EXACT SCALABLE SOFTMAX OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent state-of-the-art neural network and language models have begun to rely on softmax distributions with an extremely large number of categories. In this context calculating the softmax normalizing constant is prohibitively expensive, which has spurred a growing literature of efficiently computable but biased estimates of the softmax. In this paper we present the first two *unbiased* algorithms for optimizing the softmax whose work per iteration is independent of the number of classes and datapoints (and does not require extra work at the end of each epoch). We compare their empirical performance to the state-of-the-art on seven real world datasets, with our Implicit SGD algorithm comprehensively outperforming all competitors.

1 INTRODUCTION

Under the softmax model[1] the probability that a random variable y takes on the label $\ell \in \{1, ..., K\}$, is given by

$$p(y = \ell | x; W) = \frac{e^{x^\top w_\ell}}{\sum_{k=1}^{K} e^{x^\top w_k}}, \tag{1}$$

where $x \in \mathbb{R}^D$ is the covariate, $w_k \in \mathbb{R}^D$ is the vector of parameters for the k-th class, and $W = [w_1, w_2, ..., w_K]^\top \in \mathbb{R}^{K \times D}$ is the parameter matrix. Given a dataset of N label-covariate pairs $\mathcal{D} = \{(y_i, x_i)\}_{i=1}^N$, the ridge regularized maximum log-likelihood problem is given by

$$L(W) = \sum_{i=1}^{N} x_i^\top w_{y_i} - \log(\sum_{k=1}^{K} e^{x_i^\top w_k}) - \frac{\mu}{2}\|W\|_2^2. \tag{2}$$

This paper focusses on how to numerically maximize (2) when both N and K are large. This is increasingly the case in modern applications such as natural language processing (Partalas et al., 2015). The softmax also has numerous applications in others fields such as economics and biomedicine (Rust & Zahorik, 1993; Gopal & Yang, 2013) and appears as a convex surrogate for the (hard) maximum loss in discrete optimization (Maddison et al., 2016) and network flows (Shahrokhi & Matula, 1990).

The difficulty in maximizing $L(W)$ for large values of K is that the normalizing sum $\sum_{k=1}^{K} e^{x_i^\top w_k}$ becomes prohibitively expensive to calculate. Several approximations that avoid calculating the normalizing sum have been proposed to address this difficulty. These include tree-structured methods (Bengio et al., 2003; Daume III et al., 2016; Grave et al., 2016), sampling methods (Bengio & Senécal, 2008; Mnih & Teh, 2012; Joshi et al., 2017) and self-normalization (Andreas & Klein, 2015). Alternative models, such as the spherical family of losses (de Brébisson & Vincent, 2015; Vincent et al., 2015), which do not require normalization have also been proposed to sidestep the issue entirely (Martins & Astudillo, 2016). Krishnapuram et al. (2005) avoid calculating the sum using a maximization-majorization approach based on lower-bounding the eigenvalues of the Hessian matrix.

All[2] of these approximations are computationally tractable for large N and K, but do not converge to the true optimum $W^* = \operatorname{argmax} L(W)$. The goal of this paper is to develop methods that are both tractable and converge to the *exact* solution of (2) for large N and K.

[1] Also known as the multinomial logit model.

[2] The method of Krishnapuram et al. (2005) does converge to the true optimum, but has $O(ND)$ runtime per iteration which is not feasible for large N.

Following Raman et al. (2016), we recast (2) as a double-sum. The double-sum representation is amenable to Stochastic Gradient Descent (SGD) with only $O(D)$ computation per iteration. The difficulty is that vanilla SGD is numerically unstable when applied to this formulation. To counter this instability we develop a new SGD method called U-max, which is guaranteed to have bounded gradients and converge to the true solution of (2) for all sufficiently small learning rates. We also discuss a second method that employs Implicit SGD, a stochastic gradient method that is known to be more stable than vanilla SGD and yet has similar convergence properties (Toulis et al., 2016). We show that the Implicit SGD updates for the double-sum formulation can be efficiently computed in a numerically stable manner.

We compare the performance of U-max and Implicit SGD to the state-of-the-art (biased) methods for optimizing the softmax which have runtime $O(D)$ per iteration. Implicit SGD clearly outperforms all competitors, having a prediction error rate on average 40% lower than the next best algorithm. U-max has more variable performance due to its sensitivity to the learning rate.

In summary, our contributions in this paper are as follows:

1. Provide a simple derivation of the softmax double-sum formulation and identify why vanilla SGD is numerically unstable when applied to this formulation (Section 2).
2. Propose the U-max algorithm to stabilize the SGD updates and prove its convergence (Section 3.1).
3. Derive efficient Implicit SGD updates and analyze their runtime (Section 3.2).
4. Conduct experiments which show that Implicit SGD outperforms the previous state-of-the-art (Section 4).

2 Convex double-sum formulation

2.1 Derivation of double-sum

In order to apply an SGD method that samples both datapoints and classes each iteration, we need to represent (2) as a double sum over datapoints and classes. We begin by rewriting (2) in a more convenient form,

$$L(W) = -\sum_{i=1}^{N} \log(1 + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i})}) - \frac{\mu}{2}\|W\|_2^2. \tag{3}$$

The key to converting (3) into its double-sum representation is to express the negative logarithm using its convex conjugate:

$$\begin{aligned} -\log(a) &= \max_{v<0}\{av - (-\log(-v) - 1)\} \\ &= \max_{u}\{-u - \exp(-u)a + 1\} \end{aligned} \tag{4}$$

where $u = -\log(-v)$ and the optimal value of u is $u^*(a) = \log(a)$. Applying (4) to each of the logarithmic terms in (3) yields

$$\begin{aligned} L(W) &= \sum_{i=1}^{N} \max_{u_i \in \mathbb{R}}\{-u_i - e^{-u_i}(1 + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i})}) + 1\} - \frac{\mu}{2}\|W\|_2^2 \\ &= -\min_{u \geq 0}\{f(u, W)\} + N, \end{aligned}$$

where

$$f(u, W) = \sum_{i=1}^{N} \sum_{k \neq y_i} \frac{u_i + e^{-u_i}}{K-1} + e^{x_i^\top (w_k - w_{y_i}) - u_i} + \frac{\mu}{2}\|W\|_2^2 \tag{5}$$

is our double-sum representation and the optimal solution for u_i is $u_i^*(W) = \log(1 + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i})}) \geq 0$. Clearly f is a jointly convex function in u and W. In Appendix A we prove that the optimal value of u and W is contained in a compact convex set and that f is strongly

convex within this set. Thus performing projected-SGD on f is guaranteed to converge to a unique optimum with a convergence rate of $O(1/T)$ where T is the number of iterations (Lacoste-Julien et al., 2012).

Raman et al. (2016) derived a similar expression for f, which is equivalent to applying the convex conjugate substitution to (2) instead of (3). Our double-sum formulation leads to more stable stochastic gradients and faster convergence, as will be discussed in Section 2.3. Their method for optimizing f also requires that the value of $u_i^*(W)$ be calculated exactly at the end of each epoch for all $i = 1, ..., N$, at a cost of $\mathcal{O}(NKD)$. This is a significant bottleneck, even under their parallel implementation, and makes their method ill-suited for when N, K and D are all large. The goal in this paper is to avoid the $\mathcal{O}(NKD)$ bottleneck and to optimize f only using stochastic gradient descent methods that take $O(D)$ per iteration.

2.2 Numerical instability of vanilla SGD

The challenge in optimizing f using SGD is that it is not numerically stable. Note that $f = \mathbb{E}_{ik}[f_{ik}]$ where $i \sim \text{unif}(\{1, ..., N\})$, $k \sim \text{unif}(\{1, ..., K\} - \{y_i\})$ and

$$f_{ik}(u, W) = N\left(u_i + e^{-u_i} + (K-1)e^{x_i^\top(w_k - w_{y_i}) - u_i}\right) + \frac{\mu}{2}(\beta_{y_i}\|w_{y_i}\|_2^2 + \beta_k\|w_k\|_2^2), \quad (6)$$

where $\beta_j = \frac{N}{n_j + (N-n_j)(K-1)}$ is the inverse of the probability of class j being sampled either through i or k, and $n_j = |\{i : y_i = j, i = 1, ..., N\}|$. The corresponding stochastic gradient is:

$$\begin{aligned}
\nabla_{w_k} f_{ik}(u, W) &= N(K-1)e^{x_i^\top(w_k - w_{y_i}) - u_i} x_i + \mu\beta_k w_k \\
\nabla_{w_{y_i}} f_{ik}(u, W) &= -N(K-1)e^{x_i^\top(w_k - w_{y_i}) - u_i} x_i + \mu\beta_{y_i} w_{y_i} \\
\nabla_{w_{j \notin \{k, y_i\}}} f_{ik}(u, W) &= 0 \\
\nabla_{u_i} f_{ik}(u, W) &= -N(K-1)e^{x_i^\top(w_k - w_{y_i}) - u_i} + N(1 - e^{-u_i}) \quad (7)
\end{aligned}$$

If u_i equals its optimal value $u_i^*(W) = \log(1 + \sum_{k \neq y_i} e^{x_i^\top(w_k - w_{y_i})})$ then $e^{x_i^\top(w_k - w_{y_i}) - u_i} \leq 1$ and the magnitude of the $N(K-1)$ terms in the stochastic gradient is bounded by $N(K-1)\|x_i\|_2$. However if $u_i \ll x_i^\top(w_k - w_{y_i})$, then $e^{x_i^\top(w_k - w_{y_i}) - u_i} \gg 1$ and the magnitude of the $N(K-1)$ terms can become extremely large. In practice we found that applying vanilla SGD lead to numerical instability because of overflow. Increasing the number of bits or decreasing the step size helped with stability, but led to much slower convergence.

The same problem of numerical instability arises if we approach optimizing (3) via stochastic composition optimization (Wang et al., 2016). As is shown in Appendix B, stochastic composition optimization yields near-identical expressions for the stochastic gradients in (7) and has the same numerical stability issues.

All of the (biased) sampled softmax optimizers in the literature (Bengio & Senécal, 2008; Mnih & Teh, 2012; Joshi et al., 2017) are numerically stable because their approximation to $u_i^*(W)$ is always greater than or equal to $x_i^\top(w_k - w_{y_i})$. For example, in one-vs-each (Titsias, 2016), $u_i^*(W)$ is approximated by $\log(1 + e^{x_i^\top(w_k - w_{y_i})}) \geq x_i^\top(w_k - w_{y_i})$.

The U-max method that we will propose in Section 3.1 stabilizes SGD by increasing the value of u_i whenever it is significantly below $x_i^\top(w_k - w_{y_i})$. This is similar in effect to clipping the gradients; however, unlike clipping, this method is guaranteed to converge to the true optimum. The other approach we pursue for stabilizing SGD is to use Implicit SGD. Implicit SGD is ideally suited to situations where the magnitude of the gradient sharply decreases along a gradient descent direction, as is the case when $u_i \ll x_i^\top(w_k - w_{y_i})$. The Implicit SGD method is discussed in Section 3.2.

