OVERFITTING MANAGEMENT IN DEEP NETWORKS: A SENSITIVITY CORRECTION TO THE TRAINING LOSS

Anonymous authors
Paper under double-blind review

ABSTRACT

Overfitting in deep neural networks (DNNs) is handled by a combination of techniques: an L2 penalty and dropout, each with its own hyperparameter, and early stopping on a held-out validation set. All are set from outside the fit: tuned or monitored on held-out data rather than estimated within it. In classical statistics, by contrast, the hyperparameter controlling L2 can be learned from the data. Empirical Bayes estimates it as a variance component while fitting the model, and the penalized objective is minimized to convergence with no stopping rule. This paper pursues the same goal in a DNN: estimating from the training data, within SGD, the strength at which a fit neither overfits nor underfits. The classical construction does not transfer unchanged, and the adaptation and extension it requires are the substance of the work. The difficulty is that the very expressive power that makes a DNN useful enables it to hide overfitting. A DNN can satisfy an L2 penalty by holding a weight small while amplifying the gain applied downstream, so influence grows while magnitude does not, and the penalty stops binding. The same freedom opens the door to a kind of overfitting that neither the penalty nor its classical estimate can see, and that is why in DNNs L2 is left as a hyperparameter and monitoring is needed to decide when to stop. We add a sensitivity correction to the L2 penalty, pricing what that penalty cannot see: the sensitivity the network grants itself. The adjusted loss has its roots in the hierarchical likelihood of Lee and Nelder (in a linear mixed model it reduces exactly to REML, so the correction is an extension of REML to DNNs). Regularization strength is then estimated during ordinary training, and no validation set is needed to set it or to stop training. The quantity controlled is the model's effective degrees of freedom, the capacity a fit uses rather than the parameters it owns. We call the approach OMID, for Overfitting Management in Deep Networks, and focus first on embedding-dominated architectures. We test OMID in simulation, where the data-generating process is known, and on the public Criteo and Avazu click-through-rate benchmarks. Both held-out loss and ranking accuracy improve. Under tuned L2 with early stopping, held-out loss reaches its best within a few epochs and rises sharply thereafter, and that best is not the best available. With the correction, loss stops turning: it keeps falling as training continues and ends below that value at every data size tested, with the largest gains where data is scarcest. Training and held-out performance no longer diverge, and further passes bring diminishing returns rather than damage. Expected calibration error drops by factors of three to twenty. We close with the extension now under way, to sequence models and large language models, where transformer and self-attention architectures change how sensitivity must be addressed.

1 INTRODUCTION

In standard deep learning practice, overfitting is managed through external heuristics rather than intrinsic estimation. An L_2 penalty, or equivalently weight decay, shrinks weights according to a hyperparameter tuned via external search; dropout deactivates units at a hand-tuned rate; and early stopping halts optimization when performance degrades on a held-out validation set. In these dominant paradigms, the regularizer is not inferred from the training data during the fitting process.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

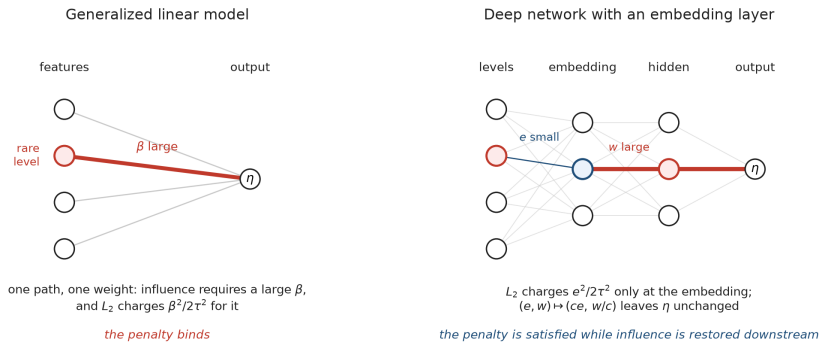


Figure 1: A level seen once, in two architectures; edge thickness shows weight magnitude. **Left:** in a generalized linear model it reaches the output along one edge, so fitting it needs a large β, and L_2 penalises it $\beta^2/2\tau^2$ for exactly that. **Right:** it reaches the output through an embedding weight e and a downstream weight w, with $\eta = we$. L_2 applies at e, which can be made small while w restores the influence: for any $c \neq 0$, $(e, w) \mapsto (ce,\ w/c)$ leaves η unchanged and shrinks $\|e\|^2$ by c^2.

