Composed Fine-Tuning: Freezing Pre-Trained Denoising Autoencoders for Improved Generalization

Sang Michael Xie [1] Tengyu Ma [1] Percy Liang [1]

Abstract

We focus on prediction problems with structured outputs that are subject to output validity constraints, e.g. pseudocode-to-code translation where the code must compile. While labeled input-output pairs are expensive to obtain, "unlabeled" outputs, i.e. outputs without corresponding inputs, are freely available (e.g. code on GitHub) and provide information about output validity. Pre-training captures this structure by training a denoiser to denoise corrupted versions of unlabeled outputs. We first show that standard fine-tuning after pre-training destroys some of this structure. We then propose *composed fine-tuning*, which trains a predictor composed with the pre-trained denoiser. Importantly, the denoiser is fixed to preserve output structure. Like standard fine-tuning, the predictor is also initialized with the pre-trained denoiser. We prove for two-layer ReLU networks that composed fine-tuning significantly reduces the complexity of the predictor, thus improving generalization. Empirically, we show that composed fine-tuning improves over standard fine-tuning on two pseudocode-to-code translation datasets (3% and 6% relative). The improvement is magnified on out-of-distribution (OOD) examples (4% and 25% relative), suggesting that reducing predictor complexity improves OOD extrapolation.

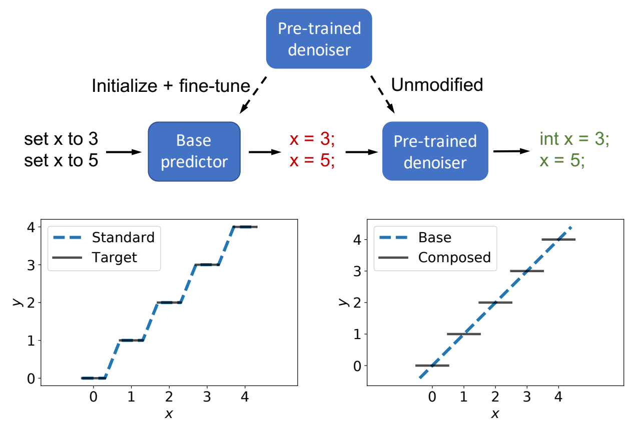


Figure 1: **(Top)** Composed fine-tuning: we fine-tune a base predictor composed with a pre-trained denoising autoencoder (denoiser) that is unmodified during fine-tuning. The base predictor is initialized from the same pre-trained denoiser. Composing with the fixed denoiser preserves the output structure information in the pre-trained denoiser. This offloads the complexity of capturing output structure to the denoiser, reducing the complexity of the base predictor. **(Bottom-Left)** Univariate regression example where a staircase function requires a complex linear spline fit. **(Bottom-Right)** A simple linear function can fit a staircase function when composed with the denoiser which projects onto the valid outputs (the integers).

1. Introduction

We study prediction problems whose outputs have validity constraints. For example, in pseudocode-to-code translation (Kulal et al., 2019), the output code must compile. Other examples include machine translation (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015) and molecule generation (Méndez-Lucio et al., 2020; Senior et al., 2020), where outputs should be grammatical or chemically valid, respectively. Expensively-obtained labeled data may not contain enough examples to learn a model that captures the complex output structure. In these problems, there are often lots of "unlabeled" outputs—outputs without a corresponding input (e.g., GitHub has over 40 million public code repositories as of February 2021 (Github, 2021)).

Pre-training with denoising autoencoders (Lewis et al., 2020; Raffel et al., 2019) leverages unlabeled outputs to learn output structure and transfer it to downstream tasks. They pre-train by synthetically corrupting unlabeled outputs and learning to denoise them, yielding a pre-trained denoiser that generates valid outputs. The pre-trained denoiser is then used to initialize and do standard fine-tuning on a smaller labeled dataset from a downstream task such as machine translation. This pre-training+fine-tuning paradigm is particularly attractive since practitioners (with less computational resources) do not need to access the large unlabeled dataset during fine-tuning.

[1]Department of Computer Science, Stanford University. Correspondence to: Sang Michael Xie <xie@cs.stanford.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

In this paper, we first show that standard fine-tuning after pre-training may not optimally leverage the pre-trained denoiser, complementing recent works on the suboptimality of fine-tuning in language models (Dodge et al., 2020; Lee et al., 2020; Zhang et al., 2021). To show this, we run a simple experiment that shows that standard fine-tuning destroys some of the output structure information in the pre-trained model. In particular, we show that re-applying the pre-trained denoiser post-hoc to the outputs of the standard fine-tuned model improves the validity and correctness of the output by 0.5–1.5% on a pseudocode-to-code dataset (see Section 6.1).

To address this, we propose *composed fine-tuning* (Figure 1 top), which fine-tunes a base predictor composed with a fixed pre-trained denoiser. Here, the denoiser enforces the output validity structure learned during pre-training. As in standard fine-tuning, the base predictor is initialized with the same pre-trained denoiser. The main difference with our simple experiment is that we fine-tune with the composed denoiser instead of applying the denoiser at test-time.

By factorizing into two modules, base predictor and denoiser, the framework also allows the base predictor to be simpler by offloading the complexity of modeling output structure to the denoiser. For intuition, Figure 1 (bottom-left/right) shows a pictorial example of a staircase function where valid outputs are integers and requires a complex spline to represent. When composing before a denoiser (which rounds to the nearest integer), a simple linear base predictor can represent the staircase function. Thus, composed fine-tuning reduces the complexity (and thus sample efficiency) of the base predictor.

We formalize this intuition theoretically, showing that beyond preserving the output structure information in the pre-trained denoiser, composed fine-tuning reduces the complexity of the base predictor needed to represent the target function. We show this complexity gap can be arbitrarily large for 2-layer ReLU networks representing a family of functions from $\mathbb{R} \to \mathbb{R}^k$, depending on the stability of the target function to be represented.

Empirically, we show on two pseudocode-to-code datasets that composed fine-tuning improves validity and correctness over standard fine-tuning, showing that composed fine-tuning preserves the pre-trained output structure better. We consider the more difficult full-program translation task rather than line-by-line translation (with compiler side information) studied by previous work (Kulal et al., 2019; Yasunaga & Liang, 2020). First we introduce SANSTYPE, a synthetic pseudocode-to-code dataset where the pseudocode specifies all but the variable types. On SANSTYPE, composed fine-tuning corrects invalid code at test time and also simplifies the base predictor by helping with global type inference. Secondly on SPOC (Kulal et al., 2019), a recent pseudocode-to-code dataset based on programming competition problems, we improve the proportion of correct programs by about 1% over standard fine-tuning and 3-4% over a baseline Transformer. We also show that composed fine-tuning gives additive gains on top of other semi-supervised methods, including backtranslation, which has access to a large unlabeled dataset for every downstream task. Composed fine-tuning obtains further gains over backtranslation by about 1.5% on SPOC.