2.3 Choice of double-sum formulation

In Section 2.1 we claimed that applying the convex conjugate substitution to (3) instead of (2) leads to more stable gradients and faster convergence. Here we provide some theoretical intuition to support the claim and later in Section 4 we will provide some numerical evidence.

Let us consider applying the convex conjugate substitution to (2) instead of (3). The corresponding component function f_{ik} is

$$\tilde{f}_{ik}(\tilde{u}, W) = N\left(\tilde{u}_i - x_i^\top w_{y_i} + e^{x_i^\top w_{y_i} - \tilde{u}_i} + (K-1)e^{x_i^\top w_k - \tilde{u}_i}\right) + \frac{\mu}{2}(\beta_{y_i}\|w_{y_i}\|_2^2 + \beta_k\|w_k\|_2^2) \tag{8}$$

and the optimal solution for $\tilde{u}_i$ is $\tilde{u}_i^*(W^*) = \log(\sum_{k=1}^K e^{x_i^\top w_k^*})$. The functions f and $\tilde{f}$ are identical if we set $u_i = \tilde{u}_i - x_i^\top w_{y_i}$. Typically $x_i^\top w_{y_i}^* = \text{argmax}_k\{x_i^\top w_k^*\} \geq 0$ and so the $\tilde{u}_i$, $x_i^\top w_{y_i}$ and $e^{x_i^\top w_{y_i} - \tilde{u}_i}$ terms in (8) are of the greatest magnitude[3]. Even though at optimality these terms should roughly cancel, this will not be the case during the early stages of optimization, leading to stochastic gradients with large magnitude.

In contrast the function f_{ik} in (6) only has $x_i^\top w_{y_i}$ appearing as a negative exponent. Thus if $x_i^\top w_{y_i}$ is large then the magnitude of the stochastic gradients will be small. Since the converge of SGD is inversely proportional to the magnitude of its gradients (Lacoste-Julien et al., 2012), we expect the f_{ik} formulation to converge faster. In Section 4 we present numerical results confirming this fact.

3 Stable SGD methods

3.1 U-max method

As explained in Section 2.2, vanilla SGD becomes unstable when $u_i < x_i^\top(w_k - w_{y_i})$. However we know[4] that $u_i^*(W) \geq x_i^\top(w_k - w_{y_i})$. Hence instability occurs only when u_i is less than its optimum value for the current value of W. If $u_i < x_i^\top(w_k - w_{y_i})$ then increasing u_i will bring it closer to $u_i^*(W)$ and decrease the objective $f(u, W)$. If we only know the value of $x_i^\top(w_k - w_{y_i})$, the most that we can increase u_i without overshooting $u_i^*(W)$ is to set $u_i = \log(1 + e^{x_i^\top(w_k - w_{y_i})})$. Since $u_i = \log(1 + e^{x_i^\top(w_k - w_{y_i})}) \geq x_i^\top(w_k - w_{y_i})$, this also restores the stability of the SGD gradients.

This is exactly the mechanism behind the U-max algorithm – see Algorithm 1 in Appendix C for its pseudocode. U-max is the same as vanilla SGD except for two modifications: (a) u_i is set equal to $\log(1 + e^{x_i^\top(w_k - w_{y_i})})$ whenever $u_i \leq \log(1 + e^{x_i^\top(w_k - w_{y_i})}) - \delta$ for some threshold $\delta > 0$, (b) u_i is projected onto $[0, B_u]$, and W onto $\{W : \|W\|_2 \leq B_W\}$, where B_u and B_W are set so that the optimal $u_i^* \in [0, B_u]$ and the optimal W^* satisfies $\|W^*\|_2 \leq B_W$. See Appendix A for more details on how to set B_u and B_W.

Theorem 1. *Let $B_f = \max_{\|W\|_2^2 \leq B_W^2, 0 \leq u \leq B_u,} \max_{ik} \|\nabla f_{ik}(u, W)\|_2$. Suppose learning rate $\eta_t \leq \delta^2/(4B_f^2)$, then U-max with threshold δ converges to the optimum of (2), and the rate is at least as fast as SGD with same learning rate, in expectation.*

Proof. The proof is provided in Appendix D. □

Since $\delta \geq x_i^\top(w_k - w_{y_i}) - u_i$ (otherwise u_i would be increased to $\log(1 + e^{x_i^\top(w_k - w_{y_i})})$), the magnitude of the U-max stochastic gradients is bounded above by $N(K-1)e^\delta$. Thus, when δ is small the magnitude of the gradients will be small, and they can be computed in a numerically stable manner. However, Theorem 1 implies that the learning rate needs to be set low to guarantee convergence. When δ is large the learning rate can be large, but the gradient computation will be less numerically stable. Thus, in U-max, there is a trade-off between the learning rate η and numerical stability that is controlled by δ.

3.2 Implicit SGD

Another method that solves the numerical instability issues is Implicit SGD[5] (Bertsekas, 2011; Toulis et al., 2016). Implicit SGD uses the update equation

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla f(\theta^{(t+1)}, \xi_t), \tag{9}$$

[3] Recall that $\tilde{u}_i = u_i + x_i^\top w_{y_i} \geq x_i^\top w_{y_i}$ and so $\tilde{u}_i$ will be large if $x_i^\top w_{y_i}$ is.

[4] Since $u_i^*(W) = \log(1 + \sum_{j \neq y_i} e^{x_i^\top(w_k - w_{y_i})}) \geq x_i^\top(w_k - w_{y_i})$.

[5] Also known to as an "incremental proximal algorithm" (Bertsekas, 2011).

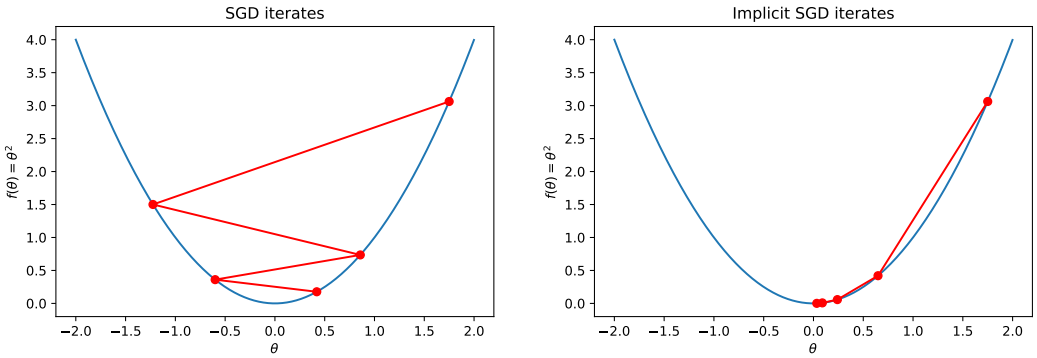

Figure 1: Illustration of the difference between SGD and Implicit SGD in optimizing $f(\theta) = \theta^2$. The learning rate was set to be $\eta = 1.75$ in both cases.

where $\theta^{(t)}$ is the value of the tth iterate, f is the function we seek to minimize and ξ_t is a random variable controlling the stochastic gradient such that $\nabla f(\theta) = \mathbb{E}_{\xi_t}[\nabla f(\theta, \xi_t)]$. The update (9) differs from vanilla SGD in that $\theta^{(t+1)}$ appears on both the left and right side of the equation, whereas in vanilla SGD it appears only on the left side. In our case $\theta = (u, W)$ and $\xi_t = (i_t, k_t)$ with $\nabla f(\theta^{(t+1)}, \xi_t) = \nabla f_{i_t,k_t}(u, W)$.

It is easiest to explain the difference between SGD and Implicit SGD using an example. Consider the quadratic function $f(\theta) = \theta^2$. The SGD vanilla update is $\theta^{(t+1)} = \theta^{(t)} \cdot (1 - \eta_t)$ whereas the Implicit SGD update is $\theta^{(t+1)} = \theta^{(t)}/(1 + \eta_t)$ where η_t is the learning rate. An illustration of the iterates of vanilla SGD and Implicit SGD are plotted in Figure 1. We can observe that Implicit SGD converges faster to the optimum and takes smaller steps. Notably, Implicit SGD never overshoots the optimum, whereas SGD does. This property is particularly important in our application where the numerical instability comes from SGD gradients that grossly overshoot the optimum. Another desirable property of Implicit SGD is that it is more robust to the learning rate as compared to vanilla SGD (Toulis et al., 2016). Again this is important since a good value for the learning rate is never known a priori.

The difficulty in applying Implicit SGD is that in each iteration one has to compute a solution to (9). The tractability of this procedure is problem dependent. We show that computing a solution to (9) is indeed tractable for the problem considered in this paper. The details of these mechanisms are laid out in full in Appendix E. Below we comment on their runtime per iteration.

Proposition 1. *Consider the Implicit SGD algorithm where in each iteration only one datapoint i and one class $k \neq y_i$ is sampled. Then the Implicit SGD iterate $\theta^{(t+1)}$ can be computed to within ϵ accuracy in runtime $O(D + \log(\epsilon^{-1}))$.*

Proposition 2. *Consider the Implicit SGD algorithm where in each iteration n datapoints and m classes are sampled. Then the Implicit SGD update $\theta^{(t+1)}$ can be computed to within ϵ accuracy in runtime $O(n(n + m)(D + n\epsilon^{-1}))$.*

Proof. The proofs are provided in Appendix E. □

The reason for the ϵ^{-1} factor in the runtime of Proposition 2 is because the Implicit SGD update optimization problem contains a large number of variables and can only be solved with first-order methods. The optimization problem in Proposition 1 is much smaller and so second-order methods can be used, giving the $O(\log(\epsilon^{-1}))$ runtime.

The runtime in Proposition 2 consists of two terms: the $O(n(n + m)D)$ term for taking the inner products $x_i^\top w_k$ and the $O(n^2(n + m)\epsilon^{-1})$ term for computing the resulting Implicit SGD update. When $n = O(D\epsilon)$ the cost of computing the Implicit updated is approximately the same as taking the inner products, and so Implicit SGD will have a similar speed as vanilla SGD.

In the $O(\cdot)$ runtimes above we have ignored the time required to read or write the vectors w_k. This cost may, indeed, be the bottleneck in the practical performance of our algorithms. The advantage

Table 1: Datasets with a summary of their properties. Where the number of classes, dimension or number of examples has been altered, the original value is displayed in brackets.