This contrasts fundamentally with model training in classical statistics for hierarchical and multilevel models. In these models the effects are treated as random draws from a population whose variance is estimated directly from the data. This learned variance adaptively determines the optimal strength of regularization. Consequently, the penalized objective can be minimized to convergence: there is no need for empirical stopping rules or artificial data partitioning, because the model intrinsically estimates the regularizer during optimization.

The standard algorithms for variance components run two loops, whether the inner step uses a Laplace or quadrature approximation, as in lme4 (Bates et al., 2015), or a linearization with a closed-form variance update, as in pseudo-likelihood methods (Wolfinger & O'Connell, 1993). The inner one solves for the random effects with the variance components held fixed; the outer one updates those components given that solve, and the two alternate until they agree. Deep networks are not trained that way. SGD makes one pass over a single differentiable loss, and there is no outer loop to host an update of the variance components.

Lee and Nelder proposed an alternative that reaches the same end in a single loss (Lee et al., 2006). It has three parts,

$$\underbrace{\text{data loss}}_{\text{fit}} + \underbrace{L_2 \text{ penalty}}_{\text{magnitude}} + \underbrace{\text{one extra term}}_{\text{the subject of this paper}},$$

two of which every deep network already has. The effects and the variance components move under the same gradient, with no outer loop. We do not reproduce the derivation here; in the linear mixed model the third term recovers REML exactly. It is standard in hierarchical and multilevel models, and to the best of our knowledge it has not been adapted to deep networks.

In a generalized linear mixed model that third term has one job: it makes the estimates of the variance components, and of the random effects themselves, unbiased. Carried into a deep network it acquires a second and more consequential one. It blocks a route to overfitting that exists only in a deep network — one the classical model never had to defend against, and one no penalty on magnitude can reach.

Take a categorical feature and one of its levels. In a generalized linear model that level has a single coefficient. Raise it and every prediction for rows carrying that level rises; lower it and they fall. That is the only control the model has over how much the level matters, and how far the output moves per unit of coefficient is set by the data, not by the fit.[1] So bounding the size of the coefficient bounds the level's influence, and L_2 is enough.

[1]The Fisher information $H_k = \sum_i p_i(1-p_i)x_{ik}^2$ does move with the fit through the mean–variance weight $p_i(1-p_i)$, but that weight is bounded by $1/4$ and tied to the data; it is not a lever.

A deep network gives the level a vector of embedding values instead, and those values do not reach the output directly. They pass through the layers above, which multiply them on the way. How far the output moves when one embedding value changes therefore depends on those layers — and the network fits those layers too. It can shrink every value in the table, satisfying the penalty, and raise the multipliers above so that the output moves exactly as much as before. Figure 1 shows the smallest case in which this happens.

An L_2 penalty therefore does not control how much influence an input feature has. A model can satisfy it and still be highly sensitive to a handful of examples, which is memorization rather than generalization. The variance components are not identifiable either: distinct scales describe the same predictive model.

The goal of this paper is to control that distinction — memorization against generalization — from inside the loss. The quantity we work with is the effective degrees of freedom: how much of a model's capacity the data actually recruits, as opposed to how many parameters it has. The correction both exposes that quantity and caps it, while the model trains. Morris et al. (2026) approach the same distinction from another direction, measuring capacity as bits stored per parameter; the two quantities are related, though not the same, and we draw the comparison where it is informative.

2 THE H-LIKELIHOOD IN A DEEP NETWORK

We focus on categorical data entering a multilayer perceptron through embedding tables: each feature has one table, with a row per level of that feature. An observation takes a single level of each feature, so it looks up one row per table; those rows are concatenated into one vector, which passes through the MLP to the output. We call this a concat-MLP for short. The simplest instance has one table with a dense stack above it, which is what we describe first; several tables extend it directly.

Write $e \in \mathbb{R}^{K\times d}$ for the table of a feature with K levels, e_j for the row carrying level j, and θ for the dense parameters, so that the output before the link is $\eta = g(\theta; e_j)$. With g the identity this is a generalized linear model, each level contributing through its own coefficient; with layers in g it is a deep network. What separates the dense from the sparse parameters is how much data stands behind each entry: every example touches every dense parameter, while a row of e — the vector for one level — is touched only by the examples carrying that level.

The classical treatment makes each row a *random effect*, $e_j \sim \mathcal{N}(0, \tau_e^2 I)$, and estimates τ_e^2 from the data by maximizing the marginal likelihood $p(y \mid \tau_e^2) = \int p(y \mid e)\, p(e \mid \tau_e^2)\, de$. The penalty this implies pulls on every row with the same force; what differs is the resistance, since a row supported by many observations barely moves while a row seen once or twice is pulled almost to zero. The prior variance setting that force is inferred rather than chosen, and that is the property we are after. We take the response to be Bernoulli, as the experiments are mostly click or no click — any other member of the family changes only the Fisher weight — and for a Bernoulli response the integral has no closed form.

