Sluice networks: Learning what to share between loosely related tasks

Sebastian Ruder[1,2]*, **Joachim Bingel**[3], **Isabelle Augenstein**[4]*, **Anders Søgaard**[3]
[1]Insight Research Centre, National University of Ireland, Galway
[2]Aylien Ltd., Dublin, Ireland
[3]Department of Computer Science, University of Copenhagen, Denmark
[4]Department of Computer Science, UCL, UK
s.ruder1@nuigalway.ie, {bingel|soegaard}@di.ku.dk, i.augenstein@ucl.ac.uk

Abstract

Multi-task learning is partly motivated by the observation that humans bring to bear what they know about related problems when solving new ones. Similarly, deep neural networks can profit from related tasks by sharing parameters with other networks. However, humans do not consciously decide to transfer knowledge between tasks (and are typically not aware of the transfer). In machine learning, it is hard to estimate if sharing will lead to improvements; especially if tasks are only loosely related. To overcome this, we introduce SLUICE NETWORKS, a general framework for multi-task learning where trainable parameters control the amount of sharing – including which parts of the models to share. Our framework goes beyond and generalizes over previous proposals in enabling hard or soft sharing of all combinations of subspaces, layers, and skip connections. We perform experiments on three task pairs from natural language processing, and across seven different domains, using data from OntoNotes 5.0, and achieve up to 15% average error reductions over common approaches to multi-task learning. We analyze when the architecture is particularly helpful, as well as its ability to fit noise. We show that a) label entropy is predictive of gains in sluice networks, confirming findings for hard parameter sharing, and b) while sluice networks easily fit noise, they are robust across domains in practice.

1 Motivation

While there is theory providing guarantees for effectivity of some types of multi-task learning [2, 3], none of these hold for the loosely related tasks to which multi-task learning is most often applied in natural language processing (NLP) or computer vision [8, 25]. To compensate for the lack of theory in the case of loosely related tasks, researchers have recently started to explore multi-task learning from a more experimental point of view, correlating performance gains with task properties to achieve a better understanding of when models can profit from auxiliary tasks [4, 19]. While such works have shed partial light on the effectiveness of particular approaches to multi-task learning, it remains hard to predict what parts of networks benefit from sharing, and to what extent they do so. Our limited understanding of multi-task learning is also a practical problem. With tens, if not hundreds of approaches to multi-task learning – from hard to soft parameter sharing, from sharing all layers to sharing only inner or outer layers, to sharing only specific subspaces – it is not feasible to exhaustively explore the space of multi-task learning when developing models for specific problems.

Most previous work in multi-task learning therefore only considers at most a couple of architectures for sharing [25, 22, 19]. In contrast, we present a framework that unifies such different approaches by

*Work done while the author was visiting the University of Copenhagen.

introducing trainable parameters for the components that differentiate multi-task learning approaches. We build on recent work trying to learn where to split merged networks [21], as well as work trying to learn how best to combine private and shared subspaces [5, 18].

Our model is empirically justified and deals with the *dirtiness* [16] of loosely related tasks. We show that it is a generalization of various multi-task learning algorithms such as hard parameter sharing [7], low supervision [25], and cross-stitch networks [21], as well as transfer learning algorithms such as frustratingly easy domain adaptation [9]. Moreover, we study what task properties predict gains, and what properties correlate with learning certain types of sharing, as well as the inductive bias of the resulting architecture.

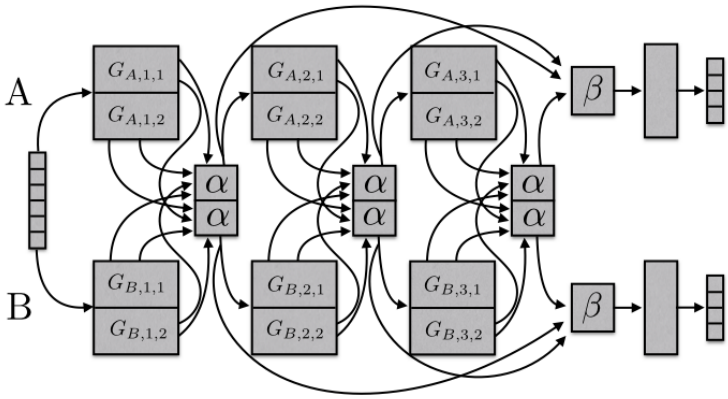

Figure 1: A SLUICE NETWORK with one main task A and one auxiliary task B. It consists of a shared input layer (shown left), two task-specific output layers (right), and three hidden layers per task, each partitioned into two subspaces. α parameters control which subspaces are shared between main and auxiliary task, while β parameters control which layer outputs are used for prediction.