DATASET	CLASSES	DIMENSION	EXAMPLES
MNIST	10	780	60,000
Bibtex	147 (159)	1,836	4,880
Delicious	350 (983)	500	12,920
Eurlex	838 (3,993)	5,000	15,539
AmazonCat-13K	2,709 (2,919)	10,000 (203,882)	100,000 (1,186,239)
Wiki10	4,021 (30,938)	10,000 (101,938)	14,146
Wiki-small	18,207 (28,955)	10,000 (2,085,164)	90,737 (342,664)

of sampling multiple points and classes each iteration can be seen by considering the ratio of inner product calculations to memory access operations. If we can maximize the number of inner products per memory access operation, then we can extract the maximum amount of information about our function per unit time (measured in memory access operations), leading to better performance. In Proposition 1 we have 2 inner products (one for the true class and one for the sampled class) and 2 memory access operations, which gives a ratio of 1. In Proposition 2 we have $n(n+m)$ inner products (n datapoints with n true classes and m sampled classes) and $n+m$ memory access operations, for a ratio of n. Thus sampling multiple points and classes each iteration leads to a higher inner product to memory access operation ratio, which should lead to faster runtime.

The benefit of sampling multiple points and classes must be balanced by the $O(\epsilon^{-1})$ vs $O(\log(\epsilon^{-1}))$ runtime to solve the resulting Implicit SGD update equation and the requirement that $n = O(D\epsilon)$ so that Implicit SGD has a similar runtime to vanilla SGD. Ultimately the choice of whether to sample a single or multiple datapoints and classes will depend on the properties of the hardware available and the value of D, which is problem dependent.

4 EXPERIMENTS

Here we report the results of experiments comparing the performance of U-max and Implicit SGD to the state-of-the-art on seven different datasets. We'll begin by specifying the experimental setup and then move onto the results.

Data. We ran our algorithms on the MNIST, Bibtex, Delicious, Eurlex, AmazonCat-13K, Wiki10, and Wiki-small datasets[6], the properties of which are summarized in Table 1. Most of the datasets are multi-label and, as is standard practice (Titsias, 2016), we took the first label as being the true label and discarded the remaining labels. To make the computation more manageable, we truncated the number of features to be at most 10,000 and the training and test size to be at most 100,000. If, as a result of the dimension truncation, a datapoint had no non-zero features then it was discarded. The features of each dataset were normalized to have unit L_2 norm. All of the datasets were pre-separated into training and test sets. We only focus on the performance on the algorithms on the training set, as the goal in this paper is to investigate how best to optimize the softmax likelihood, which is given over the training set.

Algorithms. We compared our algorithms to the state-of-the-art methods for optimizing the softmax which have runtime $O(D)$ per iteration[7]. The competitors include Noise Contrastive Estimation (NCE) (Mnih & Teh, 2012), Importance Sampling (IS) (Bengio & Senécal, 2008) and One-Vs-Each (OVE) (Titsias, 2016). Note that these methods are all biased and will not converge to the true softmax optimum, but something close to it. For these algorithms we set $n = 100, m = 5$,

[6]All of the datasets were downloaded from `http://manikvarma.org/downloads/XC/XMLRepository.html`, except Wiki-small which was obtained from `http://lshtc.iit.demokritos.gr/`.

[7]Or $O(nmD)$ per mini-batch where n are the number of datapoints in the mini-batch and m the number of classes.

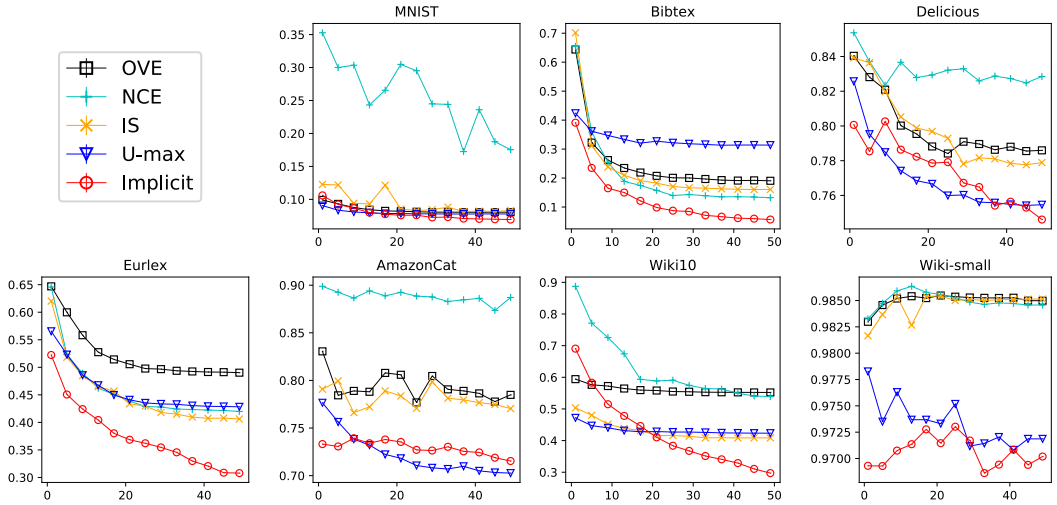

Figure 2: **Prediction error.** The x-axis is the number of epochs and the y-axis is the fraction of points for which $y_i \neq \text{argmax}_k\{x_i^\top w_k\}$ for the current value of W.

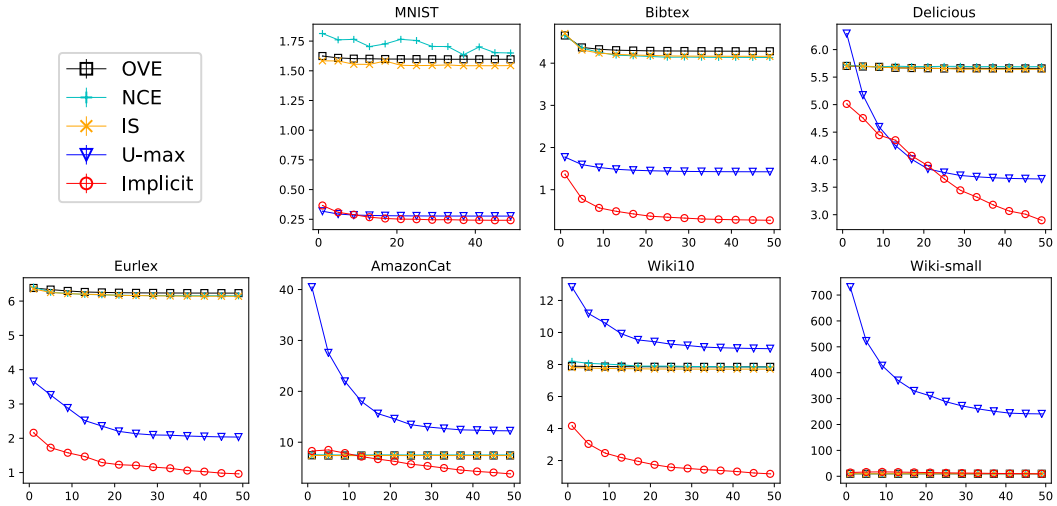

Figure 3: **Log-loss.** The x-axis is the number of epochs and the y-axis is the log-loss from (2) calculated at the current value of W.

which are standard settings[8]. For our Implicit SGD method we chose to implement the version in Proposition 1 which has $n = 1, m = 1$. Likewise for U-max we set $n = 1, m = 1$ and the threshold parameter $\delta = 1$. The ridge regularization parameter μ was set to zero for all algorithms.

Epochs and losses. Each algorithm is run for 50 epochs on each dataset. The learning rate is decreased by a factor of 0.9 each epoch. Both the prediction error and log-loss (2) are recorded at the end of 10 evenly spaced epochs over the 50 epochs.

Learning rate. The magnitude of the gradient differs in each algorithm, due to them either under or over-estimating the log-sum derivative from (2). To set a reasonable learning rate for each algorithm, we ran them on the Eurlex dataset with learning rates $\eta = 10^{0,\pm1,\pm2,\pm3,\pm4}$ and saw which learning

[8]We also experimented setting $n = 1, m = 1$ in these methods and there was virtually no difference except the runtime was slower. For example in Appendix F we plot the performance of NCE for different learning rates with $n = 1, m = 1$ and $n = 100, m = 5$ and there is very little difference between the two.

Table 2: Relative prediction error and log-loss. The values for each dataset are normalized by dividing by the corresponding Implicit SGD error/log-loss.

DATASET	PREDICTION ERROR					LOG-LOSS				
	OVE	NCE	IS	U-M.	IMP.	OVE	NCE	IS	U-M.	IMP.
MNIST	1.16	2.53	1.20	1.12	**1.00**	6.60	6.82	6.38	1.15	**1.00**
Bibtex	3.36	2.33	2.83	5.52	**1.00**	15.58	15.04	15.14	5.18	**1.00**
Delicious	1.05	1.11	1.04	1.01	**1.00**	1.95	1.96	1.95	1.26	**1.00**
Eurlex	1.59	1.36	1.32	1.39	**1.00**	6.47	6.39	6.38	2.11	**1.00**
AmazonCat	1.10	1.24	1.08	**0.98**	1.00	1.96	1.98	1.95	3.23	**1.00**
Wiki10	1.86	1.81	1.37	1.42	**1.00**	6.72	6.71	6.60	7.69	**1.00**
Wiki-small	1.02	1.01	1.02	1.00	**1.00**	1.04	1.05	1.04	25.70	**1.00**
Average	1.59	1.63	1.41	1.78	**1.00**	5.76	5.71	5.63	6.62	**1.00**

rate gave the best performance for each algorithm (the results are presented in Appendix F). This learning rate is used in all subsequent experiments.

Results. Plots of the performance of the algorithms on each dataset are displayed in Figures 2 and 3 with the relative performance compared to Implicit SGD given in Table 2. The Implicit SGD method has the best performance on virtually all datasets for both the prediction error and the log-loss metrics. Not only does it converge faster in the first few epochs, it also converges to the true optimum value (unlike the biased methods that prematurely plateau). On average after 50 epochs, Implicit SGD has a prediction error 41% lower than the next best algorithm and its log-loss is a factor of 5.63 lower than the next best. The U-max algorithm has more varied performance, it was the only algorithm to outperform Implicit SGD's prediction error on one dataset but other times had the worst performance. On AmazonCat and Wiki-Small the U-max log-loss becomes extremely large. The reason for this can be explained by Theorem 1, which does not guarantee convergence for U-max if the learning rate is too high. It is likely that the learning rate of U-max was too high when applied to those datasets.

The results of running each method on the Eurlex dataset with learning rates $\eta = 10^{0,\pm1,\pm2,\pm3,\pm4}$ is presented in Appendix F. The results are consistent with those in Figures 2 and 3, with Implicit having the best performance for most learning rate settings. In Appendix F we also present the results of U-max run on the alternative double-sum formulation discussed in Section 2.3, and its performance is significantly worse than using our double-sum formulation.

