Summary and discussion of: "Dropout Training as Adaptive Regularization"

Statistics Journal Club, 36-825

Kirstin Early and Calvin Murdock

November 21, 2014

1 Introduction

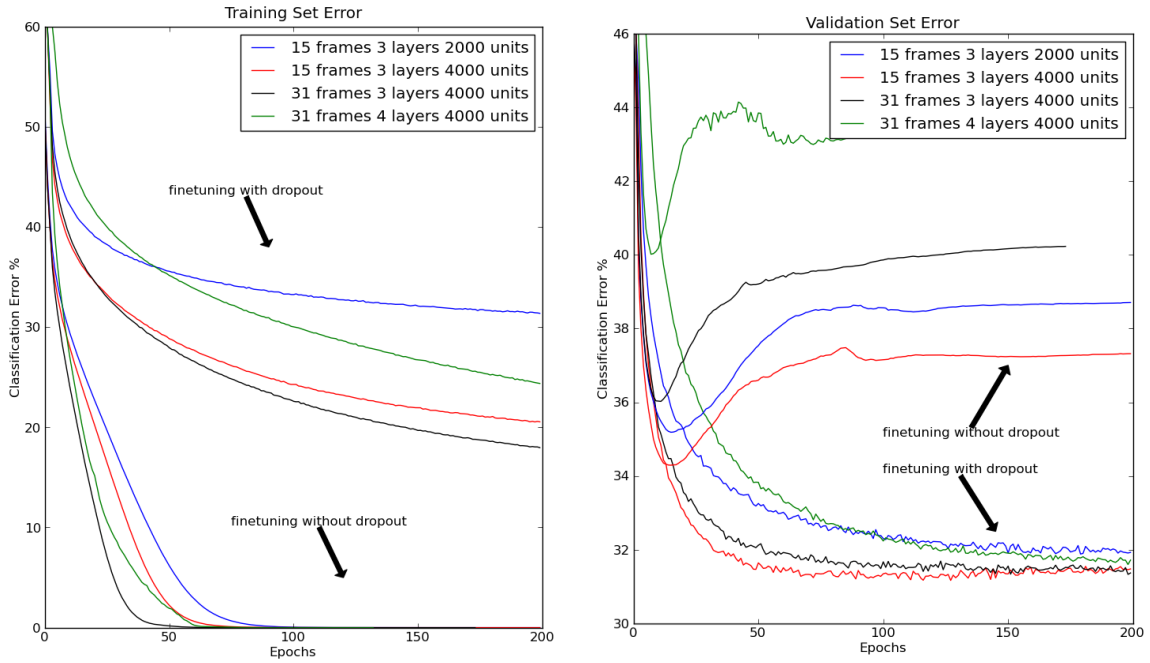
Multi-layered (i.e. *deep*) artificial neural networks have recently undergone a resurgence in popularity due to improved processing capabilities and the increasing availability of large datasets. Popular in the 1980's and before, they were largely abandoned in favor of convex methods (such as support vector machines) that came with optimality guarantees and often gave better results in far less time. However, with the modern ubiquity of GPUs and distributed computing, the same methods that were once spurned for their computational intractability have become the de-facto standard for large-scale commercial applications in companies such as Google, Facebook, Baidu, etc. This almost universal adoption of deep learning (as it's now called) is not without reason; variants of these methods have achieved state-of-the-art performance (often by a significant margin over other competing algorithms) on numerous tasks such as image classification, speech recognition, etc.

While improved computational power has allowed these models to be trained in a moderate amount of time, their performance is tied to the quantity and quality of the available training data. Deep neural networks are often very large, often consisting upwards of millions (or even billions) of parameters whose values must all be learnt from data. Thus, like any statistical model, they are susceptible to overfitting and typically require huge quantities of labeled training examples. Furthermore, the optimization landscape for the network parameters is highly non-convex with many local optima. So even with sufficient training data, the performance of a trained model could still be poor. These problems have been addressed empirically by a surprisingly simple method referred to as *dropout*. First introduced by Geoffrey Hinton, dropout has become an important component in most modern deep learning implementations because it results in reduced overfitting and often reaches better local minima. Even so, very little is understood theoretically about why this should be the case. The paper *Dropout Training as Adaptive Regularization* is one of several recent papers that attempts to understand the role of dropout in training deep neural networks.

