Homework 2 Solution

Fall 2019

Exercises

1. a) Regularization in machine learning refers to strategies that are explicitly designed to reduce test error (which sometimes requires increasing the training error). Goodfellow et. al. formally define regularization in Chapter 7 of the text as "any modication we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

   b) Regularization is needed in machine learning to help avoid overfitting. This is typically achieved through placing extra constraints and penalties on model parameters. If applied correctly, regularization can lead to improved performance on the testing set by improving the generalization capabilities learned during training.

   c) The primary regularization techniques discussed in the text are L^2 and L^1 regularization, data augmentation, noise injection, multitask learning, early stopping, parameter sharing, sparse representations, bagging, dropout, and adversarial training. Other techniques include feature selection, and direct constraints on a complexity measure of the hypothesis space. Any other technique provided in a solution is also acceptable as long as it is not mentioned in the chapter.

2. a) In every step, each weight is shrunk by a multiplicative factor of $1-\epsilon\alpha$ before applying the normal gradient descent update. To understand the impact on the obtained solution, consider a quadratic approximation of $J(\cdot)$ in the neighborhood of $\boldsymbol{w}^* = \arg\min_w J(\boldsymbol{w})$ where $\boldsymbol{w}^*$ gives the optimal (minimal) unregularized training cost (i.e., we use a second order Taylor series approximation of $J(\cdot)$ about $\boldsymbol{w}^*$). Note that in case of fitting a linear regression model with the mean squared error, the second order approximation is perfect. The quadratic approximation of $J(\cdot)$ is given by:

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^*)^T\boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*), \tag{1}$$

where, $\boldsymbol{H}$ is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$. Note that there is no first-order term because the gradient is 0 at $\boldsymbol{w}^*$. Similarly, because $\boldsymbol{w}^*$ is the minimum of J, $\boldsymbol{H}$ is positive semidefinite.
The minimum of $\tilde{J}$ can be found by setting its gradient, with respect to $\boldsymbol{w}$, equal to 0:

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*) = 0. \tag{2}$$

Since we are interested in the effect of the regularization on the weights during training, we can add the weight decay gradient to the gradient of $\tilde{J}$ and solve for the parameters, $\boldsymbol{w}$, that will result in a minimum of of the regularized cost, $\tilde{J}$, which we will denote $\tilde{\boldsymbol{w}}$:

$$\alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}}-\boldsymbol{w}^*) = 0 \tag{3}$$

$$(\boldsymbol{H}+\alpha\boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^* \tag{4}$$

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1} \boldsymbol{H} \boldsymbol{w}^* \tag{5}$$

From (5), it is clear that as $\alpha \to 0$, $\tilde{\boldsymbol{w}} \to \boldsymbol{w}^*$. Now, let us decompose $\boldsymbol{H}$ into a diagonal matrix $\boldsymbol{\Lambda}$, and an orthonormal basis of eigenvectors, $\mathbf{Q}$, such that $\boldsymbol{H} = \boldsymbol{Q \Lambda Q^{-1}}$ so that we can observe the effects of $\tilde{\boldsymbol{w}}$ as α grows.

$$\tilde{\boldsymbol{w}} = (\boldsymbol{Q \Lambda Q^T} + \alpha \boldsymbol{I})^{-1} \boldsymbol{Q \Lambda Q^T} \boldsymbol{w}^* \tag{6}$$

$$\tilde{\boldsymbol{w}} = [\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})\boldsymbol{Q}^T]^{-1} \boldsymbol{Q \Lambda Q^T} \boldsymbol{w}^* \tag{7}$$

$$\tilde{\boldsymbol{w}} = \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda} \boldsymbol{Q}^T \boldsymbol{w}^* \tag{8}$$

Note that $\tilde{w}$ is obtained by rescaling w^* along the axes defined by the eigenvectors of $\mathbf{H}$, and the component of $\boldsymbol{w}^*$ that is aligned with the i^{th} eigenvector of $\mathbf{H}$ is rescaled by a factor of $\frac{\lambda_i}{\lambda_i + \alpha}$. When $\lambda_i >> \alpha$, the effect of the regularization is small whereas components with $\lambda_i << \alpha$ will shrink to have nearly zero magnitude.

b) From the above approximation, one can see that $\tilde{w}$ is obtained through an inverse of H multiplied by a projection of w^*. Hence, if H is ill-conditioned - with a large condition number - then L^2 regularization can amplify imperfections in the optimization strategy (i.e., if our estimate of w^* has errors). This is intuitive, because an ill-conditioned Hessian will imply that L^2 regularization will aggressively shrink certain weight components, which makes the final result more sensitive to errors in remaining components.