Composed fine-tuning provides larger gains on out-of-distribution (OOD) examples, including on unseen pseudocode in SANSTYPE and unseen programs in SPOC. On SANSTYPE, the relative improvement over standard fine-tuning grows from 5% to 24% on the OOD split. On SPOC, the relative improvement increases from 3% to 4% on unseen programs. We also test on an image generation task where all test inputs are OOD. In particular, the inputs are font type and character type pairs, and the output is a valid font character image. At test time, the composed predictor produces font images of unseen font-character pairings with noticeably better clarity and styling than the baseline.

2. Setup

We consider a prediction problem from an input space $\mathcal{X}$ (e.g., pseudocode) to an output space $\mathcal{Y}$ (e.g., code) where there is an unknown subset of *valid* outputs $\mathcal{V} \subseteq \mathcal{Y}$ (e.g., code that compiles) and the true output is always valid (in $\mathcal{V}$). We are given a pre-trained *denoiser* $\Pi : \mathcal{Y} \to \mathcal{V}$, which "projects" a possibly invalid output in $\mathcal{Y}$ to the valid set $\mathcal{V}$. In practice, the denoiser is approximate and learned on synthetic corruptions of valid unlabeled outputs. The goal in the fine-tuning step is to learn a *predictor* $f : \mathcal{X} \to \mathcal{Y}$ using a small downstream labeled dataset $(x_1,y_1),...,(x_n,y_n)$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{V}$.

Base and composed predictors. Let $\Pi \circ f_{\text{base}}$ be a *composed predictor* that approximates the target function $f^\star$. In the context of $\Pi \circ f_{\text{base}}$, we call f_{base} the *base predictor*.

Standard fine-tuning. Standard fine-tuning parameterizes a probabilistic model p_α for the predictor and maximizes the log-likelihood on the labeled dataset $\mathbb{E}_{x,y}[\log p_\alpha(y \mid x)]$. In seq2seq tasks, both the input and output are sequences of tokens in the space of strings ($\mathcal{X} = \mathcal{Y}$). This allows standard fine-tuning to initialize α with the parameters of the pre-trained denoiser. We define the standard fine-tuned model to be $f_{\text{std}}(x) = \text{argmax}_y p_\alpha(y \mid x)$. We also compare to a more sophisticated fine-tuning method in Section 7.

3. Standard fine-tuning destroys some pre-trained output structure

In this section, we show that standard fine-tuning is suboptimal. To do so, we consider pseudocode-to-code translation, where the input space $\mathcal{X}$ is pseudocode and the set of valid outputs $\mathcal{V}$ is code that compiles and executes correctly.

We consider SANSTYPE, a synthetic pseudocode-to-code

```
set var_8 to "str_2";
instantiate var_2;
read var_2 from stdin;
add "str_4" to the beginning of var_8;
set var_2 to false;
set var_5 to true;
if var_2 is true, swap the values of var_2 and var_5;
if var_2 is true, set var_2 to the value of var_5 and var_5 to the value of var_2;
output var_8 to stdout;
print var_2;
output var_5 to stdout;
```

```
int main () {
  string var_8 = "str_2";
  bool var_2;
  cin >> var_2;
  var_8 = "str_4" + var_8;
  var_2 = false;
  var_5 = true;
  if ( var_2 ) {
    string temp = var_2;
    var_2 = var_5;
    var_5 = temp; }
  if ( var_2 ) {
    bool temp = var_2;
    var_2 = var_5;
    var_5 = temp; }
  cout << var_8;
  cout << var_2;
  cout << var_5;
  return 0; }
```

```
int main () {
  string var_8 = "str_2";
  bool var_2;
  cin >> var_2;
  var_8 = "str_4" + var_8;
  var_2 = false;
  bool var_5 = true;
  if ( var_2 ) {
    bool temp = var_2;
    var_2 = var_5;
    var_5 = temp; }
  if ( var_2 ) {
    bool temp = var_2;
    var_2 = var_5;
    var_5 = temp; }
  cout << var_8;
  cout << var_2;
  cout << var_5;
  return 0; }
```

Figure 2: **(Left-Middle)** Example input and output of the base predictor on SANSTYPE. **(Right)** Output of the denoiser, which instantiates `var_5` and corrects the type of `temp`. Correcting the types requires global type inference across the entire program.

dataset we generated from pseudocode and code templates (see Figure 2 for an example). In SANSTYPE, the pseudocode does not supply types (e.g., `set x to 5`) and thus the model must learn to perform global type inference to generate correct code (e.g., `int x = 5` if `x` has not been previously initialized). We pre-train a denoiser with corruptions that delete and substitute tokens from unlabeled code examples.

Table 1 shows the results of standard fine-tuning on SANSTYPE. We find that directly applying the pre-trained denoiser Π to the outputs of the fine-tuned model f_{std} at test-time (i.e., a test-time denoiser) improves both validity and correctness of the output code by about 0.5% to 1.5%, depending on the number of layers. Note that increasing the number of layers does not improve the performance of f_{std}, but improves the performance of $\Pi \circ f_{\text{std}}$. This suggests that the denoiser Π improves with more layers but standard fine-tuning destroys the improved output structure.

4. Composed fine-tuning

Based on the observations in the previous section, we instantiate composed fine-tuning. In this framework, we assume the pre-trained denoiser Π is given as a probabilistic model p_Π and pre-trained on unlabeled output data. We learn the composed predictor $\Pi \circ f_\theta$ on labeled examples. We train a probabilistic model p_θ for the base predictor by

	# Layers	Test-time denoiser?	Compile Err	Exec Err	Correct
Std fine-tuning	6	N	13.2	7.0	79.8
Std fine-tuning	6	Y	12.8	6.8	80.4
Std fine-tuning	12	N	13.0	7.2	79.8
Std fine-tuning	12	Y	11.8	7.0	81.2

Table 1: Proportion of generated code (%) resulting in a compilation error, execution-time error, or correct code in the SANSTYPE task. Re-applying the pre-trained denoiser to the outputs of the fine-tuned model (intialized from the same denoiser) improves the performance.

maximizing (over θ):

$$\mathbb{E}_{x,y}[\mathbb{E}_{y' \sim p_\theta}[\log p_\Pi(y \mid y')]] + \lambda \mathbb{E}_{x,y}[\log p_\theta(y \mid x)]. \quad (1)$$

The first term is a lower bound on the log-likelihood of the composed predictor via p_Π and p_θ (see Appendix D). We optimize a lower bound since optimizing the log-likelihood directly requires computing an intractable sum over the output space. The second term is the log-likelihood of only p_θ, which is the same as in standard fine-tuning. We define the base predictor as $f_\theta(x) = \arg\max_y p_\theta(y \mid x)$.