1.1 A Motivating Example

To motivate the use of dropout in deep learning, we begin with an empirical example of its success originally given in [3]. Specifically, the authors considered the TIMIT dataset,

a benchmark for recognition of clean speech with a small vocabulary. This task was approached using a neural network with four fully-connected hidden layers of 4000 units per layer and 185 softmax output units for representing the 39 distinct output classes. Figure 1 summarizes the results of this experiment both with and without dropout. Because of the relatively small amount of available training data, the network demonstrates obvious overfitting resulting in almost perfect training accuracy but very poor testing performance. However, we can see that dropout appears to act as a type of regularizer and gives worse training performance but improved testing performance. Furthermore, we can see that the final convergence accuracy is better than what would have been achieved by stopping the training process early. This demonstrates that dropout could allow for finding better local minima. The resulting model achieved state-of-the-art recognition rates: 22.7% without dropout and 19.7% with dropout.



(a) Training

(b) Testing

Figure 1: Classification error rates for the TIMIT benchmamrk on both training (a) and testing (b) data. Note that without dropout, the network overfits to the training data resulting in almost perfect training performance but much worse testing accuracy.

2 Artificial Feature Noising as Regularization

Because neural networks are highly nonlinear, consisting of layered compositions of functions, they are difficult to analyze theoretically. Instead, we consider feature noising in Generalized Linear Models (GLMs), which can be considered single-layer neural networks.

A GLM is a parametric model for regression that explicitly defines the conditional

probability distribution of an output $y \in \mathcal{Y}$ given an input feature vector $x \in \mathbb{R}^d$:

$$p_\beta(y|x) = h(y) \exp\{yx \cdot \beta - A(x \cdot \beta)\} \tag{1}$$

Given an i.i.d. training set $\{x_i, y_i\}_{i=1}^n$, the model parameters β can be found by minimizing the empirical loss defined to be the negative log likelihood of the data:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \ell_{x_i,y_i}(\beta), \quad \ell_{x_i,y_i}(\beta) = -\log p_\beta(y_i|x_i) \tag{2}$$

Artificial feature noising replaces the observed feature vectors x_i with noisy copies $\tilde{x}_i = \nu(x_i, \xi_i)$ where ν is a function of the input x_i and an i.i.d. random variable ξ_i. We consider two types of noise:

- Additive Gaussian noise: $\nu(x_i, \xi_i) = x_i + \xi_i$, where ξ_i is a Gaussian random variable $\xi_i \sim \mathcal{N}(0, \sigma^2 I_d)$.
- Dropout noise: $\nu(x_i, \xi_i) = x_i \odot \xi_i$, where ξ_i is a vector of independent scaled Bernoulli$(1-\delta)$ random variables with values $\xi_{ij} \in \{0, (1-\delta)^{-1}\}$. This can be equivalently written as follows:

$$\tilde{x}_{ij} := \begin{cases} 0 & \text{w.p. } \delta \\ x_{ij}/(1-\delta) & \text{w.p. } 1-\delta \end{cases} \tag{3}$$

Note that the expectation of the augmented variable $\tilde{x}_i$ with respect to the artificial feature noise ξ is equal to the original feature vectors x_i, i.e. $\mathbb{E}_\xi[\tilde{x}_i] = \mathbb{E}[\tilde{x}_i|\{x_i, y_i\}] = x_i$. Thus, while simply replacing our data set with an augmented version $\{\tilde{x}_i, y_i\}_{i=1}^n$ would result in a loss of information, if we average over multiple copies with different instantiations of noise, then this noise will average out. If we consider averaging over m of these augmented copies of our data set (where $\tilde{x}_i^{(j)}$ denotes the augmented version of x_i from the jth copy, the optimization problem in Equation 2 becomes:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \frac{1}{m} \sum_{j=1}^m -\log p_\beta(y_i|\tilde{x}_i^{(j)}) \tag{4}$$

