T \to +\infty \), it becomes a uniform distribution. **The same temperature parameter is applied both to the student and the teacher at training time, further revealing more signals for each training example.** At inference, \( T \) is set to 1 and recover the standard Softmax.

**Hands-on coding in PyTorch — Compressing BERT**

We want to compress a large language model using distilling. For distilling, we'll use the Kullback-Leibler loss since the optimizations are equivalent:

\[ KL(p \| q) = \mathbb{E}_p(\log\left(\frac{p}{q}\right)) = \sum_i p_i \cdot \log(p_i) - \sum_i p_i \cdot \log(q_i) \]

When computing the gradients with respect to \( q \) (the student distribution) we obtain the same gradients. It allows us to leverage PyTorch implementation for faster computation:
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Optimizer

KD_loss = nn.KLDivLoss(reduction='batchmean')

def kd_step(teacher: nn.Module,
            student: nn.Module,
            temperature: float,
            inputs: torch.tensor,
            optimizer: Optimizer):
    teacher.eval()
    student.train()

    with torch.no_grad():
        logits_t = teacher(inputs=inputs)
        logits_s = student(inputs=inputs)

        loss = KD_loss(input=F.log_softmax(logits_s/temperature, dim=-1),
                        target=F.softmax(logits_t/temperature, dim=-1))

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

A Knowledge distillation training step in PyTorch. Copy the gist from here.

Using the teacher signal, we are able to train a **smaller language model**, we call **DistilBERT**, from the **supervision of BERT** (we used the English bert-base-uncased version of Bert).

Following Hinton et al., the training loss is a linear combination of the **distillation loss** and the masked **language modeling loss**. Our student is a small version of BERT in which we removed the token-type embeddings and the pooler (used for the next sentence classification task) and kept the rest of the architecture identical while reducing the numbers of layers by a factor of two.

Overall, our distilled model, DistilBERT, has about half the total number of parameters of BERT base and retains 95% of Bert’s performances on the language understanding benchmark GLUE.

> **Note 1 — Why not reducing the hidden size as well?**

Reducing it from 768 to 512 would reduce the total number of parameters by ~2. However, in modern frameworks, most of the operations are highly
optimized and variations on the last dimension of the tensor (hidden dimension) have a small impact on most of the operations used in the Transformer architecture (linear layers and layer normalization). In our experiments, the number of layers was the determining factor for the inference time, more than the hidden size. Smaller does not necessarily imply faster...

Note 2 — Some works on distillation like Tang et al. use the L2 distance as a distillation loss directly on downstream tasks. Our early experiments suggested that the cross-entropy loss leads to significantly better performance in our case. We hypothesize that in a language modeling setup, the output space (vocabulary) is significantly larger than the dimension of the downstream task output space. The logits may thus compensate for each other in the L2 loss.

Training a sub-network is not only about the architecture. It is also about finding the right initialization for the sub-network to converge (see The Lottery Ticket Hypothesis for instance). We thus initialize our student, DistilBERT, from its teacher, Bert, by taking one layer out of two, leveraging the common hidden size between student and teacher.

We also used a few training tricks from the recent RoBERTa paper which showed that the way Bert is trained is crucial for its final performance. Following RoBERTa, we trained DistilBERT on very large batches leveraging gradient accumulation (up to 4000 examples per batch), with dynamic masking and removed the next sentence prediction objective.

Our training setup is voluntarily limited in terms of resources. We train DistilBERT on eight 16GB V100 GPUs for approximately three and a half days using the concatenation of Toronto Book Corpus and English Wikipedia (same data as original BERT).

The code for DistilBert is adapted in part from Facebook XLM’s code and in part from our PyTorch version of Google AI Bert and is available in our pytorch-transformers library 🦋 along with several trained and fine-tuned versions of DistilBert and the code to reproduce the training and fine-tuning.

Model performances — Testing DistilBERT
We compare the performance of DistilBERT on the development sets of the GLUE benchmark against two baselines: BERT base (DistilBERT’s teacher) and a strong non-transformer baseline from NYU: two BiLSTMs on top of ELMo. We use the giant library from NYU for ELMo baselines and pytorch-transformers for the BERT baseline.

As shown in the following table, DistilBERT’s performances compare favorably with the baselines while having respectively about half and one third the number of parameters (more on this below). Among the 9 tasks, DistilBERT is always on par or improving over the ELMo baseline (up to 14 points of accuracy on QNLI). DistilBERT also compares surprisingly well to BERT: we are able to retain more than 95% of the performance while having 40% fewer parameters.