Lee et al. (2006) give an objective that avoids the integral and the two-loop scheme it forces,

$$h_{\mathrm{a}}(\tau_e^2, e) \;=\; \log p(y \mid e, \theta) \;+\; \log p(e \mid \tau_e^2) \;-\; \tfrac{1}{2}\log\big|H_{ee}/(2\pi)\big|, \qquad H_{ee} = -\partial^2 h/\partial e\, \partial e^\top, \tag{1}$$

a bona fide likelihood for the prior variance: maximizing it estimates τ_e^2, and in the linear mixed model the maximizer is exactly REML. The third term is what makes that true — without it the first two are not a likelihood for τ_e^2 at all, and maximizing them over-shrinks.

Classically θ is a *fixed effect*, carrying no prior, and the adjustment applies to the random effects alone. We keep that division. Written as a training loss, Equation 1 reads

$$L \;=\; \underbrace{\mathrm{NLL}(y \mid e, \theta)}_{\text{data loss}} \;+\; \underbrace{\tfrac{1}{2}\sum_j \frac{\|e_j\|^2}{\tau_e^2}}_{L_2} \;+\; \underbrace{\tfrac{1}{2}\log\det\big(H_{ee} + \tfrac{1}{\tau_e^2} I\big)}_{\text{correction}}, \tag{2}$$

where H_{ee} measures how sharply the loss responds to moving an embedding parameter, equivalently how much the output depends on it. We call this its *sensitivity*. Equation 2 is $-h_{\mathrm{a}}$ up to an additive $Kd \log \tau_e$, which is free of the parameters but not of τ_e, so it can be dropped while τ_e is held fixed

and not once it is estimated. Equation 1 is written with the Hessian of h. For a generalized linear model with the canonical link the Hessian and the Fisher information coincide; in a network they do not, because the Hessian picks up a residual-weighted term that can be negative, so away from a mode it need not be positive definite. We evaluate the Fisher form, a sum of outer products with non-negative weights, which is positive semi-definite everywhere — as a log-determinant requires.

The correction sits on e because that is where overfitting is the concern: every example touches every dense parameter, so each of them is pinned by the whole dataset, while a row of e is seen only as often as its level occurs and most levels are rare. It does not leave θ untouched either: how much the output depends on a row is decided partly by the layers above the table, so H_{ee} is a function of θ, the correction has a gradient with respect to θ, and the dense weights are penalised for the amplification they apply to the rows.

What remains is to evaluate $\log\det(H_{ee}+\tau_e^{-2}I)$, and with H_{ee} of size $Kd\times Kd$ that is prohibitive. The matrix is sparse, however, and its structure helps. Two levels couple only if some example carries both, and within one table that never happens, since an example carries a single level per feature, so the only nonzero blocks away from the per-level diagonal are those between tables. Taking them to be zero leaves one $d\times d$ block per level and the log-determinant becomes a sum of per-level terms, each one Cholesky factorization with d in the tens, which is tractable. Cheaper still is to ignore the block as well and keep only its diagonal,

$$\tfrac{1}{2}\log\det\big(H_{ee}+\tfrac{1}{\tau_e^2}I\big) \quad\longrightarrow\quad \tfrac{1}{2}\sum_j \log\big(H_j+\tfrac{1}{\tau_e^2}\big), \tag{3}$$

one scalar term per entry of the table, that is one per coordinate of each level's vector, with $H_j = \sum_i w_i(\partial\eta_i/\partial e_j)^2$ summed over the examples that touch entry j and w_i the Fisher weight of the likelihood.

Algorithm: OMID-Sparse

input: data (x, y), model g, prior variance τ_e^2, an optimizer (Adam throughout)

repeat

1. forward pass; evaluate the data term $\mathrm{NLL}(y \mid e, \theta)$
2. for each table entry j the batch touches, form $H_j = \sum_{i\in B_j} w_i\,(\partial\eta_i/\partial e_j)^2$ over the examples B_j that touch it, with w_i the Fisher weight ($p_i(1-p_i)$ for a logit, μ_i for a Poisson rate)
3. form the objective of Equation 2 under the substitution of Equation 3,
$$L = \mathrm{NLL}(y \mid e, \theta) + \tfrac{1}{2}\|e\|^2/\tau_e^2 + \tfrac{1}{2}\textstyle\sum_j \log\big(H_j + 1/\tau_e^2\big),$$
with each H_j left attached to the graph, so the gradient flows through it
4. update (e, θ) with one optimizer step on $\nabla_{(e,\theta)}L$