In the limit where m becomes large, we can replace the average with an expectation, which is effectively equivalent to integrating over all possible instantiations of feature noise:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^n \mathbb{E}_\xi[-\log p_\beta(y_i|\tilde{x}_i)] \tag{5}$$

For GLMs, the negative log likelihood can be simplified further:

$$\sum_{i=1}^n \mathbb{E}_\xi[-\log p_\beta(y_i|\tilde{x}_i)] = \sum_{i=1}^n -(y\mathbb{E}_\xi[\tilde{x}_i] \cdot \beta - \mathbb{E}_\xi[A(\tilde{x}_i \cdot \beta]) \tag{6}$$

$$= \sum_{i=1}^n -(yx_i \cdot \beta - \mathbb{E}_\xi[A(x_i \cdot \beta]) + R(\beta) \tag{7}$$

$$= \sum_{i=1}^n -\log p_\beta(y_i|x_i) + R(\beta) \tag{8}$$

This is equivalent to the negative log likelihood of the original data, with an additional term $R(\beta)$ defined to be

$$R(\beta) = \sum_{i=1}^{n} \mathbb{E}_{\xi}[A(\tilde{x}_i \cdot \beta)] - A(x_i \cdot \beta) \tag{9}$$

Since the log partition function A of a GLM is always convex, R must always be positive by Jensen's inequality. In addition, this term is independent of the training labels and can thus be interpreted as a regularizer akin to ridge regression (which penalizes the L_2 norm of β) or lasso (which penalizes the L_1 norm of β) that summarizes entirely the effects of feature noising.

2.1 A Quadratic Approximation to the Noising Penalty

While the regularizer in Equation 9 is exactly equivalent to feature noising in a GLM with an infinite number of noisy copies of the data, it is difficult to interpret and effectively impossible to implement in general. Thus, we approximate it by taking a second-order Taylor expansion of A around $x \cdot \beta$ to gain more insight. Specifically,

$$\mathbb{E}_{\xi}[A(\tilde{x} \cdot \beta)] - A(x \cdot \beta) \approx \frac{1}{2} A''(x \cdot \beta) \mathrm{Var}_{\xi}[\tilde{x} \cdot \beta] \tag{10}$$

since the first-order term $\mathbb{E}_{\xi}[A'(x \cdot \beta)(\tilde{x} - x)]$ vanishes because $\mathbb{E}_{\xi}[\tilde{x}] = x$. This allows Equation 9 to be approximated as follows:

$$R^q(\beta) := \frac{1}{2} \sum_{i=1}^{n} A''(x_i \cdot \beta) \mathrm{Var}_{\xi}[\tilde{x_i} \cdot \beta] \tag{11}$$

This can be interpreted as penalizing both the variance of the response y_i in the GLM $A''(x_i \cdot \beta)$ and the variance of the estimated parameters due to noising $\mathrm{Var}_{\xi}[\tilde{x_i} \cdot \beta]$. Empirically, this quadratic approximation has been found to be accurate for logistic regression, although it tends to overestimate the true penalty when $p \approx 0.5$ and underestimate it for very confident predictions. Thus, fitting a logistic regression model with this approximate regularization term results in very similar results to using actual dropout.

2.2 Regularization based on Additive Noise

This quadratic noising regularizer in Equation 11 is general and can be applied to any GLM. To give some intuition for the effects of feature noising on learning process, we first focus on Gaussian additive noise and two specific examples: linear regression and logistic regression.

If $\tilde{x} = x + \epsilon$ is generated by adding Gaussian noise with $\mathrm{Var}[\epsilon] = \sigma^2 I_d$. Then, $\mathrm{Var}[\tilde{x} \cdot \beta] = \sigma^2 \|\beta\|_2^2$.