2 An Architecture for Learning to Share

We introduce a novel architecture for multi-task learning, which we refer to as a SLUICE NETWORK, sketched in Figure 1 for the case of two tasks. The network learns to share parameters between augmented, deep recurrent neural networks [13]. The recurrent networks could easily be replaced with multi-layered perceptrons or convolutional neural networks for other applications.

The two networks A and B share an embedding layer associating the elements of an input sequence, in our case English words, with vector representations via word and character embeddings. The two sequences of vectors are then passed on to their respective inner recurrent layers. Each of these layers is divided into subspaces, e.g., for A into $G_{\mathrm{A},1,1}$ and $G_{\mathrm{A},1,2}$, which allow the network to learn task-specific and shared representations, if beneficial. The output of the inner layer of network A is then passed to its second layer, as well as to the second layer of network B. This traffic of information is mediated by a set of parameters α in a way such that the second layer of each network receives a weighted combination of the output of the two inner layers. The subspaces have different weights. Importantly, these weights are trainable and allow the model to learn whether to share, whether to restrict sharing to a shared subspace, etc. Finally, a weighted combination of the outputs of the outer recurrent layers $G_{.,3,.}$ as well as the weighted outputs of the inner layers are mediated through β parameters, which reflect a mixture over the representations at various depths of the network. In sum, sluice networks have the capacity to learn what layers and subspaces should be shared, as well as at what layers the network has learned the best representations of the input sequences.

Matrix Regularization We cast learning what to share as a *matrix regularization* problem, following [15, 29]. Assume M different tasks that are loosely related, with M potentially non-overlapping datasets $\mathcal{D}_1, \ldots, \mathcal{D}_M$. Each task is associated with a deep neural network with K layers $L_1, \ldots L_K$. We assume that all the deep networks have the same hyper-parameters at the outset. With loosely related tasks, one task may be better modeled with one hidden layer; another one with two [25]. Our

architecture, however, is flexible enough to learn this, if we initially associate each task with the *union* of the *a priori* task networks.

Let $W \in \mathbb{R}^{M \times D}$ be a matrix in which each row i corresponds to a model θ_i with D parameters. The loss functions $\mathcal{L}_i$ are cross-entropy functions of the form $-\sum_y p(y) \log q(y)$ below, but note that sluice networks are not restricted to tasks with the same loss functions. Let λ_i be weights that determine the importance of the different tasks during training.

The loss that sluice networks minimize, with a penalty term $\Omega(W)$, is then as follows:

$$\lambda_1 \mathcal{L}_1(\mathbf{f}(x; \theta_1), z) + \ldots + \lambda_M \mathcal{L}_M(\mathbf{f}(x; \theta_M), z) + \Omega(W)) \tag{1}$$

Sluice networks not only learn the parameters in W, but also some of the parameters of the regularizer Ω, through the α weights, while the β weights are used to learn the parameters of the mixture functions $\mathbf{f}(\cdot)$.

Learning Matrix Regularizers We now explain how updating α parameters can lead to different matrix regularizers. Each matrix W consists of M rows where M is the number of tasks. Each row is of length D with D the number of parameters. Subvectors $L_{m,k}$ correspond to the parameters of network m at layer k. Each layer consists of two subspaces with parameters $G_{m,k,1}$ and $G_{m,k,2}$.

Recall that our architecture is partly motivated by the observation that for loosely related tasks, only certain features in specific layers should be shared, while many of the layers and subspaces may remain more task-specific [25]. We want to learn what to share while inducing models for the different tasks. For simplicity, we ignore subspaces at first and assume only two tasks A and B. The outputs $h_{A,k,t}$ and $h_{B,k,t}$ of the k-th layer for time step t for task A and B respectively interact through what [21] refer to as *cross-stitch units* α (see Figure 1). Omitting t for simplicity, the output of the α layers is:

$$\begin{bmatrix} \widetilde{h}_{\mathrm{A},k} \\ \widetilde{h}_{\mathrm{B},k} \end{bmatrix} = \begin{bmatrix} \alpha_{\mathrm{AA}} & \alpha_{\mathrm{AB}} \\ \alpha_{\mathrm{BA}} & \alpha_{\mathrm{BB}} \end{bmatrix} \begin{bmatrix} {h_{\mathrm{A},k}}^\top, & {h_{\mathrm{B},k}}^\top \end{bmatrix} \tag{2}$$

where $\widetilde{h}_{\mathrm{A},k}$ is a linear combination of the outputs that is fed to the $k+1$-th layer of task A, and $\begin{bmatrix} a^\top, b^\top \end{bmatrix}$ designates the stacking of two vectors $a, b \in \mathbb{R}^D$ to a matrix $M \in \mathbb{R}^{2 \times D}$.

