Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [41] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test *set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.*

The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions[1]*, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.*

1. Introduction

Deep convolutional neural networks [22, 21] have led to a series of breakthroughs for image classification [21, 50, 40]. Deep networks naturally integrate low/mid/high-level features [50] and classifiers in an end-to-end multi-layer fashion, and the "levels" of features can be enriched by the number of stacked layers (depth). Recent evidence [41, 44] reveals that network depth is of crucial importance, and the leading results [41, 44, 13, 16] on the challenging ImageNet dataset [36] all exploit "very deep" [41] models, with a depth of sixteen [41] to thirty [16]. Many other non-trivial visual recognition tasks [8, 12, 7, 32, 27] have also greatly benefited from very deep models.

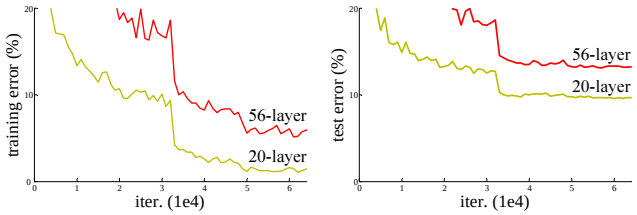


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [1, 9], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 9, 37, 13] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with back-propagation [22].

When deeper networks are able to start converging, a *degradation* problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is *not caused by overfitting*, and adding more layers to a suitably deep model leads to *higher training error*, as reported in [11, 42] and thoroughly verified by our experiments. Fig. 1 shows a typical example.

The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution *by construction* to the deeper model: the added layers are *identity* mapping, and the other layers are copied from the learned shallower model. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart. But experiments show that our current solvers on hand are unable to find solutions that

[1] http://image-net.org/challenges/LSVRC/2015/ and http://mscoco.org/dataset/#detections-challenge2015.

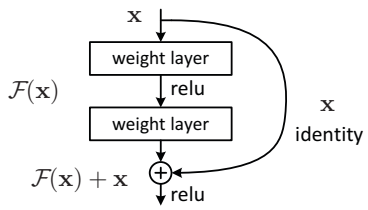


Figure 2. Residual learning: a building block.

are comparably good or better than the constructed solution (or unable to do so in feasible time).

In this paper, we address the degradation problem by introducing a *deep residual learning* framework. Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping. Formally, denoting the desired underlying mapping as $\mathcal{H}(\mathbf{x})$, we let the stacked nonlinear layers fit another mapping of $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The original mapping is recast into $\mathcal{F}(\mathbf{x})+\mathbf{x}$. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

The formulation of $\mathcal{F}(\mathbf{x})+\mathbf{x}$ can be realized by feedforward neural networks with "shortcut connections" (Fig. 2). Shortcut connections [2, 34, 49] are those skipping one or more layers. In our case, the shortcut connections simply perform *identity* mapping, and their outputs are added to the outputs of the stacked layers (Fig. 2). Identity shortcut connections add neither extra parameter nor computational complexity. The entire network can still be trained end-to-end by SGD with backpropagation, and can be easily implemented using common libraries (*e.g.*, Caffe [19]) without modifying the solvers.

We present comprehensive experiments on ImageNet [36] to show the degradation problem and evaluate our method. We show that: 1) Our extremely deep residual nets are easy to optimize, but the counterpart "plain" nets (that simply stack layers) exhibit higher training error when the depth increases; 2) Our deep residual nets can easily enjoy accuracy gains from greatly increased depth, producing results substantially better than previous networks.

Similar phenomena are also shown on the CIFAR-10 set [20], suggesting that the optimization difficulties and the effects of our method are not just akin to a particular dataset. We present successfully trained models on this dataset with over 100 layers, and explore models with over 1000 layers.

On the ImageNet classification dataset [36], we obtain excellent results by extremely deep residual nets. Our 152-layer residual net is the deepest network ever presented on ImageNet, while still having lower complexity than VGG nets [41]. Our ensemble has **3.57%** top-5 error on the ImageNet *test* set, and *won the 1st place in the ILSVRC 2015 classification competition*. The extremely deep representations also have excellent generalization performance on other recognition tasks, and lead us to further *win the 1st places on: ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation* in ILSVRC & COCO 2015 competitions. This strong evidence shows that the residual learning principle is generic, and we expect that it is applicable in other vision and non-vision problems.

2. Related Work

Residual Representations. In image recognition, VLAD [18] is a representation that encodes by the residual vectors with respect to a dictionary, and Fisher Vector [30] can be formulated as a probabilistic version [18] of VLAD. Both of them are powerful shallow representations for image retrieval and classification [4, 48]. For vector quantization, encoding residual vectors [17] is shown to be more effective than encoding original vectors.

In low-level vision and computer graphics, for solving Partial Differential Equations (PDEs), the widely used Multigrid method [3] reformulates the system as subproblems at multiple scales, where each subproblem is responsible for the residual solution between a coarser and a finer scale. An alternative to Multigrid is hierarchical basis preconditioning [45, 46], which relies on variables that represent residual vectors between two scales. It has been shown [3, 45, 46] that these solvers converge much faster than standard solvers that are unaware of the residual nature of the solutions. These methods suggest that a good reformulation or preconditioning can simplify the optimization.

Shortcut Connections. Practices and theories that lead to shortcut connections [2, 34, 49] have been studied for a long time. An early practice of training multi-layer perceptrons (MLPs) is to add a linear layer connected from the network input to the output [34, 49]. In [44, 24], a few intermediate layers are directly connected to auxiliary classifiers for addressing vanishing/exploding gradients. The papers of [39, 38, 31, 47] propose methods for centering layer responses, gradients, and propagated errors, implemented by shortcut connections. In [44], an "inception" layer is composed of a shortcut branch and a few deeper branches.

Concurrent with our work, "highway networks" [42, 43] present shortcut connections with gating functions [15]. These gates are data-dependent and have parameters, in contrast to our identity shortcuts that are parameter-free. When a gated shortcut is "closed" (approaching zero), the layers in highway networks represent *non-residual* functions. On the contrary, our formulation always learns residual functions; our identity shortcuts are never closed, and all information is always passed through, with additional residual functions to be learned. In addition, high-

way networks have not demonstrated accuracy gains with extremely increased depth (*e.g.*, over 100 layers).

3. Deep Residual Learning

3.1. Residual Learning

Let us consider $\mathcal{H}(\mathbf{x})$ as an underlying mapping to be fit by a few stacked layers (not necessarily the entire net), with $\mathbf{x}$ denoting the inputs to the first of these layers. If one hypothesizes that multiple nonlinear layers can asymptotically approximate complicated functions[2], then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, *i.e.*, $\mathcal{H}(\mathbf{x}) - \mathbf{x}$ (assuming that the input and output are of the same dimensions). So rather than expect stacked layers to approximate $\mathcal{H}(\mathbf{x})$, we explicitly let these layers approximate a residual function $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. The original function thus becomes $\mathcal{F}(\mathbf{x})+\mathbf{x}$. Although both forms should be able to asymptotically approximate the desired functions (as hypothesized), the ease of learning might be different.

This reformulation is motivated by the counterintuitive phenomena about the degradation problem (Fig. 1, left). As we discussed in the introduction, if the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart. The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers. With the residual learning reformulation, if identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings.

In real cases, it is unlikely that identity mappings are optimal, but our reformulation may help to precondition the problem. If the optimal function is closer to an identity mapping than to a zero mapping, it should be easier for the solver to find the perturbations with reference to an identity mapping, than to learn the function as a new one. We show by experiments (Fig. 7) that the learned residual functions in general have small responses, suggesting that identity mappings provide reasonable preconditioning.

3.2. Identity Mapping by Shortcuts

We adopt residual learning to every few stacked layers. A building block is shown in Fig. 2. Formally, in this paper we consider a building block defined as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \tag{1}$$

Here $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of the layers considered. The function $\mathcal{F}(\mathbf{x}, \{W_i\})$ represents the residual mapping to be learned. For the example in Fig. 2 that has two layers, $\mathcal{F} = W_2\sigma(W_1\mathbf{x})$ in which σ denotes ReLU [29] and the biases are omitted for simplifying notations. The operation $\mathcal{F} + \mathbf{x}$ is performed by a shortcut connection and element-wise addition. We adopt the second nonlinearity after the addition (*i.e.*, $\sigma(\mathbf{y})$, see Fig. 2).

The shortcut connections in Eqn.(1) introduce neither extra parameter nor computation complexity. This is not only attractive in practice but also important in our comparisons between plain and residual networks. We can fairly compare plain/residual networks that simultaneously have the same number of parameters, depth, width, and computational cost (except for the negligible element-wise addition).

The dimensions of $\mathbf{x}$ and $\mathcal{F}$ must be equal in Eqn.(1). If this is not the case (*e.g.*, when changing the input/output channels), we can perform a linear projection W_s by the shortcut connections to match the dimensions:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}. \tag{2}$$

We can also use a square matrix W_s in Eqn.(1). But we will show by experiments that the identity mapping is sufficient for addressing the degradation problem and is economical, and thus W_s is only used when matching dimensions.

The form of the residual function $\mathcal{F}$ is flexible. Experiments in this paper involve a function $\mathcal{F}$ that has two or three layers (Fig. 5), while more layers are possible. But if $\mathcal{F}$ has only a single layer, Eqn.(1) is similar to a linear layer: $\mathbf{y} = W_1\mathbf{x} + \mathbf{x}$, for which we have not observed advantages.

We also note that although the above notations are about fully-connected layers for simplicity, they are applicable to convolutional layers. The function $\mathcal{F}(\mathbf{x}, \{W_i\})$ can represent multiple convolutional layers. The element-wise addition is performed on two feature maps, channel by channel.

3.3. Network Architectures

We have tested various plain/residual nets, and have observed consistent phenomena. To provide instances for discussion, we describe two models for ImageNet as follows.

Plain Network. Our plain baselines (Fig. 3, middle) are mainly inspired by the philosophy of VGG nets [41] (Fig. 3, left). The convolutional layers mostly have 3×3 filters and follow two simple design rules: (i) for the same output feature map size, the layers have the same number of filters; and (ii) if the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer. We perform downsampling directly by convolutional layers that have a stride of 2. The network ends with a global average pooling layer and a 1000-way fully-connected layer with softmax. The total number of weighted layers is 34 in Fig. 3 (middle).

It is worth noticing that our model has *fewer* filters and *lower* complexity than VGG nets [41] (Fig. 3, left). Our 34-layer baseline has 3.6 billion FLOPs (multiply-adds), which is only 18% of VGG-19 (19.6 billion FLOPs).

[2]This hypothesis, however, is still an open question. See [28].

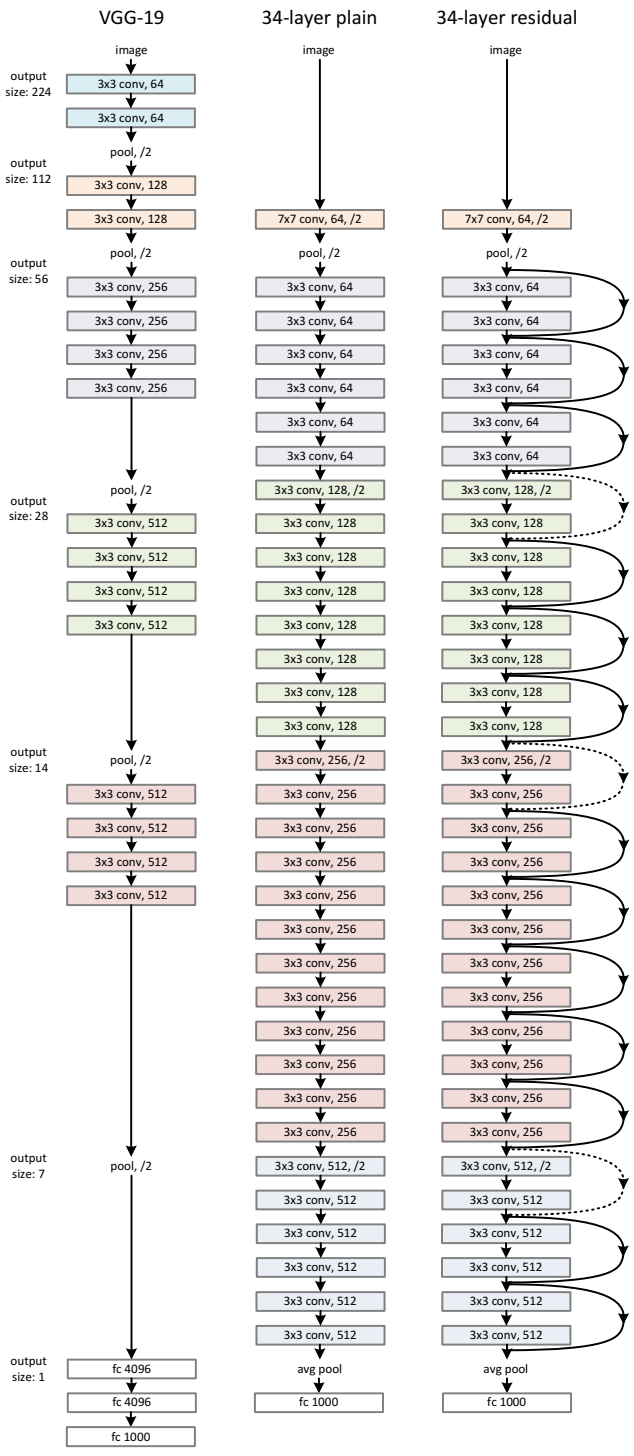


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

Residual Network. Based on the above plain network, we insert shortcut connections (Fig. 3, right) which turn the network into its counterpart residual version. The identity shortcuts (Eqn.(1)) can be directly used when the input and output are of the same dimensions (solid line shortcuts in Fig. 3). When the dimensions increase (dotted line shortcuts in Fig. 3), we consider two options: (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter; (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1×1 convolutions). For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.

3.4. Implementation

Our implementation for ImageNet follows the practice in [21, 41]. The image is resized with its shorter side randomly sampled in $[256, 480]$ for scale augmentation [41]. A 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted [21]. The standard color augmentation in [21] is used. We adopt batch normalization (BN) [16] right after each convolution and before activation, following [16]. We initialize the weights as in [13] and train all plain/residual nets from scratch. We use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus, and the models are trained for up to 60×10^4 iterations. We use a weight decay of 0.0001 and a momentum of 0.9. We do not use dropout [14], following the practice in [16].

In testing, for comparison studies we adopt the standard 10-crop testing [21]. For best results, we adopt the fully-convolutional form as in [41, 13], and average the scores at multiple scales (images are resized such that the shorter side is in $\{224, 256, 384, 480, 640\}$).

4. Experiments

4.1. ImageNet Classification

We evaluate our method on the ImageNet 2012 classification dataset [36] that consists of 1000 classes. The models are trained on the 1.28 million training images, and evaluated on the 50k validation images. We also obtain a final result on the 100k test images, reported by the test server. We evaluate both top-1 and top-5 error rates.

Plain Networks. We first evaluate 18-layer and 34-layer plain nets. The 34-layer plain net is in Fig. 3 (middle). The 18-layer plain net is of a similar form. See Table 1 for detailed architectures.

The results in Table 2 show that the deeper 34-layer plain net has higher validation error than the shallower 18-layer plain net. To reveal the reasons, in Fig. 4 (left) we compare their training/validation errors during the training procedure. We have observed the degradation problem - the

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

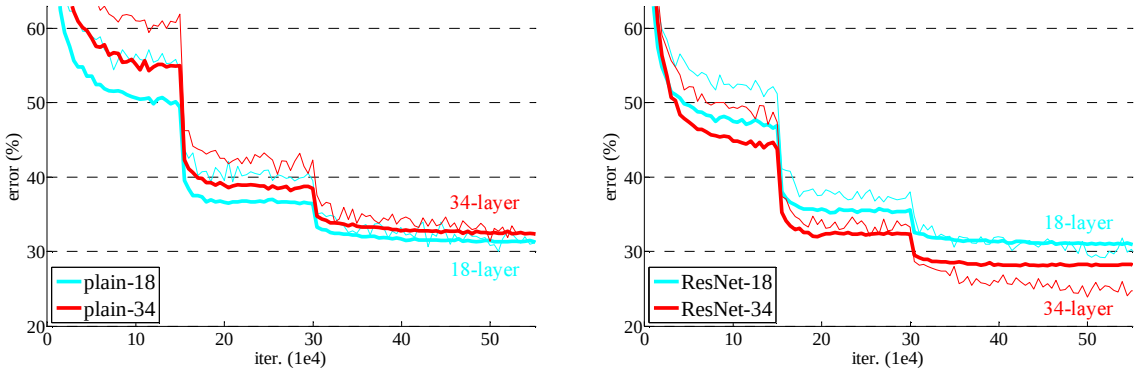


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

34-layer plain net has higher *training* error throughout the whole training procedure, even though the solution space of the 18-layer plain network is a subspace of that of the 34-layer one.

We argue that this optimization difficulty is *unlikely* to be caused by vanishing gradients. These plain networks are trained with BN [16], which ensures forward propagated signals to have non-zero variances. We also verify that the backward propagated gradients exhibit healthy norms with BN. So neither forward nor backward signals vanish. In fact, the 34-layer plain net is still able to achieve competitive accuracy (Table 3), suggesting that the solver works to some extent. We conjecture that the deep plain nets may have exponentially low convergence rates, which impact the reducing of the training error[3]. The reason for such optimization difficulties will be studied in the future.

Residual Networks. Next we evaluate 18-layer and 34-layer residual nets (*ResNets*). The baseline architectures are the same as the above plain nets, expect that a shortcut connection is added to each pair of 3×3 filters as in Fig. 3 (right). In the first comparison (Table 2 and Fig. 4 right), we use identity mapping for all shortcuts and zero-padding for increasing dimensions (option A). So they have *no extra parameter* compared to the plain counterparts.

We have three major observations from Table 2 and Fig. 4. First, the situation is reversed with residual learning – the 34-layer ResNet is better than the 18-layer ResNet (by 2.8%). More importantly, the 34-layer ResNet exhibits considerably lower training error and is generalizable to the validation data. This indicates that the degradation problem is well addressed in this setting and we manage to obtain accuracy gains from increased depth.

Second, compared to its plain counterpart, the 34-layer

[3]We have experimented with more training iterations (3×) and still observed the degradation problem, suggesting that this problem cannot be feasibly addressed by simply using more iterations.

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

ResNet reduces the top-1 error by 3.5% (Table 2), resulting from the successfully reduced training error (Fig. 4 right *vs*. left). This comparison verifies the effectiveness of residual learning on extremely deep systems.

Last, we also note that the 18-layer plain/residual nets are comparably accurate (Table 2), but the 18-layer ResNet converges faster (Fig. 4 right *vs*. left). When the net is "not overly deep" (18 layers here), the current SGD solver is still able to find good solutions to the plain net. In this case, the ResNet eases the optimization by providing faster convergence at the early stage.

Identity *vs*. Projection Shortcuts. We have shown that parameter-free, identity shortcuts help with training. Next we investigate projection shortcuts (Eqn.(2)). In Table 3 we compare three options: (A) zero-padding shortcuts are used for increasing dimensions, and all shortcuts are parameter-free (the same as Table 2 and Fig. 4 right); (B) projection shortcuts are used for increasing dimensions, and other shortcuts are identity; and (C) all shortcuts are projections.

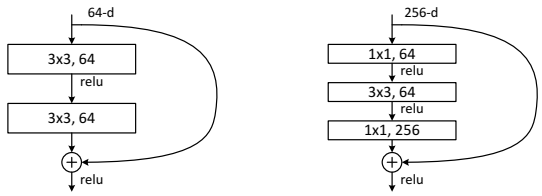


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

Table 3 shows that all three options are considerably better than the plain counterpart. B is slightly better than A. We argue that this is because the zero-padded dimensions in A indeed have no residual learning. C is marginally better than B, and we attribute this to the extra parameters introduced by many (thirteen) projection shortcuts. But the small differences among A/B/C indicate that projection shortcuts are not essential for addressing the degradation problem. So we do not use option C in the rest of this paper, to reduce memory/time complexity and model sizes. Identity shortcuts are particularly important for not increasing the complexity of the bottleneck architectures that are introduced below.

Deeper Bottleneck Architectures. Next we describe our deeper nets for ImageNet. Because of concerns on the training time that we can afford, we modify the building block as a *bottleneck* design[4]. For each residual function $\mathcal{F}$, we use a stack of 3 layers instead of 2 (Fig. 5). The three layers are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing (restoring) dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions. Fig. 5 shows an example, where both designs have similar time complexity.

The parameter-free identity shortcuts are particularly important for the bottleneck architectures. If the identity shortcut in Fig. 5 (right) is replaced with projection, one can show that the time complexity and model size are doubled, as the shortcut is connected to the two high-dimensional ends. So identity shortcuts lead to more efficient models for the bottleneck designs.

50-layer ResNet: We replace each 2-layer block in the

[4]Deeper *non*-bottleneck ResNets (*e.g*., Fig. 5 left) also gain accuracy from increased depth (as shown on CIFAR-10), but are not as economical as the bottleneck ResNets. So the usage of bottleneck designs is mainly due to practical considerations. We further note that the degradation problem of plain nets is also witnessed for the bottleneck designs.

34-layer net with this 3-layer bottleneck block, resulting in a 50-layer ResNet (Table 1). We use option B for increasing dimensions. This model has 3.8 billion FLOPs.

101-layer and 152-layer ResNets: We construct 101-layer and 152-layer ResNets by using more 3-layer blocks (Table 1). Remarkably, although the depth is significantly increased, the 152-layer ResNet (11.3 billion FLOPs) still has *lower complexity* than VGG-16/19 nets (15.3/19.6 billion FLOPs).

The 50/101/152-layer ResNets are more accurate than the 34-layer ones by considerable margins (Table 3 and 4). We do not observe the degradation problem and thus enjoy significant accuracy gains from considerably increased depth. The benefits of depth are witnessed for all evaluation metrics (Table 3 and 4).

Comparisons with State-of-the-art Methods. In Table 4 we compare with the previous best single-model results. Our baseline 34-layer ResNets have achieved very competitive accuracy. Our 152-layer ResNet has a single-model top-5 validation error of 4.49%. This single-model result outperforms all previous ensemble results (Table 5). We combine six models of different depth to form an ensemble (only with two 152-layer ones at the time of submitting). This leads to **3.57%** top-5 error on the test set (Table 5). *This entry won the 1st place in ILSVRC 2015.*

4.2. CIFAR-10 and Analysis

We conducted more studies on the CIFAR-10 dataset [20], which consists of 50k training images and 10k testing images in 10 classes. We present experiments trained on the training set and evaluated on the test set. Our focus is on the behaviors of extremely deep networks, but not on pushing the state-of-the-art results, so we intentionally use simple architectures as follows.

The plain/residual architectures follow the form in Fig. 3 (middle/right). The network inputs are 32×32 images, with the per-pixel mean subtracted. The first layer is 3×3 convolutions. Then we use a stack of $6n$ layers with 3×3 convolutions on the feature maps of sizes $\{32, 16, 8\}$ respectively, with $2n$ layers for each feature map size. The numbers of filters are $\{16, 32, 64\}$ respectively. The subsampling is performed by convolutions with a stride of 2. The network ends with a global average pooling, a 10-way fully-connected layer, and softmax. There are totally $6n$+2 stacked weighted layers. The following table summarizes the architecture:

output map size	32×32	16×16	8×8
# layers	1+2n	2n	2n
# filters	16	32	64

When shortcut connections are used, they are connected to the pairs of 3×3 layers (totally $3n$ shortcuts). On this dataset we use identity shortcuts in all cases (*i.e.*, option A), so our residual models have exactly the same depth, width, and number of parameters as the plain counterparts.

method			error (%)
Maxout [10]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Table 6. Classification error on the **CIFAR-10** test set. All methods are with data augmentation. For ResNet-110, we run it 5 times and show "best (mean±std)" as in [43].

We use a weight decay of 0.0001 and momentum of 0.9, and adopt the weight initialization in [13] and BN [16] but with no dropout. These models are trained with a mini-batch size of 128 on two GPUs. We start with a learning rate of 0.1, divide it by 10 at 32k and 48k iterations, and terminate training at 64k iterations, which is determined on a 45k/5k train/val split. We follow the simple data augmentation in [24] for training: 4 pixels are padded on each side, and a 32×32 crop is randomly sampled from the padded image or its horizontal flip. For testing, we only evaluate the single view of the original 32×32 image.

We compare $n = \{3, 5, 7, 9\}$, leading to 20, 32, 44, and 56-layer networks. Fig. 6 (left) shows the behaviors of the plain nets. The deep plain nets suffer from increased depth, and exhibit higher training error when going deeper. This phenomenon is similar to that on ImageNet (Fig. 4, left) and on MNIST (see [42]), suggesting that such an optimization difficulty is a fundamental problem.

Fig. 6 (middle) shows the behaviors of ResNets. Also similar to the ImageNet cases (Fig. 4, right), our ResNets manage to overcome the optimization difficulty and demonstrate accuracy gains when the depth increases.

We further explore $n = 18$ that leads to a 110-layer ResNet. In this case, we find that the initial learning rate of 0.1 is slightly too large to start converging[5]. So we use 0.01 to warm up the training until the training error is below 80% (about 400 iterations), and then go back to 0.1 and continue training. The rest of the learning schedule is as done previously. This 110-layer network converges well (Fig. 6, middle). It has *fewer* parameters than other deep and thin

[5]With an initial learning rate of 0.1, it starts converging (<90% error) after several epochs, but still reaches similar accuracy.

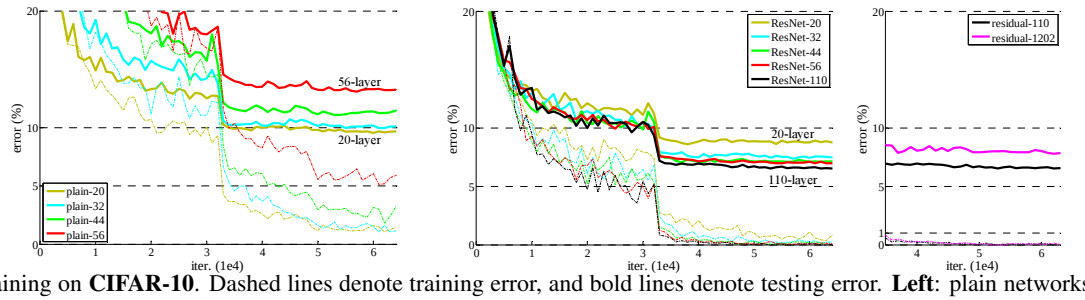


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left**: plain networks. The error of plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1202 layers.

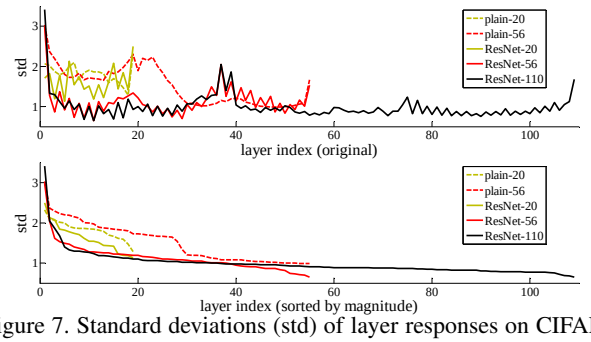


Figure 7. Standard deviations (std) of layer responses on CIFAR-10. The responses are the outputs of each 3×3 layer, after BN and before nonlinearity. **Top**: the layers are shown in their original order. **Bottom**: the responses are ranked in descending order.

networks such as FitNet [35] and Highway [42] (Table 6), yet is among the state-of-the-art results (6.43%, Table 6).

Analysis of Layer Responses. Fig. 7 shows the standard deviations (std) of the layer responses. The responses are the outputs of each 3×3 layer, after BN and before other nonlinearity (ReLU/addition). For ResNets, this analysis reveals the response strength of the residual functions. Fig. 7 shows that ResNets have generally smaller responses than their plain counterparts. These results support our basic motivation (Sec.3.1) that the residual functions might be generally closer to zero than the non-residual functions. We also notice that the deeper ResNet has smaller magnitudes of responses, as evidenced by the comparisons among ResNet-20, 56, and 110 in Fig. 7. When there are more layers, an individual layer of ResNets tends to modify the signal less.

Exploring Over 1000 layers. We explore an aggressively deep model of over 1000 layers. We set $n = 200$ that leads to a 1202-layer network, which is trained as described above. Our method shows *no optimization difficulty*, and this 10^3-layer network is able to achieve *training error* <0.1% (Fig. 6, right). Its test error is still fairly good (7.93%, Table 6).

But there are still open problems on such aggressively deep models. The testing result of this 1202-layer network is worse than that of our 110-layer network, although both have similar training error. We argue that this is because of overfitting. The 1202-layer network may be unnecessarily large (19.4M) for this small dataset. Strong regularization such as maxout [10] or dropout [14] is applied to obtain the best results ([10, 25, 24, 35]) on this dataset. In this paper, we use no maxout/dropout and just simply impose regularization via deep and thin architectures by design, without distracting from the focus on the difficulties of optimization. But combining with stronger regularization may improve results, which we will study in the future.

training data	07+12	07++12
test data	VOC 07 test	VOC 12 test
VGG-16	73.2	70.4
ResNet-101	**76.4**	**73.8**

Table 7. Object detection mAP (%) on the PASCAL VOC 2007/2012 test sets using **baseline** Faster R-CNN. See also Table 10 and 11 for better results.

metric	mAP@.5	mAP@[.5, .95]
VGG-16	41.5	21.2
ResNet-101	**48.4**	**27.2**

Table 8. Object detection mAP (%) on the COCO validation set using **baseline** Faster R-CNN. See also Table 9 for better results.

4.3. Object Detection on PASCAL and MS COCO

Our method has good generalization performance on other recognition tasks. Table 7 and 8 show the object detection baseline results on PASCAL VOC 2007 and 2012 [5] and COCO [26]. We adopt *Faster R-CNN* [32] as the detection method. Here we are interested in the improvements of replacing VGG-16 [41] with ResNet-101. The detection implementation (see appendix) of using both models is the same, so the gains can only be attributed to better networks. Most remarkably, on the challenging COCO dataset we obtain a 6.0% increase in COCO's standard metric (mAP@[.5, .95]), which is a 28% relative improvement. This gain is solely due to the learned representations.

Based on deep residual nets, we won the 1st places in several tracks in ILSVRC & COCO 2015 competitions: ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation. The details are in the appendix.

References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[2] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[3] W. L. Briggs, S. F. McCormick, et al. *A Multigrid Tutorial*. Siam, 2000.

[4] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *BMVC*, 2011.

[5] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *IJCV*, pages 303–338, 2010.

[6] S. Gidaris and N. Komodakis. Object detection via a multi-region & semantic segmentation-aware cnn model. In *ICCV*, 2015.

[7] R. Girshick. Fast R-CNN. In *ICCV*, 2015.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[9] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[10] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv:1302.4389*, 2013.

[11] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *CVPR*, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

[15] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[17] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, 33, 2011.

[18] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *TPAMI*, 2012.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[20] A. Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[21] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[23] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

[24] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. *arXiv:1409.5185*, 2014.

[25] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv:1312.4400*, 2013.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*. 2014.

[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[28] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *NIPS*, 2014.

[29] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[30] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, 2007.

[31] T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *AISTATS*, 2012.

[32] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[33] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. *arXiv:1504.06066*, 2015.

[34] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge university press, 1996.

[35] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *arXiv:1409.0575*, 2014.

[37] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120*, 2013.

[38] N. N. Schraudolph. Accelerated gradient descent by factor-centering decomposition. Technical report, 1998.

[39] N. N. Schraudolph. Centering neural network gradient factors. In *Neural Networks: Tricks of the Trade*, pages 207–226. Springer, 1998.

[40] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014.

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[42] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.

[43] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. *1507.06228*, 2015.

[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[45] R. Szeliski. Fast surface interpolation using hierarchical basis functions. *TPAMI*, 1990.

[46] R. Szeliski. Locally adapted hierarchical basis preconditioning. In *SIGGRAPH*, 2006.

[47] T. Vatanen, T. Raiko, H. Valpola, and Y. LeCun. Pushing stochastic gradient towards second-order methods–backpropagation learning with transformations in nonlinearities. In *Neural Information Processing*, 2013.

[48] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.

[49] W. Venables and B. Ripley. Modern applied statistics with s-plus. 1999.

[50] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. In *ECCV*, 2014.

A. Object Detection Baselines

In this section we introduce our detection method based on the baseline Faster R-CNN [32] system. The models are initialized by the ImageNet classification models, and then fine-tuned on the object detection data. We have experimented with ResNet-50/101 at the time of the ILSVRC & COCO 2015 detection competitions.

Unlike VGG-16 used in [32], our ResNet has no hidden fc layers. We adopt the idea of "Networks on Conv feature maps" (NoC) [33] to address this issue. We compute the full-image shared conv feature maps using those layers whose strides on the image are no greater than 16 pixels (*i.e.*, conv1, conv2_x, conv3_x, and conv4_x, totally 91 conv layers in ResNet-101; Table 1). We consider these layers as analogous to the 13 conv layers in VGG-16, and by doing so, both ResNet and VGG-16 have conv feature maps of the same total stride (16 pixels). These layers are shared by a region proposal network (RPN, generating 300 proposals) [32] and a Fast R-CNN detection network [7]. RoI pooling [7] is performed before conv5_1. On this RoI-pooled feature, all layers of conv5_x and up are adopted for each region, playing the roles of VGG-16's fc layers. The final classification layer is replaced by two sibling layers (classification and box regression [7]).