Since the pre-trained Π is imperfect, the hyperparameter λ in the objective trades off between fitting the composition $\Pi \circ f_\theta$ and fitting f_θ directly to the data. For discrete outputs, the first term in the objective involves an expectation over a discrete space, which may require REINFORCE (Williams, 1992), straight-through estimation (Bengio et al., 2013), or a Gumbel-softmax reparameterization (Jang et al., 2017; Maddison et al., 2016) to optimize.

Adaptability to different denoisers. The compatibility between the base predictor and the denoiser can depend on the noising distribution used to pre-train the denoiser. We note that training the base predictor composed with the denoiser allows the base predictor to adapt to the choice of noising distribution. In Section 7.4 and Appendix B, we show that composed fine-tuning gives gains across a variety of noising distributions in both text and image domains.

Reducing complexity of the base predictor. Composing with the denoiser allows the base predictor more room for error by relying on the denoiser to correct them. For example, the base predictor can translate the pseudocode in SANSTYPE without getting all the global type inference decisions correct, leading to a simpler base predictor. Figure 2 (left) gives an example pseudocode input in SANSTYPE. With composed fine-tuning, the base predictor could learn to make code translations on a mostly local, line-by-line basis (a simpler solution) while relying on the denoiser to correct types, which requires type inference globally over the code.

5. Denoisers can reduce model complexity

In this section, we formalize the intuition that composed fine-tuning reduces the complexity of the base predictor. We study standard vs. composed fine-tuning from an approximation theory standpoint and use complexity measures on predictors as surrogates for sample complexity. Let $\|\cdot\|$ be a norm on a chosen hypothesis class $\mathcal{H}$. We compare the minimum norm of the base predictor f_{base} trained via composition with a given denoiser Π against the standard predictor $f_{\text{std}} \in \text{argmin}_{f\in\mathcal{H}}\{\|f\| : f(x) = f^\star(x), x \in \mathcal{X}\}$, a minimum norm predictor which represents $f^\star$ directly as in standard fine-tuning. Since we consider the minimizers, we do not capture the effects of initialization or optimization.

We aim to represent a target function $f^\star : \mathcal{X} \to \mathcal{V}$. We assume access to a denoiser $\Pi : \mathcal{Y} \to \mathcal{V}$ which projects to the nearest valid output for an appropriate metric on the output space (breaking ties arbitrarily). In Section 5.1, we give a simple example for when composing with a denoiser ($\Pi \circ f_{\text{base}}$) can drastically reduce the complexity of the learned predictor. Since f_{base} becomes easier to approximate, we may expect better generalization (Bartlett et al., 2017; Neyshabur et al., 2017; Wei & Ma, 2020; 2019). In Section 5.2, we theoretically show for two-layer ReLU networks that the complexity required to directly represent $f^\star$ can be arbitrarily larger than representing with a composed predictor depending on the stability of $f^\star$.

5.1. Motivating example

Figure 1 shows a staircase function $f^\star$ that requires a complex standard predictor f_{std} but the minimum norm base predictor f^*_{base} has low complexity. For $0<\delta<1$, let the input space $\mathcal{X} = \uplus_{i=1}^N [i-(1-\delta)/2, i+(1-\delta)/2]$ be a union of N disjoint intervals and the valid outputs $\mathcal{V} = \mathbb{Z}$ be the integers, a subset of the ambient output space $\mathcal{Y} = \mathbb{R}$. The staircase function is $f^\star(x) = \lfloor x \rceil$ defined on $\mathcal{X}$, which rounds a linear function onto the integers. Following Savarese et al. (2019), we define the norm of a univariate function $f : \mathbb{R} \to \mathbb{R}$ as

$$\|f\| = \frac{1}{2}\max\left(\int_{-\infty}^{\infty} |f''(x)|^2 dx, |f'(-\infty) + f'(+\infty)|\right). \tag{2}$$

"Complex" functions with constantly changing first derivatives will have a higher norm.

Consider representing $f^\star$ with linear splines, a family of piecewise linear functions. In linear splines, the norm in Equation (2) becomes roughly the sum of absolute changes in slope between piecewise segments. If we represent $f^\star$ directly with a linear spline f_{std}, the norm of f_{std} has to be large due to the large number of slope changes: $\|f_{\text{std}}\| = (N-1)/\delta$ (Figure 1 left).

Suppose we have access to a denoiser $\Pi(y) = \lfloor y \rceil$, which projects onto $\mathcal{V} = \mathbb{Z}$. Then a linear function f^*_{base} composed with Π can represent the staircase on $\mathcal{X}$, reducing the norm to 1 (Figure 1 right). By not requiring f^*_{base} to represent the local complexity and discreteness in $f^\star$, the base predictor f^*_{base} better captures the underlying globally linear structure of $f^\star$. In this case, the simpler linear base predictor also enables perfect extrapolation outside the training domain.

5.2. Analysis for 2-layer ReLU networks

We extend to more general hypothesis classes and high dimensional (discrete) outputs. Formally, we take the valid set $\mathcal{V} = \{y_1^*, ..., y_N^*\}$ to be a discrete set over N output values in $\mathbb{R}^k$ and $f^\star$ is a piecewise constant function defined on N disjoint intervals $\mathcal{X} = \uplus_{i=1}^N [x_i^l, x_i^u]$ (in ascending order), where there is a $\delta > 0$ gap between each interval and the next. The target function $f^\star$ is defined such that if $x \in [x_i^l, x_i^u]$, then $f^\star(x) = y_i^*$. We leave analysis of discrete, high-dimensional input spaces (such as in text tasks) to future work.

We study 2-layer ReLU networks, often studied as a first step towards understanding the expressivity of neural networks (Neyshabur et al., 2014; Savarese et al., 2019; Eldan & Shamir, 2016). Following Savarese et al. (2019), we define $f_\theta \in \mathcal{H}$ as

$$f_\theta(x) = \sum_{l=1}^{h} w_l^{(2)} \left[\langle w_l^{(1)}, x\rangle + b_l^{(1)}\right]_+ + b_l^{(2)}$$

on $x \in \mathbb{R}^d$, where we will take $d = 1$ throughout. Here, $[x]_+ = \max(x, 0)$ is the element-wise ReLU nonlinearity. The parameters θ contain the hidden unit size $h \in \mathbb{N}$ and all weights and biases. We let $W^{(1)} \in \mathbb{R}^{h\times d}$ denote the matrix with $w_l^{(1)} \in \mathbb{R}^d$ as rows and let $b^{(1)}, b^{(2)}, w^{(2)} \in \mathbb{R}^h$ be vectors with $b_l^{(1)}, b_l^{(2)}, w_l^{(2)} \in \mathbb{R}$ as elements respectively. We let Θ denote this parameter space.