<table>
<thead>
<tr>
<th></th>
<th>CoLA</th>
<th>MNLJ</th>
<th>MNLJ-MM</th>
<th>MRPC</th>
<th>QNLI</th>
<th>QQP</th>
<th>RTE</th>
<th>SST-2</th>
<th>STS-B</th>
<th>WNLI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLUE BASELINE (ELMo + BiLSTMs)</td>
<td>68.7</td>
<td>44.1</td>
<td>69.6 (avg)</td>
<td>70.8</td>
<td>42.3</td>
<td>71.1</td>
<td>88.0</td>
<td>94.3</td>
<td>52.4</td>
<td>91.5</td>
</tr>
<tr>
<td>BERT base</td>
<td>78.8</td>
<td>55.8</td>
<td>83.7</td>
<td>84.1</td>
<td>86.3</td>
<td>90.5</td>
<td>91.1</td>
<td>90.9</td>
<td>87.7</td>
<td>66.6</td>
</tr>
<tr>
<td>DistilBERT</td>
<td>75.2</td>
<td>42.5</td>
<td>81.6</td>
<td>81.1</td>
<td>82.4</td>
<td>88.3</td>
<td>85.5</td>
<td>90.6</td>
<td>87.7</td>
<td>60.0</td>
</tr>
</tbody>
</table>

Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are medians of 5 runs with different seeds.

In terms of inference time, DistilBERT is more than **60% faster and smaller** than BERT and **120% faster and smaller** than ELMo + BiLSTM 🧵

<table>
<thead>
<tr>
<th></th>
<th>Nb of parameters (millions)</th>
<th>Inference Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLUE BASELINE (ELMo + BiLSTMs)</td>
<td>180</td>
<td>895</td>
</tr>
<tr>
<td>BERT base</td>
<td>110</td>
<td>668</td>
</tr>
<tr>
<td>DistilBERT</td>
<td>66</td>
<td>410</td>
</tr>
</tbody>
</table>

To further investigate the speed-up/size trade-off of DistilBERT, we compare, in the left table, the number of parameters of each model along with the inference time needed to do a full pass on the STS-B dev set on CPU (using a batch size of 1).

🔍 **Downstream task: Distillation & transfer-learning**

We further study the use of DistilBERT on downstream tasks under efficient inference constraints. We use our compact pre-trained language model by fine-tuning it a classification task. A nice way to actually **mix distillation**
pre-training and transfer-learning!

<table>
<thead>
<tr>
<th>review</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>With a cast of stalwart British character acto...</td>
<td>0</td>
</tr>
<tr>
<td>Five passengers at a bus depot tell each other...</td>
<td>0</td>
</tr>
<tr>
<td>Well, I had seen &quot;They all laughed&quot; when it ca...</td>
<td>1</td>
</tr>
<tr>
<td>In what seemed like the longest 1hour 35 I've ...</td>
<td>0</td>
</tr>
<tr>
<td>huge Ramones fan. i do like the ramones, and i...</td>
<td>1</td>
</tr>
</tbody>
</table>

Extract from the IMDB Review dataset — Source: Kaggle

We selected the **IMDB Review Sentiment Classification** which is composed of 50'000 reviews in English labeled as positive or negative: 25'000 for training and 25'000 for test (and with balanced classes). We trained on a single 12GB K80.

First, we train **bert-base-uncased** on our dataset. Our dear BERT 😍 reaches an **accuracy of 99.98%** (average of 3 runs). That’s almost perfect!

We then train DistilBERT, using the same hyper-parameters. The compressed model reaches **an accuracy of 99.53%** (average of 3 runs). An absolute difference of 0.5% in performances for a 60% reduction in latency and 40% in size 📈

Another common application of NLP is **Question Answering**. We compared the results of the **bert-base-uncased** version of BERT with DistilBERT on the SQuAD 1.1 dataset. On the development set, BERT reaches an F1 score of 88.5 and an EM (Exact-match) score of 81.2. We train DistilBERT on the same set of hyper-parameters and reach scores of **85.1 F1 and 76.5 EM, within 3 to 5 points of the full BERT**.

We also studied whether we could add another step of distillation during the adaptation phase by finetuning DistilBERT on SQuAD using the finetuned BERT model as a teacher with a knowledge distillation loss.

---

*Here we are finetuning by distilling a question answering model into a language model previously pre-trained with knowledge distillation! That a lot*
In this case, we were able to reach interesting performances given the size of the network: **86.2 F1 and 78.1 EM**, ie. **within 3 points of the full model**!

🎉 **Less is more: smaller models also spark joy 🌟**

We are very excited about DistilBERT’s potential. The work we’ve presented is just the beginning of what can be done and raises many questions: How far can we compress these models with knowledge distillation? Can these technics be used to get further insights into the knowledge stored in the large version? What aspects of linguistic/semantics do we lose in this type of compression?…

One essential aspect of our work at HuggingFace is open-source and knowledge sharing as you can see from our GitHub and medium pages. We think it is both the easiest and fairest way for everyone to participate and reap the fruits of the remarkable progress of deep learning for NLP.

Thus, together with this blog post, we release the code of our experiments 🌟 (in particular the code to reproduce the training and fine-tuning of DistilBERT) along with a trained version of DistilBERT in our pytorch-transformers library🌟.

*Many thanks to Sam Bowman, Alex Wang and Thibault Févry for feedback and discussions!*