# Inverse Classification with Softmax Classifiers: Efficient Optimization

**Miguel Á. Carreira-Perpiñán** [1] **Suryabhan Singh Hada** [1]

## Abstract

In recent years, a certain type of problems have become of interest where one wants to query a trained classifier. Specifically, one wants to find the closest instance to a given input instance such that the classifier's predicted label is changed in a desired way. Examples of these "inverse classification" problems are counterfactual explanations, adversarial examples and model inversion. All of them are fundamentally optimization problems over the input instance vector involving a fixed classifier, and it is of interest to achieve a fast solution for interactive or real-time applications. We focus on solving this problem efficiently for two of the most widely used classifiers: logistic regression and softmax classifiers. Owing to special properties of these models, we show that the optimization can be solved in closed form for logistic regression, and iteratively but extremely fast for the softmax classifier using Newton's method. This allows us to solve either case exactly (to nearly machine precision) in a runtime of milliseconds to around a second even for very high-dimensional instances and many classes.

## 1. Introduction and related work

In an abstract sense, machine learning classifiers can be regarded as manipulating three objects: the input feature vector $\mathbf{x}$, the output label $y$ and the classifier model $f$. *Training* or *learning* $(\mathbf{x}, y \to f)$ is the problem of inferring $f$ from $\mathbf{x}$ and $y$ (the training set), and is usually formulated as an optimization problem. *Inference* or *prediction* $(\mathbf{x}, f \to y)$ is the problem of inferring $y$ from $\mathbf{x}$ and $f$ (the training set), and is usually formulated via the application of an explicit function $f$ to $\mathbf{x}$. Both training and inference have been long studied for a wide range of classifiers. The third problem $(f, y \to \mathbf{x})$, which can be called *inverse classification*, is to find $\mathbf{x}$ from $y$ and $f$, and can also be formulated as an optimization problem. This problem is far less popular than the other two but has received significant attention in recent years in specific settings. One of them is in *adversarial examples* (Szegedy et al., 2014; Goodfellow et al., 2015). We are given a classifier $f$, such as a neural net, and an input instance $\mathbf{x}$ which is classified by $f$ as (say) class 1. The goal is to perturb $\mathbf{x}$ slightly (or equivalently to find a close instance to $\mathbf{x}$) such that it is then classified by $f$ as (say) class 2. The motivation is to trick the classifier into predicting another class and trigger some action (such as having a self-driving car misclassify a stop sign as a yield sign, say). Another setting is in *counterfactual explanations* (Wachter et al., 2018; Ustun et al., 2019; Russell, 2019; Karimi et al., 2019). Again, we are given a classifier $f$ and an input instance $\mathbf{x}$ which is classified by $f$ as (say) class 1, which is undesirable. The goal is to change $\mathbf{x}$ in a minimally costly way so that it is classified as (say) class 2, which is desirable. As an example, $\mathbf{x}$ could represent a loan applicant (age, salary, etc.) and $f$ could predict whether to approve or not the loan. *Activation maximization* (Simonyan et al., 2014; Zeiler & Fergus, 2014; Mahendran & Vedaldi, 2016; Dosovitskiy & Brox, 2016; Nguyen et al., 2016; Hada & Carreira-Perpiñán, 2019) is another setting, where the goal is to understand the inner workings of a deep net by seeking input vectors that cause a particular neuron in the net to have a large output. Problems of these types have become particularly important in the context of transparency in AI, where users may want to interpret, explain, understand, query or manipulate a trained classifier.

Such inverse classification problems can be naturally formulated as an optimization involving a distance or cost in feature space and a loss in label space. Many recent works exist on adversarial examples and counterfactual explanations, but they have mostly focused in their motivation or application, rather than on optimization issues—indeed, these works (often based on deep nets) typically use gradient descent as provided by existing ML frameworks such as scikit-learn, PyTorch or TensorFlow. Here we are interested in the numerical optimization aspects of such problems, which we recognize as a novel type of optimization problem with specific characteristics that deserves special attention. Such problems are computationally much

[1]EECS, UC Merced. Correspondence to: Miguel Á. Carreira-Perpiñán <mcarreira-perpinan@ucmerced.edu>, Suryabhan Singh Hada <shada@ucmerced.edu>.