Extending the α-layers to include subspaces, for 2 tasks and 2 subspaces, we obtain an α matrix $\in \mathbb{R}^{4 \times 4}$ that not only controls the interaction between the layers of both tasks, but also between their subspaces:

$$\begin{bmatrix} \widetilde{h}_{\mathrm{A}_1,k} \\ \vdots \\ \widetilde{h}_{\mathrm{B}_2,k} \end{bmatrix} = \begin{bmatrix} \alpha_{\mathrm{A}_1\mathrm{A}_1} & \ldots & \alpha_{\mathrm{B}_2\mathrm{A}_1} \\ \vdots & \ddots & \vdots \\ \alpha_{\mathrm{A}_1\mathrm{B}_2} & \ldots & \alpha_{\mathrm{B}_2\mathrm{B}_2} \end{bmatrix} \begin{bmatrix} {h_{\mathrm{A}_1,k}}^\top, & \ldots, & {h_{\mathrm{B}_2,k}}^\top \end{bmatrix} \tag{3}$$

where $h_{\mathrm{A}_1,k}$ is the output of the first subspace of the k-th layer of task A and $\widetilde{h}_{\mathrm{A}_1,k}$ is the linear combination for the first subspace of task A. The input to the $k+1$-th layer of task A is then the concatenation of both subspace outputs: $h_{\mathrm{A},k} = \begin{bmatrix} \widetilde{h}_{\mathrm{A}_1,k}, & \widetilde{h}_{\mathrm{A}_2,k} \end{bmatrix}$.

Different α weights correspond to different matrix regularizers Ω, including several ones that have been proposed previously for multi-task learning. We review those in §3. For now just observe that if all α-values are set to 0.25 (or any other constant), we obtain hard parameter sharing [6], which is equivalent to a heavy L_0-regularizer.

Adding Inductive Bias Naturally, we can also add inductive bias to sluice networks by partially constraining the regularizer or adding to the learned penalty. Inspired by recent work on shared-space component analysis [31, 24], we add a penalty to enforce a division of labor and discourage redundancy between shared and task-specific subspaces. While the networks can theoretically learn such a separation, an explicit constraint empirically leads to better results and enables the sluice networks to take better advantage of subspace-specific α-values. This is modeled by an orthogonality constraint [5] between the layer-wise subspaces of each model:

$$\mathcal{L}_c = \sum_{m=1}^{M} \sum_{k=1}^{K} \| {G_{m,k,1}}^\top G_{m,k,2} \|_F^2 \tag{4}$$

where M is the number of tasks, K is the number of layers, $\| \cdot \|_F^2$ is the squared Frobenius norm and $G_{m,k,1}$ and $G_{k,2,m}$ are the first and second subspace respectively in the k-th layer of m-th task model.

Learning Mixtures Many tasks have an implicit hierarchy that informs their interaction. Rather than predefining it [25, 11], we enable our model to learn hierarchical relations by associating different tasks with different layers if this is beneficial for learning. Inspired by advances in residual learning [12], we employ skip-connections from each layer, controlled using β parameters. This layer acts as a mixture model, returning a mixture of expert predictions:

$$\widetilde{h}_{\mathrm{A}}^{\top} = \begin{bmatrix} \beta_{\mathrm{A},1} \\ \cdots \\ \beta_{\mathrm{A},k} \end{bmatrix}^{\top} \begin{bmatrix} {h_{\mathrm{A},1}}^{\top}, & \ldots & {h_{\mathrm{A},k}}^{\top} \end{bmatrix} \tag{5}$$

where $h_{\mathrm{A},k}$ is the output of layer k of model A, while $\widetilde{h}_{\mathrm{A},t}$ is the linear combination of all layer outputs of model A that is fed into the final softmax layer.

Complexity Our model only adds a minimal number of additional parameters compared to single-task models of the same architecture. α parameters scale linearly with the number of layers and quadratically with the number of tasks and subspaces, while β parameters scale linearly with the number of tasks and the number of layers. For a sluice network with M tasks, K layers per task, and 2 subspaces per layer, we thus obtain $4KM^2$ additional α parameters and KM β parameters. Training sluice networks is not much slower than training hard parameter sharing networks, with only an 5–7% increase in training time.

3 Previous Proposals as Instantiations of Sluice Networks

The architecture is very flexible and can be seen as a generalization over several existing algorithms for transfer and multi-task learning, including [7, 9, 25, 21]. We show how to derive each of these below.

Hard Parameter Sharing in the two networks appears if all α values are set to the same constant [7, 8]. This is equivalent to a mean-constrained ℓ_0-regularizer $\Omega(\cdot) = |\cdot|_0^{\bar{w}_i}$ and $\sum_i \lambda_i \mathcal{L}_i < 1$. If the sum of weighted losses are smaller than 1, the loss with penalty is always the highest when all parameters are shared.