2.2.1 Linear Regression

For linear regression, $A(z) = \frac{1}{2} z^2$ so $A''(z) = 1$. Plugging this information into Equation 11 results in the simplified noising penalty:

$$R^q(\beta) := \frac{1}{2} \sigma^2 n \|\beta\|_2^2 \tag{12}$$

Since the quadratic approximation is exact, this demonstrates the known relation that Gaussian feature noising is equivalent to ridge regression.

2.2.2 Linear Regression

For logistic regression, $A''(x_i \cdot \beta)p_i(1-p_i)$ where

$$p_i := 1/(1+\exp(-x_i^T\beta)) \tag{13}$$

is the predicted probability that $y_i = 1$. The resulting quadratic penalty is:

$$R^q(\beta) := \frac{1}{2}\sigma^2 \|\beta\|_2^2 \sum_{i=1}^n p_i(1-p_i) \tag{14}$$

Thus, the noising penalty not only encourages parsimony by encouraging $\|\beta\|_2^2$ to be small, but also confident predictions by encouraging p_i to be far from $\frac{1}{2}$.

2.3 Regularization Based on Dropout Noise

With dropout noise as defined in (3), then

$$\mathrm{Var}_\xi[\tilde{x_i} \cdot \beta] = \frac{\delta}{1-\delta}\sum_{j=1}^d x_{ij}^2\beta_j^2. \tag{15}$$

Then, substituting into (11), the quadratic approximation to the regularizer with dropout noise is

$$\begin{aligned} R^q(\beta) &= \frac{1}{2}\frac{\delta}{1-\delta}\sum_{i=1}^n A''(x_i\cdot\beta)\sum_{j=1}^d x_{ij}^2\beta_j^2 \\ &= \frac{1}{2}\frac{\delta}{1-\delta}\beta^T \mathrm{diag}(X^T V(\beta) X)\beta, \end{aligned} \tag{16}$$

where $X \in \mathbb{R}^{n\times d}$ is the design matrix and $V(\beta) \in \mathbb{R}^{n\times n}$ is a diagonal matrix with entries $A''(x_i \cdot \beta)$. If we define $\beta^\star$ to be the MLE as $n \to \infty$, then

$$\frac{1}{n}X^T V(\beta^\star)X = \frac{1}{n}\sum_{i=1}^n \nabla^2 \ell_{x_i,y_i}(\beta^\star) = \hat{\mathcal{I}}, \tag{17}$$

an estimate of the Fisher information $\mathcal{I}$. That is, the dropout regularizer is equivalent to applying an L_2 penalty after normalizing the feature vector by $\mathrm{diag}(\mathcal{I})^{-1/2}$—the L_2 penalty is applied in a basis where the features have been "balanced out."

For *linear regression*, where $V(\beta) = I$ (since $A(z) = \frac{1}{2}z^2 \implies A''(z) = 1$), the quadratic penalty term becomes

$$\begin{aligned} R^q(\beta) &= \frac{1}{2}\frac{\delta}{1-\delta}\beta^T \mathrm{diag}(X^T X)\beta \\ &= \frac{1}{2}\frac{\delta}{1-\delta}\sum_{j=1}^d \beta_j^2 x_j^T x_j, \end{aligned} \tag{18}$$

where $x_j := j$-th column of X. Thus, the dropout regularizer is equivalent to applying an L_2 penalty (ridge regression) on a column-normalized X.

For *logistic regression*, where $A''(x_i \cdot \beta) = p_i(1-p_i)$, the quadratically-approximated regularizer becomes

$$R^q(\beta) = \frac{1}{2}\frac{\delta}{1-\delta}\sum_{i=1}^{n}\sum_{j=1}^{d} p_i(1-p_i)x_{ij}^2\beta_j^2. \tag{19}$$

In this case, the interaction among $p_i(1-p_i), \beta_j^2$, and x_{ij}^2 allows for large $p_i(1-p_i)$ and β_j^2 terms as long as x_{ij}^2 is small. In particular, note that there is no penalty on β_j for which $x_{ij} = 0$. Therefore, dropout can learn feature weights for rare (i.e., $x_{ij} = 0$, often) yet highly-discriminative (i.e., β_j^2 is large) features, since it incurs no penalty in increasing β_j in this case.