For the usage of BN layers, after pre-training, we compute the BN statistics (means and variances) for each layer on the ImageNet training set. Then the BN layers are fixed during fine-tuning for object detection. As such, the BN layers become linear activations with constant offsets and scales, and BN statistics are not updated by fine-tuning. We fix the BN layers mainly for reducing memory consumption in Faster R-CNN training.

PASCAL VOC

Following [7, 32], for the PASCAL VOC 2007 *test* set, we use the 5k *trainval* images in VOC 2007 and 16k *trainval* images in VOC 2012 for training ("07+12"). For the PASCAL VOC 2012 *test* set, we use the 10k *trainval+test* images in VOC 2007 and 16k *trainval* images in VOC 2012 for training ("07++12"). The hyper-parameters for training Faster R-CNN are the same as in [32]. Table 7 shows the results. ResNet-101 improves the mAP by >3% over VGG-16. This gain is solely because of the improved features learned by ResNet.

MS COCO

The MS COCO dataset [26] involves 80 object categories. We evaluate the PASCAL VOC metric (mAP @ IoU = 0.5) and the standard COCO metric (mAP @ IoU = .5:.05:.95). We use the 80k images on the train set for training and the 40k images on the val set for evaluation. Our detection system for COCO is similar to that for PASCAL VOC. We train the COCO models with an 8-GPU implementation, and thus the RPN step has a mini-batch size of 8 images (*i.e.*, 1 per GPU) and the Fast R-CNN step has a mini-batch size of 16 images. The RPN step and Fast R-CNN step are both trained for 240k iterations with a learning rate of 0.001 and then for 80k iterations with 0.0001.

Table 8 shows the results on the MS COCO validation set. ResNet-101 has a 6% increase of mAP@[.5, .95] over VGG-16, which is a 28% relative improvement, solely contributed by the features learned by the better network. Remarkably, the mAP@[.5, .95]'s absolute increase (6.0%) is nearly as big as mAP@.5's (6.9%). This suggests that a deeper network can improve both recognition and localization.

B. Object Detection Improvements

For completeness, we report the improvements made for the competitions. These improvements are based on deep features and thus should benefit from residual learning.

MS COCO

Box refinement. Our box refinement partially follows the iterative localization in [6]. In Faster R-CNN, the final output is a regressed box that is different from its proposal box. So for inference, we pool a new feature from the regressed box and obtain a new classification score and a new regressed box. We combine these 300 new predictions with the original 300 predictions. Non-maximum suppression (NMS) is applied on the union set of predicted boxes using an IoU threshold of 0.3 [8], followed by box voting [6]. Box refinement improves mAP by about 2 points (Table 9).

Global context. We combine global context in the Fast R-CNN step. Given the full-image conv feature map, we pool a feature by global Spatial Pyramid Pooling [12] (with a "single-level" pyramid) which can be implemented as "RoI" pooling using the entire image's bounding box as the RoI. This pooled feature is fed into the post-RoI layers to obtain a global context feature. This global feature is concatenated with the original per-region feature, followed by the sibling classification and box regression layers. This new structure is trained end-to-end. Global context improves mAP@.5 by about 1 point (Table 9).

Multi-scale testing. In the above, all results are obtained by single-scale training/testing as in [32], where the image's shorter side is $s = 600$ pixels. Multi-scale training/testing has been developed in [12, 7] by selecting a scale from a feature pyramid, and in [33] by using maxout layers. In our current implementation, we have performed multi-scale *testing* following [33]; we have not performed multi-scale training because of limited time. In addition, we have performed multi-scale testing only for the Fast R-CNN step (but not yet for the RPN step). With a trained model, we compute conv feature maps on an image pyramid, where the image's shorter sides are $s \in \{200, 400, 600, 800, 1000\}$.

training data	COCO train		COCO trainval	
test data	COCO val		COCO test-dev	
mAP	@.5	@[.5, .95]	@.5	@[.5, .95]
baseline Faster R-CNN (VGG-16)	41.5	21.2		
baseline Faster R-CNN (ResNet-101)	48.4	27.2		
+box refinement	49.9	29.9		
+context	51.1	30.0	53.3	32.2
+multi-scale testing	53.8	32.5	**55.7**	**34.9**
ensemble			**59.0**	**37.4**

Table 9. Object detection improvements on MS COCO using Faster R-CNN and ResNet-101.

system	net	data	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
baseline	VGG-16	07+12	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
baseline	ResNet-101	07+12	76.4	79.8	80.7	76.2	68.3	55.9	85.1	85.3	**89.8**	56.7	87.8	69.4	88.3	88.9	80.9	78.4	41.7	78.6	79.8	85.3	72.0
baseline+++	ResNet-101	COCO+07+12	**85.6**	**90.0**	**89.6**	**87.8**	**80.8**	**76.1**	**89.9**	**89.9**	89.6	**75.5**	**90.0**	**80.7**	**89.6**	**90.3**	**89.1**	**88.7**	**65.4**	**88.1**	**85.6**	**89.0**	**86.8**

Table 10. Detection results on the PASCAL VOC 2007 test set. The baseline is the Faster R-CNN system. The system "baseline+++" include box refinement, context, and multi-scale testing in Table 9.

system	net	data	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
baseline	VGG-16	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
baseline	ResNet-101	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
baseline+++	ResNet-101	COCO+07++12	**83.8**	**92.1**	**88.4**	**84.8**	**75.9**	**71.4**	**86.3**	**87.8**	**94.2**	**66.8**	**89.4**	**69.2**	**93.9**	**91.9**	**90.9**	**89.6**	**67.9**	**88.2**	**76.8**	**90.3**	**80.0**

Table 11. Detection results on the PASCAL VOC 2012 test set (`http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11&compid=4`). The baseline is the Faster R-CNN system. The system "baseline+++" include box refinement, context, and multi-scale testing in Table 9.

We select two adjacent scales from the pyramid following [33]. RoI pooling and subsequent layers are performed on the feature maps of these two scales [33], which are merged by maxout as in [33]. Multi-scale testing improves the mAP by over 2 points (Table 9).

Using validation data. Next we use the 80k+40k trainval set for training and the 20k test-dev set for evaluation. The test-dev set has no publicly available ground truth and the result is reported by the evaluation server. Under this setting, the results are an mAP@.5 of 55.7% and an mAP@[.5, .95] of 34.9% (Table 9). This is our single-model result.

Ensemble. In Faster R-CNN, the system is designed to learn region proposals and also object classifiers, so an ensemble can be used to boost both tasks. We use an ensemble for proposing regions, and the union set of proposals are processed by an ensemble of per-region classifiers. Table 9 shows our result based on an ensemble of 3 networks. The mAP is 59.0% and 37.4% on the test-dev set. *This result won the 1st place in the detection task in COCO 2015.*

PASCAL VOC

We revisit the PASCAL VOC dataset based on the above model. With the single model on the COCO dataset (55.7% mAP@.5 in Table 9), we fine-tune this model on the PASCAL VOC sets. The improvements of box refinement, context, and multi-scale testing are also adopted. By doing so we achieve 85.6% mAP on PASCAL VOC 2007 (Table 10) and 83.8% on PASCAL VOC 2012 (Table 11)[6]. The result on PASCAL VOC 2012 is 10 points higher than the previous state-of-the-art result [6].

	val2	test
GoogLeNet [44] (ILSVRC'14)	-	43.9
our single model (ILSVRC'15)	60.5	58.8
our ensemble (ILSVRC'15)	**63.6**	**62.1**

Table 12. Our results (mAP, %) on the ImageNet detection dataset. Our detection system is Faster R-CNN [32] with the improvements in Table 9, using ResNet-101.

ImageNet Detection

The ImageNet Detection (DET) task involves 200 object categories. The accuracy is evaluated by mAP@.5. Our object detection algorithm for ImageNet DET is the same as that for MS COCO in Table 9. The networks are pre-trained on the 1000-class ImageNet classification set, and are fine-tuned on the DET data. We split the validation set into two parts (val1/val2) following [8]. We fine-tune the detection models using the DET training set and the val1 set. The val2 set is used for validation. We do not use other ILSVRC 2015 data. Our single model with ResNet-101 has

[6] `http://host.robots.ox.ac.uk:8080/anonymous/3OJ4OJ.html`, submitted on 2015-11-26.

LOC method	LOC network	testing	LOC error on GT CLS	classification network	top-5 LOC error on predicted CLS
VGG's [41]	VGG-16	1-crop	33.1 [41]		
RPN	ResNet-101	1-crop	13.3		
RPN	ResNet-101	dense	11.7		
RPN	ResNet-101	dense		ResNet-101	14.4
RPN+RCNN	ResNet-101	dense		ResNet-101	**10.6**
RPN+RCNN	ensemble	dense		ensemble	**8.9**

Table 13. Localization error (%) on the ImageNet validation. In the column of "LOC error on GT class" ([41]), the ground truth class is used. In the "testing" column, "1-crop" denotes testing on a center crop of 224×224 pixels, "dense" denotes dense (fully convolutional) and multi-scale testing.

58.8% mAP and our ensemble of 3 models has 62.1% mAP on the DET test set (Table 12). *This result won the 1st place in the ImageNet detection task in ILSVRC 2015*, surpassing the second place by **8.5 points** (absolute).

C. ImageNet Localization

The ImageNet Localization (LOC) task [36] requires to classify and localize the objects. Following [40, 41], we assume that the image-level classifiers are first adopted for predicting the class labels of an image, and the localization algorithm only accounts for predicting bounding boxes based on the predicted classes. We adopt the "per-class regression" (PCR) strategy [40, 41], learning a bounding box regressor for each class. We pre-train the networks for ImageNet classification and then fine-tune them for localization. We train networks on the provided 1000-class ImageNet training set.

Our localization algorithm is based on the RPN framework of [32] with a few modifications. Unlike the way in [32] that is category-agnostic, our RPN for localization is designed in a *per-class* form. This RPN ends with two sibling 1×1 convolutional layers for binary classification (*cls*) and box regression (*reg*), as in [32]. The *cls* and *reg* layers are both in a *per-class* from, in contrast to [32]. Specifically, the *cls* layer has a 1000-d output, and each dimension is *binary logistic regression* for predicting being or not being an object class; the *reg* layer has a 1000×4-d output consisting of box regressors for 1000 classes. As in [32], our bounding box regression is with reference to multiple translation-invariant "anchor" boxes at each position.

As in our ImageNet classification training (Sec. 3.4), we randomly sample 224×224 crops for data augmentation. We use a mini-batch size of 256 images for fine-tuning. To avoid negative samples being dominate, 8 anchors are randomly sampled for each image, where the sampled positive and negative anchors have a ratio of 1:1 [32]. For testing, the network is applied on the image fully-convolutionally.

Table 13 compares the localization results. Following [41], we first perform "oracle" testing using the ground truth class as the classification prediction. VGG's paper [41] reports a center-crop error of 33.1% (Table 13) using ground truth classes. Under the same setting, our RPN method using ResNet-101 net significantly reduces the center-crop error to 13.3%. This comparison demonstrates the excellent performance of our framework. With dense (fully convolutional) and multi-scale testing, our ResNet-101 has an error of 11.7% using ground truth classes. Using ResNet-101 for predicting classes (4.6% top-5 classification error, Table 4), the top-5 localization error is 14.4%.

method	top-5 localization err	
	val	test
OverFeat [40] (ILSVRC'13)	30.0	29.9
GoogLeNet [44] (ILSVRC'14)	-	26.7
VGG [41] (ILSVRC'14)	26.9	25.3
ours (ILSVRC'15)	**8.9**	**9.0**

Table 14. Comparisons of localization error (%) on the ImageNet dataset with state-of-the-art methods.

The above results are only based on the *proposal network* (RPN) in Faster R-CNN [32]. One may use the *detection network* (Fast R-CNN [7]) in Faster R-CNN to improve the results. But we notice that on this dataset, one image usually contains a single dominate object, and the proposal regions highly overlap with each other and thus have very similar RoI-pooled features. As a result, the image-centric training of Fast R-CNN [7] generates samples of small variations, which may not be desired for stochastic training. Motivated by this, in our current experiment we use the original R-CNN [8] that is RoI-centric, in place of Fast R-CNN.

Our R-CNN implementation is as follows. We apply the per-class RPN trained as above on the training images to predict bounding boxes for the ground truth class. These predicted boxes play a role of class-dependent proposals. For each training image, the highest scored 200 proposals are extracted as training samples to train an R-CNN classifier. The image region is cropped from a proposal, warped to 224×224 pixels, and fed into the classification network as in R-CNN [8]. The outputs of this network consist of two sibling fc layers for *cls* and *reg*, also in a per-class form. This R-CNN network is fine-tuned on the training set using a mini-batch size of 256 in the RoI-centric fashion. For testing, the RPN generates the highest scored 200 proposals for each predicted class, and the R-CNN network is used to update these proposals' scores and box positions.

This method reduces the top-5 localization error to 10.6% (Table 13). This is our single-model result on the validation set. Using an ensemble of networks for both classification and localization, we achieve a top-5 localization error of 9.0% on the test set. This number significantly outperforms the ILSVRC 14 results (Table 14), showing a 64% relative reduction of error. *This result won the 1st place in the ImageNet localization task in ILSVRC 2015.*

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

1 Introduction

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labeled images were relatively small — on the order of tens of thousands of images (e.g., NORB [16], Caltech-101/256 [8, 9], and CIFAR-10/100 [12]). Simple recognition tasks can be solved quite well with datasets of this size, especially if they are augmented with label-preserving transformations. For example, the current-best error rate on the MNIST digit-recognition task (<0.3%) approaches human performance [4]. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets. And indeed, the shortcomings of small image datasets have been widely recognized (e.g., Pinto et al. [21]), but it has only recently become possible to collect labeled datasets with millions of images. The new larger datasets include LabelMe [23], which consists of hundreds of thousands of fully-segmented images, and ImageNet [6], which consists of over 15 million labeled high-resolution images in over 22,000 categories.

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don't have. Convolutional neural networks (CNNs) constitute one such class of models [16, 11, 13, 18, 15, 22, 26]. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Despite the attractive qualities of CNNs, and despite the relative efficiency of their local architecture, they have still been prohibitively expensive to apply in large scale to high-resolution images. Luckily, current GPUs, paired with a highly-optimized implementation of 2D convolution, are powerful enough to facilitate the training of interestingly-large CNNs, and recent datasets such as ImageNet contain enough labeled examples to train such models without severe overfitting.

The specific contributions of this paper are as follows: we trained one of the largest convolutional neural networks to date on the subsets of ImageNet used in the ILSVRC-2010 and ILSVRC-2012 competitions [2] and achieved by far the best results ever reported on these datasets. We wrote a highly-optimized GPU implementation of 2D convolution and all the other operations inherent in training convolutional neural networks, which we make available publicly[1]. Our network contains a number of new and unusual features which improve its performance and reduce its training time, which are detailed in Section 3. The size of our network made overfitting a significant problem, even with 1.2 million labeled training examples, so we used several effective techniques for preventing overfitting, which are described in Section 4. Our final network contains five convolutional and three fully-connected layers, and this depth seems to be important: we found that removing any convolutional layer (each of which contains no more than 1% of the model's parameters) resulted in inferior performance.

In the end, the network's size is limited mainly by the amount of memory available on current GPUs and by the amount of training time that we are willing to tolerate. Our network takes between five and six days to train on two GTX 580 3GB GPUs. All of our experiments suggest that our results can be improved simply by waiting for faster GPUs and bigger datasets to become available.

2 The Dataset

ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories. The images were collected from the web and labeled by human labelers using Amazon's Mechanical Turk crowd-sourcing tool. Starting in 2010, as part of the Pascal Visual Object Challenge, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held. ILSVRC uses a subset of ImageNet with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images.

ILSVRC-2010 is the only version of ILSVRC for which the test set labels are available, so this is the version on which we performed most of our experiments. Since we also entered our model in the ILSVRC-2012 competition, in Section 6 we report our results on this version of the dataset as well, for which test set labels are unavailable. On ImageNet, it is customary to report two error rates: top-1 and top-5, where the top-5 error rate is the fraction of test images for which the correct label is not among the five labels considered most probable by the model.

ImageNet consists of variable-resolution images, while our system requires a constant input dimensionality. Therefore, we down-sampled the images to a fixed resolution of 256×256. Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image. We did not pre-process the images in any other way, except for subtracting the mean activity over the training set from each pixel. So we trained our network on the (centered) raw RGB values of the pixels.

3 The Architecture

The architecture of our network is summarized in Figure 2. It contains eight learned layers — five convolutional and three fully-connected. Below, we describe some of the novel or unusual features of our network's architecture. Sections 3.1-3.4 are sorted according to our estimation of their importance, with the most important first.

[1] http://code.google.com/p/cuda-convnet/

3.1 ReLU Nonlinearity

The standard way to model a neuron's output f as a function of its input x is with $f(x) = \tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity $f(x) = \max(0, x)$. Following Nair and Hinton [20], we refer to neurons with this nonlinearity as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with $\tanh$ units. This is demonstrated in Figure 1, which shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network. This plot shows that we would not have been able to experiment with such large neural networks for this work if we had used traditional saturating neuron models.

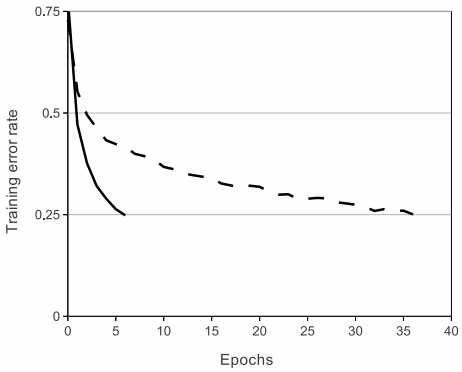


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with $\tanh$ neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

We are not the first to consider alternatives to traditional neuron models in CNNs. For example, Jarrett et al. [11] claim that the nonlinearity $f(x) = |\tanh(x)|$ works particularly well with their type of contrast normalization followed by local average pooling on the Caltech-101 dataset. However, on this dataset the primary concern is preventing overfitting, so the effect they are observing is different from the accelerated ability to fit the training set which we report when using ReLUs. Faster learning has a great influence on the performance of large models trained on large datasets.

3.2 Training on Multiple GPUs

A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it. It turns out that 1.2 million training examples are enough to train networks which are too big to fit on one GPU. Therefore we spread the net across two GPUs. Current GPUs are particularly well-suited to cross-GPU parallelization, as they are able to read from and write to one another's memory directly, without going through host machine memory. The parallelization scheme that we employ essentially puts half of the kernels (or neurons) on each GPU, with one additional trick: the GPUs communicate only in certain layers. This means that, for example, the kernels of layer 3 take input from all kernel maps in layer 2. However, kernels in layer 4 take input only from those kernel maps in layer 3 which reside on the same GPU. Choosing the pattern of connectivity is a problem for cross-validation, but this allows us to precisely tune the amount of communication until it is an acceptable fraction of the amount of computation.

The resultant architecture is somewhat similar to that of the "columnar" CNN employed by Cireşan et al. [5], except that our columns are not independent (see Figure 2). This scheme reduces our top-1 and top-5 error rates by 1.7% and 1.2%, respectively, as compared with a net with half as many kernels in each convolutional layer trained on one GPU. The two-GPU net takes slightly less time to train than the one-GPU net[2].

[2]The one-GPU net actually has the same number of kernels as the two-GPU net in the final convolutional layer. This is because most of the net's parameters are in the first fully-connected layer, which takes the last convolutional layer as input. So to make the two nets have approximately the same number of parameters, we did not halve the size of the final convolutional layer (nor the fully-conneced layers which follow). Therefore this comparison is biased in favor of the one-GPU net, since it is bigger than "half the size" of the two-GPU net.

3.3 Local Response Normalization

ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, we still find that the following local normalization scheme aids generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

where the sum runs over n "adjacent" kernel maps at the same spatial position, and N is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants k, n, α, and β are hyper-parameters whose values are determined using a validation set; we used $k = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$. We applied this normalization after applying the ReLU nonlinearity in certain layers (see Section 3.5).

This scheme bears some resemblance to the local contrast normalization scheme of Jarrett et al. [11], but ours would be more correctly termed "brightness normalization", since we do not subtract the mean activity. Response normalization reduces our top-1 and top-5 error rates by 1.4% and 1.2%, respectively. We also verified the effectiveness of this scheme on the CIFAR-10 dataset: a four-layer CNN achieved a 13% test error rate without normalization and 11% with normalization[3].

3.4 Overlapping Pooling

Pooling layers in CNNs summarize the outputs of neighboring groups of neurons in the same kernel map. Traditionally, the neighborhoods summarized by adjacent pooling units do not overlap (e.g., [17, 11, 4]). To be more precise, a pooling layer can be thought of as consisting of a grid of pooling units spaced s pixels apart, each summarizing a neighborhood of size $z \times z$ centered at the location of the pooling unit. If we set $s = z$, we obtain traditional local pooling as commonly employed in CNNs. If we set $s < z$, we obtain overlapping pooling. This is what we use throughout our network, with $s = 2$ and $z = 3$. This scheme reduces the top-1 and top-5 error rates by 0.4% and 0.3%, respectively, as compared with the non-overlapping scheme $s = 2, z = 2$, which produces output of equivalent dimensions. We generally observe during training that models with overlapping pooling find it slightly more difficult to overfit.

3.5 Overall Architecture

Now we are ready to describe the overall architecture of our CNN. As depicted in Figure 2, the net contains eight layers with weights; the first five are convolutional and the remaining three are fully-connected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels. Our network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction distribution.

The kernels of the second, fourth, and fifth convolutional layers are connected only to those kernel maps in the previous layer which reside on the same GPU (see Figure 2). The kernels of the third convolutional layer are connected to all kernel maps in the second layer. The neurons in the fully-connected layers are connected to all neurons in the previous layer. Response-normalization layers follow the first and second convolutional layers. Max-pooling layers, of the kind described in Section 3.4, follow both response-normalization layers as well as the fifth convolutional layer. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

The first convolutional layer filters the $224 \times 224 \times 3$ input image with 96 kernels of size $11 \times 11 \times 3$ with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring

[3]We cannot describe this network in detail due to space constraints, but it is specified precisely by the code and parameter files provided here: http://code.google.com/p/cuda-convnet/.

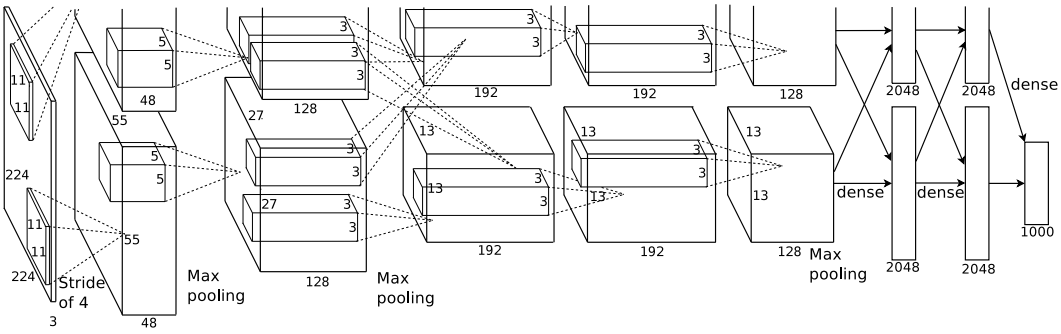


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random 224×224 patches (and their horizontal reflections) from the 256×256 images and training our network on these extracted patches[4]. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five 224×224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components,

[4]This is the reason why the input images in Figure 2 are $224 \times 224 \times 3$-dimensional.

with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1. Therefore to each RGB image pixel $I_{xy} = [I^R_{xy}, I^G_{xy}, I^B_{xy}]^T$ we add the following quantity:

$$[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

where $\mathbf{p}_i$ and λ_i are ith eigenvector and eigenvalue of the 3×3 covariance matrix of RGB pixel values, respectively, and α_i is the aforementioned random variable. Each α_i is drawn only once for all the pixels of a particular training image until that image is used for training again, at which point it is re-drawn. This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.

4.2 Dropout

Combining the predictions of many different models is a very successful way to reduce test errors [1, 3], but it appears to be too expensive for big neural networks that already take several days to train. There is, however, a very efficient version of model combination that only costs about a factor of two during training. The recently-introduced technique, called "dropout" [10], consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in back-propagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. At test time, we use all the neurons but multiply their outputs by 0.5, which is a reasonable approximation to taking the geometric mean of the predictive distributions produced by the exponentially-many dropout networks.

We use dropout in the first two fully-connected layers of Figure 2. Without dropout, our network exhibits substantial overfitting. Dropout roughly doubles the number of iterations required to converge.

5 Details of learning

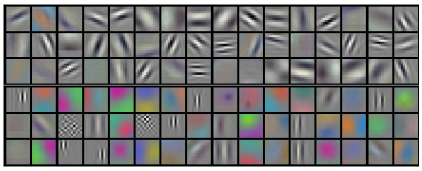


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

We trained our models using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005. We found that this small amount of weight decay was important for the model to learn. In other words, weight decay here is not merely a regularizer: it reduces the model's training error. The update rule for weight w was

$$\begin{aligned} v_{i+1} &:= 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i} \\ w_{i+1} &:= w_i + v_{i+1} \end{aligned}$$

where i is the iteration index, v is the momentum variable, ϵ is the learning rate, and $\left\langle \frac{\partial L}{\partial w}\big|_{w_i} \right\rangle_{D_i}$ is the average over the ith batch D_i of the derivative of the objective with respect to w, evaluated at w_i.

We initialized the weights in each layer from a zero-mean Gaussian distribution with standard deviation 0.01. We initialized the neuron biases in the second, fourth, and fifth convolutional layers, as well as in the fully-connected hidden layers, with the constant 1. This initialization accelerates the early stages of learning by providing the ReLUs with positive inputs. We initialized the neuron biases in the remaining layers with the constant 0.

We used an equal learning rate for all layers, which we adjusted manually throughout training. The heuristic which we followed was to divide the learning rate by 10 when the validation error rate stopped improving with the current learning rate. The learning rate was initialized at 0.01 and

reduced three times prior to termination. We trained the network for roughly 90 cycles through the training set of 1.2 million images, which took five to six days on two NVIDIA GTX 580 3GB GPUs.

6 Results

Our results on ILSVRC-2010 are summarized in Table 1. Our network achieves top-1 and top-5 test set error rates of **37.5%** and **17.0%**[5]. The best performance achieved during the ILSVRC-2010 competition was 47.1% and 28.2% with an approach that averages the predictions produced from six sparse-coding models trained on different features [2], and since then the best published results are 45.7% and 25.7% with an approach that averages the predictions of two classifiers trained on Fisher Vectors (FVs) computed from two types of densely-sampled features [24].

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

We also entered our model in the ILSVRC-2012 competition and report our results in Table 2. Since the ILSVRC-2012 test set labels are not publicly available, we cannot report test error rates for all the models that we tried. In the remainder of this paragraph, we use validation and test error rates interchangeably because in our experience they do not differ by more than 0.1% (see Table 2). The CNN described in this paper achieves a top-5 error rate of 18.2%. Averaging the predictions of five similar CNNs gives an error rate of 16.4%. Training one CNN, with an extra sixth convolutional layer over the last pooling layer, to classify the entire ImageNet Fall 2011 release (15M images, 22K categories), and then "fine-tuning" it on ILSVRC-2012 gives an error rate of 16.6%. Averaging the predictions of two CNNs that were pre-trained on the entire Fall 2011 release with the aforementioned five CNNs gives an error rate of **15.3%**. The second-best contest entry achieved an error rate of 26.2% with an approach that averages the predictions of several classifiers trained on FVs computed from different types of densely-sampled features [7].

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk* were "pre-trained" to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

Finally, we also report our error rates on the Fall 2009 version of ImageNet with 10,184 categories and 8.9 million images. On this dataset we follow the convention in the literature of using half of the images for training and half for testing. Since there is no established test set, our split necessarily differs from the splits used by previous authors, but this does not affect the results appreciably. Our top-1 and top-5 error rates on this dataset are **67.4%** and **40.9%**, attained by the net described above but with an additional, sixth convolutional layer over the last pooling layer. The best published results on this dataset are 78.1% and 60.9% [19].

6.1 Qualitative Evaluations

Figure 3 shows the convolutional kernels learned by the network's two data-connected layers. The network has learned a variety of frequency- and orientation-selective kernels, as well as various colored blobs. Notice the specialization exhibited by the two GPUs, a result of the restricted connectivity described in Section 3.5. The kernels on GPU 1 are largely color-agnostic, while the kernels on on GPU 2 are largely color-specific. This kind of specialization occurs during every run and is independent of any particular random weight initialization (modulo a renumbering of the GPUs).

[5]The error rates without averaging predictions over ten patches as described in Section 4.1 are 39.0% and 18.3%.

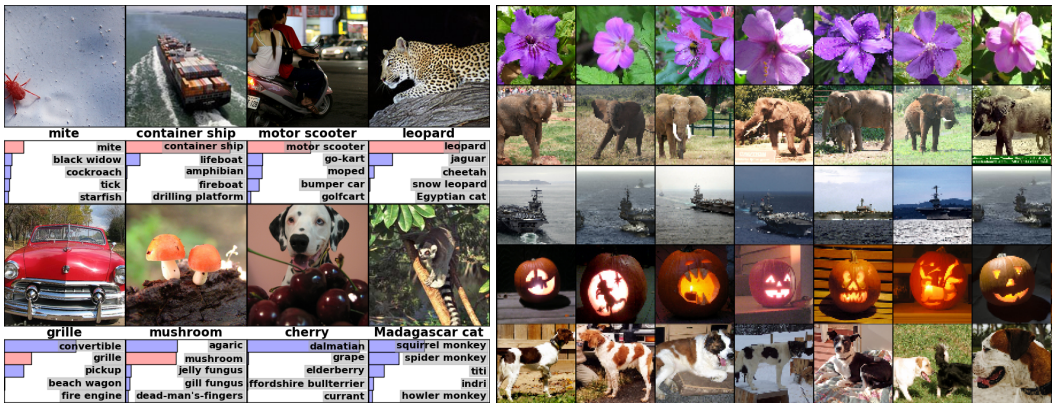


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

In the left panel of Figure 4 we qualitatively assess what the network has learned by computing its top-5 predictions on eight test images. Notice that even off-center objects, such as the mite in the top-left, can be recognized by the net. Most of the top-5 labels appear reasonable. For example, only other types of cat are considered plausible labels for the leopard. In some cases (grille, cherry) there is genuine ambiguity about the intended focus of the photograph.

Another way to probe the network's visual knowledge is to consider the feature activations induced by an image at the last, 4096-dimensional hidden layer. If two images produce feature activation vectors with a small Euclidean separation, we can say that the higher levels of the neural network consider them to be similar. Figure 4 shows five images from the test set and the six images from the training set that are most similar to each of them according to this measure. Notice that at the pixel level, the retrieved training images are generally not close in L2 to the query images in the first column. For example, the retrieved dogs and elephants appear in a variety of poses. We present the results for many more test images in the supplementary material.

Computing similarity by using Euclidean distance between two 4096-dimensional, real-valued vectors is inefficient, but it could be made efficient by training an auto-encoder to compress these vectors to short binary codes. This should produce a much better image retrieval method than applying auto-encoders to the raw pixels [14], which does not make use of image labels and hence has a tendency to retrieve images with similar patterns of edges, whether or not they are semantically similar.

7 Discussion

Our results show that a large, deep convolutional neural network is capable of achieving record-breaking results on a highly challenging dataset using purely supervised learning. It is notable that our network's performance degrades if a single convolutional layer is removed. For example, removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So the depth really is important for achieving our results.

To simplify our experiments, we did not use any unsupervised pre-training even though we expect that it will help, especially if we obtain enough computational power to significantly increase the size of the network without obtaining a corresponding increase in the amount of labeled data. Thus far, our results have improved as we have made our network larger and trained it longer but we still have many orders of magnitude to go in order to match the infero-temporal pathway of the human visual system. Ultimately we would like to use very large and deep convolutional nets on video sequences where the temporal structure provides very helpful information that is missing or far less obvious in static images.

References

[1] R.M. Bell and Y. Koren. Lessons from the netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.

[2] A. Berg, J. Deng, and L. Fei-Fei. Large scale visual recognition challenge 2010. www.image-net.org/challenges. 2010.

[3] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] D. Cireşan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *Arxiv preprint arXiv:1202.2745*, 2012.

[5] D.C. Cireşan, U. Meier, J. Masci, L.M. Gambardella, and J. Schmidhuber. High-performance neural networks for visual object classification. *Arxiv preprint arXiv:1102.0183*, 2011.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[7] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L. Fei-Fei. *ILSVRC-2012*, 2012. URL `http://www.image-net.org/challenges/LSVRC/2012/`.

[8] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[9] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007. URL `http://authors.library.caltech.edu/7694`.

[10] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[11] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *International Conference on Computer Vision*, pages 2146–2153. IEEE, 2009.

[12] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, 2009.

[13] A. Krizhevsky. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 2010.

[14] A. Krizhevsky and G.E. Hinton. Using very deep autoencoders for content-based image retrieval. In *ESANN*, 2011.