Group Lasso The ℓ_1/ℓ_2 group lasso regularizer is $\sum_{g=1}^{G} ||G_{1,i,g}||_2$, a weighted sum over the ℓ_2 norms of the groups, often used to enforce subspace sharing [33, 26]. Our architecture learns a ℓ_1/ℓ_2 group lasso over the two subspaces (with the same degrees of freedom), when all $\alpha_{A,B}$ and $\alpha_{B,A}$-values are set to 0. When the outer layer α-values are not shared, we get block communication between the networks.

Frustratingly Easy Domain Adaptation The approach to domain adaptation in [9], which relies on a shared and a private space for each task or domain, can be encoded in sluice networks by setting all $\alpha_{A,B}$- and $\alpha_{B,A}$-weights associated with $G_{i,k,1}$ to 0, while setting all $\alpha_{A,B}$-weights associated with $G_{i,k,2}$ to $\alpha_{B,B}$, and $\alpha_{B,A}$-weights associated with $G_{i,k,2}$ to $\alpha_{A,A}$. Note that [9] discusses three subspaces. We split the space in two, leading to three subspaces, if we only share one half across the two networks.

Low Supervision [25] propose a model where only the inner layers of two deep recurrent works are shared. This is obtained using heavy mean-constrained L_0 regularization over the first layer $L_{i,1}$, e.g., $\Omega(W) = \sum_i^K ||L_{i,1}||_0$ with $\sum_i \lambda_i \mathcal{L}(i) < 1$, while for the auxiliary task, only the first layer β parameter is set to 1.

Cross-Stitch Networks [21] introduce cross-stitch networks that have α values control the flow between layers of two convolutional neural networks. Their model corresponds to setting the α-values associated with $G_{i,j,1}$ be identical to those for $G_{i,j,2}$, and by letting all but the β-value associated with the outer layer be 0.

In our experiments, we include hard parameter sharing, low supervision, and cross-stitch networks as baselines. We do not report results for group lasso and frustratingly easy domain adaptation, which were consistently inferior on development data by some margin.

4 Experiments

4.1 Experimental Setup

Data We want to experiment with multiple loosely related NLP tasks, but also study performance across domains to make sure our architecture is not prone to overfitting. As testbed for our experiments, we therefore choose the OntoNotes 5.0 dataset [28], not only due to its high inter-annotator agreement [14], but also because it enables us to analyze the generalization ability of our models across different tasks and domains. The OntoNotes

	Domains													
	Broadcast conversation (bc)		Broadcast news (bn)		Magazines (mz)		Newswire (nw)		Pivot corpus (pc)		Telephone conversation (tc)		Weblogs (wb)	
	# sent	# words	# sent	# words	# sent	# words	# sent	# words	# sent	# words	# sent	# words	# sent	# words
Train	11846	173289	10658	206902	6905	164217	34944	878223	21520	297049	11274	90403	16734	388851
Dev	2112	29957	1292	25271	641	15421	5893	147955	1780	25206	1367	11200	2297	49393
Test	2206	35947	1357	26424	779	17874	2326	60756	1869	25883	1306	10916	2281	52225

Table 1: Number of sentences and words for the splits of each domain in the OntoNotes 5.0 dataset.

dataset provides data annotated for an array of tasks across different languages and domains. We present experiments with the English portions of datasets, for which we show statistics in Table 1.[2]

Tasks In multi-task learning, one task is usually considered the main task, while other tasks are used as auxiliary tasks to improve performance on the main task. As main tasks, we use chunking (CHUNK), named entity recognition (NER), and a simplified version of semantic role labeling (SRL) where we only identify headwords, and pair them with part-of-speech tagging (POS) as an auxiliary task, following [25]. Example annotations for each task can be found in Table 2.

Model We use a state-of-the-art BiLSTM-based sequence labeling model [23] as the building block of our model. The BiLSTM consists of 3 layers with a hidden dimension of 100. At every time step, the model receives as input the concatenation between the 64-dimensional embedding of a word and its character-level embedding produced by a Bi-LSTM over 100-dimensional character embeddings. Both word and character embeddings are randomly initialized. The output layer is an MLP with a dimensionality of 100. We initialize α parameters with a bias towards one source subspace for each direction and initialize β parameters with a bias towards the last layer.

WORDS	Abramov	had	a	car	accident
CHUNK	O	B-VP	B-NP	I-NP	I-NP
NER	B-PERSON	O	O	O	O
SRL	B-ARG0	B-V	B-ARG1	I-ARG1	I-ARG1
POS	NNP	VBD	DT	NN	NN

Table 2: Example annotations for CHUNK, NER, SRL, and POS.