5 Conclusion

In this paper we have presented the U-max and Implicit SGD algorithms for optimizing the softmax distribution. These are the *first* algorithms that require only $O(D)$ computation per iteration (without extra work at the end of each epoch) that converge to the true softmax optimum. Implicit SGD can be efficiently implemented and clearly out-performs the previous state-of-the-art on seven different datasets. The result is a new method that enables optimizing the softmax for extremely large number of samples and classes.

So far Implicit SGD has only been applied to the simple softmax, but could also be applied to any neural network where the final layer is the softmax. Applying Implicit SGD to word2vec type models, which can be viewed as softmaxes where both x and w are parameters to be fit, might be particularly fruitful.

REFERENCES

Jacob Andreas and Dan Klein. When and why are log-linear models self-normalizing? In *HLT-NAACL*, pp. 244–249, 2015.

Yoshua Bengio and Jean-Sébastien Senécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 19(4):713–722, 2008.

Yoshua Bengio, Jean-Sébastien Senécal, et al. Quick training of probabilistic neural nets by importance sampling. In *AISTATS*, 2003.

Dimitri P Bertsekas. Incremental proximal methods for large scale convex optimization. *Mathematical programming*, 129(2):163, 2011.

Robert M Corless, Gaston H Gonnet, David EG Hare, David J Jeffrey, and Donald E Knuth. On the Lambert W function. *Advances in Computational mathematics*, 5(1):329–359, 1996.

Hal Daume III, Nikos Karampatziakis, John Langford, and Paul Mineiro. Logarithmic time one-against-some. *arXiv:1606.04988*, 2016.

Alexandre de Brébisson and Pascal Vincent. An exploration of softmax alternatives belonging to the spherical loss family. *arXiv:1511.05042*, 2015.

Siddharth Gopal and Yiming Yang. Distributed training of large-scale logistic models. In *ICML (2)*, pp. 289–297, 2013.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. Efficient softmax approximation for GPUs. *arXiv:1609.04309*, 2016.

Bikash Joshi, Massih-Reza Amini, Ioannis Partalas, Franck Iutzeler, and Yury Maximov. Aggressive sampling for multi-class to binary reduction with applications to text classification. *arXiv:1701.06511*, 2017.

Balaji Krishnapuram, Lawrence Carin, Mario AT Figueiredo, and Alexander J Hartemink. Sparse multinomial logistic regression: Fast algorithms and generalization bounds. *IEEE transactions on pattern analysis and machine intelligence*, 27(6):957–968, 2005.

Simon Lacoste-Julien, Mark Schmidt, and Francis Bach. A simpler approach to obtaining an O(1/t) convergence rate for the projected stochastic subgradient method. *arXiv:1212.2002*, 2012.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv:1611.00712*, 2016.

André FT Martins and Ramón Fernandez Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. *CoRR, abs/1602.02068*, 2016.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv:1206.6426*, 2012.

Ioannis Partalas, Aris Kosmopoulos, Nicolas Baskiotis, Thierry Artieres, George Paliouras, Eric Gaussier, Ion Androutsopoulos, Massih-Reza Amini, and Patrick Galinari. LSHTC: A benchmark for large-scale text classification. *arXiv:1503.08581*, 2015.

Parameswaran Raman, Shin Matsushima, Xinhua Zhang, Hyokun Yun, and SVN Vishwanathan. DS-MLR: Exploiting double separability for scaling up distributed multinomial logistic regression. *arXiv:1604.04706*, 2016.

Roland T Rust and Anthony J Zahorik. Customer satisfaction, customer retention, and market share. *Journal of retailing*, 69(2):193–215, 1993.

Farhad Shahrokhi and David W Matula. The maximum concurrent flow problem. *Journal of the ACM (JACM)*, 37(2):318–334, 1990.

Michalis K Titsias. One-vs-each approximation to softmax for scalable estimation of probabilities. *arXiv:1609.07410*, 2016.

Panos Toulis, Dustin Tran, and Edo Airoldi. Towards stability and optimality in stochastic gradient descent. In *Artificial Intelligence and Statistics*, pp. 1290–1298, 2016.

Pascal Vincent, Alexandre de Brébisson, and Xavier Bouthillier. Efficient exact gradient update for training deep networks with very large sparse targets. In *Advances in Neural Information Processing Systems*, pp. 1108–1116, 2015.

Mengdi Wang, Ji Liu, and Ethan Fang. Accelerating stochastic composition optimization. In *Advances in Neural Information Processing Systems*, pp. 1714–1722, 2016.

A PROOF OF VARIABLE BOUNDS AND STRONG CONVEXITY

We first establish that the optimal values of u and W are bounded. Next, we show that within these bounds the objective is strongly convex and its gradients are bounded.

Lemma 1 ((Raman et al., 2016))**.** *The optimal value of W is bounded as $\|W^*\|_2^2 \leq B_W^2$ where $B_W^2 = \frac{2}{\mu} N \log(K)$.*

Proof.

$$-N\log(K) = L(0) \leq L(W^*) \leq -\frac{\mu}{2}\|W^*\|_2^2$$

Rearranging gives the desired result. □

Lemma 2. *The optimal value of u_i is bounded as $u_i^* \leq B_u$ where $B_u = \log(1 + (K-1)e^{2B_x B_w})$ and $B_x = \max_i\{\|x_i\|_2\}$*

Proof.

$$\begin{aligned} u_i^* &= \log(1 + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i})}) \\ &\leq \log(1 + \sum_{k \neq y_i} e^{\|x_i\|_2 (\|w_k\|_2 + \|w_{y_i}\|_2)}) \\ &\leq \log(1 + \sum_{k \neq y_i} e^{2B_x B_w}) \\ &= \log(1 + (K-1)e^{2B_x B_w}) \end{aligned}$$

□

Lemma 3. *If $\|W\|_2^2 \leq B_W^2$ and $u_i \leq B_u$ then $f(u, W)$ is strongly convex with convexity constant greater than or equal to $\min\{\exp(-B_u), \mu\}$.*

Proof. Let us rewrite f as

$$\begin{aligned} f(u, W) &= \sum_{i=1}^N u_i + e^{-u_i} + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i}) - u_i} + \frac{\mu}{2}\|W\|_2^2 \\ &= \sum_{i=1}^N a_i^\top \theta + e^{-u_i} + \sum_{k \neq y_i} e^{b_{ik}^\top \theta} + \frac{\mu}{2}\|W\|_2^2. \end{aligned}$$

where $\theta = (u^\top, w_1^\top, ..., w_k^\top) \in \mathbb{R}^{N+KD}$ with a_i and b_{ik} being appropriately defined. The Hessian of f is

$$\nabla^2 f(\theta) = \sum_{i=1}^N e^{-u_i} e_i e_i^\top + \sum_{k \neq y_i} e^{b_{ik}^\top \theta} b_{ik} b_{ik}^\top + \mu \cdot diag\{0_N, 1_{KD}\}$$

where e_i is the ith canonical basis vector, 0_N is an N-dimensional vector of zeros and 1_{KD} is a KD-dimensional vector of ones. It follows that

$$\begin{aligned} \nabla^2 f(\theta) &\succeq I \cdot \min\{\min_{0 \leq u \leq B_u} \{e^{-u_i}\}, \mu\} \\ &= I \cdot \min\{\exp(-B_u), \mu\} \\ &\succeq 0. \end{aligned}$$

□

Lemma 4. *If $\|W\|_2^2 \leq B_W^2$ and $u_i \leq B_u$ then the 2-norm of both the gradient of f and each stochastic gradient f_{ik} are bounded by*

$$B_f = N \max\{1, e^{B_u} - 1\} + 2(N e^{B_u} B_x + \mu \max_k\{\beta_k\} B_W).$$

Proof. By Jensen's inequality

$$\begin{aligned}\max_{\|W\|_2^2 \le B_W^2, 0 \le u \le B_u} \|\nabla f(u,W)\|_2 &= \max_{\|W\|_2^2 \le B_W^2, 0 \le u \le B_u} \|\nabla \mathbb{E}_{ik} f_{ik}(u,W)\|_2 \\ &\le \max_{\|W\|_2^2 \le B_W^2, 0 \le u \le B_u} \mathbb{E}_{ik} \|\nabla f_{ik}(u,W)\|_2 \\ &\le \max_{\|W\|_2^2 \le B_W^2, 0 \le u \le B_u} \max_{ik} \|\nabla f_{ik}(u,W)\|_2.\end{aligned}$$

Using the results from Lemmas 1 and 2 and the definition of f_{ik} from (6),

$$\begin{aligned}\|\nabla_{u_i} f_{ik}(u,W)\|_2 &= \|N\left(1 - e^{-u_i} - (K-1)e^{x_i^\top (w_k - w_{y_i}) - u_i}\right)\|_2 \\ &= N|1 - e^{-u_i}(1 + (K-1)e^{x_i^\top (w_k - w_{y_i})})| \\ &\le N \max\{1, (1 + (K-1)e^{\|x_i\|_2(\|w_k\|_2 + \|w_{y_i}\|_2)}) - 1\} \\ &\le N \max\{1, e^{B_u} - 1\}\end{aligned}$$

and for j indexing either the sampled class $k \neq y_i$ or the true label y_i,

$$\begin{aligned}\|\nabla_{w_j} f_{ik}(u,W)\|_2 &= \| \pm N(K-1)e^{x_i^\top (w_k - w_{y_i}) - u_i} x_i + \mu \beta_j w_j\|_2 \\ &\le N(K-1)e^{\|x_i\|_2(\|w_k\|_2 + \|w_{y_i}\|_2)} \|x_i\|_2 + \mu \beta_j \|w_j\|_2 \\ &\le N e^{B_u} B_x + \mu \max_k \{\beta_k\} B_W.\end{aligned}$$

Letting

$$B_f = N \max\{1, e^{B_u} - 1\} + 2(N e^{B_u} B_x + \mu \max_k \{\beta_k\} B_W)$$

we have

$$\|\nabla f_{ik}(u,W)\|_2 \le \|\nabla_{u_i} f_{ik}(u,W)\|_2 + \|\nabla_{w_k} f_{ik}(u,W)\|_2 + \|\nabla_{w_{y_i}} f_{ik}(u,W)\|_2 = B_f.$$

In conclusion:

$$\max_{\|W\|_2^2 \le B_W^2, 0 \le u \le B_u} \|\nabla f(u,W)\|_2 \le \max_{\|W\|_2^2 \le B_W^2, u_i \le B_u,} \max_{ik} \|\nabla f_{ik}(u,W)\|_2 \le B_f.$$

□

B Stochastic Composition Optimization

We can write the equation for $L(W)$ from (3) as (where we have set $\mu = 0$ for notational simplicity),

$$\begin{aligned}L(W) &= -\sum_{i=1}^N \log(1 + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i})}) \\ &= \mathbb{E}_i[h_i(\mathbb{E}_k[g_k(W)])]\end{aligned}$$

where $i \sim unif(\{1, ..., N\})$, $k \sim unif(\{1, ..., K\})$, $h_i(v) \in \mathbb{R}$, $g_k(W) \in \mathbb{R}^N$ and

$$h_i(v) = -N \log(1 + e_i^\top v)$$

$$[g_k(W)]_i = \begin{cases} K e^{x_i^\top (w_k - w_{y_i})} & \text{if } k \neq y_i \\ 0 & \text{otherwise} \end{cases}.$$

Here $e_i^\top v = v_i \in \mathbb{R}$ is a variable that is explicitly kept track of with $v_i \approx \mathbb{E}_k[g_k(W)]_i = \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i})}$ (with exact equality in the limit as $t \to \infty$). Clearly v_i in stochastic composition optimization has a similar role as u_i has in our formulation for f in (5).