until converged

3 THE SENSITIVITY CORRECTION'S ROLE IN OVERFITTING MANAGEMENT

Equation 2 is a loss, minimized over the dense and sparse parameters by ordinary gradient-based optimization, and over τ_e as well. To understand the role of the extra term, we analyze the gradient it introduces, which factors in two:

$$\frac{\partial}{\partial e_j}\Big[\tfrac{1}{2}\log\big(H_j+\tfrac{1}{\tau_e^2}\big)\Big] = \tfrac{1}{2}\underbrace{\frac{1}{H_j+1/\tau_e^2}}_{\text{(A) rate}}\quad\underbrace{\frac{\partial H_j}{\partial e_j}}_{\text{(B) growth in sensitivity}}. \tag{4}$$

(B) is what the penalty responds to: raise the model's dependence on entry j and H_j rises with it. (A) scales that response, and it is the posterior variance — large where evidence is thin, small where it is plentiful. So the same increase in dependence adds almost nothing to the penalty on a level seen a hundred thousand times, and a great deal on a level seen once, with one global prior variance going in and the differentiation coming from each level's own evidence.

Summed over all parameters, this evidence-weighted penalty limits how many parameters the fit can recruit beyond what the data supports, which is where overfitting comes from. The quantity it limits is the *effective degrees of freedom*: not how many parameters the model owns, but how many of them the fit actually uses. A table of millions of rows whose entries stay near their prior adds almost nothing to it, because those rows do not move with the data. A closely related concept is *optimism*, coined by Efron (1986) for the gap between training and test loss — the amount by which training error understates the error on unseen test or validation data. Efron defines both not by a parameter count but through the covariance between each observation and its own fitted value,

$$\mathrm{eDF} \;=\; \sum_{i=1}^{N} \mathrm{Cov}(y_i, \hat{\lambda}_i), \qquad \mathbb{E}[\text{optimism}] \;=\; \frac{2}{N} \sum_{i=1}^{N} \mathrm{Cov}(y_i, \hat{\lambda}_i), \tag{5}$$

where $\hat{\lambda}_i$ is the fitted natural parameter and error is measured on the log-likelihood scale. For a Bernoulli response $\hat{\lambda}_i$ is the fitted log-odds; for a Gaussian one it is $\hat{y}_i/\sigma^2$. The definition says where that gap comes from. A fit that memorizes makes $\hat{y}_i$ move with the noise in the particular y_i it was trained on, and that covariance is precisely what the expression counts. Overfitting in an over-parameterized network is therefore not a consequence of having many parameters but of *the fit covarying with the noise in its own labels* — a distinction a parameter count cannot express and this one can.

For the penalized fit of Equation 2 the Fisher approximation to that covariance has one term per entry, $\mathrm{df}_j = H_j/(H_j + 1/\tau_e^2) \in [0, 1]$. It is close to 1 when the data has enough evidence to set that entry, and close to 0 when there is too little and the prior sets it instead. Adding the terms up gives the effective degrees of freedom of the whole fit. Note that df depends on τ: a smaller τ makes $1/\tau^2$ larger and every term smaller, so the same fit reports fewer degrees of freedom. Counts from different models are therefore only comparable when evaluated at one common reference τ, which is how we report them.

One point deserves emphasis, because it is one of the key contributions of this paper. The term is not ours: it comes from the h-likelihood of Lee et al. (2006), and in the linear mixed model it is REML, which standard software has used for decades. But the role it plays there is a narrow one. When η is linear in e, the sensitivity $\partial\eta_i/\partial e_j$ is the feature value, fixed before fitting begins, so $\partial H_j/\partial e_j = 0$, factor (B) is zero, and the term has no gradient with respect to the random effects at all. It corrects the likelihood so the variance component can be estimated, and that is all it does. A link function does not change this in any way the model can use: H_j moves with the fitted mean, but nothing lowers it except fitting the data differently.

In a deep network the same term takes on a new role. With learned parameters between the embedding and the output, $H_j = H_j(e, \theta)$, so the model can lower H_j and leave its predictions untouched — shrink a row, restore the influence downstream, the manoeuvre of Figure 1. Factor (B) is no longer zero, the term acquires a gradient with respect to the parameters, and it becomes an active force on the fit, penalising the one manoeuvre an L_2 penalty cannot see. **Identifying that role, and making the term a training loss because of it, is what this paper contributes.**

4 Implementation and numerical study

4.1 A known-truth example

We first compare OMID-Sparse against L_2 alone on synthetic data, where the truth is known.