Training and Evaluation We train our models with SGD, an initial learning rate of 0.1, and learning rate decay. During training, we randomly sample from the data for each task. We perform early stopping with patience of 2 and hyperparameter optimization on the in-domain development data of the newswire domain. We use the same hyperparameters for all comparison models across all domains. We train our models on each domain and evaluate them both on the in-domain test set as well as on the test sets of all other domains to evaluate their out-of-domain generalization ability.

Baseline Models As baselines, we compare against i) a single-task model only trained on chunking; ii) the low supervision model by [25], which predicts the auxiliary task at the first layer; iii) an MTL model based on hard parameter sharing [6]; and iv) cross-stitch networks [21]. We compare these against our complete sluice network with subspace constraints and learned α and β parameters. We provide a detailed ablation analysis of our model in Section 5.

4.2 Model Comparison

We first investigate how well sluice networks perform on in-domain and out-of-domain test data compared to state-of-the-art multi-task learning models. To this end, we first evaluate all models on chunking with POS tagging as auxiliary task.

Results We show results on in-domain and out-of-domain tests sets in Table 3. On average, sluice networks significantly outperform all other model architectures on both in-domain and out-of-domain data. Single task models and hard parameter sharing achieve the lowest results, almost on par, and are outperformed by low supervision models and cross-stitch networks. Sluice networks perform best for all domains, except for the telephone conversation (tc) domain, where they are outperformed by cross-stitch networks.

In total, this shows that our proposed model for learning which parts of multi-task models to share, with a small set of additional parameters to learn, can achieve significant and consistent improvements in results.

[2]Note that not all sentences are annotated with all tasks.

	In-domain results								
	System	bc	bn	mz	nw	pt	tc	wb	Avg
Baselines	Single task	90.80	92.20	91.97	92.76	97.13	89.84	92.95	92.52
	Hard parameter sharing	90.31	91.73	92.33	92.22	96.40	90.59	92.84	92.35
	Low supervision	90.95	91.70	92.37	93.40	96.87	90.93	93.82	92.86
	Cross-stitch network	91.40	92.49	92.59	93.52	96.99	**91.47**	94.00	93.21
Ours	Sluice network	**91.72**	**92.90**	**92.90**	**94.25**	**97.17**	90.99	**94.40**	**93.48**
	Out-of-domain results								
Baselines	Single task	85.95	87.73	86.81	84.29	90.91	84.55	73.36	84.80
	Hard parameter sharing	86.31	87.73	86.96	84.99	90.76	84.48	73.56	84.97
	Low supervision	86.53	88.39	87.15	85.02	90.19	84.48	73.24	85.00
	Cross-stitch network	87.13	88.40	87.67	85.37	91.65	**85.51**	73.97	85.67
Ours	Sluice network	**87.95**	**88.95**	**88.22**	**86.23**	**91.87**	85.32	**74.48**	**86.15**

Table 3: Accuracy scores on in-domain and out-of-domain test sets for chunking (main task) with POS tagging as auxiliary task for different target domains for baselines and Sluice networks. Out-of-domain results for each target domain are averages across the 6 remaining source domains. Average error reduction over single-task performance is 12.8% for in-domain; 8.9% for out-of-domain. In-domain error reduction over hard parameter sharing is 14.8%.

	Named entity recognition								
	System	nw (ID)	bc	bn	mz	pt	tc	wb	OOD Avg
Baselines	Single task	95.04	93.42	93.81	93.25	94.29	94.27	92.52	93.59
	Hard parameter sharing	94.16	91.36	93.18	93.37	**95.17**	93.23	**92.99**	93.22
	Low supervision	94.94	91.97	93.69	92.83	94.26	93.51	92.51	93.13
	Cross-stitch network	95.09	92.39	93.79	93.05	94.14	93.60	92.59	93.26
Ours	Sluice network	**95.52**	**93.50**	**94.16**	**93.49**	93.61	**94.33**	92.48	**93.60**
	Semantic role labeling								
Baselines	Single task	97.41	**95.67**	95.24	95.86	95.28	98.27	97.82	96.36
	Hard parameter sharing	97.09	95.50	95.00	95.77	**95.57**	98.46	97.64	96.32
	Low supervision	97.26	95.57	95.09	95.89	95.50	98.68	97.79	96.42
	Cross-stitch network	97.32	95.44	95.14	95.82	**95.57**	**98.69**	97.67	96.39
Ours	Sluice network	**97.67**	95.64	**95.30**	**96.12**	95.07	98.61	**98.01**	**96.49**

Table 4: Test accuracy scores for different target domains with nw as source domain for named entity recognition (main task) and simplified semantic role labeling with POS tagging as auxiliary task for baselines and Sluice networks. ID: in-domain. OOD: out-of-domain.