If i, k are sampled with $k \neq y_i$ in stochastic composition optimization then the updates are of the form (Wang et al., 2016)

$$\begin{aligned}w_{y_i} &= w_{y_i} + \eta_t N K \frac{e^{x_i^\top (z_k - z_{y_i})}}{1 + v_i} x_i \\ w_k &= w_k - \eta_t N K \frac{e^{x_i^\top (z_k - z_{y_i})}}{1 + v_i} x_i,\end{aligned}$$

where z_k is a smoothed value of w_k. These updates have the same numerical instability issues as vanilla SGD on f in (5): it is possible that $\frac{e^{x_i^\top z_k}}{1+v_i} \gg 1$ where ideally we should have $0 \le \frac{e^{x_i^\top z_k}}{1+v_i} \le 1$.

C U-MAX PSEUDOCODE

Algorithm 1: U-max

Input : Data $\mathcal{D} = \{(y_i, x_i) : y_i \in \{1, \ldots, K\}, x_i \in \mathbb{R}^d\}_{i=1}^N$, number of classes K, number of datapoints N, learning rate η_t, class sampling probability $\beta_k = \frac{N}{n_k + (N - n_k)(K-1)}$, threshold parameter $\delta > 0$, bound B_W on W such that $\|W\|_2 \leq B_W$ and bound B_u on u such that $u_i \leq B_u$ for $i = 1, ..., N$

Output: W

```
Initialize
for k = 1 to K do
    w_k ← 0
end
for i = 1 to N do
    u_i ← log(K)
end

Run SGD
for t = 1 to T do
    Sample indices
    i ~ unif({1, ..., N})
    k ~ unif({1, ..., K} − {y_i})

    Increase u_i
    if u_i < log(1 + e^{x_i^⊤ (w_k − w_{y_i})}) − δ then
        u_i ← log(1 + e^{x_i^⊤ (w_k − w_{y_i})})

    SGD step
    w_k ← w_k − η_t[N(K − 1) e^{x_i^⊤ (w_k − w_{y_i}) − u_i} x_i + μ β_k w_k]
    w_{y_i} ← w_{y_i} − η_t[−N(K − 1) e^{x_i^⊤ (w_k − w_{y_i}) − u_i} x_i + μ β_{y_i} w_{y_i}]
    u_i ← u_i − η_t[N(1 − e^{−u_i} − (K − 1) e^{x_i^⊤ (w_k − w_{y_i}) − u_i})]

    Projection
    w_k ← w_k · min{1, B_W / ||w_k||_2}
    w_{y_i} ← w_{y_i} · min{1, B_W / ||w_{y_i}||_2}
    u_i ← max{0, min{B_u, u_i}}
end
```

D PROOF OF CONVERGENCE OF U-MAX METHOD

In this section we will prove the claim made in Theorem 1, that U-max converges to the softmax optimum. Before proving the theorem, we will need a lemma.

Lemma 5. *For any $\delta > 0$, if $u_i \leq \log(1 + e^{x_i^\top (w_k - w_{y_i})}) - \delta$ then setting $u_i = \log(1 + e^{x_i^\top (w_k - w_{y_i})})$ decreases $f(u, W)$ by at least $\delta^2/2$.*

Proof. As in Lemma 3, let $\theta = (u^\top, w_1^\top, ..., w_k^\top) \in \mathbb{R}^{N+KD}$. Then setting $u_i = \log(1 + e^{x_i^\top (w_k - w_{y_i})})$ is equivalent to setting $\theta = \theta + \Delta e_i$ where e_i is the ith canonical basis vector and $\Delta = \log(1 + e^{x_i^\top (w_k - w_{y_i})}) - u_i \geq \delta$. By a second order Taylor series expansion

$$f(\theta) - f(\theta + \Delta e_i) \geq \nabla f(\theta + \Delta e_i)^\top e_i \Delta + \frac{\Delta^2}{2} e_i^\top \nabla^2 f(\theta + \lambda \Delta e_i) e_i \tag{10}$$

for some $\lambda \in [0, 1]$. Since the optimal value of u_i for a given value of W is $u_i^*(W) = \log(1 + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i})}) \geq \log(1 + e^{x_i^\top (w_k - w_{y_i})})$, we must have $\nabla f(\theta + \Delta e_i)^\top e_i \leq 0$. From Lemma 3

we also know that

$$
\begin{aligned}
e_i^\top \nabla^2 f(\theta + \lambda\Delta e_i) e_i &= \exp(-(u_i + \lambda\Delta)) + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i}) - (u_i + \lambda\Delta)} \\
&= \exp(-\lambda\Delta) e^{-u_i} (1 + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i})}) \\
&= \exp(-\lambda\Delta) \exp(-(\log(1 + e^{x_i^\top (w_k - w_{y_i})}) - \Delta))(1 + \sum_{k \neq y_i} e^{x_i^\top (w_k - w_{y_i})}) \\
&\geq \exp(\Delta - \lambda\Delta) \\
&\geq \exp(\Delta - \Delta) \\
&= 1.
\end{aligned}
$$

Putting in bounds for the gradient and Hessian terms in (10),

$$
f(\theta) - f(\theta + \Delta e_i) \geq \frac{\Delta^2}{2} \geq \frac{\delta^2}{2}.
$$

□

Now we are in a position to prove Theorem 1.

Proof of Theorem 1. Let $\theta^{(t)} = (u^{(t)}, W^{(t)}) \in \Theta$ denote the value of the tth iterate. Here $\Theta = \{\theta : \|W\|_2^2 \leq B_W^2, u_i \leq B_u\}$ is a convex set containing the optimal value of $f(\theta)$.

Let $\pi_i^{(\delta)}(\theta)$ denote the operation of setting $u_i = \log(1 + e^{x_i^\top (w_k - w_{y_i})})$ if $u_i \leq \log(1 + e^{x_i^\top (w_k - w_{y_i})}) - \delta$. If indices i, k are sampled for the stochastic gradient and $u_i \leq \log(1 + e^{x_i^\top (w_k - w_{y_i})}) - \delta$, then the value of f at the $t + 1^{st}$ iterate is bounded as

$$
\begin{aligned}
f(\theta^{(t+1)}) &= f(\pi_i(\theta^{(t)}) - \eta_t \nabla f_{ik}(\pi_i(\theta^{(t)}))) \\
&\leq f(\pi_i(\theta^{(t)})) + \max_{\theta \in \Theta} \|\eta_t \nabla f_{ik}(\pi_i(\theta))\|_2 \max_{\theta \in \Theta} \|\nabla f(\theta)\|_2 \\
&\leq f(\pi_i(\theta^{(t)})) + \eta_t B_f^2 \\
&\leq f(\theta^{(t)}) - \delta^2/2 + \eta_t B_f^2 \\
&\leq f(\theta^{(t)} - \eta_t \nabla f_{ik}(\theta^{(t)})) - \delta^2/2 + 2\eta_t B_f^2 \\
&\leq f(\theta^{(t)} - \eta_t \nabla f_{ik}(\theta^{(t)})),
\end{aligned}
$$

since $\eta_t \leq \delta^2/(4B_f^2)$ by assumption. Alternatively if $u_i \geq \log(1 + e^{x_i^\top (w_k - w_{y_i})}) - \delta$ then

$$
\begin{aligned}
f(\theta^{(t+1)}) &= f(\pi_i(\theta^{(t)}) - \eta_t \nabla f_{ik}(\pi_i(\theta^{(t)}))) \\
&= f(\theta^{(t)} - \eta_t \nabla f_{ik}(\theta^{(t)})).
\end{aligned}
$$

Either way $f(\theta^{(t+1)}) \leq f(\theta^{(t)} - \eta_t \nabla f_{ik}(\theta^{(t)}))$. Taking expectations with respect to i, k,

$$
\mathbb{E}_{ik}[f(\theta^{(t+1)})] \leq \mathbb{E}_{ik}[f(\theta^{(t)} - \eta_t \nabla f_{ik}(\theta^{(t)}))].
$$

Finally let P denote the projection of θ onto Θ. Since Θ is a convex set containing the optimum we have $f(P(\theta)) \leq f(\theta)$ for any θ, and so

$$
\mathbb{E}_{ik}[f(P(\theta^{(t+1)}))] \leq \mathbb{E}_{ik}[f(\theta^{(t)} - \eta_t \nabla f_{ik}(\theta^{(t)}))],
$$

which shows that the rate of convergence in expectation of U-max is at least as fast as that of standard SGD.

□

E UPDATE EQUATIONS FOR IMPLICIT SGD

In this section we will derive the updates for Implicit SGD. We will first consider the simplest case where only one datapoint (x_i, y_i) and a single class is sampled in each iteration with no regularizer. Then we will derive the more complicated update for when there are multiple datapoints and sampled classes with a regularizer.