Measure of complexity. Following Savarese et al. (2019), the complexity of a network is associated with the squared Euclidean norm of the weights

$$C(\theta) = \frac{1}{2}(\|w^{(2)}\|_2^2 + \|W^{(1)}\|_F^2).$$

The norm of f is the minimum norm required to represent it:

$$\|f\| = \inf_{\hat{\theta}\in\Theta} C(\hat{\theta}) \text{ s.t. } f_{\hat{\theta}} = f. \tag{3}$$

Savarese et al. (2019) showed that this norm is equivalent to Equation 2 for univariate networks. Since these complexity measures typically appear in generalization bounds (Bartlett et al., 2017; Neyshabur et al., 2017), we expect to improve generalization error by reducing these complexity measures.

Minimum complexity reduces with a denoiser. Given $\Pi(y) \in \text{argmin}_{y^*\in\mathcal{V}} \|y^* - y\|_2$ which is projection onto $\mathcal{V}$ (breaking ties arbitrarily), we want to compare the norms of

f_{std} that represents $f^\star$ directly and the minimum norm base predictor that represents $f^\star$:

$$f^*_{\text{base}} = \operatorname*{argmin}_{f \in \mathcal{H}} \{\|f\| : \Pi \circ f(x) = f^\star(x), x \in \mathcal{X}\}. \quad (4)$$

Note that $\|f^*_{\text{base}}\| \leq \|f_{\text{std}}\|$ since f_{std} is a feasible solution. Thus composing cannot increase the norm.

Adjacent intervals measure stability. Our result depends crucially on the number of *non-adjacent* pairs of intervals in $f^\star$. Suppose the output dimension is $k=1$. We define a pair of interval indices $(i, i+1)$ as *adjacent* if there is no valid output value $y \in \mathcal{V}$ in between y^*_i and y^*_{i+1}; that is, none of $y^*_i < y < y^*_{i+1}$ or $y^*_i > y > y^*_{i+1}$ hold. The number of non-adjacent interval pairs characterizes the instability of $f^\star$. Let $|J|$ be the number of non-adjacent pairs and $|I|$ be the number of adjacent pairs, where $|I|+|J|=N-1$. Our bound also depends on $L = \min_i |y^*_i - y^*_{i+1}|$ and $U = \max_i |y^*_i - y^*_{i+1}|$, the min and max separation between valid points. For higher output dimensions ($k>1$), let $y^*_{i,j}$ be the j-th output coordinate of the i-th valid point and let $|J_j|, |I_j|, L_j, U_j$ be the analogous quantities for each output coordinate $j \in [k]$.

Theorem 1. *Let the valid output space $\mathcal{V} \subset \mathbb{R}^k$ be a set over N multivariate output values $\{y^*_1, ..., y^*_N\}$ in $\mathcal{V}$. Let $f^\star : \mathbb{R} \to \mathbb{R}^k$ be a piecewise constant function defined on $\mathcal{X} = \uplus_{i=1}^N [x^l_i, x^u_i]$ where $f^\star(x) = y^*_i$ if $x \in [x^l_i, x^u_i]$. Let Δ_x be the length of the smallest interval in $\mathcal{X}$. Then*

$$\frac{\|f_{std}\|}{\|f^*_{base}\|} = \Omega\left(\frac{N \max_j L_j}{\sum_{j=1}^k U_j \left(|J_j| + \delta \frac{|I_j|}{\Delta_x} \right)} \right) \quad (5)$$

See Appendix A for a proof. Here, $|J_j|$ measures the instability of the target function coordinate j. If $|J_j|$ are sublinear in N and valid points are evenly spaced, then the gap is $\Omega(1/\delta)$ which can be arbitrarily large for a fixed output dimension as $\delta \to 0$ and $N \to \infty$. If any $|J_j|$ is linear in N (many non-adjacent intervals, unstable), then there is only a constant factor gap in the worst case. Overall, if $f^\star$ is stable with respect to its discrete output space, we can learn a simpler base predictor that still represents $f^\star$ when composed with the denoiser.

6. Experiments

We show on two pseudocode-to-code datasets that composed fine-tuning uses the same pre-trained model to improve over standard fine-tuning. Composed fine-tuning also improves other fine-tuning methods and semi-supervised methods such as backtranslation. We also show that composed fine-tuning can lead to better OOD performance on shifted pseudocode distributions and unseen programs in pseudocode-to-code, as well as unseen inputs in an image generation task.

6.1. Pseudocode-to-code

We evaluate composed fine-tuning on two pseudocode-to-code datasets, SANSTYPE and SPOC.

Prediction task and pre-trained model. We consider full-program pseudocode-to-code translation, where inputs $\mathcal{X}$ are human-generated pseudocode. The ambient output space $\mathcal{Y}$ is all possible strings and the set of valid outputs $\mathcal{V}$ are strings that compile with the g++ compiler. In contrast to previous works which decompose the problem into line-by-line translation and use information from the compiler (Kulal et al., 2019; Yasunaga & Liang, 2020), we translate the entire program at once without compiler access. Following Yasunaga & Liang (2020), the pre-training objective for both pseudocode-to-code datasets consists of repairing random semantic and syntactic corruptions of unlabeled code examples (using some domain knowledge of code). See Appendix C and Appendix E for pre-training details for each dataset.

Models and regularization. We use Transformers (Vaswani et al., 2017) for the base predictor and the denoiser. In all models, we use weight decay, dropout, attention dropout, and ReLU dropout as regularization and use $\lambda = 1$ to balance between the fitting the composed and direct objectives. During inference, we use greedy decoding for simplicity, although optimizations such as beam search or compiler access could improve the results further.

Metrics. A generated program has three possible outcomes: compilation error, execution error, or correct. A program is *correct* if, executed on a set of input test cases, its outputs match the set of gold outputs. We measure the proportion of programs that fall into these outcomes.

6.1.1. SANSTYPE

In SANSTYPE, the pseudocode specifies all but the declaration types, leaving global type constraints to the model (see Figure 2 and Appendix). Composed fine-tuning can avoid modeling everything with a complex model by learning a base predictor f_θ that can do local translation while the denoiser Π enforces global constraints such as type correctness.