[15] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, et al. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, 1990.

[16] Y. LeCun, F.J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–97. IEEE, 2004.

[17] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE, 2010.

[18] H. Lee, R. Grosse, R. Ranganath, and A.Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[19] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Metric Learning for Large Scale Image Classification: Generalizing to New Classes at Near-Zero Cost. In *ECCV - European Conference on Computer Vision*, Florence, Italy, October 2012.

[20] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. 27th International Conference on Machine Learning*, 2010.

[21] N. Pinto, D.D. Cox, and J.J. DiCarlo. Why is real-world visual object recognition hard? *PLoS computational biology*, 4(1):e27, 2008.

[22] N. Pinto, D. Doukhan, J.J. DiCarlo, and D.D. Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS computational biology*, 5(11):e1000579, 2009.

[23] B.C. Russell, A. Torralba, K.P. Murphy, and W.T. Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1):157–173, 2008.

[24] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1665–1672. IEEE, 2011.

[25] P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, volume 2, pages 958–962, 2003.

[26] S.C. Turaga, J.F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, and H.S. Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation*, 22(2):511–538, 2010.

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

[*]Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

[†]Work performed while at Google Brain.

[‡]Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

transduction problems such as language modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15].

Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [21] and conditional computation [32], while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.

Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences [2, 19]. In all but a few cases [27], however, such attention mechanisms are used in conjunction with a recurrent network.

In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.

2 Background

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU [16], ByteNet [18] and ConvS2S [9], all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions [12]. In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention as described in section 3.2.

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 27, 28, 22].

End-to-end memory networks are based on a recurrent attention mechanism instead of sequence-aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks [34].

To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. In the following sections, we will describe the Transformer, motivate self-attention and discuss its advantages over models such as [17, 18] and [9].

3 Model Architecture

Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 35]. Here, the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $\mathbf{z} = (z_1, ..., z_n)$. Given $\mathbf{z}$, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive [10], consuming the previously generated symbols as additional input when generating the next.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.

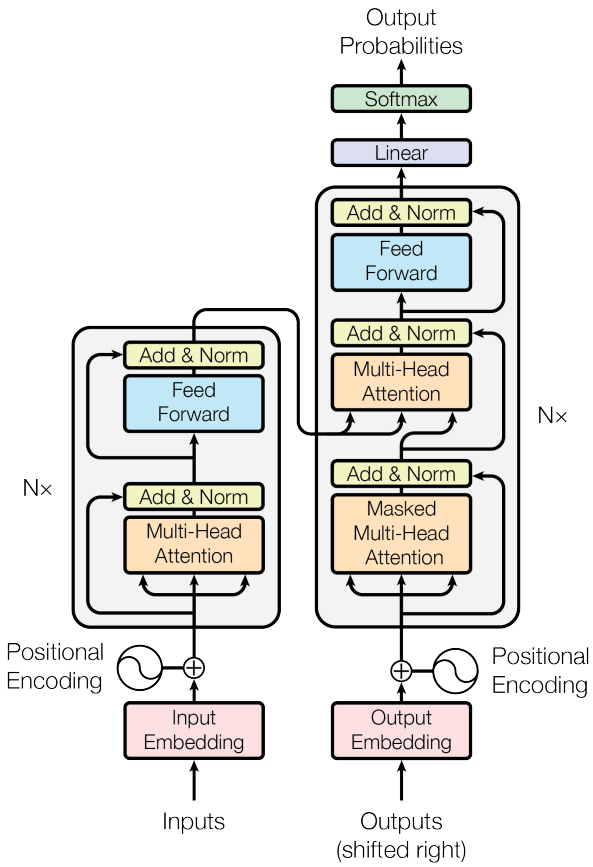


Figure 1: The Transformer - model architecture.

3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, where $\mathrm{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

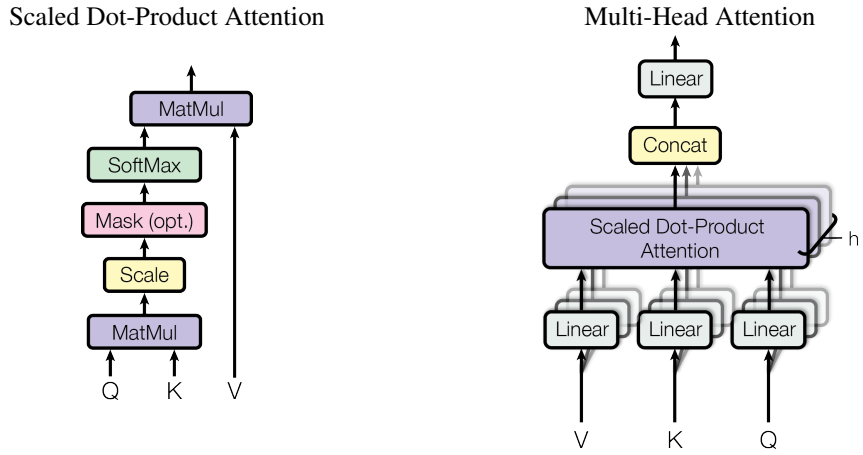


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

3.2.1 Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

3.2.2 Multi-Head Attention

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

[4] To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O \\ \text{where } \text{head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_\text{model}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

3.2.3 Applications of Attention in our Model

The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

3.3 Position-wise Feed-Forward Networks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_\text{model} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

3.4 Embeddings and Softmax

Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_model. We also use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation, similar to [30]. In the embedding layers, we multiply those weights by $\sqrt{d_\text{model}}$.

3.5 Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

4 Why Self-Attention

In this section we compare various aspects of self-attention layers to the recurrent and convolutional layers commonly used for mapping one variable-length sequence of symbol representations $(x_1, ..., x_n)$ to another sequence of equal length $(z_1, ..., z_n)$, with $x_i, z_i \in \mathbb{R}^d$, such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention we consider three desiderata.

One is the total computational complexity per layer. Another is the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required.

The third is the path length between long-range dependencies in the network. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies [12]. Hence we also compare the maximum path length between any two input and output positions in networks composed of the different layer types.

As noted in Table 1, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires $O(n)$ sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality d, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece [38] and byte-pair [31] representations. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size r in

the input sequence centered around the respective output position. This would increase the maximum path length to $O(n/r)$. We plan to investigate this approach further in future work.

A single convolutional layer with kernel width $k < n$ does not connect all pairs of input and output positions. Doing so requires a stack of $O(n/k)$ convolutional layers in the case of contiguous kernels, or $O(log_k(n))$ in the case of dilated convolutions [18], increasing the length of the longest paths between any two positions in the network. Convolutional layers are generally more expensive than recurrent layers, by a factor of k. Separable convolutions [6], however, decrease the complexity considerably, to $O(k \cdot n \cdot d + n \cdot d^2)$. Even with $k = n$, however, the complexity of a separable convolution is equal to the combination of a self-attention layer and a point-wise feed-forward layer, the approach we take in our model.

As side benefit, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

We trained on the standard WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs. Sentences were encoded using byte-pair encoding [3], which has a shared source-target vocabulary of about 37000 tokens. For English-French, we used the significantly larger WMT 2014 English-French dataset consisting of 36M sentences and split tokens into a 32000 word-piece vocabulary [38]. Sentence pairs were batched together by approximate sequence length. Each training batch contained a set of sentence pairs containing approximately 25000 source tokens and 25000 target tokens.

5.2 Hardware and Schedule

We trained our models on one machine with 8 NVIDIA P100 GPUs. For our base models using the hyperparameters described throughout the paper, each training step took about 0.4 seconds. We trained the base models for a total of 100,000 steps or 12 hours. For our big models,(described on the bottom line of table 3), step time was 1.0 seconds. The big models were trained for 300,000 steps (3.5 days).

5.3 Optimizer

We used the Adam optimizer [20] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{3}$$

This corresponds to increasing the learning rate linearly for the first $warmup\_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup\_steps = 4000$.

5.4 Regularization

We employ three types of regularization during training:

Residual Dropout We apply dropout [33] to the output of each sub-layer, before it is added to the sub-layer input and normalized. In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of $P_{drop} = 0.1$.

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Label Smoothing During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ [36]. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

6 Results

6.1 Machine Translation

On the WMT 2014 English-to-German translation task, the big transformer model (Transformer (big) in Table 2) outperforms the best previously reported models (including ensembles) by more than 2.0 BLEU, establishing a new state-of-the-art BLEU score of 28.4. The configuration of this model is listed in the bottom line of Table 3. Training took 3.5 days on 8 P100 GPUs. Even our base model surpasses all previously published models and ensembles, at a fraction of the training cost of any of the competitive models.

On the WMT 2014 English-to-French translation task, our big model achieves a BLEU score of 41.0, outperforming all of the previously published single models, at less than $1/4$ the training cost of the previous state-of-the-art model. The Transformer (big) model trained for English-to-French used dropout rate $P_{drop} = 0.1$, instead of 0.3.

For the base models, we used a single model obtained by averaging the last 5 checkpoints, which were written at 10-minute intervals. For the big models, we averaged the last 20 checkpoints. We used beam search with a beam size of 4 and length penalty $\alpha = 0.6$ [38]. These hyperparameters were chosen after experimentation on the development set. We set the maximum output length during inference to input length + 50, but terminate early when possible [38].

Table 2 summarizes our results and compares our translation quality and training costs to other model architectures from the literature. We estimate the number of floating point operations used to train a model by multiplying the training time, the number of GPUs used, and an estimate of the sustained single-precision floating-point capacity of each GPU [5].

6.2 Model Variations

To evaluate the importance of different components of the Transformer, we varied our base model in different ways, measuring the change in performance on English-to-German translation on the development set, newstest2013. We used beam search as described in the previous section, but no checkpoint averaging. We present these results in Table 3.

In Table 3 rows (A), we vary the number of attention heads and the attention key and value dimensions, keeping the amount of computation constant, as described in Section 3.2.2. While single-head attention is 0.9 BLEU worse than the best setting, quality also drops off with too many heads.

[5] We used values of 2.8, 3.7, 6.0 and 9.5 TFLOPS for K80, K40, M40 and P100, respectively.

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	**Training**	**WSJ 23 F1**
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

In Table 3 rows (B), we observe that reducing the attention key size d_k hurts model quality. This suggests that determining compatibility is not easy and that a more sophisticated compatibility function than dot product may be beneficial. We further observe in rows (C) and (D) that, as expected, bigger models are better, and dropout is very helpful in avoiding over-fitting. In row (E) we replace our sinusoidal positional encoding with learned positional embeddings [9], and observe nearly identical results to the base model.

6.3 English Constituency Parsing

To evaluate if the Transformer can generalize to other tasks we performed experiments on English constituency parsing. This task presents specific challenges: the output is subject to strong structural

constraints and is significantly longer than the input. Furthermore, RNN sequence-to-sequence models have not been able to attain state-of-the-art results in small-data regimes [37].

We trained a 4-layer transformer with $d_{model} = 1024$ on the Wall Street Journal (WSJ) portion of the Penn Treebank [25], about 40K training sentences. We also trained it in a semi-supervised setting, using the larger high-confidence and BerkleyParser corpora from with approximately 17M sentences [37]. We used a vocabulary of 16K tokens for the WSJ only setting and a vocabulary of 32K tokens for the semi-supervised setting.

We performed only a small number of experiments to select the dropout, both attention and residual (section 5.4), learning rates and beam size on the Section 22 development set, all other parameters remained unchanged from the English-to-German base translation model. During inference, we increased the maximum output length to input length + 300. We used a beam size of 21 and $\alpha = 0.3$ for both WSJ only and the semi-supervised setting.

Our results in Table 4 show that despite the lack of task-specific tuning our model performs surprisingly well, yielding better results than all previously reported models with the exception of the Recurrent Neural Network Grammar [8].

In contrast to RNN sequence-to-sequence models [37], the Transformer outperforms the Berkeley-Parser [29] even when training only on the WSJ training set of 40K sentences.

7 Conclusion

In this work, we presented the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers. On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, we achieve a new state of the art. In the former task our best model outperforms even all previously reported ensembles.

We are excited about the future of attention-based models and plan to apply them to other tasks. We plan to extend the Transformer to problems involving input and output modalities other than text and to investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs such as images, audio and video. Making generation less sequential is another research goals of ours.

The code we used to train and evaluate our models is available at `https://github.com/tensorflow/tensor2tensor`.

Acknowledgements We are grateful to Nal Kalchbrenner and Stephan Gouws for their fruitful comments, corrections and inspiration.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[6] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proc. of NAACL*, 2016.

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122v2*, 2017.

[10] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Zhongqiang Huang and Mary Harper. Self-training PCFG grammars with latent annotations across languages. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 832–841. ACL, August 2009.

[15] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[16] Łukasz Kaiser and Samy Bengio. Can active memory replace attention? In *Advances in Neural Information Processing Systems, (NIPS)*, 2016.

[17] Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. In *International Conference on Learning Representations (ICLR)*, 2016.

[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099v2*, 2017.

[19] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. In *International Conference on Learning Representations*, 2017.

[20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[21] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for LSTM networks. *arXiv preprint arXiv:1703.10722*, 2017.

[22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[23] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[24] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[25] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[26] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159. ACL, June 2006.

[27] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model. In *Empirical Methods in Natural Language Processing*, 2016.

[28] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[29] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 433–440. ACL, July 2006.

[30] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[32] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[33] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[34] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2440–2448. Curran Associates, Inc., 2015.

[35] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[37] Vinyals & Kaiser, Koo, Petrov, Sutskever, and Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, 2015.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[39] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep recurrent models with fast-forward connections for neural machine translation. *CoRR*, abs/1606.04199, 2016.

[40] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 434–443. ACL, August 2013.

Attention Visualizations

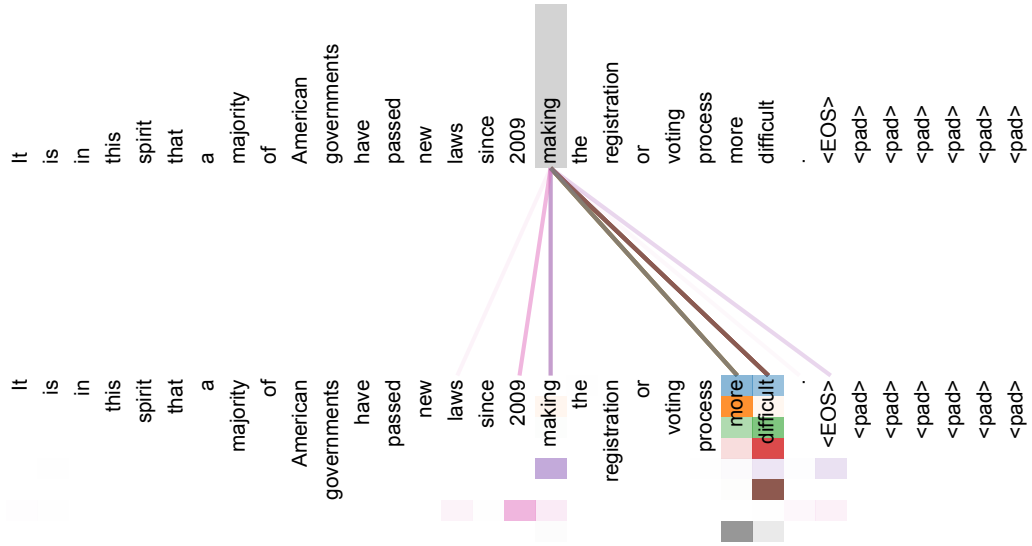


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

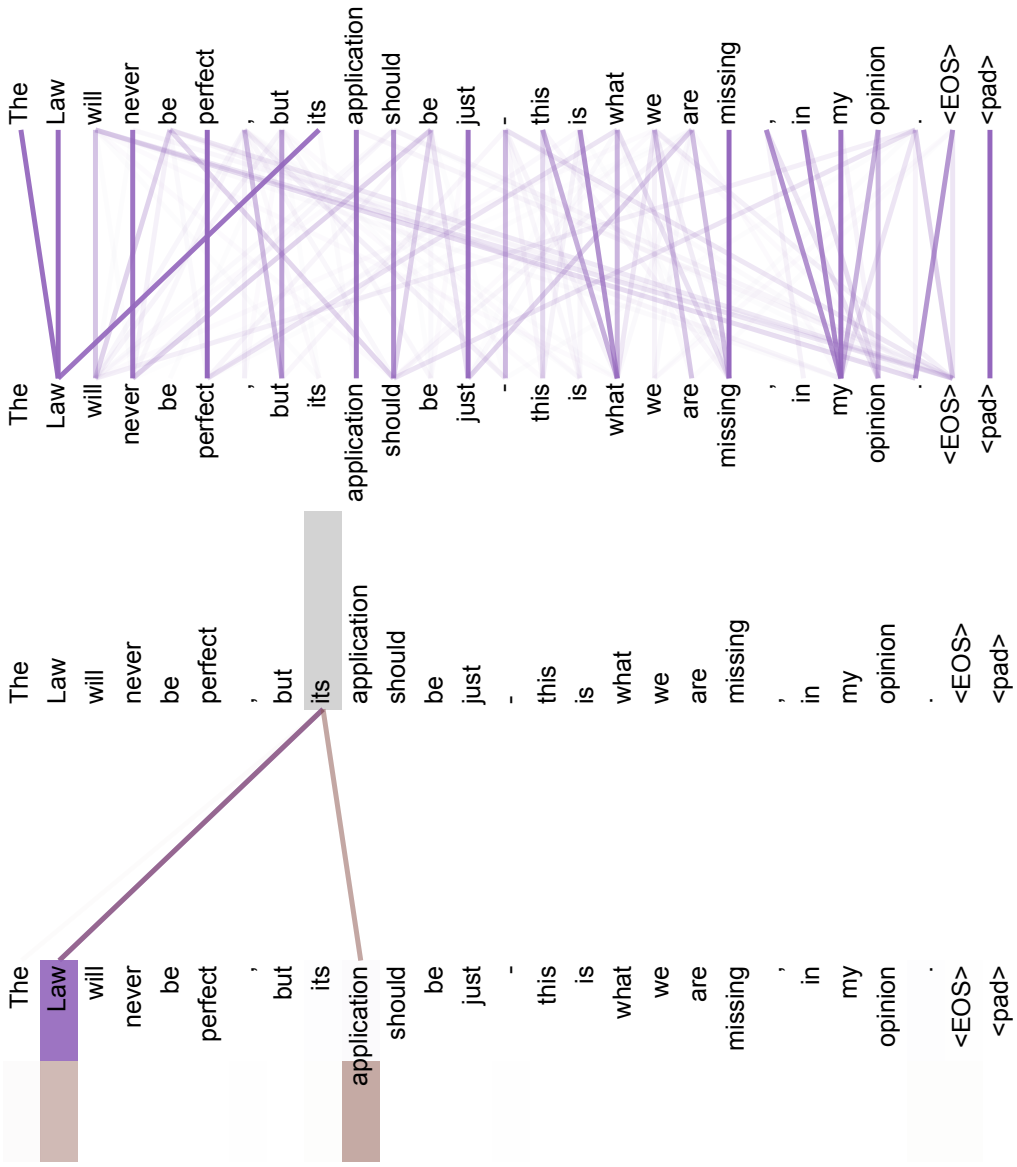


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

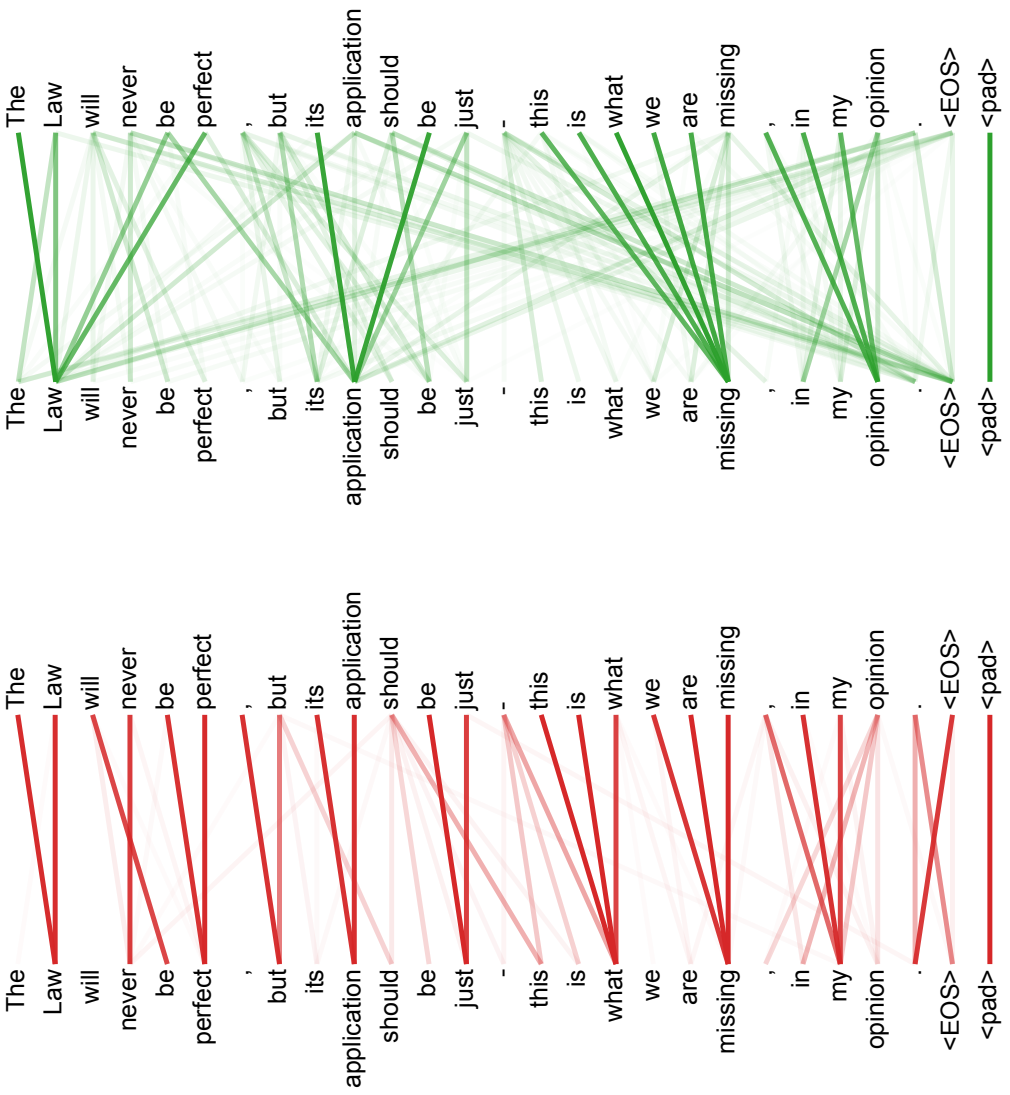


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

1 Introduction

Language model pre-training has been shown to be effective for improving many natural language processing tasks (Dai and Le, 2015; Peters et al., 2018a; Radford et al., 2018; Howard and Ruder, 2018). These include sentence-level tasks such as natural language inference (Bowman et al., 2015; Williams et al., 2018) and paraphrasing (Dolan and Brockett, 2005), which aim to predict the relationships between sentences by analyzing them holistically, as well as token-level tasks such as named entity recognition and question answering, where models are required to produce fine-grained output at the token level (Tjong Kim Sang and De Meulder, 2003; Rajpurkar et al., 2016).

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attend to previous tokens in the self-attention layers of the Transformer (Vaswani et al., 2017). Such restrictions are sub-optimal for sentence-level tasks, and could be very harmful when applying fine-tuning based approaches to token-level tasks such as question answering, where it is crucial to incorporate context from both directions.

In this paper, we improve the fine-tuning based approaches by proposing BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. BERT alleviates the previously mentioned unidirectionality constraint by using a "masked language model" (MLM) pre-training objective, inspired by the Cloze task (Taylor, 1953). The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked

word based only on its context. Unlike left-to-right language model pre-training, the MLM objective enables the representation to fuse the left and the right context, which allows us to pre-train a deep bidirectional Transformer. In addition to the masked language model, we also use a "next sentence prediction" task that jointly pre-trains text-pair representations. The contributions of our paper are as follows:

- We demonstrate the importance of bidirectional pre-training for language representations. Unlike Radford et al. (2018), which uses unidirectional language models for pre-training, BERT uses masked language models to enable pre-trained deep bidirectional representations. This is also in contrast to Peters et al. (2018a), which uses a shallow concatenation of independently trained left-to-right and right-to-left LMs.
- We show that pre-trained representations reduce the need for many heavily-engineered task-specific architectures. BERT is the first fine-tuning based representation model that achieves state-of-the-art performance on a large suite of sentence-level *and* token-level tasks, outperforming many task-specific architectures.
- BERT advances the state of the art for eleven NLP tasks. The code and pre-trained models are available at `https://github.com/google-research/bert`.

2 Related Work

There is a long history of pre-training general language representations, and we briefly review the most widely-used approaches in this section.

2.1 Unsupervised Feature-based Approaches

Learning widely applicable representations of words has been an active area of research for decades, including non-neural (Brown et al., 1992; Ando and Zhang, 2005; Blitzer et al., 2006) and neural (Mikolov et al., 2013; Pennington et al., 2014) methods. Pre-trained word embeddings are an integral part of modern NLP systems, offering significant improvements over embeddings learned from scratch (Turian et al., 2010). To pre-train word embedding vectors, left-to-right language modeling objectives have been used (Mnih and Hinton, 2009), as well as objectives to discriminate correct from incorrect words in left and right context (Mikolov et al., 2013).

These approaches have been generalized to coarser granularities, such as sentence embeddings (Kiros et al., 2015; Logeswaran and Lee, 2018) or paragraph embeddings (Le and Mikolov, 2014). To train sentence representations, prior work has used objectives to rank candidate next sentences (Jernite et al., 2017; Logeswaran and Lee, 2018), left-to-right generation of next sentence words given a representation of the previous sentence (Kiros et al., 2015), or denoising auto-encoder derived objectives (Hill et al., 2016).

ELMo and its predecessor (Peters et al., 2017, 2018a) generalize traditional word embedding research along a different dimension. They extract *context-sensitive* features from a left-to-right and a right-to-left language model. The contextual representation of each token is the concatenation of the left-to-right and right-to-left representations. When integrating contextual word embeddings with existing task-specific architectures, ELMo advances the state of the art for several major NLP benchmarks (Peters et al., 2018a) including question answering (Rajpurkar et al., 2016), sentiment analysis (Socher et al., 2013), and named entity recognition (Tjong Kim Sang and De Meulder, 2003). Melamud et al. (2016) proposed learning contextual representations through a task to predict a single word from both left and right context using LSTMs. Similar to ELMo, their model is feature-based and not deeply bidirectional. Fedus et al. (2018) shows that the cloze task can be used to improve the robustness of text generation models.

2.2 Unsupervised Fine-tuning Approaches

As with the feature-based approaches, the first works in this direction only pre-trained word embedding parameters from unlabeled text (Collobert and Weston, 2008).

More recently, sentence or document encoders which produce contextual token representations have been pre-trained from unlabeled text and fine-tuned for a supervised downstream task (Dai and Le, 2015; Howard and Ruder, 2018; Radford et al., 2018). The advantage of these approaches is that few parameters need to be learned from scratch. At least partly due to this advantage, OpenAI GPT (Radford et al., 2018) achieved previously state-of-the-art results on many sentence-level tasks from the GLUE benchmark (Wang et al., 2018a). Left-to-right language model-

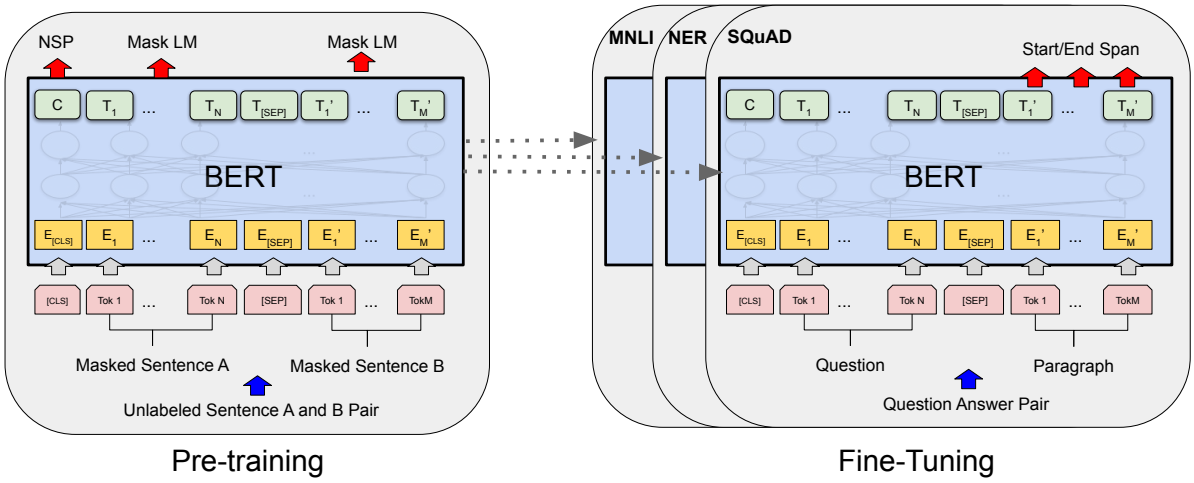


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

ing and auto-encoder objectives have been used for pre-training such models (Howard and Ruder, 2018; Radford et al., 2018; Dai and Le, 2015).

2.3 Transfer Learning from Supervised Data

There has also been work showing effective transfer from supervised tasks with large datasets, such as natural language inference (Conneau et al., 2017) and machine translation (McCann et al., 2017). Computer vision research has also demonstrated the importance of transfer learning from large pre-trained models, where an effective recipe is to fine-tune models pre-trained with ImageNet (Deng et al., 2009; Yosinski et al., 2014).

3 BERT

We introduce BERT and its detailed implementation in this section. There are two steps in our framework: *pre-training* and *fine-tuning*. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters. The question-answering example in Figure 1 will serve as a running example for this section.

A distinctive feature of BERT is its unified architecture across different tasks. There is minimal difference between the pre-trained architecture and the final downstream architecture.

Model Architecture BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in Vaswani et al. (2017) and released in the `tensor2tensor` library.[1] Because the use of Transformers has become common and our implementation is almost identical to the original, we will omit an exhaustive background description of the model architecture and refer readers to Vaswani et al. (2017) as well as excellent guides such as "The Annotated Transformer."[2]

In this work, we denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A.[3] We primarily report results on two model sizes: $\textbf{BERT}_{\textbf{BASE}}$ (L=12, H=768, A=12, Total Parameters=110M) and $\textbf{BERT}_{\textbf{LARGE}}$ (L=24, H=1024, A=16, Total Parameters=340M).

$\text{BERT}_{\text{BASE}}$ was chosen to have the same model size as OpenAI GPT for comparison purposes. Critically, however, the BERT Transformer uses bidirectional self-attention, while the GPT Transformer uses constrained self-attention where every token can only attend to context to its left.[4]

[1] https://github.com/tensorflow/tensor2tensor

[2] http://nlp.seas.harvard.edu/2018/04/03/attention.html

[3] In all cases we set the feed-forward/filter size to be $4H$, i.e., 3072 for the $H = 768$ and 4096 for the $H = 1024$.

[4] We note that in the literature the bidirectional Trans-

Input/Output Representations To make BERT handle a variety of down-stream tasks, our input representation is able to unambiguously represent both a single sentence and a pair of sentences (e.g., ⟨ Question, Answer ⟩) in one token sequence. Throughout this work, a "sentence" can be an arbitrary span of contiguous text, rather than an actual linguistic sentence. A "sequence" refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together.

We use WordPiece embeddings (Wu et al., 2016) with a 30,000 token vocabulary. The first token of every sequence is always a special classification token (`[CLS]`). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks. Sentence pairs are packed together into a single sequence. We differentiate the sentences in two ways. First, we separate them with a special token (`[SEP]`). Second, we add a learned embedding to every token indicating whether it belongs to sentence `A` or sentence `B`. As shown in Figure 1, we denote input embedding as E, the final hidden vector of the special `[CLS]` token as $C \in \mathbb{R}^H$, and the final hidden vector for the i^{th} input token as $T_i \in \mathbb{R}^H$.

For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings. A visualization of this construction can be seen in Figure 2.

3.1 Pre-training BERT

Unlike Peters et al. (2018a) and Radford et al. (2018), we do not use traditional left-to-right or right-to-left language models to pre-train BERT. Instead, we pre-train BERT using two unsupervised tasks, described in this section. This step is presented in the left part of Figure 1.

Task #1: Masked LM Intuitively, it is reasonable to believe that a deep bidirectional model is strictly more powerful than either a left-to-right model or the shallow concatenation of a left-to-right and a right-to-left model. Unfortunately, standard conditional language models can only be trained left-to-right *or* right-to-left, since bidirectional conditioning would allow each word to indirectly "see itself", and the model could trivially predict the target word in a multi-layered context.

In order to train a deep bidirectional representation, we simply mask some percentage of the input tokens at random, and then predict those masked tokens. We refer to this procedure as a "masked LM" (MLM), although it is often referred to as a *Cloze* task in the literature (Taylor, 1953). In this case, the final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, as in a standard LM. In all of our experiments, we mask 15% of all WordPiece tokens in each sequence at random. In contrast to denoising auto-encoders (Vincent et al., 2008), we only predict the masked words rather than reconstructing the entire input.