3. a) The L^1 regularization on $\boldsymbol{w}$ is defined as

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|. \tag{9}$$

Therefore, the regularized L^1 objective function, $\tilde{J}(\cdot)$, can be represented in terms of the unregularized objective function, $J(\cdot)$, as follows:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha ||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}), \tag{10}$$

Since we are assuming no correlation between the input features, we can simplify the presentation by saying that the Hessian, $\boldsymbol{H}$, is diagonal. Finally, the quadratic approximation of the L^1 regularized objective function is given by

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w^*}; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i} (w_i - w_i^*)^2 + \alpha |w_i| \right]. \tag{11}$$

Now, we need to choose w_i to minimize

$$\left[\frac{1}{2} H_{i,i} (w_i - w_i^*)^2 + \alpha |w_i| \right], \tag{12}$$

which we can do by taking the derivative of $\tilde{J}(\cdot)$ w.r.t. w_i, setting the result equal to 0, and solving for w_i. This results in

$$w_i = sign(w_i^*) max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}. \tag{13}$$

There are two possibilities:

i. $|w_i^*| \leq \frac{\alpha}{H_{i,i}}$: This results in $w_i = 0$ because the contribution of $J(\cdot)$ to $\tilde{J}(\cdot)$ is overwhelmed by the L^1 regularization.

ii. $|w_i^*| > \frac{\alpha}{H_{i,i}}$: This causes the regularization to decrease the magnitude of w_i by $\frac{\alpha}{H_{i,i}}$.

b) L^1 regularization results in a sparse solution (i.e., some parameters have optimal values of zero), which suggests that certain input features may safely be discarded. Feature selection, in machine learning, refers to choosing a consistent subset of samples/components for each input into the model. Therefore, since L^1 regularization results in sparse model parameters, only a subset of input features are forward propagated because samples that are multiplied by 0 do not contribute to model learning and are hence treated as discarded samples.

4. a) Norm penalties, which typically drive parameter values towards the origin, can result in very small parameter values (i.e., $\boldsymbol{w} \approx 0$), causing nonconvex optimization procedures to get stuck in local minima. In a neural network, this typically produces 'dead units,' which contribute very little, or nothing, to the function learned by the neural network because their values are always very small (or inactive). When training with a norm penalty of the weights, the very small weight values may be locally optimal even if the cost can be significantly reduced by increasing the weights.

b) Explicit constraints implemented by reprojection can work much better, in terms of avoiding dead units and getting stuck in local optima, because they do not drive the model parameters towards the origin. Furthermore, explicit constraints implemented by reprojection only affect training when the weights become large and attempt to leave the constraint region. Thus, when high learning rates are used, the network can encounter a positive feedback loop in which large weights result in large gradients, which result in large updates to the weights (due to the high learning rate). As a result of this positive feedback loop, the model weights rapidly move away from the origin. Explicit constraints with reprojection prevent the feedback loop from increasing without bound, yet allows it to explore the favorable region with relatively large weights and small cost.

c) Constraining the norm of each column (which corresponds to weights connected to a single unit) separately prevents any one hidden unit from having very large weights. This is useful for regularization because it avoids relying on only a small subset of hidden units for learning (similar role to dropout).

5. a) In machine learning, we will often want to find the inverse of a matrix. However, the inverse of that matrix may not necessarily exist. For example, when doing linear regression or PCA, if $\boldsymbol{X}^T\boldsymbol{X}$ is singular, which can occur when there are fewer training samples than the number of features per sample, then $\boldsymbol{X}^T\boldsymbol{X}$ is singular and hence does not have an inverse. In this case, many regulariztion techniques are available that instead invert $\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I}$, which is a regularized matrix that can be guaranteed to be have an inverse. In general, when facing under-constrained or underdetermined problems, there can exist infinitely many solutions that are *consistent* with the training examples (in other words, fit them perfectly). In such situation, Occam's razor states that we should choose the *simplest consistent hypothesis*. One could hence think of regularization in these contexts as imposing a definition of *complexity*, that lead the model to its simplest setting that fits the training data. For example, if we have norm penalties (like L_2 regularization), then settings with larger weight norms are considered to be more complex.

6. a) Dataset augmentation for image object recognition is used to exploit expert/domain knowledge for *synthetically* generating more training data in order to improve the network's generalization capabilities. These operations include translating each image by a few pixels in each direction, rotating the image, scaling the image, and injecting noise into each image.

b) Data augmentation techniques are not always appropriate for letters and numbers, because certain translations could change the class label of an image. For example, a horizontal flip of an image of a 'b' would turn the letter into a 'd.' Similarly, a 180° rotation of a '6' would change it to a '9.'

c) Let's denote the perturbed model (the model with the added noise in the weights) as $\hat{y}_{\epsilon_{\boldsymbol{w}}}(\boldsymbol{x})$. Although we have injected noise into $\boldsymbol{w}$, we are still interested in minimizing the squared error, between the true and predicted labels, of the network. Thus, our objective function remains identical to (4) from the problem statement, but with $\hat{y}_{\epsilon_{\boldsymbol{w}}}(\boldsymbol{x})$ substituted for $\hat{y}(x)$. This gives us

$$\hat{J} = \mathbb{E}_{p(x,y,\epsilon_{\boldsymbol{w}})}[(\hat{y}_{\epsilon_{\boldsymbol{w}}}(\boldsymbol{x}) - y)^2] \tag{14}$$

$$\hat{J} = \mathbb{E}_{p(x,y,\epsilon_{\boldsymbol{w}})}[\hat{y}^2_{\epsilon_{\boldsymbol{w}}} - 2y\hat{y}_{\epsilon_{\boldsymbol{w}}}(\boldsymbol{x}) + y^2] \tag{15}$$

Ignoring the effect of the nonlinearity, $\mathbb{E}_{p(x,y,\epsilon_{\boldsymbol{w}})}[2y\hat{y}_{\epsilon_{\boldsymbol{w}}}(\boldsymbol{x})]$ is equal to the corresponding term obtained without using regularization (because the mean of $\epsilon_{\boldsymbol{w}}$ is 0). Also, $\mathbb{E}_{p(x,y,\epsilon_{\boldsymbol{w}})}[y^2]$ does not depend on the prediction. Therefore, for small η (i.e., the magnitude of injected noise is small), the minimization of $\hat{J}$ is equivalent to minimizing J with an additional regularization term: $\eta\mathbb{E}_{p(\boldsymbol{x},y}[||\nabla_{\boldsymbol{w}}\hat{y}||^2]$. This form of regularization encourages the parameters to go to regions where small perturbations of the weight have a relatively small influence in the output. Thus, the effect of $\epsilon_{\boldsymbol{w}}$ is that it forces the model not only to local minima, but minima that are surrounded by flat regions.

d) Label smoothing refers to regularizing a model by flipping output labels with a certain probability (say ϵ). It encodes a prior belief that the provided labels in the training set may not be completely trust-worthy and that they are correct with probability $1-\epsilon$. For softmatx output units with k output units, it can be incorporated in the cost function (instead of flipping labels) by replacing the 0 or 1 value in the one-hot-encoded vectors with $\frac{\epsilon}{k-1}$ and $1-\epsilon$, respectively. In this case, it is said that the softmax-based classifier is probided *soft targets*. Label smoothing has the advantage of preventing the pursuit of hard probabilities without discouraging correct classifications.

7. a) The purely generative criterion, refers to learning the marginal distribution ($P(\boldsymbol{x})$) or even the joint distribution ($P(\boldsymbol{x}, y)$). In both cases, we are trying to learn the structure of an underlying data generating distribution from samples available in the training set. The discriminative criterion refers to learning a conditional distribution ($P(y|\boldsymbol{x})$) of class labels (or output values in case of regression) given an input data vector.

   b) i Semi-supervised learning refers to a scenario where both unsupervised learning (generative criterion) is performed based on available unlabeled data, and also supervised learning (discriminative criterion) is performed based on labeled data. It is usually the case that there is a larger number of unlabeled than labeled samples. The unsupervised learning can help identify the structure of the data generating distribution (for example, clusters) to facilitate the classifier training in presence of few labeled examples (for example, we need only one label per cluster).

   ii Suppose we are training a neural network tasked with predicting $\boldsymbol{y}^i$ given $\boldsymbol{x}$. In multitask learning, different supervised tasks share the same input, $\boldsymbol{x}$, as well as some intermediate level representation of the input. The lower layers of the neural network can be shared across such tasks, while task-specific parameters can be learned on top of those yielding a shared representation. In other words, multitask learning can help create an alternate representation of the input (hence generative training) while simultaneously performing a classification task (discriminative training). Even if the whole network is shared among different tasks, training it to perform the various tasks will push the network into learning the underlying structure of the data generating distribution, and hence can be considered to be implicitly trained with a generative criterion.

   iii Different models can be constructed in which a generative model learning $P(\boldsymbol{x})$ or $P(\boldsymbol{x}, y)$ shares parameters with a discriminative model learning $P(y|\boldsymbol{x})$. Further, this parameter sharing can extend to multiple discriminative models learning various tasks. Also, as illustrated in part ii), parameter sharing may only take place in earlier layers to learn common underlying patterns to the various tasks as well as the unsupervised learning model. This can enable a balance between the generative and discriminative criteria.

8. a) A validation set is a subset of data that is not initially used for training the model. Instead, it is used to track the generalization capabilities of the model during training, tune hyper-parameters, and choose the number of epochs. After using it, the validation set can be combined with the training set to further tune the model.

b) When a model has sufficient capabilities for overfitting a task (for example, when we have a very large number of parameters), we will often observe that the training set loss will decrease over time (epochs), but the validation set loss will initially decrease and then increase again. This results in a U-shaped curve for the validation loss. Ultimately, we are interested in using the parameter setting, that was used when the validation loss was at its minimum of the U-shaped curve, as the final model parameters. This is achieved through *early stopping.* Every time the validation error decreases from the minimum seen so far, we save a copy of the model parameters. This process terminates when the validation loss has not decreased for a pre-defined number of iterations. After training, we return to those parameters instead of the final values.

c) Choosing the number of epochs after combining the validation set with the training set is challenging because there is not a good way of knowing whether to retrain for the same number of parameter updates or the same number of passes through the dataset. On the second round of training, each pass through the dataset will require more parameter updates because the training set is bigger.

d) When we continue training, we no longer have a guide for when to stop in terms of a number of steps. Instead, we can monitor the average loss function on the validation set and continue training until it falls below the value of the training set loss at which the early stopping procedure stopped. This avoids re-training the model, which is an expensive decision. However, there is no guarantee for training termination in this case.

9. a) From the hint, and the solution to Problem 2 above, we know that we can begin by modeling the cost function of $J(\cdot)$ with a quadratic approximation in the neighborhood of the minimum of the unregularized cost, $\boldsymbol{w}^*$:

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*), \tag{16}$$

where $\boldsymbol{H}$ is the Hessian of J w.r.t. to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$. Furthermore, given the assumption that $\boldsymbol{w}^*$ is a minimum, we know that $\boldsymbol{H}$ is positive semidefinite. Under a Taylor Series approximation, the gradient is given by

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*). \tag{17}$$

Now, let's study the trajectory followed by $\boldsymbol{w}$ during training. We can observe the approximate behavior of gradient descent on J by analyzing gradient descent on $\tilde{J}$:

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}^{(\tau-1)}) \tag{18}$$

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{19}$$

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{20}$$

Using eigendecomposition, we know that $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\boldsymbol{T}}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues and $\boldsymbol{Q}$ is an orthonormal basis of eigenvectors.

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\boldsymbol{T}})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{21}$$

$$\boldsymbol{Q}^T(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*) = (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})\boldsymbol{Q}^T(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{22}$$

Assuming that $\boldsymbol{w}^{(0)} = 0$, the parameter trajectory during training after τ parameter updates is as follows:

$$\boldsymbol{Q}^T\boldsymbol{w}^{(\tau)} = [\boldsymbol{I} - (\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})^\tau]\boldsymbol{Q}^T\boldsymbol{w}^* \tag{23}$$

We also know that the expression for $\boldsymbol{Q}^T\tilde{\boldsymbol{w}}$ for L^2 regularization is given by:

$$\boldsymbol{Q}^T\tilde{\boldsymbol{w}} = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* \tag{24}$$

$$\boldsymbol{Q}^T\tilde{\boldsymbol{w}} = [\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha]\boldsymbol{Q}^T\boldsymbol{w}^* \tag{25}$$

If we choose hyperparameters ϵ, α, and τ such that

$$(\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})^\tau = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha, \tag{26}$$

Note that it has to be the case that ϵ is chosen to be small enough to guarantee $|1 - \epsilon\lambda_i| < 1$. We then have that $\boldsymbol{w}^\tau = \tilde{\boldsymbol{w}}$, and hence, L^2 regularization and early stopping can be seen as equivalent under this quadratic approximation of the objective function. Further, by taking the logarithm of each side in (26), and under the assumption that $\epsilon\lambda_i$ and $\frac{\lambda_i}{\alpha}$ are very small quantities for all eigenvalues, and then using the approximation $\log(1 + x) \approx x$, we obtain that,

$$\tau \approx \frac{1}{\epsilon\alpha}. \tag{27}$$

10. a) *Bagging* is typically carried out by training multiple models separately, each on an independently sampled subset of the training data (note that the subsets can overlap), and then having all the models vote on the output for testing samples, which should lead to reducing the generalization error. *Bagging* can even be useful if the entire training dataset is used for training each model because neural networks can reach a wide enough variety of solutions even if all the models are trained on the same dataset. Differences in random initialization, random selection of minibatches, hyperparameters, or outcomes of nondeterministic implementations like noise injection, are often enough to cause different members of the ensemble to make independent errors given identical training data.

    b) *Boosting* is applied by building ensembles of neural networks through incrementally adding neural networks to the ensemble. An individual neural network can be viewed as an ensemble, when incrementally adding hidden units to the network, as every snapshot of the network can be viewed as a member model of the ensemble.

Bonus

11. a) Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks. Specically, dropout trains the ensemble consisting of all subnetworks that can be formed by removing nonoutput units from an underlying base network. In most neural networks, we can remove a unit from a network by multiplying its output by 0. Dropout aims to replicate *bagging*, but with an exponentially large number of neural networks made up of subnetworks of the underlying model.

    b) Dropout training is not quite the same as bagging training. In the case of bagging, the models are all independent. In the case of dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network. This parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory. In the case of bagging, each model is trained to convergence on its respective training set. In the case of dropout, typically most models are not explicitly trained at all usually, the model is large enough that it would be infeasible to sample all possible subnetworks within the lifetime of the universe. Instead, a tiny fraction of the possible subnetworks are each trained for a single step, and the parameter sharing causes the remaining subnetworks to arrive at good settings of the parameters. These are the only differences. Beyond these, dropout follows the bagging algorithm. For example, the training set encountered by each subnetwork is indeed a subset of the original training set sampled with replacement.

12. a) The *weight scaling inference rule* is as follows: we can approximate the ensemble by the model having all units, but with the weights going out of unit i multiplied by the probability of including unit i. *Goodfellow et. al. (2013)* found experimentally that the weight scaling approximation to the ensemble performance can work better (in terms of the classication accuracy metric) than Monte Carlo approximations with 1000 realizations.

    b) i Dropout boosting trains the entire ensemble to jointly maximize the log-likelihood on the training set. In other words, at every training step, the parameters of a single realization, after applying the dropout mask, are updated according to the objective of minimizing the entire ensemble's loss function, and not only the loss function of the current realization as in normal dropout.

    ii DropConnect is a special case of dropout where each product between a single scalar weight and a single hidden unit state is considered a unit that can be dropped.

    iii Using non-sparse masks refers to dropout noise that does not have binary entries (no zeroes imply no sparsity). An example is using a mask vector μ that has a Gaussian distribution with mean $\mathbf{1}$ and covariance $\mathbf{I}$.

References

[1] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y., "Maxout Networks," in *International Conference on Machine Learning*, 2013.