Data-generating process (DGP): a concat-MLP with $M = 8$ categorical features, $K = 1000$ levels each and $d = 5$ embedding dimensions, one true embedding row per level, feeding two hidden layers of size 64 and 16 with $\tanh$ activations to a Poisson head. Because the DGP is known, so is the true rate behind every test point, and the fit can be scored against it rather than against a noisy realisation of it: we report the RMSE between the fitted and the *true* log-rate.

We fit two models to the same data, full batch, differing only in the activation function. The $\tanh$ model matches the DGP's architecture exactly, so it can represent the truth; the ReLU model is identical except for the activation.

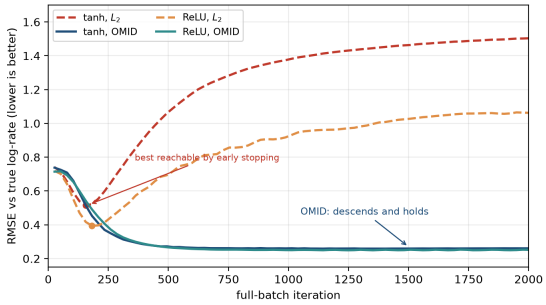


Figure 2: 8 features, 1000 levels each, 100k sampled rows, Poisson. RMSE on test data.

Level popularity follows a Zipf law, $p_k \propto k^{-1/2}$ over ranks $k = 1, \ldots, K$, so across 100k training examples the median level count is 72 and the rarest level is seen 31 times. Both arms carry the same L_2 penalty and start from the same prior, $\tau_0^2 = 1$. The baseline keeps that L_2 fixed; OMID adds the correction term and estimates τ^2 during training.

As Figure 2 shows, under L_2 alone the test error falls early in training but soon turns and climbs, for both models; with the correction, and τ^2 estimated throughout, it keeps falling and then holds. **The best point on the L_2 path is the best early stopping can give, and it is worse than where the OMID error ends.**

4.2 Running at scale

The run above was full batch because it could be: N was 100k, so every update could use the whole training set. Production systems train on hundreds of millions to billions of rows, where a full pass per update is out of the question. The data term is estimated instead on a minibatch B of size b, without bias,

$$\sum_{i=1}^{N} \ell_i(e, \theta) \;\approx\; \frac{N}{b} \sum_{i \in B} \ell_i(e, \theta), \tag{6}$$

and the same substitution in the curvature gives $\hat{H}_j = (N/b) \sum_{i \in B \cap B_j} w_i (\partial \eta_i / \partial e_j)^2$, likewise unbiased for H_j.

Two of the three terms in Equation 2 are content with that. The data term is a sum over examples, so a rescaled minibatch sum estimates it without bias; the L_2 term does not involve the data at all. The correction is the exception. H_j is a sum over examples too, but it enters through a logarithm, and an unbiased estimate of H_j is not an unbiased estimate of $\log(H_j + \tau_e^{-2})$. The logarithm is concave, so Jensen's inequality gives

$$\mathbb{E}\big[\tfrac{1}{2} \log(\hat{H}_j + \tau_e^{-2})\big] \;\le\; \tfrac{1}{2} \log(H_j + \tau_e^{-2}), \tag{7}$$

with equality only when $\hat{H}_j$ has no variance. Unbiasedness does not survive the logarithm: $\hat{H}_j$ is unbiased for H_j, but $\frac{1}{2} \log(\hat{H}_j + \tau_e^{-2})$ is not unbiased for $\frac{1}{2} \log(H_j + \tau_e^{-2})$. The term under-penalises, the effective degrees of freedom blow up, and the model overfits.

Equation 4 locates the damage. The gradient is a product of the rate (A), which is the posterior variance, and the derivative (B); estimating H_j on a minibatch puts noise into both. The two need not come from the same sample. We evaluate (A) on a large reference sample of size $G \gg b$, where Equation 7 is close to tight, and take (B) from the current minibatch, so the penalty contributes

$$\tfrac{1}{2} \, \frac{1}{H_{\mathrm{ref},j} + \tau_e^{-2}} \, \frac{\partial \hat{H}_j}{\partial e_j},$$

an accurate rate against a cheap direction. Only step 2 of the algorithm changes. The reference sample is refreshed on a slow schedule, and only the rows a batch touches are updated, so the cost stays proportional to the active rows rather than to the table.