Although this allows us to obtain a bidirectional pre-trained model, a downside is that we are creating a mismatch between pre-training and fine-tuning, since the `[MASK]` token does not appear during fine-tuning. To mitigate this, we do not always replace "masked" words with the actual `[MASK]` token. The training data generator chooses 15% of the token positions at random for prediction. If the i-th token is chosen, we replace the i-th token with (1) the `[MASK]` token 80% of the time (2) a random token 10% of the time (3) the unchanged i-th token 10% of the time. Then, T_i will be used to predict the original token with cross entropy loss. We compare variations of this procedure in Appendix C.2.

Task #2: Next Sentence Prediction (NSP) Many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI) are based on understanding the *relationship* between two sentences, which is not directly captured by language modeling. In order to train a model that understands sentence relationships, we pre-train for a binarized *next sentence prediction* task that can be trivially generated from any monolingual corpus. Specifically, when choosing the sentences `A` and `B` for each pre-training example, 50% of the time `B` is the actual next sentence that follows `A` (labeled as `IsNext`), and 50% of the time it is a random sentence from the corpus (labeled as `NotNext`). As we show in Figure 1, C is used for next sentence prediction (NSP).[5] Despite its simplicity, we demonstrate in Section 5.1 that pre-training towards this task is very beneficial to both QA and NLI. [6]

former is often referred to as a "Transformer encoder" while the left-context-only version is referred to as a "Transformer decoder" since it can be used for text generation.

[5] The final model achieves 97%-98% accuracy on NSP.

[6] The vector C is not a meaningful sentence representation without fine-tuning, since it was trained with NSP.

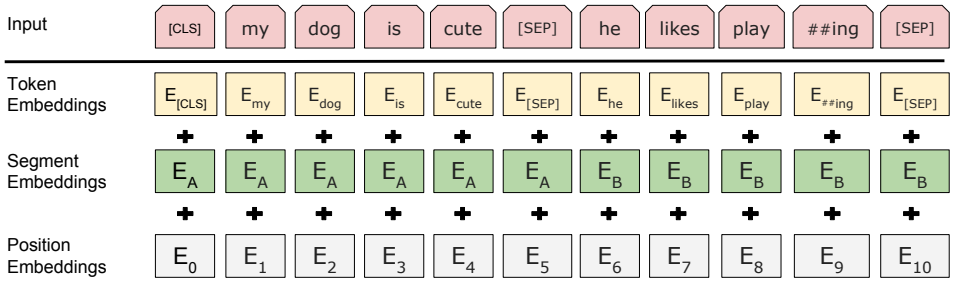


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

The NSP task is closely related to representation-learning objectives used in Jernite et al. (2017) and Logeswaran and Lee (2018). However, in prior work, only sentence embeddings are transferred to down-stream tasks, where BERT transfers all parameters to initialize end-task model parameters.

Pre-training data The pre-training procedure largely follows the existing literature on language model pre-training. For the pre-training corpus we use the BooksCorpus (800M words) (Zhu et al., 2015) and English Wikipedia (2,500M words). For Wikipedia we extract only the text passages and ignore lists, tables, and headers. It is critical to use a document-level corpus rather than a shuffled sentence-level corpus such as the Billion Word Benchmark (Chelba et al., 2013) in order to extract long contiguous sequences.

3.2 Fine-tuning BERT

Fine-tuning is straightforward since the self-attention mechanism in the Transformer allows BERT to model many downstream tasks—whether they involve single text or text pairs—by swapping out the appropriate inputs and outputs. For applications involving text pairs, a common pattern is to independently encode text pairs before applying bidirectional cross attention, such as Parikh et al. (2016); Seo et al. (2017). BERT instead uses the self-attention mechanism to unify these two stages, as encoding a concatenated text pair with self-attention effectively includes *bidirectional* cross attention between two sentences.

For each task, we simply plug in the task-specific inputs and outputs into BERT and fine-tune all the parameters end-to-end. At the input, sentence `A` and sentence `B` from pre-training are analogous to (1) sentence pairs in paraphrasing, (2) hypothesis-premise pairs in entailment, (3) question-passage pairs in question answering, and (4) a degenerate text-$\varnothing$ pair in text classification or sequence tagging. At the output, the token representations are fed into an output layer for token-level tasks, such as sequence tagging or question answering, and the `[CLS]` representation is fed into an output layer for classification, such as entailment or sentiment analysis.

Compared to pre-training, fine-tuning is relatively inexpensive. All of the results in the paper can be replicated in at most 1 hour on a single Cloud TPU, or a few hours on a GPU, starting from the exact same pre-trained model.[7] We describe the task-specific details in the corresponding subsections of Section 4. More details can be found in Appendix A.5.

4 Experiments

In this section, we present BERT fine-tuning results on 11 NLP tasks.

4.1 GLUE

The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018a) is a collection of diverse natural language understanding tasks. Detailed descriptions of GLUE datasets are included in Appendix B.1.

To fine-tune on GLUE, we represent the input sequence (for single sentence or sentence pairs) as described in Section 3, and use the final hidden vector $C \in \mathbb{R}^H$ corresponding to the first input token (`[CLS]`) as the aggregate representation. The only new parameters introduced during fine-tuning are classification layer weights $W \in \mathbb{R}^{K \times H}$, where K is the number of labels. We compute a standard classification loss with C and W, i.e., $\log(\text{softmax}(CW^T))$.

[7] For example, the BERT SQuAD model can be trained in around 30 minutes on a single Cloud TPU to achieve a Dev F1 score of 91.0%.

[8] See (10) in `https://gluebenchmark.com/faq`.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

We use a batch size of 32 and fine-tune for 3 epochs over the data for all GLUE tasks. For each task, we selected the best fine-tuning learning rate (among 5e-5, 4e-5, 3e-5, and 2e-5) on the Dev set. Additionally, for $\text{BERT}_{\text{LARGE}}$ we found that fine-tuning was sometimes unstable on small datasets, so we ran several random restarts and selected the best model on the Dev set. With random restarts, we use the same pre-trained checkpoint but perform different fine-tuning data shuffling and classifier layer initialization.[9]

Results are presented in Table 1. Both $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ outperform all systems on all tasks by a substantial margin, obtaining 4.5% and 7.0% respective average accuracy improvement over the prior state of the art. Note that $\text{BERT}_{\text{BASE}}$ and OpenAI GPT are nearly identical in terms of model architecture apart from the attention masking. For the largest and most widely reported GLUE task, MNLI, BERT obtains a 4.6% absolute accuracy improvement. On the official GLUE leaderboard[10], $\text{BERT}_{\text{LARGE}}$ obtains a score of 80.5, compared to OpenAI GPT, which obtains 72.8 as of the date of writing.

We find that $\text{BERT}_{\text{LARGE}}$ significantly outperforms $\text{BERT}_{\text{BASE}}$ across all tasks, especially those with very little training data. The effect of model size is explored more thoroughly in Section 5.2.

4.2 SQuAD v1.1

The Stanford Question Answering Dataset (SQuAD v1.1) is a collection of 100k crowd-sourced question/answer pairs (Rajpurkar et al., 2016). Given a question and a passage from Wikipedia containing the answer, the task is to predict the answer text span in the passage.

As shown in Figure 1, in the question answering task, we represent the input question and passage as a single packed sequence, with the question using the A embedding and the passage using the B embedding. We only introduce a start vector $S \in \mathbb{R}^H$ and an end vector $E \in \mathbb{R}^H$ during fine-tuning. The probability of word i being the start of the answer span is computed as a dot product between T_i and S followed by a softmax over all of the words in the paragraph: $P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$. The analogous formula is used for the end of the answer span. The score of a candidate span from position i to position j is defined as $S \cdot T_i + E \cdot T_j$, and the maximum scoring span where $j \geq i$ is used as a prediction. The training objective is the sum of the log-likelihoods of the correct start and end positions. We fine-tune for 3 epochs with a learning rate of 5e-5 and a batch size of 32.

Table 2 shows top leaderboard entries as well as results from top published systems (Seo et al., 2017; Clark and Gardner, 2018; Peters et al., 2018a; Hu et al., 2018). The top results from the SQuAD leaderboard do not have up-to-date public system descriptions available,[11] and are allowed to use any public data when training their systems. We therefore use modest data augmentation in our system by first fine-tuning on TriviaQA (Joshi et al., 2017) befor fine-tuning on SQuAD.

Our best performing system outperforms the top leaderboard system by +1.5 F1 in ensembling and +1.3 F1 as a single system. In fact, our single BERT model outperforms the top ensemble system in terms of F1 score. Without TriviaQA fine-

[9] The GLUE data set distribution does not include the Test labels, and we only made a single GLUE evaluation server submission for each of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$.

[10] https://gluebenchmark.com/leaderboard

[11] QANet is described in Yu et al. (2018), but the system has improved substantially after publication.

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
$\text{BERT}_{\text{LARGE}}$ (Single)	78.7	81.9	80.0	83.1

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components.

tuning data, we only lose 0.1-0.4 F1, still outperforming all existing systems by a wide margin.[12]

4.3 SQuAD v2.0

The SQuAD 2.0 task extends the SQuAD 1.1 problem definition by allowing for the possibility that no short answer exists in the provided paragraph, making the problem more realistic.

We use a simple approach to extend the SQuAD v1.1 BERT model for this task. We treat questions that do not have an answer as having an answer span with start and end at the `[CLS]` token. The probability space for the start and end answer span positions is extended to include the position of the `[CLS]` token. For prediction, we compare the score of the no-answer span: $s_{\texttt{null}} = S{\cdot}C + E{\cdot}C$ to the score of the best non-null span $\hat{s_{i,j}} = \max_{j \geq i} S{\cdot}T_i + E{\cdot}T_j$. We predict a non-null answer when $\hat{s_{i,j}} > s_{\texttt{null}} + \tau$, where the threshold τ is selected on the dev set to maximize F1. We did not use TriviaQA data for this model. We fine-tuned for 2 epochs with a learning rate of 5e-5 and a batch size of 48.

The results compared to prior leaderboard entries and top published work (Sun et al., 2018; Wang et al., 2018b) are shown in Table 3, excluding systems that use BERT as one of their components. We observe a +5.1 F1 improvement over the previous best system.

[12] The TriviaQA data we used consists of paragraphs from TriviaQA-Wiki formed of the first 400 tokens in documents, that contain at least one of the provided possible answers.

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

Table 4: SWAG Dev and Test accuracies. †Human performance is measured with 100 samples, as reported in the SWAG paper.

4.4 SWAG

The Situations With Adversarial Generations (SWAG) dataset contains 113k sentence-pair completion examples that evaluate grounded commonsense inference (Zellers et al., 2018). Given a sentence, the task is to choose the most plausible continuation among four choices.

When fine-tuning on the SWAG dataset, we construct four input sequences, each containing the concatenation of the given sentence (sentence `A`) and a possible continuation (sentence `B`). The only task-specific parameters introduced is a vector whose dot product with the `[CLS]` token representation C denotes a score for each choice which is normalized with a softmax layer.

We fine-tune the model for 3 epochs with a learning rate of 2e-5 and a batch size of 16. Results are presented in Table 4. $\text{BERT}_{\text{LARGE}}$ outperforms the authors' baseline ESIM+ELMo system by +27.1% and OpenAI GPT by 8.3%.

5 Ablation Studies

In this section, we perform ablation experiments over a number of facets of BERT in order to better understand their relative importance. Additional

Tasks	Dev Set				
	MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$\text{BERT}_{\text{BASE}}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

ablation studies can be found in Appendix C.

5.1 Effect of Pre-training Tasks

We demonstrate the importance of the deep bidirectionality of BERT by evaluating two pre-training objectives using exactly the same pre-training data, fine-tuning scheme, and hyperparameters as $\text{BERT}_{\text{BASE}}$:

No NSP: A bidirectional model which is trained using the "masked LM" (MLM) but without the "next sentence prediction" (NSP) task.

LTR & No NSP: A left-context-only model which is trained using a standard Left-to-Right (LTR) LM, rather than an MLM. The left-only constraint was also applied at fine-tuning, because removing it introduced a pre-train/fine-tune mismatch that degraded downstream performance. Additionally, this model was pre-trained without the NSP task. This is directly comparable to OpenAI GPT, but using our larger training dataset, our input representation, and our fine-tuning scheme.

We first examine the impact brought by the NSP task. In Table 5, we show that removing NSP hurts performance significantly on QNLI, MNLI, and SQuAD 1.1. Next, we evaluate the impact of training bidirectional representations by comparing "No NSP" to "LTR & No NSP". The LTR model performs worse than the MLM model on all tasks, with large drops on MRPC and SQuAD.

For SQuAD it is intuitively clear that a LTR model will perform poorly at token predictions, since the token-level hidden states have no right-side context. In order to make a good faith attempt at strengthening the LTR system, we added a randomly initialized BiLSTM on top. This does significantly improve results on SQuAD, but the results are still far worse than those of the pre-trained bidirectional models. The BiLSTM hurts performance on the GLUE tasks.

We recognize that it would also be possible to train separate LTR and RTL models and represent each token as the concatenation of the two models, as ELMo does. However: (a) this is twice as expensive as a single bidirectional model; (b) this is non-intuitive for tasks like QA, since the RTL model would not be able to condition the answer on the question; (c) this it is strictly less powerful than a deep bidirectional model, since it can use both left and right context at every layer.

5.2 Effect of Model Size

In this section, we explore the effect of model size on fine-tuning task accuracy. We trained a number of BERT models with a differing number of layers, hidden units, and attention heads, while otherwise using the same hyperparameters and training procedure as described previously.

Results on selected GLUE tasks are shown in Table 6. In this table, we report the average Dev Set accuracy from 5 random restarts of fine-tuning. We can see that larger models lead to a strict accuracy improvement across all four datasets, even for MRPC which only has 3,600 labeled training examples, and is substantially different from the pre-training tasks. It is also perhaps surprising that we are able to achieve such significant improvements on top of models which are already quite large relative to the existing literature. For example, the largest Transformer explored in Vaswani et al. (2017) is (L=6, H=1024, A=16) with 100M parameters for the encoder, and the largest Transformer we have found in the literature is (L=64, H=512, A=2) with 235M parameters (Al-Rfou et al., 2018). By contrast, $\text{BERT}_{\text{BASE}}$ contains 110M parameters and $\text{BERT}_{\text{LARGE}}$ contains 340M parameters.

It has long been known that increasing the model size will lead to continual improvements on large-scale tasks such as machine translation and language modeling, which is demonstrated by the LM perplexity of held-out training data shown in Table 6. However, we believe that this is the first work to demonstrate convincingly that scaling to extreme model sizes also leads to large improvements on very small scale tasks, provided that the model has been sufficiently pre-trained. Peters et al. (2018b) presented

mixed results on the downstream task impact of increasing the pre-trained bi-LM size from two to four layers and Melamud et al. (2016) mentioned in passing that increasing hidden dimension size from 200 to 600 helped, but increasing further to 1,000 did not bring further improvements. Both of these prior works used a feature-based approach — we hypothesize that when the model is fine-tuned directly on the downstream tasks and uses only a very small number of randomly initialized additional parameters, the task-specific models can benefit from the larger, more expressive pre-trained representations even when downstream task data is very small.

5.3 Feature-based Approach with BERT

All of the BERT results presented so far have used the fine-tuning approach, where a simple classification layer is added to the pre-trained model, and all parameters are jointly fine-tuned on a downstream task. However, the feature-based approach, where fixed features are extracted from the pre-trained model, has certain advantages. First, not all tasks can be easily represented by a Transformer encoder architecture, and therefore require a task-specific model architecture to be added. Second, there are major computational benefits to pre-compute an expensive representation of the training data once and then run many experiments with cheaper models on top of this representation.

In this section, we compare the two approaches by applying BERT to the CoNLL-2003 Named Entity Recognition (NER) task (Tjong Kim Sang and De Meulder, 2003). In the input to BERT, we use a case-preserving WordPiece model, and we include the maximal document context provided by the data. Following standard practice, we formulate this as a tagging task but do not use a CRF layer in the output. We use the representation of the first sub-token as the input to the token-level classifier over the NER label set.

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

Table 6: Ablation over BERT model size. #L = the number of layers; #H = hidden size; #A = number of attention heads. "LM (ppl)" is the masked LM perplexity of held-out training data.

System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	**93.1**
Fine-tuning approach		
$\text{BERT}_{\text{LARGE}}$	96.6	92.8
$\text{BERT}_{\text{BASE}}$	96.4	92.4
Feature-based approach ($\text{BERT}_{\text{BASE}}$)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

To ablate the fine-tuning approach, we apply the feature-based approach by extracting the activations from one or more layers *without* fine-tuning any parameters of BERT. These contextual embeddings are used as input to a randomly initialized two-layer 768-dimensional BiLSTM before the classification layer.

Results are presented in Table 7. $\text{BERT}_{\text{LARGE}}$ performs competitively with state-of-the-art methods. The best performing method concatenates the token representations from the top four hidden layers of the pre-trained Transformer, which is only 0.3 F1 behind fine-tuning the entire model. This demonstrates that BERT is effective for both fine-tuning and feature-based approaches.

6 Conclusion

Recent empirical improvements due to transfer learning with language models have demonstrated that rich, unsupervised pre-training is an integral part of many language understanding systems. In particular, these results enable even low-resource tasks to benefit from deep unidirectional architectures. Our major contribution is further generalizing these findings to deep *bidirectional* architectures, allowing the same pre-trained model to successfully tackle a broad set of NLP tasks.

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649.

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2018. Character-level language modeling with deeper self-attention. *arXiv preprint arXiv:1808.04444*.

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853.

Luisa Bentivogli, Bernardo Magnini, Ido Dagan, Hoa Trang Dang, and Danilo Giampiccolo. 2009. The fifth PASCAL recognizing textual entailment challenge. In *TAC*. NIST.

John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 120–128. Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*. Association for Computational Linguistics.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

Z. Chen, H. Zhang, X. Zhang, and L. Zhao. 2018. Quora question pairs.

Christopher Clark and Matt Gardner. 2018. Simple and effective multi-paragraph reading comprehension. In *ACL*.

Kevin Clark, Minh-Thang Luong, Christopher D Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

William Fedus, Ian Goodfellow, and Andrew M Dai. 2018. Maskgan: Better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *ACL*. Association for Computational Linguistics.

Minghao Hu, Yuxing Peng, Zhen Huang, Xipeng Qiu, Furu Wei, and Ming Zhou. 2018. Reinforced mnemonic reader for machine reading comprehension. In *IJCAI*.

Yacine Jernite, Samuel R. Bowman, and David Sontag. 2017. Discourse-based objectives for fast unsupervised sentence representation learning. *CoRR*, abs/1705.00557.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL*.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2011. The winograd schema challenge. In *Aaai spring symposium: Logical formalizations of commonsense reasoning*, volume 46, page 47.

Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations. In *International Conference on Learning Representations*.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *NIPS*.

Oren Melamud, Jacob Goldberger, and Ido Dagan. 2016. context2vec: Learning generic context embedding with bidirectional LSTM. In *CoNLL*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Andriy Mnih and Geoffrey E Hinton. 2009. A scalable hierarchical distributed language model. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1081–1088. Curran Associates, Inc.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *EMNLP*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *ACL*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018a. Deep contextualized word representations. In *NAACL*.

Matthew Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. 2018b. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning. Technical report, OpenAI.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. In *ICLR*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Fu Sun, Linyang Li, Xipeng Qiu, and Yang Liu. 2018. U-net: Machine reading comprehension with unanswerable questions. *arXiv preprint arXiv:1810.06638*.

Wilson L Taylor. 1953. Cloze procedure: A new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *CoNLL*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 384–394.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018a. Glue: A multi-task benchmark and analysis platform

for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355.

Wei Wang, Ming Yan, and Chen Wu. 2018b. Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. 2018. QANet: Combining local convolution with global self-attention for reading comprehension. In *ICLR*.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. Swag: A large-scale adversarial dataset for grounded commonsense inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

Appendix for "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"

We organize the appendix into three sections:

- Additional implementation details for BERT are presented in Appendix A;
- Additional details for our experiments are presented in Appendix B; and
- Additional ablation studies are presented in Appendix C.

  We present additional ablation studies for BERT including:

  – Effect of Number of Training Steps; and
  – Ablation for Different Masking Procedures.

A Additional Details for BERT

A.1 Illustration of the Pre-training Tasks

We provide examples of the pre-training tasks in the following.

Masked LM and the Masking Procedure Assuming the unlabeled sentence is `my dog is hairy`, and during the random masking procedure we chose the 4-th token (which corresponding to `hairy`), our masking procedure can be further illustrated by

- 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is hairy` → `my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy` → `my dog is apple`
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy` → `my dog is hairy`. The purpose of this is to bias the representation towards the actual observed word.

The advantage of this procedure is that the Transformer encoder does not know which words it will be asked to predict or which have been replaced by random words, so it is forced to keep a distributional contextual representation of *every* input token. Additionally, because random replacement only occurs for 1.5% of all tokens (i.e., 10% of 15%), this does not seem to harm the model's language understanding capability. In Section C.2, we evaluate the impact this procedure.

Compared to standard langauge model training, the masked LM only make predictions on 15% of tokens in each batch, which suggests that more pre-training steps may be required for the model

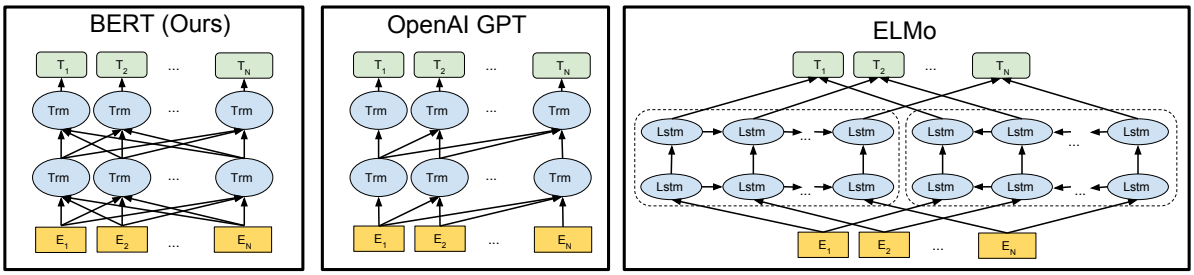


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

to converge. In Section C.1 we demonstrate that MLM does converge marginally slower than a left-to-right model (which predicts every token), but the empirical improvements of the MLM model far outweigh the increased training cost.

Next Sentence Prediction The next sentence prediction task can be illustrated in the following examples.

Input = `[CLS] the man went to [MASK] store [SEP]`
`he bought a gallon [MASK] milk [SEP]`
Label = `IsNext`

Input = `[CLS] the man [MASK] to the store [SEP]`
`penguin [MASK] are flight ##less birds [SEP]`
Label = `NotNext`

A.2 Pre-training Procedure

To generate each training input sequence, we sample two spans of text from the corpus, which we refer to as "sentences" even though they are typically much longer than single sentences (but can be shorter also). The first sentence receives the `A` embedding and the second receives the `B` embedding. 50% of the time `B` is the actual next sentence that follows `A` and 50% of the time it is a random sentence, which is done for the "next sentence prediction" task. They are sampled such that the combined length is $\leq$ 512 tokens. The LM masking is applied after WordPiece tokenization with a uniform masking rate of 15%, and no special consideration given to partial word pieces.

We train with batch size of 256 sequences (256 sequences * 512 tokens = 128,000 tokens/batch) for 1,000,000 steps, which is approximately 40 epochs over the 3.3 billion word corpus. We use Adam with learning rate of 1e-4, $\beta_1 = 0.9$, $\beta_2 = 0.999$, L2 weight decay of 0.01, learning rate warmup over the first 10,000 steps, and linear decay of the learning rate. We use a dropout probability of 0.1 on all layers. We use a `gelu` activation (Hendrycks and Gimpel, 2016) rather than the standard `relu`, following OpenAI GPT. The training loss is the sum of the mean masked LM likelihood and the mean next sentence prediction likelihood.

Training of $\text{BERT}_{\text{BASE}}$ was performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total).[13] Training of $\text{BERT}_{\text{LARGE}}$ was performed on 16 Cloud TPUs (64 TPU chips total). Each pre-training took 4 days to complete.

Longer sequences are disproportionately expensive because attention is quadratic to the sequence length. To speed up pretraing in our experiments, we pre-train the model with sequence length of 128 for 90% of the steps. Then, we train the rest 10% of the steps of sequence of 512 to learn the positional embeddings.

A.3 Fine-tuning Procedure

For fine-tuning, most model hyperparameters are the same as in pre-training, with the exception of the batch size, learning rate, and number of training epochs. The dropout probability was always kept at 0.1. The optimal hyperparameter values are task-specific, but we found the following range of possible values to work well across all tasks:

- **Batch size**: 16, 32

[13] https://cloudplatform.googleblog.com/2018/06/Cloud-TPU-now-offers-preemptible-pricing-and-global-availability.html

- **Learning rate (Adam)**: 5e-5, 3e-5, 2e-5
- **Number of epochs**: 2, 3, 4

We also observed that large data sets (e.g., 100k+ labeled training examples) were far less sensitive to hyperparameter choice than small data sets. Fine-tuning is typically very fast, so it is reasonable to simply run an exhaustive search over the above parameters and choose the model that performs best on the development set.

A.4 Comparison of BERT, ELMo ,and OpenAI GPT

Here we studies the differences in recent popular representation learning models including ELMo, OpenAI GPT and BERT. The comparisons between the model architectures are shown visually in Figure 3. Note that in addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

The most comparable existing pre-training method to BERT is OpenAI GPT, which trains a left-to-right Transformer LM on a large text corpus. In fact, many of the design decisions in BERT were intentionally made to make it as close to GPT as possible so that the two methods could be minimally compared. The core argument of this work is that the bi-directionality and the two pre-training tasks presented in Section 3.1 account for the majority of the empirical improvements, but we do note that there are several other differences between how BERT and GPT were trained:

- GPT is trained on the BooksCorpus (800M words); BERT is trained on the BooksCorpus (800M words) and Wikipedia (2,500M words).
- GPT uses a sentence separator (`[SEP]`) and classifier token (`[CLS]`) which are only introduced at fine-tuning time; BERT learns `[SEP]`, `[CLS]` and sentence `A`/`B` embeddings during pre-training.
- GPT was trained for 1M steps with a batch size of 32,000 words; BERT was trained for 1M steps with a batch size of 128,000 words.
- GPT used the same learning rate of 5e-5 for all fine-tuning experiments; BERT chooses a task-specific fine-tuning learning rate which performs the best on the development set.

To isolate the effect of these differences, we perform ablation experiments in Section 5.1 which demonstrate that the majority of the improvements are in fact coming from the two pre-training tasks and the bidirectionality they enable.

A.5 Illustrations of Fine-tuning on Different Tasks

The illustration of fine-tuning BERT on different tasks can be seen in Figure 4. Our task-specific models are formed by incorporating BERT with one additional output layer, so a minimal number of parameters need to be learned from scratch. Among the tasks, (a) and (b) are sequence-level tasks while (c) and (d) are token-level tasks. In the figure, E represents the input embedding, T_i represents the contextual representation of token i, [CLS] is the special symbol for classification output, and [SEP] is the special symbol to separate non-consecutive token sequences.

B Detailed Experimental Setup

B.1 Detailed Descriptions for the GLUE Benchmark Experiments.

Our GLUE results in Table1 are obtained from `https://gluebenchmark.com/leaderboard` and `https://blog.openai.com/language-unsupervised`. The GLUE benchmark includes the following datasets, the descriptions of which were originally summarized in Wang et al. (2018a):

MNLI Multi-Genre Natural Language Inference is a large-scale, crowdsourced entailment classification task (Williams et al., 2018). Given a pair of sentences, the goal is to predict whether the second sentence is an *entailment*, *contradiction*, or *neutral* with respect to the first one.

QQP Quora Question Pairs is a binary classification task where the goal is to determine if two questions asked on Quora are semantically equivalent (Chen et al., 2018).

QNLI Question Natural Language Inference is a version of the Stanford Question Answering Dataset (Rajpurkar et al., 2016) which has been converted to a binary classification task (Wang et al., 2018a). The positive examples are (question, sentence) pairs which do contain the correct answer, and the negative examples are (question, sentence) from the same paragraph which do not contain the answer.

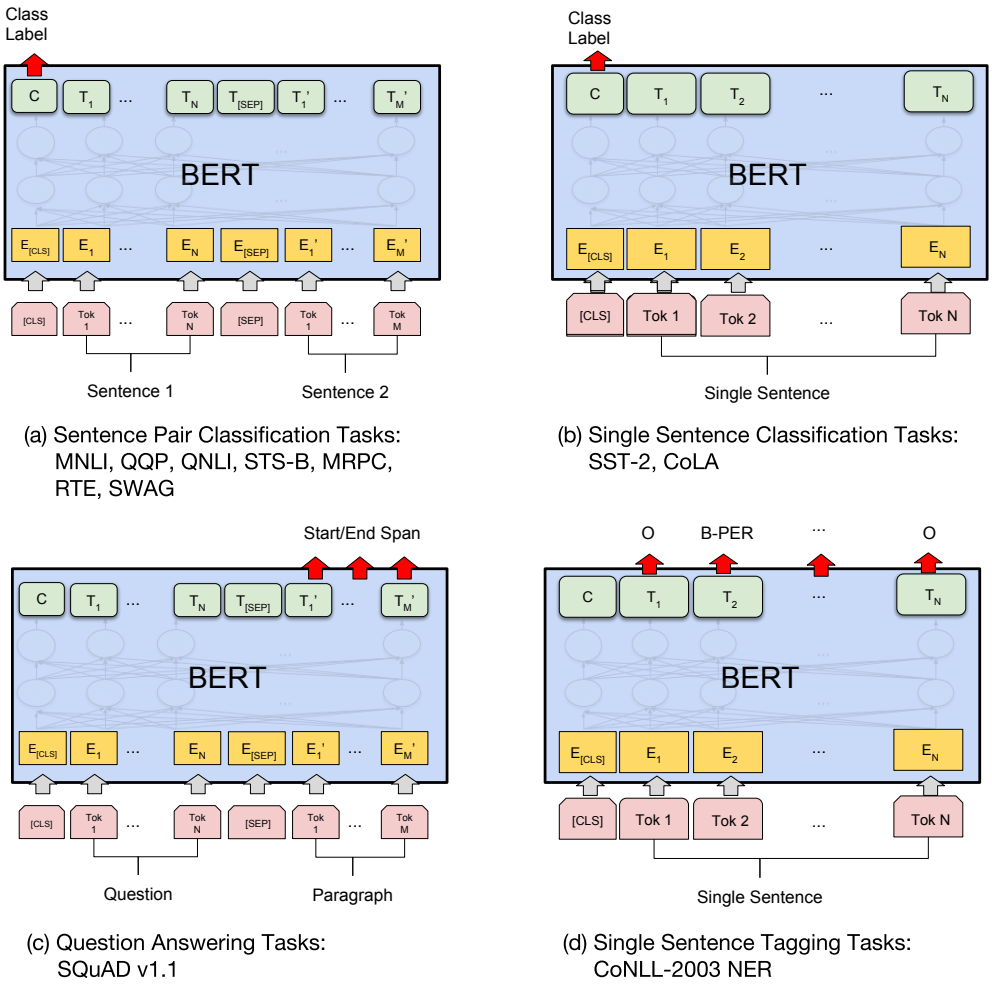


Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

SST-2 The Stanford Sentiment Treebank is a binary single-sentence classification task consisting of sentences extracted from movie reviews with human annotations of their sentiment (Socher et al., 2013).

CoLA The Corpus of Linguistic Acceptability is a binary single-sentence classification task, where the goal is to predict whether an English sentence is linguistically "acceptable" or not (Warstadt et al., 2018).

STS-B The Semantic Textual Similarity Benchmark is a collection of sentence pairs drawn from news headlines and other sources (Cer et al., 2017). They were annotated with a score from 1 to 5 denoting how similar the two sentences are in terms of semantic meaning.

MRPC Microsoft Research Paraphrase Corpus consists of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent (Dolan and Brockett, 2005).

RTE Recognizing Textual Entailment is a binary entailment task similar to MNLI, but with much less training data (Bentivogli et al., 2009).[14]

WNLI Winograd NLI is a small natural language inference dataset (Levesque et al., 2011). The GLUE webpage notes that there are issues with the construction of this dataset, [15] and every trained system that's been submitted to GLUE has performed worse than the 65.1 baseline accuracy of predicting the majority class. We therefore exclude this set to be fair to OpenAI GPT. For our GLUE submission, we always predicted the ma-

[14]Note that we only report single-task fine-tuning results in this paper. A multitask fine-tuning approach could potentially push the performance even further. For example, we did observe substantial improvements on RTE from multi-task training with MNLI.

[15]https://gluebenchmark.com/faq

jority class.

C Additional Ablation Studies

C.1 Effect of Number of Training Steps

Figure 5 presents MNLI Dev accuracy after fine-tuning from a checkpoint that has been pre-trained for k steps. This allows us to answer the following questions:

1. Question: Does BERT really need such a large amount of pre-training (128,000 words/batch * 1,000,000 steps) to achieve high fine-tuning accuracy?
   Answer: Yes, $\text{BERT}_{\text{BASE}}$ achieves almost 1.0% additional accuracy on MNLI when trained on 1M steps compared to 500k steps.