2.4 Dropout Regularization in Online Learning

Stochastic gradient descent (SGD) updates the estimate for the weight vector β at the $t+1$-st training example according to $\hat{\beta}_{t+1} = \hat{\beta}_t - \eta_t g_t$, where $g_t := \nabla \ell_{x_t,y_t}\hat{\beta}_t)$ is the gradient of the loss at the t-th training example. This process is equivalent to solving an L_2-regularized linear problem at each step (linear approximation to loss + L_2 penalty on β):

$$\hat{\beta}_{t+1} = \arg\min_{\beta}\left\{\ell_{x_t,y_t}(\hat{\beta}) + g_t \cdot (\beta - \hat{\beta}_t) + \frac{1}{2\eta_t}||\beta - \hat{\beta}_t||_2^2\right\} \tag{20}$$

Traditional SGD has difficulty (is slow) when learning weights for rare, highly discriminative features (a setting in which dropout training can succeed), which motivated[2] to propose AdaGrad, which changes the SGD update rule to $\hat{\beta}_{t+1} = \hat{\beta}_t - \eta A_t^{-1} g_t$, where A_t is learned online; Duchi et al. use $A_t := \text{diag}(G_t)^{1/2}$, where $G_t := \sum_{i=1}^{t} g_i g_i^T$.

The authors propose changing the L_2 regularization term in (20) to their dropout regularizer:

$$\hat{\beta}_{t+1} = \arg\min_{\beta}\left\{\ell_{x_t,y_t}(\hat{\beta}) + g_t \cdot (\beta - \hat{\beta}_t) + R^q(\beta - \hat{\beta}_t; \hat{\beta}_t)\right\}, \tag{21}$$

where $R^q(\beta - \hat{\beta}_t; \hat{\beta}_t)$ is centered at $\hat{\beta}_t$:

$$R^q(\beta - \hat{\beta}_t; \hat{\beta}_t) := \frac{1}{2}(\beta - \hat{\beta}_t)^T \text{diag}(H_t)(\beta - \hat{\beta}_t), \tag{22}$$

where $H_t := \sum_{i=1}^{t} \nabla^2 \ell_{x_i,y_i}(\hat{\beta})$. Therefore, dropout descent is equivalent to adaptive SGD with $A_t = \text{diag}(H_t)$. For GLMs, G_t and H_t are both consistent estimates of the Fisher information, so both AdaGrad and dropout SGD achieve their similar performance improvement (particularly for learning weights for rare yet highly discriminative features) by scaling features by the Fisher information. However, AdaGrad has a more aggressive learning rate.

2.5 Semi-Supervised Dropout Training

Note that the regularizer defined in (9) does not depend on the labels y. Therefore, we can use *unlabeled* training examples to get a better estimate of $R(\beta)$. If we have an unlabeled dataset $\{z_i\}_{i=1}^m$, we define a semi-supervised penalty estimate by

$$R_\star(\beta) := \frac{n}{n+\alpha m}\left(R(\beta) + \alpha R_{\text{unlabeled}}(\beta)\right), \tag{23}$$

where $\alpha \in (0, 1]$ is the discount factor for how much to weight the unlabeled data and $R_{\text{unlabeled}}(\beta)$ indicates the value of $R(\beta)$ applied to the unlabeled dataset Z. The authors use this semi-supervised paradigm on text classification and including unlabeled data in the regularizer improved performance in all their reported experiments.

3 Simulations

3.1 Data

We generated data as described in Appendix A.1 of the paper to illustrate the effectiveness of dropout on learning rare, highly discriminative features. Data samples are 1050-dimensional, with 50 discriminative features (each of which is active only 4% of the time) and 1000 noise features. The discriminative features are drawn from an exponential distribution scaled such that the second moment is normalized; noise variables are generated from $\mathcal{N}(0, 1)$. Each label y_i is drawn from a Bernoulli distribution with parameter $\sigma(x_i \cdot \beta)$, where the first 50 coordinates of β are 0.2792 (chosen such that $\mathbb{E}\left[|x_i \cdot \beta|\right] = 2$ when there is a signal).