Dataset generation. The synthetic programs involve 1–4 variables (bools, ints, and strings) drawn from 10 possible variable names, which are first initialized (by reading stdin) and then processed by up to 1–5 random operations, including 3 unary operations per type and 2 binary operations on ints. There are 100 possible integer values and 10 possible string values. Pseudocode for each line of code is generated from a few possible templates. We generate 1000 labeled examples and 20000 unlabeled code examples.

In-distribution and OOD test sets. The in-distribution test set has pseudocode generated from the same templates

	# Layers	Compile Err	Exec Err	Correct	Relative
SANSTYPE					
Baseline	6	51.4	12.0	36.6	-54.1
Standard fine-tuning	6	13.2	7.0	79.8	0.0
Composed fine-tuning	6	9.6	6.0	**84.4**	**5.8**
SANSTYPE (OOD)					
Baseline	6	62.8	23.0	14.2	-66.5
Standard fine-tuning	6	35.0	22.4	42.4	0.0
Composed fine-tuning	6	18.4	28.4	**52.8**	**24.5**
SPOC TESTW					
Baseline	10	48.7	16.6	34.5	-6.8
Standard fine-tuning	10	47.5	15.4	37.0	0.0
Composed fine-tuning	10	46.1	15.8	**38.1**	**3.0**
SPOC TESTP (OOD)					
Baseline	10	75.5	12.3	12.2	-17.6
Standard fine-tuning	10	75.3	10.0	14.8	0.0
Composed fine-tuning	10	74.2	10.4	**15.4**	**4.1**

Table 2: Proportion of generated code (%) resulting in a compilation error, execution-time error, or correct code in the SANSTYPE and SPOC pseudocode-to-code tasks. Last column is the relative improvement in correctness over standard fine-tuning.

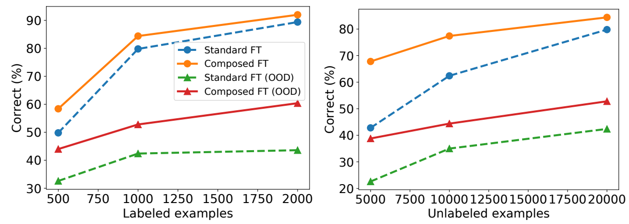
as the training set. The OOD test set uses a new set of pseudocode templates created by mixing and matching tokens from the in-distribution templates. For example, for the print statement `cout « <var>`, the in-distribution pseudocode templates include `print <var>` and `output <var> to stdout` while the OOD pseudocode contains the templates `print <var> to stdout`, `output <var>`, and `stdout <var>`. Although a benign shift, this is enough to significantly lower the performance of all models.

6.1.2. SPOC

We also evaluate on the challenging SPOC pseudocode-to-code dataset (Kulal et al., 2019), which contains code scraped from `codeforces.com` and pseudocode written by crowdsourced workers. Since we consider the full-program translation task instead of line-by-line as in previous works (Kulal et al., 2019; Yasunaga & Liang, 2020), we filter out training examples where the code is longer than 1000 tokens after pre-processing, retaining over 95% of the training examples.

In-distribution and OOD test sets. We use the two given SPOC test sets, TESTW and TESTP. TESTW tests for generalization to pseudocode (for seen programs) written by different crowdsourced workers, while TESTP tests for generalization to unseen problems. We consider TESTP to be the out-of-distribution test set. We report results on the full (unfiltered) test sets.

Results. Table 2 shows the results for a baseline Transformer trained from scratch, standard fine-tuning, and composed fine-tuning for SANSTYPE and SPOC. Composed fine-tuning gives about 6% and 3% relative improvement over standard fine-tuning for SANSTYPE and SPOC respectively on in-distribution test sets. On OOD test sets, the



(a) Vary labeled data (b) Vary unlabeled data

Figure 3: Performance with varying labeled and unlabeled data in SANSTYPE. Composed fine-tuning improves more over standard fine-tuning as sample size decreases.

relative improvement grows to 25% and 4% respectively. Thus, composed fine-tuning improves generalization on in-distribution examples and particularly out-of-distribution examples. Figure 3 shows results on varying unlabeled and labeled data sizes, where composed fine-tuning improves more over standard fine-tuning as sample size decreases, and the larger relative improvement in the OOD split holds across all sample sizes.

Figure 2 gives an example SANSTYPE input with the output of the base and composed predictors. With the denoiser, the base predictor does not have to output all the correct variable types. Here, the denoiser correctly instantiates `var_5` and corrects the type of `temp`.

7. Ablations and other comparisons

In this section, we consider applying composed fine-tuning to other fine-tuning methods and to backtranslation. We also perform an ablation on the effect of composed training with the denoiser and compare to scaled-up versions of the baselines with double the number of layers. Composed fine-tuning improves over these alternatives.

7.1. Scaled-up comparisons

Although composed fine-tuning does not optimize the denoiser, the final composed model consists of roughly double the number of layers as the baselines (12 vs 6 layers). Thus, we also compare to scaled-up baselines with double the amount of trainable parameters as the composed model. Note that the pre-trained denoiser is also double the size of the denoiser in composed fine-tuning models. We find that composed fine-tuning still improves over scaled-up baselines.

Table 3 shows the results of scaled-up baseline and standard fine-tuning models on SANSTYPE (models with 12 layers). While the scaled baseline predictor improves over the unscaled version, scaling up does not signficantly change the standard fine-tuning model. On SPOC, we find that scaling up (from 10 to 20 layers) does not significantly change the baseline performance. While the compilation error rate

	# Layers	Test-time denoiser?	Compile Err	Exec Err	Correct
Composed fine-tuning	6	Y	10.6	5.0	**84.4**
Scaled up					
Baseline	12	N	41.4	17.0	41.6
Standard fine-tuning	12	N	13.0	7.2	79.8
Test-time denoiser					
Baseline	6	Y	23.6	18.0	58.4
Standard fine-tuning	6	Y	12.8	6.8	80.4

Table 3: Results of scaled-up comparisons and baselines with a test-time denoiser on SANSTYPE. Proportion of generated code (%) resulting in a compilation error, execution-time error, or correct code. On SPOC, we find that scaling up does not significantly change the baseline performance and that using a test-time denoiser does not improve correctness of SPOC baselines.