4.3 Performance across Tasks

We now compare sluice nets across different combinations of main and auxiliary tasks. In particular, we evaluate them on NER with POS tagging as auxiliary task and simplified semantic role labeling with POS tagging as auxiliary task. We show results in Table 4. Sluice networks outperform the comparison models for both tasks on in-domain test data as well as on out-of-domain test data on average. They yield the best performance on 5 out of 7 domains and 4 out of 7 domains for NER and semantic role labeling respectively.

5 Analysis

Task Properties and Performance [4] correlate meta-characteristics of task pairs and gains from hard parameter sharing across a large set of NLP task pairs. Inspired by this study, we correlate various meta-characteristics with error reductions and α, β values in sluice networks, as well as in hard parameter sharing. Most importantly, we find that a) multi-task learning gains, also in sluice networks, are higher when there is less training data, and b) sluice networks learn to share more when there is more variance in the training data

Task sharing	Layer sharing	bc	bn	mz	nw	pt	tc	wb	Avg
constant α (hard)	Concatenation	86.70	88.24	87.20	85.19	90.64	85.33	73.75	85.29
	Skip-connections ($\beta = 1$)	86.65	88.10	86.82	84.91	90.92	84.89	73.62	85.13
	Mixture (learned β)	86.59	88.03	87.19	85.12	90.99	84.90	73.48	85.19
learned α (soft)	Concatenation	87.37	88.94	87.99	86.02	**91.96**	**85.83**	74.28	86.05
	Skip-connections	87.08	88.62	87.74	85.77	91.92	85.81	74.04	85.85
	Mixture	87.10	88.61	87.71	85.44	91.61	85.55	74.09	85.73
	Mixture + subspaces	**87.95**	**88.95**	**88.22**	**86.23**	91.87	85.32	**74.48**	**86.15**

Table 5: Ablation analysis. Accuracy scores on out-of-domain (OOD) test sets for Chunking (main task) with POS tagging as auxiliary task for different target domains for different configurations of sluice networks. OOD scores for each target domain are averaged across the 6 remaining source domains.

(cross-task αs are higher, intra-task αs lower). Generally, α values at the inner layers correlate more highly with meta-characteristics than α values at the outer layers.

Ablation Analysis Different types of sharing may empirically be more important than others. In order to investigate this, we perform an ablation analysis (Table 5). We investigate the impact of i) the α parameters; ii) the β parameters; and iii) the division into subspaces with an orthogonality penalty. We also evaluate whether concatenation of the outputs of each layer is a reasonable alternative to our mixture model.

Overall, we find that learnable α parameters are preferable over constant α parameters. Learned β parameters marginally outperform skip-connections in the hard parameter sharing setting, while skip-connections are competitive with learned β values in the learned α setting. In addition, modeling subspaces explicitly helps for almost all domains. Finally, concatenation of layer outputs is a viable form to share information across layers.

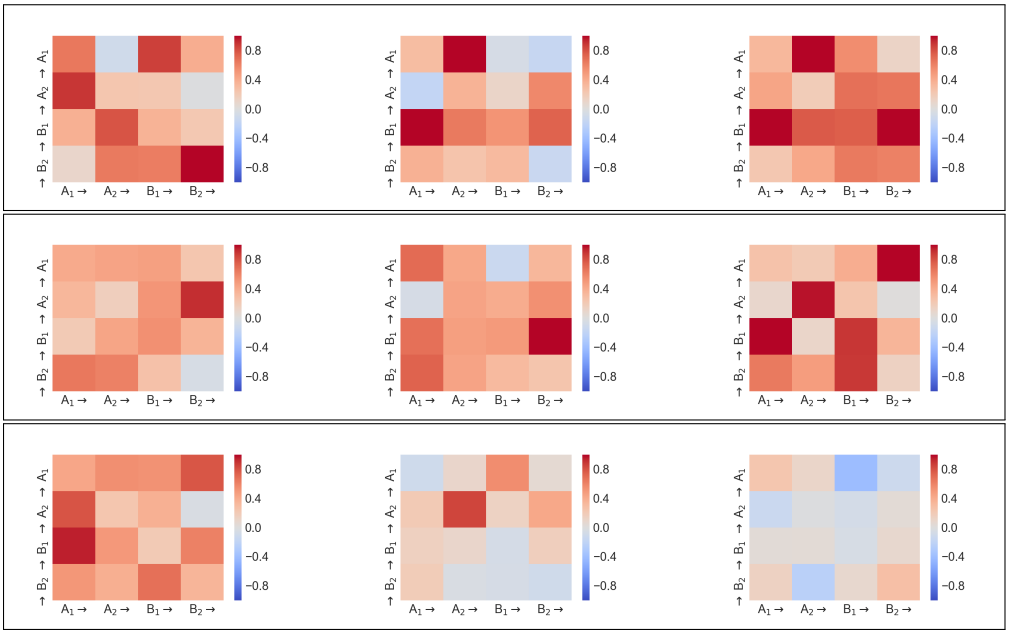

Figure 2: Heat maps of learned α parameters in trained sluice networks across (top to bottom): Chunking, NER, and SRL. We present inner, middle, and outer layer left to right.

Analysis of α values Figure 2 presents the final α weights in the sluice networks for Chunking, NER, and SRL, trained with newswire as training data. We see that a) for the low-level simplified SRL, there is more sharing at inner layers, which is in line with [25], while Chunking and NER also rely on the outer layer, and b) more information is shared from the more complex target tasks than vice versa.

Ability to Fit Noise Sluice networks can learn to disregard sharing completely, so we expect them to be as good as single-task networks to fit random noise, potentially even better. We verify this by computing a learning

curve for random relabelings of 200 sentences annotated with syntactic chunking brackets, as well as 100 gold standard POS-annotated sentences. The figure in 3 shows that hard parameter sharing, while learning faster because of the smoother loss surface in multi-task learning, is a good regularizer, confirming the findings in [25], whereas the sluice network is even better at fitting noise than the single-task models. While ability to fit noise is not necessarily a problem [32], this means that it can be beneficial to add inductive bias to the regularizer, especially when working with small amounts of data.

6 Related Work

In the context of deep neural networks, multi-task learning is often done with *hard or soft parameter sharing* of hidden layers. Hard parameter sharing was introduced by [6]. There, all hidden layers are shared between tasks which are projected into output layers specific to different tasks. This approach to multi-task learning is easy to implement, reduces overfitting and is guaranteed to work for (certain types of) closely related tasks [2, 20].

[22] apply a variation of hard parameter sharing to multi-domain multi-task sequence tagging with a shared CRF layer and domain-specific projection layers. [30] also use hard parameter sharing to jointly learn different sequence-tagging tasks (NER, POS tagging, Chunking) across languages. They also use word and character embeddings and share character embeddings in their model. [19] explore a similar set-up, but sharing is limited to the initial layer. In all three papers, the amount of sharing between the networks is fixed in advance.

In soft parameter sharing, on the other hand, each task has separate parameters and separate hidden layers, as in our architecture, but the loss at the outer layer is regularized by the current distance between the models. In [10], for example, the loss is regularized by the L_2 distance between (selective parts of) the main and auxiliary models. Other regularization schemes used in multi-task learning include the ℓ_1/ℓ_2 group lasso [1] and the trace norm [17].

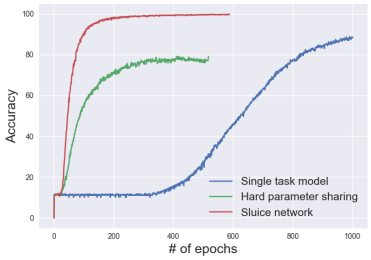

Figure 3: Random noise learning curves. Note that we only plot accuracies for hard parameter sharing and sluice networks until they plateau.

Selective Sharing Several authors have discussed which parts of the model to share. [25] perform experiments on which hidden layers to share in the context of hard parameter sharing with deep recurrent neural networks for sequence tagging. They show that low-level tasks, i.e. easy natural language processing tasks typically used for preprocessing such as part of speech tagging and named entity recognition, should be supervised at lower layers when used as auxiliary tasks.

Another line of work looks into separating the learned space into a private (i.e. task-specific) and shared space [31, 24, 27] to more explicitly capture the difference between task-specific and cross-task features. To enforce such behavior, constraints are enforced to prevent the models from duplicating information between subspaces. [5] use shared and private encoders regularized with orthogonality and similarity constraints for domain adaptation for computer vision. [18] use a similar technique for sentiment analysis.

In contrast to all the work mentioned above, we do not limit ourselves to a predefined way of sharing, but let the model learn which parts of the network to share using latent variables, the weights of which are learned in an end-to-end fashion.

The work most related to ours is [21], who also look into learning what to share in multi-task learning. However, they only consider a very small class of the architectures that are learnable in sluice networks. Specifically, they restrict themselves to learning *split architectures*. In such architectures, two n-layer networks share the innermost k layers with $0 \leq k \leq n$, and they learn k with a mechanism that is very similar to our α-values. Our work can be seen as a generalization of [21], including a more in-depth analysis of augmented works.

7 Conclusion

We introduced SLUICE NETWORKS, a framework for learning what to share in multi-task learning using trainable parameters. Our approach is a generalization of recent work, but goes well beyond this in enabling the network to learn selective sharing of layers, subspaces, and skip connections. In experiments with NLP task pairs in Ontonotes 5.0, we show up to 15% average error reduction over hard parameter sharing at only a 5–7% increase in training time. We provide an analysis of the ability of sluice networks to fit noise, as well as what properties are predictive of gains with sluice networks, seeing that the effect size correlates highly with label entropy, confirming previous findings for hard parameter sharing [19, 4].

Acknowledgments

This research was funded by the Irish Research Council Grant Number EBPPG/2014/30, Science Foundation Ireland Grant Number SFI/12/RC/2289, ERC Starting Grant LOWLANDS No. 313695, Elsevier, as well as by Trygfonden.

References

[1] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex Multi-Task Feature Learning. *Machine Learning*, 73(3):243–272, 2008.

[2] Jonathan Baxter. A Model of Inductive Bias Learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[3] Shai Ben-David and Reba Schuller. Exploiting Task Relatedness for Multiple Task Learning. In *Learning Theory and Kernel Machines*, pages 567–580. Springer, 2003.

[4] Joachim Bingel and Anders Søgaard. Identifying beneficial task relations for multi-task learning in deep neural networks. In *Proceedings of EACL*, 2017.

[5] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain Separation Networks. In *Proceedings of NIPS*, 2016.

[6] Rich Caruana. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Proceedings of ICML*, 1993.

[7] Rich Caruana. Multitask Learning. In *Learning to Learn*, pages 95–133. Springer, 1998.

[8] Ronan Collobert and Jason Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of ICML*, 2008.

[9] Hal Daumé III. Frustratingly Easy Domain Adaptation. In *Proceedings of ACL*, 2007.

[10] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. Low Resource Dependency Parsing: Cross-lingual Parameter Sharing in a Neural Network Parser. In *Proceedings of ACL*, 2015.

[11] Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. In *Proceedings of NIPS Continual Learning and Deep Networks Workshop*, 2016.

[12] 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.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. OntoNotes: The 90 % Solution. In *Proceedings of NAACL-HLT*, 2006.

[15] Laurent Jacob, Jean-Philippe Vert, Francis R Bach, and Jean-Philippe Vert. Clustered Multi-Task Learning: A Convex Formulation. In *Proceedings of NIPS*, pages 745–752, 2009.

[16] Ali Jalali, Sujay Sanghavi, Chao Ruan, and Pradeep K Ravikumar. A Dirty Model for Multi-task Learning. In *Proceedings of NIPS*, 2010.

[17] Shuiwang Ji and Jieping Ye. An Accelerated Gradient Method for Trace Norm Minimization. *Proceedings of ICML*, 2009.

[18] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Adversarial Multi-task Learning for Text Classification. In *Proceedings of ACL*, 2017.

[19] Héctor Martínez Alonso and Barbara Plank. When is multitask learning effective? Semantic sequence prediction under varying data conditions. In *Proceedings of EACL*, 2017.

[20] Andreas Maurer. Bounds for Linear Multi Task Learning. *Journal of Machine Learning Research*, 7:117–139, 2007.

[21] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-Stitch Networks for Multi-Task Learning. In *Proceedings of CVPR*, 2016.

[22] Nanyun Peng and Mark Dredze. Multi-task Multi-domain Representation Learning for Sequence Tagging. *CoRR*, abs/1608.02689, 2016.

[23] Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss. In *Proceedings of ACL*, 2016.

[24] Mathieu Salzmann, Carl Henrik Ek, Raquel Urtasun, and Trevor Darrell. Factorized Orthogonal Latent Spaces. *Journal of Machine Learning Research - Proceedings Track*, 9:701–708, 2010.

[25] Anders Søgaard and Yoav Goldberg. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of ACL*, 2016.

[26] Grzegorz Świrszcz and Aurélie C. Lozano. Multi-level Lasso for Sparse Multi-task Regression. In *Proceedings of ICML*, 2012.

[27] Seppo Virtanen, Arto Klami, and Samuel Kaski. Bayesian CCA via Group Sparsity. In *Proceedings of ICML*, 2011.

[28] Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, and Ann Houston. OntoNotes Release 5.0 LDC2013T19. *Linguistic Data Consortium*, 2013.

[29] Yongxin Yang and Timothy M. Hospedales. Trace Norm Regularised Deep Multi-Task Learning. In *Proceedings of ICLR - Workshop Track*, 2017.

[30] Zhilin Yang, Ruslan Salakhutdinov, and William Cohen. Multi-Task Cross-Lingual Sequence Tagging from Scratch. *CoRR*, abs/1603.06270, 2016.

[31] Jia Yangqing, Salzmann Mathieu, and Darrell Trevor. Factorized Latent Spaces with Structured Sparsity. In *Proceedings of NIPS*, 2010.

[32] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *Proceedings of ICLR*, 2017.

[33] Yang Zhou, Rong Jin, and Steven Hoi. Exclusive Lasso for Multi-task Feature Selection. In *Proceedings of AISTATS*, 2010.