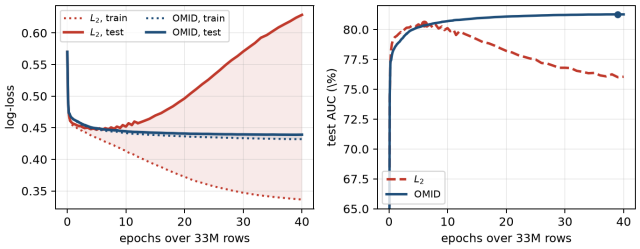


Figure 3: Criteo, all 33M training rows, 40 epochs. **Left:** train and test log-loss, the shaded gap being the optimism of Equation 5. **Right:** test AUC, with each arm's best marked.

4.3 The diagonal approximation

Equation 3 drops the off-diagonal of H_{ee}. Hadamard's inequality bounds the determinant of a positive definite A by $\det A \le \prod_i A_{ii}$, so

$$\log\det\big(H_{ee} + \tfrac{1}{\tau_e^2} I\big) \;\le\; \sum_j \log\big(H_j + \tfrac{1}{\tau_e^2}\big), \tag{8}$$

with equality only when H_{ee} is already diagonal, and the degrees of freedom are overestimated with it: $\sum_j H_j/(H_j + \tau_e^{-2})$ exceeds $\mathrm{tr}(H_{ee}(H_{ee} + \tau_e^{-2} I)^{-1})$. The gap is the dependence the diagonal ignores: treating the coordinates as independent counts evidence shared between correlated ones once for each of them. What is overestimated is not the model's capacity, fixed by the parameter count, but how much of it the fit has recruited.

A discount $k \le 1$ on the correction would absorb the excess. With $k = 1$ the penalty applied is stronger than the exact one, which errs in a safe direction: it cannot produce overfitting, but it can produce underfitting. Measured against the exact form on Criteo, the diagonal overstates the degrees of freedom about fivefold.

5 CTR prediction on real data

Criteo is the standard public benchmark for click-through-rate prediction. We use the official Criteo_x1 split of Zhu et al. (2021) — 33M training rows, a fixed test set of 4.5M — and every feature, 13 numeric and 26 categorical. Levels seen fewer than ten times in training all share a single out-of-vocabulary row, one such row per feature, and a feature with K surviving levels gets an embedding of width $\lceil K^{1/4} \rceil$, which over this vocabulary runs from 2 to 26; the numeric features are quantile-bucketised into 64 bins and embedded at width 10, giving 39 embedded fields into hidden layers of size 400, 300 and 100. The first dense layer is row-normalised each step. Both arms start from a byte-identical initialization and from the same prior, $\tau_0 = 0.5$ on every run. The baseline holds that value throughout; OMID estimates it. Both arms train with Adam under a cosine decay from 2.5×10^{-3} to 5×10^{-4} over the run. The dense layers carry an L_2 penalty in both arms at $\tau_{\text{dense}} = 0.32$, tuned as a hyperparameter and no part of the adjustment.

Figure 3 tells the same story as the simulation. On the left, OMID's training and test losses stay close for the whole run while the baseline's diverge, its optimism blowing up. On the right the test AUC says it from the other side: OMID improves gradually and never collapses, and it ends above the baseline's peak. We obtain the same picture on other click-through benchmarks, Avazu among them.

OMID estimates a prior variance for each feature family, not one for the table. All 39 fields start at $\tau_0 = 0.5$ and 39 separate values are estimated; they end between 0.17 and 1.39. A single global weight decay, fixed for every field, cannot do this.

OMID shines even more when data become scarce. We repeated the experiment on a tenth of the corpus, 3.3M rows, with the embedding table built from all 33M so that the model is identical and only the data differs. OMID wins again at 3.3M, and by a wider margin than at 33M. With less

Table 1: Criteo results at two training sizes.

rows		AUC best	best epoch	AUC drop
33M	L_2	80.46%	6	-4.41%
	OMID	$\mathbf{81.27}\%$	39	$\mathbf{0.00}\%$
3.3M	L_2	78.16%	4	-6.54%
	OMID	$\mathbf{79.36}\%$	100	$\mathbf{0.00}\%$

data the baseline turns in the wrong direction earlier, which is what scarcer data should do: it is the regime most prone to overfitting, and the one where the correction is worth most.[2]

Equation 5 gives each fit its own effective degrees of freedom, and reporting it is instructive. Summed over the whole training set, OMID determines 245k at 33M rows and 31k at 3.3M, out of an embedding table holding 19.5M parameters in either case, against 9.6M and 1.9M for the baseline.[3] This is how OMID fights overfitting: it caps the capacity a fit is allowed to recruit, where the baseline recruits roughly in proportion to the rows it is given. That is not a claim that a 245k-parameter model can be found with the same fit — the capacity is spread thinly, a few rows carrying a great deal and most sitting near the prior — but it does suggest that most of the table is doing little.