*Workshop on "Beyond first-order methods in ML systems" at the $37^{th}$ International Conference on Machine Learning*, Vienna, Austria, 2020. 

smaller than the training problem (which involves a dataset and model parameters as variables), but it is still important to solve them fast: some of the applications mentioned above require real-time or interactive processing, and this processing may often take place in limited-computation devices such as mobile phones. Besides, the optimization can be nontrivial if the number of features is in the thousands or milions.

In this paper, we consider logistic regression and softmax classifiers, which are among the simplest yet most widely used classifiers in practice; and the Euclidean distance as cost. We will define inverse classification as a natural optimization problem, characterize it in theory and provide what probably are the most efficient solutions possible for either type of classifier.

## 2. Inverse classification as optimization: softmax classifier

Consider a $K$-class softmax classifier, where the softmax probability of class $i \in \{1,\dots,K\}$ for instance $\mathbf{x} \in \mathbb{R}^D$ is defined as $\mathbf{p}(\mathbf{x}) = (p_1(\mathbf{x}),\dots,p_K(\mathbf{x}))^T$ with

$$p_i(\mathbf{x}) = e^{z_i} \big/ \textstyle\sum_{j=1}^K e^{z_j},\ i \in \{1,\dots,K\} \qquad (1)$$

where $\mathbf{z} = \mathbf{A}\mathbf{x} + \mathbf{b} \in \mathbb{R}^K$ and the parameters $\mathbf{A} \in \mathbb{R}^{K\times D}$ and $\mathbf{b} \in \mathbb{R}^{D\times 1}$ are a matrix of weights and a vector of biases, respectively. Throughout this paper, we assume the parameters (hence the classifier) are fixed, presumably by having been learned on a training set. Obviously, each $p_i$ value is in (0,1) and their sum is 1. Define the function $g_i(\mathbf{x}) = -\ln p_i(\mathbf{x}) > 0$. Write $\mathbf{A}^T = (\mathbf{a}_1 \cdots \mathbf{a}_K)$ where each $\mathbf{a}_i$ is of $D \times 1$ ($i$th row of $\mathbf{A}$, transposed). Define $\bar{\mathbf{A}}_k = \mathbf{A} - \mathbf{1}\mathbf{a}_k^T$ of $K \times D$, where $\mathbf{1}$ is a vector of ones (and likewise $\overline{\mathbf{a}}_{ki} = \mathbf{a}_i - \mathbf{a}_k$) and $\overline{\mathbf{b}}_k = \mathbf{b} - b_k\mathbf{1}$ of $K \times 1$. Note that $\bar{\mathbf{A}}_k$ has row $k$ of zeros.

**Theorem 2.1.** *For each $i \in \{1,\dots,K\}$ we have that $g_i$ is convex (but not strongly convex).*

We define the optimization problem[1]

$$\min_{\mathbf{x}\in\mathbb{R}^D} E(\mathbf{x};\lambda,k) = \tfrac{\lambda}{2}\|\mathbf{x}-\overline{\mathbf{x}}\|^2 + g_k(\mathbf{x}) \qquad (2)$$

where $\overline{\mathbf{x}} \in \mathbb{R}^D$ is the *source instance*, $k \in \{1,\dots,K\}$ the *target class* and $\lambda > 0$ trades off distance to $\overline{\mathbf{x}}$ with class-$k$ probability. Next, we give several results about the problem and about Newton's method and discuss their implications in section 2.1. An alternative formulation of interest is "$\min_{\mathbf{x}} \|\mathbf{x}-\overline{\mathbf{x}}\|$ s.t. $g_k(\mathbf{x}) \geq \alpha$". However, this is a harder optimization problem, and besides we can solve it by trying different values of $\lambda$ in (2). This can be done efficiently by scanning a solution path over $\lambda$ via warm-start, and it also has the advantage of providing a range of solutions (and their corresponding probabilities).