E.1 SINGLE DATAPOINT, SINGLE CLASS, NO REGULARIZER

Equation (6) for the stochastic gradient for a single datapoint and single class with $\mu = 0$ is

$$f_{ik}(u, W) = N(u_i + e^{-u_i} + (K-1)e^{x_i^\top(w_k - w_{y_i}) - u_i}).$$

The Implicit SGD update corresponds to finding the variables optimizing

$$\min_{u,W} \left\{ 2\eta f_{ik}(u, W) + \|u - \tilde{u}\|_2^2 + \|W - \tilde{W}\|_2^2 \right\},$$

where η is the learning rate and the tilde refers to the value of the old iterate (Toulis et al., 2016, Eq. 6). Since f_{ik} is only a function of u_i, w_k, w_{y_i} the optimization reduces to

$$\begin{aligned}
&\min_{u_i, w_k, w_{y_i}} \left\{ 2\eta f_{ik}(u_i, w_k, w_{y_i}) + (u_i - \tilde{u}_i)^2 + \|w_{y_i} - \tilde{w}_{y_i}\|_2^2 + \|w_k - \tilde{w}_k\|_2^2 \right\} \\
&= \min_{u_i, w_k, w_{y_i}} \Bigg\{ 2\eta N(u_i + e^{-u_i} + (K-1)e^{x_i^\top(w_k - w_{y_i}) - u_i}) \\
&\qquad + (u_i - \tilde{u}_i)^2 + \|w_{y_i} - \tilde{w}_{y_i}\|_2^2 + \|w_k - \tilde{w}_k\|_2^2 \Bigg\}.
\end{aligned}$$

The optimal value of w_k, w_{y_i} must deviate from the old value $\tilde{w}_k, \tilde{w}_{y_i}$ in the direction of x_i. Furthermore we can observe that the deviation of w_k must be exactly opposite that of w_{y_i}, that is:

$$\begin{aligned}
w_{y_i} &= \tilde{w}_{y_i} + a\frac{x_i}{2\|x_i\|_2^2} \\
w_k &= \tilde{w}_k - a\frac{x_i}{2\|x_i\|_2^2}
\end{aligned} \tag{11}$$

for some $a \geq 0$. The optimization problem reduces to

$$\min_{u_i, a \geq 0} \left\{ 2\eta N(u_i + e^{-u_i} + (K-1)e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i})}e^{-a-u_i}) + (u_i - \tilde{u}_i)^2 + a^2\frac{1}{2\|x_i\|_2^2} \right\}. \tag{12}$$

We'll approach this optimization problem by first solving for a as a function of u_i and then optimize over u_i. Once the optimal value of u_i has been found, we can calculate the corresponding optimal value of a. Finally, substituting a into (11) will give us our updated value of W.

We solve for a by setting its derivative equal to zero in (12)

$$\begin{aligned}
0 &= \partial_a \left\{ 2\eta N(u_i + e^{-u_i} + (K-1)e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i})}e^{-a-u_i}) + (u_i - \tilde{u}_i)^2 + a^2\frac{1}{2\|x_i\|_2^2} \right\} \\
&= -2\eta N(K-1)e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - u_i}e^{-a} + a\frac{1}{\|x_i\|_2^2} \\
\Leftrightarrow ae^a &= 2\eta N(K-1)\|x_i\|_2^2 e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - u_i}.
\end{aligned} \tag{13}$$

The solution for a can be written in terms of the principle branch of the Lambert W function P,

$$\begin{aligned}
a(u_i) &= P(2\eta N(K-1)\|x_i\|_2^2 e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - u_i}) \\
&= P(e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - u_i + \log(2\eta N(K-1)\|x_i\|_2^2)}).
\end{aligned}$$

Substituting the solution to $a(u_i)$ into (12), we now only need minimize over u_i:

$$\begin{aligned}
&\min_{u_i} \left\{ 2\eta N u_i + 2\eta N e^{-u_i} + 2\eta N(K-1)e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i})}e^{-a(u_i)-u_i} + (u_i - \tilde{u}_i)^2 + a(u_i)^2\frac{1}{2\|x_i\|_2^2} \right\} \\
&= \min_{u_i} \left\{ 2\eta N u_i + 2\eta N e^{-u_i} + a(u_i)\|x_i\|_2^{-2} + (u_i - \tilde{u}_i)^2 + a(u_i)^2\frac{1}{2\|x_i\|_2^2} \right\}
\end{aligned} \tag{14}$$

where we used the fact that $e^{-P(z)} = P(z)/z$. The derivative with respect to u_i in (14) is

$$\begin{aligned}
&\partial_{u_i} \left\{ 2\eta N u_i + 2\eta N e^{-u_i} + a(u_i)\|x_i\|_2^{-2} + (u_i - \tilde{u}_i)^2 + a(u_i)^2\frac{1}{2\|x_i\|_2^2} \right\} \\
&= 2\eta N - 2\eta N e^{-u_i} + \partial_{u_i}a(u_i)\|x_i\|_2^{-2} + 2(u_i - \tilde{u}_i) + 2a(u_i)\partial_{u_i}a(u_i)\frac{1}{2\|x_i\|_2^2} \\
&= 2\eta N - 2\eta N e^{-u_i} - \frac{a(u_i)}{1 + a(u_i)}\|x_i\|_2^{-2} + 2(u_i - \tilde{u}_i) - \frac{a(u_i)^2}{(1 + a(u_i))\|x_i\|_2^2}
\end{aligned}$$

where to calculate $\partial_{u_i} a(u_i)$ we used the fact that $\partial_z P(z) = \frac{P(z)}{z(1+P(z))}$ and so

$$\begin{aligned}\partial_{u_i} a(u_i) &= -\frac{a(u_i)}{e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - u_i + \log(2\eta N(K-1)\|x_i\|_2^2)}(1+a(u_i))} e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - u_i + \log(2\eta N(K-1)\|x_i\|_2^2)} \\ &= -\frac{a(u_i)}{1+a(u_i)}.\end{aligned}$$

The second derivative of u_i in (14) maybe be calculated similarly as

$$\begin{aligned}&\partial_{u_i}^2 \left\{ 2\eta N u_i + 2\eta N e^{-u_i} + a(u_i)\|x_i\|_2^{-2} + (u_i - \tilde{u}_i)^2 + a(u_i)^2 \frac{1}{2\|x_i\|_2^2} \right\} \\ &= 2\eta N e^{-u_i} + \frac{a(u_i)}{(1+a(u_i))^3}\|x_i\|_2^{-2} + 2 + \frac{a(u_i)^2(2+a(u_i))}{(1+a(u_i))^3\|x_i\|_2^2}.\end{aligned}$$

Now that we have access to the first and second derivatives of u_i, we can numerically solve for u_i using techniques such as Newton's method, which can give ϵ-accurate solution in runtime $O(\log(\epsilon^{-1}))$. Since (12) is a strongly convex function, we are guaranteed to find a unique optimum for u.

Two challenges in performing the numerical optimization of u_i is finding an appropriate initial iterate $u_i^{(0)}$ and calculating P for large arguments. Setting the initial iterate to be

$$u_i^{(0)} = \max\{\tilde{u}_i, x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) + \log(2\eta N(K-1)\|x_i\|_2^2)\} \tag{15}$$

ensures that $a(u_i^{(0)}) \leq P(1) \approx 0.57$ and the objective in (14) is reasonably small. If the argument to P is large, then we can employ an asymptotic formula, such as (Corless et al., 1996)

$$P(z) \approx \log(z) - \log(\log(z)) + \frac{\log(\log(z))}{\log(z)}. \tag{16}$$

In practice we solved for u_i using Newton's method starting with the initial iterate in (15) and setting $P(z)$ to be approximated by (16) if $z > 15$. We found this to be numerically stable under nearly all conditions (except if η or $\|x_i\|$ was extremely large).

E.2 Multiple datapoints, multiple classes

The Implicit SGD update when there are multiple datapoints, multiple classes, with a regularizer is similar to the singe datapoint, singe class, no regularizer case described above. However, there are a few significant differences. Firstly, we will require some pre-computation to find a low-dimensional representation of the x values in each mini-batch. Secondly, we will integrate out u_i for each datapoint (not w_k). And thirdly, since the dimensionality of the simplified optimization problem is large, we'll require first order or quasi-Newton methods to find the optimal solution.

E.2.1 Defining the mini-batch

The first step is to define our mini-batches of size n. We will do this by partitioning the datapoint indices into sets $S_1, ..., S_J$ with $S_j = \{j_\ell : \ell = 1, ..., n\}$ for $j = 1, ..., \lfloor N/n \rfloor$, $S_J = \{J_\ell : \ell = 1, ..., N \bmod n\}$, $S_i \cap S_j = \varnothing$ and $\cup_{j=1}^J S_j = \{1, ..., N\}$.

Next we define the set of classes C_j which can be sampled for the jth mini-batch. The set C_j is defined to be all sets of m distinct classes that are not equal to any of the labels y for points in the mini-batch, that is, $C_j = \{(k_1, ..., k_m) : k_i \in \{1, ..., K\},\ k_i \neq k_\ell \,\forall \ell \in \{1, ..., m\} - \{i\},\ k_i \neq y_\ell \,\forall \ell \in S_j\}$.

Now we can write down our objective from (5) in terms of an expectation of functions corresponding to our mini-batches:

$$f(u, W) = \mathbb{E}[f_{j,C}(u, W)]$$

where j is sampled with probability $p_j = |S_j|/N$ and C is sampled uniformly from C_j and

$$f_{j,C}(u, W) = p_j^{-1} \sum_{i \in S_j} \left(u_i + e^{-u_i} + \sum_{k \in S_j - \{i\}} e^{x_i^\top (w_k - w_{y_i}) - u_i} + \frac{K-n}{m} \sum_{k \in C} e^{x_i^\top (w_k - w_{y_i}) - u_i} \right) + \frac{\mu}{2} \sum_{k \in C \cup S_j} \beta_k \|w_k\|_2^2.$$

The value of the regularizing constant β_k is such that $\mathbb{E}[I[k \in C \cup S_j]\beta_k] = 1$, which requires that

$$\beta_k^{-1} = 1 - \frac{1}{J} \sum_{j=1}^{J} I[k \neq S_j](1 - \frac{m}{K - |S_j|}).$$

E.2.2 Simplifying the Implicit SGD update equation

The Implicit SGD update corresponds to solving

$$\min_{u,W} \left\{ 2\eta f_{j,C}(u, W) + \|u - \tilde{u}\|_2^2 + \|W - \tilde{W}\|_2^2 \right\},$$

where η is the learning rate and the tilde refers to the value of the old iterate (Toulis et al., 2016, Eq. 6). Since $f_{j,C}$ is only a function of $u_{S_j} = \{u_i : i \in S_j\}$ and $W_{j,C} = \{w_k : k \in S_j \cup C\}$ the optimization reduces to

$$\min_{u_{S_j}, W_{j,C}} \left\{ 2\eta f_{j,C}(u_{S_j}, W_{j,C}) + \|u_{S_j} - \tilde{u}_{S_j}\|_2^2 + \|W_{j,C} - \tilde{W}_{j,C}\|_2^2 \right\}.$$

The next step is to analytically minimize the u_{S_j} terms. The optimization problem in (19) decomposes into a sum of separate optimization problems in u_i for $i \in S_j$,

$$\min_{u_i} \left\{ 2\eta p_j^{-1}(u_i + e^{-u_i} d_i) + (u_i - \tilde{u}_i)^2 \right\}$$

where

$$d_i(W_{j,C}) = 1 + \sum_{k \in S_j - \{i\}} e^{x_i^\top (w_k - w_{y_i})} + \frac{K-n}{m} \sum_{k \in C} e^{x_i^\top (w_k - w_{y_i})}.$$