2. Question: Does MLM pre-training converge slower than LTR pre-training, since only 15% of words are predicted in each batch rather than every word?
   Answer: The MLM model does converge slightly slower than the LTR model. However, in terms of absolute accuracy the MLM model begins to outperform the LTR model almost immediately.

C.2 Ablation for Different Masking Procedures

In Section 3.1, we mention that BERT uses a mixed strategy for masking the target tokens when pre-training with the masked language model (MLM) objective. The following is an ablation study to evaluate the effect of different masking strategies.

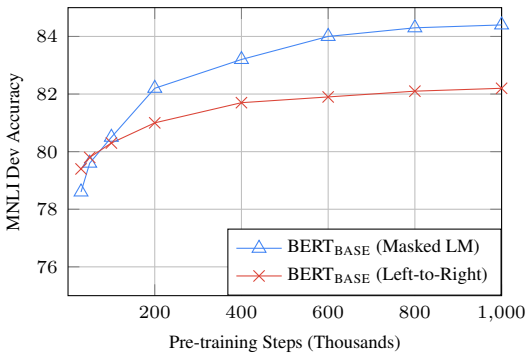


Figure 5: Ablation over number of training steps. This shows the MNLI accuracy after fine-tuning, starting from model parameters that have been pre-trained for k steps. The x-axis is the value of k.

Note that the purpose of the masking strategies is to reduce the mismatch between pre-training and fine-tuning, as the `[MASK]` symbol never appears during the fine-tuning stage. We report the Dev results for both MNLI and NER. For NER, we report both fine-tuning and feature-based approaches, as we expect the mismatch will be amplified for the feature-based approach as the model will not have the chance to adjust the representations.

Masking Rates			Dev Set Results		
MASK	SAME	RND	MNLI	NER	
			Fine-tune	Fine-tune	Feature-based
80%	10%	10%	84.2	95.4	94.9
100%	0%	0%	84.3	94.9	94.0
80%	0%	20%	84.1	95.2	94.6
80%	20%	0%	84.4	95.2	94.7
0%	20%	80%	83.7	94.8	94.6
0%	0%	100%	83.6	94.9	94.6

Table 8: Ablation over different masking strategies.

The results are presented in Table 8. In the table, MASK means that we replace the target token with the `[MASK]` symbol for MLM; SAME means that we keep the target token as is; RND means that we replace the target token with another random token.

The numbers in the left part of the table represent the probabilities of the specific strategies used during MLM pre-training (BERT uses 80%, 10%, 10%). The right part of the paper represents the Dev set results. For the feature-based approach, we concatenate the last 4 layers of BERT as the features, which was shown to be the best approach in Section 5.3.

From the table it can be seen that fine-tuning is surprisingly robust to different masking strategies. However, as expected, using only the MASK strategy was problematic when applying the feature-based approach to NER. Interestingly, using only the RND strategy performs much worse than our strategy as well.

Improving Language Understanding by Generative Pre-Training

Alec Radford
OpenAI
alec@openai.com

Karthik Narasimhan
OpenAI
karthikn@openai.com

Tim Salimans
OpenAI
tim@openai.com

Ilya Sutskever
OpenAI
ilyasu@openai.com

Abstract

Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification. Although large unlabeled text corpora are abundant, labeled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. We demonstrate that large gains on these tasks can be realized by *generative pre-training* of a language model on a diverse corpus of unlabeled text, followed by *discriminative fine-tuning* on each specific task. In contrast to previous approaches, we make use of task-aware input transformations during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. We demonstrate the effectiveness of our approach on a wide range of benchmarks for natural language understanding. Our general task-agnostic model outperforms discriminatively trained models that use architectures specifically crafted for each task, significantly improving upon the state of the art in 9 out of the 12 tasks studied. For instance, we achieve absolute improvements of 8.9% on commonsense reasoning (Stories Cloze Test), 5.7% on question answering (RACE), and 1.5% on textual entailment (MultiNLI).

1 Introduction

The ability to learn effectively from raw text is crucial to alleviating the dependence on supervised learning in natural language processing (NLP). Most deep learning methods require substantial amounts of manually labeled data, which restricts their applicability in many domains that suffer from a dearth of annotated resources [61]. In these situations, models that can leverage linguistic information from unlabeled data provide a valuable alternative to gathering more annotation, which can be time-consuming and expensive. Further, even in cases where considerable supervision is available, learning good representations in an unsupervised fashion can provide a significant performance boost. The most compelling evidence for this so far has been the extensive use of pre-trained word embeddings [10, 39, 42] to improve performance on a range of NLP tasks [8, 11, 26, 45].

Leveraging more than word-level information from unlabeled text, however, is challenging for two main reasons. First, it is unclear what type of optimization objectives are most effective at learning text representations that are useful for transfer. Recent research has looked at various objectives such as language modeling [44], machine translation [38], and discourse coherence [22], with each method outperforming the others on different tasks.[1] Second, there is no consensus on the most effective way to transfer these learned representations to the target task. Existing techniques involve a combination of making task-specific changes to the model architecture [43, 44], using intricate learning schemes [21] and adding auxiliary learning objectives [50]. These uncertainties have made it difficult to develop effective semi-supervised learning approaches for language processing.

[1] https://gluebenchmark.com/leaderboard

Preprint. Work in progress.

In this paper, we explore a semi-supervised approach for language understanding tasks using a combination of unsupervised pre-training and supervised fine-tuning. Our goal is to learn a universal representation that transfers with little adaptation to a wide range of tasks. We assume access to a large corpus of unlabeled text and several datasets with manually annotated training examples (target tasks). Our setup does not require these target tasks to be in the same domain as the unlabeled corpus. We employ a two-stage training procedure. First, we use a language modeling objective on the unlabeled data to learn the initial parameters of a neural network model. Subsequently, we adapt these parameters to a target task using the corresponding supervised objective.

For our model architecture, we use the *Transformer* [62], which has been shown to perform strongly on various tasks such as machine translation [62], document generation [34], and syntactic parsing [29]. This model choice provides us with a more structured memory for handling long-term dependencies in text, compared to alternatives like recurrent networks, resulting in robust transfer performance across diverse tasks. During transfer, we utilize task-specific input adaptations derived from traversal-style approaches [52], which process structured text input as a single contiguous sequence of tokens. As we demonstrate in our experiments, these adaptations enable us to fine-tune effectively with minimal changes to the architecture of the pre-trained model.

We evaluate our approach on four types of language understanding tasks – natural language inference, question answering, semantic similarity, and text classification. Our general task-agnostic model outperforms discriminatively trained models that employ architectures specifically crafted for each task, significantly improving upon the state of the art in 9 out of the 12 tasks studied. For instance, we achieve absolute improvements of 8.9% on commonsense reasoning (Stories Cloze Test) [40], 5.7% on question answering (RACE) [30], 1.5% on textual entailment (MultiNLI) [66] and 5.5% on the recently introduced GLUE multi-task benchmark [64]. We also analyzed zero-shot behaviors of the pre-trained model on four different settings and demonstrate that it acquires useful linguistic knowledge for downstream tasks.

2 Related Work

Semi-supervised learning for NLP Our work broadly falls under the category of semi-supervised learning for natural language. This paradigm has attracted significant interest, with applications to tasks like sequence labeling [24, 33, 57] or text classification [41, 70]. The earliest approaches used unlabeled data to compute word-level or phrase-level statistics, which were then used as features in a supervised model [33]. Over the last few years, researchers have demonstrated the benefits of using word embeddings [11, 39, 42], which are trained on unlabeled corpora, to improve performance on a variety of tasks [8, 11, 26, 45]. These approaches, however, mainly transfer word-level information, whereas we aim to capture higher-level semantics.

Recent approaches have investigated learning and utilizing more than word-level semantics from unlabeled data. Phrase-level or sentence-level embeddings, which can be trained using an unlabeled corpus, have been used to encode text into suitable vector representations for various target tasks [28, 32, 1, 36, 22, 12, 56, 31].

Unsupervised pre-training Unsupervised pre-training is a special case of semi-supervised learning where the goal is to find a good initialization point instead of modifying the supervised learning objective. Early works explored the use of the technique in image classification [20, 49, 63] and regression tasks [3]. Subsequent research [15] demonstrated that pre-training acts as a regularization scheme, enabling better generalization in deep neural networks. In recent work, the method has been used to help train deep neural networks on various tasks like image classification [69], speech recognition [68], entity disambiguation [17] and machine translation [48].

The closest line of work to ours involves pre-training a neural network using a language modeling objective and then fine-tuning it on a target task with supervision. Dai et al. [13] and Howard and Ruder [21] follow this method to improve text classification. However, although the pre-training phase helps capture some linguistic information, their usage of LSTM models restricts their prediction ability to a short range. In contrast, our choice of transformer networks allows us to capture longer-range linguistic structure, as demonstrated in our experiments. Further, we also demonstrate the effectiveness of our model on a wider range of tasks including natural language inference, paraphrase detection and story completion. Other approaches [43, 44, 38] use hidden representations from a

pre-trained language or machine translation model as auxiliary features while training a supervised model on the target task. This involves a substantial amount of new parameters for each separate target task, whereas we require minimal changes to our model architecture during transfer.

Auxiliary training objectives Adding auxiliary unsupervised training objectives is an alternative form of semi-supervised learning. Early work by Collobert and Weston [10] used a wide variety of auxiliary NLP tasks such as POS tagging, chunking, named entity recognition, and language modeling to improve semantic role labeling. More recently, Rei [50] added an auxiliary language modeling objective to their target task objective and demonstrated performance gains on sequence labeling tasks. Our experiments also use an auxiliary objective, but as we show, unsupervised pre-training already learns several linguistic aspects relevant to target tasks.

3 Framework

Our training procedure consists of two stages. The first stage is learning a high-capacity language model on a large corpus of text. This is followed by a fine-tuning stage, where we adapt the model to a discriminative task with labeled data.

3.1 Unsupervised pre-training

Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$, we use a standard language modeling objective to maximize the following likelihood:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta) \tag{1}$$

where k is the size of the context window, and the conditional probability P is modeled using a neural network with parameters Θ. These parameters are trained using stochastic gradient descent [51].

In our experiments, we use a multi-layer *Transformer decoder* [34] for the language model, which is a variant of the transformer [62]. This model applies a multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers to produce an output distribution over target tokens:

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\ P(u) &= \texttt{softmax}(h_n W_e^T) \end{aligned} \tag{2}$$

where $U = (u_{-k}, \ldots, u_{-1})$ is the context vector of tokens, n is the number of layers, W_e is the token embedding matrix, and W_p is the position embedding matrix.

3.2 Supervised fine-tuning

After training the model with the objective in Eq. 1, we adapt the parameters to the supervised target task. We assume a labeled dataset $\mathcal{C}$, where each instance consists of a sequence of input tokens, $x^1, \ldots, x^m$, along with a label y. The inputs are passed through our pre-trained model to obtain the final transformer block's activation h_l^m, which is then fed into an added linear output layer with parameters W_y to predict y:

$$P(y | x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y). \tag{3}$$

This gives us the following objective to maximize:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y | x^1, \ldots, x^m). \tag{4}$$

We additionally found that including language modeling as an auxiliary objective to the fine-tuning helped learning by (a) improving generalization of the supervised model, and (b) accelerating convergence. This is in line with prior work [50, 43], who also observed improved performance with such an auxiliary objective. Specifically, we optimize the following objective (with weight λ):

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C}) \tag{5}$$

Overall, the only extra parameters we require during fine-tuning are W_y, and embeddings for delimiter tokens (described below in Section 3.3).

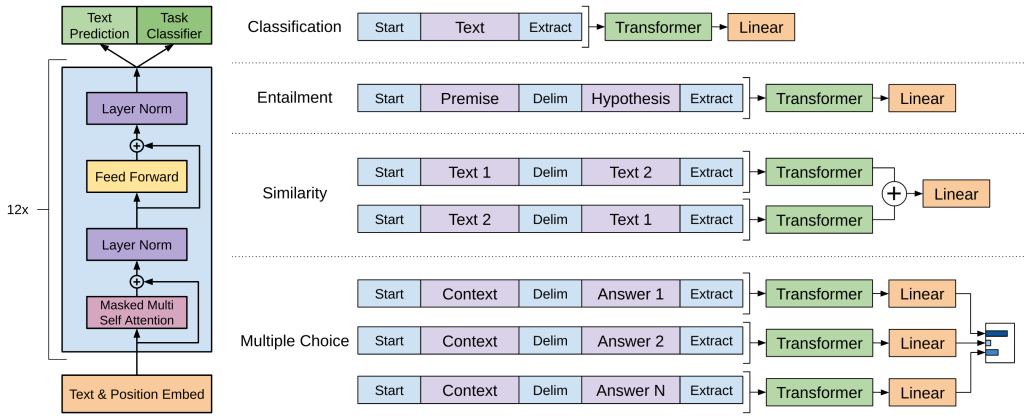


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

3.3 Task-specific input transformations

For some tasks, like text classification, we can directly fine-tune our model as described above. Certain other tasks, like question answering or textual entailment, have structured inputs such as ordered sentence pairs, or triplets of document, question, and answers. Since our pre-trained model was trained on contiguous sequences of text, we require some modifications to apply it to these tasks. Previous work proposed learning task specific architectures on top of transferred representations [44]. Such an approach re-introduces a significant amount of task-specific customization and does not use transfer learning for these additional architectural components. Instead, we use a traversal-style approach [52], where we convert structured inputs into an ordered sequence that our pre-trained model can process. These input transformations allow us to avoid making extensive changes to the architecture across tasks. We provide a brief description of these input transformations below and Figure 1 provides a visual illustration. All transformations include adding randomly initialized start and end tokens ($\langle s \rangle$, $\langle e \rangle$).

Textual entailment For entailment tasks, we concatenate the premise p and hypothesis h token sequences, with a delimiter token ($) in between.

Similarity For similarity tasks, there is no inherent ordering of the two sentences being compared. To reflect this, we modify the input sequence to contain both possible sentence orderings (with a delimiter in between) and process each independently to produce two sequence representations h_l^m which are added element-wise before being fed into the linear output layer.

Question Answering and Commonsense Reasoning For these tasks, we are given a context document z, a question q, and a set of possible answers $\{a_k\}$. We concatenate the document context and question with each possible answer, adding a delimiter token in between to get $[z; q; \$; a_k]$. Each of these sequences are processed independently with our model and then normalized via a softmax layer to produce an output distribution over possible answers.

4 Experiments

4.1 Setup

Unsupervised pre-training We use the BooksCorpus dataset [71] for training the language model. It contains over 7,000 unique unpublished books from a variety of genres including Adventure, Fantasy, and Romance. Crucially, it contains long stretches of contiguous text, which allows the generative model to learn to condition on long-range information. An alternative dataset, the 1B Word Benchmark, which is used by a similar approach, ELMo [44], is approximately the same size

Table 1: A list of the different tasks and datasets used in our experiments.

Task	Datasets
Natural language inference	SNLI [5], MultiNLI [66], Question NLI [64], RTE [4], SciTail [25]
Question Answering	RACE [30], Story Cloze [40]
Sentence similarity	MSR Paraphrase Corpus [14], Quora Question Pairs [9], STS Benchmark [6]
Classification	Stanford Sentiment Treebank-2 [54], CoLA [65]

but is shuffled at a sentence level - destroying long-range structure. Our language model achieves a very low token level perplexity of 18.4 on this corpus.

Model specifications Our model largely follows the original transformer work [62]. We trained a 12-layer decoder-only transformer with masked self-attention heads (768 dimensional states and 12 attention heads). For the position-wise feed-forward networks, we used 3072 dimensional inner states. We used the Adam optimization scheme [27] with a max learning rate of 2.5e-4. The learning rate was increased linearly from zero over the first 2000 updates and annealed to 0 using a cosine schedule. We train for 100 epochs on minibatches of 64 randomly sampled, contiguous sequences of 512 tokens. Since layernorm [2] is used extensively throughout the model, a simple weight initialization of $N(0, 0.02)$ was sufficient. We used a bytepair encoding (BPE) vocabulary with 40,000 merges [53] and residual, embedding, and attention dropouts with a rate of 0.1 for regularization. We also employed a modified version of L2 regularization proposed in [37], with $w = 0.01$ on all non bias or gain weights. For the activation function, we used the Gaussian Error Linear Unit (GELU) [18]. We used learned position embeddings instead of the sinusoidal version proposed in the original work. We use the *ftfy* library[2] to clean the raw text in BooksCorpus, standardize some punctuation and whitespace, and use the *spaCy* tokenizer.[3]

Fine-tuning details Unless specified, we reuse the hyperparameter settings from unsupervised pre-training. We add dropout to the classifier with a rate of 0.1. For most tasks, we use a learning rate of 6.25e-5 and a batchsize of 32. Our model finetunes quickly and 3 epochs of training was sufficient for most cases. We use a linear learning rate decay schedule with warmup over 0.2% of training. λ was set to 0.5.

4.2 Supervised fine-tuning

We perform experiments on a variety of supervised tasks including natural language inference, question answering, semantic similarity, and text classification. Some of these tasks are available as part of the recently released GLUE multi-task benchmark [64], which we make use of. Figure 1 provides an overview of all the tasks and datasets.

Natural Language Inference The task of natural language inference (NLI), also known as recognizing textual entailment, involves reading a pair of sentences and judging the relationship between them from one of `entailment`, `contradiction` or `neutral`. Although there has been a lot of recent interest [58, 35, 44], the task remains challenging due to the presence of a wide variety of phenomena like lexical entailment, coreference, and lexical and syntactic ambiguity. We evaluate on five datasets with diverse sources, including image captions (SNLI), transcribed speech, popular fiction, and government reports (MNLI), Wikipedia articles (QNLI), science exams (SciTail) or news articles (RTE).

Table 2 details various results on the different NLI tasks for our model and previous state-of-the-art approaches. Our method significantly outperforms the baselines on four of the five datasets, achieving absolute improvements of upto 1.5% on MNLI, 5% on SciTail, 5.8% on QNLI and 0.6% on SNLI over the previous best results. This demonstrates our model's ability to better reason over multiple sentences, and handle aspects of linguistic ambiguity. On RTE, one of the smaller datasets we evaluate on (2490 examples), we achieve an accuracy of 56%, which is below the 61.7% reported by a multi-task biLSTM model. Given the strong performance of our approach on larger NLI datasets, it is likely our model will benefit from multi-task training as well but we have not explored this currently.

[2] `https://ftfy.readthedocs.io/en/latest/`

[3] `https://spacy.io/`

Table 2: Experimental results on natural language inference tasks, comparing our model with current state-of-the-art methods. 5x indicates an ensemble of 5 models. All datasets use accuracy as the evaluation metric.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

Table 3: Results on question answering and commonsense reasoning, comparing our model with current state-of-the-art methods.. 9x means an ensemble of 9 models.

Method	Story Cloze	RACE-m	RACE-h	RACE
val-LS-skip [55]	76.5	-	-	-
Hidden Coherence Model [7]	77.6	-	-	-
Dynamic Fusion Net [67] (9x)	-	55.6	49.4	51.2
BiAttention MRU [59] (9x)	-	60.2	50.3	53.3
Finetuned Transformer LM (ours)	**86.5**	**62.9**	**57.4**	**59.0**

Question answering and commonsense reasoning Another task that requires aspects of single and multi-sentence reasoning is question answering. We use the recently released RACE dataset [30], consisting of English passages with associated questions from middle and high school exams. This corpus has been shown to contain more reasoning type questions that other datasets like CNN [19] or SQuaD [47], providing the perfect evaluation for our model which is trained to handle long-range contexts. In addition, we evaluate on the Story Cloze Test [40], which involves selecting the correct ending to multi-sentence stories from two options. On these tasks, our model again outperforms the previous best results by significant margins - up to 8.9% on Story Cloze, and 5.7% overall on RACE. This demonstrates the ability of our model to handle long-range contexts effectively.

Semantic Similarity Semantic similarity (or paraphrase detection) tasks involve predicting whether two sentences are semantically equivalent or not. The challenges lie in recognizing rephrasing of concepts, understanding negation, and handling syntactic ambiguity. We use three datasets for this task – the Microsoft Paraphrase corpus (MRPC) [14] (collected from news sources), the Quora Question Pairs (QQP) dataset [9], and the Semantic Textual Similarity benchmark (STS-B) [6]. We obtain state-of-the-art results on two of the three semantic similarity tasks (Table 4) with a 1 point absolute gain on STS-B. The performance delta on QQP is significant, with a 4.2% absolute improvement over Single-task BiLSTM + ELMo + Attn.

Classification Finally, we also evaluate on two different text classification tasks. The Corpus of Linguistic Acceptability (CoLA) [65] contains expert judgements on whether a sentence is grammatical or not, and tests the innate linguistic bias of trained models. The Stanford Sentiment Treebank (SST-2) [54], on the other hand, is a standard binary classification task. Our model obtains an score of 45.4 on CoLA, which is an especially big jump over the previous best result of 35.0, showcasing the innate linguistic bias learned by our model. The model also achieves 91.3% accuracy on SST-2, which is competitive with the state-of-the-art results. We also achieve an overall score of 72.8 on the GLUE benchmark, which is significantly better than the previous best of 68.9.

Table 4: Semantic similarity and classification results, comparing our model with current state-of-the-art methods. All task evaluations in this table were done using the GLUE benchmark. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

Method	**Classification**		**Semantic Similarity**			**GLUE**
	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	
Sparse byte mLSTM [16]	-	**93.2**	-	-	-	-
TF-KLD [23]	-	-	**86.0**	-	-	-
ECNU (mixed ensemble) [60]	-	-	-	81.0	-	-
Single-task BiLSTM + ELMo + Attn [64]	35.0	90.2	80.2	55.5	66.1	64.8
Multi-task BiLSTM + ELMo + Attn [64]	18.9	91.6	83.5	72.8	63.3	68.9
Finetuned Transformer LM (ours)	**45.4**	91.3	82.3	**82.0**	**70.3**	**72.8**

Overall, our approach achieves new state-of-the-art results in 9 out of the 12 datasets we evaluate on, outperforming ensembles in many cases. Our results also indicate that our approach works well across datasets of different sizes, from smaller datasets such as STS-B (≈5.7k training examples) – to the largest one – SNLI (≈550k training examples).

5 Analysis

Impact of number of layers transferred We observed the impact of transferring a variable number of layers from unsupervised pre-training to the supervised target task. Figure 2(left) illustrates the performance of our approach on MultiNLI and RACE as a function of the number of layers transferred. We observe the standard result that transferring embeddings improves performance and that each transformer layer provides further benefits up to 9% for full transfer on MultiNLI. This indicates that each layer in the pre-trained model contains useful functionality for solving target tasks.

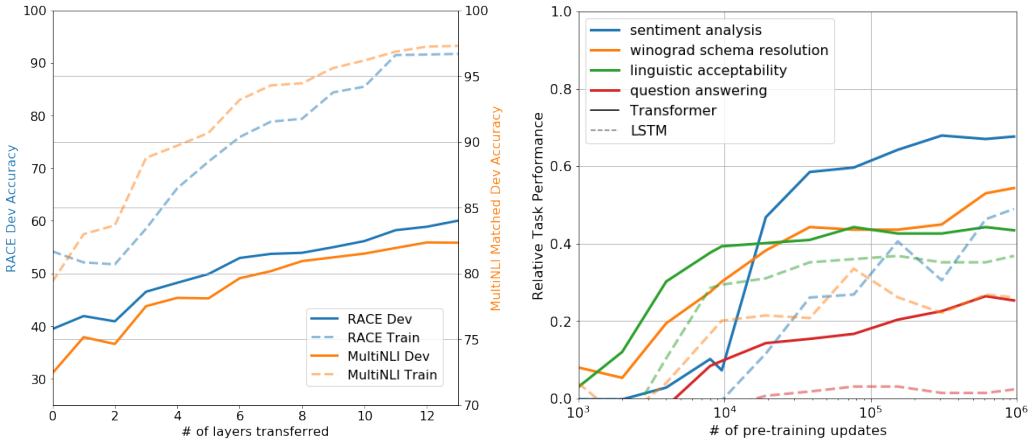


Figure 2: (**left**) Effect of transferring increasing number of layers from the pre-trained language model on RACE and MultiNLI. (**right**) Plot showing the evolution of zero-shot performance on different tasks as a function of LM pre-training updates. Performance per task is normalized between a random guess baseline and the current state-of-the-art with a single model.

Zero-shot Behaviors We'd like to better understand why language model pre-training of transformers is effective. A hypothesis is that the underlying generative model learns to perform many of the tasks we evaluate on in order to improve its language modeling capability and that the more structured

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

attentional memory of the transformer assists in transfer compared to LSTMs. We designed a series of heuristic solutions that use the underlying generative model to perform tasks without supervised finetuning. We visualize the effectiveness of these heuristic solutions over the course of generative pre-training in Fig 2(right). We observe the performance of these heuristics is stable and steadily increases over training suggesting that generative pretraining supports the learning of a wide variety of task relevant functionality. We also observe the LSTM exhibits higher variance in its zero-shot performance suggesting that the inductive bias of the Transformer architecture assists in transfer.

For CoLA (linguistic acceptability), examples are scored as the average token log-probability the generative model assigns and predictions are made by thresholding. For SST-2 (sentiment analysis), we append the token *very* to each example and restrict the language model's output distribution to only the words *positive* and *negative* and guess the token it assigns higher probability to as the prediction. For RACE (question answering), we pick the answer the generative model assigns the highest average token log-probability when conditioned on the document and question. For DPRD [46] (winograd schemas), we replace the definite pronoun with the two possible referrents and predict the resolution that the generative model assigns higher average token log-probability to the rest of the sequence after the substitution.

Ablation studies We perform three different ablation studies (Table 5). First, we examine the performance of our method without the auxiliary LM objective during fine-tuning. We observe that the auxiliary objective helps on the NLI tasks and QQP. Overall, the trend suggests that larger datasets benefit from the auxiliary objective but smaller datasets do not. Second, we analyze the effect of the Transformer by comparing it with a single layer 2048 unit LSTM using the same framework. We observe a 5.6 average score drop when using the LSTM instead of the Transformer. The LSTM only outperforms the Transformer on one dataset – MRPC. Finally, we also compare with our transformer architecture directly trained on supervised target tasks, without pre-training. We observe that the lack of pre-training hurts performance across all the tasks, resulting in a 14.8% decrease compared to our full model.

6 Conclusion

We introduced a framework for achieving strong natural language understanding with a single task-agnostic model through generative pre-training and discriminative fine-tuning. By pre-training on a diverse corpus with long stretches of contiguous text our model acquires significant world knowledge and ability to process long-range dependencies which are then successfully transferred to solving discriminative tasks such as question answering, semantic similarity assessment, entailment determination, and text classification, improving the state of the art on 9 of the 12 datasets we study. Using unsupervised (pre-)training to boost performance on discriminative tasks has long been an important goal of Machine Learning research. Our work suggests that achieving significant performance gains is indeed possible, and offers hints as to what models (Transformers) and data sets (text with long range dependencies) work best with this approach. We hope that this will help enable new research into unsupervised learning, for both natural language understanding and other domains, further improving our understanding of how and when unsupervised learning works.

References

[1] S. Arora, Y. Liang, and T. Ma. A simple but tough-to-beat baseline for sentence embeddings. 2016.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[4] L. Bentivogli, P. Clark, I. Dagan, and D. Giampiccolo. The fifth pascal recognizing textual entailment challenge. In *TAC*, 2009.

[5] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. *EMNLP*, 2015.

[6] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

[7] S. Chaturvedi, H. Peng, and D. Roth. Story comprehension for predicting what happens next. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1603–1614, 2017.

[8] D. Chen and C. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

[9] Z. Chen, H. Zhang, X. Zhang, and L. Zhao. Quora question pairs. https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs, 2018.

[10] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[11] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[12] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes. Supervised learning of universal sentence representations from natural language inference data. *EMNLP*, 2017.

[13] A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pages 3079–3087, 2015.

[14] W. B. Dolan and C. Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[15] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[16] S. Gray, A. Radford, and K. P. Diederik. Gpu kernels for block-sparse weights. 2017.

[17] Z. He, S. Liu, M. Li, M. Zhou, L. Zhang, and H. Wang. Learning entity representation for entity disambiguation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 30–34, 2013.

[18] D. Hendrycks and K. Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv preprint arXiv:1606.08415*, 2016.

[19] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

[20] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[21] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. *Association for Computational Linguistics (ACL)*, 2018.

[22] Y. Jernite, S. R. Bowman, and D. Sontag. Discourse-based objectives for fast unsupervised sentence representation learning. *arXiv preprint arXiv:1705.00557*, 2017.

[23] Y. Ji and J. Eisenstein. Discriminative improvements to distributional sentence similarity. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 891–896, 2013.

[24] F. Jiao, S. Wang, C.-H. Lee, R. Greiner, and D. Schuurmans. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 209–216. Association for Computational Linguistics, 2006.

[25] T. Khot, A. Sabharwal, and P. Clark. Scitail: A textual entailment dataset from science question answering. In *Proceedings of AAAI*, 2018.

[26] Y. Kim. Convolutional neural networks for sentence classification. *EMNLP*, 2014.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

[29] N. Kitaev and D. Klein. Constituency parsing with a self-attentive encoder. *ACL*, 2018.

[30] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy. Race: Large-scale reading comprehension dataset from examinations. *EMNLP*, 2017.

[31] G. Lample, L. Denoyer, and M. Ranzato. Unsupervised machine translation using monolingual corpora only. *ICLR*, 2018.

[32] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[33] P. Liang. *Semi-supervised learning for natural language*. PhD thesis, Massachusetts Institute of Technology, 2005.

[34] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating wikipedia by summarizing long sequences. *ICLR*, 2018.

[35] X. Liu, K. Duh, and J. Gao. Stochastic answer networks for natural language inference. *arXiv preprint arXiv:1804.07888*, 2018.

[36] L. Logeswaran and H. Lee. An efficient framework for learning sentence representations. *ICLR*, 2018.

[37] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

[38] B. McCann, J. Bradbury, C. Xiong, and R. Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6297–6308, 2017.

[39] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[40] N. Mostafazadeh, M. Roth, A. Louis, N. Chambers, and J. Allen. Lsdsem 2017 shared task: The story cloze test. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pages 46–51, 2017.

[41] K. Nigam, A. McCallum, and T. Mitchell. Semi-supervised text classification using em. *Semi-Supervised Learning*, pages 33–56, 2006.

[42] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[43] M. E. Peters, W. Ammar, C. Bhagavatula, and R. Power. Semi-supervised sequence tagging with bidirectional language models. *ACL*, 2017.

[44] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *NAACL*, 2018.

[45] Y. Qi, D. S. Sachan, M. Felix, S. J. Padmanabhan, and G. Neubig. When and why are pre-trained word embeddings useful for neural machine translation? *NAACL*, 2018.

[46] A. Rahman and V. Ng. Resolving complex cases of definite pronouns: the winograd schema challenge. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 777–789. Association for Computational Linguistics, 2012.

[47] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *EMNLP*, 2016.

[48] P. Ramachandran, P. J. Liu, and Q. V. Le. Unsupervised pretraining for sequence to sequence learning. *arXiv preprint arXiv:1611.02683*, 2016.

[49] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2007.

[50] M. Rei. Semi-supervised multitask learning for sequence labeling. *ACL*, 2017.

[51] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[52] T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, and P. Blunsom. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*, 2015.

[53] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[54] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[55] S. Srinivasan, R. Arora, and M. Riedl. A simple and effective approach to the story cloze test. *arXiv preprint arXiv:1803.05547*, 2018.

[56] S. Subramanian, A. Trischler, Y. Bengio, and C. J. Pal. Learning general purpose distributed sentence representations via large scale multi-task learning. *arXiv preprint arXiv:1804.00079*, 2018.

[57] J. Suzuki and H. Isozaki. Semi-supervised sequential labeling and segmentation using giga-word scale unlabeled data. *Proceedings of ACL-08: HLT*, pages 665–673, 2008.

[58] Y. Tay, L. A. Tuan, and S. C. Hui. A compare-propagate architecture with alignment factorization for natural language inference. *arXiv preprint arXiv:1801.00102*, 2017.

[59] Y. Tay, L. A. Tuan, and S. C. Hui. Multi-range reasoning for machine comprehension. *arXiv preprint arXiv:1803.09074*, 2018.

[60] J. Tian, Z. Zhou, M. Lan, and Y. Wu. Ecnu at semeval-2017 task 1: Leverage kernel-based traditional nlp features and neural networks to build a universal model for multilingual and cross-lingual semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 191–197, 2017.

[61] Y. Tsvetkov. Opportunities and challenges in working with low-resource languages. *CMU*, 2017.

[62] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

[63] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[64] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[65] A. Warstadt, A. Singh, and S. R. Bowman. Corpus of linguistic acceptability. http://nyu-mll.github.io/cola, 2018.

[66] A. Williams, N. Nangia, and S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *NAACL*, 2018.

[67] Y. Xu, J. Liu, J. Gao, Y. Shen, and X. Liu. Towards human-level machine reading comprehension: Reasoning and inference with multiple strategies. *arXiv preprint arXiv:1711.04964*, 2017.

[68] D. Yu, L. Deng, and G. Dahl. Roles of pre-training and fine-tuning in context-dependent dbn-hmms for real-world speech recognition. In *Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[69] R. Zhang, P. Isola, and A. A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *CVPR*, volume 1, page 6, 2017.