3.2 Experiments

We performed logistic regression on data simulated as described in 3.1, under several regularization conditions:

3.2.1 Maximum likelihood estimation, with no regularization

First, we compared with unregularized MLE estimation using the GLMNET package. The corresponding objective function is:

$$\min_\beta -\ell(p) \tag{24}$$

where $\ell(p) = \sum_{i=1}^{n} y_i \log p_i + (1 - y_i) \log(1 - p_i)$ and $p_i = 1/(1 + \exp(-x_i \cdot \beta))$.

3.2.2 MLE with L_2 regularization

We also performed MLE optimization with an L2 regularization term with the optimal λ parameter given in the original paper. The objective function is:

$$\min_\beta -\ell(p) + \frac{1}{2}\lambda||\beta||_2^2 \tag{25}$$

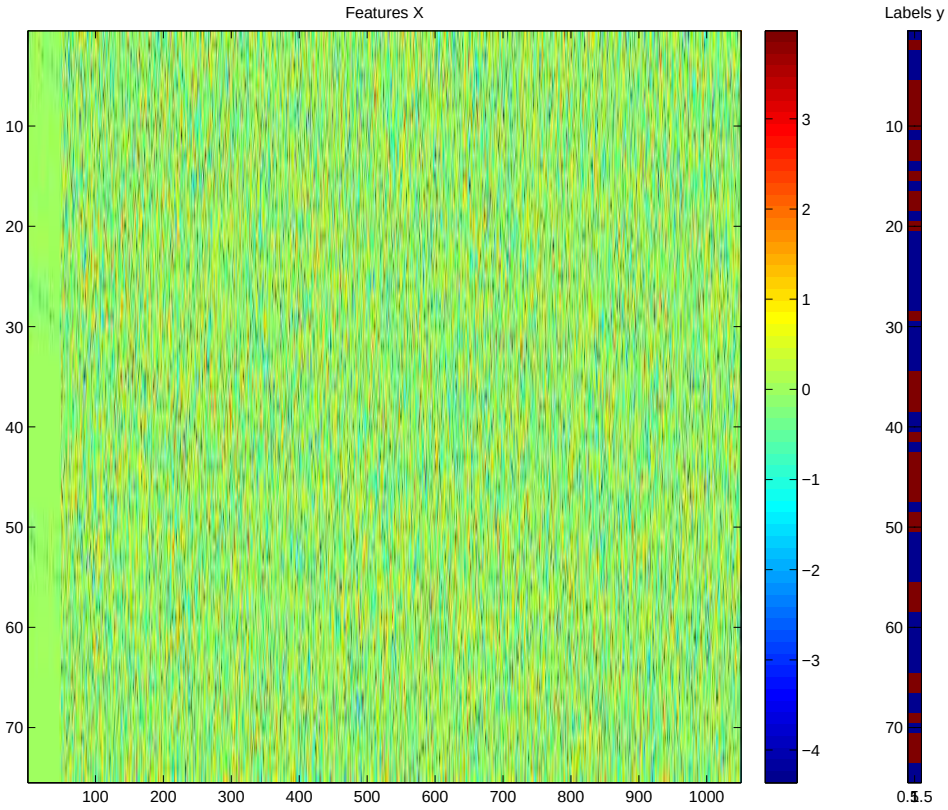


Figure 2: Generated features X and labels y.