Compressing OMID models: a fit that recruits little capacity ought to be compressible. Which rows to drop, though, cannot be judged by how far they sit from zero. A rarely-seen level can carry a large vector that almost no data stands behind, and a heavily-used level a small one the predictions lean on. What settles it is how much evidence stands behind a row, and OMID has already measured exactly that. The saving is worth having in an application such as a recommender, where the embedding table dominates the serving footprint and with it the serving cost.

We score each level of each feature by

$$z_{m,k} \;=\; \hat{b}_{m,k}^{\top}\big(H_{m,k} + 1/\tau_m^2\big)\hat{b}_{m,k}, \tag{9}$$

where m indexes the feature and k the level, $\hat{b}_{m,k}$ is the fitted embedding row, and $H_{m,k} + 1/\tau_m^2$ is that row's posterior precision — the curvature the data supplies for it together with the prior, both already estimated during training. The score is the size of a row normalised by its posterior variance, a Mahalanobis length: being large counts for little if the data has not determined it. Nothing new has to be computed and there is **no retraining**; we sort by $z_{m,k}$ and drop the smallest rows from the table.

Table 2: Results from compressing the OMID model. Rows are dropped to a parameter budget, so a column is the same model size at both training sizes.

parameters kept	100%	50%	25%	10%	5%	2.5%
AUC, 33M	81.27%	81.26%	81.26%	81.22%	81.16%	81.02%
AUC, 3.3M	79.36%	79.36%	79.36%	79.35%	79.35%	79.33%

A quarter of the parameters at 33M, and a tenth at 3.3M, reproduce the full model. The score is what makes this work: at a twentieth of the parameters, ranking by frequency costs 0.42 percentage points and by row norm 1.33, against 0.11 here. We report parameter counts rather than served bytes, and have not tried a min-count filter with this ranking on top, which is the comparison a practitioner would want.

[2]The 3.3M run takes 100 epochs rather than 40 because an epoch here is a tenth the optimizer steps at the same batch size; it was still improving at the end.

[3]On a subsample the concavity of $H/(H + \tau^{-2})$ biases this downward, severely so for the baseline, whose capacity is spread across many rarely-seen rows; these are summed over all rows.

6 RELATED WORK

Immer et al. (2021) are the closest work, and the starting point is common: they set the strength of regularisation by maximising a Laplace approximation to the marginal likelihood, which carries the same log-determinant of the same Gauss–Newton curvature we use, and they too need no validation set. Their goal ends there — a prior variance estimated from the training data for the fit the MAP objective produces — so the log-determinant enters only the hyperparameter update. We share that goal, but argue that an L_2 penalty is not enough however well its strength is chosen, and that the correction has to sit inside training. We do not treat it as model or prior selection: it is the training objective itself that has to change if overfitting is to be managed.

The recommendation literature has noted that click-through models can start to overfit after a single epoch, which is consistent with what we see here. Fan et al. (2024) diagnose the same locus and re-initialize the table each epoch; Zheng et al. (2023) rescale the L_2 term column-wise to permit very large batches. These are engineering remedies without an estimand, and we do not compare against them.

We would also draw the reader's attention to recent work on the storage capacity of deep networks, which locates the point at which the data outrun what a model can memorise. Morris et al. (2026) find that a model memorises until its storage is full and is then forced to share information between datapoints, which is what generalising is. Measuring on data with no structure to generalise, so that whatever is retained has been memorised, they put the capacity of a GPT-style transformer at roughly 3.6 bits a parameter. What they count is bits, and so is our correction. Under a minimum-description-length reading the prior fixes the range a parameter may lie in, of width τ, and the curvature H the resolution at which the data can locate it there; the logarithm of their ratio, $\frac{1}{2}\log(1+\tau^2 H)$, is the bits the data have deposited in it, and summed over the table this is the term in our objective, with the effective degrees of freedom of Equation 5 its companion count in parameters. What differs is the use: they measure the bits a trained model has retained, where we charge for them while it trains.