[70] X. Zhu. Semi-supervised learning literature survey. 2005.

[71] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] **Jeffrey Wu** [*1] **Rewon Child** [1] **David Luan** [1] **Dario Amodei** [**1] **Ilya Sutskever** [**1]

Abstract

Natural language processing tasks, such as question answering, machine translation, reading comprehension, and summarization, are typically approached with supervised learning on task-specific datasets. We demonstrate that language models begin to learn these tasks without any explicit supervision when trained on a new dataset of millions of webpages called WebText. When conditioned on a document plus questions, the answers generated by the language model reach 55 F1 on the CoQA dataset - matching or exceeding the performance of 3 out of 4 baseline systems without using the 127,000+ training examples. The capacity of the language model is essential to the success of zero-shot task transfer and increasing it improves performance in a log-linear fashion across tasks. Our largest model, GPT-2, is a 1.5B parameter Transformer that achieves state of the art results on 7 out of 8 tested language modeling datasets in a zero-shot setting but still underfits WebText. Samples from the model reflect these improvements and contain coherent paragraphs of text. These findings suggest a promising path towards building language processing systems which learn to perform tasks from their naturally occurring demonstrations.

1. Introduction

Machine learning systems now excel (in expectation) at tasks they are trained for by using a combination of large datasets, high-capacity models, and supervised learning (Krizhevsky et al., 2012) (Sutskever et al., 2014) (Amodei et al., 2016). Yet these systems are brittle and sensitive to slight changes in the data distribution (Recht et al., 2018) and task specification (Kirkpatrick et al., 2017). Current systems are better characterized as narrow experts rather than competent generalists. We would like to move towards more general systems which can perform many tasks – eventually without the need to manually create and label a training dataset for each one.

The dominant approach to creating ML systems is to collect a dataset of training examples demonstrating correct behavior for a desired task, train a system to imitate these behaviors, and then test its performance on independent and identically distributed (IID) held-out examples. This has served well to make progress on narrow experts. But the often erratic behavior of captioning models (Lake et al., 2017), reading comprehension systems (Jia & Liang, 2017), and image classifiers (Alcorn et al., 2018) on the diversity and variety of possible inputs highlights some of the shortcomings of this approach.

Our suspicion is that the prevalence of single task training on single domain datasets is a major contributor to the lack of generalization observed in current systems. Progress towards robust systems with current architectures is likely to require training and measuring performance on a wide range of domains and tasks. Recently, several benchmarks have been proposed such as GLUE (Wang et al., 2018) and decaNLP (McCann et al., 2018) to begin studying this.

Multitask learning (Caruana, 1997) is a promising framework for improving general performance. However, multitask training in NLP is still nascent. Recent work reports modest performance improvements (Yogatama et al., 2019) and the two most ambitious efforts to date have trained on a total of 10 and 17 `(dataset, objective)` pairs respectively (McCann et al., 2018) (Bowman et al., 2018). From a meta-learning perspective, each `(dataset, objective)` pair is a single training example sampled from the distribution of datasets and objectives. Current ML systems need hundreds to thousands of examples to induce functions which generalize well. This suggests that multitask training many need just as many effective training pairs to realize its promise with current approaches. It will be very difficult to continue to scale the creation of datasets and the design of objectives to the degree that may be required to brute force our way there with current techniques. This motivates exploring additional setups for performing multitask learning.

The current best performing systems on language tasks

[*,**]Equal contribution [1]OpenAI, San Francisco, California, United States. Correspondence to: Alec Radford <alec@openai.com>.

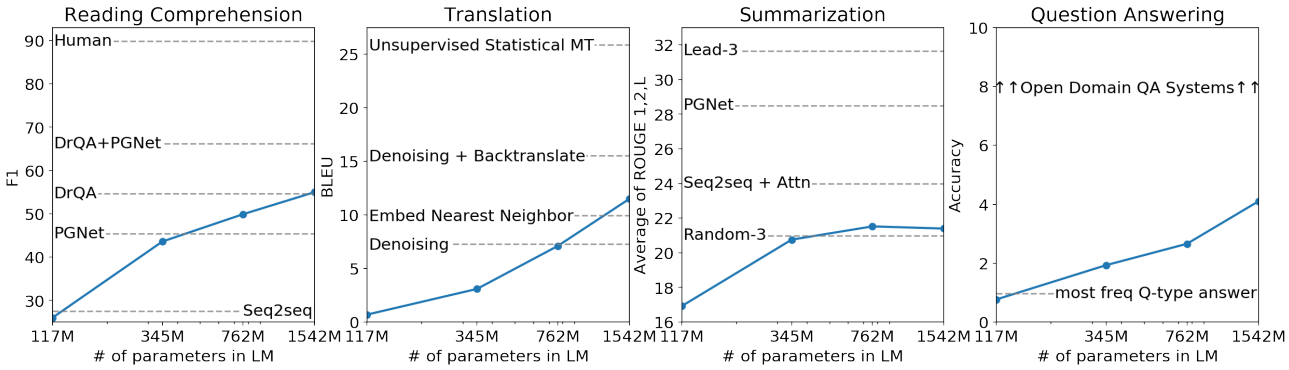


Figure 1. Zero-shot task performance of WebText LMs as a function of model size on many NLP tasks. Reading Comprehension results are on CoQA (Reddy et al., 2018), translation on WMT-14 Fr-En (Artetxe et al., 2017), summarization on CNN and Daily Mail (See et al., 2017), and Question Answering on Natural Questions (Kwiatkowski et al., 2019). Section 3 contains detailed descriptions of each result.

utilize a combination of pre-training and supervised fine-tuning. This approach has a long history with a trend towards more flexible forms of transfer. First, word vectors were learned and used as inputs to task-specific architectures (Mikolov et al., 2013) (Collobert et al., 2011), then the contextual representations of recurrent networks were transferred (Dai & Le, 2015) (Peters et al., 2018), and recent work suggests that task-specific architectures are no longer necessary and transferring many self-attention blocks is sufficient (Radford et al., 2018) (Devlin et al., 2018).

These methods still require supervised training in order to perform a task. When only minimal or no supervised data is available, another line of work has demonstrated the promise of language models to perform specific tasks, such as commonsense reasoning (Schwartz et al., 2017) and sentiment analysis (Radford et al., 2017).

In this paper, we connect these two lines of work and continue the trend of more general methods of transfer. We demonstrate language models can perform down-stream tasks in a zero-shot setting – without any parameter or architecture modification. We demonstrate this approach shows potential by highlighting the ability of language models to perform a wide range of tasks in a zero-shot setting. We achieve promising, competitive, and state of the art results depending on the task.

2. Approach

At the core of our approach is language modeling. Language modeling is usually framed as unsupervised distribution estimation from a set of examples $(x_1, x_2, ..., x_n)$ each composed of variable length sequences of symbols $(s_1, s_2, ..., s_n)$. Since language has a natural sequential ordering, it is common to factorize the joint probabilities over symbols as the product of conditional probabilities (Jelinek & Mercer, 1980) (Bengio et al., 2003):

$$p(x) = \prod_{i=1}^{n} p(s_n|s_1, ..., s_{n-1}) \quad (1)$$

This approach allows for tractable sampling from and estimation of $p(x)$ as well as any conditionals of the form $p(s_{n-k}, ..., s_n|s_1, ..., s_{n-k-1})$. In recent years, there have been significant improvements in the expressiveness of models that can compute these conditional probabilities, such as self-attention architectures like the Transformer (Vaswani et al., 2017).

Learning to perform a single task can be expressed in a probabilistic framework as estimating a conditional distribution $p(output|input)$. Since a general system should be able to perform many different tasks, even for the same input, it should condition not only on the input but also on the task to be performed. That is, it should model $p(output|input, task)$. This has been variously formalized in multitask and meta-learning settings. Task conditioning is often implemented at an architectural level, such as the task specific encoders and decoders in (Kaiser et al., 2017) or at an algorithmic level such as the inner and outer loop optimization framework of MAML (Finn et al., 2017). But as exemplified in McCann et al. (2018), language provides a flexible way to specify tasks, inputs, and outputs all as a sequence of symbols. For example, a translation training example can be written as the sequence `(translate to french, english text, french text)`. Likewise, a reading comprehension training example can be written as `(answer the question, document, question, answer)`. McCann et al. (2018) demonstrated it was possible to train a single model, the MQAN,

to infer and perform many different tasks on examples with this type of format.

Language modeling is also able to, in principle, learn the tasks of McCann et al. (2018) without the need for explicit supervision of which symbols are the outputs to be predicted. Since the supervised objective is the the same as the unsupervised objective but only evaluated on a subset of the sequence, the global minimum of the unsupervised objective is also the global minimum of the supervised objective. In this slightly toy setting, the concerns with density estimation as a principled training objective discussed in (Sutskever et al., 2015) are side stepped. The problem instead becomes whether we are able to, in practice, optimize the unsupervised objective to convergence. Preliminary experiments confirmed that sufficiently large language models are able to perform multitask learning in this toy-ish setup but learning is much slower than in explicitly supervised approaches.

While it is a large step from the well-posed setup described above to the messiness of "language in the wild", Weston (2016) argues, in the context of dialog, for the need to develop systems capable of learning from natural language directly and demonstrated a proof of concept – learning a QA task without a reward signal by using forward prediction of a teacher's outputs. While dialog is an attractive approach, we worry it is overly restrictive. The internet contains a vast amount of information that is passively available without the need for interactive communication. Our speculation is that a language model with sufficient capacity will begin to learn to infer and perform the tasks demonstrated in natural language sequences in order to better predict them, regardless of their method of procurement. If a language model is able to do this it will be, in effect, performing unsupervised multitask learning. We test whether this is the case by analyzing the performance of language models in a zero-shot setting on a wide variety of tasks.

2.1. Training Dataset

Most prior work trained language models on a single domain of text, such as news articles (Jozefowicz et al., 2016), Wikipedia (Merity et al., 2016), or fiction books (Kiros et al., 2015). Our approach motivates building as large and diverse a dataset as possible in order to collect natural language demonstrations of tasks in as varied of domains and contexts as possible.

A promising source of diverse and nearly unlimited text is web scrapes such as Common Crawl. While these archives are many orders of magnitude larger than current language modeling datasets, they have significant data quality issues. Trinh & Le (2018) used Common Crawl in their work on commonsense reasoning but noted a large amount of documents "whose content are mostly unintelligible". We observed similar data issues in our initial experiments with Common Crawl. Trinh & Le (2018)'s best results were achieved using a small subsample of Common Crawl which included only documents most similar to their target dataset, the Winograd Schema Challenge. While this is a pragmatic approach to improve performance on a specific task, we want to avoid making assumptions about the tasks to be performed ahead of time.

"I'm not the cleverest man in the world, but like they say in French: **Je ne suis pas un imbecile [I'm not a fool].**
In a now-deleted post from Aug. 16, Soheil Eid, Tory candidate in the riding of Joliette, wrote in French: "**Mentez mentez, il en restera toujours quelque chose**," which translates as, "**Lie lie and something will always remain.**"
"I hate the word '**perfume**,"' Burr says. 'It's somewhat better in French: '**parfum**.'
If listened carefully at 29:55, a conversation can be heard between two guys in French: "**-Comment on fait pour aller de l'autre coté? -Quel autre coté?**", which means "**- How do you get to the other side? - What side?**".
If this sounds like a bit of a stretch, consider this question in French: **As-tu aller au cinéma?**, or **Did you go to the movies?**, which literally translates as Have-you to go to movies/theater?
"**Brevet Sans Garantie Du Gouvernement**", translated to English: "**Patented without government warranty**".

Table 1. Examples of naturally occurring demonstrations of English to French and French to English translation found throughout the WebText training set.

Instead, we created a new web scrape which emphasizes document quality. To do this we only scraped web pages which have been curated/filtered by humans. Manually filtering a full web scrape would be exceptionally expensive so as a starting point, we scraped all outbound links from Reddit, a social media platform, which received at least 3 karma. This can be thought of as a heuristic indicator for whether other users found the link interesting, educational, or just funny.

The resulting dataset, WebText, contains the text subset of these 45 million links. To extract the text from HTML responses we use a combination of the Dragnet (Peters & Lecocq, 2013) and Newspaper[1] content extractors. All results presented in this paper use a preliminary version of WebText which does not include links created after Dec 2017 and which after de-duplication and some heuristic based cleaning contains slightly over 8 million documents for a total of 40 GB of text. We removed all Wikipedia documents from WebText since it is a common data source for other datasets and could complicate analysis due to over-

[1] https://github.com/codelucas/newspaper

lapping training data with test evaluation tasks.

2.2. Input Representation

A general language model (LM) should be able to compute the probability of (and also generate) any string. Current large scale LMs include pre-processing steps such as lowercasing, tokenization, and out-of-vocabulary tokens which restrict the space of model-able strings. While processing Unicode strings as a sequence of UTF-8 bytes elegantly fulfills this requirement as exemplified in work such as Gillick et al. (2015), current byte-level LMs are not competitive with word-level LMs on large scale datasets such as the One Billion Word Benchmark (Al-Rfou et al., 2018). We observed a similar performance gap in our own attempts to train standard byte-level LMs on WebText.

Byte Pair Encoding (BPE) (Sennrich et al., 2015) is a practical middle ground between character and word level language modeling which effectively interpolates between word level inputs for frequent symbol sequences and character level inputs for infrequent symbol sequences. Despite its name, reference BPE implementations often operate on Unicode code points and not byte sequences. These implementations would require including the full space of Unicode symbols in order to model all Unicode strings. This would result in a base vocabulary of over 130,000 before any multi-symbol tokens are added. This is prohibitively large compared to the 32,000 to 64,000 token vocabularies often used with BPE. In contrast, a byte-level version of BPE only requires a base vocabulary of size 256. However, directly applying BPE to the byte sequence results in suboptimal merges due to BPE using a greedy frequency based heuristic for building the token vocabulary. We observed BPE including many versions of common words like `dog` since they occur in many variations such as `dog. dog! dog?` . This results in a sub-optimal allocation of limited vocabulary slots and model capacity. To avoid this, we prevent BPE from merging across character categories for any byte sequence. We add an exception for spaces which significantly improves the compression efficiency while adding only minimal fragmentation of words across multiple vocab tokens.

This input representation allows us to combine the empirical benefits of word-level LMs with the generality of byte-level approaches. Since our approach can assign a probability to any Unicode string, this allows us to evaluate our LMs on any dataset regardless of pre-processing, tokenization, or vocab size.

2.3. Model

We use a Transformer (Vaswani et al., 2017) based architecture for our LMs. The model largely follows the details of the OpenAI GPT model (Radford et al., 2018) with a few modifications. Layer normalization (Ba et al., 2016) was moved to the input of each sub-block, similar to a pre-activation residual network (He et al., 2016) and an additional layer normalization was added after the final self-attention block. A modified initialization which accounts for the accumulation on the residual path with model depth is used. We scale the weights of residual layers at initialization by a factor of $1/\sqrt{N}$ where N is the number of residual layers. The vocabulary is expanded to 50,257. We also increase the context size from 512 to 1024 tokens and a larger batchsize of 512 is used.

Parameters	Layers	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

Table 2. Architecture hyperparameters for the 4 model sizes.

3. Experiments

We trained and benchmarked four LMs with approximately log-uniformly spaced sizes. The architectures are summarized in Table 2. The smallest model is equivalent to the original GPT, and the second smallest equivalent to the largest model from BERT (Devlin et al., 2018). Our largest model, which we call GPT-2, has over an order of magnitude more parameters than GPT. The learning rate of each model was manually tuned for the best perplexity on a 5% held-out sample of WebText. All models still underfit WebText and held-out perplexity has as of yet improved given more training time.

3.1. Language Modeling

As an initial step towards zero-shot task transfer, we are interested in understanding how WebText LM's perform at zero-shot domain transfer on the primary task they are trained for – language modeling. Since our model operates on a byte level and does not require lossy pre-processing or tokenization, we can evaluate it on any language model benchmark. Results on language modeling datasets are commonly reported in a quantity which is a scaled or exponentiated version of the average negative log probability per canonical prediction unit - usually a character, a byte, or a word. We evaluate the same quantity by computing the log-probability of a dataset according to a WebText LM and dividing by the number of canonical units. For many of these datasets, WebText LMs would be tested significantly out-of-distribution, having to predict aggressively standardized text, tokenization artifacts such as disconnected punctuation and contractions, shuffled sentences, and even the string

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**	**40.31**	**0.97**	**1.02**	22.05	44.575
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**	42.16

Table 3. Zero-shot results on many datasets. No training or fine-tuning was performed for any of these results. PTB and WikiText-2 results are from (Gong et al., 2018). CBT results are from (Bajgar et al., 2016). LAMBADA accuracy result is from (Hoang et al., 2018) and LAMBADA perplexity result is from (Grave et al., 2016). Other results are from (Dai et al., 2019).

<UNK> which is extremely rare in WebText - occurring only 26 times in 40 billion bytes. We report our main results in Table 3 using invertible de-tokenizers which remove as many of these tokenization / pre-processing artifacts as possible. Since these de-tokenizers are invertible, we can still calculate the log probability of a dataset and they can be thought of as a simple form of domain adaptation. We observe gains of 2.5 to 5 perplexity for GPT-2 with these de-tokenizers.

WebText LMs transfer well across domains and datasets, improving the state of the art on 7 out of the 8 datasets in a zero-shot setting. Large improvements are noticed on small datasets such as Penn Treebank and WikiText-2 which have only 1 to 2 million training tokens. Large improvements are also noticed on datasets created to measure long-term dependencies like LAMBADA (Paperno et al., 2016) and the Children's Book Test (Hill et al., 2015). Our model is still significantly worse than prior work on the One Billion Word Benchmark (Chelba et al., 2013). This is likely due to a combination of it being both the largest dataset and having some of the most destructive pre-processing - 1BW's sentence level shuffling removes all long-range structure.

3.2. Children's Book Test

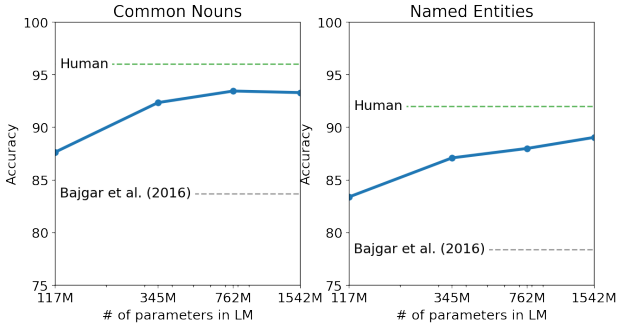


Figure 2. Performance on the Children's Book Test as a function of model capacity. Human performance are from Bajgar et al. (2016), instead of the much lower estimates from the original paper.

The Children's Book Test (CBT) (Hill et al., 2015) was created to examine the performance of LMs on different categories of words: named entities, nouns, verbs, and prepositions. Rather than reporting perplexity as an evaluation metric, CBT reports accuracy on an automatically constructed cloze test where the task is to predict which of 10 possible choices for an omitted word is correct. Following the LM approach introduced in the original paper, we compute the probability of each choice and the rest of the sentence conditioned on this choice according to the LM, and predict the one with the highest probability. As seen in Figure 2 performance steadily improves as model size is increased and closes the majority of the gap to human performance on this test. Data overlap analysis showed one of the CBT test set books, The Jungle Book by Rudyard Kipling, is in WebText, so we report results on the validation set which has no significant overlap. GPT-2 achieves new state of the art results of 93.3% on common nouns and 89.1% on named entities. A de-tokenizer was applied to remove PTB style tokenization artifacts from CBT.

3.3. LAMBADA

The LAMBADA dataset (Paperno et al., 2016) tests the ability of systems to model long-range dependencies in text. The task is to predict the final word of sentences which require at least 50 tokens of context for a human to successfully predict. GPT-2 improves the state of the art from 99.8 (Grave et al., 2016) to 8.6 perplexity and increases the accuracy of LMs on this test from 19% (Dehghani et al., 2018) to 52.66%. Investigating GPT-2's errors showed most predictions are valid continuations of the sentence, but are not valid final words. This suggests that the LM is not using the additional useful constraint that the word must be the final of the sentence. Adding a stop-word filter as an approximation to this further increases accuracy to 63.24%, improving the overall state of the art on this task by 4%. The previous state of the art (Hoang et al., 2018) used a different restricted prediction setting where the outputs of the model were constrained to only words that appeared in the context. For GPT-2, this restriction is harmful rather than helpful

since 19% of answers are not in context. We use a version of the dataset without preprocessing.

3.4. Winograd Schema Challenge

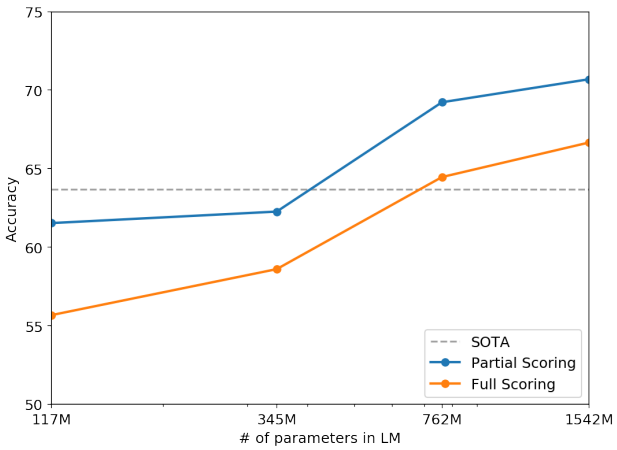


Figure 3. Performance on the Winograd Schema Challenge as a function of model capacity.

The Winograd Schema challenge (Levesque et al., 2012) was constructed to measure the capability of a system to perform commonsense reasoning by measuring its ability to resolve ambiguities in text. Recently Trinh & Le (2018) demonstrated significant progress on this challenge using LMs, by predicting the resolution of the ambiguity with higher probability. We follow their problem formulation and visualize the performance of our models with both full and partial scoring techniques in Figure 3. GPT-2 improves state of the art accuracy by 7%, achieving 70.70%. The dataset is quite small with only 273 examples so we recommend reading Trichelair et al. (2018) to help contextualize this result.

3.5. Reading Comprehension

The Conversation Question Answering dataset (CoQA) Reddy et al. (2018) consists of documents from 7 different domains paired with natural language dialogues between a question asker and a question answerer about the document. CoQA tests reading comprehension capabilities and also the ability of models to answer questions that depend on conversation history (such as "Why?").

Greedy decoding from GPT-2 when conditioned on a document, the history of the associated conversation, and a final token `A:` achieves 55 F1 on the development set. This matches or exceeds the performance of 3 out of 4 baseline systems without using the 127,000+ manually collected question answer pairs those baselines were trained on. The supervised SOTA, a BERT based system (Devlin et al., 2018), is nearing the 89 F1 performance of humans. While GPT-2's performance is exciting for a system without any supervised training, some inspection of its answers and errors suggests GPT-2 often uses simple retrieval based heuristics such as *answer with a name from the document in response to a who question*.

	R-1	R-2	R-L	R-AVG
Bottom-Up Sum	**41.22**	**18.68**	**38.34**	**32.75**
Lede-3	40.38	17.66	36.62	31.55
Seq2Seq + Attn	31.33	11.81	28.83	23.99
GPT-2 `TL;DR:`	29.34	8.27	26.58	21.40
Random-3	28.78	8.63	25.52	20.98
GPT-2 no hint	21.58	4.03	19.47	15.03

Table 4. Summarization performance as measured by ROUGE F1 metrics on the CNN and Daily Mail dataset. Bottom-Up Sum is the SOTA model from (Gehrmann et al., 2018)

3.6. Summarization

We test GPT-2's ability to perform summarization on the CNN and Daily Mail dataset (Nallapati et al., 2016). To induce summarization behavior we add the text `TL;DR:` after the article and generate 100 tokens with Top-k random sampling (Fan et al., 2018) with $k = 2$ which reduces repetition and encourages more abstractive summaries than greedy decoding. We use the first 3 generated sentences in these 100 tokens as the summary. While qualitatively the generations resemble summaries, as shown in Table 14, they often focus on recent content from the article or confuse specific details such as how many cars were involved in a crash or whether a logo was on a hat or shirt. On the commonly reported ROUGE 1,2,L metrics the generated summaries only begin to approach the performance of classic neural baselines and just barely outperforms selecting 3 random sentences from the article. GPT-2's performance drops by 6.4 points on the aggregate metric when the task hint is removed which demonstrates the ability to invoke task specific behavior in a language model with natural language.

3.7. Translation

We test whether GPT-2 has begun to learn how to translate from one language to another. In order to help it infer that this is the desired task, we condition the language model on a context of example pairs of the format `english sentence = french sentence` and then after a final prompt of `english sentence =` we sample from the model with greedy decoding and use the first generated sentence as the translation. On the WMT-14 English-French test set, GPT-2 gets 5 BLEU, which is slightly worse than a word-by-word substitution with a bilingual lexicon inferred in previous work on unsupervised word translation

Question	Generated Answer	Correct	Probability
Who wrote the book the origin of species?	Charles Darwin	✓	83.4%
Who is the founder of the ubuntu project?	Mark Shuttleworth	✓	82.0%
Who is the quarterback for the green bay packers?	Aaron Rodgers	✓	81.1%
Panda is a national animal of which country?	China	✓	76.8%
Who came up with the theory of relativity?	Albert Einstein	✓	76.4%
When was the first star wars film released?	1977	✓	71.4%
What is the most common blood type in sweden?	A	✗	70.6%
Who is regarded as the founder of psychoanalysis?	Sigmund Freud	✓	69.3%
Who took the first steps on the moon in 1969?	Neil Armstrong	✓	66.8%
Who is the largest supermarket chain in the uk?	Tesco	✓	65.3%
What is the meaning of shalom in english?	peace	✓	64.0%
Who was the author of the art of war?	Sun Tzu	✓	59.6%
Largest state in the us by land mass?	California	✗	59.2%
Green algae is an example of which type of reproduction?	parthenogenesis	✗	56.5%
Vikram samvat calender is official in which country?	India	✓	55.6%
Who is mostly responsible for writing the declaration of independence?	Thomas Jefferson	✓	53.3%
What us state forms the western boundary of montana?	Montana	✗	52.3%
Who plays ser davos in game of thrones?	Peter Dinklage	✗	52.1%
Who appoints the chair of the federal reserve system?	Janet Yellen	✗	51.5%
State the process that divides one nucleus into two genetically identical nuclei?	mitosis	✓	50.7%
Who won the most mvp awards in the nba?	Michael Jordan	✗	50.2%
What river is associated with the city of rome?	the Tiber	✓	48.6%
Who is the first president to be impeached?	Andrew Johnson	✓	48.3%
Who is the head of the department of homeland security 2017?	John Kelly	✓	47.0%
What is the name given to the common currency to the european union?	Euro	✓	46.8%
What was the emperor name in star wars?	Palpatine	✓	46.5%
Do you have to have a gun permit to shoot at a range?	No	✓	46.4%
Who proposed evolution in 1859 as the basis of biological development?	Charles Darwin	✓	45.7%
Nuclear power plant that blew up in russia?	Chernobyl	✓	45.7%
Who played john connor in the original terminator?	Arnold Schwarzenegger	✗	45.2%

Table 5. The 30 most confident answers generated by GPT-2 on the development set of Natural Questions sorted by their probability according to GPT-2. None of these questions appear in WebText according to the procedure described in Section 4.

(Conneau et al., 2017b). On the WMT-14 French-English test set, GPT-2 is able to leverage its very strong English language model to perform significantly better, achieving 11.5 BLEU. This outperforms several unsupervised machine translation baselines from (Artetxe et al., 2017) and (Lample et al., 2017) but is still much worse than the 33.5 BLEU of the current best unsupervised machine translation approach (Artetxe et al., 2019). Performance on this task was surprising to us, since we deliberately removed non-English webpages from WebText as a filtering step. In order to confirm this, we ran a byte-level language detector[2] on WebText which detected only 10MB of data in the French language which is approximately 500x smaller than the monolingual French corpus common in prior unsupervised machine translation research.

3.8. Question Answering

A potential way to test what information is contained within a language model is to evaluate how often it generates the correct answer to factoid-style questions. Previous showcasing of this behavior in neural systems where all information is stored in parameters such as *A Neural Conversational Model* (Vinyals & Le, 2015) reported qualitative results due to the lack of high-quality evaluation datasets. The recently introduced Natural Questions dataset (Kwiatkowski et al., 2019) is a promising resource to test this more quantitatively. Similar to translation, the context of the language model is seeded with example question answer pairs which helps the model infer the short answer style of the dataset. GPT-2 answers 4.1% of questions correctly when evaluated by the exact match metric commonly used on reading comprehension datasets like SQUAD.[3] As a comparison point, the smallest model does not exceed the 1.0% accuracy of an incredibly simple baseline which returns the most common answer for each question type (who, what, where, etc...). GPT-2 answers 5.3 times more questions correctly, suggesting that model capacity has been a major factor in the poor performance of neural systems on this kind of task as of yet. The probability GPT-2 assigns to its generated answers is well calibrated and GPT-2 has an accuracy of 63.1% on the 1% of questions it is most confident in. The 30 most confident answers generated by GPT-2 on development set questions are shown in Table 5. The performance of GPT-2 is still much, much, worse than the 30 to 50% range of open domain question answering systems which hybridize information retrieval with extractive document question answering (Alberti et al., 2019).

[2] https://github.com/CLD2Owners/cld2

[3] Alec, who previously thought of himself as good at random trivia, answered 17 of 100 randomly sampled examples correctly when tested in the same setting as GPT-2. He actually only got 14 right but he should have gotten those other 3

	PTB	WikiText-2	enwik8	text8	Wikitext-103	1BW
Dataset train	**2.67%**	0.66%	**7.50%**	2.34%	**9.09%**	**13.19%**
WebText train	0.88%	**1.63%**	6.31%	**3.94%**	2.42%	3.75%

Table 6. Percentage of test set 8 grams overlapping with training sets.

4. Generalization vs Memorization

Recent work in computer vision has shown that common image datasets contain a non-trivial amount of near-duplicate images. For instance CIFAR-10 has 3.3% overlap between train and test images (Barz & Denzler, 2019). This results in an over-reporting of the generalization performance of machine learning systems. As the size of datasets increases this issue becomes increasingly likely which suggests a similar phenomena could be happening with WebText. Therefore it is important to analyze how much test data also shows up in the training data.

To study this we created Bloom filters containing 8-grams of WebText training set tokens. To improve recall, strings were normalized to contain only lower-cased alphanumeric words with a single space as a delimiter. The Bloom filters were constructed such that the false positive rate is upper bounded by $\frac{1}{10^8}$. We further verified the low false positive rate by generating 1M strings, of which zero were found by the filter.

These Bloom filters let us calculate, given a dataset, the percentage of 8-grams from that dataset that are also found in the WebText training set. Table 6 shows this overlap analysis for the test sets of common LM benchmarks. Common LM datasets' test sets have between 1-6% overlap with WebText train, with an average of overlap of 3.2%. Somewhat surprisingly, many datasets have larger overlaps with their own training splits, with an average of 5.9% overlap.

Our approach optimizes for recall, and while manual inspection of the overlaps shows many common phrases, there are many longer matches that are due to duplicated data. This is not unique to WebText. For instance, we discovered that the test set of WikiText-103 has an article which is also in the training dataset. Since there are only 60 articles in the test set there is at least an overlap of 1.6%.[4] Potentially more worryingly, 1BW has an overlap of nearly 13.2% with its own training set according to our procedure.

For the Winograd Schema Challenge, we found only 10 schemata which had any 8-gram overlaps with the WebText training set. Of these, 2 were spurious matches. Of the remaining 8, only 1 schema appeared in any contexts that gave away the answer.

For CoQA, about 15% of documents in the news domain are already in WebText and the model performs about 3 F1 better on these. CoQA's development set metric reports the average performance over 5 different domains and we measure a gain of about 0.5-1.0 F1 due to overlap across the various domains. However, no actual training questions or answers are in WebText since CoQA was released after the cutoff date for links in WebText.

On LAMBADA, the average overlap is 1.2%. GPT-2 performs about 2 perplexity better on examples with greater than 15% overlap. Recalculating metrics when excluding all examples with any overlap shifts results from 8.6 to 8.7 perplexity and reduces accuracy from 63.2% to 62.9%. This very small change in overall results is likely due to only 1 in 200 examples having significant overlap.

Overall, our analysis suggests that data overlap between WebText training data and specific evaluation datasets provides a small but consistent benefit to reported results. However, for most datasets we do not notice significantly larger overlaps than those already existing between standard training and test sets, as Table 6 highlights.

Understanding and quantifying how highly similar text impacts performance is an important research question. Better de-duplication techniques such as scalable fuzzy matching could also help better answer these questions. For now, we recommend the use of n-gram overlap based de-duplication as an important verification step and sanity check during the creation of training and test splits for new NLP datasets.

Another potential way of determining whether the performance of WebText LMs is attributable to memorization is inspecting their performance on their own held-out set. As shown in Figure 4, performance on both the training and test sets of WebText are similar and improve together as model size is increased. This suggests even GPT-2 is still underfitting on WebText in many ways.

GPT-2 is also able to write news articles about the discovery of talking unicorns. An example is provided in Table 13.

5. Related Work

A significant portion of this work measured the performance of larger language models trained on larger datasets. This

[4] A significant portion of additional overlap is due to editors reusing some paragraphs across multiple articles with a shared theme such as various battles in the Korean War.