	# Layers	Test-time denoiser?	Unlabeled + Labeled?	Compile Err	Exec Err	Correct
BART-style fine-tuning	6	N	N	10.6	6.2	83.2
BART-style fine-tuning	6	Y	N	10.0	6.4	83.6
BART-style fine-tuning	12	N	N	12.2	6.6	81.2
Composed BART-style fine-tuning	6	Y	N	8.6	6.0	**85.4**
Backtranslation (BT)	6	N	Y	3.6	10.0	86.4
Backtranslation (BT)	6	Y	Y	3.4	10.6	86.0
Backtranslation (BT)	12	N	Y	1.2	12.6	86.2
Composed fine-tuning + BT	6	Y	Y	3.2	9.8	**87.0**

Table 4: Results of BART-style fine-tuning and backtranslation on SANSTYPE. Proportion of generated code (%) resulting in a compilation error, execution-time error, or correct code. Backtranslation uses both unlabeled and labeled data together when learning the downstream task for targeted usage of unlabeled data.

decreases slightly, scaling up slightly degrades the correct rate of its output programs.

7.2. Comparisons to baselines with a test-time denoiser

We compare to baselines where we re-apply the pre-trained denoiser at test time (a test-time denoiser). With the test-time denoiser, the baselines have the same architecture and number of trainable parameters as the composed predictor. The only difference is that composed fine-tuning composes the base predictor with the denoiser during training. Composed fine-tuning improves over models with a test-time denoiser, highlighting the importance of composed training with the fixed denoiser.

Table 3 shows the results on SANSTYPE. While standard fine-tuning improves with a test-time denoiser, composed fine-tuning still improves over standard fine-tuning by 4%. We find that using a test-time denoiser does not improve correctness of SPOC baselines.

7.3. Composed fine-tuning improves other semi-supervised methods

We compare to a more sophisticated fine-tuning method from BART (Lewis et al., 2020) and backtranslation (Sennrich et al., 2016b), showing complementary gains even when scaling up or applying a test-time denoiser to these methods.

BART-style fine-tuning. BART (Lewis et al., 2020) proposes a modified two-stage fine-tuning strategy which freezes the later layers of the encoder-decoder in the first stage and then fine-tunes the entire network in the second stage (see Appendix C for details). We apply composed fine-tuning to this by initializing the base predictor from the result of BART-style fine-tuning and doing composed training with the fixed pre-trained denoiser. Table 4 shows that this improves further over BART-style fine-tuning (by 2-4%), showing that the benefits are complementary.

Backtranslation. A feature of pre-training is that we do not need access to unlabeled data when fine-tuning on the downstream task. In contrast, backtranslation (Sennrich et al., 2016b) is a strong method that uses the large unlabeled dataset together with the labeled data in downstream learning, which is expensive but allows the method to target the usage of unlabeled data to the downstream task. Backtranslation uses an output-to-input model (learned on the labeled data) applied on unlabeled outputs to generate additional synthetic inputs. We employ composed fine-tuning on top by initializing the base predictor from the model trained by backtranslation, showing complementary benefits to backtranslation (0.6% to 1%) in Table 4.

7.4. Effect of pre-training objectives

Naively, better pre-trained denoisers should improve the gains with composed fine-tuning. For example, the SPOC denoiser is not as effective as the SANSTYPE denoiser, which correlates with less gains with composed fine-tuning. Interestingly, the relationship between pre-trained denoiser quality and downstream performance is not straightforward. Table 5 shows results when we train a SANSTYPE denoiser using random word deletions (no perturbations with any domain knowledge of code). The baseline with test-time denoiser worsens from 58.4% to 46.2% (correct rate), suggesting that the denoiser is worse when trained with the random deletion objective. However, the downstream standard and composed fine-tuning performance improve substantially for both in- and out-of-distribution examples (by 2% and 8-10% respectively). Thus the new denoiser (using more canonical perturbations) is worse at correcting baseline outputs but improves fine-tuning results. The effect of pre-training on the downstream performance after adaptation should be a topic of further investigation.

8. OOD generalization in image generation

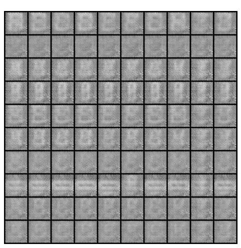
We evaluate the general framework of composed training with a denoiser on font image generation from unseen attribute combinations, a natural OOD generalization task.

	Denoiser	# Layers	Compile Err	Exec Err	Correct
Test-time denoiser					
Baseline	Code-based	6	23.6	18.0	**58.4**
Baseline	Random deletions	6	35.6	18.2	46.2
SansType					
Standard fine-tuning	Code-based	6	13.2	7.0	79.8
Standard fine-tuning	Random deletions	6	8.0	10.8	**81.2**
Composed fine-tuning	Code-based	6	9.6	6.0	84.4
Composed fine-tuning	Random deletions	6	5.2	8.6	**86.2**
SansType (OOD)					
Standard fine-tuning	Code-based	6	35.0	22.4	42.4
Standard fine-tuning	Random deletions	6	30.2	15.8	**54.0**
Composed fine-tuning	Code-based	6	18.4	28.4	52.8
Composed fine-tuning	Random deletions	6	24.4	15.4	**60.2**

Table 5: SansType results on standard and composed fine-tuning when changing the pre-training denoising autoencoder objective from correcting code-based perturbations to correcting random, unstructured word deletions. The denoising quality of the random deletion denoiser is worse when applied to a baseline model, but it leads to better downstream performance both in-distribution and OOD.

(a) Baseline

(b) Composed



(c) Base

Figure 4: Generated letters A-J for 10 randomly selected fonts. **(a)** The baseline predictor makes blurry outputs with many artifacts. **(b)** The composed predictor (base + denoiser) makes clearer outputs with more distinct font patterns. **(c)** The base predictor produces blurrier outputs corresponding to a lower norm model.

Generating missing characters is useful when prototyping a new font (Miyazaki et al., 2020; Tenenbaum & Freeman, 2000). From a few user-supplied characters, the model fills in the rest. This task closely mirrors the theoretical setup, where the input is low-dimensional (index of the font and character type) to a high-dimensional output (image). Qualitatively, image samples from our composed predictor are clearer and has less artifacts.

Prediction task and pre-trained model. We map two one-hot vectors, corresponding to the character identity (out of 62 possible) and the font (out of 100 fonts), to generate 32×32 grayscale font images. Here, valid font images have clean lines and adhere to the font styling. We train using the pixel-wise squared error loss and tune L2 regularization strength on a validation set. To train the composed predictor, we set $\lambda = 0$ in (1), using only the composed loss. The denoiser is pre-trained to sharpen unlabeled font images distorted by a Gaussian blur filter with randomly sampled radii in $[0, 2]$. We also report gains with other noising functions (embossing, contrast perturbations) in Appendix B. We do not consider fine-tuning from the denoiser in this task since the predictor is not an image-to-image model.