Setting the derivative of u_i equal to zero yields the solution

$$u_i(W_{j,C}) = \tilde{u}_i - \eta p_j^{-1} + P(\eta p_j^{-1} d_i(W_{j,C}) \exp(\eta p_j^{-1} - \tilde{u}_i))$$

where P is the principle branch of the Lambert W function. Substituting this solution into our optimization problem and simplifying yields

$$\min_{W_{j,C}} \left\{ \sum_{i \in S_j} (1 + P(\eta p_j^{-1} d_i(W_{j,C}) \exp(\eta p_j^{-1} - \tilde{u}_i)))^2 + \|W_{j,C} - \tilde{W}_{j,C}\|_2^2 + \frac{\mu}{2} \sum_{k \in C \cup S_j} \beta_k \|w_k\|_2^2 \right\}, \tag{17}$$

where we have used the identity $e^{-P(z)} = P(z)/z$. We can decompose (17) into two parts by splitting $W_{j,C} = W_{j,C}^{\|} + W_{j,C}^{\perp}$, its components parallel and perpendicular to the span of $\{x_i : i \in S_j\}$ respectively. Since the leading term in (17) only depends on $W_{j,C}^{\|}$, the two resulting sub-problems are

$$\min_{W_{j,C}^{\|}} \left\{ \sum_{i \in S_j} (1 + P(\eta p_j^{-1} d_i(W_{j,C}^{\|}) \exp(\eta p_j^{-1} - \tilde{u}_i)))^2 + \|W_{j,C}^{\|} - \tilde{W}_{j,C}^{\|}\|_2^2 + \frac{\mu}{2} \sum_{k \in C \cup S_j} \beta_k \|w_k^{\|}\|_2^2 \right\},$$

$$\min_{W_{j,C}^{\perp}} \left\{ \|W_{j,C}^{\perp} - \tilde{W}_{j,C}^{\perp}\|_2^2 + \frac{\mu}{2} \sum_{k \in C \cup S_j} \beta_k \|w_k^{\perp}\|_2^2 \right\} \tag{18}$$

Let us focus on the perpendicular component first. Simple calculus yields the optimal value $w_k^\perp = \frac{1}{1+\mu\beta_k/2}\tilde{w}_k^\perp$ for $k \in S_j \cup C$.

Moving onto the parallel component, let the span of $\{x_i : i \in S_j\}$ have an orthonormal basis[9] $V_j = (v_{j1}, ..., v_{jn}) \in \mathbb{R}^{D\times n}$ with $x_i = V_j b_i$ for some $b_i \in \mathbb{R}^n$. With this basis we can write $w_k^\parallel = \tilde{w}_k^\parallel + V_j a_k$ for $a_k \in \mathbb{R}^n$ which reduces the parallel component optimization problem to[10]

$$\min_{A_{j,C}} \left\{ \sum_{i\in S_j} (1 + P(z_{ijC}(A_{j,C})))^2 + \sum_{k\in S_j\cup C} (1 + \frac{\mu\beta_k}{2})\|a_k\|_2^2 + \mu\beta_k \tilde{w}_k^\top V_j a_k \right\}, \tag{19}$$

where $A_{j,C} = \{a_k : k \in S_j \cup C\} \in \mathbb{R}^{(n+m)\times n}$ and

$$z_{ijC}(A_{j,C}) = \eta p_j^{-1} \exp(\eta p_j^{-1}) \Bigg(\exp(-\tilde{u}_i) + \sum_{k\in S_j - \{i\}} e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - \tilde{u}_i} e^{b_i^\top (a_k - a_{y_i})} + \frac{K-n}{m} \sum_{k\in C} e^{x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - \tilde{u}_i} e^{b_i^\top (a_k - a_{y_i})} \Bigg).$$

The $e^{b_i^\top(a_k - a_{y_i})}$ factors come from

$$\begin{aligned} x_i^\top w_k &= x_i^\top (\tilde{w}_k^\parallel + a_k^\top V_j) \\ &= x_i^\top \tilde{w}_k + (V_j b_i)^\top V_j a_k \\ &= x_i^\top \tilde{w}_k + b_i^\top V_j^\top V_j a_k \\ &= x_i^\top \tilde{w}_k + b_i^\top a_k, \end{aligned}$$

since V_j is an orthonormal basis.

E.2.3 Optimizing the Implicit SGD update equation

To optimize (19) we need to be able to take the derivative:

$$\begin{aligned} &\nabla_{a_\ell} \left(\sum_{i\in S_j} (1 + P(z_{ijC}(A_{j,C})))^2 + \sum_{k\in S_j\cup C} (1 + \frac{\mu\beta_k}{2})\|a_k\|_2^2 + \mu\beta_k \tilde{w}_k^\top V_j a_k \right) \\ &= \sum_{i\in S_j} 2(1 + P(z_{ijC}(A_{j,C})))\partial_{z_{ijC}(A_{j,C})} P(z_{ijC}(A_{j,C})) \nabla_{a_\ell} z_{ijC}(A_{j,C}) \\ &\qquad + (2 + \mu\beta_\ell)a_\ell + \mu\beta_\ell \tilde{w}_\ell^\top V_j \\ &= \sum_{i\in S_j} 2(1 + P(z_{ijC}(A_{j,C}))) \frac{P(z_{ijC}(A_{j,C}))}{z_{ijC}(A_{j,C})(1 + P(z_{ijC}(A_{j,C})))} \nabla_{a_\ell} z_{ijC}(A_{j,C}) \\ &\qquad + (2 + \mu\beta_\ell)a_\ell + \mu\beta_\ell \tilde{w}_\ell^\top V_j \\ &= \sum_{i\in S_j} 2\frac{P(z_{ijC}(A_{j,C}))}{z_{ijC}(A_{j,C})} \nabla_{a_\ell} z_{ijC}(A_{j,C}) + (2 + \mu\beta_\ell)a_\ell + \mu\beta_\ell \tilde{w}_\ell^\top V_j \\ &= \sum_{i\in S_j} 2e^{-P(z_{ijC}(A_{j,C}))} \nabla_{a_\ell} z_{ijC}(A_{j,C}) + (2 + \mu\beta_\ell)a_\ell + \mu\beta_\ell \tilde{w}_\ell^\top V_j \end{aligned}$$

[9] We have assumed here that $dim(span(\{x_i : i \in S_j\})) = n$, which will be most often the case. If the dimension of the span is lower than n then let V_j be of dimension $D \times dim(span(\{x_i : i \in S_j\}))$.

[10] Note that we have used $\tilde{w}_k$ instead of $\tilde{w}_k^\parallel$ in writing the parallel component optimization problem. This does not make a difference as $\tilde{w}_k$ always appears as an inner product with a vector in the span of $\{x_i : i \in S_j\}$.

where we used that $\partial_z P(z) = \frac{P(z)}{z(1+P(z))}$ and $e^{-P(z)} = P(z)/z$. To complete the calculation of the derivate we need,

$$\begin{aligned}
\nabla_{a_\ell} z_{ijC}(A_{j,C}) = \nabla_{a_\ell} \eta p_j^{-1} \exp(\eta p_j^{-1}) \Bigg(& \exp(-\tilde{u}_i) + \sum_{k \in S_j - \{i\}} e^{x_i^\top (\tilde{w}_\ell - \tilde{w}_{y_i}) - \tilde{u}_i} e^{b_i^\top (a_\ell - a_{y_i})} \\
& + \frac{K-n}{m} \sum_{k \in C} e^{x_i^\top (\tilde{w}_\ell - \tilde{w}_{y_i}) - \tilde{u}_i} e^{b_i^\top (a_\ell - a_{y_i})} \Bigg) \\
= \eta p_j^{-1} \exp(\eta p_j^{-1}) b_i & \\
\cdot \Bigg(& I[\ell \in S_j - \{i\}] e^{x_i^\top (\tilde{w}_\ell - \tilde{w}_{y_i}) - \tilde{u}_i} e^{b_i^\top (a_\ell - a_{y_i})} \\
& + I[\ell \in C] \frac{K-n}{m} e^{x_i^\top (\tilde{w}_\ell - \tilde{w}_{y_i}) - \tilde{u}_i} e^{b_i^\top (a_\ell - a_{y_i})} \\
& - I[\ell = y_i] \Bigg(\sum_{k \in S_j - \{i\}} e^{x_i^\top (\tilde{w}_\ell - \tilde{w}_{y_i}) - \tilde{u}_i} e^{b_i^\top (a_\ell - a_{y_i})} \\
& \qquad + \frac{K-n}{m} \sum_{k \in C} e^{x_i^\top (\tilde{w}_\ell - \tilde{w}_{y_i}) - \tilde{u}_i} e^{b_i^\top (a_\ell - a_{y_i})} \Bigg) \Bigg).
\end{aligned}$$

In order to calculate the full derivate with respect to $A_{j,C}$ we need to calculate $b_i^\top a_k$ for all $i \in S_j$ and $k \in S_j \cup C$. This is a total of $n(n+m)$ inner products of n-dimensional vectors, costing $O(n^2(n+m))$. To find the optimum of (19) we can use any optimization procedure that only uses gradients, such as L-BFGS. Since (19) is strongly convex, standard first order methods can solve to ϵ accuracy in $O(\epsilon^{-1})$ iterations. Thus once we can calculate all of the terms in (19), we can solve it to ϵ accuracy in runtime $O(n^2(n+m)\epsilon^{-1})$.

Once we have solved for $A_{j,C}$, we can reconstruct the optimal solution for the parallel component of w_k as $w_k^\parallel = \tilde{w}_k^\parallel + V_j a_k$. Recall that the solution to the perpendicular component is $w_k^\perp = \frac{1}{1+\mu\beta_k/2}\tilde{w}_k^\perp$. Thus our optimal solution is $w_k = \tilde{w}_k^\parallel + V_j a_k + \frac{1}{1+\mu\beta_k/2}\tilde{w}_k^\perp$.

If the features x_i are sparse, then we'd prefer to do a sparse update to w, saving computation time. We can achieve this by letting

$$w_k = \gamma_k \cdot r_k$$

where γ_k is a scalar and r_k a vector. Updating $w_k = \tilde{w}_k^\parallel + V_j a_k + \frac{1}{1+\mu\beta_k/2}\tilde{w}_k^\perp$ is equivalent to

$$\begin{aligned}
\gamma_k &= \tilde{\gamma}_k \cdot \frac{1}{1+\mu\beta_k/2} \\
r_k &= \tilde{r}_k + \mu\beta_k/2 \cdot \tilde{r}_k^\parallel + \tilde{\gamma}_k^{-1}(1+\mu\beta_k/2) \cdot V_j a_k.
\end{aligned}$$

Since we only update r_k along the span of $\{x_i : i \in S_j\}$, its update is sparse.