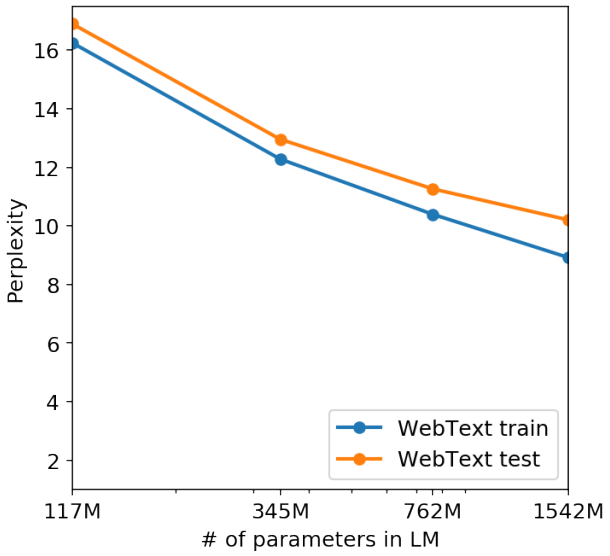


Figure 4. The performance of LMs trained on WebText as a function of model size.

is similar to the work of Jozefowicz et al. (2016) which scaled RNN based language models on the 1 Billion Word Benchmark. Bajgar et al. (2016) also previously improved results on the Children's Book Test by creating a much larger training dataset out of Project Gutenberg to supplement the standard training dataset. Hestness et al. (2017) conducted a thorough analysis of how the performance of various deep learning models changes as a function of both model capacity and dataset size. Our experiments, while much noisier across tasks, suggest similar trends hold for sub-tasks of an objective and continue into the 1B+ parameter regime.

Interesting learned functionality in generative models has been documented before such as the cells in an RNN language model performing line-width tracking and quote/comment detection Karpathy et al. (2015). More inspirational to our work was the observation of Liu et al. (2018) that a model trained to generate Wikipedia articles also learned to translate names between languages.

Previous work has explored alternative approaches to filtering and constructing a large text corpus of web pages, such as the iWeb Corpus (Davies, 2018).

There has been extensive work on pre-training methods for language tasks. In addition to those mentioned in the introduction, GloVe (Pennington et al., 2014) scaled word vector representation learning to all of Common Crawl. An influential early work on deep representation learning for text was *Skip-thought Vectors* (Kiros et al., 2015). McCann et al. (2017) explored the use of representations derived from machine translation models and Howard & Ruder (2018) improved the RNN based fine-tuning approaches of (Dai & Le, 2015). (Conneau et al., 2017a) studied the transfer performance of representations learned by natural language inference models and (Subramanian et al., 2018) explored large-scale multitask training.

(Ramachandran et al., 2016) demonstrated that seq2seq models benefit from being initialized with pre-trained language models as encoders and decoders. More recent work has shown that LM pre-training is helpful when fine-tuned for difficult generation tasks like chit-chat dialog and dialog based question answering systems as well (Wolf et al., 2019) (Dinan et al., 2018).

6. Discussion

Much research has been dedicated to learning (Hill et al., 2016), understanding (Levy & Goldberg, 2014), and critically evaluating (Wieting & Kiela, 2019) the representations of both supervised and unsupervised pre-training methods. Our results suggest that unsupervised task learning is an additional promising area of research to explore. These findings potentially help explain the widespread success of pre-training techniques for down-stream NLP tasks as we show that, in the limit, one of these pre-training techniques begins to learn to perform tasks directly without the need for supervised adaption or modification.

On reading comprehension the performance of GPT-2 is competitive with supervised baselines in a zero-shot setting. However, on other tasks such as summarization, while it is qualitatively performing the task, its performance is still only rudimentary according to quantitative metrics. While suggestive as a research result, in terms of practical applications, the zero-shot performance of GPT-2 is still far from use-able.

We have studied the zero-shot performance of WebText LMs on many canonical NLP tasks, but there are many additional tasks that could be evaluated. There are undoubtedly many practical tasks where the performance of GPT-2 is still no better than random. Even on common tasks that we evaluated on, such as question answering and translation, language models only begin to outperform trivial baselines when they have sufficient capacity.

While zero-shot performance establishes a baseline of the potential performance of GPT-2 on many tasks, it is not clear where the ceiling is with finetuning. On some tasks, GPT-2's fully abstractive output is a significant departure from the extractive pointer network (Vinyals et al., 2015) based outputs which are currently state of the art on many question answering and reading comprehension datasets. Given the prior success of fine-tuning GPT, we plan to investigate fine-tuning on benchmarks such as decaNLP and GLUE, especially since it is unclear whether the additional

training data and capacity of GPT-2 is sufficient to overcome the inefficiencies of uni-directional representations demonstrated by BERT (Devlin et al., 2018).

7. Conclusion

When a large language model is trained on a sufficiently large and diverse dataset it is able to perform well across many domains and datasets. GPT-2 zero-shots to state of the art performance on 7 out of 8 tested language modeling datasets. The diversity of tasks the model is able to perform in a zero-shot setting suggests that high-capacity models trained to maximize the likelihood of a sufficiently varied text corpus begin to learn how to perform a surprising amount of tasks without the need for explicit supervision.[5]

Acknowledgements

Thanks to everyone who wrote the text, shared the links, and upvoted the content in WebText. Many millions of people were involved in creating the data that GPT-2 was trained on. Also thanks to all the Googlers who helped us with training infrastructure, including Zak Stone, JS Riehl, Jonathan Hseu, Russell Power, Youlong Cheng, Noam Shazeer, Solomon Boulos, Michael Banfield, Aman Gupta, Daniel Sohn, and many more. Finally thanks to the people who gave feedback on drafts of the paper: Jacob Steinhardt, Sam Bowman, Geoffrey Irving, and Madison May.

References

Al-Rfou, R., Choe, D., Constant, N., Guo, M., and Jones, L. Character-level language modeling with deeper self-attention. *arXiv preprint arXiv:1808.04444*, 2018.

Alberti, C., Lee, K., and Collins, M. A bert baseline for the natural questions. *arXiv preprint arXiv:1901.08634*, 2019.

Alcorn, M. A., Li, Q., Gong, Z., Wang, C., Mai, L., Ku, W.-S., and Nguyen, A. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. *arXiv preprint arXiv:1811.11553*, 2018.

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pp. 173–182, 2016.

Artetxe, M., Labaka, G., Agirre, E., and Cho, K. Unsupervised neural machine translation. *arXiv preprint arXiv:1710.11041*, 2017.

Artetxe, M., Labaka, G., and Agirre, E. An effective approach to unsupervised machine translation. *arXiv preprint arXiv:1902.01313*, 2019.

[5]Preliminary code for downloading and using the small model is available at https://github.com/openai/gpt-2

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bajgar, O., Kadlec, R., and Kleindienst, J. Embracing data abundance: Booktest dataset for reading comprehension. *arXiv preprint arXiv:1610.00956*, 2016.

Barz, B. and Denzler, J. Do we train on test data? purging cifar of near-duplicates. *arXiv preprint arXiv:1902.00423*, 2019.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Bowman, S. R., Pavlick, E., Grave, E., Van Durme, B., Wang, A., Hula, J., Xia, P., Pappagari, R., McCoy, R. T., Patel, R., et al. Looking for elmo's friends: Sentence-level pretraining beyond language modeling. *arXiv preprint arXiv:1812.10860*, 2018.

Caruana, R. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*, 2017a.

Conneau, A., Lample, G., Ranzato, M., Denoyer, L., and Jégou, H. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017b.

Dai, A. M. and Le, Q. V. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pp. 3079–3087, 2015.

Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Davies, M. The 14 billion word iweb corpus. *https://corpus.byu.edu/iWeb/*, 2018.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dinan, E., Roller, S., Shuster, K., Fan, A., Auli, M., and Weston, J. Wizard of wikipedia: Knowledge-powered conversational agents. *arXiv preprint arXiv:1811.01241*, 2018.

Fan, A., Lewis, M., and Dauphin, Y. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Gehrmann, S., Deng, Y., and Rush, A. M. Bottom-up abstractive summarization. *arXiv preprint arXiv:1808.10792*, 2018.

Gillick, D., Brunk, C., Vinyals, O., and Subramanya, A. Multilingual language processing from bytes. *arXiv preprint arXiv:1512.00103*, 2015.

Gong, C., He, D., Tan, X., Qin, T., Wang, L., and Liu, T.-Y. Frage: frequency-agnostic word representation. In *Advances in Neural Information Processing Systems*, pp. 1341–1352, 2018.

Grave, E., Joulin, A., and Usunier, N. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M., Ali, M., Yang, Y., and Zhou, Y. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Hill, F., Bordes, A., Chopra, S., and Weston, J. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301*, 2015.

Hill, F., Cho, K., and Korhonen, A. Learning distributed representations of sentences from unlabelled data. *arXiv preprint arXiv:1602.03483*, 2016.

Hoang, L., Wiseman, S., and Rush, A. M. Entity tracking improves cloze-style reading comprehension. *arXiv preprint arXiv:1810.02891*, 2018.

Howard, J. and Ruder, S. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pp. 328–339, 2018.

Jelinek, F. and Mercer, R. L. Interpolated estimation of markov source parameters from sparse data. *In Proceedings of the Workshop on Pattern Recognition in Practice, Amsterdam, The Netherlands: North-Holland, May.*, 1980.

Jia, R. and Liang, P. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

Karpathy, A., Johnson, J., and Fei-Fei, L. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, pp. 201611835, 2017.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. Skip-thought vectors. In *Advances in neural information processing systems*, pp. 3294–3302, 2015.

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.

Kwiatkowski, T., Palomaki, J., Rhinehart, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Kelcey, M., Devlin, J., et al. Natural questions: a benchmark for question answering research. 2019.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

Lample, G., Conneau, A., Denoyer, L., and Ranzato, M. Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*, 2017.

Levesque, H., Davis, E., and Morgenstern, L. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2012.

Levy, O. and Goldberg, Y. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pp. 2177–2185, 2014.

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., and Shazeer, N. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

McCann, B., Bradbury, J., Xiong, C., and Socher, R. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pp. 6294–6305, 2017.

McCann, B., Keskar, N. S., Xiong, C., and Socher, R. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Pennington, J., Socher, R., and Manning, C. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Peters, M. E. and Lecocq, D. Content extraction using diverse feature sets. In *Proceedings of the 22nd International Conference on World Wide Web*, pp. 89–90. ACM, 2013.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Radford, A., Jozefowicz, R., and Sutskever, I. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. 2018.

Ramachandran, P., Liu, P. J., and Le, Q. V. Unsupervised pretraining for sequence to sequence learning. *arXiv preprint arXiv:1611.02683*, 2016.

Recht, B., Roelofs, R., Schmidt, L., and Shankar, V. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

Reddy, S., Chen, D., and Manning, C. D. Coqa: A conversational question answering challenge. *arXiv preprint arXiv:1808.07042*, 2018.

Schwartz, R., Sap, M., Konstas, I., Zilles, L., Choi, Y., and Smith, N. A. Story cloze task: Uw nlp system. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pp. 52–55, 2017.

See, A., Liu, P. J., and Manning, C. D. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Subramanian, S., Trischler, A., Bengio, Y., and Pal, C. J. Learning general purpose distributed sentence representations via large scale multi-task learning. *arXiv preprint arXiv:1804.00079*, 2018.

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.

Sutskever, I., Jozefowicz, R., Gregor, K., Rezende, D., Lillicrap, T., and Vinyals, O. Towards principled unsupervised learning. *arXiv preprint arXiv:1511.06440*, 2015.

Trichelair, P., Emami, A., Cheung, J. C. K., Trischler, A., Suleman, K., and Diaz, F. On the evaluation of common-sense reasoning in natural language understanding. *arXiv preprint arXiv:1811.01778*, 2018.

Trinh, T. H. and Le, Q. V. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Vinyals, O. and Le, Q. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *Advances in Neural Information Processing Systems*, pp. 2692–2700, 2015.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Weston, J. E. Dialog-based language learning. In *Advances in Neural Information Processing Systems*, pp. 829–837, 2016.

Wieting, J. and Kiela, D. No training required: Exploring random encoders for sentence classification. *arXiv preprint arXiv:1901.10444*, 2019.

Wolf, T., Sanh, V., Chaumond, J., and Delangue, C. Transfertransfo: A transfer learning approach for neural network based conversational agents. *arXiv preprint arXiv:1901.08149*, 2019.

Yogatama, D., d'Autume, C. d. M., Connor, J., Kocisky, T., Chrzanowski, M., Kong, L., Lazaridou, A., Ling, W., Yu, L., Dyer, C., et al. Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*, 2019.

8. Appendix A: Samples

8.1. Model capacity

To complement the reported perplexity gains of bigger LMs on WebText show in Figure 4, Tables 7 through 11 show side-by-side completions of the smallest WebText LM and GPT-2 on random unseen WebText test set articles.

8.2. Text Memorization

We observe some memorizing behavior in GPT-2 on longer strings that are repeated many times in the dataset such as famous quotes or speeches. For example, when conditioned on the first sentence and a half of the Gettysburg Address (which occurs approximately 40 times throughout WebText), an argmax decode from GPT-2 recovers the speech. Even when sampling without truncation, we find that the model copies the speech for awhile before drifting, albeit in a similar style. It typically drifts within 100-200 tokens, and displays widening diversity once it drifts.

To quantify how often exact memorization shows up in samples, we generated samples from GPT-2 conditioned on WebText test set articles and compared the overlap rates of GPT-2's generations to the overlap rates of the ground-truth completions. The results of this analysis are shown below and suggest that GPT-2 repeats text from the training set less often then the baseline rate of held-out articles.

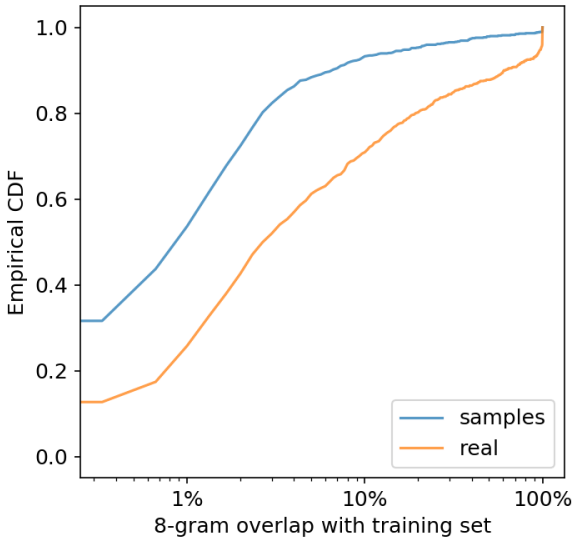


Figure 5. CDF of percentage 8-gram overlap with WebText training set, for both WebText test set and samples (conditioned on WebText test set, with top-k truncated random sampling with $k = 40$). Most samples have less than 1% overlap, including over 30% of samples with no overlap, whereas the median for test set is 2.6% overlap.

8.3. Diversity

Table 12 shows multiple completions of the same random WebText test set context, showing the diversity of completions with standard sampling settings.

8.4. Robustness

Table 13 shows the previously mentioned talking unicorns news article. We find the model to be capable of handling out of distribution contexts, but the quality of these samples is generally lower.

Context (WebText test)

Outfit 1: Typical

This pairing was the first outfit I thought of when I bought the shoes. It's like a summer version of this Jake Grantham outfit; in fact, my shoes are close to the colors of his Nike Racers! Instead of a heavy Harris Tweed jacket and denim shirt, I'm wearing a cotton DB jacket and and a linen shirt. Both fabrics (in these colors) are an absolute must for summer, as they go with both dark and and light pants! As you can see, they pair wonderfully with the dark jeans and shoes. It's a pseudo menswear/prep outfit.

Overall, this is a very casual outfit which is why I paired my sneakers with it. I'm not about wearing a full wool suit with sneakers (as GQ shows a lot) but I'm definitely open to keeping things casual, like this cotton DB. Casual fabrics are key to pulling off your sneakers in a dressed down menswear outfit. I'd even suggest to wear these sneakers with a khaki chino suit or a white linen suit. Just be sure to ditch the tie or wear a tee or polo; wearing a tie with sneakers is a bit too much

Small model completion	**GPT-2 completion**
.	for me!
Outfit 2: High End	Outfit 2: Casual Jacket
This outfit was just in the eye of the storm. It's a hybrid of what ASICS does with streetwear (like J. Crew) and what J. Crew does with casual pieces that are a bit more traditional. Again, as I mentioned before, it's a low budget item and they can be very expensive. These will definitely be an issue if you're trying to work as a casual sneakermaker, and that's OK but don't hesitate if you want to try the lower price versions of their outfits!	I know that some people might be opposed to wearing sneakers in a turtleneck, but I wanted to be true to myself, so I went with a slim fitting turtleneck in a color more similar to my favorite color of the day. Overall, it's a classic turtleneck, with nothing too flashy. I did swap my shoes for these ASICS Tiger Killshots and I wish I hadn't did that. The shoe itself is very comfortable, however, I found that wearing them in these sneakers made my feet look a bit larger. I'd wear them with denim, chinos, whatever, but would never recommend wearing them alone. There are just too many variables involved.
This version features a very low cut-cut neckline (which is not a bad thing), slightly longer legs, and a pair of long legs to keep things nice. My preferred color of color these items are blue-gray (which is my first colors, as they are a light white), or black (which is my favorite of my designs). As always, my pick of colors from this store have nothing to do with that other than the design. For a high end, I would recommend this to everyone!	Outfit 3: Dress
Outfit 3: High Light	This is the final outfit I wore the ASICS Tiger Killshots. This version of the ASICS is a much more modern and streamlined version. The shoe has the most modern and minimal design that I can think of! With a lot of black, these sneakers look very cool. They'll definitely be very comfortable for summer.
This was a "high light" look while I had a chance to get some color. I think the color is based on	To me, wearing them with a suit is the easiest way to have a good pair of sneaker shoes. That's pretty much it! If

Table 7. Random unseen contexts (top), and non-cherry-picked completions from both the smallest (left) and largest (right) models. Contexts are 768 tokens, with approximately 256 tokens worth of paragraphs shown. Completions are 256 tokens and fully shown. Top-k random sampling with $k = 40$ was used for generation.

Context (WebText test)	
Some of the most glorious historical attractions in Spain date from the period of Muslim rule, including The Mezquita, built as the Great Mosque of Córdoba and the Medina Azahara, also in Córdoba and now in ruins but still visitable as such and built as the Madinat al-Zahra, the Palace of al-Andalus; and the Alhambra in Granada, a splendid, intact palace. There are also two synagogues still standing that were built during the era of Muslim Spain: Santa María la Blanca in Toledo and the Synagogue of Córdoba, in the Old City. Reconquista and Imperial era [edit]	
Small model completion	**GPT-2 completion**
The era of Muslim Spain is very different from that of other parts of the world, with a rather extensive medieval history. While the Spanish Empire existed during the second half of the 9th century BC it was quite slowly transformed to a small, secularized empire by 1150 A.D., in a much more prosperous and diverse state. In 1050 CE the emperor of Naples, at least partially in characterised by a series of wars and invasions from the north-western coast to the western coast of the region (e.g., on the Danube River which extended all the way from Gibraltar into the south of Spain, but was quickly destroyed on the same day by the Roman army), assumed the title of Emperor of Naples. He made a large contribution over the last few decades to restore and maintain the empire, the first of which to the west. His successors were Ferdinand, who took power from the Papal Throne in 1148, while the first imperial successor was King Alfred II of Sweden. The empire was again made one of the few independent states in the European Continent and although no more than 200 million people lived on its territory during this period the people had much to say to the Emperor. From 1720 on the Muslims made up half the population of Spain.	The Reconquista (literally, ”The return to the country”) of 1092 and 1095 saw the establishment in the Iberian peninsula of a united kingdom of Castile and Aragon in which a Muslim dynasty would exist. This was the last time in Europe where Muslim rule would be considered ”native” and the last occasion where Christian-inflamed and organized conflicts would make themselves known in Europe. The reign of Muslim Moors in Spain is often referred to, especially among non-Muslims, as the ”Conquest of Granada” and Spain is sometimes referred to as Spain in ”Granada”. The last years of Muslim rule were followed by the Spanish Reconquista where the Muslims were driven or compelled out of the Iberian Peninsula with the victory of Ferdinand IX. in the Battle of Poitiers in March 1189 and the conquest of Jerusalem in 1187 by Saladin. The Reconquista period in Spain came to an end when the Moors retreated north. However the Moorish rule would not be fully replaced by the Christian rule until the Reconquista era resumed when the Reconquista lasted from 1229 to 1492. The last Muslim state in Spain, Granada, was conquered by the

Table 8. Random unseen contexts (top), and non-cherry-picked completions from both the smallest (left) and largest (right) models. Contexts are 768 tokens, with approximately 256 tokens worth of paragraphs shown. Completions are 256 tokens and fully shown. Top-k random sampling with $k = 40$ was used for generation.

Context (WebText test)

Pinky

The pink ghost's AI is designed to "feel" opposite of the red ghost's behavior. Pinky actually attempts to get out in front of Pac-Man. This is accomplished by setting the target 4 tiles ahead of Pac-Man's current location in the direction that Pac-Man is travelling. One exception to this is when Pac-Man is traveling up. Due to an overflow bug in the code, the calculation includes a left offset equal to the expected up offset.

The use of Pac-Man's expected position allows the player to exploit the navigation code for Pinky. This was most often seen in situations where Pac-Man and Pinky are in close proximity to each other. For example, if they were headed toward one another, once they are less than 4 tiles apart, the target tile is now behind Pinky. If he encounters an intersection while in this configuration, he will attempt to double-back (by turning) in order to reach the target tile.

Player's would often perform what was referred to as a "head fake" by changing directions rapidly just as Pinky was approaching an intersection. By displacing the target tile just as Pinky

Small model completion	**GPT-2 completion**
was coming to the left, they would get the "correct" direction. If the location is actually farther from what Pac-Man was at, the player's will not respond to the ghost's movement, and the ghost will instead return to the player's current location. This is because Pac-Man does not move by turning. The ghost will have to go to another location of some kind to return where he began, and this is achieved in many ways. It will often simply leave the location, as for example if Pac-Man did jump off a cliff, he would return to the spot where he was before vanishing into space. There is a good number of possibilities these ghost paths may be possible, but the most obvious ones are if the player wants to avoid getting caught up in the game, with a goal as easy as keeping Pac-Man in the game for 1/3 as long as he was. Chasing In Pac-Man's case, this would be similar to how Pac-Man can "catch up" in a fight. A ghost could chase through enemies. This is useful for many reasons, both for Pac-Man and his AI. Unlike most Ghost games, this ghost simply travels in the direction from	was about to cross the intersection, Pac-Man would be able to dodge the ghosts' projectiles and return to the safe location of the safe house. Buster The yellow ghost's AI is specifically designed to try to avoid Pac-Man from leaving the safe house. The player must press the SPACE bar to reveal the safe house's doors, then go into them to find the yellow ghost hiding in the next room. The player must also note that the ghost will not attempt to exit the safe house if it sees Pac-Man already there. In fact, the ghost will not attempt an exit in the first place unless they're already in the safe house. These two behaviors are based on the fact that the player does not always attempt to reach the safe house before Pac-Man (or Blinky). The player must also beware of this glitch if a ghost (or another player) is already in the safe house and attempting to get out as soon as Pac-Man or Blinky is gone. At this point, the ghost will attempt to exit the safe house in a similar manner as it is in "Chase" mode. Pac-Man There are some ghosts that follow Pac-Man. These ghosts are

Table 9. Random unseen contexts (top), and non-cherry-picked completions from both the smallest (left) and largest (right) models. Contexts are 768 tokens, with approximately 256 tokens worth of paragraphs shown. Completions are 256 tokens and fully shown. Top-k random sampling with $k = 40$ was used for generation.

Context (WebText test)	
Corporal Michael P. Goeldin was an unskilled laborer from Ireland when he enlisted in Company A in November 1860. Goldein survived the war. Corporal Patrick O'Neal, also from Ireland, first enlisted in 1854 and served with Company L, 3d U.S. Artillery, in Oregon. He returned to the East Coast and enlisted in the company in 1860. O'Neal served until 1874, when he was named superintendent of the National Cemetery at Willets Point, New York. Corporal Benjamin Browne was a shoemaker from Orange County, New York. In August 1862, he enlisted in the newly formed 124th New York Volunteers, and was one of sixty-one men who transferred into Company A that October. Browne reenlisted in the company in February 1864 while it was camped at Brandy Station. He returned to civilian life after completing his enlistment in 1867. On 10 June, Artificer William Collins was promoted to corporal, probably to fill a combat leadership void for the crossing of the James River. Collins's service record does not reflect the qualities he demonstrated to earn this promotion, but he had obviously overcome some serious problems. Born in Sacketts Harbor, New York, Collins enlisted in the company in December 1853 at the age of twenty-two, and reenlisted in December 1858. Just a month before the war began in April 1861, Collins went "over the hill" and was not caught until three years later. Returned to the company on 22 March 1864, he was tried	
Small model completion	**GPT-2 completion**
on military charges and was discharged from the U.S. Military. He returned to the company, and was assigned to Division of Internal Affairs during the Civil War. As an engineer, Collins also had training in engineering from his native New York. After graduate studies in the Union College of Engineering, he was employed in the United States Army for two years at the same time. There he worked under Captain Daniel Pfeiffer and was promoted to colonel in May 1870, three years after Captain Pfeiffer's enlistment. Collins returned home to England after completing his first year in the First Division to serve the next two years in the Army Corps of Engineers, and by his second year, he began to experience his service duty.¡—endoftext—¿MOSCOW, July 5 (Reuters) - U.K.-based Russian state media and government sources and experts say Moscow has sought to downplay the "military buildup" of the Syrian opposition while maintaining support for President Donald Trump's plan to build a wall along the U.S.-Mexico border, calling it a "military coup". U.S.-led Syria war veteran and Syria envoy Michael Flynn (R), right, gestures with U.S.-based Syrian political analyst Ahmet Koryat, with	in June of that year. Sentenced to be shot for desertion, he remained on active duty. Captain James J. Dall was born in Virginia in 1829 during the Revolution and joined his father in the Army in 1836. He served with the 2d Virginia Infantry until 1845, when he went to the Western Reserve Regiment; and, on his discharge, he reenlisted with Company I, 25th New York Volunteers (which in 1845 numbered about 4,500 men). One of his brothers, James, served with his company in the same capacity in the same brigade. While the regiment remained in Connecticut through 1846, Captain Dall served with Company I at various times before his discharge. A veteran of the Mexican War, he had served two years on the staff of Brigadier General John J. Sullivan at San Antonio. During the Peninsula Campaign of 1863, he commanded a company of twenty-three and was in charge of the battery of fifty-four heavy mortars and the gunboat Pinta of the Battery, a gunboat and several other boats. Captain Dall was on active duty with Company I in the fall of 1865. Two years later, he was promoted to second lieutenant. Captain John A. O'

Table 10. Random unseen contexts (top), and non-cherry-picked completions from both the smallest (left) and largest (right) models. Contexts are 768 tokens, with approximately 256 tokens worth of paragraphs shown. Completions are 256 tokens and fully shown. Top-k random sampling with $k = 40$ was used for generation.

Context (WebText test)

Xmonad, Xresources, and Fonts

Xresources can be an absolute pain (they were for me). So, this a bit of help:

First of all, the way that fonts are generally handled in Xmonad is through XFT . Xft is the X FreeType Interface library and is a library that handles the actual rendering of fonts in Xmonad/urxvt (Xresources specifies a bunch of resources to things that launch under the X window manager. Settings for individual programs are specified in Xresources like so: Urxvt.background:background settins here).

Anyway, fonts are set using this syntax in Xresources :

urxvt*font:xft:your font name goes here:size=the size of the font goes here,xft:fall-back font 1 goes here

So, thats the most basic part of fonts in urxvt specified via Xresources.

Fixing /.ssh/ Permissions in Cygwin

In Cygwin on Windows I found that I could not set the permissions of my /.ssh/ folder to be 0600, as is required for ssh to allow you to use keys. The symptom I had was that I no matter what I did, it always modfied the owner and the group

Small model completion

was set to /.ssh/ , which was the way things were. This could not be fixed completely. I found I could make that much more workable with just typing the username with your mouse then trying to set permissions to be ”0600” for the group, which is what my system is now configured to do.

However, as of Cygwin 7 on Linux I have found that doing so is now ”unnecessary” for my system! I also installed a package called chroot. It does that by allowing you to install/upgrade to your system directory.

So, as of Windows 8 and up, installing that can require chroot, so I tried and tried and found that it wasn’t so bad when installing the package on Linux and then uninstalling it. As of my last write-up on Cygwin, the script just does its job (the sudo commands do the rest), if you try it you will have to log the change.

If that wasn’t confusing enough, this is what I tried and just went back to my /.ssh/ directory and the chroot script. I found I was finally running out of permissions (because i forgot to create a symlink to ssh from the file system and

GPT-2 completion

, making these files read-only. I tried just starting ssh and using a program like passwd which modifies the owner/group to 0644 permissions, but it didn’t work. It was annoying but not nearly as bad as it could be, just annoying to have to fiddle with permissions.

So I decided to do this as the reverse:

For each .ssh folder in your Cygwin directory, set the permission of each file and directory to 2644. Then start ssh with: ssh-add /.ssh/

The first time you connect, your permissions of the /.ssh folder that you just had in Cygwin will be set to 0600. Then every time you connect again, your permissions will change to 0644.

Makefile for Xresources as used in /.xsessionrc

These are made in my .xsessionrc :

.xprofile: .xsession-errors: color=””

For example:

.xprofile: color=blue

As the colors are stored in a separate file, the color in your /.xsession-errors will look like cyan instead of blue.

This is mostly because, as of now, only the color is assigned to your

Table 11. Random unseen contexts (top), and non-cherry-picked completions from both the smallest (left) and largest (right) models. Contexts are 768 tokens, with approximately 256 tokens worth of paragraphs shown. Completions are 256 tokens and fully shown. Top-k random sampling with $k = 40$ was used for generation.

Context (WebText test)		
Aaron loves mint chocolate cake, but he requires that it be paired with mini chocolate chips, so I threw some of those in between the layers. I also had a few Peppermint Jo Jos on hand so I crushed them up and threw some of those in along with some crushed meringue cookies because, why not? It's a total smorgasbord of minty chocolate chippy cookie crunchy goodness. I didn't measure how much of each topping I used, but after I tasted the finished product, I wish I had added more. You can add anything you want- crushed candy canes, peppermint bark, etc. And don't be afraid to use a heavy hand. Texture = good. If you don't have 7-inch cake pans, you can get 3 shorter 8-inch layers out of this		
GPT-2 Completion 1	**GPT-2 Completion 2**	**GPT-2 Completion 3**
recipe. I made the longer version, then used the pan for a layer of buttercream frosting and stuck 1/3 of it into the fridge for a frosting topper. Just make sure you have all of this frosting prepared earlier than you make this cake and you won't have to wait on it as your egg whites sit on the bottom of your mixing bowl. You want the egg whites whipped to stiff peaks? Then put your cake dish over a pot of simmering water and heat it up until it's just barely warm. It doesn't take more than 3-5 minutes; just enough to get it started	recipe. If you do, and you know how to make that cake, then your life will never be the same. I have two very simple rules when it comes to cookie crumbs or chocolate chips. 1) When I buy raw sugar, I do not use it until it's cooled down and has at least half a teaspoon of sugar on it. That's my rule. And 2) If I have a recipe that calls for something and I want to use it in a recipe, I do not substitute it with something else on-line. Like this. I found this really cute card with cute little kittens on	recipe, which will likely yield about the same amount of cake. Chocolate Buttermilk Cake Print Chunks of chocolate cake, coated in buttermilk and topped with crunchy chopped chocolate, drizzled with chocolate sauce. Simple but a nice change from chocolate cake! Author: Momma Made Ingredients For the chocolate buttermilk cake 1 1/2 cup unsalted butter , room temperature 1 1/2 cup granulated sugar 3 large eggs plus 1 egg yolk For the chocolate glaze 1/3 cup cocoa powder 1 3/4 cups powdered sugar 6 ounces semis
GPT-2 Completion 4	**GPT-2 Completion 5**	**GPT-2 Completion 6**
recipe. Meringue Cookies 1 3/4 cup butter, softened 1 cup sugar 1 egg yolk 3 T. heavy cream 3 1/2-4 cups flour pinch of salt Peppermint-Jojo Topping 1 1/4 cups powdered sugar 1 cup chopped pecans 1/2 cup finely chopped mint leaves 1/2 cup chopped fresh mint (about 1/2 sheet) 1 tsp. peppermint extract 1/4 tsp. sea salt pinch of sea	recipe. My 2 failures at Christmas morning were not due to poor frosting skills. That was another success story. The cake was beautifully moist, and well, just a beautiful cake. It's got very little air in it or moisture, but all the chocolate and mint was perfectly blended together. One thing that made me cringe a bit was that all those ingredients- egg yolks, sugar, melted butter, chocolate chips, melted chocolate, and mint- had to be mixed right into the cake batter, which is super time-consuming. A friend suggested that if I put the batter in a separate bowl, I	recipe, which will give you about 6-9 servings of delicious chocolate cake. To prevent your layers from sticking together, I used my 8-inch round tart pan, lined it with parchment paper, then cut up 4 7-inch round cakes from it and sandwiched the layers between them (this also worked well for the 1-layer chocolate cake as well). When you're ready to bake this cake, remove the parchment paper, preheat oven to 350 degrees and bake for about 30 – 40 minutes, until the top is golden and a tester comes out clean. Chocolate Buttermilk Layer Cake with Peppermint Jo

Table 12. Non-cherry-picked completions from GPT-2 generated from the same context (from WebText test). Context is 384 tokens (shown truncated), and generations are 128 tokens. Top-k random sampling with $k = 40$ was used for generation.