Data. We use a dataset of 56k fonts originally scraped from the Internet (Bernhardsson, 2016). Out of the 6200 labeled examples (62 characters $\times$ 100 fonts), we split randomly into 2500 training examples, 100 validation examples, and 3600 test examples. The training examples contain a random subset of the characters for each font. The models must generate the unseen characters of each font with the correct font styling at test-time. The denoiser uses additional unlabeled images for $\sim$50k other fonts.

Models and metrics. The baseline and base predictors 7-layer fully-connected networks from attribute to image (see Appendix B) . The denoiser Π is a 3-layer U-Net (Ronneberger et al., 2015) image-to-image model. We test image sample quality directly by computing the pixel-wise squared error with respect to ground truth test images.

Results. The composed predictor achieves an 11% reduction in test MSE (0.193 to 0.171) compared to the baseline predictor. We also find that adding dropout improves the MSE further to 0.165, suggesting that dropout helps in finding a lower complexity base predictor. We visualize the predicted images for some randomly-selected fonts for comparison (Figure 4). The base predictor outputs noisy gray images, suggesting that it has learned a lower complexity model. In contrast, directly outputting clearly defined lines and transitions between black and white pixels requires a relatively high complexity model. Additional results on varying labeled and unlabeled data size are in Appendix B, where the performance of the Composed model improves upon the baseline on all instances.

9. Related work and discussion

Sequence-to-sequence pre-training. Denoising autoencoders (DAE) are classical building blocks for unsupervised deep representation learning (Vincent et al., 2008; 2010). Recently, DAEs have been used for large scale sequence-to-sequence pre-training as in BART (Lewis et al., 2020) or T5 (Raffel et al., 2019). Machine translation methods also use monolingual data in both the source and target languages to improve their models (Sennrich et al., 2016b; Cheng et al., 2016). Pretraining methods use language modeling on monolingual data to initialize the encoder and decoder (Ramachandran et al., 2018; Skorokhodov et al., 2018; Devlin et al., 2019). Our work is complementary to these pre-training methods in that we aim to improve the fine-tuning step to fully leverage the pre-trained model.

Backtranslation. Back-translation methods generate additional synthetic parallel examples by training on the backwards (target to source) problem (Sennrich et al., 2016b). While back-translation is a strong baseline method, it requires using the large unlabeled dataset for every downstream task. We showed that applying composed fine-tuning to backtranslation gives complementary gains.

Semantic parsing and structured prediction. Some recent semantic parsing works have explicitly provided output constraints using abstract syntax trees (AST) and enforcing type constraints (Yin & Neubig, 2017; Krishnamurthy et al., 2017; Xiao et al., 2016; Dong & Lapata, 2016). Krishnamurthy et al. (2017) note that enforcing type constraints during training not only prevents invalid outputs but also improves generalization, supporting our results. Structured prediction spans applications including speech (Zhang & Wu, 2013), vision (Mueller, 2013), and medical diagnosis (Jagannatha & Yu, 2016). Many approaches use graphical models (on top of neural models) for enforcing validity, e.g. HMMs and CRFs in OCR and sequence tagging (Kassel, 1995; Huang et al., 2015). These approaches typically require carefully engineering the graphical model to integrate with a neural component and do not consider the simplicity benefits of composition. Mostajabi et al. (2018) have a similar goal of squeezing more information out of structured output spaces. However, we utilize unlabeled output data and use the frozen denoiser in the final model, improving OOD generalization. Since we do denoising autoencoding, we use the entire denoiser (instead of half) to correct outputs. Work on approximate inference for structured prediction (Tu & Gimpel, 2019) model the energy scoring and inference processes jointly, which is possibly complex to learn. We learn the denoiser (which like an inference algorithm in this comparison) separately on unlabeled outputs. Composing *simplifies* the base predictor for OOD improvements.

Semi-supervised learning. Like semi-supervised learning, composed fine-tuning leverages large amounts of unlabeled data through the pre-trained denoiser. However, semi-supervised learning works typically use unlabeled *input* data (Tarvainen & Valpola, 2017; Miyato et al., 2018; Shu et al., 2018; Berthelot et al., 2019), whereas we have "unlabeled" outputs. In classification, unlabeled outputs can help with handling label shift (Lipton et al., 2018; Azizzadenesheli et al., 2019), but otherwise there is very little output structure. If both unlabeled inputs and outputs are available, our method is complementary with semi-supervised methods.

Catastrophic forgetting. Methods for mitigating catastrophic forgetting (French, 1999; McCloskey & Cohen, 1989; Ratcliff, 1990), the phenomenon that continually-trained models lose the ability to do earlier tasks, often try to learn a model that does well on all tasks it has been trained on. This motivates multi-task learning/rehearsal (Rusu et al., 2015; Parisotto et al., 2015) (which uses data from all tasks at once, training jointly) and elastic weight consolidation (Kirkpatrick et al., 2017) (which tries to partition the network into separate experts for each task). Our goal is different: we do not require the fine-tuned model to do the pre-training task well, and we do not access the unlabeled data during downstream learning, making multi-task methods inapplicable.

Lightweight fine-tuning. While we consider using the pre-trained denoiser as initialization for the base predictor in this paper, this requires a model double the size of the denoiser. Lightweight fine-tuning methods such as adapter-tuning, prefix-tuning, and prompt tuning (Houlsby et al., 2019; Li & Liang, 2021; Lester et al., 2021) that keep the denoiser fixed when fine-tuning the base predictor are solutions for making this less expensive, since we can store the parameters of the denoiser once for both the base predictor and denoiser. Lightweight fine-tuning works also find that freezing pre-trained parameters tends to improve OOD results, though we also find in-domain improvements. Our theoretical analysis gives a starting point for understanding why OOD performance can improve with frozen parameters.

10. Conclusion

Many tasks in machine learning are no longer classification or regression but require generating outputs with rich structure (images, text, music, proteins, etc.), for which unpaired outputs are very common. We introduce composed fine-tuning to improve the fine-tuning step after pre-training. Our work prompts further study into what pre-training captures about the unlabeled data and which fine-tuning methods can maximize the benefits of pre-training, especially in tasks with a rich output structure.

11. Acknowledgements

We thank Michi Yasunaga, Robin Jia, Albert Gu, Karan Goel, Rohan Taori, and reviewers for helpful discussions and comments. SMX is supported by an NDSEG Graduate Fellowship. The work is partially supported by a PECASE award, SDSI, and SAIL at Stanford University.

12. Reproducibility

All data and code for reproducing the experiments are on our CodaLab worksheet and GitHub repository.

References

Azizzadenesheli, K., Liu, A., Yang, F., and Anandkumar, A. Regularized learning for domain adaptation under label shifts. In *International Conference on Learning Representations (ICLR)*, 2019.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Bartlett, P., Foster, D. J., and Telgarsky, M. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Bengio, Y., Leonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv*, 2013.

Bernhardsson, E. Analyzing 50k fonts using deep neural networks, 2016.

Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., and Raffel, C. Mixmatch: A holistic approach to semi-supervised learning. *arXiv*, 2019.

Cheng, Y., Xu, W., He, Z., He, W., Wu, H., Sun, M., and Liu, Y. Semi-supervised learning for neural machine translation. In *Association for Computational Linguistics (ACL)*, 2016.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Association for Computational Linguistics (ACL)*, pp. 4171–4186, 2019.

Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., and Smith, N. Finetuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv*, 2020.

Dong, L. and Lapata, M. Language to logical form with neural attention. In *Association for Computational Linguistics (ACL)*, 2016.

Eldan, R. and Shamir, O. The power of depth for feedforward neural networks. In *Conference on Learning Theory (COLT)*, 2016.

French, R. M. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3, 1999.

Github. Repository search for public repositories. `https://github.com/search?q=is:public`, 2021.

Guzmán, F., Chen, P.-J., Ott, M., Pino, J., Lample, G., Koehn, P., Chaudhary, V., and Ranzato, M. Two new evaluation datasets for low-resource machine translation: Nepali-English and sinhala-English. *arXiv*, 2019.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for NLP. *arXiv*, 2019.

Huang, Z., Xu, W., and Yu, K. Bidirectional LSTM-CRF models for sequence tagging. *arXiv*, 2015.

Jagannatha, A. and Yu, H. Structured prediction models for RNN based sequence labeling in clinical text. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2017.

Kassel, R. H. *A comparison of approaches to on-line handwritten character recognition*. PhD thesis, Massachusetts Institute of Technology, 1995.

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*, 2017.

Krishnamurthy, J., Dasigi, P., and Gardner, M. Neural semantic parsing with type constraints for semi-structured tables. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

Kudo, T. and Richardson, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Kulal, S., Pasupat, P., Chandra, K., Lee, M., Padon, O., Aiken, A., and Liang, P. Spoc: Search-based pseudocode to code. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Lee, C., Cho, K., , and Kang, W. Mixout: Effective regularization to finetune large-scale pretrained language models. In *International Conference on Learning Representations (ICLR)*, 2020.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv*, 2021.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Association for Computational Linguistics (ACL)*, 2020.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv*, 2021.

Lipton, Z. C., Wang, Y., and Smola, A. J. Detecting and correcting for label shift with black box predictors. In *International Conference on Machine Learning (ICML)*, 2018.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. *The psychology of learning and motivation*, 24, 1989.

Miyato, T., ichi Maeda, S., Ishii, S., and Koyama, M. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

Miyazaki, T., Tsuchiya, T., Sugaya, Y., Omachi, S., Iwamura, M., Uchida, S., and Kise, K. Automatic generation of typographic font from a small font subset. *IEEE Computer Graphics and Applications*, 40, 2020.

Mostajabi, M., Maire, M., and Shakhnarovich, G. Regularizing deep networks by modeling and predicting label structure. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

Mueller, A. *Methods for Learning Structured Prediction in Semantic Segmentation of Natural Images*. PhD thesis, University of Bonn, 2013.

Méndez-Lucio, O., Baillif, B., Clevert, D.-A., Rouquié, D., and Wichard, J. De novo generation of hit-like molecules from gene expression signatures using artificial intelligence. *Nature Communications*, 11, 2020.

Neyshabur, B., Tomioka, R., and Srebro, N. In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv*, 2014.

Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Ramachandran, P., Liu, P. J., and Le, Q. V. Unsupervised pretraining for sequence to sequence learning. *arXiv*, 2018.

Ratcliff, R. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97, 1990.

Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional networks for biomedical image segmentation. *arXiv*, 2015.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Savarese, P., Evron, I., Soudry, D., and Srebro, N. How do infinite width bounded norm networks look in function space? *arXiv*, 2019.

Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W. R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D. T., Silver, D., Kavukcuoglu, K., and Hassabis, D. Improved protein structure prediction using potentials from deep learning. *Nature*, 577, 2020.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. In *Association for Computational Linguistics (ACL)*, 2016a.

Sennrich, R., Haddow, B., and Birch, A. Improving neural machine translation models with monolingual data. *arXiv*, 2016b.

Shu, R., Bui, H. H., Narui, H., and Ermon, S. A DIRT-T approach to unsupervised domain adaptation. In *International Conference on Learning Representations (ICLR)*, 2018.

Skorokhodov, I., Rykachevskiy, A., Emelyanenko, D., Slotin, S., and Ponkratov, A. Semi-supervised neural machine translation with language models. In *AMTA 2018 Workshop on Technologies for MT of Low Resource Languages (LoResMT)*, 2018.

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

Tarvainen, A. and Valpola, H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in neural information processing systems*, pp. 1195–1204, 2017.

Tenenbaum, J. B. and Freeman, W. T. Separating style and content with bilinear models. *Neural Computation*, 12, 2000.

Tu, L. and Gimpel, K. Benchmarking approximate inference methods for neural structured prediction. In *North American Association for Computational Linguistics (NAACL)*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Vincent, P., Larochelle, H., Bengio, Y., , and Manzagol, P.-A. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning (ICML)*, 2008.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research (JMLR)*, 11, 2010.

Wei, C. and Ma, T. Data-dependent sample complexity of deep neural networks via Lipschitz augmentation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Wei, C. and Ma, T. Improved sample complexities for deep networks and robust classification via an all-layer margin. In *International Conference on Learning Representations (ICLR)*, 2020.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Xiao, C., Dymetman, M., and Gardent, C. Sequence-based structured prediction for semantic parsing. In *Association for Computational Linguistics (ACL)*, 2016.

Yasunaga, M. and Liang, P. Graph-based, self-supervised program repair from diagnostic feedback. In *International Conference on Machine Learning (ICML)*, 2020.

Yin, P. and Neubig, G. A syntactic neural model for general-purpose code generation. In *Association for Computational Linguistics (ACL)*, pp. 440–450, 2017.

Zhang, T., Wu, F., Katiyar, A., Weinberger, K. Q., and Artzi, Y. Revisiting few-sample BERT fine-tuning. In *International Conference on Learning Representations (ICLR)*, 2021.

Zhang, X.-L. and Wu, J. Denoising deep neural networks based voice activity detection. *arXiv*, 2013.