3.2.3 MLE on dropout-noised datasets

We performed the same MLE optimization as described in 3.2.1, except with many copies of the augmented dropout-noised data. Given a dataset $X \in \mathbb{R}^{n \times d}$, we create N noised versions $\tilde{X}_i \in \mathbb{R}^{n \times d}$ for $i = 1, ..., N$ and concatenate to form $\tilde{X} = \begin{bmatrix} \tilde{X}_1 \\ \vdots \\ \tilde{X}_N \end{bmatrix} \in \mathbb{R}^{Nn \times d}$. The augmented $\tilde{y} \in \mathbb{R}^{Nn}$ is just the original $y \in \mathbb{R}^n$ repeated N times. Then, we solve

$$\min_{\beta} -\ell(\tilde{p}) \tag{26}$$

where $\ell(\tilde{p}) = \sum_{i=1}^{N} n\tilde{y}_i \log \tilde{p}_i + (1 - \tilde{y}_i) \log(1 - \tilde{p}_i)$ and $\tilde{p}_i = 1/(1 + \exp(-\tilde{x}_i \cdot \beta))$.

3.2.4 MLE with dropout regularization

Here, we use the quadratic approximation to dropout noise, rather than optimizing over an augmented dataset, as described in 3.2.3:

$$\min_{\beta} -\ell(p) + \lambda R^q(\beta), \tag{27}$$

where $R^q(\beta) = \frac{1}{2}\frac{\delta}{1-\delta}\sum_{i=1}^{n}\sum_{j=1}^{d} p_i(1-p_i)x_{ij}^2\beta_j^2$, the quadratic dropout approximation for logistic regression, as defined in (19).

Note that this objective is nonconvex, due to the $p_i(1-p_i)$ term in $R^q(\beta)$. However, if p is given, then the objective is convex. Therefore, we first assumed that p was fixed and solved for the optimal $\beta^\star$, updated $p = 1/(1+\exp(-\tilde{x}_i \cdot \beta^\star))$, and repeated until convergence. This approach worked quite well in our simulations.

3.2.5 MLE with dropout regularization with full-Fisher-estimate

Because dropout can be interpreted rescaling the features by an estimate of the diagonal of an estimated Fisher information matrix, we also attempted to explicitly rescale the features by the full estimated matrix. Specifically, for logistic regression, the optimization problem becomes:

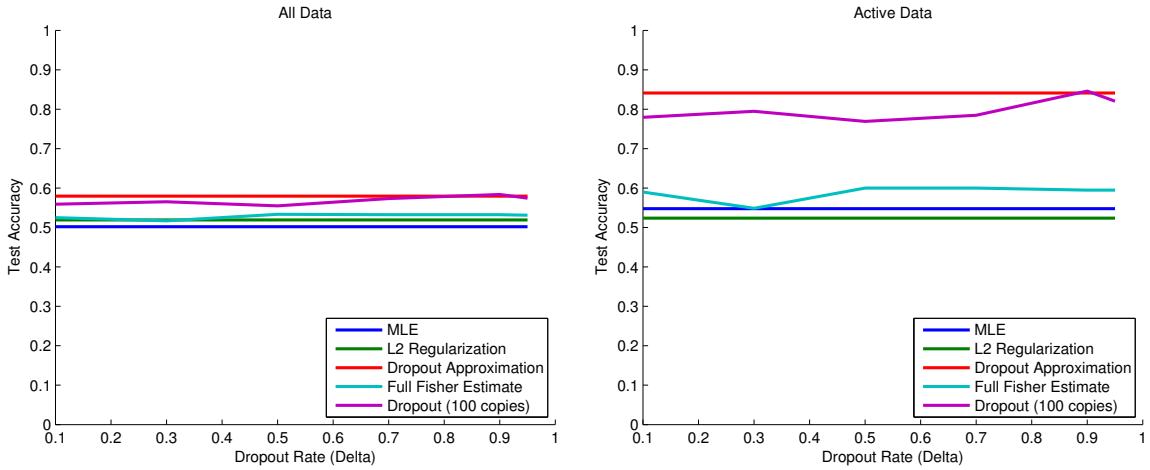
$$\min_{\beta} -\ell(p) + \frac{1}{2}\lambda\sum_{j=1}^{d}\beta_j^2\sum_{i=1}^{n} p_i(1-p_i)\sum_{k=1}^{n} x_{ij}x_{ik} \tag{28}$$

As shown in Figure 3, this resulted in slightly improved performance in comparison to the regular and L2-regularized MLE, it was not as effective as dropout or its quadratic approximation. This could be explained by an insufficient amount data to accurately estimate the full Fisher information matrix.