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

Table 13. Conditional generation on an out-of-distribution context by GPT-2. Cherry pick of 10 samples generated with $k = 40$.

<table>
<tr><td colspan="2">Article: Prehistoric man sketched an incredible array of prehistoric beasts on the rough limestone walls of a cave in modern day France 36,000 years ago.
Now, with the help of cutting-edge technology, those works of art in the Chauvet-Pont-d'Arc Cave have been reproduced to create the biggest replica cave in the world.
The manmade cavern named the Caverne du Pont-d'Arc has been built a few miles from the original site in Vallon-Pont-D'arc in Southern France and contains 1,000 painstakingly-reproduced drawings as well as around 450 bones and other features...
Cavemen and women sketched an incredible array of prehistoric beasts on the rough limestone walls of a cave 36,000 years ago and now a replica has been created (pictured)
...</td></tr>
<tr><td>GPT-2: The original site in Vallon-Pont-D'arc in Southern France is a Unesco World Heritage site and is the oldest known and the best preserved cave decorated by man. The replica cave was built a few miles from the original site in Vallon-Pont-D'Arc in Southern France. The cave contains images of 14 different species of animals including woolly rhinoceros, mammoths, and big cats.</td><td>Reference: Cave mimics famous Caverne du Pont-d'Arc in France, the oldest cave decorated by man and the best preserved. The replica contains all 1,000 paintings which include 425 such as a woolly rhinoceros and mammoths. Minute details were copied using 3D modelling and anamorphic techniques, often used to shoot widescreen images. The modern cave also includes replica paw prints of bears, bones and details preserved in the original cave.</td></tr>
<tr><td colspan="2">Article: 'If you can find my submarine, it's yours,' Russian oil billionaire Roman Abramovich once said.
And it seems the oligarch and Chelsea FC owner, whose submarine is just one of the extras that came with his £300million superyacht Eclipse (perfect for getting to shore undetected), is not the only wealthy businessman splashing out on underwater exploration.
Dubbed Earth's real 'final frontier', the oceans are still so little-explored that billionaires are queuing up to buy vessels that give them a glimpse of the dramatic seascapes and incredible wildlife of the world's oceans.
So if you have a spare few million in the bank and want some holiday snaps and Instagram posts that will really trump everyone else, you may want to snap up one of these...
Whale of a time: The OrcaSub takes you 2000 feet beneath the surface in two pressurised Perspex viewing domes for optimum exploration
...</td></tr>
<tr><td>GPT-2: The U-Boat Worx submarine can dive to 984 feet (300 metres) and is capable of diving to the bottom of the ocean. It can be used for diving, snorkelling and even snorkel diving. The submarine is also equipped with a snorkelling platform.</td><td>Reference: Feel like a James Bond villain with the most advanced submerged ocean vehicles. Some convert from a yacht into a submarine, and others detached off for a speedy exploration. Oliver's Travels offers a Mile Low package on its submarine, complete with chef and butler for £175,000 a night.</td></tr>
<tr><td colspan="2">Article: Amina Ali Qassim is sitting with her youngest grandchild on her lap, wiping away tears with her headscarf. Only a few months old, this is the baby girl whose ears she desperately tried to cover the night the aerial bombardment started. She lay awake, she says, in a village mosque on the Yemeni island of Birim, counting explosions as the baby cried.
It could have been worse though. They could have still been in their house when the first missile landed.
"Our neighbor shouted to my husband 'you have to leave, they're coming.' And we just ran. As soon as we left the house, the first missile fell right by it and then a second on it. It burned everything to the ground," Qassim tells us
...</td></tr>
<tr><td>GPT-2: Yemen is in the middle of a civil war. Saudi Arabia is leading the coalition bombing campaign. It's been bombing Yemen for more than two months now.</td><td>Reference: Amina Ali Qassim's family sought shelter in a mosque before fleeing Yemen. Thousands like them are boarding boats to sail to Djibouti. Saudi Arabia has been pounding Yemen in a bid to defeat Houthi rebels.</td></tr>
</table>

Table 14. Summaries generated by GPT-2 and reference summaries on the test set of the CNN and Daily Mail dataset.

English reference This re-release, titled The Next Day Extra, was presented in the form of three disks: the original album, unpublished studio sessions and remixes, plus a DVD containing the four clips that have already been unveiled.	**GPT-2 French translation** Les nouvelles re-releases, tout en premier disc, nécessaire de l'album, un studio session et remixes, plus une DVD de l'écran de quelques clips qui ont été déjà échappés.
French reference Un homme expliquait que le fonctionnement de la hernia fonctionnelle qu'il avait reconnaît avant de faire, le fonctionnement de la hernia fonctionnelle que j'ai réussi, j'ai réussi.	**GPT-2 English translation** This release, originally released on three discs: the original album, sessions studio inédites and remixes, a DVD containing the four clips that were released on the previous day.
English reference One man explained that the free hernia surgery he'd received will allow him to work again.	**GPT-2 French translation** Un homme expliquait que le fonctionnement de la hernia fonctionnelle qu'il avait reconnaît avant de faire, le fonctionnement de la hernia fonctionnelle que j'ai réussi, j'ai réussi.
French reference Un homme a expliqué que l'opération gratuite qu'il avait subie pour soigner une hernie lui permettrait de travailler à nouveau.	**GPT-2 English translation** A man told me that the operation gratuity he had been promised would not allow him to travel.
English reference Speaking to an open government conference in London via video link, Mr Kerry said: "There is no question that the President and I and others in government have actually learned of some things that had been happening on an automatic pilot because the ability has been there, going back to World War Two and to the very difficult years of the Cold War, and then, of course, 9/11."	**GPT-2 French translation** Le premier conference de l'édition internationale en ligne sur la vide, le ministre de l'Édition internationale a été vide, lui dit : "Il n'y a qu'une question que le président et moi et tous les autres personnes d'une administration qui a été évident par la fois, par la part de la guerre, par la part de la guer
French reference S'exprimant lors d'une conférence intergouvernementale à Londres par liaison vidéo, M. Kerry a déclaré: "Il est indéniable que le Président, moi-même et d'autres membres du gouvernement avons pris connaissance de certaines choses en mode pilote automatique parce que nous en avions la possibilité, dès la Seconde guerre mondiale et jusqu'aux années difficiles de la Guerre froide, puis bien sûr le 11 septembre."	**GPT-2 English translation** In a conférence between the United States and London, Secretary of State John Kerry said: "It is indeniable that the President, myself and others of the government have been aware of certain certain choices that have been made in the past in order to be able to do certain things in a more automated way."

Table 15. English to French and French to English translations generated by GPT-2.

Context (passage and previous question/answer pairs)

The 2008 Summer Olympics torch relay was run from March 24 until August 8, 2008, prior to the 2008 Summer Olympics, with the theme of "one world, one dream". Plans for the relay were announced on April 26, 2007, in Beijing, China. The relay, also called by the organizers as the "Journey of Harmony", lasted 129 days and carried the torch 137,000 km (85,000 mi) – the longest distance of any Olympic torch relay since the tradition was started ahead of the 1936 Summer Olympics.

After being lit at the birthplace of the Olympic Games in Olympia, Greece on March 24, the torch traveled to the Panathinaiko Stadium in Athens, and then to Beijing, arriving on March 31. From Beijing, the torch was following a route passing through six continents. The torch has visited cities along the Silk Road, symbolizing ancient links between China and the rest of the world. The relay also included an ascent with the flame to the top of Mount Everest on the border of Nepal and Tibet, China from the Chinese side, which was closed specially for the event.

Q: What was the theme
A: "one world, one dream".

Q: What was the length of the race?
A: 137,000 km

Q: Was it larger than previous ones?
A: No

Q: Where did the race begin?
A: Olympia, Greece

Q: Is there anything notable about that place?
A: birthplace of Olympic Games

Q: Where did they go after?
A: Athens

Q: How many days was the race?
A: seven

Q: Did they visit any notable landmarks?
A: Panathinaiko Stadium

Q: And did they climb any mountains?
A:

Model answer: Everest
Turker answers: unknown, yes, Yes, yes

Table 16. Selected CoQA completion.

Context (passage and previous question/answer pairs)

Tom goes everywhere with Catherine Green, a 54-year-old secretary. He moves around her office at work and goes shopping with her. "Most people don't seem to mind Tom," says Catherine, who thinks he is wonderful. "He's my fourth child," she says. She may think of him and treat him that way as her son. He moves around buying his food, paying his health bills and his taxes, but in fact Tom is a dog.

Catherine and Tom live in Sweden, a country where everyone is expected to lead an orderly life according to rules laid down by the government, which also provides a high level of care for its people. This level of care costs money.

People in Sweden pay taxes on everything, so aren't surprised to find that owning a dog means more taxes. Some people are paying as much as 500 Swedish kronor in taxes a year for the right to keep their dog, which is spent by the government on dog hospitals and sometimes medical treatment for a dog that falls ill. However, most such treatment is expensive, so owners often decide to offer health and even life _ for their dog.

In Sweden dog owners must pay for any damage their dog does. A Swedish Kennel Club official explains what this means: if your dog runs out on the road and gets hit by a passing car, you, as the owner, have to pay for any damage done to the car, even if your dog has been killed in the accident.

Q: How old is Catherine?
A: 54

Q: where does she live?
A:

Model answer: Stockholm
Turker answers: Sweden, Sweden, in Sweden, Sweden

Table 17. Selected CoQA completion.

Generative Adversarial Nets

Ian J. Goodfellow[*], **Jean Pouget-Abadie**[†], **Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair**[‡], **Aaron Courville, Yoshua Bengio**[§]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

1 Introduction

The promise of deep learning is to discover rich, hierarchical models [2] that represent probability distributions over the kinds of data encountered in artificial intelligence applications, such as natural images, audio waveforms containing speech, and symbols in natural language corpora. So far, the most striking successes in deep learning have involved discriminative models, usually those that map a high-dimensional, rich sensory input to a class label [14, 20]. These striking successes have primarily been based on the backpropagation and dropout algorithms, using piecewise linear units [17, 8, 9] which have a particularly well-behaved gradient . Deep *generative* models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to difficulty of leveraging the benefits of piecewise linear units in the generative context. We propose a new generative model estimation procedure that sidesteps these difficulties. [1]

In the proposed *adversarial nets* framework, the generative model is pitted against an adversary: a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution. The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistiguishable from the genuine articles.

[*] Ian Goodfellow is now a research scientist at Google, but did this work earlier as a UdeM student

[†] Jean Pouget-Abadie did this work while visiting Université de Montréal from Ecole Polytechnique.

[‡] Sherjil Ozair is visiting Université de Montréal from Indian Institute of Technology Delhi

[§] Yoshua Bengio is a CIFAR Senior Fellow.

[1] All code and hyperparameters available at `http://www.github.com/goodfeli/adversarial`

This framework can yield specific training algorithms for many kinds of model and optimization algorithm. In this article, we explore the special case when the generative model generates samples by passing random noise through a multilayer perceptron, and the discriminative model is also a multilayer perceptron. We refer to this special case as *adversarial nets*. In this case, we can train both models using only the highly successful backpropagation and dropout algorithms [16] and sample from the generative model using only forward propagation. No approximate inference or Markov chains are necessary.

2 Related work

Until recently, most work on deep generative models focused on models that provided a parametric specification of a probability distribution function. The model can then be trained by maximizing the log likelihood. In this family of model, perhaps the most succesful is the deep Boltzmann machine [25]. Such models generally have intractable likelihood functions and therefore require numerous approximations to the likelihood gradient. These difficulties motivated the development of "generative machines"–models that do not explicitly represent the likelihood, yet are able to generate samples from the desired distribution. Generative stochastic networks [4] are an example of a generative machine that can be trained with exact backpropagation rather than the numerous approximations required for Boltzmann machines. This work extends the idea of a generative machine by eliminating the Markov chains used in generative stochastic networks.

Our work backpropagates derivatives through generative processes by using the observation that

$$\lim_{\sigma \to 0} \nabla_{\boldsymbol{x}} \mathbb{E}_{\epsilon \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})} f(\boldsymbol{x} + \epsilon) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}).$$

We were unaware at the time we developed this work that Kingma and Welling [18] and Rezende *et al.* [23] had developed more general stochastic backpropagation rules, allowing one to backpropagate through Gaussian distributions with finite variance, and to backpropagate to the covariance parameter as well as the mean. These backpropagation rules could allow one to learn the conditional variance of the generator, which we treated as a hyperparameter in this work. Kingma and Welling [18] and Rezende *et al.* [23] use stochastic backpropagation to train variational autoencoders (VAEs). Like generative adversarial networks, variational autoencoders pair a differentiable generator network with a second neural network. Unlike generative adversarial networks, the second network in a VAE is a recognition model that performs approximate inference. GANs require differentiation through the visible units, and thus cannot model discrete data, while VAEs require differentiation through the hidden units, and thus cannot have discrete latent variables. Other VAE-like approaches exist [12, 22] but are less closely related to our method.

Previous work has also taken the approach of using a discriminative criterion to train a generative model [29, 13]. These approaches use criteria that are intractable for deep generative models. These methods are difficult even to approximate for deep models because they involve ratios of probabilities which cannot be approximated using variational approximations that lower bound the probability. Noise-contrastive estimation (NCE) [13] involves training a generative model by learning the weights that make the model useful for discriminating data from a fixed noise distribution. Using a previously trained model as the noise distribution allows training a sequence of models of increasing quality. This can be seen as an informal competition mechanism similar in spirit to the formal competition used in the adversarial networks game. The key limitation of NCE is that its "discriminator" is defined by the ratio of the probability densities of the noise distribution and the model distribution, and thus requires the ability to evaluate and backpropagate through both densities.

Some previous work has used the general concept of having two neural networks compete. The most relevant work is predictability minimization [26]. In predictability minimization, each hidden unit in a neural network is trained to be different from the output of a second network, which predicts the value of that hidden unit given the value of all of the other hidden units. This work differs from predictability minimization in three important ways: 1) in this work, the competition between the networks is the sole training criterion, and is sufficient on its own to train the network. Predictability minimization is only a regularizer that encourages the hidden units of a neural network to be statistically independent while they accomplish some other task; it is not a primary training criterion. 2) The nature of the competition is different. In predictability minimization, two networks' outputs are compared, with one network trying to make the outputs similar and the other trying to make the

outputs different. The output in question is a single scalar. In GANs, one network produces a rich, high dimensional vector that is used as the input to another network, and attempts to choose an input that the other network does not know how to process. 3) The specification of the learning process is different. Predictability minimization is described as an optimization problem with an objective function to be minimized, and learning approaches the minimum of the objective function. GANs are based on a minimax game rather than an optimization problem, and have a value function that one agent seeks to maximize and the other seeks to minimize. The game terminates at a saddle point that is a minimum with respect to one player's strategy and a maximum with respect to the other player's strategy.

Generative adversarial networks has been sometimes confused with the related concept of "adversarial examples" [28]. Adversarial examples are examples found by using gradient-based optimization directly on the input to a classification network, in order to find examples that are similar to the data yet misclassified. This is different from the present work because adversarial examples are not a mechanism for training a generative model. Instead, adversarial examples are primarily an analysis tool for showing that neural networks behave in intriguing ways, often confidently classifying two images differently with high confidence even though the difference between them is imperceptible to a human observer. The existence of such adversarial examples does suggest that generative adversarial network training could be inefficient, because they show that it is possible to make modern discriminative networks confidently recognize a class without emulating any of the human-perceptible attributes of that class.

3 Adversarial nets

The adversarial modeling framework is most straightforward to apply when the models are both multilayer perceptrons. To learn the generator's distribution p_g over data $\boldsymbol{x}$, we define a prior on input noise variables $p_{\boldsymbol{z}}(\boldsymbol{z})$, then represent a mapping to data space as $G(\boldsymbol{z}; \theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. We also define a second multilayer perceptron $D(\boldsymbol{x}; \theta_d)$ that outputs a single scalar. $D(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $\log(1 - D(G(\boldsymbol{z})))$. In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \tag{1}$$

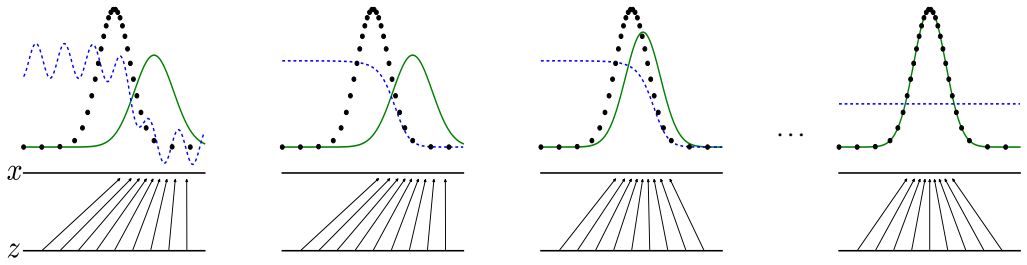
In the next section, we present a theoretical analysis of adversarial nets, essentially showing that the training criterion allows one to recover the data generating distribution as G and D are given enough capacity, i.e., in the non-parametric limit. See Figure 1 for a less formal, more pedagogical explanation of the approach. In practice, we must implement the game using an iterative, numerical approach. Optimizing D to completion in the inner loop of training is computationally prohibitive, and on finite datasets would result in overfitting. Instead, we alternate between k steps of optimizing D and one step of optimizing G. This results in D being maintained near its optimal solution, so long as G changes slowly enough. The procedure is formally presented in Algorithm 1.

In practice, equation 1 may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1 - D(G(\boldsymbol{z})))$ saturates. Rather than training G to minimize $\log(1 - D(G(\boldsymbol{z})))$ we can train G to maximize $\log D(G(\boldsymbol{z}))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.

4 Theoretical Results

The generator G implicitly defines a probability distribution p_g as the distribution of the samples $G(\boldsymbol{z})$ obtained when $\boldsymbol{z} \sim p_{\boldsymbol{z}}$. Therefore, we would like Algorithm 1 to converge to a good estimator of p_{data}, if given enough capacity and training time. The results of this section are done in a non-parametric setting, e.g. we represent a model with infinite capacity by studying convergence in the space of probability density functions.

We will show in section 4.1 that this minimax game has a global optimum for $p_g = p_{\text{data}}$. We will then show in section 4.2 that Algorithm 1 optimizes Eq 1, thus obtaining the desired result.



(a) (b) (c) (d)

Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

4.1 Global Optimality of $p_g = p_{\text{data}}$

We first consider the optimal discriminator D for any given generator G.

Proposition 1. *For G fixed, the optimal discriminator D is*

$$D^*_G(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2}$$

Proof. The training criterion for the discriminator D, given any generator G, is to maximize the quantity $V(G, D)$

$$\begin{aligned} V(G,D) &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_{z} p_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(g(\boldsymbol{z})))dz \\ &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx \end{aligned} \tag{3}$$

For any $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$, the function $y \to a \log(y) + b \log(1 - y)$ achieves its maximum in $[0, 1]$ at $\frac{a}{a+b}$. The discriminator does not need to be defined outside of $Supp(p_{\text{data}}) \cup Supp(p_g)$, concluding the proof. □

Note that the training objective for D can be interpreted as maximizing the log-likelihood for estimating the conditional probability $P(Y = y|\boldsymbol{x})$, where Y indicates whether $\boldsymbol{x}$ comes from p_{data} (with $y = 1$) or from p_g (with $y = 0$). The minimax game in Eq. 1 can now be reformulated as:

$$\begin{aligned} C(G) =& \max_D V(G,D) \\ =& \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\ =& \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] \\ =& \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \end{aligned} \tag{4}$$

Theorem 1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.*

Proof. For $p_g = p_{\text{data}}$, $D_G^*(\boldsymbol{x}) = \frac{1}{2}$, (consider Eq. 2). Hence, by inspecting Eq. 4 at $D_G^*(\boldsymbol{x}) = \frac{1}{2}$, we find $C(G) = \log \frac{1}{2} + \log \frac{1}{2} = -\log 4$. To see that this is the best possible value of $C(G)$, reached only for $p_g = p_{\text{data}}$, observe that

$$\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[-\log 2] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[-\log 2] = -\log 4$$

and that by subtracting this expression from $C(G) = V(D_G^*, G)$, we obtain:

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right) \tag{5}$$

where KL is the Kullback–Leibler divergence. We recognize in the previous expression the Jensen–Shannon divergence between the model's distribution and the data generating process:

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right) \tag{6}$$

Since the Jensen–Shannon divergence between two distributions is always non-negative, and zero iff they are equal, we have shown that $C^* = -\log(4)$ is the global minimum of $C(G)$ and that the only solution is $p_g = p_{\text{data}}$, i.e., the generative model perfectly replicating the data distribution. □

4.2 Convergence of Algorithm 1

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x} \sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))]$$

then p_g converges to p_{data}

Proof. Consider $V(G, D) = U(p_g, D)$ as a function of p_g as done in the above criterion. Note that $U(p_g, D)$ is convex in p_g. The subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained. In other words, if $f(x) = \sup_{\alpha \in \mathcal{A}} f_\alpha(x)$ and $f_\alpha(x)$ is convex in x for every α, then $\partial f_\beta(x) \in \partial f$ if $\beta = \arg\sup_{\alpha \in \mathcal{A}} f_\alpha(x)$. This is equivalent to computing a gradient descent update for p_g at the optimal D given the corresponding G. $\sup_D U(p_g, D)$ is convex in p_g with a unique global optima as proven in Thm 1, therefore with sufficiently small updates of p_g, p_g converges to p_x, concluding the proof. □

In practice, adversarial nets represent a limited family of p_g distributions via the function $G(\boldsymbol{z}; \theta_g)$, and we optimize θ_g rather than p_g itself, so the proofs do not apply. However, the excellent performance of multilayer perceptrons in practice suggests that they are a reasonable model to use despite their lack of theoretical guarantees.

Model	MNIST	TFD
DBN [3]	138 ± 2	1909 ± 66
Stacked CAE [3]	121 ± 1.6	$\mathbf{2110 \pm 50}$
Deep GSN [5]	214 ± 1.1	1890 ± 29
Adversarial nets	$\mathbf{225 \pm 2}$	$\mathbf{2057 \pm 26}$

Table 1: Parzen window-based log-likelihood estimates. The reported numbers on MNIST are the mean log-likelihood of samples on test set, with the standard error of the mean computed across examples. On TFD, we computed the standard error across folds of the dataset, with a different σ chosen using the validation set of each fold. On TFD, σ was cross validated on each fold and mean log-likelihood on each fold were computed. For MNIST we compare against other models of the real-valued (rather than binary) version of dataset.

5 Experiments

We trained adversarial nets an a range of datasets including MNIST[21], the Toronto Face Database (TFD) [27], and CIFAR-10 [19]. The generator nets used a mixture of rectifier linear activations [17, 8] and sigmoid activations, while the discriminator net used maxout [9] activations. Dropout [16] was applied in training the discriminator net. While our theoretical framework permits the use of dropout and other noise at intermediate layers of the generator, we used noise as the input to only the bottommost layer of the generator network.

We estimate probability of the test set data under p_g by fitting a Gaussian Parzen window to the samples generated with G and reporting the log-likelihood under this distribution. The σ parameter of the Gaussians was obtained by cross validation on the validation set. This procedure was introduced in Breuleux *et al.* [7] and used for various generative models for which the exact likelihood is not tractable [24, 3, 4]. Results are reported in Table 1. This method of estimating the likelihood has somewhat high variance and does not perform well in high dimensional spaces but it is the best method available to our knowledge. Advances in generative models that can sample but not estimate likelihood directly motivate further research into how to evaluate such models. In Figures 2 and 3 we show samples drawn from the generator net after training. While we make no claim that these samples are better than samples generated by existing methods, we believe that these samples are at least competitive with the better generative models in the literature and highlight the potential of the adversarial framework.

6 Advantages and disadvantages

This new framework comes with advantages and disadvantages relative to previous modeling frameworks. The disadvantages are primarily that there is no explicit representation of $p_g(\boldsymbol{x})$, and that D must be synchronized well with G during training (in particular, G must not be trained too much without updating D, in order to avoid "the Helvetica scenario" in which G collapses too many values of $\mathbf{z}$ to the same value of $\mathbf{x}$ to have enough diversity to model p_{data}), much as the negative chains of a Boltzmann machine must be kept up to date between learning steps. The advantages are that Markov chains are never needed, only backprop is used to obtain gradients, no inference is needed during learning, and a wide variety of functions can be incorporated into the model. Table 2 summarizes the comparison of generative adversarial nets with other generative modeling approaches.

The aforementioned advantages are primarily computational. Adversarial models may also gain some statistical advantage from the generator network not being updated directly with data examples, but only with gradients flowing through the discriminator. This means that components of the input are not copied directly into the generator's parameters. Another advantage of adversarial networks is that they can represent very sharp, even degenerate distributions, while methods based on Markov chains require that the distribution be somewhat blurry in order for the chains to be able to mix between modes.

7 Conclusions and future work

This framework admits many straightforward extensions:

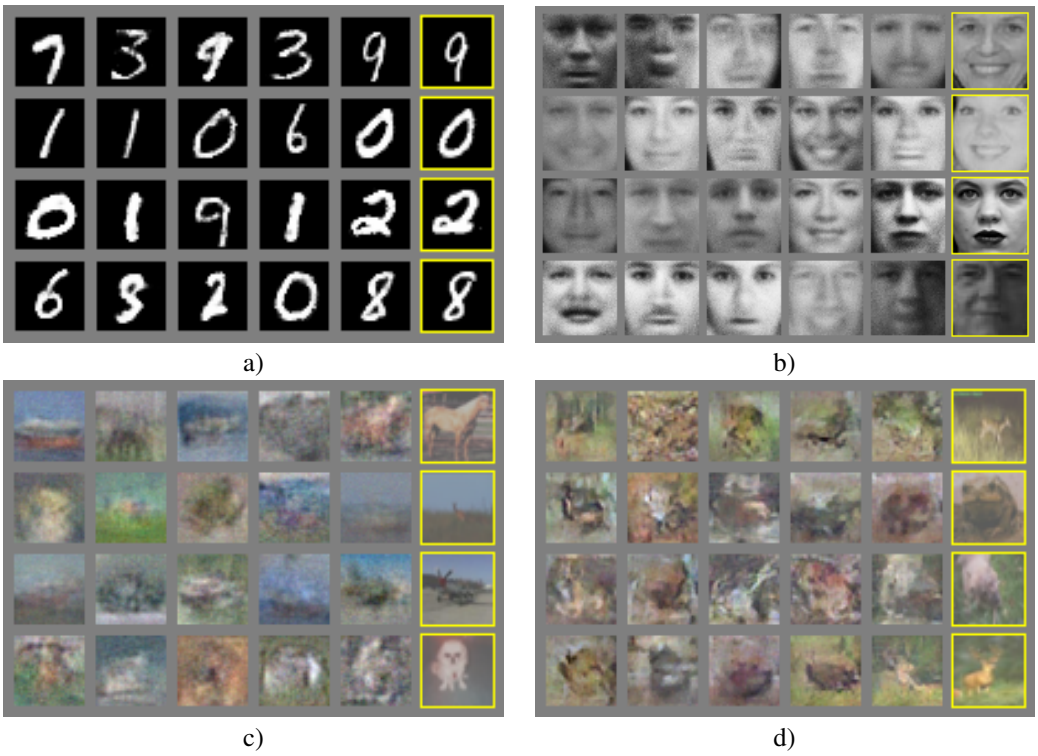


Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

Figure 3: Digits obtained by linearly interpolating between coordinates in $\boldsymbol{z}$ space of the full model.

1. A *conditional generative* model $p(\boldsymbol{x} \mid \boldsymbol{c})$ can be obtained by adding $\boldsymbol{c}$ as input to both G and D.
2. *Learned approximate inference* can be performed by training an auxiliary network to predict $\boldsymbol{z}$ given $\boldsymbol{x}$. This is similar to the inference net trained by the wake-sleep algorithm [15] but with the advantage that the inference net may be trained for a fixed generator net after the generator net has finished training.
3. One can approximately model all conditionals $p(\boldsymbol{x}_S \mid \boldsymbol{x}_{\not S})$ where S is a subset of the indices of $\boldsymbol{x}$ by training a family of conditional models that share parameters. Essentially, one can use adversarial nets to implement a stochastic extension of the deterministic MP-DBM [10].
4. *Semi-supervised learning*: features from the discriminator or inference net could improve performance of classifiers when limited labeled data is available.
5. *Efficiency improvements:* training could be accelerated greatly by devising better methods for coordinating G and D or determining better distributions to sample $\mathbf{z}$ from during training.

This paper has demonstrated the viability of the adversarial modeling framework, suggesting that these research directions could prove useful.

	Deep directed graphical models	Deep undirected graphical models	Generative autoencoders	Adversarial models
Training	Inference needed during training.	Inference needed during training. MCMC needed to approximate partition function gradient.	Enforced tradeoff between mixing and power of reconstruction generation	Synchronizing the discriminator with the generator. Helvetica.
Inference	Learned approximate inference	Variational inference	MCMC-based inference	Learned approximate inference
Sampling	No difficulties	Requires Markov chain	Requires Markov chain	No difficulties
Evaluating $p(x)$	Intractable, may be approximated with AIS	Intractable, may be approximated with AIS	Not explicitly represented, may be approximated with Parzen density estimation	Not explicitly represented, may be approximated with Parzen density estimation
Model design	Models need to be designed to work with the desired inference scheme — some inference schemes support similar model families as GANs	Careful design needed to ensure multiple properties	Any differentiable function is theoretically permitted	Any differentiable function is theoretically permitted

Table 2: Challenges in generative modeling: a summary of the difficulties encountered by different approaches to deep generative modeling for each of the major operations involving a model.

Acknowledgments

We would like to acknowledge Patrice Marcotte, Olivier Delalleau, Kyunghyun Cho, Guillaume Alain and Jason Yosinski for helpful discussions. Yann Dauphin shared his Parzen window evaluation code with us. We would like to thank the developers of Pylearn2 [11] and Theano [6, 1], particularly Frédéric Bastien who rushed a Theano feature specifically to benefit this project. Arnaud Bergeron provided much-needed support with LaTeX typesetting. We would also like to thank CIFAR, and Canada Research Chairs for funding, and Compute Canada, and Calcul Québec for providing computational resources. Ian Goodfellow is supported by the 2013 Google Fellowship in Deep Learning. Finally, we would like to thank Les Trois Brasseurs for stimulating our creativity.

References

[1] Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

[2] Bengio, Y. (2009). *Learning deep architectures for AI*. Now Publishers.

[3] Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2013). Better mixing via deep representations. In *ICML'13*.

[4] Bengio, Y., Thibodeau-Laufer, E., and Yosinski, J. (2014a). Deep generative stochastic networks trainable by backprop. In *ICML'14*.

[5] Bengio, Y., Thibodeau-Laufer, E., Alain, G., and Yosinski, J. (2014b). Deep generative stochastic networks trainable by backprop. In *Proceedings of the 30th International Conference on Machine Learning (ICML'14)*.

[6] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

[7] Breuleux, O., Bengio, Y., and Vincent, P. (2011). Quickly generating representative samples from an RBM-derived process. *Neural Computation*, **23**(8), 2053–2073.

[8] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *AISTATS'2011*.

[9] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013a). Maxout networks. In *ICML'2013*.

[10] Goodfellow, I. J., Mirza, M., Courville, A., and Bengio, Y. (2013b). Multi-prediction deep Boltzmann machines. In *NIPS'2013*.

[11] Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., and Bengio, Y. (2013c). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*.

[12] Gregor, K., Danihelka, I., Mnih, A., Blundell, C., and Wierstra, D. (2014). Deep autoregressive networks. In *ICML'2014*.

[13] Gutmann, M. and Hyvarinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10)*.

[14] Hinton, G., Deng, L., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012a). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, **29**(6), 82–97.

[15] Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, **268**, 1558–1161.

[16] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580.

[17] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pages 2146–2153. IEEE.

[18] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[19] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

[20] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *NIPS'2012*.

[21] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

[22] Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. Technical report, arXiv preprint arXiv:1402.0030.

[23] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. Technical report, arXiv:1401.4082.

[24] Rifai, S., Bengio, Y., Dauphin, Y., and Vincent, P. (2012). A generative process for sampling contractive auto-encoders. In *ICML'12*.

[25] Salakhutdinov, R. and Hinton, G. E. (2009). Deep Boltzmann machines. In *AISTATS'2009*, pages 448–455.

[26] Schmidhuber, J. (1992). Learning factorial codes by predictability minimization. *Neural Computation*, **4**(6), 863–879.

[27] Susskind, J., Anderson, A., and Hinton, G. E. (2010). The Toronto face dataset. Technical Report UTML TR 2010-001, U. Toronto.

[28] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2014). Intriguing properties of neural networks. *ICLR*, **abs/1312.6199**.

[29] Tu, Z. (2007). Learning generative models via discriminative approaches. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE.