Hyperparameter Tuning for Deep Learning in Natural Language Processing

Ahmad Aghaebrahimian
Zurich University of Applied Sciences
Switzerland
agha@zhaw.ch

Mark Cieliebak
Zurich University of Applied Sciences
Switzerland
ciel@zhaw.ch

Abstract

Deep Neural Networks have advanced rapidly over the past several years. However, it still seems like a black art for many people to make use of them efficiently. The reason for this complexity is that obtaining a consistent and outstanding result from a deep architecture requires optimizing many parameters known as hyperparameters. Hyperparameter tuning is an essential task in deep learning, which can make significant changes in network performance. This paper is the essence of over 3000 GPU hours on optimizing a network for a text classification task on a wide array of hyperparameters. We provide a list of hyperparameters to tune in addition to their tuning impact on the network performance. The hope is that such a listing will provide the interested researchers a mean to prioritize their efforts and to modify their deep architecture for getting the best performance with the least effort.

1 Introduction

The application of Deep Neural Networks (DNN) such as Convolution Neural Networks (CNN) (LeCun et al., 1989) or Recurrent Neural Networks (RNN) (Rumelhart et al., 1986) and its variants (e.g., Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) or Gated Recurrent Unit (GRU) (Cho et al., 2014)) has accelerated since the beginning of this decade partly due to the abundance of data available for training. Since past several years, DNNs have found their way in many areas of Artificial Intelligence (AI) such as image processing or Natural Language Processing (NLP) and have yielded superior performance in almost all of them. However, a DNN comes with a series of hyperparameters which need to be tuned if one expects to obtain state-of-the-art or even better results using them. Some of these hyperparameters, such as the number of layers or the number of neurons per layer, are bound directly to the deep neural architecture, while others - such as drop-out rate - are independent of the architecture. In addition to these hyperparameters, there are other network choices such as the classifier type that affects the network performance to a large extent. Our list of parameters to tune includes both of these hyperparameters and network choices. Since none of these parameters, including network choices and hyperparameters, can be learned within the network directly, from now on, we use the term hyperparameter to refer to both.

Recognizing the best choice of hyperparameters is often a cumbersome process to a level that some people consider it a "black art" (Snoek et al., 2012). Scarcity of proper research on the impact of these parameters on the network performance often leads to a waste of a lot of time, especially for younger researchers with little experience. In this paper, we adopt a state-of-the-art multi-label classifier to investigate the impact of 12 categories of hyperparameters on the task of multi-label text classification. The task in multi-label text classification is to assign one or more labels to each text.

Word embeddings types, word embeddings sizes, word embeddings updating, character embeddings, deep architectures (CNN, LSTM, GRU), optimizers, gradient control, classifiers, drop out, deep vs. wide networks, and pooling are the settings studied in this work. To make the experiment manageable, several groups of these parameters are set on an individual grid to serve as an ad-hoc grid search scheme for finding the most promising hyperparameters by focusing on the most promising optimized area.

We provide the readers with an insight into the

impact of each hyperparameter on this specific task. This study is performed by running over 400 different configurations in over 3000 GPU hours. The contribution of this work is to provide a prioritized list of hyperparameters to optimize.

2 Related Work

Hyperparameter tuning is often performed using grid search/brute force, where all possible combinations of the hyperparameters with all of their values form a grid and an algorithm is trained for each combination. However, this method becomes incomputable already for small numbers of hyperparameters. For instance, in our study with 12 categories of hyperparameters each with four instances on average, we would have a grid with several million nods, which would be highly computationally expensive. To address this issue Bergstra et al. (2013) proposed a method for randomized parameter tuning and showed that for each of their datasets there are only a few impactful parameters on which more values should be tried. However, due to the random mechanism in this approach, each trial is independent of the others. Hence, it does not learn anything from other experiments. To address this problem Snoek et al. (2012) proposed a Bayesian optimization method using a statistical model for mapping hyperparameters to an objective function. However, Bayesian optimization adds another layer of complexity to the problem. Therefore, this method has not gained much popularity since its proposal.

The most effective and straightforward method for hyperparameter tuning is still ad-hoc grid search (Hutter et al., 2015) where the researcher manually tries the most correlated parameters on the same grid to gradually and iteratively find the most impactful set of hyperparameters with the best values.

3 Multi-Label Classification

Multi-label text classification is the task of assigning one or more labels to each text. News classification is an example of such a task. For this task, we adopted a state-of-the-art architecture for multi-label classification (Aghaebrahimian and Cieliebak, 2019). The schema of the model is illustrated in Figure 1.

The architecture consists of two channels of bi-GRU deep structures with an attention mechanism and a dense sigmoid layer on the top. The illustrated schema is the optimized network which created the best results for the task. One channel is devoted to the most informative words given each class, which are extracted using the χ^2 method. The other channel is used for input tokens. For more information about the architecture, please refer to Aghaebrahimian and Cieliebak (2019).

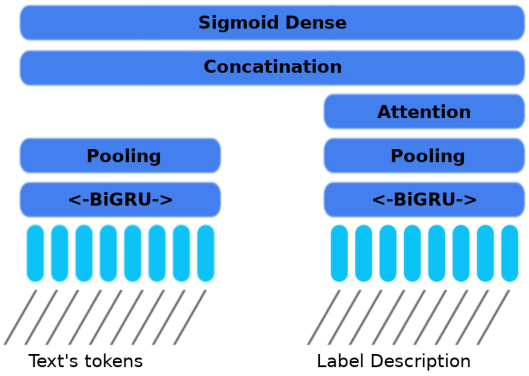

Figure 1: The system architecture

The dataset used for this experiment is a proprietary dataset with roughly 60K articles with a total number of 28 labels. The dataset contains about 250K different words and assigns 2.5 labels to each article on average. It is randomly divided into 80%,10%, and 10% parts for training, validating, and testing accordingly.

The textual data is preprocessed by removing non-alphanumeric values and replacing numeric values with a unique symbol. The resulting strings are tokenized and truncated to 3k tokens. Shorter texts are padded with 0 to fixate all the texts to the same length.

Two measures are used for evaluation. F1 (Micro) is used as a measure of performance. It is computed by calculating F1 scores for each article and averaging them over all articles in the test data. The second metric, Epochs, is reported as a measure of time required for the network with a specific setting to converge. The early stopping method is used as criterion for convergence, which is recognized when after three consecutive epochs no decrease in validation loss is observed. All models are trained in batches with 64 instances in each.

4 Experimental results

There are 12 categories of hyperparameters which are tuned in this study. Some of the hyperparam-

eters, such as the deep architecture or the classifier type, are network choices while others, such as the embeddings type or the dropout rate, are variables pertaining to different parts of the network. The results of hyperparameter optimization on each criterion are reported in the following subsections.

All parameters except the parameter under investigation in each experiment are kept constant. All other parameters that are not part of this study, such as the seed number or batch size, are also kept constant throughout all the experiments.

4.1 Word Embeddings Grid

In this grid, we tune the word embeddings type, the size, and the method of updating. Low dimensional dense word vectors known as word embeddings have been proven to be highly effective in representing words, and often lead to significantly better performance (Collobert et al., 2011). Depending on the method used for their training, they can provide different levels of syntactic and semantic information about each word. Many factors can affect the quality of word embeddings, including the data on which they were trained, their number of dimension, their domain, and preprocessing steps involved in the training. We investigated five widely studied pre-trained word embeddings including Word2Vec (Mikolov et al., 2013) trained on Google News dataset with 100 billion tokens, Glove (Pennington et al., 2014) with three variants (one trained on Wikipedia with 64 billion tokens and two others trained on the Common Crawl, one on 42 and the other on 840 billion tokens), FastText (Bojanowski et al., 2016), dependency_based (Levy and Goldberg, 2014), and ELMo (Peters et al., 2018). As shown in Table 1, the Glove embeddings trained on the Common Crawl yields significantly better results compared to other embeddings except for Elmo. Elmo and Glove-840 yield roughly similar results. However, due to the much larger word vector size in Elmo, it is much more computationally expensive and takes much longer time to converge.

Each pre-trained embedding comes with a specific vector size. The Glove embeddings are available in 50, 100, 200, and 300-dimensional word vectors. Elmo provides 1024 dimensional vectors, and other embeddings all are with 300-dimensional word vectors. The results for size tuning are reported in Table 2. Except for the 50-dimensional vectors, which is sub-optimal, all other dimensions yield superior results with an unnoticeable difference in the number of Epochs.

Word embedding type	Epochs	Results
Word2Vec (Mikolov et al., 2013)	26	81.9 %
Glove-6 (Pennington et al., 2014)	25	81.7 %
Glove-42 (Pennington et al., 2014)	26	82.9 %
Glove-840 (Pennington et al., 2014)	29	**84.5 %**
FastText (Bojanowski et al., 2016)	24	79.2 %
Dependency (Levy and Goldberg, 2014)	22	81.4 %
ELMo (Peters et al., 2018)	32	84.6 %

Table 1: Embedding type tuning results. Embedding types, sizes, and update methods are on the same grid (26 configurations).

Word embedding size	Epochs	Results
50	22	81.8 %
100	25	**82.9 %**
200	27	**83.6 %**
300	29	**84.3 %**
1024	32	84.6 %

Table 2: Embedding size tuning results. Embedding types, sizes, and update methods are on the same grid search (26 configurations).

Word embeddings provide a mean of transfer learning, which means word vectors are initially learned using a large dataset containing several billion tokens and are fine-tuned on a smaller dataset for doing their specific task afterwards. This mechanism can be controlled by having word vectors frozen or fine-tuned through training. Depending on the size of the dataset on which word embeddings are being refined, updating them can improve the performance. However, as observed in Table 3 fine-tuning the word vectors yielded no significant improvement over original pre-trained ones since the dataset was not large enough.

Word embedding updating	Epochs	Results
Disabled	29	84.3 %
Enabled	31	84.5 %

Table 3: Embedding update method tuning results. Embedding types, sizes, and update methods are on the same grid search (26 configurations).

4.2 Character embedding

Word-level features are not the only features used in text analytics. Character-level features are also reported to improve model performance especially in tasks such as Named Entity Recognition (NER) (Akbik et al., 2018) or Part Of Speech (POS) (Anastasiev et al., 2018) tagging, where knowing the function of individual characters such as prefixes, suffixes, or even infixes are beneficial.

We used two different character encoding mechanisms, one CNN-based (Ma and Hovy, 2016) and the other LSTM-based (Lample et al., 2016), to investigate the impact of character-level features on the network performance. As we expected, using character-level features had no added value in the label classification task where labels were bound to words and their syntactic and semantic attributes rather than to their characters.

Character embeddings and the best of embeddings grid were tuned on the same grid. It means that in this grid, we disregard the sub-optimal settings in the embeddings grid and only focus on the winning setting. Given the winning setting, we tune the character embedding settings to investigate the impact of character embeddings (Table 4).

Character embedding	Epochs	Results
Disabled	29	84.3 %
Enabled-CNN (Ma and Hovy, 2016)	31	84.7 %
Enabled-LSTM (Lample et al., 2016)	36	84.8 %

Table 4: Character embedding tuning results. Character embeddings and the best of embeddings are in the same grid search (14 configurations).

4.3 Deep architectures

The choice of deep architecture either as a Convolution Neural Network (CNN) (LeCun et al., 1989) or as a variant of Recurrent Neural Networks (RNN) such as Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) or Gated Recurrent Unit (GRU) (Cho et al., 2014) can have a huge effect on the performance of a model.

The deep architecture type, the number of deep layers, and the number of units in each layer, as well as the optimizers, are highly dependent on each other. Therefore, we optimize all of them on the same grid with 270 different configurations.

For the CNN model, we adapted Kim (2014) model, and for RNN models, we used both variants LSTM and GRU as single and bidirectional architectures. As seen in Table 5, although CNN models converge faster than the RNNs, they can not beat RNNs performance. Among all other RNN models, bidirectional GRU yields significantly better results.

Deep architectures	Epochs	Results
LSTM (Hochreiter and Schmidhuber, 1997)	30	78.2 %
Bi-LSTM	37	82.9 %
GRU (Cho et al., 2014)	21	79.8 %
Bi-GRU	29	**84.3 %**
CNN (single channel) (Kim, 2014)	18	81.7 %
CNN (double channel) (Kim, 2014)	23	82.5 %

Table 5: Deep architecture tuning results. Deep architectures, Deep and wide networks and optimizers are in the same grid (270 configurations).

4.4 Deep vs. wide networks

The application of more deep layers and more units in each layer has been beneficial in some tasks. Adding more layers helps in more complex tasks to generate more layers of abstraction, while adding more units to each layer contributes to generating more features. Still, adding extra layers in depth and width without enough training data usually leads to overfitting. In all of our configurations, we got the best performance by having 128 units for each layer and only one layer in depth (Table 6).

Deep vs. wide network	Epochs	Results
Deep-1	29	**84.3 %**
Deep-2	26	83.7 %
Deep-3	18	74.6 %
Wide-64	30	82.9 %
Wide-128	29	**84.3 %**
Wide-256	25	83.5 %

Table 6: Deep and wide networks tuning. Deep and wide networks, Deep architectures, and optimizers are in the same grid (270 configurations).

4.5 Optimizer

The job of an optimizer is to minimize the loss in the objective function. Gradient-based methods in general, and Stochastic Gradient Descent (SGD) in particular, are one of the widely used classes of optimizers for minimizing the objective functions in machine learning. Due to high sensitivity to learning rate in SGD, other variants of optimizers such as Adagrad (Duchi et al., 2011), RMSProp (Hinton, 2012), Adam (Kingma and Ba, 2015), and Nadam (Dozat, 2015) have been proposed in recent years. In all our configurations, we got the best performance using Adam. Nadam also yields almost the same performance while converging faster (Table 7).

4.6 Pooling

Either in a CNN after the convolutional filters or in an RNN after the recurrent layers, pooling has been proven as a useful tool for extracting the most

Optimizer	Epochs	Results
SGD	22	78.4 %
Adagrad (Duchi et al., 2011)	25	82.7 %
RMSProp (Hinton, 2012)	27	83.9 %
Adam (Kingma and Ba, 2015)	29	**84.3 %**
Nadam (Dozat, 2015)	24	**84.2 %**

Table 7: Optimizer tuning results. Optimizers, Deep and wide networks, and Deep architectures are in the same grid (270 configurations).

relevant features given each task. We investigated three types of polling, namely average, max and the concatenation of both with the best of optimizer configurations on the same grid with 15 settings. The results are reported in Table 8, which shows that using both yields the best performance for our task.

Pooling	Epochs	Results
Average	29	83.2 %
Max	29	83.5 %
Both	29	**84.2 %**

Table 8: Pooling tuning results. Pooling and the best of optimizers are in the same grid search (15 configurations).

4.7 Gradient control

The derivatives which are computed in backpropagation at training time in a DNN with many layers get smaller and smaller to the point of vanishing. This is particularly true for RNN's which have a large number of layers. This makes the training difficult and time-consuming. There are two widely practiced mechanism called gradient clipping (Mikolov, 2012) and gradient normalization (Pascanu et al., 2013) to address this issue known as gradient vanishing. We set the gradient control mechanism with the best of the deep architectures from Sub-section 4.3 on the same grid with 18 configurations. In all of these configurations, we got better results using gradient normalization (Table 9).

Gradient control	Epochs	Results
Disabled	28	82.9 %
Clipping (Mikolov, 2012)	31	83.1 %
Normalization (Pascanu et al., 2013)	29	**84.2 %**

Table 9: Gradient control tuning results. Gradient control and the best of deep architectures are in the same grid search (18 configurations).

4.8 Classifier

The last layer in a classification model is considered as the most crucial layer since all the computed features in this layer are projected to their appropriate classes. Therefore the choice of this layer has an essential impact on the network performance. The choice of this layer is highly dependent on the assumptions we make about the task at hand. If the labels are independently distributed, the Sigmoid and the Softmax yield better results, while if they are conditioned on their adjacent labels (e.g., POS tagging) the Conditional Random Field (CRF) (Lafferty et al., 2001) works better. If we expect a multinomial distribution over the labels, the Softmax is the best classifier to choose while if we expect a Bernoulli distribution, the Sigmoid is the right choice. All of the facts mentioned here come from the assumptions behind each of these statistical functions.

We investigated the performance of these three classifiers with the best of the deep architectures from Sub-section 4.3 on the same grid with 18 configurations. As observed in the results presented in Table 10, the Sigmoid obtains statistically significant better result compared to two other functions. As expected, due to the independence among the labels of different samples, the CRF did not perform very well. Likewise, duo to the freedom among labels in each sample, the Softmax also performed poorly.

Classifier	Epochs	Results
Softmax	30	78.4 %
Sigmoid	29	**84.2 %**
CRF	31	77.1 %

Table 10: Classifier tuning results. The classifiers and the best of deep architectures are in the same grid search (18 configurations).

4.9 Drop out value

Deep neural networks tend to memorize or overfit, which is not a desirable behavior since we are mostly interested in the ability of the network to generalize. Drop out (Srivastava et al., 2014) is an effective tool to enhance generalizability. The first technique known as simple or naive drop out was proposed as a mechanism which randomly removes the connections between deep layers. Gal and Ghahramani (2016) proposed a new mechanism for drop out called variational, which improves the simple drop out by defining static masks for removing the connections between deep layers ('interlayer') as well as between the units inside deep layers ('intralayer'). We placed drop out methods with the best of the deep architec-

tures from Sub-section 4.3 on the same grid with 90 configurations. The results are reported in Table 11 and Table 12. As expected, the configurations with both inter- and intralayer variational method yields the best performance.

Drop out value	Epochs	Results
Disabled	24	80.2 %
Simple 0.2	26	83.2 %
Simple 0.5	27	83.8 %
Simple 0.7	29	81.5 %
Variational	32	**84.2 %**

Table 11: Simple drop out tuning results. The drop out and the best of deep architectures are in the same grid search (90 configurations).

Variational drop out method	Epochs	Results
Inter	31	83.5 %
Intra	30	83.2 %
Both	32	**84.2 %**

Table 12: Variational drop out value tuning results

5 Conclusion

In this study, we investigated various settings for a Deep Neural Network for multi-label classification. Considering the characteristics of the dataset and the task, we observed the following results: Using Sigmoid in the last layer yields statistically significant better results compared to CRF or Softmax. The Glove embedings (Pennington et al., 2014) with more than 100-dimensional vector size and without updating yields statistically significant better results compared to other word vectors. Compared to other deep architectures, biGRU yields better results when it is used as a one-depth layer with 128 units. Adam and Nadam obtain roughly the same results, while Nadam converges much faster. Pooling is better to be used as the concatenation of both max and average-pooled tensors, and it is better to use Normalization (Pascanu et al., 2013) as a mean of gradient control to control gradient vanishing. It is also a good practice to use Variational drop out (Gal and Ghahramani, 2016) both between layers and inside recurrent units to control over-fitting. Finally, we did not observe any improvement by using character embeddings.

The order in which these parameters are mentioned is the magnitude of their importance for the final performance. Parameters with no mention here did not have any noticeable impact on the system results.

References

Ahmad Aghaebrahimian and Mark Cieliebak. 2019. Towards integration of statistical hypothesis tests into deep neural networks. In *Proceedings of the 57th annual meeting of the association of Computational Linguistics (ACL)*. Florence, Italy.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA.

D. G. Anastasiev, I. O. Gusev, and E. M. Indenbom. 2018. Improving part-of-speech tagging via multitask learning and character-level word representations. In *Proceedings of the International Conference Dialogue, Computational linguistics and intellectual technologies*.

James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*. Atlanta, GA, USA.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint* arXiv:1607.04606.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of the Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.* 12:2493–2537.

Timothy Dozat. 2015. Incorporating nesterov momentum into adam.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* 12:2121–2159.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Curran Associates Inc., USA, NIPS'16.

Geoffrey Hinton. 2012. Neural networks for machine learning. *Lecture 6a - Overview of mini-batch gradient descent* .

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. In *Neural Comput.*. volume 9.

Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. 2015. Beyond manual tuning of hyperparameters. *KI - Künstliche Intelligenz* .

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California.

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

Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 1064–1074.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv:1301.3781* .

Tom Mikolov. 2012. Statistical language models based on neural networks. *Ph.D. Thesis, Brno University of Technology* .

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning (ICML)*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning Representations by Back-propagating Errors. *Nature* 323.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS)*. USA.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from over fitting. *Journal of Machine Learning Research* .