[1] All norms are $\ell_2$ norms throughout the paper.

**Theorem 2.2.** *$E(\mathbf{x};\lambda,k)$ is strongly convex. Hence, it has a unique minimizer $\mathbf{x}^*$.*

**Gradient and Hessian** It is easy to see that the gradient and Hessian of $E$ wrt $\mathbf{x}$ are:

$$\begin{aligned}\nabla_{\mathbf{x}} E(\mathbf{x};\lambda,k) &= \lambda(\mathbf{x}-\overline{\mathbf{x}}) + \bar{\mathbf{A}}_k^T\mathbf{p}(\mathbf{x}) \\ \nabla^2_{\mathbf{x}\mathbf{x}} E(\mathbf{x};\lambda,k) &= \lambda\mathbf{I} + \bar{\mathbf{A}}_k^T(\operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T)\bar{\mathbf{A}}_k. \end{aligned} \qquad (3)$$

To avoid clutter, we will usually omit the dependence on $\mathbf{x}, \lambda, k$. We will also call the matrix $\bar{\mathbf{A}}_k^T(\operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T)\bar{\mathbf{A}}_k$ the "softmax Hessian" .

**Theorem 2.3.** *The Hessian $\nabla^2 E(\mathbf{x})$ satisfies the following $\forall \mathbf{x} \in \mathbb{R}^D$: it is positive definite; its largest eigenvalue and condition number are upper bounded by $\lambda + \|\bar{\mathbf{A}}_k\|^2$ and $1 + \|\bar{\mathbf{A}}_k\|^2/\lambda$, respectively; all its $D$ eigenvalues are greater or equal than $\lambda$; if $K \leq D$ then at least $D-K+1$ eigenvalues equal $\lambda$.*

**Theorem 2.4.** *The gradient $\nabla E(\mathbf{x})$ is Lipschitz continuous with Lipschitz constant $L = \lambda + \|\bar{\mathbf{A}}_k\|^2$.*

**Theorem 2.5.** *If $\lambda \gg 1$ then $\nabla^2 E(\mathbf{x}) \approx \lambda\mathbf{I}\ \forall \mathbf{x} \in \mathbb{R}^D$. If $\lambda \ll 1$ then $\nabla^2 E(\mathbf{x}^*) \approx \lambda\mathbf{I}$.*

**Theorem 2.6.** *The inverse Hessian can be written as:*

$$\begin{aligned}\left(\nabla^2 E\right)^{-1} &= \left(\lambda\mathbf{I} + \bar{\mathbf{A}}_k^T(\operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^T)\bar{\mathbf{A}}_k\right)^{-1} \\ &= \mathbf{H}^{-1} + \frac{(\mathbf{H}^{-1}\mathbf{v})(\mathbf{H}^{-1}\mathbf{v})^T}{1 - \mathbf{v}^T\mathbf{H}^{-1}\mathbf{v}} \end{aligned} \qquad (4)$$

*where* $\mathbf{v} = \bar{\mathbf{A}}_k^T\mathbf{p}$ *and* $\mathbf{H}^{-1} = \frac{1}{\lambda}\left(\mathbf{I} - \bar{\mathbf{A}}_k^T\left(\bar{\mathbf{A}}_k\bar{\mathbf{A}}_k^T + \lambda\operatorname{diag}(\mathbf{p})^{-1}\right)^{-1}\bar{\mathbf{A}}_k\right)$.