That figure is measured on transformers trained on uniform random data, and we would not assume it transfers unchanged to a CTR embedding table. The supply of data, however, certainly does not. In a language model the dense weights see every token and carry most of the parameter budget; here almost every parameter sits in an embedding table, where the data are divided among the rows and very unevenly. A click label carries at most one bit, so 33M rows supply at most 33M bits to 19.8M parameters, under two bits per parameter. At the 3.6-bit ceiling those same parameters could hold some 71M bits — room to memorise the whole label set twice over. The tail is further out still, an order of magnitude below the ceiling rather than half of it: a level seen ten times supplies ten bits to a row of up to 26 parameters. Data do not outrun capacity here, so generalisation must be imposed from the inside, which is what capping the effective degrees of freedom does. Were the data ever to outrun it, the posterior variance in Equation 4 would shrink and the correction would recede of its own accord.

7 CONCLUSION

Nelder's adjusted profile likelihood extended the principle of REML to models in which exact marginalisation over the random effects is intractable. This paper carries that principle a large step further, into deep networks. The term is Nelder's, unchanged; what changes is what it touches. In a mixed model it lets the variance component be estimated and leaves the random effects untouched. In a deep network its curvature depends on the fitted parameters, and we argue, and show, that it should therefore bear on those parameters too: it becomes a capacity control rather than a device for estimating the variance component. It prices memorisation, caps the capacity spent on it, and turns a model that degrades with training into one that generalises.

The construction is not specific to embedding tables. The term applies wherever a model sets its own sensitivity, which in a transformer includes the dense and attention layers. As §6 argues, a model can generalise of its own accord when the data outrun its capacity, and much work with large language models sits well short of that: fine-tuning above all, where a pretrained model meets a small task set. There the capacity must be capped from the inside, and we have results showing this correction doing it. More broadly, wherever a deep network is at risk of overfitting, it has a role. Those settings are the subject of a sequel publication.

REPRODUCIBILITY STATEMENT

All datasets are public: the official Criteo_x1 split. §2 gives the objective and the exact Fisher for each likelihood, §4 gives the two bias corrections and the update cadence, and §5 gives the architecture, the vocabulary rule and the optimiser settings. The synthetic generator of §4 is specified in full and needs no data download. Every comparison reported here differs in one term: both arms run the same objective with the correction switched on or off, from the same initialization. Runs use PyTorch 2.13 on an Apple Silicon GPU through the MPS backend, with a single seed fixing both the initialization and the minibatch order. MPS reductions are not bit-reproducible, so a repeated run moves the test AUC by a few hundredths of a percentage point, well below the differences reported here.

AI USE STATEMENT

Large language models were used to assist with drafting and editing the prose of this paper. All technical content, experimental design, implementations, and results are the authors' own, and every claim was checked against the underlying runs by the authors.

REFERENCES

D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48, 2015.

B. Efron. How biased is the apparent error rate of a prediction rule? *Journal of the American Statistical Association*, 81(394):461–470, 1986.

A. Immer, M. Bauer, V. Fortuin, G. Rätsch, and M. E. Khan. Scalable marginal likelihood estimation for model selection in deep learning. In *ICML*, 2021.

Y. Lee, J. A. Nelder, and Y. Pawitan. *Generalized Linear Models with Random Effects: Unified Analysis via H-likelihood*. Chapman & Hall/CRC Monographs on Statistics and Applied Probability No. 106, 2006. Second edition, Monographs No. 153, 2017.

Z. Fan, Z. Liu, J. Liang, D. Kong, H. Li, P. Jiang, S. Li, and K. Gai. Multi-epoch learning with data augmentation for deep click-through rate prediction. *arXiv:2407.01607*, 2024.

J. X. Morris, C. Sitawarin, C. Guo, N. Kokhlikyan, G. E. Suh, A. M. Rush, K. Chaudhuri, and S. Mahloujifar. How much can language models memorize? In *ICML*, 2026. Preprint *arXiv*:2505.24832.

R. Wolfinger and M. O'Connell. Generalized linear mixed models: a pseudo-likelihood approach. *Journal of Statistical Computation and Simulation*, 48(3–4):233–243, 1993.

Z. Zheng, P. Xu, X. Zou, D. Tang, Z. Li, C. Xi, P. Wu, L. Zou, Y. Zhu, M. Chen, X. Ding, F. Xue, Z. Qin, Y. Cheng, and Y. You. CowClip: Reducing CTR prediction model training time from 12 hours to 10 minutes on 1 GPU. In *AAAI*, 2023.

J. Zhu, J. Liu, S. Yang, Q. Zhang, and X. He. Open benchmarking for click-through rate prediction. In *CIKM*, 2021. Data splits at https://openbenchmark.github.io/BARS/.