3.3 Findings

Figure 3a shows that dropout and its quadratic approximation slightly outperformed our other methods in test accuracy over all training examples. Interestingly, Figure 3b illustrates the sharp improvement of the dropout methods when performance is calculated just over the active instances, where the rare-but-useful features are being used; in contrast, the non-dropout methods show very little improvement in accuracy over active instances. This result is due to dropout's ability to focus on rare but highly discriminative features, whereas simple MLE or MLE + ridge regression cannot give preferential weight to these types of features.

Figure 4 shows the test accuracy of the approach described in 3.2.3, where we performed simple maximum likelihood estimation on augmented noised datasets, as we varied the noise parameter δ and the number of noisy copies. Generally, performance improves as the number of noisy copies increases; accuracy over all examples (Figure 4a) sharply improves when the number of training copies reaches a threshold around 20. Figure 4b gives a more nuanced result for accuracy over active instances: as the dropout rate δ increases, the number of copies required to achieve a higher accuracy actually decreases. Furthermore,

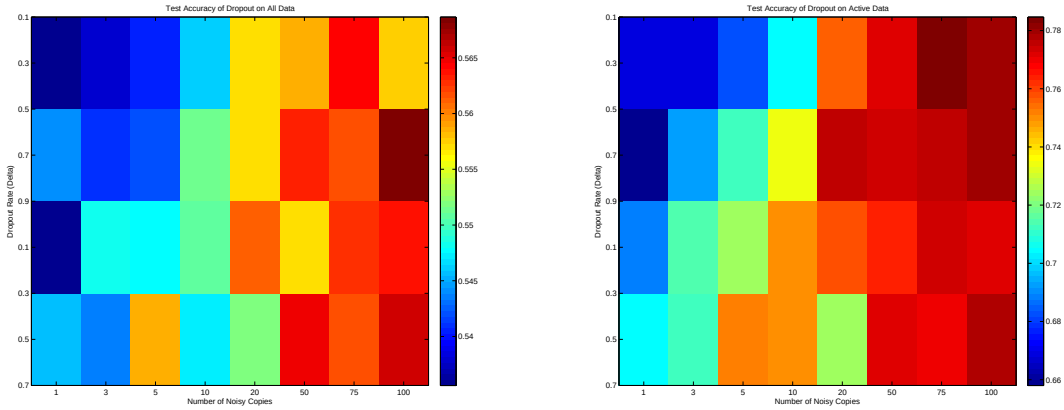


(a) Test accuracy vs. dropout rate on all training examples.

(b) Test accuracy vs. dropout rate on active training examples.

Figure 3: Test accuracy on all and active instances.

while we might expect there to be a larger variation in performance as we vary δ in the quadratic approximation, because the objective is nonconvex and we use an alternating solution method, it is likely that it is not actually converging to the optimal local minima.



(a) Test accuracy vs. dropout rate and number of noisy copies on all training examples.

(b) Test accuracy vs. dropout rate and number of noisy copies on active training examples.

Figure 4: Test accuracy on all and active instances, for MLE approach on augmented noisy datasets.

4 Discussion

One point raised in class was a comparison of random forests to dropout training. A random forest is a collection of decision trees where each split in the trees is calculated on a random subset of the total number of features d, where the number of features d' in the subset is much smaller than the total number of features (typically, if there are d features, random forests look at only $d' = \sqrt{d}$ features for each split). Then, all the random trees in the forest vote on the final classification [1]. Both dropout training and random forests are essentially getting rid of extra features (for dropout, those features that are randomly set to 0; for random forests, those randomly-not-chosen $d - d'$ features that aren't used in a split) and training on multiple random subsets of features (for dropout, the multiple noised copies of X that are used in the final training $\tilde{X}$ (as we performed in 3.2.3); for random forests, each decision tree that votes). These techniques help both methods to avoid overfitting to the training set and to find a better local optimum.

References

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

[2] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

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