Although the expressions above look complicated, they reduce the computation of the Newton direction by requiring a $K \times K$ inverse (of $\bar{\mathbf{A}}_k\bar{\mathbf{A}}_k^T + \lambda\operatorname{diag}(\mathbf{p})^{-1}$), rather than a $D \times D$ one (of $\nabla^2 E$). Since typically $K \ll D$, this is an enormous savings. Unfortunately, it is not possible to speed this up even further and avoid inverses altogether by caching a factorization of $\bar{\mathbf{A}}_k\bar{\mathbf{A}}_k^T + \lambda\operatorname{diag}(\mathbf{p})^{-1}$, because it is a diagonal update. Note that we never actually invert any matrix: for reasons of numerical stability and efficiency, expressions of the form $\mathbf{A}^{-1}\mathbf{b}$ are computed by solving a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ rather than by explicitly computing $\mathbf{A}^{-1}$ and multiplying it times $\mathbf{b}$ (in Matlab, `x = A\b` rather than `x = inv(A)*b`).

**Convergence and rate of convergence** Theorem 2.8 states that Newton's method with a line search (l.s.) will converge to the minimizer of $E$ from any starting point (global convergence). It is a consequence of our results above and the following theorem.

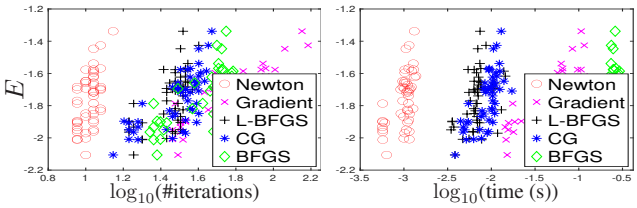


*Figure 1.* Performance of different optimization methods on a collection of 50 problem instances for MNIST, in number of iterations (left) and runtime in seconds (right).

**Theorem 2.7.** *Consider a function $f\colon \mathbb{R}^D \to \mathbb{R}$, not necessarily convex, bounded below and continuously differentiable in $\mathbb{R}^D$, and with L-Lipschitz continuous gradient (i.e., $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x}-\mathbf{y}\|\ \forall \mathbf{x},\mathbf{y} \in \mathbb{R}^D$ and $L > 0$). Consider an iteration of the form $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$, where $\mathbf{x}_0 \in \mathbb{R}^D$ is any starting point, $\mathbf{s}_k$ is a descent direction (i.e., $\mathbf{s}_k^T \nabla f(\mathbf{x}_k) < 0$ if $\nabla f(\mathbf{x}_k) \neq \mathbf{0}$) and the step size $\alpha_k$ satisfies the Wolfe conditions. Then $\sum_{k\geq 0} \cos^2\theta_k \|\nabla f(\mathbf{x}_k)\|^2 < \infty$, where $\cos\theta_k = -\mathbf{s}_k^T \nabla f(\mathbf{x}_k)/\|\mathbf{s}_k\|\|\nabla f(\mathbf{x}_k)\|$.*
*Now assume $\mathbf{s}_k = -\mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k)$ where, for each $k = 0, 1, 2\ldots$, $\mathbf{B}_k$ is positive definite and $\operatorname{cond}(\mathbf{B}_k) \leq M$ for some $M > 0$. Then $\|\nabla f(\mathbf{x}_k)\| \to \mathbf{0}$ as $k \to \infty$.*

**Theorem 2.8.** *Consider the objective function $E(\mathbf{x})$ of eq. (2) and an iteration of the form $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \left(\nabla^2 E(\mathbf{x}_k)\right)^{-1} \nabla E(\mathbf{x}_k)$ for $k = 0, 1, 2\ldots$, where $\mathbf{x}_0 \in \mathbb{R}^D$ is any starting point and the step size $\alpha_k$ satisfies the Wolfe conditions. Then $\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}^*$, the unique minimizer of $E$, and $\lim_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = \mathbf{0}$.*

Further, theorem 3.5 in Nocedal & Wright (2006) states that, if the starting point $\mathbf{x}_0$ is sufficiently close to $\mathbf{x}^*$ (which can always be achieved by the l.s.) and if using unit step sizes, then both the sequences of iterates $\{\mathbf{x}_k\}$ and of gradients $\{\nabla E(\mathbf{x}_k)\}$ converge *quadratically* to $\mathbf{x}^*$ and $\mathbf{0}$, respectively. Although the theorem we give requires a l.s. with Wolfe conditions, a backtracking search that always tries the unit step size first also works, because it is sufficient to take us near enough the minimizer and then using unit step sizes converges quadratically.