E.2.4 Runtime

There are two major tasks in calculating the terms in (19). The first is to calculate $x_i^\top \tilde{w}_k$ for $i \in S_j$ and $k \in S_j \cup C$. There are a total of $n(n+m)$ inner products of D-dimensional vectors, costing $O(n(n+m)D)$. The other task is to find the orthonormal basis V_j of $\{x_i : i \in S_j\}$, which can be achieved using the Gram-Schmidt process in $O(n^2 D)$. We'll assume that $\{V_j : j = 1, ..., J\}$ is computed only once as a pre-processing step when defining the mini-batches. It is exactly because calculating $\{V_j : j = 1, ..., J\}$ is expensive that we have fixed mini-batches that do not change during the optimization routine.

Adding the cost of calculating the $x_i^\top \tilde{w}_k$ inner products to the costing of optimizing (19) leads to the claim that solve the Implicit SGD update formula to ϵ accuracy in runtime $O(n(n+m)D + n^2(n+m)\epsilon^{-1}) = O(n(n+m)(D+n\epsilon^{-1}))$.

If n and m are small then instead of optimizing the Implicit SGD updates using a first order method, we can calculate the Hessian and solve for $A_{j,C}$ using second order methods. These methods, e.g.

Newton's method, give $O(\epsilon)$ accuracy in $O(\log(\epsilon^{-1}))$ iterations. In the special case of Proposition 1 where $n = m = 1$ the runtime of solving the Implicit SGD update to ϵ accuracy becomes $O(D + \log(\epsilon^{-1}))$, where $O(D)$ comes from calculating the $x_i^\top w_k$ and $x_i^\top w_{y_i}$ inner products.

E.2.5 INITIALIZING THE IMPLICIT SGD OPTIMIZER

As was the case in Section E.1, it is important to initialize the optimization procedure at a point where the gradient is relatively small and can be computed without numerical issues. These numerical issues arise when an exponent $x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - \tilde{u}_i + b_i^\top(a_k - a_{y_i}) \gg 0$. To ensure that this does not occur for our initial point, we can solve the following linear problem,[11]

$$\begin{aligned} R = \min_{A_{j,C}} \quad & \sum_{k \in C \cup S_j} \|a_k\|_1 \\ s.t. \quad & x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - \tilde{u}_i + b_i^\top(a_k - a_{y_i}) \leq 0 \quad \forall i \in S_j,\, k \in C \cup S_j \end{aligned} \tag{20}$$

Note that if $k = y_i$ then the constraint $0 \geq x_i^\top(\tilde{w}_k - \tilde{w}_{y_i}) - \tilde{u}_i + b_i^\top(a_k - a_{y_i}) = -\tilde{u}_i$ is automatically fulfilled since $\tilde{u}_i \geq 0$. Also observed that setting $a_k = -V_j^\top \tilde{w}_k$ satisfies all of the constraints, and so

$$R \leq \sum_{k \in C \cup S_j} \|V_j^\top \tilde{w}_k\|_1 \leq (n+m) \max_{k \in C \cup S_j} \|V_j^\top \tilde{w}_k\|_1.$$

We can use the solution to (20) to gives us an upper bound on (19). Consider the optimal value $A_{j,C}^{(R)}$ of the linear program in (20) with the value of the minimum being R. Since $A_{j,C}^{(R)}$ satisfies the constrain in (20) we have $z_{ijC}(A_{j,C}^{(R)}) \leq K\eta p_j^{-1} \exp(\eta p_j^{-1})$. Since $P(z)$ is a monotonically increasing function that is non-negative for $z \geq 0$ we also have $(1 + P(z_{ijC}(A_{j,C}^{(R)})))^2 \geq (1 + P(K\eta p_j^{-1} \exp(\eta p_j^{-1})))^2$. Turning to the norms, we can use the fact that $\|a\|_2 \leq \|a\|_1$ for any vector a to bound

$$\begin{aligned}
& \sum_{k \in S_j \cup C} (1 + \frac{\mu \beta_k}{2}) \|a_k\|_2^2 + \mu \beta_k \tilde{w}_k^\top V_j a_k \\
& \leq \sum_{k \in S_j \cup C} (1 + \frac{\mu \beta_k}{2}) \|a_k\|_1^2 + \mu \beta_k \|\tilde{w}_k^\top V_j\|_1 \|a_k\|_1 \\
& \leq \left(1 + \mu \cdot \max_{k \in S_j \cup C} \{\beta_k\}/2\right) \sum_{k \in S_j \cup C} \|a_k\|_1^2 + \mu \max_{k \in S_j \cup C} \{\beta_k \|\tilde{w}_k^\top V_j\|_1\} \sum_{k \in S_j \cup C} \|a_k\|_1 \\
& \leq \left(1 + \mu \cdot \max_{k \in S_j \cup C} \{\beta_k\}/2\right) R^2 + \mu \max_{k \in S_j \cup C} \{\beta_k\} \max_{k \in S_j \cup C} \{\|\tilde{w}_k^\top V_j\|_1\} R \\
& \leq \left(1 + \mu \cdot \max_{k \in S_j \cup C} \{\beta_k\}/2\right) \left((n+m) \max_{k \in C \cup S_j} \|V_j^\top \tilde{w}_k\|_1\right)^2 \\
& \qquad + \mu \max_{k \in S_j \cup C} \{\beta_k\} \max_{k \in S_j \cup C} \{\|\tilde{w}_k^\top V_j\|_1\} \left((n+m) \max_{k \in C \cup S_j} \|V_j^\top \tilde{w}_k\|_1\right) \\
& \leq (1 + \mu \cdot \max_{k \in S_j \cup C} \{\beta_k\})(n+m)^2 \max_{k \in C \cup S_j} \|V_j^\top \tilde{w}_k\|_1^2 \\
& \leq (1 + \mu \cdot \max_{k \in S_j \cup C} \{\beta_k\})(n+m)^2 \max_{k \in C \cup S_j} \|\tilde{w}_k\|_1^2.
\end{aligned}$$

Putting the bounds together we have that the optimal value of (19) is upper bounded by its value at the solution to (20), which in turn is upper bounded by

$$n(1 + P(K\eta p_j^{-1} \exp(\eta p_j^{-1})))^2 + (1 + \mu \cdot \max_{k \in S_j \cup C} \{\beta_k\})(n+m)^2 \max_{k \in C \cup S_j} \|\tilde{w}_k\|_1^2.$$

This bound is guarantees that our initial iterate will be numerically stable.

[11] Instead bounding the constraints on the right with 0, we could also have used any small positive number, like 5.

F LEARNING RATE PREDICTION AND LOSS

Here we present the results of using different learning rates for each algorithm applied to the Eurlex dataset. In addition to the Implicit, NCE, IS, OVE and U-max algorithms whose configuration is described in Section 4, we also provide results for NCE with $n = 1, m = 1$, denoted as NCE (1,1) and U-max run on the alternative double-sum formulation discussed in Section 2.3, denoted as U-max (2). NCE and NCE (1,1) have near identical performance, whereas U-max (2) is significantly worse than U-max.

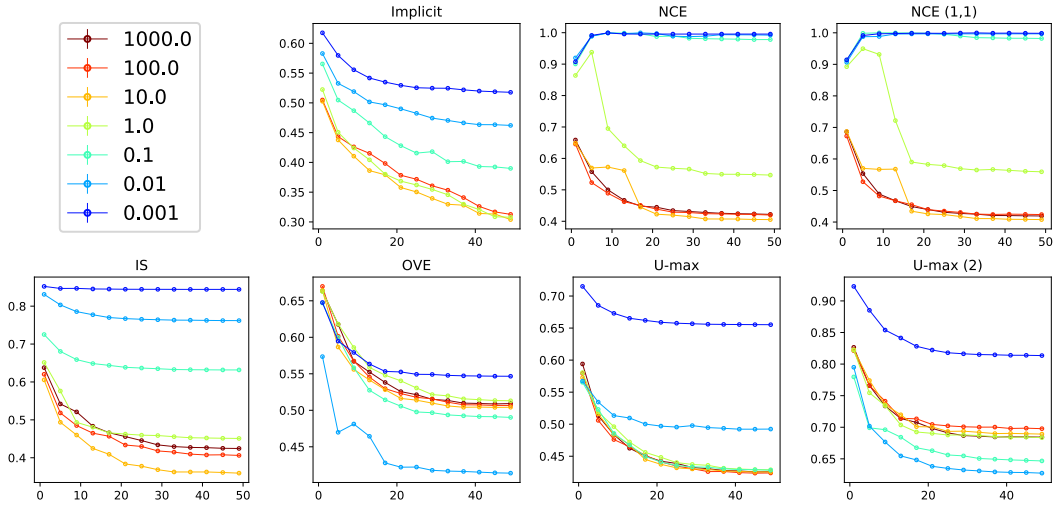

Figure 4: Prediction error on Eurlex for different learning rates.

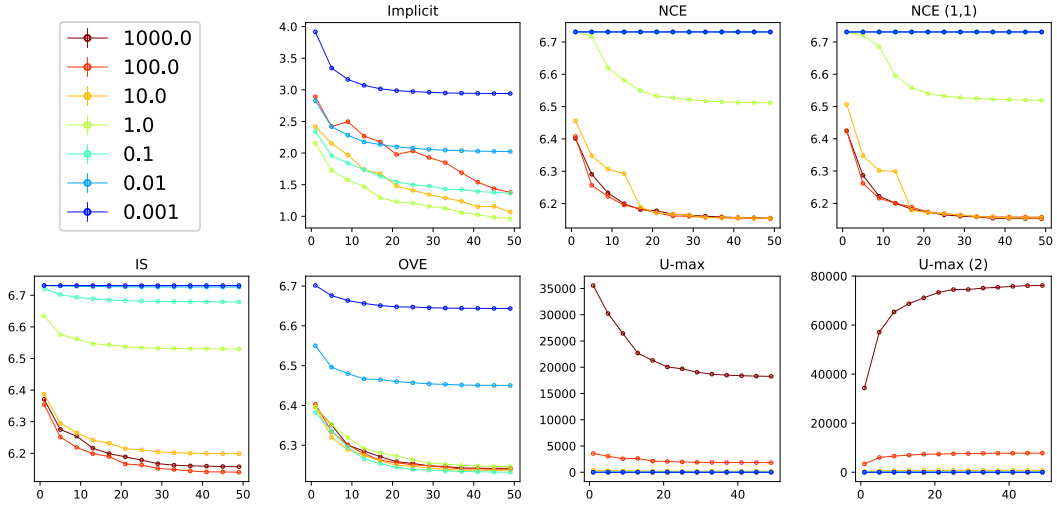

Figure 5: Prediction loss on Eurlex different learning rates.

Table 3: Optimal learning rate for each algorithm for the Eurlex log-loss.

ALGORITHM	LEARNING RATE
Implicit	1.0
NCE	100
NCE (TF)	100
IS	100
OVE	0.1
U-max	0.1
U-max (2)	0.01