For gradient descent, global convergence is also assured with a l.s. (by taking $\mathbf{B}_k = \mathbf{I}$ in theorem 2.7) or with a fixed step size of $1/L$ (Nesterov, 2004). For other algorithms (Polak-Ribière CG, and (L-)BFGS) it is harder to give robust global convergence results (Nocedal & Wright, 2006), but in any case they are inferior to Newton's method in our case.

**Computational complexity** For Newton's method, the dominant costs are as follows. There is a setup cost of $\mathcal{O}(DK^2)$ to compute $\bar{\mathbf{A}}_k \bar{\mathbf{A}}_k^T$. Per iteration, we have a cost of $\mathcal{O}(D(2K+1))$ for the gradient, $\mathcal{O}(2K^3 + 6KD)$ for the Newton direction, and $\mathcal{O}(D(K+1))$ per backtracking step; our experiments show that most iterations use a single backtracking step. If $D \gg K$ one Newton iteration costs about 3 gradient computations.

### 2.1. Discussion

Although the objective function $E$ is nonlinear, the special structure of its Hessian makes minimizing it very effective. Firstly, since $E$ is strongly convex, there is a unique solution no matter the value of $\lambda$ or the target class $k$. Second, if $K < D$ (which is the typical case in practice, fewer classes than features) then the Hessian has $D - K + 1$ eigenvalues equal to $\lambda$ and $K - 1$ greater or equal than $\lambda$. This means that the Hessian is much better conditioned than it would otherwise be; it is "round" in all but $K - 1$ directions, and all optimization methods will benefit from that. That said, using the curvature information is still important to take long steps. Finally, the Hessian is actually round everywhere if $\lambda \gg 1$ or near the minimizer if $\lambda \ll 1$.

This special structure also makes Newton's method unusually convenient. In practice, Newton's method is rarely applicable in direct form for two important reasons: 1) the Hessian is often not positive definite and hence can create directions that are not descent, so it needs modification, which is computationally costly (e.g. it may require factorizing it). 2) Computing the Newton direction requires computing the Hessian (in $\mathcal{O}(D^2)$ time and memory) and solving a linear system based on it (in $\mathcal{O}(D^3)$ time). To make this feasible with large $D$ requires approximating Newton's method, which leads to taking worse steps and losing its quadratic convergence properties. None of this is a problem here: our Hessian is positive definite everywhere, and computing the Newton direction (via theorem 2.6) requires a linear system of a $K \times K$ matrix, where $K$ is in practice small (less than 100 is the vast majority of applications), without ever forming a $D \times D$ matrix, which then scales to very large $D$. Finally, while Newton's method still requires a line search for the step size to ensure the iterates descend, near the minimizer a unit step works and leads to quadratic convergence. As a result we achieve the best of all possible worlds: global convergence (from any starting point); very fast iterations, scalable to high feature dimensionalities and many classes; and quadratic convergence order, which means it is possible to reach near machine-precision accuracy.

This argument strongly suggests that no other (first-order) method can compete with Newton's method in this case, particularly if we seek a highly accurate solution (unless we use a huge number of classes $K$), and this is clearly seen in our experiments. The runtime ranges from milliseconds to a second over problems with feature dimension ranging from $10^3$ to $10^5$ and tens of classes.

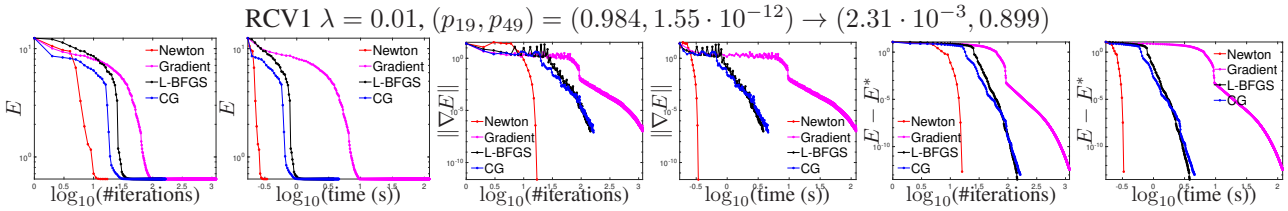


*Figure 2.* Learning curves for one problem instance of RCV1 dataset indicated by its $\lambda$ value and class probabilities, encoded as $(p_{\overline{k}}, p_k) = (\text{at } \overline{\mathbf{x}}) \rightarrow (\text{at } \mathbf{x}^*)$. Columns 1–2, 3–4 and 5–6 show $E(\mathbf{x}_k)$, $\|\nabla E(\mathbf{x}_k)\|$ and $E(\mathbf{x}_k) - E(\mathbf{x}^*)$, respectively (where $\mathbf{x}^*$ is estimated by the last iterate of Newton's method). Within each pair of columns, we show number of iterations (left) and runtime in seconds (right). All plots are log-log.

## 3. Inverse classification as optimization: logistic regression

Logistic regression corresponds to the case $K = 2$ of the softmax classifier. All the previous results apply, but in this case we can find $\mathbf{x}^*$ in closed form without the need of iterative optimization[2]

## 4. Experiments: $K$-class softmax

Our experiments are very straightforward and agree well with the theory. In short, Newton's method consistently achieves the fastest runtime of any algorithm, by far, while reaching the highest precision possible. We solve problems of the form (2) on several datasets: MNIST ($D = 784$, $K = 10$), of handwritten digit images with grayscale pixel features (LeCun et al., 1998) and RCV1 ($D = 47\,236$, $K = 51$), of documents with TFIDF features (Lewis et al., 2004). For each dataset we trained a softmax classifier using `scikit-learn` (Pedregosa et al., 2011).

We evaluated several well-known optimization methods, and for each we use the line search that we found worked best. For Newton's method, BFGS and L-BFGS we use a backtracking l.s. with initial step 1 and backtracking factor $\rho = 0.8$. For gradient descent (GD) and Polak-Ribière conjugate gradient (CG), we use a more sophisticated l.s. that allows steps longer than 1 and ensures the Wolfe conditions hold (algorithm 3.5 in (Nocedal & Wright, 2006)). All methods were implemented by us in Matlab except for CG, which uses Carl Rasmussen's `minimize.m` very efficient implementation. For L-BFGS, we tried several values of its queue size and found $m = 4$ worked best in our datasets. Each method iterates until $\|\nabla E(\mathbf{x}_k)\| < 10^{-8}$, up to a maximum of 1 000 iterations (which only GD reaches, occasionally). The initial iterate was always the source instance $\mathbf{x}_0 = \overline{\mathbf{x}}$.

For each dataset, we generated 50 instances of problem (2), each given by a source instance $\overline{\mathbf{x}}$, a target class $k$ and a value of $\lambda$. Fig. 1 shows a scatterplot of objective function value $E(\mathbf{x}^*)$, number of iterations and runtime in seconds for each method on each problem instance. It is clear that Newton's method works better than any other method, by far (note the plots are in log scale). It takes around 10 iterations to converge and between 1 and 100 ms runtime—a remarkable feat given that the problems are nonlinear and reach over $10^5$ variables. Next best are CG and L-BFGS, which take about 10 times longer; and finally, GD, which is far slower. BFGS is only applicable in small- to medium-size problems because it stores a Hessian matrix approximation explicitly; in our problems this makes it very slow even if it does not require many iterations.

Fig. 2 shows the learning curves for one specific problem instance for RCV1 dataset. Most informative are those for $E(\mathbf{x}_k) - E(\mathbf{x}^*)$ and $\|\nabla E(\mathbf{x}_k)\|$, which clearly show the asymptotic convergence rate of each method (note that while $E$ must decrease monotonically at each iteration, the gradient norm need not). For linearly convergent methods (all except Newton's) the iterates in a log-log plot should trace a straight line of negative slope equal to the logarithm of the rate; for quadratically convergent methods (Newton's), they trace an exponential curve instead. This is clearly visible in the ends of the curves. The last few iterates of Newton's method double the number of correct decimal digits at each iteration, and quickly reach the precision limit of the machine.

## 5. Conclusion

Inverse classification problems such as counterfactual explanations and adversarial examples can be formulated as optimization problems. We have theoretically characterized the important case of logistic regression and softmax classifiers (using the Euclidean distance). The former admits a closed-form solution, while the latter can be solved most efficiently with Newton's method using a suitably reformulated Hessian. In both cases, we can solve the optimization to practically machine precision and scale to feature vectors of over $10^5$ dimensions and tens of classes with a runtime under one second. This makes it suitable for real-time or interactive applications. Future work includes handling a very large number of classes; constraints on the desired instance; and investigating other classifiers, distances and problem settings.

[2]The proof of this and other results appear separately in a paper under submission.

## References


Dosovitskiy, A. and Brox, T. Inverting visual representations with convolutional networks. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, Las Vegas, NV, June 26 – July 1 2016. 1

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015. 1

Hada, S. S. and Carreira-Perpiñán, M. Á. Sampling the "Inverse Set" of a Neuron: An Approach to Understanding Neural Nets. arXiv:1910.04857, 19 2019. 1

Karimi, A.-H., Barthe, G., Balle, B., and Valera, I. Model-agnostic counterfactual explanations for consequential decisions. arXiv:1905.11190, October 8 2019. 1

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, November 1998. 4

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. RCV1: A new benchmark collection for text categorization research. *J. Machine Learning Research*, 5:361–397, April 2004. 4

Mahendran, A. and Vedaldi, A. Visualizing deep convolutional neural networks using natural pre-images. *Int. J. Computer Vision*, 120(3):233–255, December 2016. 1

Nesterov, Y. *Introductory Lectures on Convex Optimization. A Basic Course*. Number 87 in Applied Optimization. Springer-Verlag, 2004. 3

Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., and Clune, J. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pp. 3387–3395. MIT Press, Cambridge, MA, 2016. 1

Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006. 3, 4

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. Scikit-learn: Machine learning in Python. *J. Machine Learning Research*, 12:2825–2830, October 2011. Available online at `https://scikit-learn.org`. 4

Russell, C. Efficient search for diverse coherent explanations. In *Proc. ACM Conf. Fairness, Accountability, and Transparency (FAT 2019)*, pp. 20–28, Atlanta, GA, January 29–31 2019. 1

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. of the 2nd Int. Conf. Learning Representations (ICLR 2014)*, Banff, Canada, April 14–16 2014. 1

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *Proc. of the 2nd Int. Conf. Learning Representations (ICLR 2014)*, Banff, Canada, April 14–16 2014. 1

Ustun, B., Spangher, A., and Liu, Y. Actionable recourse in linear classification. In *Proc. ACM Conf. Fairness, Accountability, and Transparency (FAT 2019)*, pp. 10–19, Atlanta, GA, January 29–31 2019. 1

Wachter, S., Mittelstadt, B., and Russell, C. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard J. Law & Technology*, 31(2):841–887, Spring 2018. 1

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *Proc. 13th European Conf. Computer Vision (ECCV'14)*, pp. 818–833, Zürich, Switzerland, September 6–12 2014. 1